Matrix factorization and prediction for high dimensional zero inflated co-occurrence count data via shared parameter alternating generalized linear regression with application in NLP

by

Taejoon Kim

M.S., Korea University, South Korea, 2016

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Abstract

In this dissertation, we study methods for analyzing the cooccurrence count data derived from practical fields such as user-item data from online shopping platform, cooccurring word-word pairs in sequences of texts. Such data contain important information for developing recommender systems or studying relevance of items or words from non-numerical sources. There are no observations for covariates and the co-occurrence matrix is typically of so high dimension that it does not fit into a computer's memory for modeling.

We extract numerical data by defining windows of cooccurrence using weighted count on the continuous scale. Positive probability mass is allowed for zero observations. In this dissertation, we propose several likelihood-based mixture models with shared parameters to accommodate large amount of zero observations. Shared parameters also contribute additional difficulty in estimating the model parameters because different parts of the model need to be considered jointly during estimation. We summarize the relevance of user-item or word-word pairs by cosine similarity of their unknown dense vector representations. The unknown values are estimated via the shared parameter alternating regression models. Under this framework, we present detailed study of Shared parameter Alternating zero inflated gamma (SA-ZIG) regression and Shared parameter Alternating Tweedie (SA-Tweedie) regression, along with some applications in the Natural Language Processing (NLP) domain.

For the SA-ZIG regression, canonical link and log link models were considered. Two parameter updating schemes are proposed along with an algorithm to estimate the unknown parameters. Convergence analysis is presented analytically. Numerical studies showed that SA-ZIG with learning rate adjustment has satisfactory performance but the SA-ZIG using Fisher scoring without learning rate adjustment may fail to find the maximum likelihood estimate.

For the SA-Tweedie regression, multiple versions of an algorithm with different objective functions are proposed to estimate the parameters. Both models with and without constraints on parameters were studied. Updating formulae were given to obtain parameter estimates iteratively under each setting. A learning rate adjustment was used along with the Fisher scoring method to help the algorithm stay on track of optimizing direction. Numerical studies showed that our algorithm with Fisher scoring and learning rate adjustment outperforms the one without learning rate adjustment and gradient descent with Adam update. Pseudo-likelihood approach with alternating parameter update was also studied but found to be unsuitable for our shared parameter alternating regression models with unobserved covariates.

The proposed algorithms were applied to Wikipedia dump data to obtain token/word vector representations, which are used in some NLP tasks such as finding the most similar words, sentiment analysis, and Named Entity Recognition task. Detailed analysis and comparisons were discussed.

Matrix factorization and prediction for high dimensional zero inflated co-occurrence count data via shared parameter alternating generalized linear regression with application in NLP

by

Taejoon Kim

M.S., Korea University, South Korea, 2016

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Statistics
College of Arts and Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2023

Approved by:

Major Professor
Haiyan Wang

Copyright

Abstract

In this dissertation, we study methods for analyzing the cooccurrence count data derived from practical fields such as user-item data from online shopping platform, cooccurring word-word pairs in sequences of texts. Such data contain important information for developing recommender systems or studying relevance of items or words from non-numerical sources. There are no observations for covariates and the co-occurrence matrix is typically of so high dimension that it does not fit into a computer's memory for modeling.

We extract numerical data by defining windows of cooccurrence using weighted count on the continuous scale. Positive probability mass is allowed for zero observations. In this dissertation, we propose several likelihood-based mixture models with shared parameters to accommodate large amount of zero observations. Shared parameters also contribute additional difficulty in estimating the model parameters because different parts of the model need to be considered jointly during estimation. We summarize the relevance of user-item or word-word pairs by cosine similarity of their unknown dense vector representations. The unknown values are estimated via the shared parameter alternating regression models. Under this framework, we present detailed study of Shared parameter Alternating zero inflated gamma (SA-ZIG) regression and Shared parameter Alternating Tweedie (SA-Tweedie) regression, along with some applications in the Natural Language Processing (NLP) domain.

For the SA-ZIG regression, canonical link and log link models were considered. Two parameter updating schemes are proposed along with an algorithm to estimate the unknown parameters. Convergence analysis is presented analytically. Numerical studies showed that SA-ZIG with learning rate adjustment has satisfactory performance but the SA-ZIG using Fisher scoring without learning rate adjustment may fail to find the maximum likelihood estimate.

For the SA-Tweedie regression, multiple versions of an algorithm with different objective functions are proposed to estimate the parameters. Both models with and without constraints on parameters were studied. Updating formulae were given to obtain parameter estimates iteratively under each setting. A learning rate adjustment was used along with the Fisher scoring method to help the algorithm stay on track of optimizing direction. Numerical studies showed that our algorithm with Fisher scoring and learning rate adjustment outperforms the one without learning rate adjustment and gradient descent with Adam update. Pseudo-likelihood approach with alternating parameter update was also studied but found to be unsuitable for our shared parameter alternating regression models with unobserved covariates.

The proposed algorithms were applied to Wikipedia dump data to obtain token/word vector representations, which are used in some NLP tasks such as finding the most similar words, sentiment analysis, and Named Entity Recognition task. Detailed analysis and comparisons were discussed.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In real life, a lot of data can be summarized in a data table. It is common to study the relationship among a response variable and explanatory variables with such tables. In most of such applications, the data table also present information of explanatory variables. For example, in regression analysis, the rows of the table represent the observations while the columns of the table list different covariates and the factors that might contribute to explain the variations in the response variable. There are various statistical methods available for these tasks. In this dissertation, we consider modeling also using a data matrix. However, the matrix differ from those typically used in the regression analysis. Instead, the matrix presents some aggregated information about the pairs of members that might be from the same category or different categories.

An example of such data is the user rating of product. For example, Amazon product rating data in which each user gives rating on some products purchased by the user. The data table contains as many rows as the number of users and as many columns as the number of different products/sellers. The interest in analyzing this table is to build models that recommend products to potential buyers effectively. The entry in the table could also be amount of time spent on browsing the product by the user. In these tables, we have different products/sellers and we have different users/buyers but they do not serve as explanatory

variables. The analysis should extract or model the unavailable information behind the users or the products in order to make useful recommendations.

Another example of such tables is the Netflix movie rating data. There are a lot of subscribers and there are many movies. The data matrix contains subscribers' rating on movies. Each subscriber may or may not provide ratings for a movie watched. This leads to a huge sparse data matrix. The data could also be the amount of time watching a movie, in which case the data contains a lot of missing entries or zero that yield some positive numbers in continuous scale. The interest in analyzing this data is to find out the mechanism behind the users that make them decide to watch a movie or the hidden properties behind a movie that makes subscribers want to watch it. The eventual goal is to effectively recommend movies to get subscribers' attention or interests in watching them. This can be done by predicting the values in the matrix by modeling the underlying mechanism between subscribers/users and movies/products pairs.

Beyond predicting the value of the data matrix, the modeling process could also learn or estimate the dense vectors that represent rows or/and columns of the data matrix. The dense vector refers to much smaller dimension of vector whose entries are mostly nonzero. The dense vectors need to be in smaller dimension because the total number of unknown parameters must be less than the total number of data points in the matrix. Suppose the matrix is n by m. To make a dense representation of the user's characteristics behind each row using a d-dimensional vector, we have n by d unknown parameters. Similarly, to make a dense representation of the product behind a column's observed data, using a d-dimensional vector we have d by m unknown parameters. The total number of parameters is $d(n+m)$. In order to have enough degrees of freedom to estimate all parameters, the dimension d has to be smaller than half of $\min(n, m)$. Even when this is satisfied, estimating a total of $d(n+m)$ parameters with just the data in the n by m matrix is a challenge. The learned vector representations are useful to detect relationship across rows and columns, which indicate relevance between a user and a product. Additionally, the learned dense representations also

could be used to study clustering among rows or columns. For example, we could group users having similar dense representations which are indications of having common characteristics among the group of users. We could also group columns in the data matrix using the similarity measure with the dense representations. The grouped columns represents similar products that might be of interest to users.

The third example could be the word-word cooccurence matrix in Natural Language Processing (NLP) field. The cooccurrence matrix consists of pairs of target word and context word. The target word is a specific word of interest in the corpus and the context word is a word that occurs together with the target word within predefined window size. Hence, the $(i, j)^{th}$ entry X_{ij} in the cooccurence matrix indicates cooccurrence count of the i^{th} target word and the j^{th} context word. Since there are numerous words in the corpus and cooccurrence does not happen frequently, the cooccurrence matrix would have huge dimension with majority of entries 0. From these sparse counts in the cooccurrence matrix, we hope to capture underlying mechanism between the target words (rows) and the context words (columns). The captured mechanism would hopefully help the machine judge what words are the most plausible context word for a given target word. In addition, the learned dense vectors could be used to cluster similar words based on cosine similarity. These learned dense vectors are known as word embeddings and are commonly used in downstream NLP tasks such as sentiment analysis, word analogy, named entity recognition, etc.

In all these examples, using the cooccurrence count as data is one way of summarizing information in the data. Of course, the cooccurrence count matrix is not the only way. For example in the word2vec (Mikolov et al. 2013a,b,c), each word appearing in the context of a center word or the word appearing as the center word of a context window is treated as a binary variable. The whole vocabulary is treated as a multi class data with each category being one word. The probability of one word being the context word (or center word) is modeled by multinomial logistic regression, in which the dot product of the context word and center word's dense representation serves as the η in generalized linear model (GLM),

and the softmax of the η gives the probability of this word. The problem of this is that there are too many categories (words). A lot of effort was placed on reducing the calculation to compute the denominator in the softmax function. Hierachical softmax and negative sampling are two of such documented efforts. Considering the probability of one word in the context of another word being very small, and the total number of articles in NLP task being large, the number of times that the one word appears in the context of another word is a binomial random variable but could be approximated by Poisson distribution except that there might be too many zeros. The cooccurrence count in the GloVe paper (Pennington et al. 2014) considered the aggregated count of the two words appearing in same window is consistent with the thought of the Poisson approximation but the authors in Pennington et al. (2014) only modeled the aggregated count with weighted least square regression and did not consider the fact that the variation of such counts also changes with the mean.

Transformer (Vaswani et al. 2017) and BERT (Devlin et al. 2018) models treat texts as a sequence with ordering and then use layers of multi-head attention mechanisms to create intermediate variables before using the softmax at the end. The attention allows to capture word-word relationship in the whole sequence. A drawback of this model is that there are too many unknown parameters. For example, the BERT-base model has 110 million parameters. A model with such a large number of parameters is not suitable to train on any reasonable sized dataset without big corpus. The BERT model is successful in NLP because it was trained with huge corpus of text data from BooksCorpus having 800M words and from Wikipedia having 2,500M words. Essentially, the BERT model works well because the training data is so big that it has seen everything before going to downstream NLP tasks. If we change the application to a different domain with much smaller training data, then the model would seriously overfit because of too many parameters with insufficient degrees of freedom to estimate them. In this dissertation, we will review how the Transformer and BERT were built but our effort will not be in this direction because we would like our model to be easily trainable with medium sized data without overfitting.

Throughout this dissertation, we will focus on cooccurrence count data. Our cooccurrence count is not restricted to word-word cooccurrence in NLP. Instead, we use the cooccurrence count summarized from many potential application domain described earlier. Different from regular co-occurrence matrix which provides counts of a word appearing in another word's context in a paragraph, we consider weighted cooccurrence count in a predefined sequence such as a sentence. In the NLP setting, it would be cooccurrence of counts for words in the same sentence but not necessarily recorded as integers. Specifically, first we defined windows centered at a target word spanning k words to the left and k words to the right side restricted within the same sentence. All words in the window are considered as context words. For a context word in the window, the contribution of this word to the target word in the co-occurrence measure is defined to be the inverse of their separated distance between two words. That is,

- If two words, word$_i$ and word$_j$, appear in the same sentence (s) and their separation $d_{ij}(s)$ is no more than k words, then this sentence contributes to the cooccurence count between these two words by $1/d_{ij}(s)$.
- The overall cooccurrence count between the same pair of words is calculated as the total contribution from all sentences in the corpus.
- The cooccurrence matrix consists of the pairwise cooccurrence count between every two words in the vocabulary. For example, if the vocabulary size is 300, the cooccurrence matrix is 300 by 300 in dimension.

The cooccurrence count X_{ij} in the ith row and jth column of the matrix can be expressed as

$$X_{ij} = \begin{cases} \sum_{\text{s} \in \text{ all sentences}} 1/d_{ij}(s), & \text{if } d_{ij}(s) \leq \text{k} \\ 0, & otherwise \end{cases} \tag{1.0.1}$$

where $d_{ij}(s)$ = separation between word i and word j in sentence s. The closer a context

word is to the target word, the more contribution it has in relevance of the two words. This definition applies to other domains beyond the NLP. Consider, for example, if the data are items purchased from an online platform such as Amazon during a certain period, the time of each purchase is also recorded. When the entire sequence of purchased items are considered together, the window defines how the items are related in functionality or being jointly needed by customers around the same time period. Therefore, the cooccurrence count captures the adjacency of items purchased together in addition to simple count.

In the next section, we will briefly review relevant literature for these tasks. These literature contain current development in Recommender systems and in NLP.

Chapter 2

Literature Review

2.1 The word2vec and GloVe

When we have text data, the first thing we have to consider is transforming the text into some numerical values so that we could analyze or model them. They are popularly referred as word embedding which literally embed a word into dense numerical vector space. The two main streams, the word2vec and GloVe, of the word embedding field in NLP were developed by Google and Stanford university in 2013 and 2014, respectively. The word2vec models the probability of one pair of words appearing together with one context window at a time. The GloVe aggregates the count for the same word pairs in the entire training corpus. These two are related to each other and both of them have revolutionary effect in NLP field. We will review these two here since they work with cooccurrence count indirectly (word2vec) or directly (GloVe).

2.1.1 The word2vec model

The word2vec uses simple neural network structure having one hidden layer for modeling conditional probability of one word being observed when a sequence of words is given or modeling conditional probability of a sequence of words being observed when a specific word

is given. The first case is called continuous bag of words (CBOW) model and the second case is called skip-gram model. Throughout training the models, they obtain dense word vector representations and various strategies were used to make the training effective and feasible. We examine these two models and their strategies in detail in the next several paragraphs.

The CBOW model predicts the center word from a window of surrounding context words. The order of context words does not influence prediction. On the other hand, the skip-gram model predicts surrounding context words for the given center word. The predictions are made by computing conditional probability which utilizes the softmax function.

We illustrate with skip-gram architecture to see how word2vec learns the word embeddings. First, collect a large corpus of text (e.g., Google news, Wikipedia, etc.) as the training data. The skip-gram model look at the position t of each word in the text in a 'window'. Denote the center word as c and outside word o in the context. Then represent every word in the text by two vectors v_w, when w is a center word, and u_w when w is a outside (context) word. The inner product $u_o^T v_c$ of the word vectors for c and o represents the similarity between the two vectors since cos(angle) $= \frac{u_o^T v_c}{||u_o|| \cdot ||v_c||}$. Larger inner product gives larger similarity when o and c are similar. The probability of o given c (or vice versa for CBOW) using the softmax is defined as

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}.$$

The softmax is computationally extensive because of the following reasons:

- The inner product between v_c and the embedding of very word w in the vocabulary V needs to be computed in order to get the sum in the denominator.

- The parameter space is HUGE. Suppose d is the dimension of the vector size. E.g., $d = 300$ or 500. V is the vocabulary size, say 10000. There are $2 \times d \times V$ unknown parameters to be estimated (can be billions).

- Therefore, softmax is computationally extensive. It is practically impossible to estimate the parameters directly.

Mikolov et al. (2013a,b,c) used the Hierarchical softmax method with a binary Huffman tree and Negative sampling to approximate the softmax and reduce the active number of computations or the dimension of the parameter space. Essentially, the Hierarchical softmax replaces the flat softmax with a hierarchical structure that has the words as leaves. Computing the softmax probability of one word only need to follow the path to the leaf node of that word, without having to consider any of the other nodes (i.e., no need to normalize over the probabilities of all words in vocabulary). Because the sum of leaf probabilities is 1, the computed probability is already normalized. Since a balanced binary tree has a depth of $\log_2(|V|)$, we only need to evaluate at most $\log_2(|V|)$ nodes to obtain the probability of a word. In contrast to the regular softmax, we no longer have center or outside embeddings u_o and v_c for every word w. Instead, we have embedding v_w for every word w on the leaf node and embedding v_n for each internal node n. The probability of going right (or left) at a given node n given the word c is computed with logistic model:

$$\text{Pr(right | node n and given c)} = [1 + e^{-h^T v_n}]^{-1} \qquad \text{(sigmoid)} ,$$
$$\text{Pr(left | node n and given c)} = \text{1- Pr(right | node n and given c)} ,$$

where h is the embedding of word c. Note that it uses the dot product between h and the embedding v_n of node n in the tree instead of output word embedding v_w. Following the tree from root down to leaf using conditional probabilities we get $P(w_j|w_i)$.

Now consider a sequence of text, denote T to be the total number of training words in the vocabulary. For each $t = 1, \ldots, T$, we write the probability of word t and its surrounding context words within a fixed window size m given center word w_t and θ (i.e., the vector representations of all words that we will estimate). Assume conditional on the center word,

each context word is independent of other context word. The joint likelihood function is

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m} P\left(w_{t+j} \mid w_t; \theta\right).$$

The objective function is the (average) negative log-likelihood:

$$\ell(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

The word vectors are learned by maximizing the probability.

Mikolov et al. (2013a,b,c) used Huffman tree, which is a balanced binary tree developed in a Ph.D dissertation of a MIT student in 1952. Each leaf of the tree corresponds to a word in the given vocabulary. The weight of leaf node is proportional to relative frequency of the word in the leaf. During the tree building process, the weight of an internal node is the sum of its children nodes so that the weighted path length of a leaf is (its weight $\times$ its depth). The objective function of Huffman tree can be written as

$$\min_{\text{all possible placing of leaves}} \sum_{\text{all leaves}} \text{weighted path lengths}$$

for the given set of leaves.

Huffman trees assign short binary codes to frequent words, and this further reduces the number of output units that need to be evaluated. Because the Huffman trees are balanced, it only requires $\log_2(V)$ outputs to be evaluated. The Huffman tree based hierarchical softmax only requires to compute about $\log_2$(Unigram perplexity(V)) outputs, where

$$\text{Unigram perplexity(V)} = \left(\prod_{i=1}^{V} P\left(w_i\right)\right)^{-\frac{1}{V}},$$

which is exponentiated average negative log likelihood assuming the words are independent of each other. The $P(w_i)$ could, for example, be estimated based on the frequency of the

words in the training corpus.

Another important strategy in word2vec is the Negative sampling. The reason behind of this is that the size of our word vocabulary is big. With w_j and w_i go through all words in the dictionary, the computation is still not affordable. Many word pairs involving most frequent words do not have much meaning. For example, for 'the dog jumped on the brown fox', word pairs {dog, the}, {brown, the}, {brown, on }, {fox, the} do not really contribute to the learning. The Negative sampling addresses this by having each training sample only modifies a small percentage of the parameters, rather than all of them.

With negative sampling, we randomly select just a small number of 'negative' words for each word (say 5 - 20 words work well for smaller datasets, 2 - 5 words for large datasets). Afterward, use logistic regression for binary class rather than multi-class. For example, with d=300, $|V| = 10000$, the total number of unknown parameters is 300×10000. For each word, the parameter update will be done for one positive word, and 5 negative word, i.e., 6 total per word. With d=300, we update 1800 parameters rather than 3 million.

The Negative samples are chosen using word frequency distribution as follows:

- Discard the samples with frequent word (e.g., {dog, the} since 'the' is a frequent word) using the following probability:

$$P(w_i) = 1 - \sqrt{t/f(w_i)},$$

  where $f(w_i)$ is the frequency of word w_i and t is some number for tuning ($\approx 10^{-5}$).

- The negative samples are chosen with the following probability (The power 3/4 increases the probability for less frequent words):

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=1}^{|V|} f(w_j)^{\frac{3}{4}}}$$

The authors presented some observations and recommendations on the word2vec. The

recommended value for the size of the context window m is 10 for skip-gram, and 5 for CBOW. Typically, the dimensionality of the vectors is set to be between 100 and 1000. Quality of word embedding increases with higher dimensionality, however, after reaching some point, the gain is marginal. The Hierarchical softmax works better for infrequent words while negative sampling works better for frequent words and better with low dimensional vectors. As training epochs increase, the Hierarchical softmax stops being useful. The CBOW model is faster while skip-gram does a better job for infrequent words. Altszyler et al. (2017) studied word2vec performance in two semantic tests for different corpus size. They found that the word2vec has a steep learning curve, outperforming another word-embedding technique Latent Semantic Analysis (LSA) when it is trained with medium to large corpus size (more than 10 million words). With a small training corpus, however, the LSA showed better performance. Additionally, they show that the best parameter setting depends on the task and the training corpus.

2.1.2 The GloVe model

Contrary to the word2vec, which uses the cooccurrence of words indirectly in modeling, the GloVe model incorporates the cooccurrence of words directly in modeling. The GloVe is a model for using Global Vectors to give distributed word representations. It is a log-bilinear regression model for unsupervised learning of word representations and combines the features of two model families, the global matrix factorization and local context window methods. We will review the details of the GloVe algorithm in this subsection.

The GloVe collects word co-occurence counts to form a word co-ocurrence matrix $\mathbf{X}$. Entry X_{ij} represents how often word i appears in context of word j. In the authors' experiments, they used a context of ten words to the left and ten words to the right, which is referred as window size. The word pairs that are d words apart contribute $1/d$ to the total count. This choice accounts for the fact that distant word pairs are expected to contain less information about the words' relationship to one another. Consider only the nonzero

elements in matrix $\mathbf{X}$. The $\log(X_{ij})$ depends on $\log(X_i)$ and $\log(X_j)$, where X_i and X_j are the total counts for words i and j respectively.

Let w_i be the vector for the main word, $\tilde{w}_j$ be the vector for the context word. Then $\log(X_{ij})$ is proportional to the dot product $w_i^T \tilde{w}_j$. The dot product takes the local context into account by enforcing the rule that the embedding vectors of words i and j are similar if they frequently co-occur in similar context and vice versa. The GloVe model is the log bilinear model below

$$\log(X_{ij}) = w_i^T \tilde{w}_j + \log(X_i) + \log(X_j) + \text{ error.}$$

where $\log(X_i)$ and $\log(X_j)$ are bias terms that only depend on words i and j respectively. During modeling, they are replaced by b_i, b_j. The w_i and $\tilde{w}_j$ are both unknown parameters that need to be estimated. The cost function to estimate the unknown parameters is

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f\left(X_{ij}\right) \left(w_i^T \tilde{w}_j + b_i + b_j - \log X_{ij}\right)^2,$$

where

$$f\left(x\right) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^{\frac{3}{4}} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

with $x_{\max} = 100$ (see Pennington et al. 2014). Note that the training corpus contains billions of words in it. In such a large corpus, $x_{\max} = 100$ is only separating those extremely rare words from other words. For majority of word pairs that frequently appear together, $f(x)$ is one, which means sum of squares of errors is used as the loss function. For word pairs that rarely co-occurs together, the function $f(x)$ provides lower weight than 1, but serves as a boost in the sum of squares of errors from these rare word pairs. The weighting function f was chosen to have the following properties:

- $f(0) = 0$ and should vanish fast enough such that the $\lim_{x \to 0} f(x) \log^2(x)$ is finite.
- $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.

The GloVe minimizes the cost function with the AdaGrad (Duchi et al. 2011), stochastically sampling non-zero elements from $\mathbf{X}$, with initial learning rate of 0.05. They ran 50 iterations for vector dimensions ≤ 300, and 100 iterations otherwise. In the end, the model generates two sets of word vectors, W and $\tilde{W}$. When $\mathbf{X}$ is symmetric, W and $\tilde{W}$ are equivalent and differ only as a result of their random initializations. The sum $W + \tilde{W}$ was taken as the word vectors, which helps to reduce overfitting and noise and generally improve results in downstream tasks (Ciresan et al. 2012).

In summary, the GloVe model made a revolutionary move by combining counts for the same word pairs from all context. It significantly reduced computational demand compared to the word2vec. However, the GloVe also has some drawbacks in its loss function. **The loss function is just sum of squares of errors with some extra care on the extremely rare word pairs. For word pairs with no cooccurrence count observed, the loss function totally ignored whether the model fits the data or not. Further problem with the GloVe is that the variation of the cooccurence counts is highly likely to be non-constant. The least squared error loss is not suitable in such case.**

2.2 Transformer, BERT and its variants

2.2.1 The Transformer model

The Transformer (Vaswani et al. 2017) is one of the most popular deep learning architectures that is used for NLP tasks. The transformer architecture lays ground for the popular BERT model. Success of the transformer models also encouraged application of transformers in audio and video processing. A major advantage of the transformer models is that they can capture long term dependency through the use of multi-head attentions. This effectively

solved the vanishing gradient problem that has plagued convolutional neural network (CNN), recurrent neural network (RNN), long short term memory (LSTM) network and prevented deep networks from learning effectively.

The transformer consists of an encoder-decoder architecture. We feed the input sentence to layers of encoders. The encoders learn the representation of the input sentence and send the representation to layers of decoders. Each of decoders combines the information from encoder output and previous decoders to generate the output sentence (target sentence).

The entire transformer architecture contains 6 encoders $E_1, \ldots, E_6$, and 6 decoders $D_1, \ldots, D_6$, a final linear combination, and softmax conversion to probabilities. The inputs and outputs of all encoders and decoders are of dimension $d = 512$. All 6 encoders are stacked together in serial. E_1's inputs are pre-trained word embeddings with positional encoding added to it. E_i's inputs are the output from $E_{i-1}, i = 2, \ldots, 6$. All 6 decoders are stacked together in serial. D_1 uses the output of E_6 and embeddings of the target output word sequence with positional encoding added to it as inputs. D_i uses the output of E_6 and the output of D_{i-1} as its inputs, $i = 2, \ldots, 6$.

Each encoder has 2 sub-layers:

(1) multi-head self attention which is scaled dot product.

(2) fully connected feed-forward network with ReLU activation.

Each sub-layer has a residual connection around it. That is, the output of each sub-layer is added back to its input and then followed by layer normalization (Ba et al. 2016).

There are three matrices of unknown parameters to be learned, the query matrix Q, the key matrix K and the value matrix V. The attention mechanism contains the following:

(1) Compute the dot product between the query matrix and the key matrix. This becomes the similarity scores.

(2) Divide by the square root of the dimension of the key vector to scale it.

(3) Apply the softmax function to normalize the scores and obtain the score matrix.

(4) Compute the attention matrix by multiplying the score matrix by the value matrix.

The attentions can be computed parallelly. The resulting attentions are concatenated and multiplied by a new weight matrix to create the final attention matrix. Multi-head attentions allow the model to jointly attend to information from different representation subspaces at different positions.

The decoders are similar to the encoders except for the following:

- Beyond the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head "encoder-decoder attention", in which the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.

- The self-attention sub-layer in the decoder stack was modified to only allow to attend to earlier positions in the output sequence. This is done by masking future positions before applying the softmax step. The masking ensures that the predictions for position i can depend only on the known outputs at positions less than i.

The Transformer is the first sequence transduction model based entirely on attention. It replaced recurrent layers commonly used in encoder-decoder architectures with multi-head self-attention. The self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d. Code for Transformer is available publicly at https://github.com/tensorflow/tensor2tensor. Transformer is also extended to image and audio analysis. See Image Transformer https://arxiv.org/abs/1802.05751 and Music Transformer https://arxiv.org/abs/1809.04281 for some examples.

2.2.2 BERT (Bidirectional Encoder Representations from Transformers)

BERT model is a multi-layer bidirectional Transformer encoder. It has the same implementation as original Autoencoder Transformer. BERT takes continuous word vectors from WordPiece embeddings (Wu et al. 2016) with a 30,000 token vocabulary at the beginning. The WordPiece embeddings are known to be effective in handling rare words and in enhancing accuracy of model. To use the BERT model, we feed a sequence of texts into the model. The sequence is then tokenized by using WordPiece tokenizer. The sequence can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. Sentence pairs are packed together into a single token sequence so that the BERT can model both a single sentence and a pair of sentences in one sequence to handle a variety of down-stream tasks such as Question Answer. Hence, a 'sequence' refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together.

Then the input of the BERT model are computed as sum of three embeddings: token embeddings, segment embeddings, position embeddings. The segment embeddings indicate which sentence does the token belongs to and the position embeddings hold information about position of each token in the sequence. The token embeddings are learned by pre-training on text data from BooksCorpus having 800M words and from Wikipedia having 2,500M words. Then it learns new embeddings for tokens in the sequence by doing pre-training on task specific text. The pre-training basically refines the token representations using their contextual information presented in the sequence. If there are downstream tasks such as sentiment classification then fine-tuning model is done by adding an additional layer using the output from the pre-training as inputs. The pre-training does 2 jobs. One is masked language model (MLM). The other one is next sentence prediction (NSP). When training the BERT model, MLM and NSP are trained together, with the goal of minimizing the combined loss function of the two strategies.

- In the masked language model, to avoid trivial prediction of the target word because of seeing itself in a multi-layered context using left-to-right or right-to-left training, the mask language model masks 15% of all WordPiece tokens in each sentence at random. Then the pre-training predicts the masked words.

- For the next sentence prediction task, they choose sentence pairs A and B from any monolingual corpus. The sentence B was chosen that 50% of time is the actual next sentence and the other 50% of time is random sentence from corpus. The two sentences are put together as single sequence for the task. Pre-training of this task is very beneficial to downstream tasks such as Question Answering and Natural Language Inference. For this NSP task, the entire input sequence goes through the Transformer model. The output of the [CLS] token is transformed into a 2×1 vector, using a simple classification layer with the probability of 'IsNext' calculated using softmax.

With the WordPiece embeddings, the first token of every sequence is always a special classification token ([CLS]). Different sentences are separated with a token [SEP]. The final hidden state corresponding to [CLS] is used as the aggregate sequence representation for classification tasks. The final hidden vector of the i^{th} input token is the vector representation of the token learned from the BERT model.

In the fine-tuning, task-specific models are formed by incorporating BERT with one additional output layer. For each task, we simply plug in the task specific inputs and outputs into BERT and fine tune all the parameters. The learning rate in the fine tuning task is generally taken to be very small such as 10^{-4}, 10^{-6} because the pre-trained token representations were already in good shape.

Comparing different Transformer models, they differ in model architecture as follows.

- Tranformer: 6 encoder layers, 512-dimensional states, 8 attention heads, 6 decoder layers.

- OpenAI transformer: 12 decoder layers, 768-dimensional states, 12 attention heads.

For position-wise FFN part, BERT used 3072-dim inner states.

- BERT base: 12 encoder layers, 768-dimensional states, 12 attention heads (Total Parameters=110M, same size as OpenAI transformer).
- BERT large: 24 encoder layers, 1024-dimensional states, 16 attention heads. For position-wise FFN part, used 4096-dim inner states. (Total Parameters=345M).
- The Transformer and OpenAI transformer use constrained self-attention where every token can only attend to context to its left. The BERT transformer uses bidirectional self-attention.

In summary, BERT is a breakthrough in the use of machine learning in NLP related tasks. It uses bidirectional embeddings from Transformers to generalize deep bidirectional language model architectures for learning from context. BERT allows the same pre-trained model to successfully tackle a broad set of downstream NLP tasks by adding just one additional output layer. There is no need to do substantial task-specific architecture modifications. It obtains new state-of-the-art results on eleven natural language processing tasks. BERT is simple and powerful, marking the beginning of a new era for NLP.

In terms of training, more steps in pre-training help achieve better fine-tuning accuracy. BERT BASE achieves almost 1.0% additional accuracy on Multi-Genre Natural Language Inference when trained on 1M steps compared to 500k steps. MLM pre-training converges slower than left-to-right pre-training, since only 15% of words are predicted in each batch. But MLM shows improved accuracy very quickly. BERT pre-training takes days (such as 4 days on 16 Cloud TPUs). Its fine-tuning of all results in the paper takes at almost 1 hour on a single Cloud TPU or a few hours on a GPU.

2.2.3 Variants of BERT

Despite being successful in obtaining state-of-the-art performance using the BERT model in many NLP tasks, the BERT training time is long due to millions of parameters and

hence inspired many machine learning researchers to develop variants of BERT. ALBERT, RoBERTa, ELECTRA, SpanBERT, DistilBERT, and TinyBERT are some popular examples of the variants. Below we give brief review of these variants. For more detailed explanation and application examples, we refer the readers to Ravichandiran (2021).

ALBERT (Lan et al. 2020) is a lite version of BERT model aiming to minimize training time. It includes a few architecture changes to reduce the number of parameters. Specifically, it uses cross-layer parameter sharing and factorized embedding layer parameterization. With the cross-layer parameter sharing, not all parameters of all encoder layers are learned. Instead, ALBERT only learns the parameters of the first encoder layer, and then just share the parameters of the first encoder layer with all the other encoder layers. The factorized embedding layer parameterization reduces the number of parameters by first projecting the one-hot-encoded vectors of vocabulary V to the low dimensional WordPiece embedding space $E = 128$. The dimension of the WordPiece embedding becomes $V \times E = 30,000 \times 128$. Then, the WordPiece embeddings were projected onto a hidden layer of dimension $H = 768$. That is, the dimension is $E \times H = 128 \times 768$. Beyond reducing the number of parameters, ALBERT replaces the NLP task with the Sentence Order Prediction task which is based on inter-sentence coherence instead of topic prediction.

RoBERTa (Robustly Optimized BERT pre-training Approach) (Liu et al. 2020) is another popular variant of BERT. It was developed because researchers found that BERT is severely undertrained. They modified BERT with the following changes in pre-training: use dynamic masking instead of static masking in the MLM task; remove the NSP task; train with a large batch size; use Byte-level Byte Pair Encoding as a tokenizer.

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) (Clark et al. 2020) uses replaced token detection in pre-training stage. The replace token detection substitutes a token with a different token and train the model to classify whether the given tokens are actual or replaced tokens, instead of masking a token with the [MASK] token in MLM. ELECTRA also removed the NSP task from the pre-

training. Google provided the pre-trained ELECTRA model on Github https:/github.com/google-research/electra. It provides three different configurations: 'ELECTRA-small' with 12 encoder layers and 256 hidden size; 'ELECTRA-base' with 12 encoders and 768 hidden size; 'ELECTRA-large' with 24 encoders and 1,024 hidden size.

SpanBERT (Joshi et al. 2020) mostly focuses on Question answering task, especially the span of text. Instead of masking tokens at random positions, SpanBERT masks the random contiguous span of tokens. Then, the tokens are fed to SpanBERT and we get the representation of the tokens. To predict the span of the masked tokens, the SpanBERT model is trained with the MLM objective and a new objective called Span Boundary Objective (SBO) at a same time. In the SBO, the model only uses the representation of the tokens present in the span boundary and the position embedding of the masked token. Therefore, the loss function of SpanBERT is the addition of the MLM loss and SBO loss.

A large number of parameters and also long training time makes it hard to apply the pre-trained BERT to smaller devices having limited resources such as cell phones. DistilBERT (Sanh et al. 2019) was designed to overcome the limitation by transferring knowledge from a large pre-trained BERT to a small BERT using knowledge distillation. The knowledge distillation, also referred to as teacher-student learning, is a model compression technique in which a small model is trained to reproduce the behavior of a large pre-trained model. The large pre-trained model is the teacher and the small model is the student. A softmax function with temperature is used to extract knowledge from large network and transfer it to student network: $P_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$. When $T = 1$, this formula gives the same probability distribution as the standard softmax function. By increasing the value of T, the model retrieves a smoothed probability distribution, which gives more information about other classes.

To train the student network, the input sentence was fed to both student and teacher network with $T > 1$. The output of the teacher network is referred as soft target because the teacher network is pre-trained network. The prediction from untrained student network

is called soft prediction. DistilBERT compute the cross-entropy loss between the soft target and soft prediction and train the student network by minimizing the cross-entropy loss (also known as distillation loss). Beyond the soft target and soft prediction with $T > 1$, the model also gets hard target from the teacher network and hard prediction from the student network by setting $T = 1$. Then we compute the student loss as the cross-entropy between the hard prediction probability and hard target 0 or 1. The final loss function is the weighted sum of student loss and distillation loss. The student network is trained by minimizing the above loss function. In summary, the pre-trained network from larger network serves as the teacher network. Knowledge distillation passes the information in the teacher network to the student network by minimizing the weighted sum of student and distillation loss. DistilBERT's performance is comparable to the original BERT BASE model as it performed with almost 97% accuracy of the original BERT BASE model. Because of its lightness, DistilBERT conducts downstream inference task 60% faster than the original BERT. This makes it easy to deploy on small edge devices. The pretrained DistilBERT is available publicly by Hugging Face which trained the model on eight 16 GB V100 GPUs for approximately 90 hours. Some experiments such as question-answering task was done by researchers on the pretrained DistilBERT with iPhone 7 Plus edge device. Its inference time was 71% faster than BERT BASE model, and its model weights were only 207 MB.

TinyBERT (Jiao et al. 2019) is another variant of BERT that also uses the knowledge distillation. In TinyBERT, beyond transferring knowledge from the output layer of the teacher BERT to the student BERT, it also transfers knowledge from the embedding encoder layers. This helps the student BERT to learn more information from the teacher. Additionally, TinyBERT uses a two-stage learning framework that applies the knowledge distillation in both the pre-training and fine-tuning stage.

2.3 Models for Recommender systems

A Recommender system makes recommendations for items that are more relevant to a particular user. Depending on the domain, the recommendation may refer to what product to purchase, what video to watch from Youtube, what articles to read from web engines, etc. These are only a few examples. The Recommender systems are beneficial when a particular user has to make a choice out of numerous, sometimes overwhelming number of options. There are broad applications for the Recommender systems particulary in digital age. For example, online stores rely on recommendations from the Recommender systems to suggest products to highly interested buyers. Netflix uses the Recommender systems to recommend movies or TV series to matching viewers. How well Recommender systems perform is directly linked to customer satisfaction and loyalty which are significant factors for business profits.

Depending on the application domain, the data to be used for Recommender systems may differ a lot in the format and development stage. In the early stage of Recommender systems, researchers depend on manually created data such as user profile and item profile and manually labeled properties. The item profile represents important characteristics regarding the item. For example, the directors and actors of a movie, or genre and type of a movie are important item profile characteristics of a movie which might attract certain users. Obtaining or curating such data is laborious and difficult to be in large scale. In the past decade, the data collection for Recommender systems has evolved to implicit data collecting through collaborative effort of the public. With implicit data collection, the records could be items that a user views in an online store and the time spent viewing an item, search patterns, browsing history, etc. In the example of music or video recommendations, the data to be used in their systems are sequences of musics or videos watched by users. In the example of online shopping, the data are customers' rating on product purchased or a list of products purchased by each customer. In the example of online browsing, the data are a collection of articles or news read/clicked/browsed by a user. Some of the data may be just a collection

of items together at a single time point while some of the data may have time or spatial order. A sequence of text is an example of data having spatial order in that the words are positioned in relative order which provides linguistic meaning and grammatical arrangement. A sequence of movies watched by a user shows time signature of a user's watching list.

Earlier efforts that rely on user profiles or item profiles are referred as content filtering. As we have known, there is difficulty in obtaining the profile data, hence content filtering approach may not be able to make a recommendation when the data are not available. The other branch of Recommender systems are referred as collaborative filtering which includes two main streams: neighborhood based approach and latent factor models. In neighborhood methods, the users or items are grouped by nearest neighbors. For example, neighborhoods of a product can be other products that get similar ratings from the same user. Neighborhood based methods are not as successful as the latent factor models.

Latent factor models try to capture underlying factors for users and items, and use the latent factors for predicting ratings or preferences on a item for a user. The dot product between the latent factors of a user and item is used for predicting a rating of the user's interaction on the item. Many successful latent factor models are based on matrix factorization methods which utilize the dot product between the latent factors. For matrix factorization models, their input data matrix $\mathbf{X}$ is comprised of rows representing users and columns representing items so that each cell in the matrix, i.e., X_{ij}, is observed ratings from user-item interaction. Then the models try to map both users and items to d dimensional latent factor space. That is, $\mathbf{u}_i \in R^d$ represents latent factor for i^{th} user, and $\mathbf{p}_j \in R^d$ represents latent factor for j^{th} item. The dot product between these two vectors can be interpreted as interaction, which reflects the user's interest on the item, between the i^{th} user and j^{th} item:

$$\hat{r}_{ij} = \mathbf{u}_i^\top \mathbf{p}_j, \qquad \text{where } \hat{r}_{ij} = \text{ estimated rating of user } i \text{ on item } j.$$

Once the models are done with mapping/estimating the latent factors $\mathbf{u}_i$ and $\mathbf{p}_j$, ratings can

be easily predicted. However, the challenging point is estimating the factor vectors.

Earlier models to estimate the hidden factor vectors tried to use singular value decomposition (SVD) to decompose the input user-item matrix $\mathbf{X}$. The SVD method easily encounters problem due to sparseness of the user-item matrix. When the method only uses the observed cells in the matrix, it could easily lead to overfitting. To overcome this problem, researchers tried to impute missing cells in the data matrix but the imputation made computation extensive and created unintended bias. More recent studies suggest to use regularized methods which could avoid overfitting while only using observed ratings. The unknown values in the matrices can be estimated by minimizing the objective function via a gradient descent method:

$$\min_{\mathbf{u},\mathbf{p}} \sum_{(i,j)\in \mathrm{K}} \left(r_{ij} - \mathbf{u}_i^\top \mathbf{p}_j\right)^2 + \lambda \left(\|\mathbf{u}_i\|^2 + \|\mathbf{p}_j\|^2\right),$$

where K is a set that contains all observed user-item ratings. The matrix factorization methods can be extended by including bias terms as

$$\hat{r}_{ij} = \mu + b_i + e_j + \mathbf{u}_i^\top \mathbf{p}_j, \qquad \text{where } \hat{r}_{ij} = \text{ estimated rating of user } i \text{ on item } j.$$

where b_i and e_j are user main effect and item main effect respectively, and μ is overall mean. Then the parameters can be estimated by minimizing the following objective function

$$\sum_{(i,j)\in \mathrm{K}} (r_{ij} - \hat{r}_{ij})^2 + \lambda \left(\|\mathbf{u}_i\|^2 + \|\mathbf{p}_j\|^2 + b_i^2 + e_j^2\right).$$

This model can be even further extended by considering time varying coefficients such as

$$\hat{r}_{ij}(t) = \mu + b_i(t) + e_j(t) + \mathbf{u}_i^\top(t)\mathbf{p}_j,$$

where $\hat{r}_{ij}(t)$ is the estimated rating of user i on item j at time t. The $b_i(t)$ and $\mathbf{u}_i(t)$

represent the user's preference on the item over time, and the $e_j(t)$ reflects the trend of item popularity over time. Again the parameters could be estimated via gradient descent methods by minimizing the regularized mean squared error. Alternating least squares (ALS) method could also be used for optimizing the aforementioned objective functions. In ALS method, it first optimizes with respect to the user matrix while holding the item matrix fixed. Afterward, it optimizes with respect to the item matrix while holding the user matrix fixed. The gradient descent method tends to be faster but ALS method has the potential of allowing parallel processing. See Koren et al. (2009) for further details.

After witnessing many deep-learning models show success in areas such as NLP, image processing, some authors considered to adapt machine learning models to recommender systems. One example is Deep neural network (DNN) softmax recommender system. This system treats the problem as a multiclass prediction. There are as many classes as the number of items. The input is user query which can be dense or sparse features. The output is the predicted probability of a user's interaction with each item. The DNN softmax recommender system has many hidden layers and nonlinear activation functions. The actual outcome is that the nonlinear interaction between the user and item is modeled by the inner product of nonlinear functions of user's embedding or/and nonlinear functions of item's embedding (See Covington et al. 2016 and the references therein). Google researchers found significant improvement of recommendation accuracy when DNN softmax recommender system is used.

The machine learning scientists at Netflix also actively pursued deep-learning models in Netflix recommender systems. Unfortunately, after long struggle, the Netflix recommender system research community reported that the use of powerful deep-learning models not necessarily translate into good recommender systems (Steck et al. 2021). Instead, some weakness of deep-learning models are observed while applied to recommender systems. One example is overfitting. With Netflix's short term objectives, the deep-learning models may be misaligned with long term objectives such as user satisfaction. It has been found that adapting the deep-learning models within the Netflix recommender systems provided very limited

benefits. Instead, well-tuned traditional methods remained to be very strong competitive baseline performer in the classic recommendation setting when only user-item interaction data are used. To get more benefit from deep-learning models, the problems need to be extended such as by finding good representations for the time domain, or extending the range and modalities of inputs with additional heterogeneous features and information sources such as images, texts, and videos. Solving the traditional recommender system problems with deep-learning sometimes only has limited benefits.

In Recommender systems, unobserved cells in the data matrix are possibly because the users did not have a chance to provide rating on the items. Such observations are missing. This is different from zero entries in word-word cooccurrence count matrix in NLP setting. A zero count for a word pair means this pair of words were never used in the same window in the entire corpus. Excluding the model fit for zero count in the objective function is not suitable in NLP setting because it could lead to large fitted value for a zero and this discrepancy is never taken into account in the objective function if zero entries are ignored (as in GloVe).

Chapter 3

Shared parameter alternating zero-inflated Gamma regression

In this chapter, our goal is to model continuous data with skewed distribution with abundance of zeros but lack of covariate information. The zero inflated gamma observations are sometimes referred as semi-continuous data because the observations contain a lot of zeros and the distribution of non-zero are highly skewed continuous data.

Some examples of such data are listed in Mills (2013): insurance claim data, household expenditure data, precipitation data, etc. Wei et al. (2020) studied and simulated data using the actual deconvolved calcium imaging data using zero inflated gamma to accommodate spike of observed inactivity and trace of calcium signals in neural populations. Nobre et al. (2017) studied amount of leisure time spent on physical activity and the explanatory variables including gender, age, education level, and annual per capita family income. The zero inflated gamma model was preferred over multinomial model.

In all of the aforementioned examples, there are explicitly observed covariates or factors and the two parts of the model parameters are perpendicular to each other such that model estimation can be done by simply fitting two separate regressions: binomial regression and gamma regression. Such models, unfortunately, can not be applied to the user-item or cooc-

currence count matrix data because we do not have covariates being observed. Additionally, the mechanism that leads to the observed data entries in the data matrix for the binomial and the gamma part may be of the same nature, in which case shared parameter modeling would be more appropriate. Therefore, in this chapter, we consider shared parameter modeling of zero inflated gamma data using alternating regression. We will consider two different link functions for the gamma part: canonical link and log link.

Denoting the Bernoulli random variable g_{ij} and the Gamma random variable y_{ij} as

$$g_{ij} = \begin{cases} 0, & \text{if} \quad 1 - p_{ij} \\ 1, & \text{if} \quad p_{ij} \end{cases}, \qquad y_{ij} = \begin{cases} 0, & \text{if} \quad g_{ij} = 0 \\ \text{Gamma observation}, & \text{if} \quad g_{ij} = 1 \end{cases},$$

we have probability mass function (pmf) for the Bernoulli random variable and probability density function (pdf) for the Gamma random variable as follows:

$$\begin{aligned} p\left(g_{ij}\right) &= p_{ij}^{g_{ij}} \cdot \left(1 - p_{ij}\right)^{1 - g_{ij}}, \\ f\left(y_{ij} \mid g_{ij} = 1\right) &= \frac{1}{\Gamma\left(v_i\right)} \cdot \left(\frac{v_i y_{ij}}{\mu_{ij}}\right)^{v_i} \cdot \frac{1}{y_{ij}} \cdot \exp\left(-\frac{v_i y_{ij}}{\mu_{ij}}\right). \end{aligned}$$

Product of the two functions generates the joint distribution of the Bernoulli and Gamma random variable

$$f\left(y_{ij}, g_{ij}\right) = p\left(g_{ij}\right) \cdot f\left(y_{ij} \mid g_{ij}\right).$$

By integrating out the Bernoulli random variable, we could obtain pdf of the zero inflated Gamma (ZIG) distribution as below.

$$\begin{aligned} f\left(y_{ij}\right) &= \sum_{x=0}^{1} p\left(g_{ij} = x\right) \cdot f\left(y_{ij} \mid g_{ij} = x\right) \\ &= \left(1 - p_{ij}\right)^{I(y_{ij}=0)} \cdot \left[p_{ij}\ f\left(y_{ij} \mid g_{ij} = 1\right)\right]^{I(y_{ij}>0)} \end{aligned}$$

From the pdf of the ZIG distribution, we could define likelihood function L on all observations with the following structures under the assumption that the observations y_{ij}'s are independent of each other conditional on unobserved $\boldsymbol{\theta}_i$ and $\widetilde{\boldsymbol{\theta}}_j$, $i = 1, \ldots, n, j = 1, \ldots, n$.

$$L = \prod_{i=1}^{n} L_i; \qquad \ell = \log(L) = \sum_{i=1}^{n} \ell_i; \qquad \ell_i = \sum_{j'=1}^{n} \ell_{ij'}; \quad \text{where} \quad \ell_{ij'} = \log(f(y_{ij'})).$$

We will estimate $\boldsymbol{\theta}$ and $\widetilde{\boldsymbol{\theta}}$ alternately. This resembles block coordinate descent in which one part of parameters is updated while holding the remaining part of parameters as fixed values. In this estimation scheme, the likelihood function can be treated as a function of either $\boldsymbol{\theta}$ or $\widetilde{\boldsymbol{\theta}}$ but not concurrently at a same time, where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)^\top$, $\widetilde{\boldsymbol{\theta}} = (\widetilde{\boldsymbol{\theta}}_1, \ldots, \widetilde{\boldsymbol{\theta}}_n)^\top$, $\boldsymbol{\theta}_i = (\mathbf{w}_i^\top, b_i, e_i)^\top$ and $\widetilde{\boldsymbol{\theta}}_i = (\widetilde{\boldsymbol{w}}_i^\top, \widetilde{b}_i, \widetilde{e}_i)^\top$. When estimating $\boldsymbol{\theta}$, we treat the likelihood as a function of $\boldsymbol{\theta}$ while $\widetilde{\boldsymbol{\theta}}$ serves as given data. Reversely, when we estimate the $\widetilde{\boldsymbol{\theta}}$, we treat the likelihood as a function of $\widetilde{\boldsymbol{\theta}}$ while $\boldsymbol{\theta}$ is treated as given. For clarity, we use separate notations ℓ_i and $\widetilde{\ell}_j$ to refer to these two occasions, i.e.,

$$\ell_i = \ell_i(\boldsymbol{\theta}_i; \widetilde{\boldsymbol{\theta}}) = \sum_{j=1}^{n} \ell_{ij}(\boldsymbol{\theta}_i; \widetilde{\boldsymbol{\theta}}_j), \qquad \widetilde{\ell}_j = \widetilde{\ell}_j(\widetilde{\boldsymbol{\theta}}_j; \boldsymbol{\theta}) = \sum_{i=1}^{n} \ell_{ij}(\widetilde{\boldsymbol{\theta}}_j; \boldsymbol{\theta})$$

Note that ℓ_i and $\widetilde{\ell}_j$ simply represent the likelihood of a row or a column of the data matrix. We use ℓ_i to refer to a function of $\boldsymbol{\theta}_i$ while the log likelihood is for data in the i^{th} row of the co-occurence matrix and use $\widetilde{\ell}_j$ to refer to a function of $\widetilde{\boldsymbol{\theta}}_j$ while the log likelihood is for data in the j^{th} column of the matrix. Each of these two likelihood functions could be split into two components: one corresponds to the Bernoulli part and the other corresponds to

the Gamma part

$$\begin{aligned}
\ell_i &= \ell_i(\boldsymbol{\theta}_i;\widetilde{\boldsymbol{\theta}}) = \ell_i(\boldsymbol{\theta}_i \text{ while } \widetilde{\boldsymbol{\theta}} \text{ is known }) = \ell_i^{(1)} + \ell_i^{(2)}, \\
\ell_i^{(1)} &= \ell_i^{(1)}(\boldsymbol{\theta}_i;\widetilde{\boldsymbol{\theta}}) = \ell_i^{(1)}(\boldsymbol{\theta}_i \text{ while } \widetilde{\boldsymbol{\theta}} \text{ is known }) \\
&= \sum_{j=1}^{n} \left\{ I\left(y_{ij}=0\right) \cdot \log\left(1-p_{ij}\right) + I\left(y_{ij}>0\right) \cdot \log p_{ij} \right\}, \\
\ell_i^{(2)} &= \ell_i^{(2)}(\boldsymbol{\theta}_i;\widetilde{\boldsymbol{\theta}}) = \ell_i^{(2)}(\boldsymbol{\theta}_i \text{ while } \widetilde{\boldsymbol{\theta}} \text{ is known }) = \sum_{j=1}^{n} I\left(y_{ij}>0\right) \cdot \log f\left(y_{ij} \mid y_{ij}>0\right).
\end{aligned}$$

Similarly,

$$\begin{aligned}
\widetilde{\ell}_j &= \widetilde{\ell}_j(\widetilde{\boldsymbol{\theta}}_j;\boldsymbol{\theta}) = \widetilde{\ell}_j(\widetilde{\boldsymbol{\theta}}_j \text{ while } \boldsymbol{\theta} \text{ is known }) = \widetilde{\ell}_j^{(1)} + \widetilde{\ell}_j^{(2)}, \\
\widetilde{\ell}_j^{(1)} &= \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j;\boldsymbol{\theta}) = \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}} \text{ while } \boldsymbol{\theta} \text{ is known }) \\
&= \sum_{i=1}^{n} \left\{ I\left(y_{ij}=0\right) \cdot \log\left(1-p_{ij}\right) + I\left(y_{ij}>0\right) \cdot \log p_{ij} \right\}, \\
\widetilde{\ell}_j^{(2)} &= \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j;\boldsymbol{\theta}) = \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j \text{ while } \boldsymbol{\theta} \text{ is known }) \\
&= \sum_{i=1}^{n} I\left(y_{ij}>0\right) \cdot \log f\left(y_{ij} \mid y_{ij}>0\right),
\end{aligned} \tag{3.0.1}$$

and the alternating regression deals with two separate log likelihood functions:

$$\begin{aligned}
\ell(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}}) &= \sum_i \sum_j \ell_{ij} = \sum_i \sum_j (\ell_{ij}^{(1)} + \ell_{ij}^{(2)}) = \sum_i (\ell_i^{(1)} + \ell_i^{(2)}), \\
\widetilde{\ell}(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta}) &= \sum_i \sum_j \widetilde{\ell}_{ij} = \sum_i \sum_j (\widetilde{\ell}_{ij}^{(1)} + \widetilde{\ell}_{ij}^{(2)}) = \sum_j (\widetilde{\ell}_j^{(1)} + \widetilde{\ell}_j^{(2)}).
\end{aligned}$$

The $\ell_i^{(1)}$ part is the traditional log likelihood for binary logistic regression. The $\ell_i^{(2)}$ part is gamma log likelihood restricted to only positive observations. If the two parts do not share common parameters, then the estimation can be done separately as in Mills (2013). In our case, however, the $\ell_i^{(1)}$ and $\ell_i^{(2)}$ share some common parameters $\mathbf{w}_i$ and $\widetilde{\boldsymbol{w}}_j$. The link

function for the logistic model takes form

$$\eta_{ij} = \log \frac{p_{ij}}{1 - p_{ij}} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j \tag{3.0.2}$$

where $\mathbf{w}_i$ and b_i are unknown parameters while treating $\widetilde{\boldsymbol{w}}_j$ and $\widetilde{b}_j$ as known. For the gamma observations,

$$\log f\left(y_{ij} \mid y_{ij} > 0\right) = -\log \Gamma\left(v_{ij}\right) + v_{ij} \log\left(\frac{v_{ij}}{\mu_{ij}}\right) + \left(v_{ij} - 1\right) \log y_{ij} - \frac{v_{ij} y_{ij}}{\mu_{ij}}.$$

When we use the canonical link, the mean of the response variable and $\boldsymbol{\theta}$ and $\widetilde{\boldsymbol{\theta}}$ are connected through following formula,

$$\tau_{ij} = g(\mu_{ij}) = -\mu_{ij}^{-1} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j. \tag{3.0.3}$$

With the canonical link, the likelihood function and score equations are both function of the sufficient statistic. As the sufficient statistic carries all information about the unknown parameters, we can restrict our attention to the sufficient statistic without losing any information. This link function, however, could have difficulty in the estimation process. The natural parameter space is $\{\mu_{ij} : \mu_{ij} > 0\}$. The dot product part on the right hand side of equation (3.0.3) may yield an estimate of μ_{ij} out of this parameter space.

Another link function that is popularly used for Gamma regression is the log link $g(\mu) = \log(\mu)$ which gives the log linear model:

$$\tau_{ij} = g(\mu_{ij}) = \log(\mu_{ij}) = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j. \tag{3.0.4}$$

This model has better interpretation than the canonical link. For each unit increment in $\widetilde{\boldsymbol{\theta}}_{jk}$ or $\boldsymbol{\theta}_{ik}$, the mean increases multiplicatively by $\exp(\boldsymbol{\theta}_{ik})$ or by $\exp(\widetilde{\boldsymbol{\theta}}_{jk})$. Even though the parameters enjoy better interpretation, the estimation could also become a problem in the

sense that the dot product on the right hand side of (3.0.4) could become large leading to μ_{ij} being infinity and hence the estimation algorithm diverges.

In both links, the common dot product $\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j$ is used to reflect the fact that the cosine similarity is the key driving force behind the observed data in the table. For example, in NLP, $\mathbf{w}_i$ and $\widetilde{\boldsymbol{w}}_j$ each represents a word in a dense vector. Each one of them contains both linguistic information and word usage information in it. The observations are cooccurrence count that is linked to relevance between the words and the relevance can be captured with the cosine similarity. In the example of item-item cooccurrence matrix, $\mathbf{w}_i$ and $\widetilde{\boldsymbol{w}}_j$ represent the product information of the items including characteristics, properties, functionality, popularity, users' ratings on them, etc. The dot product again tells how the two items are relevant to each other. Sometimes the cooccurrence matrix was derived from a time series such as a sequence of watched movies in time order from a customer. In this case, the cooccurrence count tells how often the two items were considered in a similar time frame because the counts were weighted based on the position separation of the two items in the sequence. The relevance of these items summarized by the cosine similarity could reflect how the two products are alike in their property, functionality, etc. Therefore, the dot product serves as major contributor for the observed count.

In both link functions, the intercepts are allowed to be different from that in the logistic model for flexibility. In some classical statistical models such as in Moulton et al. (2002), shared parameter modeling is used by assuming one part of model parameters to be proportional to the other. Such proportionally assumed parameters is meaningful for the case where covariates are observed. In our case, when the data $\widetilde{\boldsymbol{w}}$ is not actually observed, putting in an additional proportionality parameter will only make the model non-identifiable. This is why we believe it is better to put flexibility in the intercepts instead of using the proportionality parameters.

In our case, parameters of Gamma regression part and parameters of Logistic regression part should be a same set of parameters and estimated simultaneously instead of separate

set of parameters. Mills (2013) conducted hypothesis testing to compare two groups via mixture of zero-inflated gamma and zero-inflated log-normal models. However, the parameters from the two model components were modelled separately. In a totally applied setting, Moulton et al. (2002) considered normal and log-gamma mixture model for HIV RNA data using shared parameters assuming two parts of the model are proportional to each other. They proposed such application and simulation, but no inference was given. The shared parameter modeling was also employed in some literature to achieve some model parsimony. For example, Wu and Carroll (1988) used shared parameters for both a random effects linear model and a probit-modeled censoring process. Have et al. (1998) used a similar approach for simultaneous modeling of a mean structure and informative censoring. Albert et al. (1997) used shared parameters to model the intensity of a Poisson process and a binary measure of severity.

3.1 ZIG model with canonical links

In the two parts model, if we use canonical link, the Bernoulli part would be using logit function while the gamma part would be using negative inverse link. Using the canonical link with generalized linear models enjoys the benefit that the score equations are a function of sufficient statistics. Here, we present the model with the canonical links. The negative inverse link, however, has difficulty to interpret model parameters and also has some restrictions in terms of the support of the link function, which does not match the positive value of gamma distribution. More details can be seen as we introduce the model.

Recall that the the log likelihood and the link function for Logistic regression are

$$\ell_i^{(1)} = \sum_{j=1}^{n} \left\{ I\left(y_{ij} = 0\right) \cdot \log\left(1 - p_{ij}\right) + I\left(y_{ij} > 0\right) \cdot \log p_{ij} \right\},$$
$$\eta_{ij} = \log \frac{p_{ij}}{1 - p_{ij}} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j,$$

where $\mathbf{w}_i$ and b_i are unknown parameters while treating $\widetilde{\boldsymbol{w}}_j$ and $\widetilde{b}_j$ as fixed. The log likelihood and the negative inverse link function for Gamma regression are

$$\begin{aligned}
&\ell_i^{(2)} = \sum_{j=1}^{n} I\left(y_{ij} > 0\right) \cdot \log f\left(y_{ij} \mid y_{ij} > 0\right), \\
&\quad \text{with } \log f\left(y_{ij} \mid y_{ij} > 0\right) = -\log \Gamma\left(v_{ij}\right) + v_{ij} \log\left(\frac{v_{ij}}{\mu_{ij}}\right) + \left(v_{ij} - 1\right) \log y_{ij} - \frac{v_{ij} y_{ij}}{\mu_{ij}}, \\
&\tau_{ij} = g(\mu_{ij}) = -\mu_{ij}^{-1} = \mathbf{w}_i^{\top} \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j.
\end{aligned}$$

The log likelihood function for i^{th} row of data is

$$\ell_i \;\; = \;\; \ell_i^{(1)} + \ell_i^{(2)}.$$

To get partial derivatives, we write $\ell_i^{(1)}$ as follows

$$\begin{aligned}
\ell_i^{(1)} &= \sum_{j=1}^{n} \left(1 - g_{ij}\right) \cdot \log\left(1 - p_{ij}\right) + g_{ij} \cdot \log p_{ij} \\
&= \sum_{j=1}^{n} g_{ij} \cdot \log \frac{p_{ij}}{1 - p_{ij}} + \log\left(1 - p_{ij}\right) \\
&= \sum_{j=1}^{n} g_{ij} \cdot \eta_{ij} + \log \frac{1}{1 + \exp\left(\eta_{ij}\right)} \\
&= \sum_{j=1}^{n} g_{ij} \cdot \eta_{ij} - \log\left\{1 + \exp\left(\eta_{ij}\right)\right\}.
\end{aligned}$$

Note that p_{ij} is success probability from Logistic regression, the following formulae hold:

$$\begin{aligned}
p_{ij} &= \frac{\exp\left(\mathbf{w}_i^{\top} \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j\right)}{1 + \exp\left(\mathbf{w}_i^{\top} \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j\right)} \\
\frac{\partial \eta_{ij}}{\partial \mathbf{w}_i} &= \widetilde{\boldsymbol{w}}_j, \qquad \frac{\partial \eta_{ij}}{\partial b_i} = 1, \qquad \frac{\partial p_{ij}}{\partial b_i} = p_{ij}\left(1 - p_{ij}\right), \\
\frac{\partial p_{ij}}{\partial \mathbf{w}_i} &= \frac{\exp\left(\eta_{ij}\right) \cdot \widetilde{\boldsymbol{w}}_j \cdot \left\{1 + \exp\left(\eta_{ij}\right)\right\} - \exp\left(2\eta_{ij}\right) \cdot \widetilde{\boldsymbol{w}}_j}{\left\{1 + \exp\left(\eta_{ij}\right)\right\}^2} = p_{ij}\left(1 - p_{ij}\right) \widetilde{\boldsymbol{w}}_j.
\end{aligned}$$

Therefore, the first order partial derivatives can be summarized as

$$\begin{aligned}\frac{\partial \ell_i^{(1)}}{\partial \mathbf{w}_i} &= \sum_{j=1}^{n} \left\{ g_{ij} \cdot \widetilde{\boldsymbol{w}}_j - \frac{\exp(\eta_{ij})}{1+\exp(\eta_{ij})} \widetilde{\boldsymbol{w}}_j \right\} = \sum_{j=1}^{n} (g_{ij} - p_{ij})\, \widetilde{\boldsymbol{w}}_j, \\ \frac{\partial \ell_i^{(1)}}{\partial b_i} &= \sum_{j=1}^{n} (g_{ij} - p_{ij}).\end{aligned} \tag{3.1.1}$$

The second order partial derivatives and their negative expectations are

$$\begin{aligned}\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} &= -\sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j p_{ij}(1-p_{ij})\, \widetilde{\boldsymbol{w}}_j^\top = -\sum_{j=1}^{n} p_{ij}(1-p_{ij})\, \widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top \\ \Longrightarrow \quad E\left[-\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} \right] &= \sum_{j=1}^{n} p_{ij}(1-p_{ij}) \cdot \widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top,\end{aligned} \tag{3.1.2}$$

$$\begin{aligned}\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial b_i} = -\sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j \frac{\partial p_{ij}}{\partial b_i} = -\sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j p_{ij}(1-p_{ij}) \quad \Longrightarrow \quad E\left[-\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial b_i} \right] = \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j p_{ij}(1-p_{ij}), \\ \frac{\partial^2 \ell_i^{(1)}}{\partial b_i^2} = -\sum_{j=1}^{n} p_{ij}(1-p_{ij}) \quad \Longrightarrow \quad E\left[-\frac{\partial^2 \ell_i^{(1)}}{\partial b_i^2} \right] = \sum_{j=1}^{n} p_{ij}(1-p_{ij}).\end{aligned} \tag{3.1.3}$$

Now, consider the second component of ith log likelihood $\ell_i^{(2)}$

$$\begin{aligned}\ell_i^{(2)} &= \sum_{j=1}^{n} I(y_{ij} > 0) \cdot \log f(y_{ij} \mid y_{ij} > 0) = \sum_{j=1}^{n} g_{ij} \cdot \log f(y_{ij} \mid y_{ij} > 0) \\ &= \sum_{j=1}^{n} g_{ij} \cdot \Big\{ -\log\Gamma(v_{ij}) + v_{ij}\log v_{ij} + v_{ij}\log\left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j\right) \\ &\qquad + (v_{ij}-1)\log y_{ij} + v_{ij} y_{ij}\left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j\right) \Big\} \\ &= \sum_{j=1}^{n} g_{ij} \cdot \Big\{ v_{ij}\log\left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j\right) + v_{ij} y_{ij}\left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j\right) \\ &\qquad - \log\Gamma(v_{ij}) + v_{ij}\log v_{ij} + (v_{ij}-1)\log y_{ij} \Big\}.\end{aligned} \tag{3.1.4}$$

The first order partial derivatives for the $\ell_i^{(2)}$ are

$$\begin{aligned}\frac{\partial \ell_i^{(2)}}{\partial \mathbf{w}_i} &= \sum_{j=1}^{n} g_{ij} \cdot \left\{ \frac{-v_{ij} \cdot \widetilde{\boldsymbol{w}}_j}{-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j} + v_{ij} y_{ij} \cdot \widetilde{\boldsymbol{w}}_j \right\}, \\ \frac{\partial \ell_i^{(2)}}{\partial e_i} &= \sum_{j=1}^{n} g_{ij} \cdot \left\{ \frac{-v_{ij}}{-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j} + v_{ij} \cdot y_{ij} \right\}.\end{aligned} \tag{3.1.5}$$

The second order partial derivatives and their negative expectations are

$$\begin{aligned}&\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} = \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \widetilde{\boldsymbol{w}}_j \cdot \left(-\widetilde{\boldsymbol{w}}_j^\top \right) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j \right)^2 \\ \Longrightarrow \quad & E\left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} \middle| g_{ij} = 1 \right] = v_{ij} \sum_{\substack{j=1 \\ g_{ij}=1}}^{n} \widetilde{\boldsymbol{w}}_j \cdot \left(\widetilde{\boldsymbol{w}}_j^\top \right) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j \right)^2; \end{aligned} \tag{3.1.6}$$

$$\begin{aligned}&\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial e_i} = \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \widetilde{\boldsymbol{w}}_j \cdot (-1) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j \right)^2 \\ \Longrightarrow \quad & E\left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial e_i} \middle| g_{ij} = 1 \right] = v_{ij} \sum_{\substack{j=1 \\ g_{ij}=1}}^{n} \widetilde{\boldsymbol{w}}_j / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j \right)^2; \end{aligned} \tag{3.1.7}$$

$$\begin{aligned}&\frac{\partial^2 \ell_i^{(2)}}{\partial e_i^2} = \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot (-1) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j \right)^2 \\ \Longrightarrow \quad & E\left[-\frac{\partial^2 \ell_i^{(2)}}{\partial e_i^2} \middle| g_{ij} = 1 \right] = v_{ij} \sum_{\substack{j=1 \\ g_{ij}=1}}^{n} 1 / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j \right)^2. \end{aligned} \tag{3.1.8}$$

All of the aforementioned formulae are working with the i^{th} log likelihood $\ell_i^{(1)}$ and $\ell_i^{(2)}$. When we combine the log likelihood from different rows of data, other rows do not contribute to

the partial derivative with respect to $\boldsymbol{\theta}_i$. That is,

$$\frac{\partial \ell}{\partial \mathbf{w}_i} = \frac{\partial \ell_i}{\partial \mathbf{w}_i}; \quad \frac{\partial \ell}{\partial b_i} = \frac{\partial \ell_i^{(1)}}{\partial b_i}; \quad \frac{\partial \ell}{\partial e_i} = \frac{\partial \ell_i^{(2)}}{\partial e_i};$$
$$\frac{\partial^2 \ell}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} = \frac{\partial^2 \ell_i}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}; \quad \frac{\partial^2 \ell}{\partial \mathbf{w}_i \partial b_i} = \frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial b_i}; \quad \frac{\partial^2 \ell}{\partial \mathbf{w}_i \partial e_i} = \frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial e_i};$$
$$\frac{\partial^2 \ell}{\partial b_i^2} = \frac{\partial^2 \ell_i^{(1)}}{\partial b_i^2}; \quad \frac{\partial^2 \ell}{\partial b_i \partial e_i} = \frac{\partial^2 \ell_i}{\partial b_i \partial e_i} = 0; \quad \frac{\partial^2 \ell}{\partial e_i^2} = \frac{\partial^2 \ell_i^{(2)}}{\partial e_i^2}.$$

As a result, the estimation of the components in $\boldsymbol{\theta} = (\boldsymbol{\theta}_1^\top, \ldots, \boldsymbol{\theta}_n^\top)^\top$ does not need to happen at a same time. Instead, we can cycle through the estimation of $\boldsymbol{\theta}_i$, for $i = 1, \ldots, n$, one by one. After $\boldsymbol{\theta}_i$ is updated during estimation, the estimated value of $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_i$ will be used when we update $\boldsymbol{\theta}_k, k = i+1, \ldots, n$ and so on.

The first part of the alternating regression has the following updating equation based on the Fisher scoring algorithm:

$$\boldsymbol{\theta}_i^{(t+1)} = \boldsymbol{\theta}_i^{(t)} + S^{-1}_{\boldsymbol{\theta}_i^{(t)}} U_{\boldsymbol{\theta}_i^{(t)}}, \quad i = 1, \ldots, n, \tag{3.1.9}$$

For each i and t, this update requires the value of $\boldsymbol{\theta}_i$ and their score equations and information matrix at t^{th} iteration. One iteration alone here is not taking advantage of the data in the sense that retrieving the i^{th} row of data takes significant amount of time. Therefore, for each row of data retrieved, it is better to update for a certain number of times (epochs E). Specifically, for each epoch, the $\boldsymbol{\theta}_i^{(t)}$ is updated again and again and the updated values are used to recompute the score equations and information matrix, all of which are used for next epoch's update. After all epochs are completed, $\boldsymbol{\theta}_i^{(t)}$ takes the value of $\boldsymbol{\theta}_i^{(t,E)}$ at the end of all iterations from E epochs based on the updating formula below:

$$\boldsymbol{\theta}_i^{(t,k+1)} = \boldsymbol{\theta}_i^{(t,k)} + S^{-1}_{\boldsymbol{\theta}_i^{(t,k)}} U_{\boldsymbol{\theta}_i^{(t,k)}}, \quad k = 1, \ldots, E, \tag{3.1.10}$$

This inner loop of updates makes good use of the already loaded data to refine the estimate

of $\boldsymbol{\theta}_i^{(t)}$ so that the end estimate is closer to its MLE when the same $\widetilde{\boldsymbol{\theta}}$ values are used. Note that the updated $\boldsymbol{\theta}_i$ in each epoch from (3.1.10) will lead to changes in the score equations and Fisher information matrix, whose update will in turn lead to better estimate of $\boldsymbol{\theta}_i$. These many rounds of updates will reduce the variation of update in $\boldsymbol{\theta}_i^{(t)}$ when we go through many iterations based on equation (3.1.9). Below are formulae involved in the updating equations:

$$\boldsymbol{\theta}_i^{(t)} = \left(\mathbf{w}_i^{\top(t)}, b_i^{(t)}, e_i^{(t)}\right)^\top; \qquad U_{\boldsymbol{\theta}_i^{(t)}} = \left[\left(\frac{\partial \ell_i}{\partial \mathbf{w}_i}\right)^\top, \frac{\partial \ell_i}{\partial b_i}, \frac{\partial \ell_i}{\partial e_i}\right]^\top; \qquad S_{\boldsymbol{\theta}_i^{(t)}} = \begin{bmatrix} S_{\mathbf{w}_i}^{(t)} & S_{\mathbf{w}_i b_i}^{(t)} & S_{\mathbf{w}_i e_i}^{(t)} \\ S_{b_i \mathbf{w}_i}^{(t)} & S_{b_i}^{(t)} & S_{b_i e_i}^{(t)} \\ S_{e_i \mathbf{w}_i}^{(t)} & S_{e_i b_i}^{(t)} & S_{e_i}^{(t)} \end{bmatrix}$$

with

$$\frac{\partial \ell_i}{\partial \mathbf{w}_i} = \frac{\partial \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i} + \frac{\partial \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i},$$
$$\frac{\partial \ell_i}{\partial b_i} = \frac{\partial \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial b_i}, \qquad \frac{\partial \ell_i}{\partial e_i} = \frac{\partial \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial e_i},$$

where these partial derivatives are the $\frac{\partial \ell_i^{(1)}}{\partial \mathbf{w}_i}$, $\frac{\partial \ell_i^{(2)}}{\partial \mathbf{w}_i}$, $\frac{\partial \ell_i^{(1)}}{\partial b_i}$, and $\frac{\partial \ell_i^{(2)}}{\partial e_i}$ evaluated at $\boldsymbol{\theta}_i^{(t)}$ and $\widetilde{\boldsymbol{\theta}}^{(t)}$ using formulae in (3.1.1), (3.1.5), and

$$S_{\mathbf{w}_i}^{(t)} = E\left[-\frac{\partial^2 \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\right] + E\left[-\frac{\partial^2 \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\middle| g_{ij} = 1\right];$$
$$S_{\mathbf{w}_i b_i}^{(t)} = E\left[-\frac{\partial^2 \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial b_i}\right]; \qquad S_{b_i}^{(t)} = E\left[-\frac{\partial^2 \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial b_i^2}\right]; \qquad S_{b_i e_i}^{(t)} = 0;$$
$$S_{\mathbf{w}_i e_i}^{(t)} = E\left[-\frac{\partial^2 \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial e_i}\middle| g_{ij} = 1\right]; \qquad S_{e_i}^{(t)} = E\left[-\frac{\partial^2 \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial e_i^2}\middle| g_{ij} = 1;\right],$$

where the expectations are based on formulae (3.1.2), (3.1.3), (3.1.6), (3.1.7), (3.1.8) evaluated at $\boldsymbol{\theta}_i^{(t)}$ and $\widetilde{\boldsymbol{\theta}}^{(t)}$ if it is in outer loop of update and using the values of $\boldsymbol{\theta}^{(t,k)}$ and $\widetilde{\boldsymbol{\theta}}^{(t,k)}$ when it is in the inner loop of updates.

This concludes the first part of the alternating ZIG regression. It gives iterative update

of $\boldsymbol{\theta}$ while $\widetilde{\boldsymbol{\theta}}$ fixed. After we finish update of all $\boldsymbol{\theta}_i$'s, we move on to treat the $\boldsymbol{\theta}$ as fixed to estimate $\widetilde{\boldsymbol{\theta}}$.

Starting with $\widetilde{\ell}_j = \widetilde{\ell}_j^{(1)} + \widetilde{\ell}_j^{(2)}$, we consider the partial derivatives with respect to $\widetilde{\boldsymbol{\theta}}_j$ while holding $\boldsymbol{\theta}$ fixed. The derivations are similar to those for ℓ_i but still we list them for clarity. Note that

$$\begin{aligned}
\frac{\partial p_{ij}}{\partial \widetilde{\boldsymbol{w}}_j} &= \frac{\exp(\eta_{ij}) \cdot \mathbf{w}_i \{1 + \exp(\eta_{ij})\} - \exp(2\eta_{ij}) \cdot \mathbf{w}_i}{\{1 + \exp(\eta_{ij})\}^2} = p_{ij}(1 - p_{ij})\mathbf{w}_i, \\
\frac{\partial p_{ij}}{\partial \widetilde{b}_j} &= p_{ij}(1 - p_{ij}) \cdot 1.
\end{aligned}$$

The first order partial derivatives regarding $\widetilde{\ell}_j^{(1)}$ are

$$\begin{aligned}
\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j} &= \sum_{i=1}^{n} (g_{ij} - p_{ij}) \cdot \mathbf{w}_i = \sum_{i=1}^{n} (g_{ij} - p_{ij}) \cdot \mathbf{w}_i, \\
\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j} &= \sum_{i=1}^{n} (g_{ij} - p_{ij}) = \sum_{i=1}^{n} (g_{ij} - p_{ij}).
\end{aligned}$$

The second derivatives and their expectations are

$$\begin{aligned}
&\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top} = \sum_{i=1}^{n} \mathbf{w}_i \cdot \left(-\frac{\partial p_{ij}}{\partial \widetilde{\boldsymbol{w}}_j^\top} \right) = -\sum_{i=1}^{n} \mathbf{w}_i \mathbf{w}_i^\top p_{ij}(1 - p_{ij}), \\
\Longrightarrow \quad & E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top} \right] = \sum_{i=1}^{n} \mathbf{w}_i \mathbf{w}_i^\top p_{ij}(1 - p_{ij}); \\
\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{b}_j} = -\sum_{i=1}^{n} \mathbf{w}_i p_{ij}(1 - p_{ij}) \quad \Longrightarrow \quad & E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{b}_j^2} \right] = \sum_{i=1}^{n} \mathbf{w}_i p_{ij}(1 - p_{ij}); \\
\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j^2} = -\sum_{i=1}^{n} p_{ij}(1 - p_{ij}) \quad \Longrightarrow \quad & E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j^2} \right] = \sum_{i=1}^{n} p_{ij}(1 - p_{ij}).
\end{aligned}$$

Now, consider the first order partial derivatives of the second component $\widetilde{\ell}_j^{(2)}$.

$$\begin{aligned}
\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j} &= \sum_{i=1}^{n} g_{ij} \cdot \left\{ \frac{-v_{ij} \cdot \mathbf{w}_i}{-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j} + v_{ij} y_{ij} \cdot \mathbf{w}_i \right\}, \\
\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{e}_j} &= \sum_{i=1}^{n} g_{ij} \left\{ \frac{-v_{ij}}{-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j} + v_{ij} \cdot y_{ij} \right\}.
\end{aligned}$$

The second order partial derivatives and their negative expectations are

$$\begin{aligned}
& \frac{\partial^2 \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top} = \sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot \mathbf{w}_i \cdot \left(-\mathbf{w}_i^\top\right) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j\right)^2, \\
\Longrightarrow \quad & E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top} \middle| g_{ij} = 1 \right] = v_{ij} \sum_{\substack{i=1 \\ g_{ij}=1}}^{n} \mathbf{w}_i \cdot \mathbf{w}_i^\top / \left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j\right)^2; \\
& \frac{\partial^2 \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{e}_j} = \sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot \mathbf{w}_i \cdot (-1) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j\right)^2, \\
\Longrightarrow \quad & E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{e}_j} \middle| g_{ij} = 1 \right] = v_{ij} \sum_{\substack{i=1 \\ g_{ij}=1}}^{n} \mathbf{w}_i / \left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j\right)^2; \\
& \frac{\partial^2 \widetilde{\ell}_j^{(2)}}{\partial \widetilde{e}_j^2} = \sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot (-1) / \left(-\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j - e_i - \widetilde{e}_j\right)^2, \\
\Longrightarrow \quad & E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}}{\partial \widetilde{e}_j^2} \middle| g_{ij} = 1 \right] = v_{ij} \sum_{\substack{i=1 \\ g_{ij}=1}}^{n} 1 / \left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j\right)^2.
\end{aligned}$$

Therefore, the other side of updating equation for the alternating ZIG regression based on the Fisher scoring algorithm is

$$\begin{aligned}
\widetilde{\boldsymbol{\theta}}_j^{(t+1)} &= \widetilde{\boldsymbol{\theta}}_j^{(t)} + S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}, \quad j = 1, \ldots, n, \\
\widetilde{\boldsymbol{\theta}}_j^{(t,k+1)} &= \widetilde{\boldsymbol{\theta}}_j^{(t,k)} + S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t,k)}} U_{\widetilde{\boldsymbol{\theta}}_j^{(t,k)}}, \quad k = 1, \ldots, E.
\end{aligned}$$

Again, for each j and iteration number t, this equation is iterated for certain number of epochs to get refined estimate of $\widetilde{\boldsymbol{\theta}}_j$ based on current value of $\boldsymbol{\theta}$ without having to reload

the data. The quantities in the updating equation are listed below.

$$\widetilde{\boldsymbol{\theta}}_j^{(t)} = \left(\widetilde{\boldsymbol{w}}_j^{\top(t)}, \widetilde{b}_j^{(t)}, \widetilde{e}_j^{(t)}\right)^\top; \quad U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} = \begin{bmatrix} \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{\boldsymbol{w}}_j} \\ \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{b}_j}, \\ \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{e}_j} \end{bmatrix}; \qquad S_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} = \begin{bmatrix} S^{(t)}_{\widetilde{\boldsymbol{w}}_j} & S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} & S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{e}_j} \\ S^{(t)}_{\widetilde{b}_j \widetilde{\boldsymbol{w}}_j} & S^{(t)}_{\widetilde{b}_j} & S^{(t)}_{\widetilde{b}_j \widetilde{e}_j} \\ S^{(t)}_{\widetilde{e}_j \widetilde{\boldsymbol{w}}_j} & S^{(t)}_{\widetilde{e}_j \widetilde{b}_j} & S^{(t)}_{\widetilde{e}_j} \end{bmatrix}$$

with

$$\frac{\partial \widetilde{\ell}_j}{\partial \widetilde{\boldsymbol{w}}_j} = \frac{\partial \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j} + \frac{\partial \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j},$$
$$\frac{\partial \widetilde{\ell}_j}{\partial \widetilde{b}_j} = \frac{\partial \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{b}_j}, \qquad \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{e}_j} = \frac{\partial \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{e}_j},$$

where these partial derivatives are the $\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j}$, $\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j}$, $\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j}$, and $\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{e}_j}$ evaluated at $\widetilde{\boldsymbol{\theta}}_j^{(t)}$ and $\boldsymbol{\theta}^{(t)}$, and

$$S^{(t)}_{\widetilde{\boldsymbol{w}}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top}\right] + E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top}\middle| g_{ij} = 1\right];$$
$$S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{b}_j}\right]; \quad S^{(t)}_{\widetilde{b}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)},)}{\partial \widetilde{b}_j^2}\right]; \quad S^{(t)}_{\widetilde{b}_j \widetilde{e}_j} = 0;$$
$$S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{e}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{e}_j}\middle| g_{ij} = 1\right]; \quad S^{(t)}_{\widetilde{e}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{e}_j^2}\middle| g_{ij} = 1\right].$$

The expectations involved in above formulae were using the current values $\boldsymbol{\theta}^{(t)}$ and $\widetilde{\boldsymbol{\theta}}^{(t)}$ at t^{th} iteration if it is in outer loop of update and using the values of $\boldsymbol{\theta}^{(t,k)}$ and $\widetilde{\boldsymbol{\theta}}^{(t,k)}$ when it is in the inner loop of updates.

All of the formulae above assume the parameter v_{ij} is known. When it is not known, the parameter can be estimated with MLE, bias corrected MLE, or moment estimator. Simulation studies in Du (2007) suggested that when the sample size is large, all the estimators perform similarly but the moment estimator has advantage of easy to compute. If the sample

size is medium, then bias corrected MLE is better.

The problem we encountered is the canonical link for Gamma regression part. Recall the canonical link for the Gamma regression is $g(\mu_{ij}) = -\mu_{ij}^{-1} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j$. The left hand side of the equation is required to be negative because the support of the Gamma distribution is $(0, \infty)$. However, the right hand side of the equation could freely take any value in $(-\infty, \infty)$. Due to this conflict in natural parameter space and the range of estimated value, the likelihood function sometimes could become NaN because $\log(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j)$ appears in the likelihood function (see equation (3.1.4)).

3.2 Using log link for Gamma regression

As we discussed the limitation of the canonical link for Gamma regression part in Section 3.1, we utilize the other link, i.e., log link function, to model the Gamma regression part while maintaining logistic regression part. The settings of the two-parts model are similar to those in Section 3.1 except modifying the link function from canonical to log link.

For our updating formulae on the model parameters which is based on Fisher scoring algorithm, consider the derivatives of first part of the i^{th} log likelihood $\ell_i^{(1)}$. These can be derived similarly as in section 3.1. We will work on each of the partial derivatives in the following formulae one by one.

$$\begin{aligned}
&\frac{\partial \ell}{\partial \mathbf{w}_i} = \frac{\partial \ell_i}{\partial \mathbf{w}_i}; \quad \frac{\partial \ell}{\partial b_i} = \frac{\partial \ell_i^{(1)}}{\partial b_i}; \quad \frac{\partial \ell}{\partial e_i} = \frac{\partial \ell_i^{(2)}}{\partial e_i}; \\
&\frac{\partial^2 \ell}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} = \frac{\partial^2 \ell_i}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}; \quad \frac{\partial^2 \ell}{\partial \mathbf{w}_i \partial b_i} = \frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial b_i}; \quad \frac{\partial^2 \ell}{\partial \mathbf{w}_i \partial e_i} = \frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial e_i}; \\
&\frac{\partial^2 \ell}{\partial b_i^2} = \frac{\partial^2 \ell_i^{(1)}}{\partial b_i^2}; \quad \frac{\partial^2 \ell}{\partial b_i \partial e_i} = \frac{\partial^2 \ell_i}{\partial b_i \partial e_i} = 0; \quad \frac{\partial^2 \ell}{\partial e_i^2} = \frac{\partial^2 \ell_i^{(2)}}{\partial e_i^2}.
\end{aligned}$$

Recall

$$\ell_i^{(1)} = \sum_{j=1}^{n} (1 - g_{ij}) \cdot \log(1 - p_{ij}) + g_{ij} \cdot \log p_{ij} = \sum_{j=1}^{n} g_{ij} \cdot \eta_{ij} - \log\{1 + \exp(\eta_{ij})\}. \quad (3.2.1)$$

Then the first order partial derivatives of $\ell_i^{(1)}$ are

$$\frac{\partial \ell_i^{(1)}}{\partial \mathbf{w}_i} = \sum_{j=1}^{n} \left(g_{ij} - p_{ij}\right) \widetilde{\boldsymbol{w}}_j, \qquad \frac{\partial \ell_i^{(1)}}{\partial b_i} = \sum_{j=1}^{n} \left(g_{ij} - p_{ij}\right), \tag{3.2.2}$$

and the second order partial derivatives of $\ell_i^{(1)}$ are written as

$$\begin{aligned}
\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} &= -\sum_{j=1}^{n} p_{ij}\left(1 - p_{ij}\right) \widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top, \qquad \Rightarrow E\left[-\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\right] = \sum_{j=1}^{n} p_{ij}\left(1 - p_{ij}\right) \cdot \widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top, \\
\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial b_i} &= -\sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j p_{ij}\left(1 - p_{ij}\right), \qquad \Rightarrow E\left[-\frac{\partial^2 \ell_i^{(1)}}{\partial \mathbf{w}_i \partial b_i}\right] = \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j p_{ij}\left(1 - p_{ij}\right), \\
\frac{\partial^2 \ell_i^{(1)}}{\partial b_i^2} &= -\sum_{j=1}^{n} p_{ij}\left(1 - p_{ij}\right), \qquad \Rightarrow E\left[-\frac{\partial^2 \ell_i^{(1)}}{\partial b_i^2}\right] = \sum_{j=1}^{n} p_{ij}\left(1 - p_{ij}\right).
\end{aligned} \tag{3.2.3}$$

Now, consider the second component of i^{th} log likelihood $\ell_i^{(2)}$ and same notation is used for $\tau_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}} + e_i + \widetilde{e}_j$ as in previous subsection. Then the log likelihood of observations, which account for non-zero entries in cooccurrence matrix, corresponding to gamma distribution can be expressed as

$$\begin{aligned}
\ell_i^{(2)} &= \sum_{j=1}^{n} I\left(y_{ij} > 0\right) \cdot \log f\left(y_{ij} \mid y_{ij} > 0\right) = \sum_{j=1}^{n} g_{ij} \cdot \log f\left(y_{ij} \mid y_{ij} > 0\right) \\
&= \sum_{j=1}^{n} g_{ij} \Bigg\{ -\log \Gamma(v_{ij}) + v_{ij} \log v_{ij} + (v_{ij} - 1) \log y_{ij} - v_{ij} \tau_{ij} - v_{ij} y_{ij} \frac{1}{\exp\left(\tau_{ij}\right)} \Bigg\}
\end{aligned} \tag{3.2.4}$$

where

$$\log f\left(y_{ij} \mid y_{ij} > 0\right) = -\log \Gamma\left(v_{ij}\right) + v_{ij} \log v_{ij} + \left(v_{ij} - 1\right) \log y_{ij} - v_{ij} \cdot \tau_{ij} - v_{ij} y_{ij} \frac{1}{\exp\left(\tau_{ij}\right)}.$$

The equations below give the first order partial derivatives of $\ell_i^{(2)}$ with respect to the param-

eters $\mathbf{w}_i$ and e_i.

$$\begin{aligned}
\frac{\partial \ell_i^{(2)}}{\partial \mathbf{w}_i} &= \sum_{j=1}^{n} -g_{ij} \cdot v_{ij} \left\{ \frac{\partial \tau_{ij}}{\partial \mathbf{w}_i} + \frac{y_{ij}}{\mu_{ij}} \cdot \left(-\frac{\partial \tau_{ij}}{\partial \mathbf{w}_i} \right) \right\} = \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \mu_{ij}^{-1} \left(y_{ij} - \mu_{ij} \right) \cdot \widetilde{\boldsymbol{w}}_j, \\
\frac{\partial \ell_i^{(2)}}{\partial e_i} &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \mu_{ij}^{-1} \left(y_{ij} - \mu_{ij} \right) \cdot 1.
\end{aligned} \tag{3.2.5}$$

To get the second order partial derivatives, first note that

$$\mu_{ij} = \exp\left(\tau_{ij}\right) \quad \Rightarrow \frac{\partial \mu_{ij}}{\partial \mathbf{w}_i} = \frac{\partial \mu_{ij}}{\partial \tau_{ij}} \cdot \frac{\partial \tau_{ij}}{\partial \mathbf{w}_i} = \mu_{ij} \cdot \widetilde{\mathbf{w}}_j, \quad \text{and} \quad \frac{\partial \mu_{ij}}{\partial e_i} = \frac{\partial \mu_{ij}}{\partial \tau_{ij}} \cdot \frac{\partial \tau_{ij}}{\partial e_i} = \mu_{ij} \cdot 1.$$

Then the second order partial derivatives and their negative expectations which are components of the Fisher information matrix can be derived as

$$\begin{aligned}
\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \left\{ -\mu_{ij}^{-2} \cdot \mu_{ij} \cdot \widetilde{\boldsymbol{w}}_j \left(y_{ij} - \mu_{ij} \right) - \widetilde{\boldsymbol{w}}_j \right\} \widetilde{\boldsymbol{w}}_j^\top \\
&= -\sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \left\{ \frac{y_{ij} - \mu_{ij}}{\mu_{ij}} + 1 \right\} \widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top, \\
\Longrightarrow E \left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top} \middle| g_{ij} = 1 \right] &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top; \\
\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial e_i} &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \left\{ -\mu_{ij}^{-2} \cdot \mu_{ij} \cdot 1 \left(y_{ij} - \mu_{ij} \right) - \mu_{ij}^{-1} \left(\mu_{ij} \cdot 1 \right) \right\} \widetilde{\boldsymbol{w}}_j \\
&= -\sum_{j=1}^{n} g_{ij} \cdot v_{ij} \left\{ \frac{y_{ij} - \mu_{ij}}{\mu_{ij}} + 1 \right\} \widetilde{\boldsymbol{w}}_j, \\
\Longrightarrow E \left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \mathbf{w}_i \partial e_i} \middle| g_{ij} = 1 \right] &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot \widetilde{\boldsymbol{w}}_j; \\
\frac{\partial^2 \ell_i^{(2)}}{\partial e_i^2} &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \left\{ -\mu_{ij}^{-2} \cdot \mu_{ij} \left(y_{ij} - \mu_{ij} \right) - \mu_{ij}^{-1} \cdot \mu_{ij} \cdot 1 \right\} \cdot 1 \\
&= -\sum_{j=1}^{n} g_{ij} \cdot v_{ij} \left\{ \frac{y_{ij} - \mu_{ij}}{\mu_{ij}} + 1 \right\} \cdot 1, \\
\Longrightarrow E \left[-\frac{\partial^2 \ell_i^{(2)}}{\partial e_i^2} \middle| g_{ij} = 1 \right] &= \sum_{j=1}^{n} g_{ij} \cdot v_{ij} \cdot 1.
\end{aligned} \tag{3.2.6}$$

The first part of the alternating regression has the following updating equation based on the Fisher scoring algorithm:

$$\begin{aligned}\boldsymbol{\theta}_i^{(t+1)} &= \boldsymbol{\theta}_i^{(t)} + S^{-1}_{\boldsymbol{\theta}_i^{(t)}} U_{\boldsymbol{\theta}_i^{(t)}}, \quad i = 1, \ldots, n, \\ \boldsymbol{\theta}_i^{(t,k+1)} &= \boldsymbol{\theta}_i^{(t,k)} + S^{-1}_{\boldsymbol{\theta}_i^{(t,k)}} U_{\boldsymbol{\theta}_i^{(t,k)}}, \quad k = 1, \ldots, E. \end{aligned} \tag{3.2.7}$$

This updating equation in (3.2.7) is iterated E times, say E equals 20, for the same i and t where the t is the iteration number. That is, for each iteration of the update, the estimation of $\boldsymbol{\theta}_i$ is iteratively updated for 20 epochs. The multiple epochs here allow the estimate of $\boldsymbol{\theta}_i$ to get closer to its MLE for given current values of $\widetilde{\boldsymbol{\theta}}$ as the score equations and information matrix were also updated with the estimated parameter value. There is no need to have too many epochs because the parameter $\widetilde{\boldsymbol{\theta}}$ is not the true value yet and still need to be estimated later. Below are formulae involved in the updating equations:

$$\boldsymbol{\theta}_i^{(t)} = \left(\mathbf{w}_i^{\top(t)}, b_i^{(t)}, e_i^{(t)}\right)^\top; \qquad U_{\boldsymbol{\theta}_i^{(t)}} = \left[\left(\frac{\partial \ell_i}{\partial \mathbf{w}_i}\right)^\top, \frac{\partial \ell_i}{\partial b_i}, \frac{\partial \ell_i}{\partial e_i}\right]^\top; \qquad S_{\boldsymbol{\theta}_i^{(t)}} = \begin{bmatrix} S^{(t)}_{\mathbf{w}_i} & S^{(t)}_{\mathbf{w}_i b_i} & S^{(t)}_{\mathbf{w}_i e_i} \\ S^{(t)}_{b_i \mathbf{w}_i} & S^{(t)}_{b_i} & S^{(t)}_{b_i e_i} \\ S^{(t)}_{e_i \mathbf{w}_i} & S^{(t)}_{e_i b_i} & S^{(t)}_{e_i} \end{bmatrix}$$

with

$$\begin{aligned} \frac{\partial \ell_i}{\partial \mathbf{w}_i} &= \frac{\partial \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i} + \frac{\partial \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i}, \\ \frac{\partial \ell_i}{\partial b_i} &= \frac{\partial \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial b_i}, \qquad \frac{\partial \ell_i}{\partial e_i} = \frac{\partial \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)}; \widetilde{\boldsymbol{\theta}}^{(t)})}{\partial e_i}, \end{aligned}$$

where these partial derivatives are the $\frac{\partial \ell_i^{(1)}}{\partial \mathbf{w}_i}$, $\frac{\partial \ell_i^{(2)}}{\partial \mathbf{w}_i}$, $\frac{\partial \ell_i^{(1)}}{\partial b_i}$, and $\frac{\partial \ell_i^{(2)}}{\partial e_i}$ evaluated at $\boldsymbol{\theta}_i^{(t)}$ and $\widetilde{\boldsymbol{\theta}}^{(t)}$

using equations (3.2.2), (3.2.5), and

$$S^{(t)}_{\mathbf{w}_i} = E\left[-\frac{\partial^2 \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)};\widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\right] + E\left[-\frac{\partial^2 \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)};\widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\middle| g_{ij}=1\right];$$

$$S^{(t)}_{\mathbf{w}_i b_i} = E\left[-\frac{\partial^2 \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)};\widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial b_i}\right]; \qquad S^{(t)}_{b_i} = E\left[-\frac{\partial^2 \ell_i^{(1)}(\boldsymbol{\theta}_i^{(t)};\widetilde{\boldsymbol{\theta}}^{(t)})}{\partial b_i^2}\right]; \qquad S^{(t)}_{b_i e_i} = 0;$$

$$S^{(t)}_{\mathbf{w}_i e_i} = E\left[-\frac{\partial^2 \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)};\widetilde{\boldsymbol{\theta}}^{(t)})}{\partial \mathbf{w}_i \partial e_i}\middle| g_{ij}=1\right]; \qquad S^{(t)}_{e_i} = E\left[-\frac{\partial^2 \ell_i^{(2)}(\boldsymbol{\theta}_i^{(t)};\widetilde{\boldsymbol{\theta}}^{(t)})}{\partial e_i^2}\middle| g_{ij}=1;\right].$$

The expectations involved in above formulae were given in (3.2.3), (3.2.6) except that the parameters are using the current values $\boldsymbol{\theta}^{(t)}$ and $\widetilde{\boldsymbol{\theta}}^{(t)}$ at t^{th} iteration if it is in outer loop of update and using the values of $\boldsymbol{\theta}^{(t,k)}$ and $\widetilde{\boldsymbol{\theta}}^{(t,k)}$ when it is in the inner loop of updates.

The aforementioned equations conclude one side of the alternating ZIG regression. Now, consider the other side of the alternating ZIG regression in which updates are done for $\widetilde{\boldsymbol{\theta}}$ while holding $\boldsymbol{\theta}$ fixed. First, consider the first order partial derivatives

$$\frac{\partial \widetilde{\ell}}{\partial \widetilde{\boldsymbol{w}}_j} = \sum_i \left\{\frac{\partial \widetilde{\ell}_{ij}^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j} + \frac{\partial \widetilde{\ell}_{ij}^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j}\right\} = \frac{\partial \sum_{i=1}^n \widetilde{\ell}_{ij}^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j} + \frac{\partial \sum_{i=1}^n \widetilde{\ell}_{ij}^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j} = \frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j} + \frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j}.$$

The first order partial derivatives of $\widetilde{\ell}_j^{(1)}$ are

$$\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j} = \sum_{i=1}^n (g_{ij} - p_{ij}) \cdot \mathbf{w}_i, \quad \frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j} = \sum_{i=1}^n (g_{ij} - p_{ij}), \quad \frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{e}_j} = 0.$$

The second order partial derivatives and their expectations are

$$\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top} = \sum_{i=1}^n \mathbf{w}_i \cdot \left(-\frac{\partial p_{ij}}{\partial \widetilde{\boldsymbol{w}}_j^\top}\right) = -\sum_{i=1}^n \mathbf{w}_i \mathbf{w}_i^\top p_{ij}(1-p_{ij}),$$

$$\implies \quad E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top}\right] = \sum_{i=1}^n \mathbf{w}_i \mathbf{w}_i^\top p_{ij}(1-p_{ij});$$

$$\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j \partial \widetilde{e}_j} = 0; \quad \frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{e}_j^2} = 0; \quad \implies \quad E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j \partial \widetilde{e}_j}\right] = 0; \quad E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{e}_j^2}\right] = 0;$$

$$\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{b}_j} = -\sum_{i=1}^{n} \mathbf{w}_i p_{ij}\left(1-p_{ij}\right) \quad \Longrightarrow \quad E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j \partial b_j^2}\right] = \sum_{i=1}^{n} \mathbf{w}_i p_{ij}\left(1-p_{ij}\right);$$
$$\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j^2} = -\sum_{i=1}^{n} p_{ij}\left(1-p_{ij}\right) \quad \Longrightarrow \quad E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j^2}\right] = \sum_{i=1}^{n} p_{ij}\left(1-p_{ij}\right).$$

Next, consider the first order partial derivatives of the second component $\widetilde{\ell}_j^{(2)}$

$$\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j} = \sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot \mu_{ij}^{-1}\left(y_{ij}-\mu_{ij}\right) \cdot \mathbf{w}_i, \quad \frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{e}_j} = \sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot \mu_{ij}^{-1}\left(y_{ij}-\mu_{ij}\right) \cdot 1.$$

To derive the second order partial derivatives, note that frequently used two terms are

$$\mu_{ij} = \exp\left(\tau_{ij}\right) \quad \Rightarrow \frac{\partial \mu_{ij}}{\partial \widetilde{\boldsymbol{w}}_j} = \frac{\partial \mu_{ij}}{\partial \tau_{ij}} \cdot \frac{\partial \tau_{ij}}{\partial \widetilde{\boldsymbol{w}}_j} = \mu_{ij} \cdot \mathbf{w}_i \quad \text{and} \quad \frac{\partial \mu_{ij}}{\partial \widetilde{e}_j} = \frac{\partial \mu_{ij}}{\partial \tau_{ij}} \cdot \frac{\partial \tau_{ij}}{\partial \widetilde{e}_j} = \mu_{ij} \cdot 1.$$

Using the two terms, the second derivatives are shown as

$$\frac{\partial^2 \ell_i^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^{\top}} = -\sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot \left\{\frac{y_{ij}-\mu_{ij}}{\mu_{ij}} + 1\right\} \mathbf{w}_i \mathbf{w}_i^{\top},$$
$$\Rightarrow E\left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^{\top}} \middle| g_{ij}=1\right] = -\sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot \mathbf{w}_i \cdot \mathbf{w}_i^{\top};$$
$$\frac{\partial^2 \ell_i^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{e}_j} = -\sum_{i=1}^{n} g_{ij} v_{ij} \left\{\frac{y_{ij}-\mu_{ij}}{\mu_{ij}} + 1\right\} \mathbf{w}_i \quad \Rightarrow \quad E\left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{e}_j} \middle| g_{ij}=1\right] = \sum_{i=1}^{n} g_{ij} v_{ij} \mathbf{w}_i;$$
$$\frac{\partial^2 \ell_i^{(2)}}{\partial \widetilde{e}_j^2} = -\sum_{i=1}^{n} g_{ij} \cdot v_{ij} \left\{\frac{y_{ij}-\mu_{ij}}{\mu_{ij}} + 1\right\} \cdot 1 \quad \Rightarrow \quad E\left[-\frac{\partial^2 \ell_i^{(2)}}{\partial \widetilde{e}_j^2} \middle| g_{ij}=1\right] = \sum_{i=1}^{n} g_{ij} \cdot v_{ij} \cdot 1.$$

Therefore, the other side of updating equation for the alternating ZIG regression based on the Fisher scoring algorithm is

$$\widetilde{\boldsymbol{\theta}}_j^{(t+1)} = \widetilde{\boldsymbol{\theta}}_j^{(t)} + S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}, \quad j = 1, \ldots, n, \tag{3.2.8}$$
$$\widetilde{\boldsymbol{\theta}}_j^{(t,k+1)} = \widetilde{\boldsymbol{\theta}}_j^{(t,k)} + S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t,k)}} U_{\widetilde{\boldsymbol{\theta}}_j^{(t,k)}}, \quad k = 1, \ldots, E. \tag{3.2.9}$$

The updating equation (3.2.8) is in the outer loop of iterations and the iterations in (3.2.9)

are in the inner loop of epochs for the same j and t. The quantities involved are

$$\widetilde{\boldsymbol{\theta}}_j^{(t)} = \left(\widetilde{\boldsymbol{w}}_j^{\top(t)}, \widetilde{b}_j^{(t)}, \widetilde{e}_j^{(t)}\right)^\top; \quad U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} = \begin{bmatrix} \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{\boldsymbol{w}}_j} \\ \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{b}_j}, \\ \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{e}_j} \end{bmatrix}; \quad S_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} = \begin{bmatrix} S^{(t)}_{\widetilde{\boldsymbol{w}}_j} & S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} & S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{e}_j} \\ S^{(t)}_{\widetilde{b}_j \widetilde{\boldsymbol{w}}_j} & S^{(t)}_{\widetilde{b}_j} & S^{(t)}_{\widetilde{b}_j \widetilde{e}_j} \\ S^{(t)}_{\widetilde{e}_j \widetilde{\boldsymbol{w}}_j} & S^{(t)}_{\widetilde{e}_j \widetilde{b}_j} & S^{(t)}_{\widetilde{e}_j} \end{bmatrix}$$

with

$$\frac{\partial \widetilde{\ell}_j}{\partial \widetilde{\boldsymbol{w}}_j} = \frac{\partial \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j} + \frac{\partial \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j},$$
$$\frac{\partial \widetilde{\ell}_j}{\partial \widetilde{b}_j} = \frac{\partial \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{b}_j}, \qquad \frac{\partial \widetilde{\ell}_j}{\partial \widetilde{e}_j} = \frac{\partial \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{e}_j},$$

where these partial derivatives are derivatives $\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{\boldsymbol{w}}_j}$, $\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{\boldsymbol{w}}_j}$, $\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_j}$, $\frac{\partial \widetilde{\ell}_j^{(2)}}{\partial \widetilde{e}_j}$ and

$$S^{(t)}_{\widetilde{\boldsymbol{w}}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top}\right] + E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top}\middle| g_{ij} = 1\right];$$
$$S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{b}_j}\right]; \quad S^{(t)}_{\widetilde{b}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)},)}{\partial \widetilde{b}_j^2}\right]; \quad S^{(t)}_{\widetilde{b}_j \widetilde{e}_j} = 0;$$
$$S^{(t)}_{\widetilde{\boldsymbol{w}}_j \widetilde{e}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{e}_j}\middle| g_{ij} = 1\right]; \quad S^{(t)}_{\widetilde{e}_j} = E\left[-\frac{\partial^2 \widetilde{\ell}_j^{(2)}(\widetilde{\boldsymbol{\theta}}_j^{(t)}; \boldsymbol{\theta}^{(t)})}{\partial \widetilde{e}_j^2}\middle| g_{ij} = 1\right],$$

evaluated at $\widetilde{\boldsymbol{\theta}}^{(t)}$ and $\boldsymbol{\theta}^{(t)}$ during outer loop of iterations and at $\widetilde{\boldsymbol{\theta}}^{(t,k)}$ and $\boldsymbol{\theta}^{(t,k)}$ during inner loop of iterations.

3.3 Convergence analysis

In this section, we discuss the convergence behavior of the algorithm analytically. Our algorithm contains two components: logistic regression part and gamma regression part. If these two components' parameter estimation were independent of each other, then it is the

situation of standard Generalized Linear Model (GLM) in each part. In our model, the two components' estimation can not be separated but the two components in log likelihood are additive. Hence, the convergence behavior in one component standard GLM case is still relevant. We will first talk about the convergence behavior in the standard case as this case applies to the situation when we hold the $\widetilde{\boldsymbol{\theta}}$ fixed while estimating $\boldsymbol{\theta}$ or vice versa.

In the standard GLM setting, most of the parameter estimation converges pretty fast. There are also abnormal behavior that could happen such as when estimated model component gets out of the valid range of the distribution. For example, with the gamma regression, if the canonical link (negative inverse) is used, the estimated mean may become negative every now and then even though the distribution requires positive mean. Haberman (1977), Silvapulle (1981), and Albert and Anderson (1984) presented conditions for the existence of MLE in logistic regression models. They proved that if there is overlap in the convex cones generated by the data points, the maximum likelihood estimates of the regression parameters exist and are unique. On the other hand, if the convex cones generated by data points in different groups have complete separation or quasi complete separation, then the maximum likelihood estimates do not exist. Generally, they recommended to insert a stopping rule if complete separation is found or restart the iterative algorithm with standardized observations (to have mean 0 and variance 1) if quasi complete separation is found. Albert and Anderson (1984) also recommended a procedure to check the conditions. For quasi complete separation, the estimation process diverges at least at some points. This makes the estimated probability of belonging to the correct group grows to one. Therefore, Albert and Anderson (1984) recommended to check the maximum predicted probability $p^{(t)}$ for each data point at t^{th} iteration. If the maximum probability is close to 1 and is bigger than previous iterations' maximum probability, there are two possibilites that this could happen. Albert and Anderson (1984) recommended to initially print a warning but continue the iteration because the data point is likely to be an outlier observation in its own group and there is overlap in the two convex cones. In this case, the MLE exists and is unique so the algorithm should be

allowed to continue. The other possibility is that there is quasi complete separation in the data. In this case, the process should be stopped and rerun with the observation vectors standardized with zero mean and unit variance.

Albert and Anderson (1984) also stated that the difficulties associated with complete and quasi complete separation are small sample problems. With large sample size, the probability of observing a set of separated data points is close to zero. Complete separation may occur with any type of data but it is unlikely that quasi complete separation will occur with truly continuous data.

Marschner (2011) illustrated the non-convergence problem with Poisson regression example. The author proposed a simple solution and implemented it in the R *glm*2 package. Specifically, if iteratively reweighted least squares (IRLS) produces either an infinite deviance or predicted values which fall within invalid range, then the amount of update of the parameter estimates is repeatedly halved until the update no longer shows the behavior. Moreover, it made a further step-halving, which checks the updated deviance is making a reduction compared to that in the previous iteration. If it did not show the reduction, it triggers the step-halving to make the algorithm monotonically reduces the deviance.

Based on the aforementioned studies illustrating the standard case of GLM convergence behavior, we will analytically present the convergence behavior of our alternating ZIG regression. To explain the convergence behavior of our algorithm, we cite the theorem from Silvapulle (1981) after introducing the following two convex cones.

In a binary regression problem, suppose $\mathbf{x}_1, \ldots, \mathbf{x}_r$ are covariate values of the observations in class 1, and $\mathbf{x}_{r+1}, \ldots, \mathbf{x}_n$ are covariate values of the observations in class 0. Let S, F be the relative interiors of the convex cones generated by $\mathbf{x}_1, \ldots, \mathbf{x}_r$ and $\mathbf{x}_{r+1}, \ldots, \mathbf{x}_n$ respectively. That is,

$$S = \left\{ \sum_{i}^{r} k_i \mathbf{x}_i \mid k_i > 0 \right\}, \quad F = \left\{ \sum_{r+1}^{n} k_i \mathbf{x}_i \mid k_i > 0 \right\}.$$

Below is the theorem from Silvapulle (1981). This theorem is for fitting a Logistic or Probit model for binomial data with inverse link function given by G and regression parameters

are given by $\boldsymbol{\beta}$ of dimension p.

Theorem 1. *(Theorem of Silvapulle 1981) Let the condition* Π *be defined by*

$$\Pi : S \cap F \neq \varnothing \text{ or one of } S, F \text{ is } R^p (\varnothing = \text{empty set }).$$

(i) The mle $\hat{\boldsymbol{\beta}}$ *of* $\boldsymbol{\beta}$ *exists and the minimum set* $\{\hat{\boldsymbol{\beta}}\}$ *is bounded only when* Π *is satisfied.*

(ii) Suppose that $l(\boldsymbol{\beta})$ *is a proper closed convex function on* R^p*. Then the mle* $\hat{\boldsymbol{\beta}}$ *exists and the minimum set* $\{\hat{\boldsymbol{\beta}}\}$ *is bounded if and only if* Π *is satisfied.*

(iii) Suppose that $-\log G$ *and* $-\log(1-G)$ *are convex and* $x_{i1} = 1$ *for every* i*. Then* $\hat{\boldsymbol{\beta}}$ *exists and the minimum set* $\{\hat{\boldsymbol{\beta}}\}$ *is bounded if and only if* $S \cap F \neq \varnothing$*. Let us further assume that* G *is strictly increasing at every* t *satisfying* $0 < G(t) < 1$*. Then* $\hat{\boldsymbol{\beta}}$ *is uniquely defined if and only if* $S \cap F \neq \varnothing$*.*

This theorem tells that there exists an unique mle $\hat{\boldsymbol{\beta}}$ for binomial Logistic or Probit model when the given data for covariates from different categories have overlap except in the trivial case that the entire R^p is equal to one of the two convex cones formed by the covariates. In the case there are separations between the two convex cones, the MLE does not exist for binomial regression or the solution set $\{\hat{\boldsymbol{\beta}}\}$ is unbounded. This latter case could be a problem in our alternating ZIG regression. Next, we will discuss how this affects the convergence behavior of our alternating ZIG regression.

For further discussion, we first specify our convex cones. Recall, $\widetilde{\boldsymbol{\theta}}$ serves as data when we estimate $\boldsymbol{\theta}_i$. In our context, the convex cones will be defined through $\widetilde{\boldsymbol{\theta}}$ while we estimate $\boldsymbol{\theta}_i$ and through $\boldsymbol{\theta}$ while we estimate $\widetilde{\boldsymbol{\theta}}_j$. That is, for estimating $\boldsymbol{\theta}_i$, the convex cones are

$$S_{\widetilde{\boldsymbol{\theta}}} = \left\{ \sum_{\substack{j=1 \\ y_{ij}>0}}^{n} k_j \widetilde{\boldsymbol{\theta}}_j \mid k_j > 0 \right\}, \quad F_{\widetilde{\boldsymbol{\theta}}} = \left\{ \sum_{\substack{j=1 \\ y_{ij}=0}}^{n} k_j \widetilde{\boldsymbol{\theta}}_j \mid k_j > 0 \right\}. \tag{3.3.1}$$

For estimating the $\widetilde{\boldsymbol{\theta}}_j$, the convex cones are denoted as $S_{\boldsymbol{\theta}}$ and $F_{\boldsymbol{\theta}}$, respectively.

Firstly, we consider the case that either complete separation or quasi complete separation exists in the data. We discuss the estimation of $\boldsymbol{\theta}_i$ while holding $\widetilde{\boldsymbol{\theta}}$ fixed.

According to the theorem, suppose the condition Π is not satisfied. Then there exists complete separation or quasi separation in $S_{\widetilde{\boldsymbol{\theta}}}$ and $F_{\widetilde{\boldsymbol{\theta}}}$. That is, $S_{\widetilde{\boldsymbol{\theta}}} \cap F_{\widetilde{\boldsymbol{\theta}}} = \varnothing$, $S_{\widetilde{\boldsymbol{\theta}}} \neq R^{d+1}$, and $F_{\widetilde{\boldsymbol{\theta}}} \neq R^{d+1}$. Also, suppose $S_{\widetilde{\boldsymbol{\theta}}} \neq \varnothing$ and $F_{\widetilde{\boldsymbol{\theta}}} \neq \varnothing$. Then there exists a vector direction $\mathbf{c}_j$ such that $\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \geq 0$ for $y_{ij} > 0$ and $\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \leq 0$ for $y_{ij} = 0$. Let $\ell_i^{(1)}(k)$ be the log likelihood of the Bernoulli part when the model parameters are updated toward direction $\mathbf{c}_j$ by k unit. That is, the original log likelihood $\ell_i^{(1)}$ and the updated $\ell_i^{(1)}(k)$ are as follows:

$$\begin{aligned}
\ell_i^{(1)} &= \sum_{j=1}^{n} \left\{ I\left(y_{ij} = 0\right) \cdot \log\left(1 - p_{ij}\right) + I\left(y_{ij} > 0\right) \cdot \log p_{ij} \right\} \\
&\text{where } p_{ij} = \frac{\exp\left(\eta_{ij}\right)}{1 + \exp\left(\eta_{ij}\right)}, \quad \eta_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j \\
\ell_i^{(1)}(k) &= \sum_{j=1}^{n} \left\{ I\left(y_{ij} = 0\right) \cdot \log\left(1 - p_{ij}(k)\right) + I\left(y_{ij} > 0\right) \cdot \log p_{ij}(k) \right\} \\
&\text{where } p_{ij}(k) = \frac{\exp\left(\eta_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j\right)}{1 + \exp\left(\eta_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j\right)}, \quad \eta_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j
\end{aligned} \tag{3.3.2}$$

- For $y_{ij} > 0, \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \geq 0$. Hence, as k increases, $p_{ij}(k)$ increases toward 1. This implies $\sum_{j=1}^{n} I\left(y_{ij} > 0\right) \cdot \log p_{ij}(k)$ increases toward 0.

- For $y_{ij} = 0$, $\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \leq 0$. Hence, as k increases, $p_{ij}(k)$ decreases toward 0. This implies $\sum_{j=1}^{n} I\left(y_{ij} = 0\right) \cdot \log\left(1 - p_{ij}(k)\right)$ increases toward 0.

Putting the two pieces together, we know that as k increases, $\ell_i^{(1)}(k)$ increases for any $\mathbf{w}_i, b_i$. Therefore, the maximum can not be reached until k is ∞. This means the MLE does not exist or the solution set $\hat{\boldsymbol{\theta}}$ is unbounded. This perspective can be also seen by looking partial derivative of $\ell_i^{(1)}(k)$ with respect to k. Note that

$$\frac{\partial \ell_i^{(1)}(k)}{\partial k} = \sum_{j=1}^{n} \left\{ I\left(y_{ij} = 0\right) \cdot \left(-p_{ij}(k)\right) \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right\} + \sum_{j=1}^{n} \left\{ I\left(y_{ij} > 0\right) \left(1 - p_{ij}(k)\right) \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right\} \tag{3.3.3}$$

Since the $\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \leq 0$ when $y_{ij} = 0$, we know the first term is non-negative. Similarly, $\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \geq 0$ when $y_{ij} > 0$ implies the second term is non-negative. Therefore, the partial derivative of $\ell_i^{(1)}(k)$ is non-negative. This indicates that the gradient of $\ell_i^{(1)}$ is positive unless $\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j = 0$. Therefore, there is no solution for $\ell_i^{(1)} = 0$ except the trivial solution $\mathbf{c}_j = 0$ in Binomial regression alone. Of course, this is not exactly our case because we still have the Gamma regression component to be considered together.

Now, consider the $\ell_i^{(2)}$ component with its first and second derivatives with respect to k.

$$\begin{aligned}
\ell_i^{(2)}(k) &= \sum_{j=1}^{n} g_{ij} \left\{ o_{ij} + v_{ij} \left[\log y_{ij} - \left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right) \right] - v_{ij} y_{ij} \exp\left(- \left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right) \right) \right\} \\
&= \sum_{j=1}^{n} g_{ij} \left\{ o_{ij} + v_{ij} \left[\log \frac{y_{ij}}{\exp\left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)} \right] - v_{ij} \frac{y_{ij}}{\exp\left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)} \right\},
\end{aligned}$$

where $o_{ij} = -\Gamma(v_{ij}) + v_{ij} \log v_{ij} - \log y_{ij}$.

$$\begin{aligned}
\frac{\partial \ell_i^{(2)}(k)}{\partial k} &= \sum_{j=1}^{n} g_{ij} v_{ij} \left\{ -\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j - y_{ij} \exp\left(- \left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right) \right) \cdot \left(-\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right) \right\} \\
&= \sum_{j=1}^{n} g_{ij} v_{ij} \left\{ \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \left[\frac{y_{ij}}{\exp\left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)} - 1 \right] \right\},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial^2 \ell_i^{(2)}(k)}{\partial k^2} &= \sum_{j=1}^{n} g_{ij} v_{ij} \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \left[y_{ij} \exp\left(-\tau_{ij} - k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right) \cdot \left(-\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right) \right] \\
&= -\sum_{j=1}^{n} g_{ij} v_{ij} \left(\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)^2 \cdot \frac{y_{ij}}{\exp\left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)} < 0.
\end{aligned}$$

The second derivative implies that $\ell_i^{(2)}(k)$ is concave. The term $\exp\left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)$ is the mean of y_{ij}. Hence, $\ell_i^{(2)}(k)$ is maximized when $y_{ij} = \exp\left(\tau_{ij} + k\widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \right)$. The first order partial derivative can be split into two summations: one is when the observed y_{ij} is less than

its mean, the other one is when the observed y_{ij} is greater than its mean, i.e.,

$$\frac{\partial \ell_i^{(2)}(k)}{\partial k} = \sum_{j=1}^{n} g_{ij} v_{ij} \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \left[\frac{y_{ij}}{\exp\left(\tau_{ij} + k \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j\right)} - 1 \right] I\left(y_{ij} > \exp\left(\tau_{ij} + k \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j\right)\right) \quad (3.3.4)$$

$$+ \sum_{j=1}^{n} g_{ij} v_{ij} \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j \left[\frac{y_{ij}}{\exp\left(\tau_{ij} + k \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j\right)} - 1 \right] I\left(y_{ij} < \exp\left(\tau_{ij} + k \widetilde{\boldsymbol{w}}_j^\top \mathbf{c}_j\right)\right). \quad (3.3.5)$$

The summation term in (3.3.4) is positive while that in (3.3.5) is negative. The partial derivative of ℓ_i is

$$\frac{\partial \ell_i(k)}{\partial k} = \frac{\partial \ell_i^{(1)}(k)}{\partial k} + \frac{\partial \ell_i^{(2)}(k)}{\partial k}$$

From the above discussion, $\frac{\partial \ell_i^{(1)}(k)}{\partial k}$ is greater than 0 and $\frac{\partial \ell_i^{(2)}(k)}{\partial k}$ has one term positive and the other term negative. Therefore, in order for the MLE to exist, the term in (3.3.5) must cancel the total value of $\frac{\partial \ell_i^{(1)}(k)}{\partial k}$ and the term in (3.3.4). This is possible if the shape parameter of Gamma is small enough so that there are more observations having values less than its mean. However, when the shape parameter is large the Gamma distribution is approximately Normal. In this case, the observations are symmetrically located on either side of its mean. This will make the number of observations belonging to the term (3.3.4) and (3.3.5) roughly equal. Consequently, there will not be extra values left to neutralize $\frac{\partial \ell_i^{(1)}(k)}{\partial k}$. As a result, for any fixed dispersion parameter ν, if the mean of the Gamma distribution is large and complete separation or quasi complete separation holds, it is highly likely that the MLE does not exist.

Next, consider the case when there is overlap in the two convex cones as $S_{\widetilde{\boldsymbol{\theta}}}$ and $F_{\widetilde{\boldsymbol{\theta}}}$ when we estimate the $\boldsymbol{\theta}_i$ (i.e., there is neither the complete separation nor the quasi complete separation in the data). Recall, the ZIG model is a two-parts model with Bernoulli part and Gamma part. The log likelihood function $\ell_i^{(1)}$ corresponding to the Bernoulli part (3.2.1) can be shown to be strictly concave in $\boldsymbol{\theta}_i = (\mathbf{w}_i^\top, b_i)^\top$. This is because the first component

$g_{ij}\eta_{ij}$ in $\ell_i^{(1)}$, where $\eta_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j$, is an affine function which is both convex and concave, if we hold $\widetilde{\boldsymbol{w}}_j$ and $\widetilde{b}_j$ fixed. The second component $-\log\{1+\exp(\eta_{ij})\}$ is strictly concave as its second derivative is less than 0 as shown below

$$\frac{\partial}{\partial x}[-\log\{1+\exp(x)\}] = -\frac{\exp(x)}{1+\exp(x)},$$
$$\frac{\partial^2}{\partial x^2}[-\log\{1+\exp(x)\}] = -\frac{\exp(x)}{[1+\exp(x)]^2} < 0.$$

Thus, the log likelihood corresponding to the Bernoulli part $\ell_i^{(1)}$ is strictly concave. Now, consider the log likelihood corresponding to the Gamma part $\ell_i^{(2)}$ in (3.2.4). We only need to consider the last two terms $-\nu_{ij}\tau_{ij}$ and $-\nu_{ij}y_{ij}\exp(-\tau_{ij})$ because the other terms do not involve the regression parameters, where $\tau_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j$. Note that $-\tau_{ij}$ is affine function of $\widetilde{\boldsymbol{\theta}}_j$, which is both convex and concave and $\exp(g(x))$ is convex if $g(x)$ is convex (see Boyd and Vandenberghe 2004). This leads to the term $-\nu_{ij}y_{ij}\exp(-\tau_{ij})$ being strictly concave in $\widetilde{\boldsymbol{\theta}}_j$. Hence, we have the $\ell_i^{(2)}$ strictly concave in $\widetilde{\boldsymbol{\theta}}_j = (\widetilde{\boldsymbol{w}}_j^\top, \widetilde{b}_j)^\top$. Combining the two concave components $\ell_i^{(1)}$ and $\ell_i^{(2)}$, we know that the entire log likelihood for the i^{th} row ℓ_i is a strictly concave function for any row i. Additionally, there is overlap in the two convex cones. Therefore, for any direction in the overlapping area of the convex cones, if we update the parameter along that direction will lead to the two components of $\frac{\partial \ell_i^{(1)}(k)}{\partial k}$ in formula (3.3.3) are of opposite sign of each other. In this case, $-\ell_i$ has an unique minimum when there is neither complete separation nor quasi complete separation in $S_{\widetilde{\boldsymbol{\theta}}}$ and $F_{\widetilde{\boldsymbol{\theta}}}$. Similar arguments apply when we estimate $\widetilde{\boldsymbol{\theta}}_j$ while $\boldsymbol{\theta}$ serves as data. That is, an unique MLE estimate of $\widetilde{\boldsymbol{\theta}}_j$ exists when there is neither complete separation nor quasi complete separation in $S_{\boldsymbol{\theta}}$ and $F_{\boldsymbol{\theta}}$. This means, the alternating procedure will find the MLE of $\widetilde{\boldsymbol{\theta}}_j$ when $\boldsymbol{\theta}$ is fixed and will find the MLE of $\boldsymbol{\theta}_i$ when $\widetilde{\boldsymbol{\theta}}$ is fixed in this non-separation scenario.

In summary, we conclude that our alternating ZIG regression has an unique MLE in each side of the regressions and the algorithm converges when there is overlap in data. The data refer to $\widetilde{\boldsymbol{\theta}}$ while we estimating $\boldsymbol{\theta}_i$ and refer to $\boldsymbol{\theta}$ while we estimating $\widetilde{\boldsymbol{\theta}}_j$. More

specifically, if there is overlap in data, both the $\ell_i^{(1)}$ and $\ell_i^{(2)}$ are concave functions and the two terms in the first order partial derivative in $\ell_i^{(1)}$ and $\ell_i^{(2)}$ are of opposite sign. The case for Gamma regression part will always have opposite signs which allow a maximum to exist. However, when there is complete separation or quasi complete separation, the alternating ZIG regression will fail to converge with high chance. This is because the first order partial derivative of $\ell_i^{(1)}$ is non-negative and $\ell_i^{(1)}$ increases with k. Even though the $\ell_i^{(2)}$ component is well-behaved concave curve, the entire log likelihood, unfortunately, may not have MLE exist or the solution set may be unbounded especially when the Gamma observations have large shape parameter.

3.4 Adjusting parameter update with learning rate

In the convergence analysis section, our discussion is based on holding one of the $\boldsymbol{\theta}$ or $\widetilde{\boldsymbol{\theta}}$ fixed while estimating the other one. The Fisher scoring algorithm is a modified version of Newton's method. In general, the Newton's method is an algorithm to solve $g'(x) = 0$. This algorithm starts with an initial value x_0 and compute a sequence of points via $x_{n+1} = x_n - g''(x_n)^{-1}g'(x_n)$. The Newton's method converges fast because the distance between the estimate and its true value shrinks quickly such that the distance in next step of iteration is asymptotically equivalent to the squared distance in previous iteration. That is, the Newton's method has quadratic convergence order. This convergence order holds when the initial value of the iteration is in the neighborhood of the true value and the third derivative $g'''(x)$ is continuous and $g''(x)$ is non-zero (cf. page 29 of Givens and Hoeting 2013; Osborne 1992).

The Fisher scoring algorithm is slightly different from the Newton's method. In the Fisher scoring algorithm, we replace the $-g''(x_n)$ by its expected value. This algorithm is asymptotically equivalent to the Newton's method and therefore enjoys the same asymptotic property such as consistency of the estimate of the parameter. As sample size gets large,

the convergence order increases (Osborne 1992). It has some advantages over the Newton's method in that the expected value of the Hessian matrix is positive definite which guarantees the update is uphill toward the direction of maximizing the log likelihood function assuming that the model is correct and the covariates are true explanatory variables.

As the Fisher scoring algorithm is assuming $\widetilde{\boldsymbol{\theta}}$ is the true value when we estimate $\boldsymbol{\theta}$ or vice versa, there could be complications when the parameter being fixed is not equal to the true parameter value. To see this point, note that our updating equations are all written in the context of using the Fisher scoring algorithm, which relies on the expectation of the Hessian matrix using correct distribution at the true parameter value. In particular, the algorithm uses $\widetilde{\boldsymbol{\theta}}$ as fixed value while estimating $\boldsymbol{\theta}$. The resulting estimate of $\boldsymbol{\theta}$ determines the distribution because the distribution is a function of $\boldsymbol{\theta}$ and $\widetilde{\boldsymbol{\theta}}$. When the parameter being fixed (such as $\widetilde{\boldsymbol{\theta}}$) is far from its true value, then the distribution is wrong even though it is in the right family. And the consequence of using a wrong distribution to compute the expectation of the Hessian matrix could lead to a sequence of parameter updates that converges to a limited value unequal to the true parameter (Osborne 1992). In this case, the algorithm might diverge.

To avoid this parameter update divergence problem, we introduce learning rate adjustment so that the change in parameter estimate is scaled by $lr/t^{1/4}$. This adjustment is applied to both inner loop iterations and outer loop iterations. See Algorithm 3 for the work flow.

Using the learning rate adjustment makes smaller steps in each update before changing directions. This learning rate adjustment turns out be crucial. The simulation study in next section will examine the effect of the learning rate adjustment.

Algorithm 1 Alternating regression

Require: $I^{-1}_{\boldsymbol{\theta}_i^{(t)}}$ and $I^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}$ exist.

$t \leftarrow 0$, $Loss^{(t)} \leftarrow 10^{20}$
$converged = False$
while $t \leq maxit$ **do**
  while $converged = False$ **do**
    $i \leftarrow 1$
    while $i \leq n$ **do**
      retrieve ith row of data from SQLite database.
      do n_epoch update of $U_{\boldsymbol{\theta}_i^{(t)}}$ and $S_{\boldsymbol{\theta}_i^{(t)}}$ and $\boldsymbol{\theta}_i^{(t+1)}$. ▷ see Algorithm 2
      do n_epoch update of $U_{\widetilde{\boldsymbol{\theta}}_i^{(t)}}$ and $S_{\widetilde{\boldsymbol{\theta}}_i^{(t)}}$ and $\widetilde{\boldsymbol{\theta}}_i^{(t+1)}$.
      $i \leftarrow i + 1$
    end while

    for $k = 1, \ldots, n$ **do**
      retrieve kth row of data from SQLite database.
      recompute $U_{\boldsymbol{\theta}_k^{(t+1)}}$, $U_{\widetilde{\boldsymbol{\theta}}_k^{(t+1)}}$ and their L_2 norms using $\boldsymbol{\theta}^{(t+1)}$ and $\widetilde{\boldsymbol{\theta}}^{(t+1)}$ values.
      recompute $loss_k$ and $\widetilde{loss_k}$ using $\boldsymbol{\theta}^{(t+1)}$ and $\widetilde{\boldsymbol{\theta}}^{(t+1)}$ values.
    end for
    compute the $Loss(\boldsymbol{\theta}^{(t+1)}, \widetilde{\boldsymbol{\theta}}^{(t+1)})$.
    check if the Relative change in Loss is less than ϵ.
  end while
  $t \leftarrow t + 1$
end while

Algorithm 2 Epoch update in inner loop of Algorithm 1

for `epoch` $\in$ `1, ..., n_epoch` **do**
  compute $U_{\boldsymbol{\theta}_i^{(t)}}$ and $I_{\boldsymbol{\theta}_i^{(t)}}$.
  update $\boldsymbol{\theta}_i^{(t+1)}$ using current value of $\{\ \widetilde{\boldsymbol{\theta}}_j^{(t)}, j = 1, \ldots, n\ \}$, and $\boldsymbol{\theta}_i^{(t)}$
  based on formulae in (3.2.7)
end for
for `epoch` $\in$ `1, ..., n_epoch` **do**
  compute $U_{\widetilde{\boldsymbol{\theta}}_i^{(t)}}$ and $I_{\widetilde{\boldsymbol{\theta}}_i^{(t)}}$.
  update $\widetilde{\boldsymbol{\theta}}_i^{(t+1)}$ using current value of $\{\ \boldsymbol{\theta}_j^{(t)}, j = 1, \ldots, n\ \}$, and $\widetilde{\boldsymbol{\theta}}_i^{(t)}$
  based on formulae in (3.2.9)
end for

Algorithm 3 Updating equation in alternating ZIG regression with learning rate adjustment

for $i = 1, \ldots, n$ **do**
  for $k = 1, \ldots, E$ **do**
    $\boldsymbol{\theta}_i^{(t,k+1)} = \boldsymbol{\theta}_i^{(t,k)} + \frac{lr}{t^{1/4}} \cdot S^{-1}_{\boldsymbol{\theta}_i^{(t,k)}} \cdot U_{\boldsymbol{\theta}_i^{(t,k)}}$
    $\widetilde{\boldsymbol{\theta}}_i^{(t,k+1)} = \widetilde{\boldsymbol{\theta}}_i^{(t,k)} + \frac{lr}{t^{1/4}} \cdot S^{-1}_{\widetilde{\boldsymbol{\theta}}_i^{(t,k)}} \cdot U_{\widetilde{\boldsymbol{\theta}}_i^{(t,k)}}$
  end for
  $\boldsymbol{\theta}_i^{(t)} \leftarrow \boldsymbol{\theta}_i^{(t,E)}$
  $\widetilde{\boldsymbol{\theta}}_i^{(t)} \leftarrow \widetilde{\boldsymbol{\theta}}_i^{(t,E)}$
  $\boldsymbol{\theta}_i^{(t+1)} = \boldsymbol{\theta}_i^{(t)} + \frac{lr}{t^{1/4}} \cdot S^{-1}_{\boldsymbol{\theta}_i^{(t)}} \cdot U_{\boldsymbol{\theta}_i^{(t)}}$
  $\widetilde{\boldsymbol{\theta}}_i^{(t+1)} = \widetilde{\boldsymbol{\theta}}_i^{(t)} + \frac{lr}{t^{1/4}} \cdot S^{-1}_{\widetilde{\boldsymbol{\theta}}_i^{(t)}} \cdot U_{\widetilde{\boldsymbol{\theta}}_i^{(t)}}$
end for

3.5 A simulation study with ZIG using log link

In this section, we present a simulation study to assess the performance of the alternating ZIG regression. We found through some experiments that data generation have to be very careful because the mean of the Gamma distribution was given by $\mu_{ij} = \exp(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + e_i + \widetilde{e}_j)$. When we randomly generate the $\mathbf{w}_i$'s and $\widetilde{\boldsymbol{w}}_j$'s independently from Uniform distribution with each of them having dimension 50, their dot product could easily become so large that exponentiated value μ_{ij} and the variance of the distribution ($\propto \mu_{ij}^2$) become infinity or NaN. Keeping this point in mind, we generate the data as follows:

- The w_{ij}'s were generated independently from Uniform(-0.25, 0.25) with seed 99, $i = 1, \ldots, 300$ and $j = 1, \ldots, 50$.
- The $\widetilde{w}_{ij} = w_{ij}$, $i = 1, \ldots, 300$ and $j = 1, \ldots, 50$.
- The b_i's and $\widetilde{b}_i$'s were generated independently from Uniform(0, 0.05) with seed 97 and 96, respectively, $i = 1, \ldots, 300$.
- The e_i's and $\widetilde{e}_i$'s were generated independently from Uniform(0.1, 0.35) with seed 1 and 2, respectively, $i = 1, \ldots, 300$.

- The success probability p_{ij} of positive observations was calculated as $p_{ij} = (1 + \exp(-\eta_{ij}))^{-1}$, where η_{ij} is based on the logit formula (3.0.2).

- Generate B_{ij} independently from Bernoulli distribution with success probability p_{ij}, $i = 1, \ldots, 300$ and $j = i, \ldots, 300$.

- If $B_{ij} = 0$, set $Y_{ij} = 0$. Otherwise, generate Y_{ij} from Gamma distribution with mean $\mu_{ij} = \exp(\tau_{ij})$ and dispersion parameter $\nu_{ij} = 4$, where τ_{ij} is computed based on formula (3.0.4) for $i = 1, \ldots, 300$ and $j = i, \ldots, 300$.

The matrix containing Y_{ij}, $i, j = 1, \ldots, 300$, is the data we used for the alternating ZIG regression. With the generated data matrix Y, we applied our algorithm to the data with two different sets of initial values for the unknown parameters. One of the two settings is to check whether the algorithm would perform well when the parameter estimation is less difficult when part of the parameters were initialized with true values. The other setting demands more accurate estimation in both $\mathbf{w}_i$ and $\widetilde{\boldsymbol{w}}_j$ in the right direction.

In Setting 1, we set all the parameters' initial values equal to their true values except for $\widetilde{\boldsymbol{w}}_{ij}, i = 1, \ldots, 300, j = 1, \ldots, 50$. The initial value of $\widetilde{\boldsymbol{w}}_{ij}$'s were randomly generated from Uniform(-0.25, 0.25) with seed number 98. Even though initial values of $\mathbf{w}, b, \widetilde{b}, e$ and $\widetilde{e}$ were set to be equal to the true values, the algorithm was not aware of this and these parameters were still estimated along with $\widetilde{\boldsymbol{w}}$.

In Setting 2, we generated the initial values for all the parameters randomly as follows:

- $\mathbf{w}$'s initial values were generated independently from Uniform(-0.25, 0.25) with dimension 300 by 50 and seed 102.

- $\widetilde{\boldsymbol{w}}$'s initial values were generated independently from Uniform(-0.25, 0.25) with dimension 300 by 50 and seed 103.

- b_i's and $\widetilde{b}_j$'s initial values were generated independently from Uniform(0, 0.05) with seed 104 and 105, respectively, $i, j = 1, \ldots, 300$.

- e_i's and $\widetilde{e}_j$'s were independently generated from Uniform(0.1, 0.35) with seed 106 and 107, respectively, $i, j = 1, \ldots, 300$.

We consider both update with learning rate adjustment and without learning rate adjustment.

The results of the Setting 1 are presented in Table 3.1 and Figure 3.1. We can see in Figure 3.1 that even though the norm of the score vectors went up after initially reduced, the loss function (i.e., the negative log-likelihood) consistently reduces as the iteration number increases. In the last 5 iterations, the algorithm with no learning rate adjustment reached lower values of overall loss compared to that with learning rate adjustment. However, the norm of the score vectors from the algorithm with learning rate adjustment are much smaller than those from the algorithm with no learning rate adjustment (see Table 3.1).

	no learning rate			with learning rate		
	$\|\|U_{\boldsymbol{\theta}}\|\|_{L_2}$	$\|\|U_{\widetilde{\boldsymbol{\theta}}}\|\|_{L_2}$	Overall Loss	$\|\|U_{\boldsymbol{\theta}}\|\|_{L_2}$	$\|\|U_{\widetilde{\boldsymbol{\theta}}}\|\|_{L_2}$	Overall Loss
1	85805.79	88896.77	77998.77	5596.06	5298.31	78313.71
2	86123.05	89012.56	77996.44	5596.00	5303.81	78312.30
3	86614.46	89236.36	77994.11	5581.83	5294.08	78310.92
4	87309.84	89758.44	77991.79	5579.14	5293.48	78309.59
5	87574.27	89699.66	77989.41	5556.29	5286.12	78308.15

Table 3.1: *Result of last 5 iterations for norm of the score vectors and overall loss in ZIG model with log link initialized with true parameters except $\widetilde{\boldsymbol{w}}$ which was randomly initialized. The algorithm with no learning rate achieved lower overall loss than the one with learning rate but the score vectors' norms are smaller than for the algorithm with learning rate.*

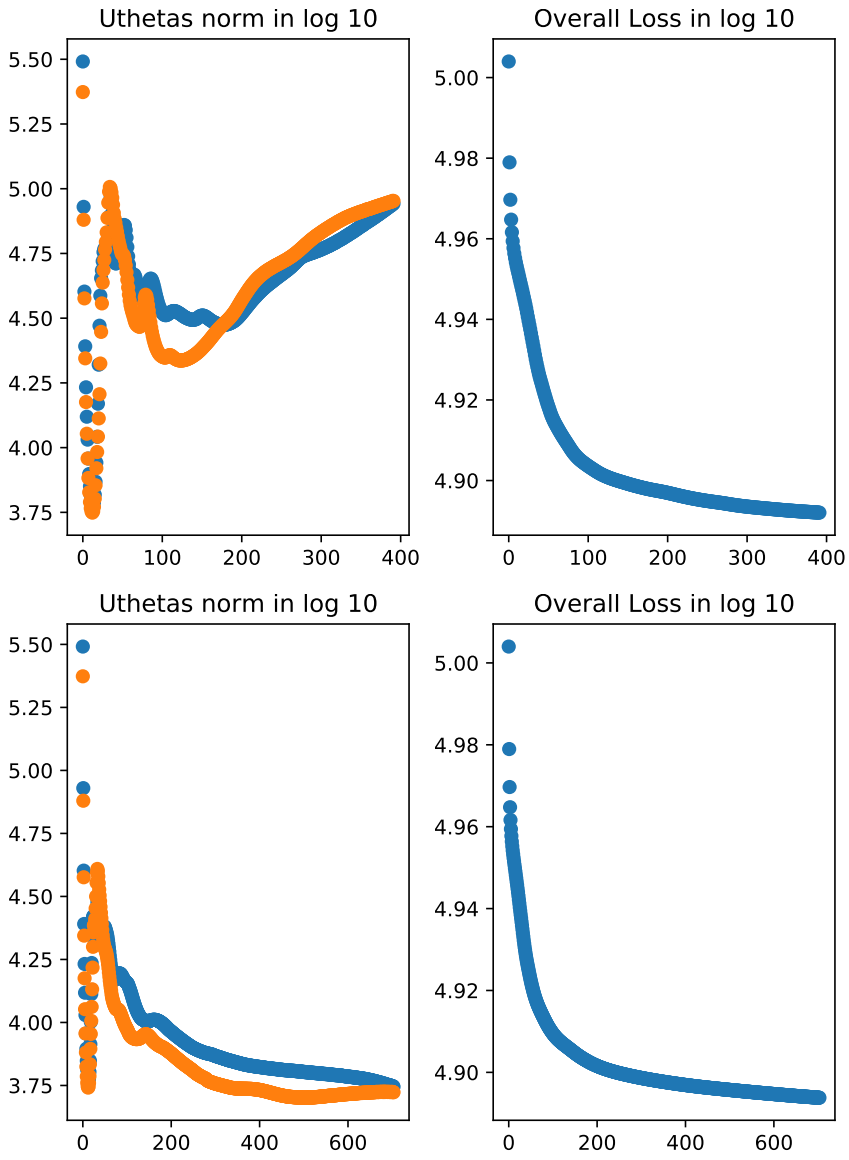

Figure 3.1: *ZIG model without learning rate on top two panels and with learning rate on bottom panels, both initialized with true parameter values except for* $\widetilde{\boldsymbol{w}}$. *The left panels give the norm of the score vectors and the right panels give overall loss in* $\log_{10}$ *scale over iteration. In the top panels, the norm of the score vector reduced drastically in early iterations but increased over later iterations even though the overall loss is consistently reduced. In the bottom panels, the norm of the score vector reduced quickly to a certain level and increased as the other case. However, the norm started to reduce and stabilized as number of iterations increased. The overall loss is showing desired reducing trend throughout all iterations.*

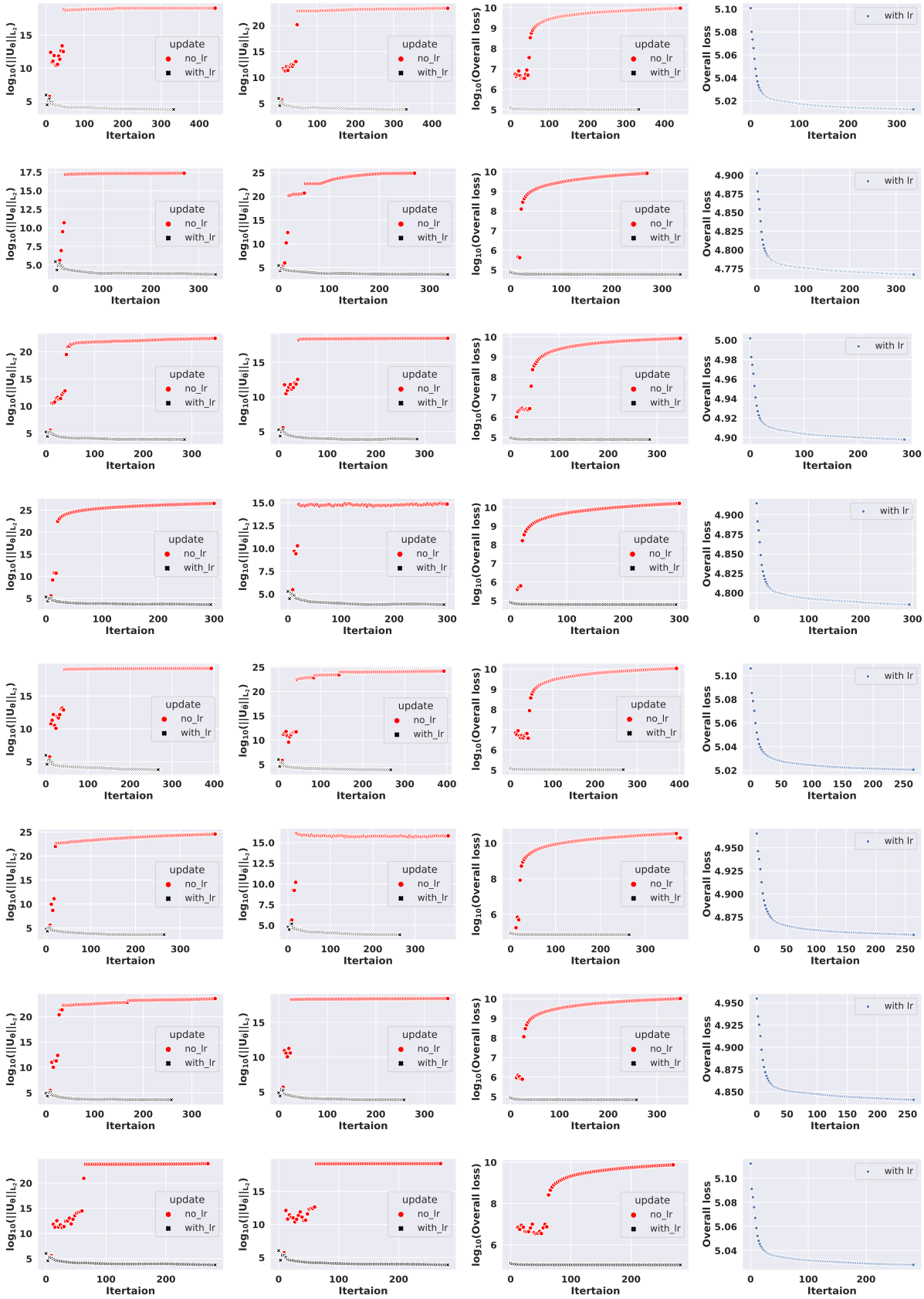

Figure 3.2: *Comparing performance of the alternating ZIG regression with log link algorithm with or without learning rate over 8 simulated datasets. Each row is for one dataset.*

The results for Setting 2 are given in Figure 3.2. The first column and the second column in Figure 3.2 show the norm of the score vectors. The third column is for the overall loss of two algorithms (with or without learning rate adjustment). The last column is given to show the algorithm with learning rate adjustment alone because the scale of the third column is too high to see the reducing trend. In this case, with the initial values of all parameters generated randomly from Uniform distribution, the uncertainty in the parameters caused the algorithm without learning rate to breakdown. We see that the algorithm with no learning rate adjustment struggles to reduce the norms and headed toward wrong direction throughout the optimization process. On the other hand, the algorithm with learning rate adjustment had a little fluctuation at the beginning but continued toward the right direction by reducing both the norm of the score vectors and the loss function as the iterations proceed. Note that the dataset was identical for these two applications and both algorithms started with the same initial values. The only difference between the two algorithms is that learning rate adjustment is used in one algorithm but not in the other algorithm. Hence, the failure of the algorithm with no learning rate adjustment is due to update being too big which leads to the algorithm going toward wrong directions.

In summary, we presented shared parameter alternating ZIG regression in this chapter. The logit link is used for modelling the zero versus positive observations. Two link functions for the Gamma part were considered: the canonical link and the log link. We derived updating formulae in both cases. An algorithm is given to alternately update the parameters $\boldsymbol{\theta}$ and $\widetilde{\boldsymbol{\theta}}$ while holding one of them fixed. The Fisher scoring algorithm with or without learning rate adjustment were employed in each side of the alternating update. A simulation study showed that the learning rate adjustment is important in the alternating regression. Without learning rate adjustment, we saw the algorithm failed to find the right optimizing direction. Application of the proposed method requires further effort.

Chapter 4

Alternating Tweedie regression model with shared parameters

In previous chapter, we have been using the zero inflated gamma to describe cooccurence count. A Gamma random variable can be thought as a single source of signal from Gamma distribution or the sum of a fixed number of Gamma distributed signals with the same scale parameter. That is, if X_i independently follows Gamma distribution with shape parameter α_i and scale parameter β, $i = 1, \ldots, n$, then $\sum_{i=1}^{n} X_i$ follows Gamma distribution with shape parameter $\sum_{i=1}^{n} \alpha_i$ and scale parameter β. This data generation mechanism may be suitable in some settings such as the word pair cooccurrence count in an NLP task, where the articles being analyzed are all from a specific technical field. However, in general, the sources that created the cooccurrence count most likely do not have homogeneous mechanism to generate a positive count versus zero. A more realistic scenario is that we have a large number of sources but not all sources generate positive cooccurrence counts. Instead, some of them do generate positive counts and the number of such sources is likely to be a random variable. The Compound Poisson Gamma distribution might fit this setting well. This distribution is equivalent to the sum of independent Gamma random variables with the same shape and scale parameters but the number of Gamma random variables is randomly

distributed from the Poisson distribution rather than being fixed. This distribution is in the Tweedie family. The same shape and scale parameters are for the same item-item pairs from difference sources but different item-item pairs could have different shape and scale parameters. Eventually, these parameters are being modeled in the regression. In this chapter, we present the alternating Tweedie regression model with shared parameters and give the parameter estimation in this model. Before we get to the regression, we first have a more detailed look at the Tweedie distribution and describe how the parameters of Gamma distribution and Tweedie distribution are related to each other.

The Tweedie distribution family contains several commonly used distributions which are determined by the power parameter p.

- The Tweedie distribution belongs to exponential family. All exponential dispersion models that are closed to scale transformation belongs to the Tweedie distribution family (see Theorem 4.1 of Jorgensen 1997). In particular, if X has Tweedie distribution with mean μ and dispersion parameter ϕ, then cX has Tweedie distribution with mean $c\mu$ and dispersion parameter $c^{2-p}\phi$. It has mean and variance relationship as follows

$$E(Y) = \mu \quad \text{Var}(Y) = \phi\mu^p,$$

  where ϕ is the dispersion parameter having value greater than 0.

- The Compound Poisson Gamma distribution corresponds to the case that the power parameter satisfies $p \in (1, 2)$. In this case, the distribution has non-negative support and can have a discrete mass at zero, making it useful to model responses that are a mixture of zeros and positive values.

- The distributions that correspond to other power values of p:

  - When $p < 0$, the distribution is the Extreme stable distribution.
  - When $p = 0$, the distribution is the Normal distribution.

- When $p = 1$, the distribution coincides with the Poisson distribution.
- When $p = 2$, the distribution is the Gamma distribution.
- When $2 < p < 3$, the distribution is the Positive stable distribution.
- When $p = 3$, the distribution is the Inverse Gaussian distribution.
- When $p > 3$, the distribution is the Positive stable distribution.
- When $p = \infty$, the distribution is the Extreme stable distribution.

- The mean domain is R if p is 0 or ∞, and is R_+ for all other $p \neq 0$. Therefore, except when the distribution is Normal or Extreme stable with $p = \infty$, the log link is frequently used as the link function for the Tweedie regression.

- The positive zero mass is only allowed when $1 \leq p < 2$. Since our cooccurrence count has many zeros, we focus our attention to the Compound Poisson Gamma distribution which is the case when $1 < p < 2$.

The Gamma and Poisson distributions together have parameters α, β, and λ. The Tweedie distribution uses parameters μ, ϕ, p. Next, we will see how they are related in the case of Compound Poisson Gamma.

Let $N \sim \text{Poisson}(\lambda)$, $\quad X_i \overset{i.i.d.}{\sim} \text{Gamma}(\alpha, \beta)$. Assume N and X_i's are independent for all i. For $Y = 0$, the probability comes from the Poisson distribution

$$\Pr[Y = 0] = \frac{\lambda^y e^{-\lambda}}{y!} = e^{-\lambda}.$$

For $Y > 0$, note that $Y \mid N = n \sim \text{Gamma}(n\alpha, \beta)$ by additive property of the Gamma distribution. Then the distribution of Y when $Y > 0$ can be obtained with the law of total probability as the product of conditional distribution of $Y \mid N$ and the marginal distribution of N,

$$f_Y(y) = \sum_{n=0}^{\infty} f_{Y,N}(y, n) = \sum_{n=0}^{\infty} f_{YIN}(y \mid n) \cdot f_N(n).$$

Recall, $N \sim \text{Poisson}(\lambda)$ and $Y \mid N = n \sim \text{Gamma}(n\alpha, \beta)$. Therefore, for $y > 0$,

$$\begin{aligned} f_Y(y) &= \sum_{n=0}^{\infty} \frac{\beta^{n\alpha}}{\Gamma(n\alpha)} y^{n\alpha-1} e^{-\beta y} \cdot \frac{\lambda^n e^{-\lambda}}{n!} \\ &= e^{-\beta y} \cdot e^{-\lambda} \cdot \sum_{n=0}^{\infty} \frac{\beta^{nx}}{\Gamma(n\alpha)} y^{n\alpha-1} \cdot \frac{\lambda^n}{n!} \\ &= \exp\left\{ -\beta y - \lambda + \log\left(\sum_{n=0}^{\infty} \frac{\beta^{n\alpha}}{\Gamma(n\alpha)} \cdot y^{n\alpha-1} \cdot \frac{\lambda^n}{n!} \right) \right\}. \end{aligned} \tag{4.0.1}$$

The above form was derived by using the Poisson distribution and conditional distribution of Gamma. Meanwhile, the Tweedie distribution has its own canonical form in exponential family format when $Y > 0$:

$$f(y) = \exp\left\{ \frac{y\theta - \kappa(\theta)}{\phi} + c(y, \phi, p) \right\},$$

where $\theta = \frac{\mu^{1-p}}{1-p}$ and $\kappa(\theta) = \frac{\mu^{2-p}}{2-p}$. This implies that

$$\lambda = \frac{\mu^{2-p}}{\phi(2-p)}, \quad \alpha = \frac{2-p}{p-1}, \quad \frac{1}{\beta} = \phi(p-1)\mu^{p-1},$$

and the $c(y, \phi, p)$ corresponds to the log sum in (4.0.1).

4.1 MLE for alternating Tweedie regression

Once we observe the cooccurence count from the data and assuming the count is from the Compound Poisson Gamma, we could express probability density function in canonical form for the cooccurrence counts as below:

For $y_{ij} = 0$ case,

$$f_0(0) = \Pr[Y_{ij} = 0] = \exp(-\lambda_{ij}),$$

where

$$\lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}(2-p_{ij})} \quad \text{(see page 5 of Bonat and Kokonendji 2017)}$$

For $y_{ij} > 0$ case,

$$f\left(y_{ij}\right) = \exp\left\{\frac{y_{ij}\theta_{ij} - \kappa\left(\theta_{ij}\right)}{\phi_{ij}} + c\left(y_{ij}, \phi_{ij}, p_{ij}\right)\right\}$$

where

$$\theta_{ij} = \frac{\mu_{ij}^{1-p_{ij}}}{1-p_{ij}} = \frac{\exp\left\{(1-p_{ij})\left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j\right)\right\}}{(1-p_{ij})},$$

$$\kappa\left(\theta_{ij}\right) = \frac{\mu_{ij}^{2-p_{ij}}}{2-p_{ij}} = \frac{\exp\left\{(2-p_{ij})\left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j\right)\right\}}{(2-p_{ij})},$$

$$c\left(y_{ij}, \phi_{ij}, p_{ij}\right) = \log\left(\sum_{k=1}^{\infty} \frac{\beta_{ij}^{k\alpha_{ij}}}{\Gamma(k\alpha_{ij})} \cdot y_{ij}^{k\alpha_{ij}-1} \cdot \frac{\lambda_{ij}^k}{k!}\right), \quad \text{if } 1 < p_{ij} < 2, \tag{4.1.1}$$

$$c(y_{ij}, \phi_{ij}, p_{ij}) = \frac{1}{\pi y_{ij}} \sum_{n=1}^{\infty} \frac{\Gamma(1-\alpha_{ij}n)\phi_{ij}^{n(-\alpha_{ij}-1)}(p_{ij}-1)^{-\alpha_{ij}n}}{(-1)^n \Gamma(1+n)(p_{ij}-2)^n y_{ij}^{-\alpha_{ij}n}} \sin(n\pi\alpha_{ij}), \text{ if } p_{ij} > 2, \tag{4.1.2}$$

$$c(y_{ij}, \phi_{ij}, p_{ij}) = \frac{1}{\pi y_{ij}} \sum_{k=1}^{\infty} \frac{\Gamma\left(1+\frac{k}{\alpha_{ij}}\right)(-y_{ij})^k}{k!\lambda_{ij}^{\frac{k}{\alpha_{ij}}} \kappa_{p_{ij}}^{\frac{k}{\alpha_{ij}}}(1)} \sin\left(\frac{-k\pi}{\alpha_{ij}}\right), \text{ if } p_{ij} < 0, \tag{4.1.3}$$

with

$$\lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}(2-p_{ij})}, \quad \alpha_{ij} = \frac{2-p_{ij}}{p_{ij}-1}, \quad \frac{1}{\beta_{ij}} = \phi_{ij}(p_{ij}-1)\mu_{ij}^{p_{ij}-1}, \quad \kappa_{p_{ij}}(1) = \frac{(p_{ij}-1)^{-\alpha_{ij}}}{p_{ij}-2}(-1)^{\frac{1}{p_{ij}-1}}.$$

We use commonly used log link function, which accounts for the mean domain well as mentioned earlier.

$$\log \mu_{ij} = \mathbf{w}_i^T \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j, \quad i, j = 1, \cdots, n \quad \text{and} \quad \mathbf{w}_i, \widetilde{\boldsymbol{w}}_j \in \mathbb{R}^d, b_i, \widetilde{b}_j \in \mathbb{R}.$$

Thus, the log-likelihood function for i^{th} row of data limited to the element $y_{ij} > 0$ case while

$\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{b}}$ are held fixed is

$$\ell_i^{(1)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) = \sum_{j=1}^{n} \left[\frac{y_{ij}\theta_{ij} - \kappa(\theta_{ij})}{\phi_{ij}} + c(y_{ij}, \phi_{ij}, p_{ij}) \right] I(y_{ij} > 0), \tag{4.1.4}$$

and the log-likelihood function for j^{th} column of data limited to the element $y_{ij} > 0$ case while $\mathbf{w}$ and $\mathbf{b}$ are held fixed is

$$\widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}, \mathbf{b}) = \sum_{i=1}^{n} \left[\frac{y_{ij}\theta_{ij} - \kappa(\theta_{ij})}{\phi_{ij}} + c(y_{ij}, \phi_{ij}, p_{ij}) \right] I(y_{ij} > 0), \tag{4.1.5}$$

where

$$p_{ij} = p_i + \tilde{p}_j, \quad \phi_{ij} = \phi_i \cdot \tilde{\phi}_j.$$

Here, the power p_{ij} is decomposed into two terms additively, one depends on the row and the other depends on the column index. The dispersion parameter is a scaling parameter so we wrote it as the product of two dispersion parameters, again one depends on the row and the other depends on the column. This decomposition is inspired by the fact that the item-item cooccurrence count data matrix is symmetric. If the data matrix is user-item matrix, this additive effect may not be sufficient. Unfortunately, if each y_{ij} has its own p_{ij}, ϕ_{ij} and μ_{ij}, then we do not have enough degrees of freedom to estimate those parameters.

Holding $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{b}}$ fixed, the total log likelihood ℓ for all rows is

$$\begin{aligned} \ell(\mathbf{w}, \mathbf{b}; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) &= \sum_{i=1}^{n}\sum_{j=1}^{n} \{\log f(y_{ij}) \cdot I(y_{ij} > 0) + \log f_0(y_{ij}) \cdot I(y_{ij} = 0)\} \\ &= \sum_{i=1}^{n} \ell_i^{(1)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) + \sum_{i=1}^{n}\sum_{j=1}^{n} \ell_{ij}^{(0)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}_j, \widetilde{b}_j), \end{aligned}$$

where

$$\ell_{ij}^{(0)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}_j, \widetilde{b}_j) = -\lambda_{ij} I(y_{ij} = 0) \text{ and } \lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}(2 - p_{ij})}.$$

This is because $\Pr[Y_{ij}=0] = \exp(-\lambda_{ij})$. For $y_{ij}=0$ case, we also have the log link. Therefore, $\log \mu_{ij} = \eta_{ij}$ and $\mu_{ij} = \exp(\eta_{ij})$.

Similarly, holding $\mathbf{w}$ and $\mathbf{b}$ fixed, the total log likelihood $\widetilde{\ell}$ for all columns is

$$\widetilde{\ell}(\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}; \mathbf{w}, \mathbf{b}) \;=\; \sum_{j=1}^{n} \widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}, \mathbf{b}) + \sum_{i=1}^{n}\sum_{j=1}^{n} \widetilde{\ell}_{ij}^{(0)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}_i, b_i),$$

where

$$\widetilde{\ell}_{ij}^{(0)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}_i, b_i) = -\lambda_{ij} I(y_{ij}=0) \text{ and } \lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}(2-p_{ij})}.$$

To derive partial derivatives for the case $y_{ij} > 0$, note that

$$\begin{aligned}
&\mu_{ij} = \kappa'(\theta_{ij}) = \exp(\eta_{ij}) \quad \Rightarrow \frac{\partial \mu_{ij}}{\partial \eta_{ij}} = \exp(\eta_{ij}),\ \mathrm{Var}(Y_{ij}) = \phi_{ij}\mu_{ij}^{p_{ij}},\\
&\eta_{ij} = \log(\mu_{ij}), \quad \frac{\partial \theta_{ij}}{\partial \mu_{ij}} = \frac{1}{\partial \mu_{ij}/\partial \theta_{ij}} = \frac{1}{\kappa''(\theta_j)} = \frac{1}{V_{ij}} = \frac{1}{\mu_{ij}^{p_{ij}}}.
\end{aligned}$$

The first order derivatives of i^{th} row log-likelihood and j^{th} column log-likelihood which are components of the Score functions in the case $y_{ij} > 0$ are

$$\begin{aligned}
\frac{\partial \ell_i^{(1)}}{\partial w_{i_1 k}} &= \sum_{j=1}^{n} \frac{\partial \ell_{ij}^{(1)}}{\partial \theta_{ij}} \cdot \frac{\partial \theta_{ij}}{\partial \mu_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial w_{i_1 k}} = \sum_{j=1}^{n} \frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \cdot \frac{1}{V_{ij}} \cdot e^{\eta_{ij}} \cdot \widetilde{w}_{jk} \cdot I(i=i_1) I(y_{ij}>0)\\
&= \sum_{j=1}^{n} \frac{y_{i_1 j}-\mu_{i_1 j}}{\phi_{i_1 j}} \cdot \frac{1}{V_{i_1 j}} \cdot \mu_{i_1 j} \cdot \widetilde{w}_{jk} \cdot I(i=i_1) I(y_{ij}>0),\\
\frac{\partial \ell_i^{(1)}}{\partial b_{i_1}} &= \sum_{j=1}^{n} \frac{\partial \ell_{ij}^{(1)}}{\partial \theta_{ij}} \cdot \frac{\partial \theta_{ij}}{\partial \mu_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_j}{\partial b_{i_1}} = \sum_{j=1}^{n} \frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot I(i=i_1) I(y_{ij}>0),\\
\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{w}_{j_1 k}} &= \sum_{i=1}^{n} \frac{\partial \widetilde{\ell}_{ij}^{(1)}}{\partial \theta_{ij}} \cdot \frac{\partial \theta_{ij}}{\partial \mu_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial \widetilde{w}_{j_1 k}} = \sum_{i=1}^{n} \frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot w_{ik} \cdot I(j=j_1) I(y_{ij}>0),\\
\frac{\partial \widetilde{\ell}_j^{(1)}}{\partial \widetilde{b}_{j_1}} &= \sum_{i=1}^{n} \frac{\partial \widetilde{\ell}_{ij}^{(1)}}{\partial \theta_{ij}} \cdot \frac{\partial \theta_{ij}}{\partial \mu_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_j}{\partial \widetilde{b}_{j_1}} = \sum_{i=1}^{n} \frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot I(j=j_1) I(y_{ij}>0).
\end{aligned}$$

The first order derivatives for $\ell_{ij}^{(0)}$ and $\widetilde{\ell}_{ij}^{(0)}$ are

$$\begin{aligned}
\frac{\partial \ell_{ij}^{(0)}}{\partial w_{i_1k}} &= -\frac{\partial \lambda_{ij}}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial \eta_{ij}}\frac{\partial \eta_{ij}}{\partial w_{i_1k}} = -\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\widetilde{w}_{jk}I(i=i_1)I(y_{ij}=0),\\
\frac{\partial \ell_{ij}^{(0)}}{\partial b_{i_1}} &= -\frac{\partial \lambda_{ij}}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial \eta_{ij}}\frac{\partial \eta_{ij}}{\partial b_{i_1}} = -\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}I(i=i_1)I(y_{ij}=0),\\
\frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{w}_{j_1k}} &= -\frac{\partial \lambda_{ij}}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial \eta_{ij}}\frac{\partial \eta_{ij}}{\partial \widetilde{w}_{j_1k}} = -\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}w_{ik}I(j=j_1)I(y_{ij}=0),\\
\frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_{j_1}} &= -\frac{\partial \lambda_{ij}}{\partial \mu_{ij}}\frac{\partial \mu_{ij}}{\partial \eta_{ij}}\frac{\partial \eta_{ij}}{\partial \widetilde{b}_{j_1}} = -\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}I(j=j_1)I(y_{ij}=0).
\end{aligned}$$

Hence, the k^{th} component, $k=1,\cdots,d$, of the score functions corresponding to i^{th} row and j^{th} column of the log likelihood are

$$\begin{aligned}
[U_{w_i}]_k &= \frac{\partial \ell}{\partial w_{ik}} = \sum_{j=1}^{n}\left\{\frac{y_{ij}-\mu_{ij}}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot\widetilde{w}_{jk}\cdot I\left(y_{ij}>0\right)+\frac{-\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\cdot\widetilde{w}_{jk}\cdot I\left(y_{ij}=0\right)\right\},\\
\left[U_{\widetilde{w}_j}\right]_k &= \frac{\partial \widetilde{\ell}}{\partial \widetilde{w}_{jk}} = \sum_{i=1}^{n}\left\{\frac{y_{ij}-\mu_{ij}}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot w_{ik}\cdot I\left(y_{ij}>0\right)+\frac{-\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\cdot w_{ik}\cdot I\left(y_{ij}=0\right)\right\},
\end{aligned}$$

and

$$\begin{aligned}
U_{b_i} &= \frac{\partial \ell}{\partial b_i} = \sum_{j=1}^{n}\left\{\frac{y_{ij}-\mu_{ij}}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot I\left(y_{ij}>0\right)+\frac{-\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\cdot I\left(y_{ij}=0\right)\right\},\\
U_{\widetilde{b}_j} &= \frac{\partial \widetilde{\ell}}{\partial \widetilde{b}_j} = \sum_{i=1}^{n}\left\{\frac{y_{ij}-\mu_{ij}}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot I\left(y_{ij}>0\right)+\frac{-\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\cdot I\left(y_{ij}=0\right)\right\}.
\end{aligned}$$

Let $\boldsymbol{\beta}_i=(\mathbf{w}_i^\top,b_i)^\top$ and $\widetilde{\boldsymbol{\beta}}_j=(\widetilde{\boldsymbol{w}}_j^\top,\widetilde{b}_j)^\top$. From the partial derivatives of the log likelihoods, we can see that the second order partial derivatives with respect to $\boldsymbol{\beta}_i$ and $\boldsymbol{\beta}_{i_1}$ is zero when i is different from i_1. That is, the cross terms $\frac{\partial^2\ell}{\partial \mathbf{w}_i\partial \mathbf{w}_{i_1}}=0$ and $\frac{\partial^2\widetilde{\ell}}{\partial \widetilde{\boldsymbol{w}}_i\partial \widetilde{\boldsymbol{w}}_{i_1}}=0$ and $\frac{\partial^2\ell}{\partial \mathbf{w}_i\partial b_{i_1}}=0$ and $\frac{\partial^2\widetilde{\ell}}{\partial \widetilde{\boldsymbol{w}}_i\partial \widetilde{b}_{i_1}}=0$ when $i\neq i_1$. Therefore, the estimation of $\boldsymbol{\beta}_i$ and $\boldsymbol{\beta}_{i1}$ can be conducted separately. Denote

$$I_{\boldsymbol{\beta}_i}=\begin{bmatrix} I_{\mathbf{w}_i\mathbf{w}_i} & I_{\mathbf{w}_ib_i}\\ I_{b_i\mathbf{w}_i} & I_{b_ib_i}\end{bmatrix},\quad I_{\widetilde{\boldsymbol{\beta}}_j}=\begin{bmatrix} I_{\widetilde{\boldsymbol{w}}_j\widetilde{\boldsymbol{w}}_j} & I_{\widetilde{\boldsymbol{w}}_j\widetilde{b}_j}\\ I_{\widetilde{b}_j\widetilde{\boldsymbol{w}}_j} & I_{\widetilde{b}_j\widetilde{b}_j}\end{bmatrix},$$

where the individual components in the matrix $I_{\boldsymbol{\beta}_i}$ and $I_{\widetilde{\boldsymbol{\beta}}_j}$ are negative expectations of the second order partial derivatives. We list them below and detailed derivation can be found in Appendix A.

The $(k,r)^{th}$ entry, $k,r = 1,\ldots,d$, of the Information matrix for $\mathbf{w}_i$, $\widetilde{\boldsymbol{w}}_j$, b_i, and $\widetilde{b}_j$ respectively are

$$\begin{aligned}
\left[I_{\mathbf{w}_{i_1}\mathbf{w}_{i_2}}\right]_{k,r} &= -E\left[\frac{\partial^2\ell}{\partial w_{i_1k}\partial w_{i_2r}}\right] = -\sum_{i=1}^{n}\sum_{j=1}^{n} E\left[\frac{\partial^2\left(\ell_{ij}^{(0)}+\ell_{ij}^{(1)}\right)}{\partial w_{i_1k}\partial w_{i_2r}}\right] \\
&= \sum_{j=1}^{n}\frac{\mu_{i_1j}^2}{\phi_{i_1j}V_{i_1j}}\cdot\widetilde{w}_{jk}\cdot\widetilde{w}_{jr}\cdot I(i_1=i_2)\cdot I\left(y_{i_1j}>0\right) + \\
&\quad \sum_{j=1}^{n}\frac{(2-p_{i_1j})}{\phi_{i_1j}}\cdot\mu_{i_1j}^{2-p_{i_1j}}\cdot\widetilde{w}_{jk}\cdot\widetilde{w}_{jr}\cdot I(i_1=i_2)\cdot I\left(y_{i_1j}=0\right), \\
\left[I_{\widetilde{\boldsymbol{w}}_{j_1}\widetilde{\boldsymbol{w}}_{j_2}}\right]_{k,r} &= -E\left[\frac{\partial^2\widetilde{\ell}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{w}_{j_2r}}\right] = -\sum_{i=1}^{n}\sum_{j=1}^{n} E\left[\frac{\partial^2\left(\widetilde{\ell}_{ij}^{(0)}+\widetilde{\ell}_{ij}^{(1)}\right)}{\partial\widetilde{w}_{j_1k}\partial\widetilde{w}_{j_2r}}\right] \\
&= \sum_{i=1}^{n}\frac{\mu_{ij_1}^2}{\phi_{ij_1}V_{ij_1}}\cdot w_{ik}\cdot w_{ir}\cdot I(j_1=j_2)\cdot I\left(y_{ij_1}>0\right) + \\
&\quad \sum_{i=1}^{n}\frac{(2-p_{ij_1})}{\phi_{ij_1}}\cdot\mu_{ij_1}^{2-p_{ij_1}}\cdot w_{ik}\cdot w_{ir}\cdot I(j_1=j_2)\cdot I\left(y_{ij_1}=0\right),
\end{aligned}$$

$$\begin{aligned}
I_{b_{i_1}b_{i_2}} &= -E\left[\frac{\partial^2\ell}{\partial b_{i_1}\partial b_{i_2}}\right] = -\sum_{i=1}^{n}\sum_{j=1}^{n} E\left[\frac{\partial^2\left(\ell_{ij}^{(0)}+\ell_{ij}^{(1)}\right)}{\partial b_{i_1}\partial b_{i_2}}\right] \\
&= \sum_{j=1}^{n}\frac{\mu_{i_1j}^2}{\phi_{i_1j}V_{i_1j}}\cdot I(i_1=i_2)\cdot I\left(y_{i_1j}>0\right) + \sum_{j=1}^{n}\frac{(2-p_{i_1j})}{\phi_{i_1j}}\cdot\mu_{i_1j}^{2-p_{i_1j}}\cdot I(i_1=i_2)\cdot I\left(y_{i_1j}=0\right), \\
I_{\widetilde{b}_{j_1}\widetilde{b}_{j_2}} &= -E\left[\frac{\partial^2\widetilde{\ell}}{\partial b_{j_1}\partial b_{j_2}}\right] = -\sum_{i=1}^{n}\sum_{j=1}^{n} E\left[\frac{\partial^2\left(\widetilde{\ell}_{ij}^{(0)}+\widetilde{\ell}_{ij}^{(1)}\right)}{\partial b_{j_1}\partial b_{j_2}}\right] \\
&= \sum_{i=1}^{n}\frac{\mu_{ij_1}^2}{\phi_{ij_1}V_{ij_1}}\cdot I(j_1=j_2)\cdot I\left(y_{ij_1}>0\right) + \sum_{i=1}^{n}\frac{(2-p_{ij_1})}{\phi_{ij_1}}\cdot\mu_{ij_1}^{2-p_{ij_1}}\cdot I(j_1=j_2)\cdot I\left(y_{ij_1}=0\right).
\end{aligned}$$

The off diagonal components of $I_{\boldsymbol{\beta}_i}$, $I_{\widetilde{\boldsymbol{\beta}}_j}$ are

$$\begin{aligned}
\left[I_{\mathbf{w}_{i_1} b_{i_2}}\right]_k &= -E\left[\frac{\partial^2 \ell}{\partial w_{i_1 k} \partial b_{i_2}}\right] = -\sum_{i=1}^{n}\sum_{j=1}^{n} E\left[\frac{\partial^2 \left(\ell_{ij}^{(0)} + \ell_{ij}^{(1)}\right)}{\partial w_{i_1 k} \partial b_{i_2}}\right] \\
&= \left\{\sum_{j=1}^{n} \frac{\mu_{i_1 j}^2}{\phi_{i_1 j}} \cdot \frac{\widetilde{w}_{jk}}{V_{i_1 j}} \cdot I\left(y_{i_1 j} > 0\right) + \sum_{j=1}^{n} \frac{(2 - p_{i_1 j})}{\phi_{i_1 j}} \cdot \mu_{i_1 j}^{2-p_{i_1 j}} \widetilde{w}_{jk} \cdot I\left(y_{i_1 j} = 0\right)\right\} \cdot I(i_1 = i_2),
\end{aligned}$$

and

$$\begin{aligned}
\left[I_{\widetilde{\boldsymbol{w}}_{j_1} \widetilde{b}_{j_2}}\right]_k &= -E\left[\frac{\partial^2 \widetilde{\ell}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{b}_{j_2}}\right] = -\sum_{i=1}^{n}\sum_{j=1}^{n} E\left[\frac{\partial^2 \left(\widetilde{\ell}_{ij}^{(0)} + \widetilde{\ell}_{ij}^{(1)}\right)}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{b}_{j_2}}\right] \\
&= \left\{\sum_{i=1}^{n} \frac{\mu_{i j_1}^2}{\phi_{i j_1}} \cdot \frac{w_{ik}}{V_{i j_1}} \cdot I\left(y_{i j_1} > 0\right) + \sum_{i=1}^{n} \frac{(2 - p_{i j_1})}{\phi_{i j_1}} \cdot \mu_{i j_1}^{2-p_{i j_1}} w_{ik} \cdot I\left(y_{i j_1} = 0\right)\right\} \cdot I(j_1 = j_2).
\end{aligned}$$

Note that the components $-E\left[\frac{\partial^2 \ell}{\partial w_{ik} \partial b_i}\right]$ are non-zero, so we can not separately estimate $\mathbf{w}_i$ and b_i. They have to be estimated together. Similarly, $\widetilde{\boldsymbol{w}}_j$ and $\widetilde{b}_j$ have to be estimated together.

Based on the Fisher scoring algorithm, the updating formulae for $\boldsymbol{\beta}_i = (\mathbf{w}_i^\top, b_i)^\top$ and $\widetilde{\boldsymbol{\beta}}_j = (\widetilde{\boldsymbol{w}}_j^\top, \widetilde{b}_j)^\top$ are respectively,

$$\begin{aligned}
\boldsymbol{\beta}_i^{(t+1)} &= \boldsymbol{\beta}_i^{(t)} + I_{\boldsymbol{\beta}_i^{(t)}}^{-1} \cdot U_{\boldsymbol{\beta}_i^{(t)}}, \\
\widetilde{\boldsymbol{\beta}}_j^{(t+1)} &= \widetilde{\boldsymbol{\beta}}_j^{(t)} + I_{\widetilde{\boldsymbol{\beta}}_j^{(t)}}^{-1} \cdot U_{\widetilde{\boldsymbol{\beta}}_j^{(t)}}.
\end{aligned} \tag{4.1.6}$$

Note that $\boldsymbol{\beta}_i$ and $\widetilde{\boldsymbol{\beta}}_j$ can not be estimated simultaneously because estimation of $\boldsymbol{\beta}_i$ requires to use the entire set of $\{\widetilde{\boldsymbol{\beta}}_j, j = 1, \ldots, n\}$ as fixed values. Reversely, $\{\boldsymbol{\beta}_i, i = 1, \ldots, n\}$ serve as data when estimating $\widetilde{\boldsymbol{\beta}}_j$. Since the estimate of $\boldsymbol{\beta}_i$ relies on the current value of $\widetilde{\boldsymbol{\beta}}$ which is not necessarily equal to the true value of the parameter, the $\boldsymbol{\beta}_i$'s estimate is not final even if the first updating equation in (4.1.6) converged while the same value of $\widetilde{\boldsymbol{\beta}}$ is used. When $\widetilde{\boldsymbol{\beta}}$ changes in further iterations, the estimate of $\boldsymbol{\beta}_i$ will change too. Running either of

equations in (4.1.6) is generally referred as the iteratively reweighted least squares (IRLS) algorithm. In our case, we have to alternatingly rerun IRLS update by switching what is held fixed. In addition, since all the $\widetilde{\boldsymbol{\beta}}$ are needed when estimating $\boldsymbol{\beta}_i$ but the IRLS update only works with one component at a time, $\widetilde{\boldsymbol{\beta}}_j$, the process internally depends on each other when we loop through i or j over different iterations. Below we will describe the algorithm for the estimating process.

The algorithm starts with setting the following initial values:

- Specify values of ϕ and p. ▷ (see Section 4.1.1)
- Initialize elements of $\boldsymbol{\beta}_i^{(0)}$ and $\widetilde{\boldsymbol{\beta}}_i^{(0)}$. For example, with i.i.d. Uniform(-0.5, 0.5) values.
- It is natural to use the negative log likelihood as the loss for fitting each row and column. However, the log likelihood functions ℓ and $\widetilde{\ell}$ both contain the $c(y_{ij}, \phi_{ij}, p_{ij})$. This term does not involve regression parameters $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$. Therefore, it is redundant to compute the term $c(y_{ij}, \phi_{ij}, p_{ij})$ when estimating $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$. This is specially the case for Tweedie regression because $c(y_{ij}, \phi_{ij}, p_{ij})$ has log sum of infinite many terms. In Python package sklearn.linear_model.TweedieRegressor, the loss is defined as Half Tweedie deviance $max(y_i, 0)^{2-p}/(1-p)/(2-p) - y_i\mu_i^{1-p}/(1-p) + \mu_i^{2-p}/(2-p)$ when the observed response is y_i (see the Python source code at https://github.com/scikit-learn/scikit-learn/blob/36958fb240fbe435673a9e3c52e769f01f36bec0/sklearn/_loss/loss.py). The first term in it does not depend on the regression parameter β. The other two terms are the negative value of $(y\theta - \kappa(\theta))$ in the log likelihood (6.0.4). Therefore, using the Half Tweedie deviance as the loss is equivalent to using the -$(y\theta - \kappa(\theta))$. We define our loss as follows:

$$Loss(\boldsymbol{\beta}, \widetilde{\boldsymbol{\beta}}) = -\sum_{i=1}^{n}\sum_{j=1}^{n}(y_{ij}\theta_{ij} - \kappa(\theta_{ij})).$$

Note that the loss is a function of $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$, which will change as the algorithm proceeds.

- Set convergence threshold such as $\epsilon = 10^{-4}$ and maximum number of iterations ($maxit$). We use relative convergence criteria on the loss function by checking if the relative change in loss defined below is less than the threshold. This is consistent with $glm2$ package in R which uses the relative convergence criteria on the log likelihood.

$$\text{Relative change in Loss } = \frac{|Loss(\boldsymbol{\beta}^{(t+1)}, \widetilde{\boldsymbol{\beta}}^{(t+1)}) - Loss(\boldsymbol{\beta}^{(t)}, \widetilde{\boldsymbol{\beta}}^{(t)})|}{|Loss(\boldsymbol{\beta}^{(t+1)}, \widetilde{\boldsymbol{\beta}}^{(t+1)})| + 0.1}. \tag{4.1.7}$$

When the algorithm stops, either the maximum number of iterations is reached or the algorithm converged. If t is less than $maxit$, the algorithm has converged. With our loss definition, we are making an assumption that ϕ and p are not involved in likelihood comparison. This makes sense if the likelihood function is evaluated at the same set of ϕ and p values. If the likelihood has to be compared with different ϕ and p pairs, the likelihood corresponds to different pairs can not be compared because the infinite summation in (6.0.1), (6.0.2), (6.0.3) are different.

Also, note that the updating formula (4.1.6) does not have a learning rate. If the model is completely correct, this updating formula will lead to good estimate of $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$. However, if there is any disturbance or model misspecification, it is likely that $I^{-1}U$ might be updating the estimated $\boldsymbol{\beta}$ or $\widetilde{\boldsymbol{\beta}}$ in a wrong direction. This is particularly prone to happen in the case that the dimension of the parameters is large. Therefore, we would like to consider adaptive update such that the pace of update is reducing as the number of iterations increases. Specifically, we introduce a learning rate and adjust it as the iteration number grows in the following manner:

$$\begin{aligned} \boldsymbol{\beta}_i^{(t+1)} &= \boldsymbol{\beta}_i^{(t)} + \frac{lr}{t^{1/4}} \cdot I^{-1}_{\boldsymbol{\beta}_i^{(t)}} \cdot U_{\boldsymbol{\beta}_i^{(t)}}, \qquad t = 1, 2, \ldots, \\ \widetilde{\boldsymbol{\beta}}_j^{(t+1)} &= \widetilde{\boldsymbol{\beta}}_j^{(t)} + \frac{lr}{t^{1/4}} \cdot I^{-1}_{\widetilde{\boldsymbol{\beta}}_j^{(t)}} \cdot U_{\widetilde{\boldsymbol{\beta}}_j^{(t)}}, \qquad t = 1, 2, \ldots. \end{aligned} \tag{4.1.8}$$

where lr is a starting learning rate that could be set to $lr = 0.5$. As the iterations increase,

Algorithm 4 Alternating Tweedie regression

Require: $I^{-1}_{\boldsymbol{\beta}_i^{(t)}}$ and $I^{-1}_{\widetilde{\boldsymbol{\beta}}_j^{(t)}}$ exist.

$t \leftarrow 0$, $Loss^{(t)} \leftarrow 10^{20}$
$converged = False$
while $t \leq maxit$ **do**
  while $converged = False$ **do**
    $i \leftarrow 1$
    while $i \leq n$ **do**
      retrieve ith row of data from SQLite database.
      for `epoch ∈ 1,..., n_epoch` **do**
        compute $U_{\boldsymbol{\beta}_i^{(t)}}$ and $I_{\boldsymbol{\beta}_i^{(t)}}$.
        update $\boldsymbol{\beta}_i^{(t+1)}$ using current value of $\{\ \widetilde{\boldsymbol{\beta}}_j^{(t)}, j=1,\ldots,n\ \}$, and $\boldsymbol{\beta}_i^{(t)}$
        based on formulae in (4.1.6) or (4.1.8) or Adam update.
      end for

      for `epoch ∈ 1,..., n_epoch` **do**
        compute $U_{\widetilde{\boldsymbol{\beta}}_i^{(t)}}$ and $I_{\widetilde{\boldsymbol{\beta}}_i^{(t)}}$.
        update $\widetilde{\boldsymbol{\beta}}_i^{(t+1)}$ using current value of $\{\ \boldsymbol{\beta}_j^{(t)}, j=1,\ldots,n\ \}$, and $\widetilde{\boldsymbol{\beta}}_i^{(t)}$
        based on formulae in (4.1.6) or (4.1.8) or Adam update.
      end for
      $i \leftarrow i+1$
    end while

    for $k = 1,\ldots,n$ **do**
      retrieve kth row of data from SQLite database.
      recompute $U_{\boldsymbol{\beta}_k^{(t+1)}}$, $U_{\widetilde{\boldsymbol{\beta}}_k^{(t+1)}}$ and their L_2 norms using $\boldsymbol{\beta}^{(t+1)}$ and $\widetilde{\boldsymbol{\beta}}^{(t+1)}$ values.
      recompute $loss_k$ and $\widetilde{loss}_k$ using $\boldsymbol{\beta}^{(t+1)}$ and $\widetilde{\boldsymbol{\beta}}^{(t+1)}$ values.
    end for
    compute the $Loss(\boldsymbol{\beta}^{(t+1)}, \widetilde{\boldsymbol{\beta}}^{(t+1)})$.
    check if the Relative change in Loss (4.1.7) is less than ϵ.
  end while
  $t \leftarrow t+1$
end while

the effective learning rate becomes $\frac{lr}{t^{1/4}}$, which allows the algorithm to slowly approach the target in later iterations.

Our learning rate adjustment is just a step size change in the Fisher scoring algorithm. The *glm*2 package in R halves the step size when estimated model components become infinity or out of their range or the algorithm diverges. As the Fisher scoring algorithm belongs to a bigger category of gradient descent/ascent algorithm, we can look more into the bigger category. There are many variants of the gradient descent adjustments. See for example Ruder (2016) for review of different gradient descent algorithms such as Momentum, Adagrad, Adadelta, RMSprop, Adam etc. The Adam is the most popular update and becomes default parameter update method for many machine learning algorithms. With Adam, the direction of update is set to be the ratio of two terms: the exponentially decaying moving average of past gradients in the numerator, and the square root of exponentially decaying moving average of past squared gradients in the denominator. Adam combines the ideas of Momentum and and Adadelta. The moving averages of the gradient and squared gradient reflect the mean and variance of the gradient. The update is defined as follows:

$$g_{\mathrm{t,j}} = \nabla_\theta \ell\left(\theta_{\mathrm{t-1}}\right)_{\mathrm{j}}; \quad \mathrm{m}_{\mathrm{t,j}} = \mathrm{B}_1\ \mathrm{m}_{\mathrm{t-1,j}} + (1-\mathrm{B}_1)\, g_{\mathrm{t,j}}, \quad v_{\mathrm{t,j}} = \mathrm{B}_2 v_{\mathrm{t-1,j}} + (1-\mathrm{B}_2)\, g^2_{\mathrm{t,j}}$$
$$\mathrm{m}_{0j} = v_{0j} = 0; \quad \hat{\mathrm{m}}_{\mathrm{t,j}} = \frac{\mathrm{m}_{\mathrm{t,j}}}{1-\mathrm{B}_1^{\mathrm{t}}}; \quad \hat{v}_{\mathrm{t,j}} = \frac{v_{\mathrm{t,j}}}{1-\mathrm{B}_2^{t}}; \quad \theta_{\mathrm{t,j}} = \theta_{\mathrm{t-1,j}} - \frac{\eta}{\sqrt{\hat{v}_{\mathrm{t,j}} + \epsilon}} \hat{\mathrm{m}}_{\mathrm{t,j}}$$

The authors of this method recommended to set $B_1 = 0.9, B_2 = 0.999, \epsilon = 10^{-8}$. The second row of the equations makes corrections for total coefficients because the coefficient behind $m_{0,j}$ is B_1^t and $m_{0,j} = 0$, which lead to the sum of coefficients of non-zero terms being $1 - B_1^t$. The same justification holds for $v_{t,j}$. This update has a step size that takes the steepness into account, as in Adadelta, but also tends to move in the same direction, as in Momentum.

Most of the gradient descent variants do not use the second derivative of the objective function, which is because it is often difficult to compute the second derivative. We believe the use of adaptive learning rate together with the Fisher information matrix in our

algorithm provides more benefit than the variants of the gradient descent algorithms. This is because the variants of gradient descent algorithm either ignore the second derivative or uses squared first derivative to approximate the second derivative. On the other hand, the Fisher information provides the variance of the first derivative (i.e., Score function) of the objective function and the inverse of the Fisher information matrix provides the asymptotic variance for the parameters to be estimated according to likelihood inference (Cox and Hinkley 2017). The plots in Figure 4.1 shows how the three updating schemes differ when they are applied to a simulated dataset generated with alternating Tweedie regression model (see section 4.2.1). We could see from the left panel which shows the log(loss) for only one row of data with repeating update over 10 epochs, the Adam update is less effective in reducing the loss for each epoch compared to the update with or without learning rate adjustment. When the algorithm continuously updates for over 100 iterations, the log(overall loss) in the right panel also shows that the Adam update is not reducing the loss as fast as the other two updating schemes. Avoiding the usage of second derivatives of the objective function not only shows no advantage in computation time but also is less effective in reducing the loss. Deriving the second derivatives and implementing it in the Fisher scoring algorithm achieves better result faster.

Note that in the while loop for i in Algorithm 4, we update $\boldsymbol{\beta}_i$ and $\widetilde{\boldsymbol{\beta}}_i$ for the same i. The $\boldsymbol{\beta}_i$'s estimate depends on current value of $\widetilde{\boldsymbol{\beta}}$ which means recently updated value of $\widetilde{\boldsymbol{\beta}}_{j_1}, j_1 = 1, \ldots, i$ are used in the calculation. Due to such dependence, all parameter updates rely on each other. As a result, the updates of different parameters can only be done sequentially. In addition, it assumes that the data is symmetric so that the i^{th} row and i^{th} column are same. An advantage of this is that the i^{th} row of data is only loaded once. This saves some computation time. If the data matrix is not symmetric, then the algorithm needs to be adjusted by adding another for loop of j such that update for $\boldsymbol{\beta}_i, i = 1, \ldots, n$ is done consecutively followed by update of $\widetilde{\boldsymbol{\beta}}_j, j = 1, \ldots, n$. In this case, we need to read-in data for each row while updating the $\boldsymbol{\beta}_i$'s and then we will need to read-in data again for updating $\widetilde{\boldsymbol{\beta}}_j$.

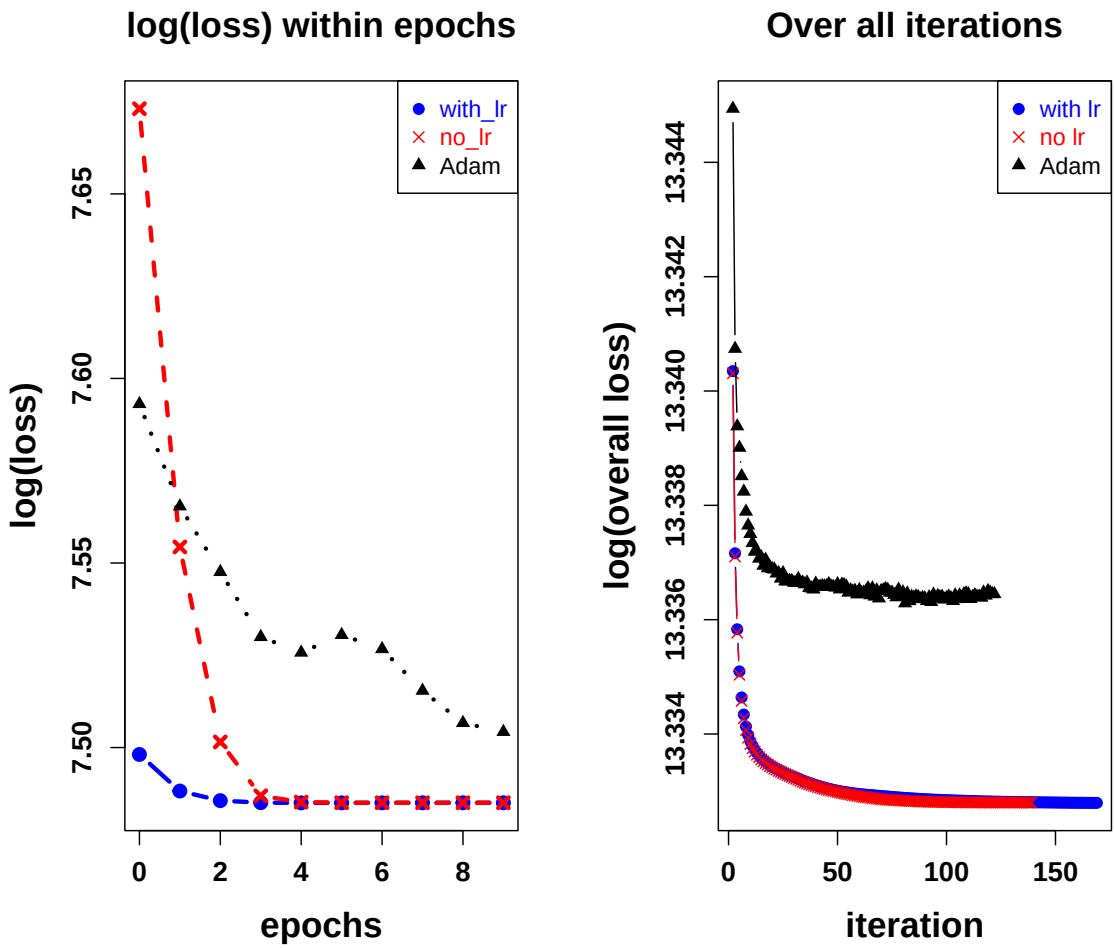

Figure 4.1: *Computed log(loss) and log(overall loss) from simulated dataset using the Fisher scoring with or without learning rate adjustment, and gradient descent algorithm with Adam method for parameter update. The left panel depicts how the loss changes over 10 epochs for one row of the parameter update. As epoch number grows, the loss has a general decreasing trend but the Adam's loss has higher values and reduces slower than the other two updates. The right panel is for overall loss versus number of iterations in* log *scale. All losses decrease as the iteration number increases but the Adam update has higher values of the overall loss.*

This potentially wastes some computational time. However, there is some potential benefit in doing so: successive update for different i's (or different j's) are not interdependent on each other. This makes parallel updating a possible solution. More specifically, when considering $\boldsymbol{\beta}_i$'s as the parameter while holding $\{\widetilde{\boldsymbol{\beta}}_j, \quad j = 1, \ldots, n\}$ as the data, the update for different i's is totally free from each other since the calculation of the Information matrix and the Score function only depends on $\{\widetilde{\boldsymbol{\beta}}_j, \quad j = 1, \ldots, n\}$.

Instead of writing our own code for parallel computing, it is tempting to use an existing parallel computing algorithm to conduct the update. Niu et al. (2011) introduced an update scheme called Hogwild which allows performing parameter update on CPUs in parallel. It works with multiple CPU processors by allowing them to access shared memory without locking the parameters. For this to work, the optimization problems have to be sparse in that each update only modifies a fraction of all parameters. Under this scenario, they showed that the Hogwild update scheme achieves almost an optimal rate of convergence because the possibility of the processors overwrite useful information is small. Our updating equations (4.1.8) only modify part of the model parameters as i or j points to specific row or column. However, that is not a sparse update. Even though we are working with only a block out of all model parameters at a time, there is no guarantee that the parameter update will be sparse. In fact, the parameters in our model are dense representation of words, items, or users, and they are not meant to be sparse. Additionally, our computations need to be carried out on GPUs. This point will be further explained in Chapter 5.

4.1.1 Impact of the parameters p and ϕ

The aforementioned formulae assume p_{ij} and ϕ_{ij} are known. When they are not known, estimation is needed. The estimation of the dispersion parameters p_{ij} and ϕ_{ij} is intractable when we use the likelihood approach.

$$\begin{aligned}
\frac{\partial \ell_i^{(1)}}{\partial \phi_{ij}} &= \sum_{j=1}^{n} -\frac{1}{\phi_{ij}^2}\left(y_{ij}\theta_{ij} - \kappa\left(\theta_{ij}\right)\right) + \frac{\partial c\left(y_{ij}, \phi_{ij}, p_{ij}\right)}{\partial \phi_{ij}}, \\
\frac{\partial \ell_i^{(1)}}{\partial p_{ij}} &= \sum_{j=1}^{n} \frac{1}{\phi_{ij}}\left[y_{ij}\frac{\partial \theta_{ij}}{\partial p_{ij}} + \frac{\partial \kappa\left(\theta_{ij}\right)}{\partial p_{ij}}\right] + \frac{\partial c\left(y_{ij}, \phi_{ij}, p_{ij}\right)}{\partial p_{ij}}
\end{aligned}$$

where $c\left(y, \phi, p\right) = \log\left(\sum_{k=1}^{\infty} \frac{\beta^{k\alpha}}{\Gamma(k\alpha)} \cdot y^{k\alpha-1} \cdot \frac{\lambda^k}{k!}\right)$. And ϕ and p are related in the following manner:

$$\lambda = \frac{\mu^{2-p}}{\phi(2-p)} \quad , \quad \alpha = \frac{2-p}{p-1} \quad , \quad \frac{1}{\beta} = \phi(p-1)\mu^{p-1}.$$

Due to excessive calculation involved in the infinite sum, the likelihood estimate of ϕ_{ij} and p_{ij} is intractable when they need to be estimated for all $i = 1, ..., n$ and $j = 1, ..., n$. Bonat and Kokonendji (2017) suggested to use profile likelihood to estimate the ϕ and p. In their small scale simulation studies, they find that the profile likelihood approach works well, particularly when true p is 0 or 2 in which case the likelihood approach fails to provide appropriate estimate with good coverage. However, the profile likelihood uses derivative free Nelder-Mead algorithm (Nelder and Mead 1965). This algorithm is really slow for large n.

For estimating power p from Tweedie model, traditional practice is to first train a Tweedie model with an arbitrary specified p and estimate the μ of all observations. The inverse weight and residuals from the model fit were then used to obtain an estimate of dispersion parameter (scale). Then the μ along with inverse weight, residuals, and scale estimate (ϕ) were used to estimate p. See the source code of function estimated_tweedie_power at https://www.statsmodels.org/devel/_modules/statsmodels/genmod/generalized_linear_model.html#GLM.estimate_tweedie_power. In our case, we have no covariates to fit the GLM model. The $\widetilde{\boldsymbol{\beta}}$ and $\boldsymbol{\beta}$ were alternately updated so we can not use them as the covariates to estimate the power p even when they were held fixed because the estimate of p will change with the estimate of $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$ and the loss functions correspond to different range of p are no longer comparable.

Instead, we will make use of the mean variance relationship $\text{Var}(Y) = \phi\mu^p$ for the Tweedie distribution. That is, $\log(\text{Var}(Y)) = \log\phi + p\log\mu$. Then we check the log of mean and log of variance of the observed response variable over different rows. The two appears to have piece-wise linear relationship. See Figure 4.2 for an example, which was generated for the January, 2022 Wikipedia dump with vocabulary size 50K. The majority of $\log\mu$ and $\log(\text{Var}(Y))$ pairs shows clear piecewise linear relationship when $\log\mu$ is between -4 and 2. For log mean smaller than -4 or greater than 1, there are limited number of data points. Based on such piecewise linear relationship, we regressed the log of sample variance against the log of sample mean of the response variable on each interval. The resulting slope and intercept estimates presented in Table 4.1 give us estimates of p and $\delta = \log(\phi)$ for each interval shown in Figure 4.2.

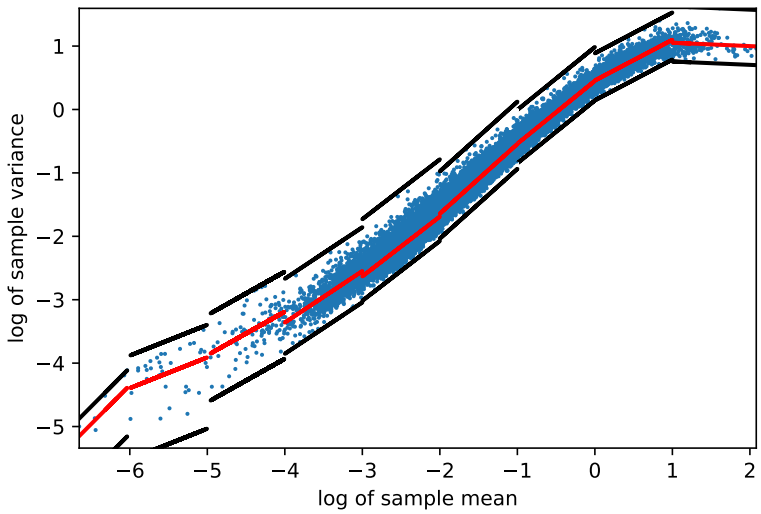

Figure 4.2: *Relationship between log of sample mean and log of sample variance from Wikipedia data with 50K vocabulary size. The three lines in each interval are the fitted linear regression line and upper and lower bounds with same slope.*

interval index	$\widehat{\delta}$	$\widehat{\delta}_{high}$	$\widehat{\delta}_{low}$	$\log(\mu)$ lower bound	$\log(\mu)$ upper bound	$\widehat{p}$
0	3.030	3.306	2.264	-7.000	-6.000	1.230
1	-1.483	-0.970	-2.607	-6.000	-5.000	0.485
2	-0.438	0.194	-1.180	-5.000	-4.000	0.688
3	-0.115	0.576	-0.607	-4.000	-3.000	0.811
4	0.197	1.099	-0.188	-3.000	-2.000	0.943
5	0.554	1.221	0.161	-2.000	-1.000	1.098
6	0.451	0.984	0.142	-1.000	0.000	0.990
7	0.457	0.887	0.145	0.000	1.000	0.642
8	1.105	1.680	0.809	1.000	2.582	-0.053

Table 4.1: *The estimated δ and p from fitting linear regression on log of sample mean and log of sample variance on each interval. The $\widehat{\delta}_{high}$ and $\widehat{\delta}_{low}$ are intercepts from the data points that are having maximum or minimum signed distance from the fitted regression line while having corresponding same slope.*

In Table 4.1, we have to carefully consider the estimated parameter $\hat{p}$. The range of p for Tweedie distribution to be well defined is $p \leq 0$, $1 \leq p < 2$, or $p \geq 2$, with $p = 0$ represents Gaussian distribution, $p = 1$ represents Poisson distribution, and $p = 2$ represents Gamma distribution. The interval 0 having $\hat{p}$ between 1 and 2 corresponds to Compound Poisson Gamma distribution. For the interval indices 4, 5, and 6, those could be considered as Poisson distribution having $p = 1$. The interval 8 could be viewed as Normal distribution with $p = 0$. On the other hand, the intervals 1, 2, 3, and 7 have $\hat{p}$ between 0 and 1 where Tweedie distribution is not well defined. Therefore, we might need to consider other distribution such as zero-inflated Tweedie distribution.

4.2 Application: Training new word representations with small Reuters news dataset

In this section, we present the application of the alternating Tweedie regression to a small dataset. The small dataset contains news articles downloaded from Reuters news archive https://www.reuters.com/news/archive/businessnews?view=page&page=5&pageSize=10. A Python code was written to imitate user clicks and around 2000 articles were downloaded. These articles were news between 2019 and summer 2021 in Business category. They are stored in a SQLite database. Using these 2000 articles as training corpus, we created a small vocabulary with top 300 most frequently used words.

Words are not easy to be directly used in artificial intelligence. Converting each one of them to a numerical vector will facilitate machine learning techniques to employ words as features. Here, we would like to illustrate with the small dataset of how to map a word to d dimensional vector. Such numerical vectors are referred as word representations. The training dataset is simply the small cooccurrence count matrix derived from the small corpus using the formula in (1.0.1).

We applied the alternating Tweedie regression described in Section 4.1.1 to this dataset. To apply the method, we start with assuming $Y_{ij} = \log(X_{ij} + 1)$ follows the Tweedie distribution with power parameter p_{ij} and dispersion parameter ϕ_{ij}. This is because the raw count X_{ij} has too much skewness to fit into the Tweedie distribution family. Additionally, some cooccurrence counts are very large and some are zero. With the convergence analysis, we know that the model fitting with large positive counts could have non-convergence problem if there is complete or quasi complete separation in the dense representations of words corresponds to the zero or nonzero count words. The histograms in Figure 4.3 show the skewness calculated for rows of the data matrix based on original cooccurrence count and log transformed cooccurrence count. The log transformed counts, even though still skewed, behave much more manageable than the raw count due to its smaller skewness. Due to above

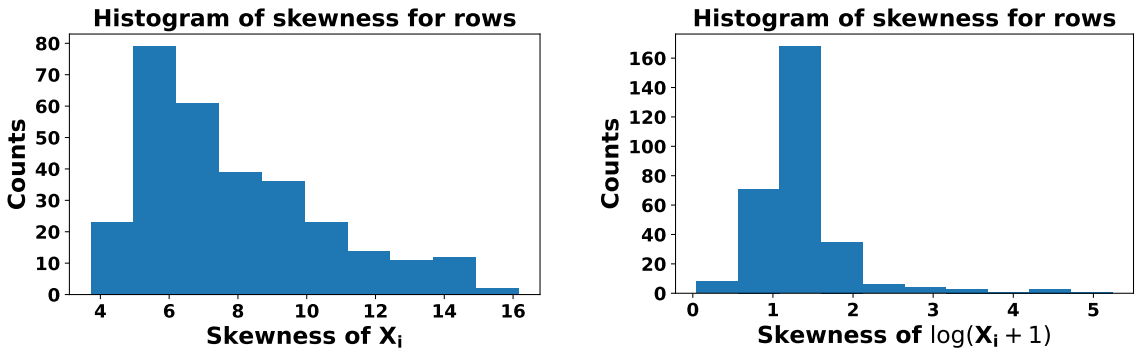

Figure 4.3: *The histograms of skewness for raw count (left panel) and log count (right panel) of rows in small Reuters dataset. The raw count shows large skewness that majority are around 6. The log count shows much less skewness.*

reason, we work with the log transformed count.

We initialized the model parameters as follows:

- The initial values for the elements of the word representations $\mathbf{w}_i$, $\widetilde{\boldsymbol{w}}_j$ and biases b_i, $\widetilde{b}_j$ were randomly generated from Uniform(-0.5, 0.5) independently.

- The dispersion parameter ϕ and the power parameter p were initialized based on the variance and mean relationship of the Tweedie random variable having format $\mathrm{Var}(Y) = \phi\mu^p$. From this relationship, we know $\log \mathrm{Var}(Y) = \delta + p \log \mu$. Therefore, $\log \mathrm{Var}(Y)$ and $\log \mu$ have linear relationship. Unfortunately, we do not have $\mathrm{Var}(Y)$ and μ and we can not estimate the mean and variance in the each element of the matrix because there is only single observation. To make the problem manageable, we compute the sample variance and sample mean using the cooccurrence counts in the same row. Then we would expect that the sample variance and sample mean pair across different rows provide the data for depicting the linear relationship. Therefore, we regress log of sample variance and log of sample mean to find the ϕ and p. The relationship appears to be piecewise linear (see Figure 4.4). Each piece of the linear regression is restricted to an interval for the log mean. The estimated δ and p in each interval are presented in Table 4.2. This is only a rough way of finding reasonable initial value of ϕ and p, denoted as $\hat{\phi}^{(0)}$ and $\hat{p}^{(0)}$. Due to symmetry nature of Y_{ij} and Y_{ji},

we need to set $\widehat{\text{Var}(Y_{ij})} = \hat{\phi}_i^{(0)} \cdot \hat{\phi}_j^{(0)} \hat{\mu}_{ij}^{\hat{p}_i^{(0)}+\hat{p}_j^{(0)}}$. This means we are assuming $p_{ij} = p_i + p_j$ and $\phi_{ij} = \phi_i \cdot \phi_j$. This would allow observations in the ith row and those in the ith column having same variance.

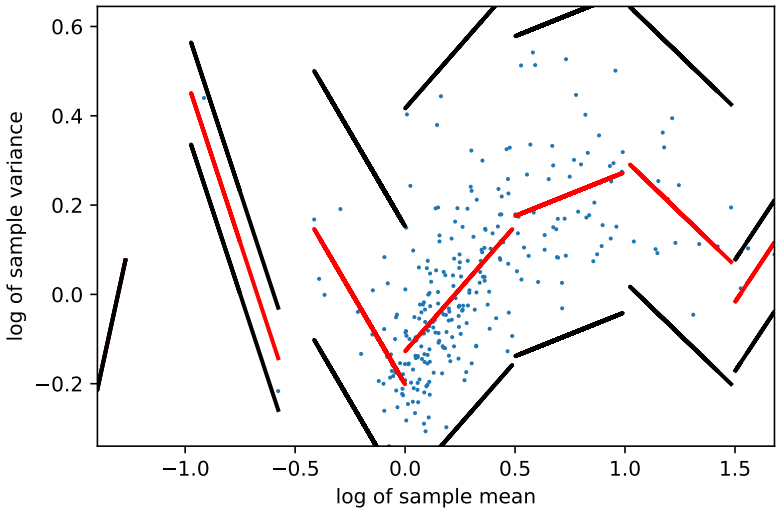

Figure 4.4: *Relationship between log of sample mean and log of sample variance from Reuters news small dataset with vocabulary size 300. The center line in each interval is the fitted piecewise linear regression line and upper and lower lines in each interval are lines having same slope as the fitted line but passing through the data point that is having maximum or minimum signed distance from the fitted line.*

We also included a different interval partition in Table 4.3. This table deviates from the previous one due to small mistake in converting the cooccurrence count from the sparse format to the matrix format. We would like to see the impact of different values of ϕ and p on the model parameter estimation by comparing results obtained with these two different tables.

interval index	$\widehat{\delta}$	$\widehat{\delta}_{high}$	$\widehat{\delta}_{low}$	$\log(\mu)$ lower bound	$\log(\mu)$ upper bound	$\widehat{p}$
0	2.955	2.955	2.955	-1.500	-1.000	2.264*
1	-1.009	-0.896	-1.125	-1.000	-0.500	-1.501
2	-0.202	0.152	-0.450	-0.500	0.000	-0.842
3	-0.127	0.416	-0.432	0.000	0.500	0.561
4	0.077	0.480	-0.237	0.500	1.000	0.197
5	0.776	1.129	0.503	1.000	1.500	-0.475
6	-1.133	-1.039	-1.288	1.500	2.180	0.744

Table 4.2: *The estimated δ and p from fitting piecewise linear regression on log of sample mean and log of sample variance on each interval from Reuters news small dataset. The $\widehat{\delta}_{high}$ and $\widehat{\delta}_{low}$ are intercepts from the data points that are having maximum or minimum signed distance from the fitted piecewise regression line while having corresponding same slope. Note: * indicates that the estimated value 2.264 is based on regression with only two observations which is highly unreliable and we could ignore the interval.*

interval index	$\widehat{\delta}$	$\widehat{\delta}_{high}$	$\widehat{\delta}_{low}$	$\log(\mu)$ lower bound	$\log(\mu)$ upper bound	$\widehat{p}$
0	0.190	0.473	-0.034	-0.500	0.000	1.373
1	-0.233	0.282	-0.541	0.000	0.500	0.871
2	0.088	0.504	-0.253	0.500	1.000	0.174
3	0.774	1.135	0.509	1.000	1.500	-0.480
4	-1.133	-1.038	-1.288	1.500	2.000	0.743

Table 4.3: *A table similar to Table 4.2 with numbers slightly wrong. This table was obtained when the sparse format of cooccurrence count was retrieved from the database with some slight mistake. We kept this table to see how the estimation in the algorithm is affected by this table.*

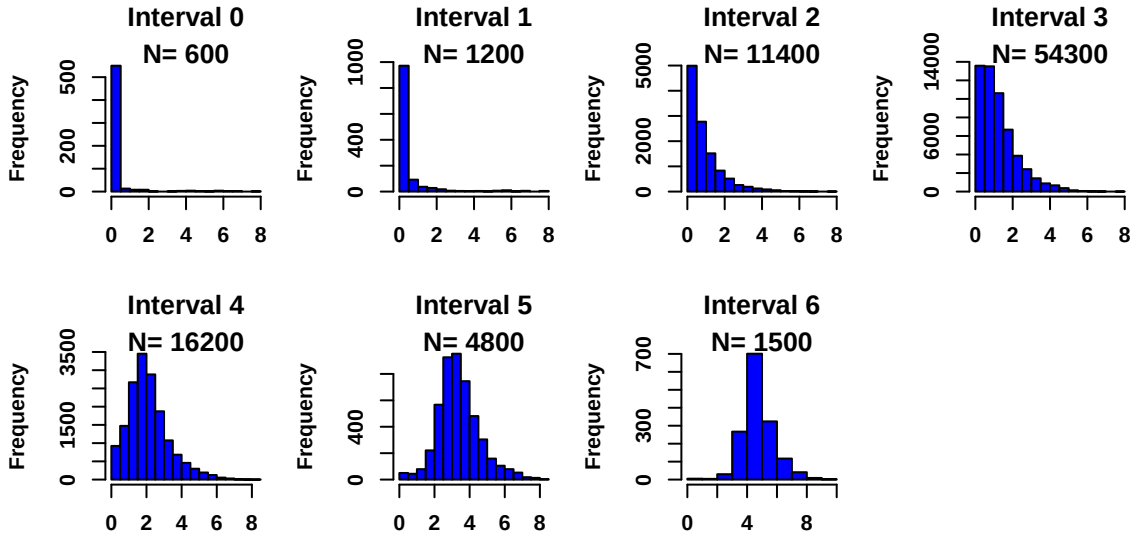

Figure 4.5: *The histograms of cooccurrence counts in each interval index from the Reuters news small dataset. The value N represents the number of cooccurrence count values in the interval.*

In Figure 4.5, we presented distribution of cooccurrence counts in each interval index from Reuters news small dataset. This figure was created by partitioning the cooccurrence counts in the entire matrix based on which interval the mean of log count in each row belongs to. All those rows belonging to the same interval were plotted in the same histogram. There are only two rows in the interval 0 and majority are zero counts (74.50%). Similarly, the interval 1 contains counts from four rows and still majority are zero (53.50%). The intervals 2, 3 and 4 contain majority of rows and the shape is highly positively skewed. There are still plenty of zeros in these intervals (13.56%, 6.08%, 1.50%, respectively), even though they do not have as much as the intervals 0 and 1. The intervals 5 and 6 have relatively symmetric shape with small portion of zero counts in interval 5 (0.19%), but none in the interval 6. Based on these plots, it seems to be reasonable to use Tweedie distribution to describe the cooccurrence counts.

With the p_{ij} and δ_{ij} given in Tables 4.2 and 4.3, we applied the Tweedie regression with learning rate adjustment and without learning rate adjustment. The change in the L_2 norm of the model parameters $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$ were recorded. The $\log_{10}$ change is summarized in Figure 4.6. Initially, all the $\log_{10}$ changes reduce quickly in early iterations regardless of learning rate adjustment was used or not and no matter the 'right' table or 'wrong' table of δ and p was used. In later iterations, these different updates showed different rate of deduction in the log of change in L_2 norm. The $\log_{10}$ change for the algorithm without learning rate adjustment seems to have higher values and converges slower than those from the algorithm with learning rate adjustment. The update with learning rate adjustment was not sensitive to using the 'right' table or 'wrong' table of δ and p. Most of the time, they provided similar results.

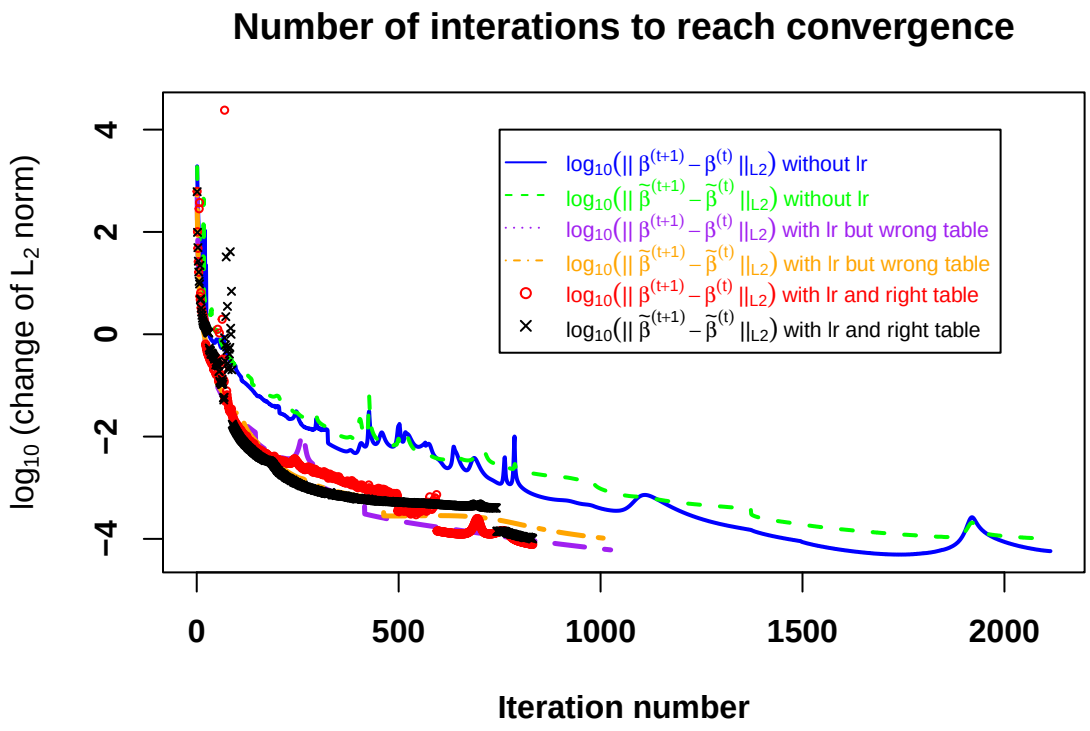

Figure 4.6: *The* $\log_{10}$ *change of* L_2 *norms in three different cases in model parameters* $\boldsymbol{\beta}$ *and* $\widetilde{\boldsymbol{\beta}}$ *versus number of iterations before the change in* $\boldsymbol{\beta}$*'s or* $\widetilde{\boldsymbol{\beta}}$*'s norm becomes negligible. The Solid line and dashed line are results of Algorithm 4 without learning rate adjustment. The dotted and dot-dashed line are using Algorithm 4 with adjustment of the learning rate but slightly wrong estimate of the parameters* δ *and* p*. The cases with circle and cross symbol are from Algorithm 4 with adjustment of the learning rate and using the right estimate of the parameters* δ *and* p *as provided in Table 4.2. Learning rate adjustment and using right table made the algorithm converge faster. Without learning rate adjustment, it takes more than twice number of iterations before the change becomes negligible.*

As we have seen, the impact of adjusting learning rate is not negligible. There is significant difference in number of iterations regarding with or without the learning rate adjustment in Algorithm 4 before the change in $\boldsymbol{\beta}$'s or $\widetilde{\boldsymbol{\beta}}$'s norm becomes negligible. The algorithm with the learning rate adjustment only took 832 iterations. However, without the learning rate adjustment, it took 2114 iterations to reach the same point which is almost two and half

times more iterations. We could see from Figure 4.6 that the case without the learning rate moves constantly in opposite directions around iteration number 400 to 900. On the other hand, with learning rate adjustment, there were significantly less number of movements toward opposite direction.

Comments about the alternating Tweedie MLE approach on the small Reuters data

Our derived formulae are based on Compound Poisson Gamma distribution which is special case of Tweedie distribution with $1 < p < 2$. This case and the case with $p = 1$, which corresponds to the Poisson distribution, are the only distributions in the Tweedie family that allows positive probability mass at zero. Unfortunately, not all of the data are suitable to be modeled with $1 \leq p < 2$. The p is the power that connects the variance function and the mean. The small Reuters data that we have tested have several rows that exhibit other distributional patterns. For example, in Figure 4.7, two rows of cooccurrence counts were plotted as histograms. Theses two rows have high proportion of zeros and highly skewed. The left panel of the plot has only 31.7% of values being non-zero and the right panel has 83% being non-zero. The estimated p corresponds to them from Table 4.2 is -1.50 and -0.83, respectively. Tweedie distributions with these negative powers have support of $\mathcal{R}$ and do not have positive probability at zero. Therefore, in such cases, Tweedie regression will not be suitable to model the two rows. Beyond the negative p cases, there are also rows of data that have p between 0 and 1, if the relationship of mean and variance from Tweedie distribution were to be used. Unfortunately, the distribution do no exist for such cases (see Proposition 4.2 on p.132 of Jorgensen 1997).

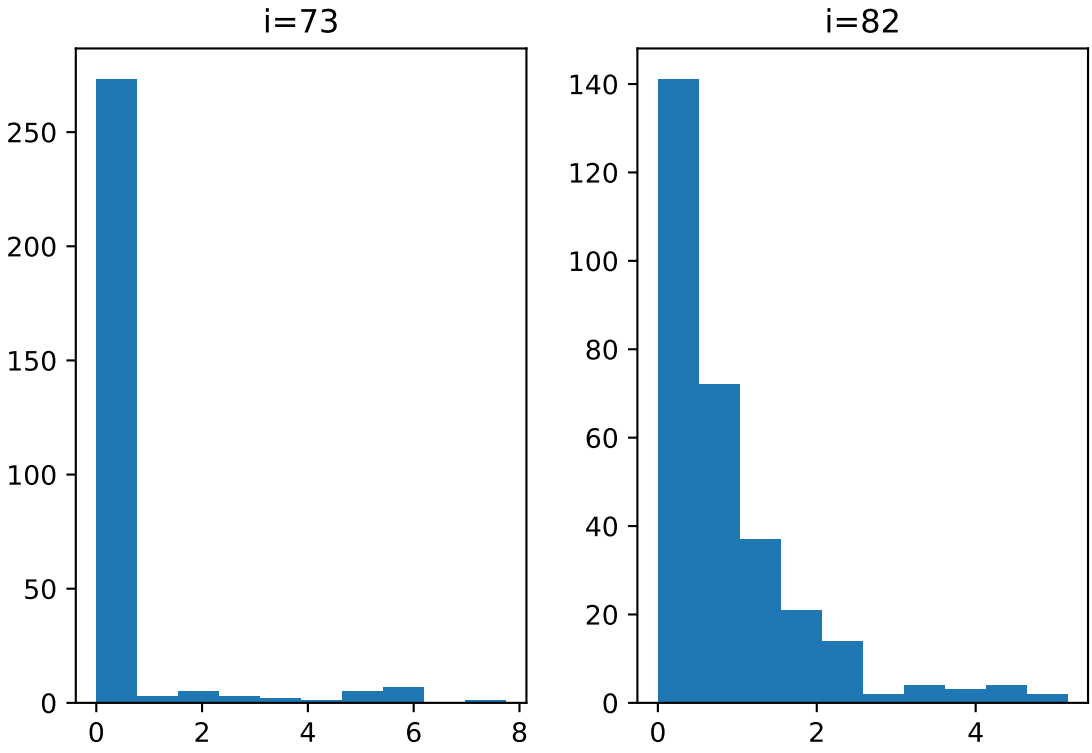

Figure 4.7: *The histograms of cooccurrence count in 73rd and 82nd row of the matrix from small Reuters news dataset. These two rows show typical cases that can not be modeled with the power p being between 1 and 2. Theses two cases have big proportion of zeros and have negative p (-1.50, -0.83, respectively), if Tweedie's mean and variance relationship were to be used. With such negative powers p, it is not possible to have probability mass at zero. The proportion of zeros in theses two rows are* 68.3% *and* 17%*, respectively.*

In our algorithm, for the cases with $p < 0$ or $0 < p < 1$, it could lead to the fitted mean count μ_{ij} being infinity if we force to fit Tweedie regression model. To avoid this from happening, we temporarily restrict the algorithm to bypass the update of parameter in such rows by setting an upper bound of L_2 norm of the Score vector to be no more than 10^{15}. This is just a temporary solution that the algorithm could proceed to estimate the rest of the model parameters. We still need to find alternative distribution models to fit such data. Similar strategy has been used in GloVe with different threshold (Pennington et al. 2014). Specifically, they restricted each component of the gradient to be within -100 and +100 in

the adaptive gradient descent algorithm. Any gradient outside of this range was forced to take the boundary value -100 or +100.

4.2.1 A small simulation study

In this section, we present a small simulation study using generated data from the Tweedie regression model. The goal of this study is to examine how fast the algorithms converge. Specifically, we would like to see how well the Fisher scoring algorithm with and without learning rate adjustment and the gradient descent algorithm with Adam update model parameters. The data generation is described below.

- We first retrieved the most frequently used 300 words in 2000 Reuters business news articles. For each word, a 50-dim vector ($\mathbf{w}$) needs to be generated for usage in the Tweedie regression model. The model components are as follows:

$$\eta_{ij} = \mathbf{w}_i^\top \cdot \widetilde{\boldsymbol{w}}_j, \quad \mu_{ij} = \exp \eta_{ij}$$

  where $\mathbf{w}_i = \mathbf{g}_i/|\mathbf{g}_i|$ and $\widetilde{\boldsymbol{w}}_j = \mathbf{g}_j/|\mathbf{g}_j|$ with $\mathbf{g}_i$ and $\mathbf{g}_j$ being the 50-dim word vector representations from pre-trained GloVe model for word$_i$ and word$_j$ respectively (https://nlp.stanford.edu/data/glove.6B.zip). Note that these $\mathbf{w}_i$'s can be arbitrarily generated from a distribution. Here, we use the GloVe word vectors so that the simulation is reproducible. Below are details:

- The Tweedie power parameter p_{ij} was generated as $p_i + p_j$ where p_i and p_j are independently sampled from Uniform(0.5, 1). With this generation mechanism, the power p_{ij} is between 1 and 2 so that the count data follows Compound Poisson Gamma distribution.

- The dispersion parameter ϕ_{ij} was generated as $\phi_i \cdot \phi_j$ where ϕ_i and ϕ_j were taken from the estimated value of the Reuters news dataset from Table 4.2.

- The count data $Y_{ij} = Y_{ji}$ were independently generated from Tweedie distribution with mean μ_{ij}, power parameter p_{ij}, and the dispersion parameter ϕ_{ij}.

We applied the alternating Tweedie regression algorithm to the generated data. For comparison, the gradient descent algorithm with Adam update was applied to the data. For the Adam update, we used the Adam optimizer (torch.optim.Adam) from the PyTorch package. A learning rate scheduler was used to reduce learning rate based on torch.optim.lr_scheduler.ReduceLROnPl The default setting of this function is used, which specifies the loss does not improve in 10 epochs, the learning rate is reduced by factor of 0.1. With the Adam update, computationally we only need to compute the loss and keep track of gradient of loss with respect to parameters being updated by setting .requires_grad_(True) for the parameter. The gradient was computed with the .backward() method of the loss tensor. The parameter update was done with the optimizer's .step() method. The optimizer's gradient was reset to zero within each epoch with the .zero_grad() method. For fair comparison, the innermost for loops of epoch are set to be 10 for all three update methods: with and without learning rate adjustment and Adam update.

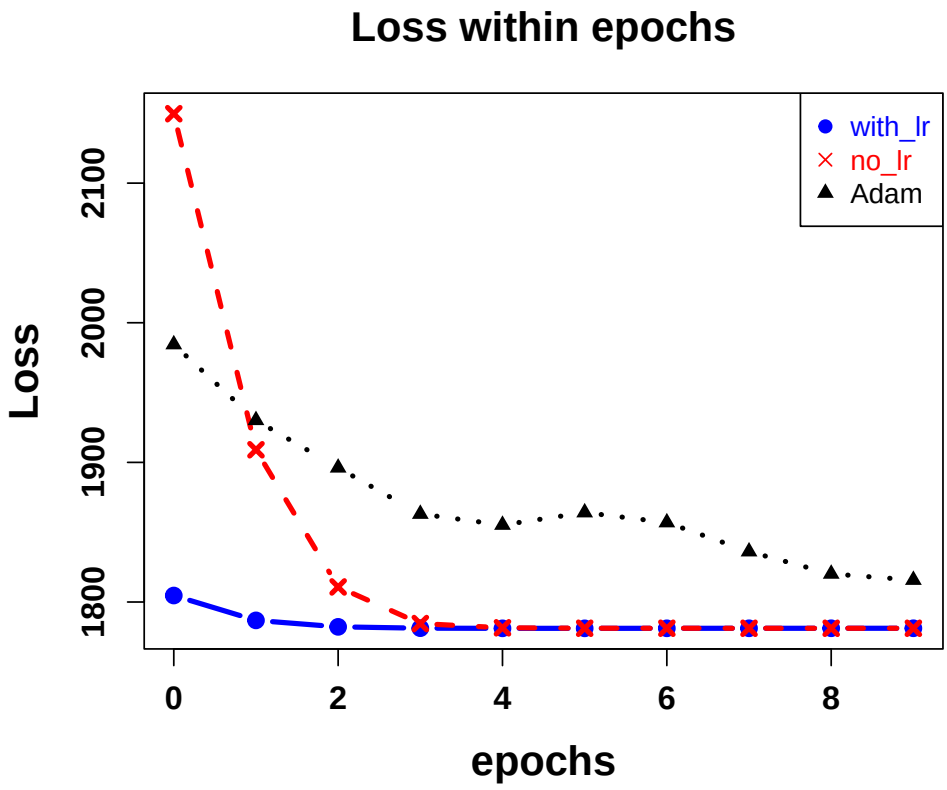

Figure 4.8: *The loss reduction was compared within epochs among three different updates: the alternating Tweedie regression algorithm with and without learning rate adjustment, and Adam update. The results are from the first iteration and first row of data matrix in our Algorithm 4.*

Figure 4.8 shows how different parameter update method performs within the innermost loop in our alternating Tweedie regression. The three different update methods are Fisher scoring type update with and without learning rate adjustment, and gradient descent Adam update. The loss reductions presented are from the first iteration and first row of data matrix in Algorithm 4. All three lead to quick reduction of the loss within 5 epochs. The updates with or without learning rate adjustment reach stable loss value quickly. On the other hand, the Adam update does not stabilize as fast as the other two methods. Further, note that the with or without learning rate adjustment updates' loss values are small compared to that of the Adam update. Taking into account the fact that all three updates used same simulated

data and started with identical initial values, the difference in the loss values are purely due to different updating schemes. Deriving the first and second derivatives manually and using the Fisher scoring algorithm makes a difference in finding right direction of the update. This point is even more obvious when we look at the overall loss over many iterations of updates in Figure 4.9. This figure contains the loss of all rows and columns of the data matrix from over 120 iterations. The Adam algorithm decreased the overall loss as the iteration increases but it is not as effective as the other two updating methods. Since the Adam update is inefficient in finding the right optimizing direction in the alternating Tweedie regression algorithm, we exclude it from further discussion and only compare the Fisher scoring type update with or without learning rate adjustment in more detail.

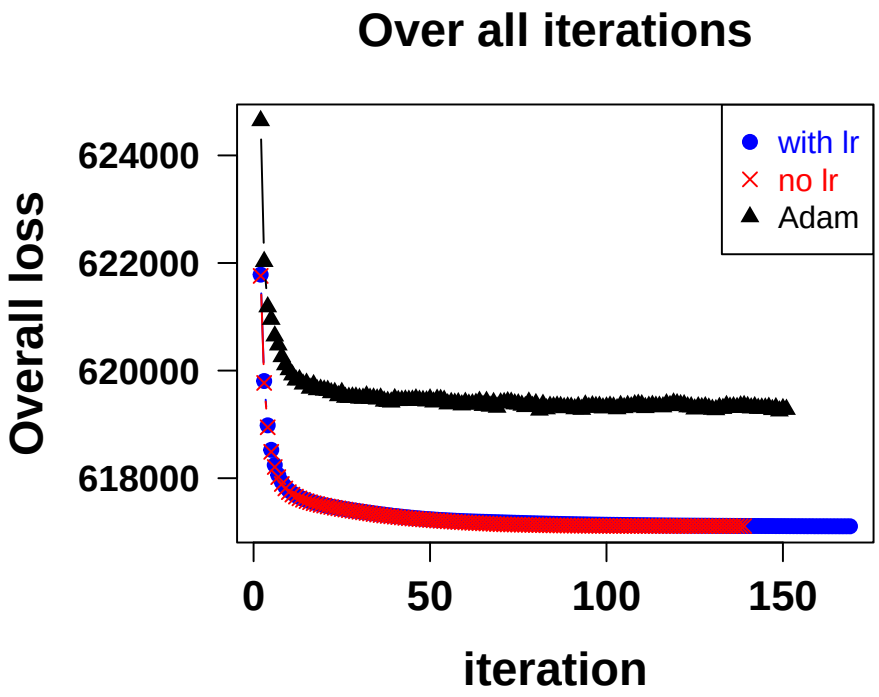

Figure 4.9: *The overall loss over iterations among three different update methods: with or without learning rate adjustment and the Adam update. The Fisher scoring type update with or without learning rate adjustment started with lower overall loss than the Adam update, and reduces the overall loss faster as the iteration number increases.*

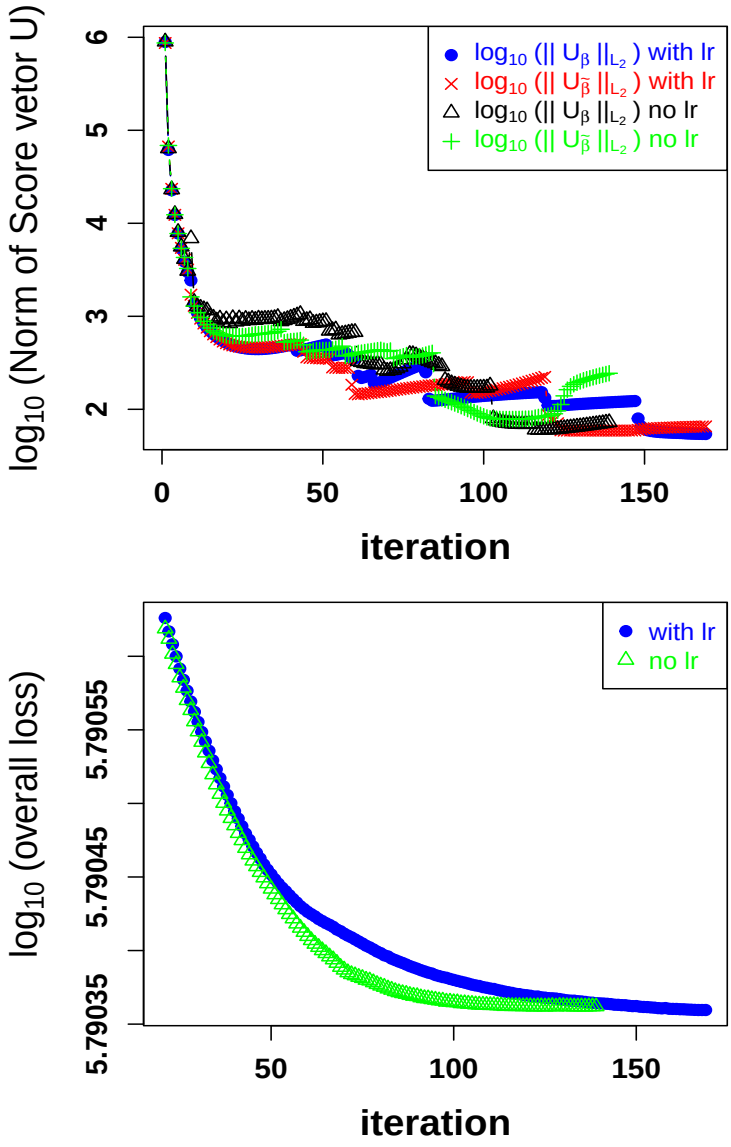

Figure 4.10: *The* $\log_{10}$ *scaled norm of the score vector and the overall loss as iteration proceeds from simulated data. The top panel shows norm of the score vector in* $\log_{10}$ *scale for two cases: with learning rate, and without learning rate. The bottom panel illustrates* $\log_{10}$ *overall loss versus iteration for the two cases. Overall, both cases are reducing the overall loss and the norm of score vector. The case with no learning rate is faster to reduce the overall loss in earlier iterations but may not achieve the minimum overall loss in the end. The case with learning rate moves slowly in earlier iterations but shows advantage in the end by finding smaller value in the overall loss.*

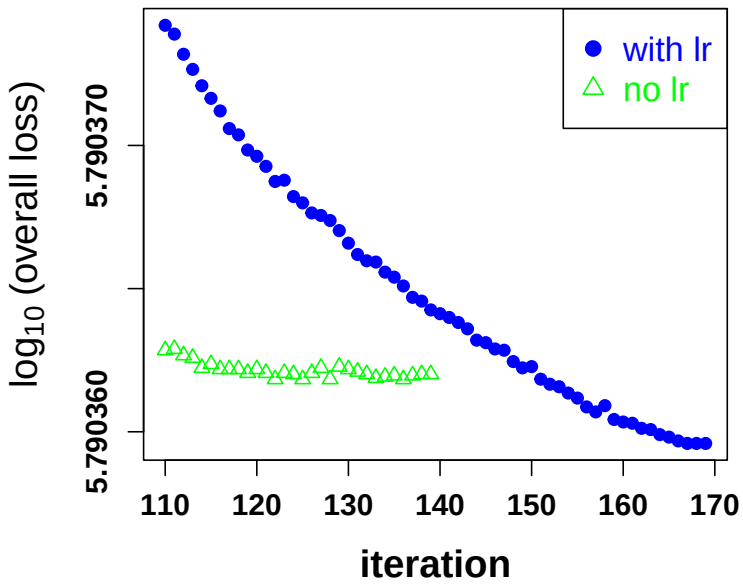

Figure 4.11: *The overall loss in* $\log_{10}$ *scale during iteration between 110 and 170 for the two cases: with learning rate, and without learning rate. The update without learning rate adjustment is stabilized in a certain value before reaching the minimum overall loss. The algorithm with learning rate adjustment continuously reduces the overall loss until satisfying the convergence criterion, even though it was less effective in earlier iterations compared to the one with no learning rate.*

Figures 4.10 and 4.11 show more detailed examination of the Fisher scoring update with or without learning rate adjustment. We can see from Figure 4.10, during the first 20 iterations, the two cases behave similarly in the norm of score vector. However, from 20th to around 130th iteration, the update with no learning rate is more effective in reducing the overall loss than the one with the learning rate. This is not the case for the norm of the score vector. From 20th to around 80th iteration, update with no learning rate is a little less effective in reducing the norm of the score vector than the update with learning rate. However, from 80th to 120th iteration, the case with no learning rate is more aggressive than the one with learning rate. After 120th iteration until convergence, the no learning rate case starts to sacrifice $U_{\widetilde{\boldsymbol{\beta}}}$ severely to improve $U_{\boldsymbol{\beta}}$ while maintaining the overall loss in the same neighborhood. On the other hand, small updates with learning rate adjustment help to make the algorithm achieve lower overall loss. Figure 4.11 shows how the log loss changes in the last 60 iterations. The one with no learning rate fluctuates a little bit around the same value 617109.2 and was trapped there causing the algorithm to believe the convergence criterion is satisfied. The one with learning rate consistently reduces the overall loss and achieves smaller overall loss 617105.8 when the convergence criterion is satisfied. To verify this pattern is not happened by chance, we generated 8 more datasets and applied the algorithm to them. The results are summarized in Figure 4.12. Basically, the behavior of the algorithm is same as we illustrated above.

The first column of Figure 4.12 shows the L_2 norm of the score vector $U_{\boldsymbol{\beta}}$ in $\log_{10}$ scale over all iterations for both with and without learning rate. Both cases perform well during the first 20 iterations by reducing the L_2 norm of the score vector $U_{\boldsymbol{\beta}}$ quickly. It is a little surprising that the one with learning rate adjustment was also able to achieve this because with learning rate adjustment $\frac{lr}{it^{1/4}}$, the updated amount in $\hat{\boldsymbol{\beta}}$ only gets between a half and a quarter of the amount received from the update without learning rate adjustment. During iterations 20 to 200, the algorithm with learning rate adjustment updates the $\hat{\boldsymbol{\beta}}$ by a much smaller fraction (23.6% to 13.3%) of the amount updated using the algorithm

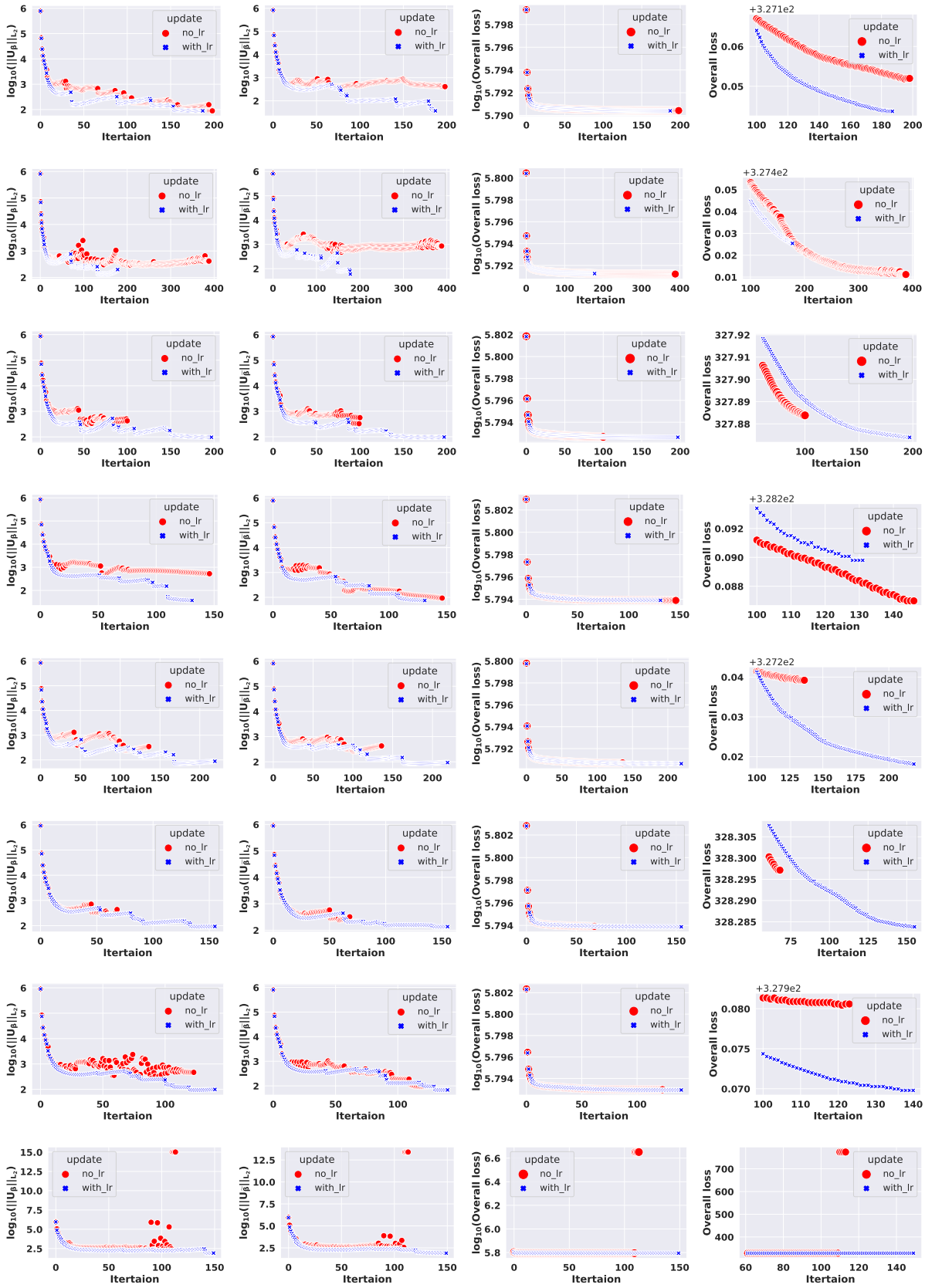

Figure 4.12: *Comparing performance of the alternating Tweedie regression algorithm with or without learning rate over 8 simulated datasets. Each row is for one dataset.*

with no learning rate adjustment. As a result, the algorithm with learning rate adjustment cautiously stays around the estimated value from previous iteration and approaches the target little by little. The result of this is that reducing the learning rate at later iterations optimizes the objective function more efficiently and consistently compared to the algorithm without learning rate adjustment. We see the norm of $U_{\boldsymbol{\beta}}$ achieves lower value faster in the case with learning rate adjustment than the other case. Further, there is less fluctuations when learning rate adjustment is used. This means the algorithm without learning rate adjustment often over-corrects its estimate for the model parameters in later iterations. The second column illustrates the L_2 norm of score vector $U_{\widetilde{\boldsymbol{\beta}}}$ in $\log_{10}$ scale over all iterations for both with and without learning rate. For the $U_{\widetilde{\boldsymbol{\beta}}}$ part, very similar pattern was observed as the first column.

The third column of Figure 4.12 shows overall loss in $\log_{10}$ over all iterations. Overall trend is both reduces the loss quickly in early iterations and gradually slow down toward the optimal loss. It seems like both with and without learning rate performs similarly well. However, if we look in more detail, we could observe some difference as shown in the last column of the figure. Compare to the reduction in norm of $U_{\boldsymbol{\beta}}$ and $U_{\widetilde{\boldsymbol{\beta}}}$, the loss function reduces consistently without much fluctuations. This aspect shows that the loss is not a monotone function with regard to $U_{\boldsymbol{\beta}}$'s and $U_{\widetilde{\boldsymbol{\beta}}}$'s norms and there are possibly many different solutions giving similar loss value.

The last column of Figure 4.12 is the zoomed in detailed performance near the end (i.e. near convergence) of the algorithm for 8 simulated datasets. The two updates, with or without learning rate, seemed to have different curves but both achieved similar final loss value in all except for one dataset in the sense that they only differ at the second digit after the decimal point. An exception happened in the last row of Figure 4.11 for the overall loss: the update with no learning rate moved out of the region around the optimal solution leading to a sudden drastic increase in the loss function and couldn't get out of there. Upon convergence, the algorithm stayed there at the much higher final loss value.

In summary, we compared performance of the alternating Tweedie regression algorithm among three different update methods. The Adam update performed well in the sense that it reduces the loss both within the innermost loop over epochs and throughout the iterations in the outer loop. However, when compared to the Fisher scoring update with or without learning rate adjustment, the Adam is inefficient since its overall loss or loss within each epoch reduces slower than the other two updating schemes. The Fisher scoring update without learning rate adjustment optimizes the algorithm in the right direction but sometimes fails to detect the right direction as we have seen in multiple simulated datasets. The Fisher scoring update with learning rate adjustment performed the best because it consistently find the right direction to reduce the overall loss and reach the smallest loss value among these three update methods. In the end of all iterations, the difference of overall loss between with and without learning rate adjustment is negligible compared to its magnitude.

4.3 Gaussian pseudo-likelihood for alternating Tweedie regression

In previous section, we studied the alternating Tweedie regression likelihood approach while treating p_{ij} and δ_{ij} fixed. Even though our numerical study in the Reuters small dataset suggests that the choice of p and δ do not make significant difference in the estimation of the regression parameters, it is likely that our 'wrong' table of p and δ is reasonably close to the 'right' table. In this section, we explore the pseudo-likelihood approach because Bonat and Kokonendji (2017) presented numerical studies showing better performance of pseudo-likelihood and quasi-likelihood than maximum likelihood approach when p is around 0 or 1. Bonat and Kokonendji (2017) also found in their simulation study that the pseudo-likelihood approach and the quasi-likelihood approach have similar efficiency when the data have small variations but the pseudo-likelihood is more efficient when the variation is large. We would like to derive the pseudo-likelihood approach for our alternating Tweedie regression and

assess its performance. We will consider two forms of the pseudo-likelihood: one is to use Gaussian distribution for all observations regardless of zero or positive case, and the other one is to use Poisson distribution for zero observations as in Compound Poisson Gamma but use Gaussian distribution for positive observations. This effort is to help the model adapt large proportion of zeros.

Throughout this section, we denote $\delta_{ij} = \delta_i + \tilde{\delta}_j$, $p_{ij} = p_i + \tilde{p}_j$, and $\phi_{ij} = \exp(\delta_{ij})$.

4.3.1 Gaussian pseudo-likelihood for all observations

The Gaussian pseudo-log likelihood from (i, j) element of the cooccurrence matrix as

$$\log f(y_{ij}) = -\frac{1}{2}\log(2\pi) - \frac{1}{2}\delta_{ij} - \frac{1}{2}p_{ij} \cdot \log \mu_{ij} - \frac{1}{2}\frac{(y_{ij} - \mu_{ij})^2}{\exp(\delta_{ij})\, \mu_{ij}^{p_{ij}}}, \tag{4.3.1}$$

$i, j = 1, \cdots, n$. Note that this pseudo-likelihood formula (4.3.1) follows Bonat and Kokonendji (2017). When a zero observation is observed, the pseudo-log likelihood contribution from this zero observation is

$$\log f_0 = -\frac{1}{2}\log(2\pi) - \frac{1}{2}\delta_{ij} - \frac{1}{2}p_{ij} \cdot \log \mu_{ij} - \frac{1}{2}\frac{\mu_{ij}^{2-p_{ij}}}{\exp(\delta_{ij})}.$$

This log likelihood value may be very different from the Tweedie log likelihood for zero observation, $\mu_{ij}^{2-p_{ij}}/\exp(\delta_{ij})/(2-p_{ij})$. If there are not many zero observations, this could be fine. However, if there are a lot of zero observations, we see potential problem could occur because of this discrepancy.

In our alternating regression, we have parameters $\boldsymbol{\theta}_i = \left(\mathbf{w}_i^\top, b_i, \delta_i, p_i\right)^\top$, $\widetilde{\boldsymbol{\theta}}_j = \left(\widetilde{\boldsymbol{w}}_j^\top, \widetilde{b}_j, \widetilde{\delta}_j, \widetilde{p}_j\right)^\top$. When we estimate $\boldsymbol{\theta}_i$ while holding $\widetilde{\boldsymbol{\theta}}_j$ fixed, we denote the log likelihood of $(i, j)^{th}$ observation in the data matrix as $\ell_{ij}(\boldsymbol{\theta}_i; \widetilde{\boldsymbol{\theta}}_j)$. On the other hand, when we estimate $\widetilde{\boldsymbol{\theta}}_j$ while holding $\boldsymbol{\theta}_i$ fixed, we denote the log likelihood of $(i, j)^{th}$ observation in the data matrix as $\widetilde{\ell}_{ij}(\widetilde{\boldsymbol{\theta}}_j; \boldsymbol{\theta}_i)$. Both of them refer to same quantity in the right hand side of equation (4.3.1). Different notation here is to emphasize that the function is treated as different functions of some of

parameters.

The parameters and the mean μ are connected with the following structure:

$$\mu_{ij} = \exp(\eta_{ij}), \quad \text{where} \quad \eta_{ij} = \log \mu_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j, \quad \mathbf{w}_i, \widetilde{\boldsymbol{w}}_j \in R^d.$$

This implies

$$\frac{\partial \mu_{ij}}{\partial \eta_{ij}} = \mu_{ij}, \quad \frac{\partial \eta_{ij}}{\partial \mathbf{w}_i} = \widetilde{\boldsymbol{w}}_j, \quad \frac{\partial \mu_{ij}}{\partial \mathbf{w}_i} = \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \frac{\partial \eta_{ij}}{\partial \mathbf{w}_i} = \mu_{ij} \widetilde{\boldsymbol{w}}_j.$$

The total pseudo-log likelihood from all rows of the data matrix is

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f(y_{ij}) = -\frac{n^2}{2}\log(2\pi) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\delta_{ij} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} \cdot \log \mu_{ij} - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{(y_{ij}-\mu_{ij})^2}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}}}.$$

When estimating $\boldsymbol{\theta}$, the total pseudo-log likelihood is denoted as $L^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})$. Similarly, when estimating $\widetilde{\boldsymbol{\theta}}$, the total pseudo-log likelihood is denoted as $\widetilde{L}^P(\widetilde{\boldsymbol{\theta}}; \boldsymbol{\theta})$. The first derivatives of $L^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})$ are obtained as below.

$$\begin{aligned}
\frac{\partial L^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i} &= -\frac{1}{2}\sum_{j=1}^{n} p_{ij}\widetilde{\boldsymbol{w}}_j + \frac{\partial}{\partial \mathbf{w}_i}\left[-\frac{1}{2}\sum_{i}^{n}\sum_{j}^{n}\frac{(y_{ij}-\mu_{ij})^2}{\exp(\delta_{ij})\mu_{ij}^{p_{ij}}}\right] \\
&= -\frac{1}{2}\sum_{j=1}^{n} p_{ij}\widetilde{\boldsymbol{w}}_j - \frac{1}{2}\sum_{j=1}^{n}\left\{\frac{\partial}{\partial \mu_{ij}}\left[\frac{(y_{ij}-\mu_{ij})^2\,\mu_{ij}^{-p_{ij}}}{\exp(\delta_{ij})}\right]\cdot\frac{\partial \mu_{ij}}{\partial \mathbf{w}_i}\right\} \\
&= -\frac{1}{2}\sum_{j=1}^{n} p_{ij}\widetilde{\boldsymbol{w}}_j - \frac{1}{2}\sum_{j=1}^{n}\left\{\left[\frac{-2\,(y_{ij}-\mu_{ij})}{\exp(\delta_{ij})\mu_{ij}^{p_{ij}}} - \frac{p_{ij}\,(y_{ij}-\mu_{ij})^2}{\exp(\delta_{ij})\mu_{ij}^{p+1}}\right]\mu_{ij}\widetilde{\boldsymbol{w}}_j\right\} \\
&= -\frac{1}{2}\sum_{j=1}^{n} p_{ij}\widetilde{\boldsymbol{w}}_j + \sum_{j=1}^{n}\frac{(y_{ij}-\mu_{ij})\,\widetilde{\boldsymbol{w}}_j}{\exp(\delta_{ij})\mu_{ij}^{p_{ij}-1}} + \frac{1}{2}\sum_{j=1}^{n}\frac{p_{ij}\,(y_{ij}-\mu_{ij})^2}{\exp(\delta_{ij})\mu_{ij}^{p_{ij}}}\widetilde{\boldsymbol{w}}_j.
\end{aligned}$$

Let

$$f_{ij} = \frac{(y_{ij}-\mu_{ij})}{\exp(\delta_{ij})\mu_{ij}^{p_{ij}-1}}, \quad g_{ij} = \frac{(y_{ij}-\mu_{ij})^2}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}}}. \tag{4.3.2}$$

The first order partial derivatives for $\boldsymbol{\theta}_i$ part can be written as

$$\begin{aligned}
\frac{\partial L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i} &= -\frac{1}{2}\sum_{j=1}^{n} p_{ij}\cdot \mathbf{w}_j + \sum_{j=1}^{n} f_{ij}\cdot \widetilde{\boldsymbol{w}}_j + \frac{1}{2}\sum_{j=1}^{n} p_{ij}\cdot g_{ij}\cdot \widetilde{\boldsymbol{w}}_j,\\
\frac{\partial L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i} &= -\frac{1}{2}\sum_{j=1}^{n} p_{ij} + \sum_{j=1}^{n} f_{ij}\cdot 1 + \frac{1}{2}\sum_{j=1}^{n} p_{ij}\cdot g_{ij}\cdot 1,\\
\frac{\partial L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \delta_i} &= -\frac{n}{2} + \frac{1}{2}\sum_{j=1}^{n} g_{ij},\\
\frac{\partial L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial p_i} &= -\frac{1}{2}\sum_{j=1}^{n} \log(\mu_{ij}) + \frac{1}{2}\sum_{j=1}^{n} \log(\mu_{ij})\cdot g_{ij}.
\end{aligned}$$

For $\widetilde{\boldsymbol{\theta}}_j$ part,

$$\begin{aligned}
\frac{\partial \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\boldsymbol{w}}_j} &= -\frac{1}{2}\sum_{i=1}^{n} p_{ij}\cdot \mathbf{w}_i + \sum_{i=1}^{n} f_{ij}\cdot \mathbf{w}_i + \frac{1}{2}\sum_{i=1}^{n} p_{ij}\cdot g_{ij}\cdot \mathbf{w}_i,\\
\frac{\partial \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{b}_j} &= -\frac{1}{2}\sum_{i=1}^{n} p_{ij} + \sum_{i=1}^{n} f_{ij}\cdot 1 + \frac{1}{2}\sum_{i=1}^{n} p_{ij}\cdot g_{ij}\cdot 1,\\
\frac{\partial \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\delta}_j} &= -\frac{n}{2} + \frac{1}{2}\sum_{i=1}^{n} g_{ij},\\
\frac{\partial \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{p}_j} &= -\frac{1}{2}\sum_{i=1}^{n} \log(\mu_{ij}) + \frac{1}{2}\sum_{i=1}^{n} \log(\mu_{ij})\cdot g_{ij}.
\end{aligned}$$

Now, compute the second order partial derivatives for $S_{\boldsymbol{\theta}_i}$ and $S_{\widetilde{\boldsymbol{\theta}}_j}$ where

$$S_{\boldsymbol{\theta}_i} = \begin{bmatrix} S_{\mathbf{w}_i} & S_{\mathbf{w}_i b_i} & S_{\mathbf{w}_i \delta_i} & S_{\mathbf{w}_i p_i} \\ S_{b_i \mathbf{w}_i} & S_{b_i} & S_{b_i \delta_i} & S_{b_i p_i} \\ S_{\delta_i \mathbf{w}_i} & S_{\delta_i b_i} & S_{\delta_i} & S_{\delta_i p_i} \\ S_{p_i \mathbf{w}_i} & S_{p_i b_i} & S_{p_i \delta_i} & S_{p_i} \end{bmatrix}, \quad S_{\widetilde{\boldsymbol{\theta}}_j} = \begin{bmatrix} S_{\widetilde{\boldsymbol{w}}_j} & S_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} & S_{\widetilde{\boldsymbol{w}}_j \widetilde{\delta}_j} & S_{\widetilde{\boldsymbol{w}}_j \widetilde{p}_j} \\ S_{\widetilde{b}_j \widetilde{\boldsymbol{w}}_j} & S_{\widetilde{b}_j} & S_{\widetilde{b}_j \widetilde{\delta}_j} & S_{\widetilde{b}_j \widetilde{p}_j} \\ S_{\widetilde{\delta}_j \widetilde{\boldsymbol{w}}_j} & S_{\widetilde{\delta}_j \widetilde{b}_j} & S_{\widetilde{\delta}_j} & S_{\widetilde{\delta}_j \widetilde{p}_j} \\ S_{\widetilde{p}_j \widetilde{\boldsymbol{w}}_j} & S_{\widetilde{p}_j \widetilde{b}_j} & S_{\widetilde{p}_j \widetilde{\delta}_j} & S_{\widetilde{p}_j} \end{bmatrix}. \quad (4.3.3)$$

The components of $S_{\boldsymbol{\theta}_i}$ are given below

$$
\begin{aligned}
S_{\mathbf{w}_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\right] = \sum_{j=1}^{n}\left\{\frac{\widetilde{\boldsymbol{w}}_j \cdot \widetilde{\boldsymbol{w}}_j^\top}{\exp(\delta_{ij})\cdot \mu_{ij}^{p_{ij}-2}} + \frac{\widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top \cdot p_{ij}^2}{2}\right\},\\
S_{b_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i^2}\right] = \sum_{j=1}^{n}\left\{\frac{1\cdot 1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{1\cdot 1\cdot p_{ij}^2}{2}\right\},\\
S_{\delta_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \delta_i^2}\right] = \frac{n}{2},\\
S_{p_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial p_i^2}\right] = \frac{1}{2}\sum_{j=1}^{n}\left(\log \mu_{ij}\right)^2,
\end{aligned}
$$

$$
\begin{aligned}
S_{\mathbf{w}_i b_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial b_i}\right] = \sum_{j=1}^{n}\left\{\frac{1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{p_{ij}^2}{2}\right\}\cdot \widetilde{\boldsymbol{w}}_j \cdot 1,\\
S_{\mathbf{w}_i \delta_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial \delta_i}\right] = \frac{1}{2}\sum_{j=1}^{n} p_{ij}\cdot \widetilde{\boldsymbol{w}}_j,\\
S_{\mathbf{w}_i p_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial p_i}\right] = \frac{1}{2}\sum_{j=1}^{n} p_{ij}\cdot \widetilde{\boldsymbol{w}}_j \cdot \log(\mu_{ij}),\\
S_{b_i \delta_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i \partial \delta_i}\right] = \frac{1}{2}\sum_{j=1}^{n} p_{ij},\\
S_{b_i p_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i \partial p_i}\right] = \frac{1}{2}\sum_{j=1}^{n} p_{ij}\log(\mu_{ij}),\\
S_{\delta_i p_i} &= E\left[-\frac{\partial^2 L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \delta_i \partial p_i}\right] = \frac{1}{2}\sum_{j=1}^{n} \log(\mu_{ij}),
\end{aligned}
$$

Similarly, the components of $S_{\widetilde{\boldsymbol{\theta}}_j}$ are given below

$$
\begin{aligned}
S_{\widetilde{\boldsymbol{w}}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\boldsymbol{w}}_j^\top}\right] = \sum_{i=1}^{n}\left\{\frac{\mathbf{w}_i \cdot \mathbf{w}_i^\top}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{\mathbf{w}_i \cdot \mathbf{w}_i^\top \cdot p_{ij}^2}{2}\right\},\\
S_{\widetilde{b}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{b}_j^2}\right] = \sum_{i=1}^{n}\left\{\frac{1\cdot 1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{1\cdot 1\cdot p_{ij}^2}{2}\right\},\\
S_{\widetilde{\delta}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\delta}_j^2}\right] = \frac{n}{2},\\
S_{\widetilde{p}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{p}_j^2}\right] = \frac{1}{2}\sum_{i=1}^{n}\left(\log \mu_{ij}\right)^2,
\end{aligned}
$$

$$
\begin{aligned}
S_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{b}_j}\right] = \sum_{i=1}^{n}\left\{\frac{1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{p_{ij}^2}{2}\right\}\cdot \mathbf{w}_i \cdot 1,\\
S_{\widetilde{\boldsymbol{w}}_j \widetilde{\delta}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{\delta}_j}\right] = \frac{1}{2}\sum_{i=1}^{n} p_{ij}\cdot \mathbf{w}_i,\\
S_{\widetilde{\boldsymbol{w}}_j \widetilde{p}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\boldsymbol{w}}_j \partial \widetilde{p}_j}\right] = \frac{1}{2}\sum_{i=1}^{n} p_{ij}\cdot \mathbf{w}_i \cdot \log(\mu_{ij}),\\
S_{\widetilde{b}_j \widetilde{\delta}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{b}_j \partial \widetilde{\delta}_j}\right] = \frac{1}{2}\sum_{i=1}^{n} p_{ij},\\
S_{\widetilde{b}_j \widetilde{p}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{b}_j \partial \widetilde{p}_j}\right] = \frac{1}{2}\sum_{i=1}^{n} p_{ij}\cdot \log(\mu_{ij}),\\
S_{\widetilde{\delta}_j \widetilde{p}_j} &= E\left[-\frac{\partial^2 \widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\delta}_j \partial \widetilde{p}_j}\right] = \frac{1}{2}\sum_{i=1}^{n} \log(\mu_{ij}).
\end{aligned}
$$

Therefore, our updating formula using Fisher scoring algorithm without learning rate ad-

justment is

$$\begin{aligned}\boldsymbol{\theta}_i^{(t+1)} &= \boldsymbol{\theta}_i^{(t)} + S^{-1}_{\boldsymbol{\theta}_i^{(t)}} \cdot U_{\boldsymbol{\theta}_i^{(t)}}, \quad i = 1, \cdots, n, \\ \widetilde{\boldsymbol{\theta}}_j^{(t+1)} &= \widetilde{\boldsymbol{\theta}}_j^{(t)} + S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} \cdot U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}, \quad j = 1, \cdots, n,\end{aligned}$$

and the algorithm with learning rate adjustment has update

$$\begin{aligned}\boldsymbol{\theta}_i^{(t+1)} &= \boldsymbol{\theta}_i^{(t)} + \frac{lr}{t^{1/4}} S^{-1}_{\boldsymbol{\theta}_i^{(t)}} \cdot U_{\boldsymbol{\theta}_i^{(t)}}, \quad i = 1, \cdots, n, \\ \widetilde{\boldsymbol{\theta}}_j^{(t+1)} &= \widetilde{\boldsymbol{\theta}}_j^{(t)} + \frac{lr}{t^{1/4}} S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} \cdot U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}, \quad j = 1, \cdots, n.\end{aligned}$$

4.3.2 Gaussian pseudo-likelihood for positive observations and Poisson for zeros

In previous subsection, we used the Gaussian distribution as the pseudo-likelihood for all observations. In real data, however, the proportion of zeros may be too big to be reasonably modeled by Gaussian distribution. Therefore, we consider to respect the Poisson mechanism in Compound Poisson Gamma distribution. Note that the reason of using the pseudo-likelihood is because we had trouble for estimating the power parameter p_{ij} and dispersion parameter ϕ_{ij} due to three different forms of infinite summation involved in the likelihood function when p_{ij} is in different range. Here, we will try to adopt Poisson distribution for zero observations but Gaussian pseudo-likelihood just for the positive observations to avoid dealing with the infinite summation in Compound Poisson Gamma. In this case, the pseudo-log likelihood is

$$\begin{aligned}\ell = \log\left[\prod_{i,j} f\left(y_{ij}\right)\right] &= \textstyle\sum_{i=1}^n \sum_{j=1}^n \left\{\log f\left(y_{ij}\right) \cdot I\left(y_{ij} > 0\right) + \log f_0\left(y_{ij}\right) \cdot I\left(y_{ij} = 0\right)\right\} \\ &= \textstyle\sum_{i=1}^n \sum_{j=1}^n \ell_{ij}^{(P)} \cdot I\left(y_{ij} > 0\right) + \sum_{i=1}^n \sum_{j=1}^n \ell_{ij}^{(0)} \cdot I\left(y_{ij} = 0\right).\end{aligned}$$

The first double summation has same formula as the $L^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})$ and $\widetilde{L}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})$ given in previous subsection except that the summation is restricted to only positive observations. For $y_{ij}=0$,

$$\begin{aligned}
&\Pr\left[Y_{ij}=0\right]=\exp\left(-\lambda_{ij}\right), \qquad \text{where } \lambda_{ij}=\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}\left(2-p_{ij}\right)}, \quad \phi_{ij}=\exp(\delta_{ij}),\\
&\ell_{ij}^{(0)}=-\lambda_{ij}, \quad \log\mu_{ij}=\eta_{ij}\Leftrightarrow\mu_{ij}=\exp\left(\eta_{ij}\right).
\end{aligned}$$

As in previous subsection, we assume $\delta_{ij}=\delta_i+\delta_j$ and $p_{ij}=p_i+p_j$.

The first order partial derivatives for $\ell_{ij}^{(0)}$ are

$$\begin{aligned}
\frac{\partial\ell_{ij}^{(0)}}{\partial w_{i_1k}} &= -\frac{\partial\lambda_{ij}}{\partial\mu_{ij}}\frac{\partial\mu_{ij}}{\partial\eta_{ij}}\frac{\partial\eta_{ij}}{\partial w_{i_1k}}=-\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\widetilde{w}_{jk}I(i=i_1)I(y_{ij}=0),\\
\frac{\partial\ell_{ij}^{(0)}}{\partial b_{i_1}} &= -\frac{\partial\lambda_{ij}}{\partial\mu_{ij}}\frac{\partial\mu_{ij}}{\partial\eta_{ij}}\frac{\partial\eta_{ij}}{\partial b_{i_1}}=-\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}I(i=i_1)I(y_{ij}=0),\\
\frac{\partial\ell_{ij}^{(0)}}{\partial\delta_{i_1}} &= -\frac{\partial\lambda_{ij}}{\partial\phi_{ij}}\cdot\frac{\partial\phi_{ij}}{\partial\delta_{i_1}}=\frac{\lambda_{ij}}{\phi_{ij}}\cdot\phi_{ij}I(i=i_1)I(y_{ij}=0)=\lambda_{ij}I(i=i_1)I(y_{ij}=0)\\
\frac{\partial\ell_{ij}^{(0)}}{\partial p_{i_1}} &= \left\{\frac{\mu_{ij}^{2-p_{ij}}\cdot\log\left(\mu_{ij}\right)}{\phi_{ij}\left(2-p_{ij}\right)}+\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}\left(2-p_{ij}\right)^2}\right\}I(i=i_1)I(y_{ij}=0)\\
&= \left\{\lambda_{ij}\log\left(\mu_{ij}\right)+\frac{\lambda_{ij}}{2-p_{ij}}\right\}I(i=i_1)I(y_{ij}=0)
\end{aligned}$$

Similarly, the first order partial derivative for $\widetilde{\ell}_{ij}^{(0)}$

$$\begin{aligned}
\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{w}_{j_1k}} &= -\frac{\partial\lambda_{ij}}{\partial\mu_{ij}}\frac{\partial\mu_{ij}}{\partial\eta_{ij}}\frac{\partial\eta_{ij}}{\partial\widetilde{w}_{j_1k}}=-\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}w_{ik}I(j=j_1)I(y_{ij}=0),\\
\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{b}_{j_1}} &= -\frac{\partial\lambda_{ij}}{\partial\mu_{ij}}\frac{\partial\mu_{ij}}{\partial\eta_{ij}}\frac{\partial\eta_{ij}}{\partial\widetilde{b}_{j_1}}=-\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}I(j=j_1)I(y_{ij}=0)\\
\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\delta}_{j_1}} &= \lambda_{ij}I(j=j_1)I(y_{ij}=0),\\
\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{p}_{j_1}} &= \left\{\lambda_{ij}\log\left(\mu_{ij}\right)+\frac{\lambda_{ij}}{2-p_{ij}}\right\}I(j=j_1)I(y_{ij}=0).
\end{aligned}$$

Therefore, the first order partial derivatives combining $y_{ij} > 0$ and $y_{ij} = 0$ cases are

$$\frac{\partial \ell}{\partial \mathbf{w}_i} = \frac{\partial \ell^P\left(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}}\right)}{\partial \mathbf{w}_i} + \sum_{j=1}^{n} \frac{\partial \ell_{ij}^{(0)}}{\partial b_i}, \qquad \frac{\partial \ell}{\partial b_i} = \frac{\partial \ell^P\left(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}}\right)}{\partial b_i} + \sum_{j=1}^{n} \frac{\partial \ell_{ij}^{(0)}}{\partial b_i},$$
$$\frac{\partial \ell}{\partial \delta_i} = \frac{\partial \ell^P\left(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}}\right)}{\partial \delta_i} + \sum_{j=1}^{n} \frac{\partial \ell_{ij}^{(0)}}{\partial \delta_i}, \qquad \frac{\partial \ell}{\partial p_i} = \frac{\partial \ell^P\left(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}}\right)}{\partial p_i} + \sum_{j=1}^{n} \frac{\partial \ell_{ij}^{(0)}}{\partial p_i},$$

where $\partial \ell_i^P\left(\boldsymbol{\theta}_i; \widetilde{\boldsymbol{\theta}}\right)/\partial \mathbf{w}_i$ and so on follow similar formulae as in previous subsection except that they are restricted to positive observations only. That is,

$$\begin{aligned}
\frac{\partial \ell^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i} &= \sum_{j=1}^{n} \left\{ -\frac{1}{2} p_{ij} \cdot \widetilde{\boldsymbol{w}}_j + f_{ij} \cdot \widetilde{\boldsymbol{w}}_j + \frac{1}{2} p_{ij} \cdot g_{ij} \cdot \widetilde{\boldsymbol{w}}_j \right\} I(y_{ij} > 0), \\
\frac{\partial \ell^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})}{\partial b_i} &= \sum_{j=1}^{n} \left\{ -\frac{1}{2} p_{ij} + f_{ij} \cdot 1 + \frac{1}{2} p_{ij} \cdot g_{ij} \right\} I(y_{ij} > 0), \\
\frac{\partial \ell^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})}{\partial \delta_i} &= \sum_{j=1}^{n} \left\{ -\frac{1}{2} + \frac{1}{2} g_{ij} \right\} I(y_{ij} > 0), \\
\frac{\partial \ell^P(\boldsymbol{\theta}; \widetilde{\boldsymbol{\theta}})}{\partial p_i} &= \sum_{j=1}^{n} \left\{ -\frac{1}{2} \log(\mu_{ij}) + \frac{1}{2} \log(\mu_{ij}) \cdot g_{ij} \right\} I(y_{ij} > 0).
\end{aligned}$$

For $\widetilde{\boldsymbol{\theta}}_j$ part, the first order partial derivatives combining $y_{ij} > 0$ and $y_{ij} = 0$ cases are

$$\frac{\partial \widetilde{\ell}}{\partial \widetilde{\boldsymbol{w}}_j} = \frac{\partial \widetilde{\ell}^P\left(\widetilde{\boldsymbol{\theta}}; \boldsymbol{\theta}\right)}{\partial \widetilde{\boldsymbol{w}}_j} + \sum_{i=1}^{n} \frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_j}, \qquad \frac{\partial \widetilde{\ell}}{\partial \widetilde{b}_j} = \frac{\partial \widetilde{\ell}^P\left(\widetilde{\boldsymbol{\theta}}; \boldsymbol{\theta}\right)}{\partial \widetilde{b}_j} + \sum_{i=1}^{n} \frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_j},$$
$$\frac{\partial \widetilde{\ell}}{\partial \widetilde{\delta}_j} = \frac{\partial \widetilde{\ell}^P\left(\widetilde{\boldsymbol{\theta}}; \boldsymbol{\theta}\right)}{\partial \widetilde{\delta}_j} + \sum_{i=1}^{n} \frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\delta}_j}, \qquad \frac{\partial \widetilde{\ell}}{\partial \widetilde{p}_j} = \frac{\partial \widetilde{\ell}^P\left(\widetilde{\boldsymbol{\theta}}; \boldsymbol{\theta}\right)}{\partial \widetilde{p}_j} + \sum_{i=1}^{n} \frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{p}_j},$$

where $\partial \widetilde{\ell}_j^P\left(\widetilde{\boldsymbol{\theta}}_j; \boldsymbol{\theta}\right)/\partial \widetilde{\boldsymbol{w}}_j$ and other such derivatives follow similar formulae as in previous

subsection except that they are restricted to positive observations. They are,

$$\begin{aligned}
\frac{\partial \widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\boldsymbol{w}}_j} &= \sum_{i=1}^{n}\left\{-\frac{1}{2}p_{ij}\cdot\mathbf{w}_i + f_{ij}\cdot\mathbf{w}_i + \frac{1}{2}p_{ij}\cdot g_{ij}\cdot\mathbf{w}_i\right\} I(y_{ij}>0),\\
\frac{\partial \widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{b}_j} &= \sum_{i=1}^{n}\left\{-\frac{1}{2}p_{ij} + f_{ij}\cdot 1 + \frac{1}{2}p_{ij}\cdot g_{ij}\right\} I(y_{ij}>0),\\
\frac{\partial \widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{\delta}_j} &= \sum_{i=1}^{n}\left\{-\frac{1}{2} + \frac{1}{2}g_{ij}\right\} I(y_{ij}>0),\\
\frac{\partial \widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial \widetilde{p}_j} &= \sum_{i=1}^{n}\left\{-\frac{1}{2}\log(\mu_{ij}) + \frac{1}{2}\log(\mu_{ij})\cdot g_{ij}\right\} I(y_{ij}>0).
\end{aligned}$$

The second order partial derivatives for $\ell_{ij}^{(0)}$ are

$$\begin{aligned}
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial w_{i_1k}\partial w_{i_2r}} &= -\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot\widetilde{w}_{jr}\cdot\widetilde{w}_{jk}\cdot I(i=i_1=i_2)I(y_{ij}=0),\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial w_{i_1k}\partial b_{i_2}} &= -\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot\widetilde{w}_{jk}\cdot I(i=i_1=i_2)I(y_{ij}=0),\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial b_{i_1}\partial b_{i_2}} &= -\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot I(i=i_1=i_2)I(y_{ij}=0),\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial \mathbf{w}_{i_1}\partial \delta_{i_2}} &= \frac{\partial}{\partial\phi_{ij}}\left\{-\frac{\mu_{ij}^{2-\phi_{ij}}}{\phi_{ij}}\cdot\widetilde{\boldsymbol{w}}_j\right\}\cdot\frac{\partial\phi_{ij}}{\partial\delta_{i_2}}I(i=i_1=i_2)I(y_{ij}=0)\\
&= \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}^2}\cdot\widetilde{\boldsymbol{w}}_j\cdot\phi_{ij}I(i=i_1=i_2)I(y_{ij}=0) = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\cdot\widetilde{\boldsymbol{w}}_j I(i=i_1=i_2)I(y_{ij}=0)\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial \mathbf{w}_{i_1}\partial p_{i_2}} &= \mu^{2-p_{ij}}\cdot\log(\mu_{ij})\cdot\frac{1}{\phi_{ij}}\cdot\widetilde{\boldsymbol{w}}_j I(i=i_1=i_2)I(y_{ij}=0)\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial b_{i_1}\partial \delta_{i_2}} &= -\frac{\partial\ell_{ij}^{(0)}}{\partial b_{i_1}}I(i=i_1=i_2)I(y_{ij}=0),\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial b_{i_1}\partial p_{i_2}} &= -\frac{\partial\ell_{ij}^{(0)}}{\partial b_{i_1}}\cdot\log(\mu_{ij})\,I(i=i_1=i_2)I(y_{ij}=0)\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial p_{i_1}\partial p_{i_2}} &= \left\{\lambda_{ij}\left(\log\mu_{ij}+\frac{1}{2-p_{ij}}\right)^2 - \frac{\lambda_{ij}}{(2-p_{ij})^2}\right\} I(i=i_1=i_2)I(y_{ij}=0)\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial \delta_{i_1}\partial \delta_{i_2}} &= -\lambda_{ij}I(i=i_1=i_2)I(y_{ij}=0)\\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial \delta_{i_1}\partial p_{i_2}} &= -\left\{\lambda_{ij}\log(\mu_{ij}) + \frac{\lambda_{ij}}{2-p_{ij}}\right\} I(i=i_1=i_2)I(y_{ij}=0)
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{w}_{j_2 r}} &= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot w_{ir} \cdot w_{jk} \cdot I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{b}_{j_2}} &= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{jj}} \cdot w_{ik} \cdot I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_{j_1} \partial \widetilde{b}_{j_2}} &= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot I(j = j_1 = j_2) I(y_{ij} = 0)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\boldsymbol{w}}_{j_1} \partial \widetilde{\delta}_{j_2}} &= -\frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\boldsymbol{w}}_j} I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\boldsymbol{w}}_{j_1} \partial \widetilde{p}_{j_2}} &= -\frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\boldsymbol{w}}_j} \cdot \log(\mu_{ij}) I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_{j_1} \partial \widetilde{\delta}_{j_2}} &= -\frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_j} I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_{j_1} \partial \widetilde{p}_{j_2}} &= -\frac{\partial \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_j} \cdot \log(\mu_{ij}) I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{p}_{j_1} \partial \widetilde{p}_{j_2}} &= \left\{ \lambda_{ij} \left(\log \mu_{ij} + \frac{1}{2-p_{ij}} \right)^2 - \frac{\lambda_{ij}}{(2-p_{ij})^2} \right\} I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\delta}_{j_1} \partial \widetilde{\delta}_{j_2}} &= -\lambda_{ij} I(j = j_1 = j_2) I(y_{ij} = 0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{\delta}_{i_1} \partial \widetilde{p}_{i_2}} &= -\left\{ \lambda_{ij} \log(\mu_{ij}) + \frac{\lambda_{ij}}{2-p_{ij}} \right\} I(i = i_1 = i_2) I(y_{ij} = 0)
\end{aligned}
$$

The components of the Fisher information matrix $S_{\boldsymbol{\theta}_i}$ for $\ell_{ij}^{(0)}$ case are

$$
\begin{aligned}
S_{\mathbf{w}_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial\mathbf{w}_i\partial\mathbf{w}_i^\top}\middle| y_{ij}=0\right] = \sum_{j=1}^{n}\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot\widetilde{\boldsymbol{w}}_j\cdot\widetilde{\boldsymbol{w}}_j^\top\cdot I(y_{ij}=0),\\
S_{\mathbf{w}_i b_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial\mathbf{w}_i\partial b_i}\middle| y_{ij}=0\right] = \sum_{j=1}^{n}\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot\widetilde{\boldsymbol{w}}_j\cdot I(y_{ij}=0),\\
S_{b_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial b_i^2}\middle| y_{ij}=0\right] = \sum_{j=1}^{n}\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot I(y_{ij}=0),\\
S_{\delta_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial\delta_i^2}\middle| y_{ij}=0\right] = -\sum_{j=1}^{n}\lambda_{ij}I(y_{ij}=0),\\
S_{\delta_i p_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial\delta_i\partial p_i}\middle| y_{ij}=0\right] = \sum_{j=1}^{n}\left\{\lambda_{ij}\log(\mu_{ij})+\frac{\lambda_{ij}}{2-p_{ij}}\right\}I(y_{ij}=0),\\
S_{p_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial p_i^2}\middle| y_{ij}=0\right] = -\sum_{j=1}^{n}\left\{\lambda_{ij}\left(\log\mu_{ij}+\frac{1}{2-p_{ij}}\right)^2-\frac{\lambda_{ij}}{(2-p_{ij})^2}\right\}I(y_{ij}=0),\\
S_{\mathbf{w}_i\delta_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial\mathbf{w}_i\partial\delta_i}\middle| y_{ij}=0\right] = \sum_{j=1}^{n}\frac{\partial\ell_{ij}^{(0)}}{\partial\mathbf{w}_i}I(y_{ij}=0),\\
S_{\mathbf{w}_i p_i}^{(0)} &= -E\left[\sum_{j=1}^{n}\frac{\partial^2\ell_{ij}^{(0)}}{\partial\mathbf{w}_i\partial p_i}\middle| y_{ij}=0\right] = \sum_{j=1}^{n}\frac{\partial\ell_{ij}^{(0)}}{\partial\mathbf{w}_i}\cdot\log(\mu_{ij})\,I(y_{ij}=0)\\
&= -\sum_{j=1}^{n}\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\widetilde{\boldsymbol{w}}_j\eta_{ij}I(y_{ij}=0)\\
S_{b_i\delta_i}^{(0)} &= \sum_{j=1}^{n}\frac{\partial\ell_{ij}^{(0)}}{\partial b_i}I(y_{ij}=0), \qquad S_{b_i p_i}^{(0)} = \sum_{j=1}^{n}\frac{\partial\ell_{ij}^{(0)}}{\partial b_i}\log(\mu_{ij})\,I(y_{ij}=0)
\end{aligned}
\tag{4.3.4}
$$

The components of the Fisher information matrix $S_{\widetilde{\boldsymbol{\theta}}_j}$ for $\widetilde{\ell}_{ij}^{(0)}$ case are

$$
\begin{aligned}
S^{(0)}_{\widetilde{\mathbf{w}}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\mathbf{w}}_j\partial\widetilde{\mathbf{w}}_j^{\top}}\middle| y_{ij}=0\right] = \sum_{i=1}^{n}\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot\mathbf{w}_i\cdot\mathbf{w}_i^{\top}\cdot I(y_{ij}=0),\\
S^{(0)}_{\widetilde{\mathbf{w}}_j\widetilde{b}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\mathbf{w}}_j\partial\widetilde{b}_j}\middle| y_{ij}=0\right] = \sum_{i=1}^{n}\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot\mathbf{w}_i\cdot I(y_{ij}=0),\\
S^{(0)}_{\widetilde{b}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{b}_j^2}\middle| y_{ij}=0\right] = \sum_{i=1}^{n}\frac{(2-p_{ij})}{\phi_{ij}}\cdot\mu_{ij}^{2-p_{ij}}\cdot I(y_{ij}=0),\\
S^{(0)}_{\widetilde{\delta}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\delta}_j^2}\middle| y_{ij}=0\right] = -\sum_{i=1}^{n}\lambda_{ij}I(y_{ij}=0),\\
S^{(0)}_{\widetilde{\delta}_j\widetilde{p}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\delta}_j\partial\widetilde{p}_j}\middle| y_{ij}=0\right] = \sum_{i=1}^{n}\left\{\lambda_{ij}\log(\mu_{ij})+\frac{\lambda_{ij}}{2-p_{ij}}\right\}I(y_{ij}=0), \qquad (4.3.5)\\
S^{(0)}_{\widetilde{p}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{p}_j^2}\middle| y_{ij}=0\right] = -\sum_{i=1}^{n}\left\{\lambda_{ij}\left(\log\mu_{ij}+\frac{1}{2-p_{ij}}\right)^2-\frac{\lambda_{ij}}{(2-p_{ij})^2}\right\}I(y_{ij}=0),\\
S^{(0)}_{\widetilde{\mathbf{w}}_j\widetilde{\delta}_j} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\mathbf{w}}_j\partial\widetilde{\delta}_j}\middle| y_{ij}=0\right] = \sum_{i=1}^{n}\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\mathbf{w}}_j}I(y_{ij}=0),\\
S^{(0)}_{\widetilde{\mathbf{w}}_j\widetilde{p}_i} &= -E\left[\sum_{i=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\mathbf{w}}_j\partial\widetilde{p}_j}\middle| y_{ij}=0\right] = \sum_{i=1}^{n}\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{\mathbf{w}}_j}\cdot\log(\mu_{ij}) = -\sum_{i=1}^{n}\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}\mathbf{w}_i\eta_{ij}I(y_{ij}=0),\\
S^{(0)}_{\widetilde{b}_j\widetilde{\delta}_j} &= \sum_{i=1}^{n}\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{b}_j}I(y_{ij}=0), \qquad S^{(0)}_{\widetilde{b}_j\widetilde{p}_j} = \sum_{i=1}^{n}\frac{\partial\widetilde{\ell}_{ij}^{(0)}}{\partial\widetilde{b}_j}\log(\mu_{ij})\,I(y_{ij}=0).
\end{aligned}
$$

The components in the Fisher information matrix are given below.

$$
\begin{aligned}
&S_{\mathbf{w}_i} = S^P_{\mathbf{w}_i}+S^{(0)}_{\mathbf{w}_i}, \qquad S_{\mathbf{w}_ib_i} = S^P_{\mathbf{w}_ib_i}+S^{(0)}_{\mathbf{w}_ib_i}, \qquad S_{\mathbf{w}_i\delta_i} = S^P_{\mathbf{w}_i\delta_i}+S^{(0)}_{\mathbf{w}_i\delta_i},\\
&S_{b_i} = S^P_{b_i}+S^{(0)}_{b_i}, \qquad S_{b_i\delta_i} = S^P_{b_i\delta_i}+S^{(0)}_{b_i\delta_i}, \qquad S_{b_ip_i} = S^P_{b_ip_i}+S^{(0)}_{b_ip_i}\\
&S_{\delta_i} = S^P_{\delta_i}+S^{(0)}_{\delta_i}, \qquad S_{\delta_ip_i} = S^P_{\delta_ip_i}+S^{(0)}_{\delta_ip_i}, \qquad S_{p_i} = S^P_{p_i}+S^{(0)}_{p_i}.
\end{aligned}
$$

where $S^{(0)}_{\mathbf{w}_i}$, $S^{(0)}_{\mathbf{w}_i b_i}$, etc. are given in (4.3.4) and

$$
\begin{aligned}
S^P_{\mathbf{w}_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial \mathbf{w}_i^\top}\middle| y_{ij}>0\right] = \sum_{j=1}^n \left\{\frac{\widetilde{\boldsymbol{w}}_j \cdot \widetilde{\boldsymbol{w}}_j^\top}{\exp(\delta_{ij})\cdot \mu_{ij}^{p_{ij}-2}} + \frac{\widetilde{\boldsymbol{w}}_j \widetilde{\boldsymbol{w}}_j^\top \cdot p_{ij}^2}{2}\right\} I(y_{ij}>0),\\
S^P_{b_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i^2}\middle| y_{ij}>0\right] = \sum_{j=1}^n \left\{\frac{1\cdot 1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{1\cdot 1\cdot p_{ij}^2}{2}\right\} I(y_{ij}>0),\\
S^P_{\delta_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \delta_i^2}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n I(y_{ij}>0),\\
S^P_{p_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial p_i^2}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n (\log\mu_{ij})^2\, I(y_{ij}>0),
\end{aligned}
$$

$$
\begin{aligned}
S^P_{\mathbf{w}_i b_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial b_i}\middle| y_{ij}>0\right] = \sum_{j=1}^n \left\{\frac{1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}} + \frac{p_{ij}^2}{2}\right\}\cdot \widetilde{\boldsymbol{w}}_j I(y_{ij}>0),\\
S^P_{\mathbf{w}_i \delta_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial \delta_i}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n p_{ij}\cdot \widetilde{\boldsymbol{w}}_j I(y_{ij}>0),\\
S^P_{\mathbf{w}_i p_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \mathbf{w}_i \partial p_i}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n p_{ij}\cdot \widetilde{\boldsymbol{w}}_j \cdot \log(\mu_{ij})\, I(y_{ij}>0),\\
S^P_{b_i \delta_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i \partial \delta_i}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n p_{ij} I(y_{ij}>0),\\
S^P_{b_i p_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial b_i \partial p_i}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n p_{ij}\log(\mu_{ij})\, I(y_{ij}>0),\\
S^P_{\delta_i p_i} &= E\left[-\frac{\partial^2 \ell^P(\boldsymbol{\theta};\widetilde{\boldsymbol{\theta}})}{\partial \delta_i \partial p_i}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{j=1}^n \log(\mu_{ij})\, I(y_{ij}>0),
\end{aligned}
$$

$$
\begin{aligned}
S_{\widetilde{\boldsymbol{w}}_j} = S^P_{\widetilde{\boldsymbol{w}}_j} + S^{(0)}_{\widetilde{\boldsymbol{w}}_j}, \qquad S_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} = S^P_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j} + S^{(0)}_{\widetilde{\boldsymbol{w}}_j \widetilde{b}_j}, \qquad S_{\widetilde{\boldsymbol{w}}_j \widetilde{\delta}_j} = S^P_{\widetilde{\boldsymbol{w}}_j \widetilde{\delta}_j} + S^{(0)}_{\widetilde{\boldsymbol{w}}_j \widetilde{\delta}_j}\\
S_{\widetilde{b}_j} = S^P_{\widetilde{b}_j} + S^{(0)}_{\widetilde{b}_j}, \qquad S_{\widetilde{b}_j \widetilde{\delta}_j} = S^P_{\widetilde{b}_j \widetilde{\delta}_j} + S^{(0)}_{\widetilde{b}_j \widetilde{\delta}_j}, \qquad S_{\widetilde{b}_j \widetilde{p}_j} = S^P_{\widetilde{b}_j \widetilde{p}_j} + S^{(0)}_{\widetilde{b}_j \widetilde{p}_j}\\
S_{\widetilde{\delta}_j} = S^P_{\widetilde{\delta}_j} + S^{(0)}_{\widetilde{\delta}_j}, \qquad S_{\widetilde{\delta}_j \widetilde{p}_j} = S^P_{\widetilde{\delta}_j \widetilde{p}_j} + S^{(0)}_{\widetilde{\delta}_j \widetilde{p}_j}, \qquad S_{\widetilde{p}_j} = S^P_{\widetilde{p}_j} + S^{(0)}_{\widetilde{p}_j}
\end{aligned}
$$

where $S^{(0)}_{\widetilde{\boldsymbol{w}}_j}$, $S^{(0)}_{\widetilde{\boldsymbol{w}}_j\widetilde{b}_j}$, etc. are given in (4.3.5) and

$$
\begin{aligned}
S^P_{\widetilde{\boldsymbol{w}}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{\boldsymbol{w}}_j\partial\widetilde{\boldsymbol{w}}_j^\top}\middle| y_{ij}>0\right] = \sum_{i=1}^n\left\{\frac{\mathbf{w}_i\cdot\mathbf{w}_i^\top}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}}+\frac{\mathbf{w}_i\cdot\mathbf{w}_i^\top\cdot p_{ij}^2}{2}\right\}I(y_{ij}>0),\\
S^P_{\widetilde{b}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{b}_j^2}\middle| y_{ij}>0\right] = \sum_{i=1}^n\left\{\frac{1\cdot 1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}}+\frac{1\cdot 1\cdot p_{ij}^2}{2}\right\}I(y_{ij}>0),\\
S^P_{\widetilde{\delta}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{\delta}_j^2}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n I(y_{ij}>0),\\
S^P_{\widetilde{p}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{p}_j^2}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n(\log\mu_{ij})^2\,I(y_{ij}>0),
\end{aligned}
$$

$$
\begin{aligned}
S^P_{\widetilde{\boldsymbol{w}}_j\widetilde{b}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{\boldsymbol{w}}_j\partial\widetilde{b}_j}\middle| y_{ij}>0\right] = \sum_{i=1}^n\left\{\frac{1}{\exp(\delta_{ij})\,\mu_{ij}^{p_{ij}-2}}+\frac{p_{ij}^2}{2}\right\}\cdot\mathbf{w}_i I(y_{ij}>0),\\
S^P_{\widetilde{\boldsymbol{w}}_j\widetilde{\delta}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{\boldsymbol{w}}_j\partial\widetilde{\delta}_j}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n p_{ij}\cdot\mathbf{w}_i I(y_{ij}>0),\\
S^P_{\widetilde{\boldsymbol{w}}_j\widetilde{p}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{\boldsymbol{w}}_j\partial\widetilde{p}_j}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n p_{ij}\cdot\mathbf{w}_i\cdot\log(\mu_{ij})\,I(y_{ij}>0),\\
S^P_{\widetilde{b}_j\widetilde{\delta}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{b}_j\partial\widetilde{\delta}_j}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n p_{ij}I(y_{ij}>0),\\
S^P_{\widetilde{b}_j\widetilde{p}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{b}_j\partial\widetilde{p}_j}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n p_{ij}\cdot\log(\mu_{ij})\,I(y_{ij}>0),\\
S^P_{\widetilde{\delta}_j\widetilde{p}_j} &= E\left[-\frac{\partial^2\widetilde{\ell}^P(\widetilde{\boldsymbol{\theta}};\boldsymbol{\theta})}{\partial\widetilde{\delta}_j\partial\widetilde{p}_j}\middle| y_{ij}>0\right] = \frac{1}{2}\sum_{i=1}^n\log(\mu_{ij})\,I(y_{ij}>0).
\end{aligned}
$$

Denote $\boldsymbol{\theta}_i=\left(\mathbf{w}_i^\top,b_i,\delta_i,p_i\right)^\top$, $\widetilde{\boldsymbol{\theta}}_j=\left(\widetilde{\boldsymbol{w}}_j^\top,\widetilde{b}_j,\widetilde{\delta}_j,\widetilde{p}_j\right)^\top$. For different i's, say $i_1\neq i_2$, the estimation of $\boldsymbol{\theta}_{i_1}$ and $\boldsymbol{\theta}_{i_2}$ can be done separately because the partial derivatives with respect to $\boldsymbol{\theta}_{i_1}$ and $\boldsymbol{\theta}_{i_2}$ are zero. Therefore, the updating formula using Fisher scoring algorithm without learning rate adjustment is

$$
\begin{aligned}
\boldsymbol{\theta}_i^{(t+1)} &= \boldsymbol{\theta}_i^{(t)}+S^{-1}_{\boldsymbol{\theta}_i^{(t)}}\cdot U_{\boldsymbol{\theta}_i^{(t)}},\quad i=1,\cdots,n,\\
\widetilde{\boldsymbol{\theta}}_j^{(t+1)} &= \widetilde{\boldsymbol{\theta}}_j^{(t)}+S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}\cdot U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}},\quad j=1,\cdots,n,
\end{aligned}
$$

and the algorithm with learning rate adjustment has update

$$\boldsymbol{\theta}_i^{(t+1)} = \boldsymbol{\theta}_i^{(t)} + \frac{lr}{t^{1/4}} \cdot S^{-1}_{\boldsymbol{\theta}_i^{(t)}} \cdot U_{\boldsymbol{\theta}_i^{(t)}}, \quad i = 1, \ldots, n,$$
$$\widetilde{\boldsymbol{\theta}}_j^{(t+1)} = \widetilde{\boldsymbol{\theta}}_j^{(t)} + \frac{lr}{t^{1/4}} \cdot S^{-1}_{\widetilde{\boldsymbol{\theta}}_j^{(t)}} \cdot U_{\widetilde{\boldsymbol{\theta}}_j^{(t)}}, \quad j = 1, \ldots, n,$$

where $S_{\boldsymbol{\theta}_i}$ and $S_{\widetilde{\boldsymbol{\theta}}_j}$ are the Fisher information matrices with structure given in (4.3.3) using components defined in this subsection.

4.3.3 Numerical performance

We tried to apply the pseudo-likelihood approaches to one of the datasets generated for assessing alternating Tweedie MLE approach. For that dataset, the alternating Tweedie MLE approach gave satisfactory results when the Fisher scoring algorithm was used regardless of using learning rate adjustment or not or using gradient descent algorithm with Adam update (see Section 4.2.1, particularly, Figures 4.8, 4.9). For the same dataset, we applied both pseudo-likelihood approaches in Section 4.3.1 and 4.3.2 to see how the algorithm performs. The exact same initial values of parameters were used as in the case of alternating Tweedie MLE approach.

The two pseudo-likelihood approaches differ in how they model the probability of zero observations. This difference, unfortunately, is not sufficient to correct the algorithm's wrong direction in parameter update. We observed consistent behavior for both approaches. Both algorithms lead to NaN or infinite values in intermediate estimate of model components at the very early stage of iterations (see Figures 4.13, 4.14). Some truncations applied to these estimated model components only made some corrections that did not pull the algorithm to correct estimating direction. Both algorithms failed to proceed after only a few iterations. We will give a little more detail about the Gaussian pseudo-likelihood plus Poisson results in the next two paragraphs. The other one has very similar behavior and we skip the details.

Using the pseudo-likelihood method with learning rate adjustment, the non-zero observa-

tions' score vectors have huge magnitude (mostly a few thousands to 7000) but the amount of update is small. If the update is in the right direction, even small update per iteration could lead to optimizing the objective function and therefore reducing the magnitude of the score vectors. Unfortunately, what we observed after the first iteration of parameter update is that all the score vectors became NaN because some of the estimated μ_{ij} are close to zero (below 10^{-7}) and some are close to infinity (exp(568)) for non-zero observations. In formula (4.3.2), we can see that near zero value of μ_{ij} will lead to large f_{ij} and g_{ij}, and the near infinity value of μ_{ij} makes the f_{ij} and g_{ij} become near infinity. The large value such as exp(568) is practically stored as infinity in computer which lead to the f_{ij} and g_{ij} to be NaN. As a result, the pseudo-likelihood algorithm could not proceed.

We tried to truncate the estimated μ_{ij} such that the $\hat{\mu}_{ij}$ is between 0.1 and 10^{10} for positive observations and between 10^{-5} and 9 for zero observations. The range for the zero case is because the probability of zero is less than 10^{-4} if the Poisson distribution has mean at least 9. With such truncation implemented in the algorithm, we are able to see more iterations of parameter estimation. However, the diverging behavior persists. Figures 4.14 presented the estimated μ and η for positive observations at the end of 3rd iterations. Most of the estimated μ's are either close to zero or infinity. This small simulation shows that the pseudo-likelihood approach does not have the ability to estimate the parameters when no covariates are observed. The nice behavior reported in the literature for pseudo-likelihood Tweedie regression critically depends on covariate values being available which is the case in most statistical inference.

In summary, pseudo-likelihood approach failed to provide updates toward the optimization direction and therefore is not recommended in alternating models when covariates are not observed.

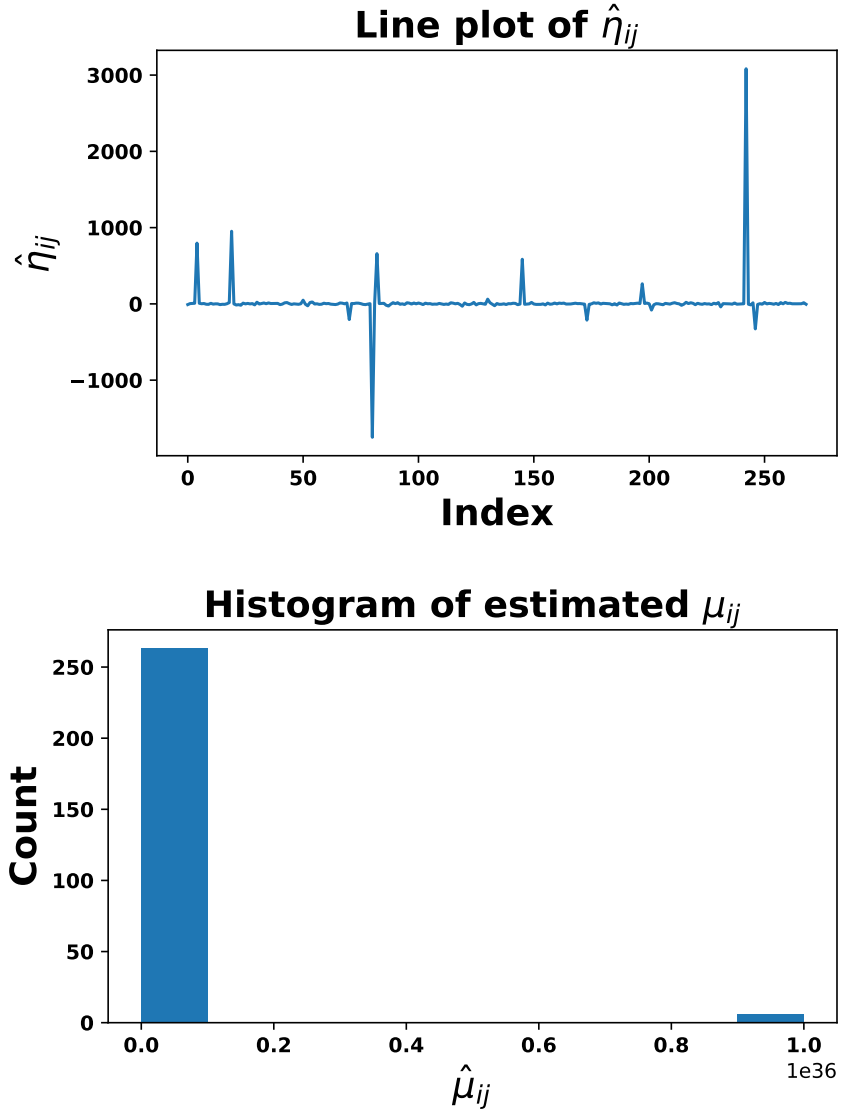

Figure 4.13: *Gaussian pseudo-likelihood estimates η_{ij} and μ_{ij} at 11th iteration for only positive observations based on the formulae in section 4.3.1. The top panel shows the line plot of the $\hat{\eta}_{ij}$ with the value of $\hat{\eta}_{ij}$ on vertical axis. These values are either close to zero or very big. The bottom panel shows the $\hat{\mu}_{ij}$ truncated at 10^{36}. Most of the $\hat{\mu}_{ij}$ are either zero or infinity (internally in computer) if it were not truncated. Even though there are sufficient number of μ_{ij}'s having values within a few thousands, the small number of large ones make the algorithm diverge despite that the amount of update on $\widehat{\boldsymbol{\theta}}$ and $\widehat{\widetilde{\boldsymbol{\theta}}}$ was restricted to be within [-100, 100] during iterations.*

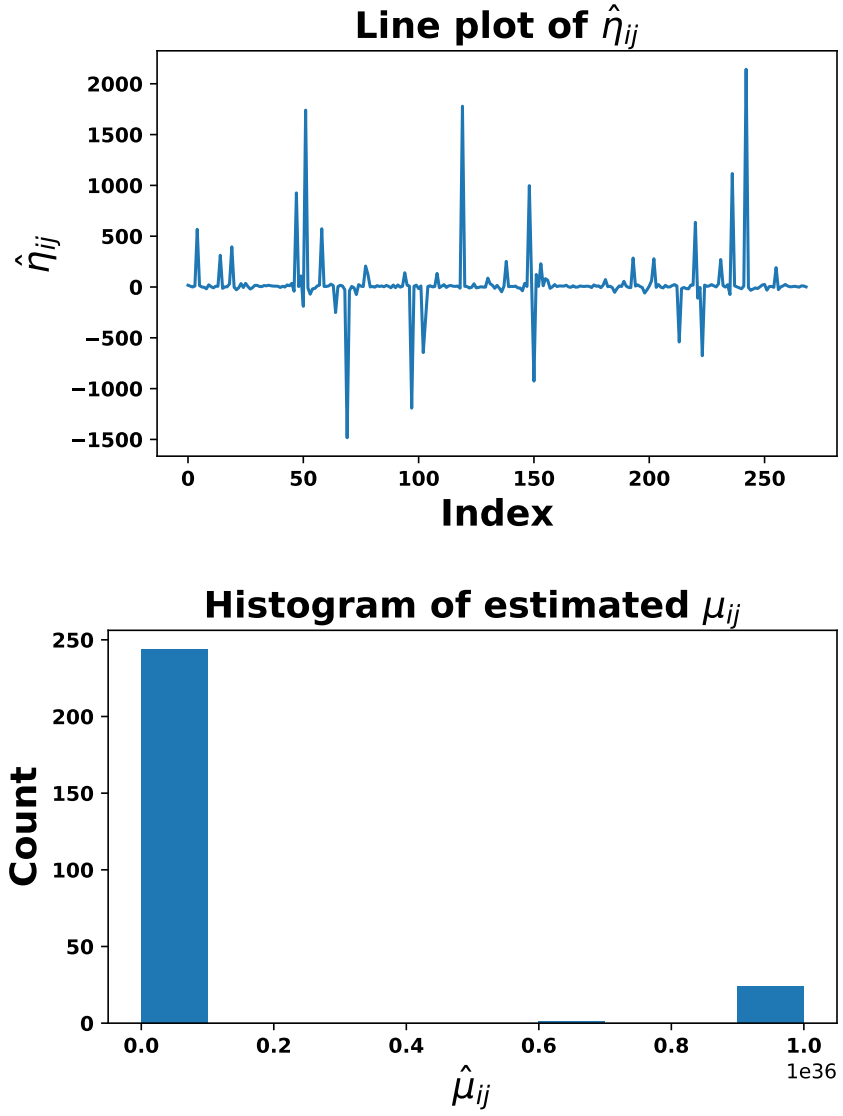

Figure 4.14: *The estimated η_{ij} and μ_{ij} at 3rd iteration for only positive observations. The top panel shows the line plot of the $\hat{\eta}_{ij}$ with the value of $\hat{\eta}_{ij}$ on vertical axis. These values are either close to zero or very big. The bottom panel shows the $\hat{\mu}_{ij}$ truncated at 10^{36}. Most of the $\hat{\mu}_{ij}$ are either zero or infinity (internally in computer) if it were not truncated.*

Chapter 5

Computational Strategy

5.1 Obtaining data, pre-processing, and constructing SQLite DB for sparse format co-occurrence matrix

We would like to use Wikipedia dump files as data for estimating the word vectors. The files contain copies of all Wikipedia pages in xml format provided by Wikipedia. They are available on the webpage https://en.wikipedia.org/wiki/Wikipedia:Database_download. We downloaded the January 20th, 2022 version from https://dumps.wikimedia.org/enwiki/20220120/. There are 61 partitioned bz2 compressed files, each of which contains xml version of page contents. The size of individual file ranges from about 300 MB to 1 GB. The total size of all files is 19 GB compressed (expands to over 86 GB when decompressed). After all 61 bz2 files were downloaded, we first need to parse the files. Each bz2 file is still too large to fit in memory for direct parsing. Instead, we parse one line of a bz2 file at a time using the WikiXmlHandler() class from webpage https://gist.github.com/WillKoehrsen/1c3afe1786cb36296b82892bd3a914f4#file-sax_content_handler-py. To extract titles and article contents from each xml page, we used another layer of parsing with the parse command from mwparserfromhell Python package. Afterward, simple pre-processing was done

to remove empty/white spaces and non-article content such as REDIRECT and thumbnail, and the contents in <> which are usually html tags and styles etc. Once the removal is completed, the collected titles and article contents were written into SQLite databases. Each bz2 file leads to one database.

From the databases collected, each article was converted to lower case and then tokenized by using RegexpTokenizer from the nltk Python package. Then unique words from each tokenized article were collected to build vocabulary, which consists of all unique words in all articles in all databases. The entire vocabulary had about 12.8 million unique words and we exploited the top 400K most frequently used words in our study.

Next, we built co-occurrence matrix corresponding to the top 400K vocabulary. Different from regular co-occurrence matrix which provides counts of a word appearing in another word's context in a paragraph, we consider weighted co-occurrence count for words in the same sentence. Specifically, first we defined windows centered at a target word spanning k words to the left and the k words to the right side restricted within the same sentence. All words in the window are considered as context words. For a context word in the window, the contribution of this word to the target word in the co-occurrence measure is defined to be the inverse of their separated distance between two words. That is, the co-occurrence count X_{ij} in the ith row and jth column of the matrix is given by

$$X_{ij} = \begin{cases} \sum_{\text{s} \in \text{ all sentences}} 1/d_{ij}(s), & \text{if } d_{ij}(s) \leq \text{k} \\ 0, & otherwise \end{cases}$$

where $d_{ij}(s)$ = separation between word i and word j in sentence s. The closer a context word is to the target word, the more contribution it has in relevance of the two words.

In addition to modifying the regular co-occurrence matrix to a sentence-level weighted co-occurrence matrix, we reformed the regular co-occurrence matrix in a sparse format to overcome the limited memory restriction on CPU. In the sparse format, the row and column index of non-zero entries along with its weight were recorded. The zero entries and their

indices were implied from the sparse format without needing any storage space. Regular format of matrix, i.e. dense matrix, containing all entries has physical limitation in terms of memory size when its dimension is large. For example, we explored the data loading and the calculation using dense format, the total memory being used were 54.29 GB and 222.59 GB for 50K by 50K and 100K by 100K matrices, respectively. Among the use of the memory, 20 GB and 80 GB were totally reserved by loading the matrices in these two cases. This is because each entry in the matrices is a floating point number and requires eight bytes for storage. Further computation with the given matrices is in need of even more space on the memory. In our experiment, therefore, we used the weighted co-occurrence matrix in sparse format. Note that the sentence-level co-occurrence count is our focus because this will avoid windows crossing different sentences.

Though conceptually it is easy to construct the sparse format co-occurrence matrix when the data are coming from a database, we had to construct one single co-occurrence matrix from 61 partitioned and pre-processed databases. Due to handling multiple DB and their size, we went through multiple steps to create the single co-occurrence matrix in sparse format:

- For a word pair with words i and j, we computed $1/d_{ij}(s)$ from each pre-processed database and recorded it along with index i and j onto 61 new databases. The size of databases has huge variation, from 6 GB to 52 GB, since the row and column indices are not unique yet.

- Continuing from the newly created 61 databases, we constructed another 61 databases consist of X_{ij} and unique row and column index i and j. The calculation of X_{ij} was done by using GROUP BY clause from SQLite. This operation is done on SQLite databases instead of done on Python because the command behind the database operation is written in C which is much more efficient than the Python.

- The X_{ij}'s from different databases need to be combined but it is impossible to read

them into the memory simultaneously. In fact, the maximum number of databases that can be handled is 10 in default setting. Thereafter, we generated intermediate 12 databases. This is done by splitting the previous 61 databases into 12 groups with roughly five databases per group. And then the content in each group were concatenated together by first attaching corresponding databases and then take the union and finally grouped by i and j index. These are achieved by command ATTACH DATABASE, UNION ALL, and GROUP BY clause from SQLite.

- Similar operation was done to combine every 4 of the 12 databases leading to 3 intermediate databases. The size of these 3 databases is much larger than the original or the earlier version of intermediate databases.

- The final step was constructing the single co-occurrence matrix from integrating the 3 intermediate databases in the similar way and stored into SQLite database. In the end, the sizes of the single co-occurrence matrix in sparse format were 15.96 GB, 25.15 GB, 34.25 GB, and 42.74 GB according to their vocabulary size 50K, 100K, 200K, and 400K, respectively.

Such incremental operations to reduce 61 to 12 and then to 3 and finally to one database are critical before any further inference can be done because loading all entries into the memory is unfeasible due to memory restriction. The sparse format not only overcomes the limitation of memory size but also enhances computational efficiency. The efficiency comes from the fact that we only need to retrieve corresponding one row of co-occurrence matrix from this final database instead of loading all entries on the memory and go through all of them.

To achieve even better computational time for loading data, we implemented a less memory demanding technique when fetching one row of the co-occurrence matrix. Note that one row of the co-occurrence matrix corresponds to many rows in the database. The technique involves two points:

- Utilizing the special column ‘rowid’ in SQLite table. The rowid is automatically gen-

erated for most SQLite databases tables when they are initiated. The SQLite tables use B-Tree structure to store their data with each entry for one row using rowid as the key. This implies fetching or sorting is fast in use of the rowid. According to the SQLite documentation, "Searching for a record with a specific rowid, or for all records with rowids within a specified range is around twice as fast as a similar search made by specifying any other PRIMARY KEY or indexed value". This rowid feature could significantly reduce the search time especially when we have a lot of rows since it takes at most $\log_2(n)$ steps to locate the rows from the B-Tree structure, where n is the number of rows in the entire table.

- Restricting number of entries. To make the computation more efficient, we further restrict the search to only look-up a portion of the rowids without going through the entire table. For example, if we are retrieving the first row of the co-occurrence matrix having vocabulary size 400K from the SQLite database, we specify a range by setting rowid from 0 to 400K with the WHERE command because there are at most 400K entries in that row. This is feasible because the recorded row indices (different from rowid) were already ordered in a non-decreasing order. Consequently, when we retrieve data having row index exactly equals to 0, there is no need to go beyond 400K entries. At the same time, we mark the number of entries already retrieved so that it can be used to specify the starting value of rowid for the next row.

These two strategies reduced significant amount of time for retrieving data. Specifically, it took around 376.42 seconds for retrieving one row of the co-occurrence matrix having vocabulary size 400K when we do not use these two strategies. With the strategies, it only took around 0.61 seconds.

5.2 Some detailed computational concerns and strategies

The large vocabulary size, it could take weeks or months to complete all necessary iterations until convergence. The computing resources we could access, unfortunately, does not allow non-interrupted computation using high memory for such a long time. For example, with Google Colab Pro membership, it only allows at most 24 hours of continuous computation time. With Beocat cluster machines, submitted jobs are often preempted in which case the job is terminated and rescheduled possibly to a different machine. Additionally, Beocat restricts a job to have maximum of 30 days of running time. Due to these restrictions, we need to write out intermediate results for any submission of job. For some of problems, we could resume by reading in the intermediate results and restart the algorithm. Some jobs are still a problem when we try to resume. A serious one is combining multiple databases into one database. If the operation is interrupted/terminated before it completes, resuming will restart from the beginning which will lead to duplicated entries in the resulting database. If the resulting database is not huge, we could easily add an additional field/attribute/column to record the stage of the combining and avoid rewriting into it by searching the record. Unfortunately, with such a large database, the searching is a problem as it took too much time. So, we can not make the algorithm to do a search every time when we are about to combine and write a content into it. In this subsection, we document some of our successful strategies to allow computing and resuming the algorithm without trouble.

First thing to consider is what intermediate results have to be written out. Some of the important ones are current estimate of word vectors and the iteration number. The iteration number turns out to be critical because it enters the updating equation:

$$\boldsymbol{\beta}_i^{(t+1)} = \boldsymbol{\beta}_i^{(t)} + \frac{lr}{t^{1/4}} \mathbf{I}^{-1}_{\boldsymbol{\beta}_i^{(t)}} \mathbf{U}_{\boldsymbol{\beta}_i^{(t)}}. \tag{5.2.1}$$

If the iteration number t is not updated, the algorithm could diverge. For example, with small Reuters data, consider to set $maxit = 2$ and the number of epochs to be 1, and resume twice. At the end of fifth and sixth iterations, the result was showing large amount of divergence as shown in the firs half of Table 5.1. On the other hand, when the algorithm runs 6 iterations without being interrupted (see the second half of Table 5.1), the result, which is correct computation, shows totally different converging pattern. Therefore, the iteration number should be written out and reloaded when the algorithm is resumed after interruption.

t	resume	number of times resume	$\|\|\mathbf{U}_{\boldsymbol{\beta}_i^{(t)}}\|\|$	$\|\|\mathbf{U}_{\tilde{\boldsymbol{\beta}}_i^{(t)}}\|\|$
1	False	NA	163391.56	167462.34
2	False	NA	26318.22	20618.02
1	True	1	8665.76	6915.76
2	True	1	2737.99	2217.22
1	True	2	5001087.00	4000937.00
2	True	2	5000456.00	4000385.50
1	False	NA	163391.56	167462.34
2	False	NA	26318.22	20618.02
3	False	NA	8778.63	7063.61
4	False	NA	3572.46	2980.23
5	False	NA	1788.05	1502.75
6	False	NA	1047.08	867.63

Table 5.1: *Iteration number t affects the updating direction when the algorithm is resumed or not. Keeping track of the correct iteration number is critical for the algorithm to update toward right direction.*

5.2.1 CPU-GPU parallelization

Co-occurrence matrix for the entire vocabulary has massive size (45.88 GB for vocabulary size 400K, 36.77 GB for vocabulary size 200K, 26.99 GB for vocabulary size 100K, and 17.13 GB for vocabulary size 50K, all in sparse format). Such sizes make it impossible to

load the entire co-occurrence matrix into memory and conduct estimation. To make the estimation feasible, we load one row of the co-occurrence matrix at a time directly from the SQLite database and the counts in the row are used to compute the score vector and information matrix. These are used in formula (5.2.1) to update the estimate of the word vector corresponding to the row of the co-occurrence matrix. Even though the update is using word counts in the row, the mean word counts depends on the word vectors of other words that appeared in any window of this word. That is, this update depends on word vector representations of all other words either directly or indirectly. Consequently, the estimate of the word vector for this word is updated while we go through the estimation of all the other words. When the algorithm updates go through all the other words(rows), it completes one iteration. Unfortunately, the counts can not be stored in the memory while the algorithm moves on to the next row in the iteration. As a result, the data retrieval from the database has to be repeated when the algorithm goes through each iteration. Many iterations are needed for the algorithm to converge. Upon convergence of the algorithm, each row of the co-occurrence matrix was retrieved as many times as the number of iterations. This creates a huge computational burden. To get an idea how much time is needed in the computation, the columns labeled 'sequential' in Table 5.2 reports the computation time of loading data for one row of the cooccurrence matrix from SQLite database and completing one iteration of parameter update for estimating word vectors on Google Colab with a machine having CUDA version 11.3 and CUDA device Tesla P100-PCIE-16GB and PyTorch version 1.11.0+cu113. It takes between 0.4 days to 21.8 days to complete one iteration of parameter update. This is too long to be practical.

Table 5.2: *Computation time of loading data for one word from SQLite database and completing one iteration of parameter update for estimating word vectors on Google Colab with a machine having CUDA version 11.3 and CUDA device Tesla P100-PCIE-16GB and PyTorch version 1.11.0+cu113.*

vocab size		d = 300 num_chunks = 30		d = 100 num_chunks = 4		d = 50 num_chunks = 10	
		sequential	CPU-GPU parallel	sequential	CPU-GPU parallel	sequential	CPU-GPU parallel
400K	per row	4.7 sec	4 sec	1.8 sec	0.95 sec	1.3 sec	0.6 sec
	per itr	21.8 days	18.5 days	8.3 days	4.4 days	6.0 days	2.8 days
200K	per row	2.3 sec	1.96 sec	1 sec	0.57 sec	0.9 sec	0.4 sec
	per itr	5.3 days	4.5 days	2.3 days	1.3 days	2.1 days	0.9 days
100K	per row	1.22 sec	1.05 sec	0.5 sec	0.39 sec	0.47 sec	0.3 sec
	per itr	1.4 days	1.2 days	0.6 days	0.5 days	0.5 days	0.3 days
50K	per row	0.68 sec	0.59 sec	0.35 sec	0.27 sec	0.35 sec	0.22 sec
	per itr	0.4 days	0.3 days	0.2 days	0.2 days	0.2 days	0.1 days

To speed up the computation, for each combination of vocabulary sizes and dimension of word vectors, we design the algorithm with CPU-GPU parallel processing and compared it with the sequential computation. The sequential method iteratively retrieves a row from the database and then computes the parameter update in a linear fashion, in which the time for data retrieval and time for computation are stacked on each other. The data retrieval accesses the database which is only allowed in CPU. On the other hand, updating parameter estimates can be done on GPU more efficiently. The two processes have to be done in a linear way in general because the estimation can not be implemented until the data is available. In the iteration over different rows, we have parallelized the data retrieval for the next row and the parameter updating on current row on two different processes. More specifically, the following is done:

- The data retrieval is done on a child process by using multiprocessing from Python

package.

- The estimation of parameters is done on the main process with a GPU device using PyTorch.
- Once a row is fetched, it is stored in a queue object for GPU to use. The queue is a shared memory object that connects the child process and the main process.
- The GPU gets the data from the queue object and removes them from the queue.
- Once the queue is empty, the child process retrieves next row of data while the GPU device conducts the computation for current row's parameters update. These two run parallelly to save time.
- It generally takes longer time to compute parameter update on GPU than data retrieval from the database on CPU. The total time of the child process and the main process for vocabulary size 50K, 100K, 200K, and 400K is 0.35, 0.47, 0.90, and 1.30 seconds respectively when the dimension of word vectors is 50, and is 0.68, 1.22, 2.30, and 4.70 seconds when the dimension of word vectors is 300 (see Table 5.2). Fetching a row takes 0.09, 0.17, 0.35, and 0.7 seconds for vocabulary size 50K, 100K, 200K, and 400K respectively.
- Due to large size of the data, the data retrieval on CPU would only proceed when the queue is empty. This helps to reduce memory usage so that there are enough resources devoted to computation.

In Table 5.2, there is another strategy hiding behind of sequential or parallel computation. This is reflected in num_chunks which is the number of chunks to partition the co-occurrence counts in one row of the data into groups such that each group only contains a small number of entries. This is essentially doing batch processing to reduce the demand on GPU memory while achieving shorter computation time required to process the entire row of data. One row of data contains as many entries as the vocabulary size even though some of them are

zeros. Note that zero entries also contribute to computing the log likelihood function so they also need to be processed by computing their corresponding parts in the score vector and the information matrix which involves computing as many matrices of dimension $(d+1)\times(d+1)$ as the number of entries in a batch. The computation and the memory usage have a trade-off in the batch calculation. This is because the Fisher information matrix contains outer product of all the $\mathbf{w}_i$ and $\mathbf{w}_i^\top$, and the $\widetilde{\boldsymbol{w}}_j$ and $\widetilde{\boldsymbol{w}}_j^\top$. These outer products could not be calculated directly due to memory constraint if no loop is used and using loop is exorbitantly slow. We used 'einsum' function to make the computation speed acceptable and without resorting to loops. The 'einsum' converts the outer product calculation into the sum of 3d array in smaller batches. The best batch size differs for different dimension of word vectors. Through trial and errors, we find that the values 30, 4, and 10 work well for dimension of vectors being 300, 100, and 50 respectively. These values, however, are not strict. A big range of values could work well. For example, with number of chunks equals 10 and the dimension equals 100, the required computation time for one row is similar to the case with $num\_chunks = 4$ and $d = 100$. Similarly for $num\_chunks = 4$, $d = 50$, the computation is close to that of $num\_chunks = 10$, $d = 50$ that are reported in Table 5.2.

5.3 Other computational efforts

Beyond the strategies described in previous section, we experimented many other ways to make the computation feasible. The ways we consider are parallel computing with multiprocessing Python package on CPU, Numba Python package with CPU, etc. In what follows, we will briefly describe what are these and computational capacity and their performance in our application.

Under the multiprocessing package, we tried the Pool class, Manager class, and shared memory objects. With the Pool class, we defined a function to compute the components of the score vector and information matrix for beta part for one row of co-occurrence matrix.

This function was used with the apply_async() and starmap_async() methods to achieve the parallel computation for different entries of the row. The computation was done with numpy arrays which does not support GPU computation so it was calculated on CPU. Both the apply_async() and starmap_async() could implement parallel computing by simply specifying target function that needs to be parallelized. The only difference between those two are how we pass arguments of the target function. The former one passes the arguments as normal Python functions, while the latter one, on the other hand, passes the arguments as an iterable (collection of the argument values to be applied by the function one at a time). The '_async()' on both methods stands for asynchronous option which does not require computation results to be collected in order from different processors. We experimented these two cases on smaller database of size 5.91 GB. The performance of the apply_async() was very slow and the calculation for parameter update in one row of the cooccurence matirx did not finish even after running for 45 minutes. The starmap_async() completed its computation in around 16 minutes. Note that without the parallelization on CPU, the computation time was only 1 minute to give the result of the score vector and information matrix. So, the Pool class using the apply_async() and starmap_async() is very inefficient.

The reason of ineffectiveness of the Pool class mainly comes from the fact that the row data has to be copied to as many times as the number of processes when it was provided as input argument. The multiple processes end up demanding more memory than the single process. If the data contains a lot of entries, the impact of multiple copies could be huge. Furthermore, computation results from the different processes have to be collected via communication among them, which induces the overheads. We tried to bypass the drawbacks through sharing data or objects among multiple processes. To share the data or objects, the multiprocessing package provides two ways: the Manager class and shared memory objects. The Manager class controls the data/objects on a server process, which is created whenever Python is initiated. The server process allows other processes to access and modify the data. The shared memory objects are objects created on shared memory space so that

other processes could manipulate the objects allocated in the shared memory space. The shared memory objects are ctypes arrays/objects which make implementation fast but have restriction on their types accepted such as Array, Value, etc. On the contrary, the Manager class is more flexible by allowing arbitrary types such as lists, dictionaries, Array, Value, etc. Its execution, however, is slower than the shared memory objects. With the same smaller experimental database and function described from previous paragraph, the computation time using the Manager class and shared memory objects were around 2 minutes and 44 seconds, respectively. Recall that the motive for using the multiprocessing package is to reduce computation time. The computation time for the Manager class was still slower than 1 minute, which is the time to complete without CPU parallelization. The shared memory objects showed improvement, however, it was not enough. We hoped the computation regarding one row to be done within a second since our computation needs to cycle through as many of rows as the vocabulary size for one iteration and the algorithm needs many iterations to converge. With all these experiments, we conclude that CPU parallelization is not fast enough for our computation to complete in a timely manner if we are considering many iterations. Similarly, totally relying on GPU is also not possible because the database access can only be done in CPU.

Chapter 6

Alternating Tweedie regression with linear and quadratic constraints

In this Chapter, we consider Alternating Tweedie regression with constraints on parameters to help identifiablity issue. We will consider three different versions of constraints and use Lagrange multiplier approach to estimate the model parameters under the alternating scheme. In one version, we place linear constraint on parameters. In the other two versions, we place quadratic constraints on the model parameters. The reason of considering different versions of constraints is to see how different constraints affect the speed of convergence for model parameter estimates. In classical regression, model estimates with quadratic constraints tend to converge faster than linear constraints. In our alternating setting, we have significant challenge arising from the alternating scheme in high dimensions. The quadratic constraints lead to huge Hessian matrix that are not block diagonal and too large to store in the memory or to compute its inverse. Consequently, we have to design a better strategy to obtain parameter update while using the Hessian matrix. Note that many gradient descent variants (including the popular Adam algorithm) have been serving the purpose of avoiding Hessian matrix in the literature. We choose to not go in that direction because the study in the earlier chapters of this dissertation has confirmed that ignoring the Hessian matrix

leads to much slower convergence of parameter estimates and could not achieve as well in the objective function as the alternating Fisher-Scoring methods we proposed.

In all three subsections of this Chapter, we assume that the data are suitable to be modeled with Tweedie distribution, which has the following canonical form in exponential family format when $Y > 0$:

$$f(y) = \exp\left\{\frac{y\theta - \kappa(\theta)}{\phi} + c(y, \phi, p)\right\},$$

where $\theta = \frac{\mu^{1-p}}{1-p}$ and $\kappa(\theta) = \frac{\mu^{2-p}}{2-p}$. and the $c(y, \phi, p)$ corresponds to the log sum $\log\left(\sum_{n=0}^{\infty} \frac{\beta^{n\alpha}}{\Gamma(n\alpha)} \cdot y^{n\alpha-1} \cdot \frac{\lambda^n}{n!}\right)$ if the data has Compound Poisson distribution.

For Y_{ij} in the i^{th} row and j^{th} column of the co-occurrence matrix, the density function is

$$f(y_{ij}) = \exp\left\{\frac{y_{ij}\theta_{ij} - \kappa(\theta_{ij})}{\phi_{ij}} + c(y_{ij}, \phi_{ij}, p_{ij})\right\},$$

where

$$\theta_{ij} = \frac{\mu_{ij}^{1-p_{ij}}}{1-p_{ij}} = \frac{\exp\left\{(1-p_{ij})\left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j + u\right)\right\}}{(1-p_{ij})},$$

$$\kappa(\theta_{ij}) = \frac{\mu_{ij}^{2-p_{ij}}}{2-p_{ij}} = \frac{\exp\left\{(2-p_{ij})\left(\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j + u\right)\right\}}{(2-p_{ij})},$$

$$c(y_{ij}, \phi_{ij}, p_{ij}) = \log\left(\sum_{k=1}^{\infty} \frac{\beta_{ij}^{k\alpha_{ij}}}{\Gamma(k\alpha_{ij})} \cdot y_{ij}^{k\alpha_{ij}-1} \cdot \frac{\lambda_{ij}^k}{k!}\right), \quad \text{if } 1 < p_{ij} < 2, \tag{6.0.1}$$

$$c(y_{ij}, \phi_{ij}, p_{ij}) = \frac{1}{\pi y_{ij}} \sum_{n=1}^{\infty} \frac{\Gamma(1-\alpha_{ij}n)\phi_{ij}^{n(-\alpha_{ij}-1)}(p_{ij}-1)^{-\alpha_{ij}n}}{(-1)^n \Gamma(1+n)(p_{ij}-2)^n y_{ij}^{-\alpha_{ij}n}} \sin(n\pi\alpha_{ij}), \text{ if } p_{ij} > 2, \tag{6.0.2}$$

$$c(y_{ij}, \phi_{ij}, p_{ij}) = \frac{1}{\pi y_{ij}} \sum_{k=1}^{\infty} \frac{\Gamma\left(1 + \frac{k}{\alpha_{ij}}\right)(-y_{ij})^k}{k! \lambda_{ij}^{\frac{k}{\alpha_{ij}}} \kappa_{p_{ij}}^{\frac{k}{\alpha_{ij}}}(1)} \sin\left(\frac{-k\pi}{\alpha_{ij}}\right), \text{ if } p_{ij} < 0, \tag{6.0.3}$$

with

$$\lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}(2-p_{ij})}, \quad \alpha_{ij} = \frac{2-p_{ij}}{p_{ij}-1}, \quad \frac{1}{\beta_{ij}} = \phi_{ij}(p_{ij}-1)\mu_{ij}^{p_{ij}-1}, \quad \kappa_{p_{ij}}(1) = \frac{(p_{ij}-1)^{-\alpha_{ij}}}{p_{ij}-2}(-1)^{\frac{1}{p_{ij}-1}}.$$

This implies that we will use the log link function in all three subsections:

$$\log \mu_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j + u, \quad i, j = 1, \cdots, n; \quad \mathbf{w}_i, \widetilde{\boldsymbol{w}}_j \in \mathbb{R}^d, b_i, \widetilde{b}_j, u \in \mathbb{R}.$$

While $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{b}}$ are held fixed , the log-likelihood function for the i^{th} row of data limited to positive y_{ij} is

$$\ell_i^{(1)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) = \sum_{j=1}^{n} \left[\frac{y_{ij}\theta_{ij} - \kappa(\theta_{ij})}{\phi_{ij}} + c(y_{ij}, \phi_{ij}, p_{ij}) \right] I(y_{ij} > 0), \tag{6.0.4}$$

and while $\mathbf{w}$ and $\mathbf{b}$ are held fixed, the log-likelihood function for j^{th} column of data limited to the element $y_{ij} > 0$ case is

$$\widetilde{\ell}_j^{(1)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}, \mathbf{b}) = \sum_{i=1}^{n} \left[\frac{y_{ij}\theta_{ij} - \kappa(\theta_{ij})}{\phi_{ij}} + c(y_{ij}, \phi_{ij}, p_{ij}) \right] I(y_{ij} > 0), \tag{6.0.5}$$

where

$$p_{ij} = p_i + \tilde{p}_j, \quad \phi_{ij} = \phi_i \cdot \tilde{\phi}_j.$$

Holding $\widetilde{\boldsymbol{w}}$ and $\widetilde{\boldsymbol{b}}$ fixed, the total log likelihood ℓ for all rows is

$$\begin{aligned}
\ell(\mathbf{w}, \mathbf{b}; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) &= = \sum_{i=1}^{n}\sum_{j=1}^{n} \ell_{ij}(\mathbf{w}, \mathbf{b}; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \{\log f(y_{ij}) \cdot I(y_{ij} > 0) + \log f_0(y_{ij}) \cdot I(y_{ij} = 0)\} \\
&= \sum_{i=1}^{n} \ell_i^{(1)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}) + \sum_{i=1}^{n}\sum_{j=1}^{n} \ell_{ij}^{(0)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}_j, \widetilde{b}_j),
\end{aligned}$$

where

$$\ell_{ij}^{(0)}(\mathbf{w}_i, b_i; \widetilde{\boldsymbol{w}}_j, \widetilde{b}_j) = -\lambda_{ij} I(y_{ij} = 0) \text{ and } \lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}(2 - p_{ij})}.$$

This is because $\Pr[Y_{ij} = 0] = \exp(-\lambda_{ij})$. For $y_{ij} = 0$ case, we also have the log link.

Therefore, $\log \mu_{ij} = \eta_{ij}$ and $\mu_{ij} = \exp(\eta_{ij})$.

Similarly, holding $\mathbf{w}$ and $\mathbf{b}$ fixed, the total log likelihood $\widetilde{\ell}$ for all columns is

$$\widetilde{\ell}(\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}; \mathbf{w}, \mathbf{b}) = \sum_{i=1}^{n}\sum_{j=1}^{n} \widetilde{\ell}_{ij}(\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}; \mathbf{w}, \mathbf{b}) \quad = \quad \sum_{j=1}^{n} \widetilde{\ell}_{j}^{(1)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}, \mathbf{b}) + \sum_{i=1}^{n}\sum_{j=1}^{n} \widetilde{\ell}_{ij}^{(0)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}_i, b_i),$$

where

$$\widetilde{\ell}_{ij}^{(0)}(\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j; \mathbf{w}_i, b_i) = -\lambda_{ij} I\left(y_{ij} = 0\right) \text{ and } \lambda_{ij} = \frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}\left(2-p_{ij}\right)}.$$

Since $\ell_{ij}(\mathbf{w}, \mathbf{b}; \widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}})$ and $\widetilde{\ell}_{ij}(\widetilde{\boldsymbol{w}}, \widetilde{\boldsymbol{b}}; \mathbf{w}, \mathbf{b})$ are referring to the same terms except that they are treated as functions of different parameters, we denote both of them as ℓ_{ij} when it is not necessary to distinguish between the two.

In the next three subsections, we add constraints to the model parameters and outline how to estimate the parameters in each case.

6.1 Alternating Tweedie regression with linear constraints: version 19-4

In this subsection, we consider the log link function and the following linear constraints,

$$\log \mu_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j + u, \quad i, j = 1, \cdots, n; \quad \mathbf{w}_i, \widetilde{\boldsymbol{w}}_j \in \mathbb{R}^d \tag{6.1.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} b_i = 0, \quad \sum_{j=1}^{n} \widetilde{b}_j = 0, \quad \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j = 0. \tag{6.1.2}$$

The decomposition in (6.1.1) mimics the two-way ANOVA. The u, b_i and $\widetilde{b}_j$ represent the overall effect, row main effect, and column main effect, respectively. The $\mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j$ represents the interaction effect between the row and column. In our data setting, it would be the user-item or word-word interaction. Such decomposition was also used in the literature for

recommender systems except that they used least squared method to estimate the model parameters. It is well known that least squares method is suitable for data that have constant variance but not valid if the data have heteroscedasticity. Different from the ANOVA model estimate, here we estimate the unknown quantities under the alternating Tweedie regression framework with the constraints.

Using Lagrange multiplier on all constraints will lead to three Lagrange multiplier coefficients. The objective function including the Lagrange multipliers to estimate these parameters can be written as

$$\begin{aligned} H &= \sum_{i=1}^{n}\sum_{j=1}^{n}\ell_{ij} + \lambda_1\sum_{i=1}^{n}b_i + \lambda_2\sum_{j=1}^{n}\widetilde{b}_j + \lambda_3\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{w}_i^{\top}\widetilde{\boldsymbol{w}}_j \\ &= H_1 + H_2 + H_3 + H_4, \end{aligned}$$

where H_1, H_2, H_3, H_4 refer to the four summations in their order. The first order partial derivatives are

$$\begin{aligned} &\frac{\partial H}{\partial w_{i_1k}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial w_{i_1k}} + \lambda_3\sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_{jk}, \\ &\frac{\partial H}{\partial b_{i_1}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial b_{i_1}} + \lambda_1, \qquad (6.1.3) \\ &\frac{\partial H}{\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial u}, \\ &\frac{\partial H}{\partial \lambda_1} = \sum_{i=1}^{n}b_i, \quad \frac{\partial H}{\partial \lambda_2} = \sum_{j=1}^{n}\widetilde{b}_j, \quad \frac{\partial H}{\partial \lambda_3} = \sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{w}_i^{\top}\widetilde{\boldsymbol{w}}_j, \\ &\frac{\partial H}{\partial \widetilde{w}_{j_1k}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \widetilde{w}_{j_1k}} + \lambda_3\sum_{j=1}^{n}w_{ik}, \\ &\frac{\partial H}{\partial \widetilde{b}_{j_1}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \widetilde{b}_{j_1}} + \lambda_2. \qquad (6.1.4) \end{aligned}$$

To obtain the estimates of λ_3 for given current value of $\widetilde{\boldsymbol{w}}$, note that

$$\frac{\partial H}{\partial \mathbf{w}_{i_1}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \mathbf{w}_{i_1}} + \lambda_3 \sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j, \quad \forall i_1.$$

Setting above partial derivative to zero, we get $\hat{\lambda}_3$ based on one row, which is

$$\hat{\lambda}_{3,i_1} = -\left(\sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j^\top \sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j\right)^{-1}\sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j^\top \sum_{j=1}^{n}\frac{\partial \ell_{i_1 j}}{\partial \mathbf{w}_{i_1}}. \tag{6.1.5}$$

Considering all rows, we obtain the estimate $\hat{\lambda}_{31}$ from combining the row estimates. Similarly, combining the estimates for all columns leads to another estimate $\hat{\lambda}_{32}$. Then we take an average to obtain the $\hat{\lambda}_3$.

$$\hat{\lambda}_{31} = \left(-n\sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j^\top \sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j\right)^{-1}\sum_{j=1}^{n}\widetilde{\boldsymbol{w}}_j^\top \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \mathbf{w}_i}, \tag{6.1.6}$$

$$\hat{\lambda}_{32} = \left(-n\sum_{i=1}^{n}\mathbf{w}_i^\top \sum_{i=1}^{n}\mathbf{w}_i\right)^{-1}\sum_{i=1}^{n}\mathbf{w}_i^\top \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \widetilde{\boldsymbol{w}}_j}, \tag{6.1.7}$$

$$\hat{\lambda}_3 = \frac{\left(\hat{\lambda}_{31} + \hat{\lambda}_{32}\right)}{2}. \tag{6.1.8}$$

Similarly, we obtain the estimate of λ_1 and λ_2 by setting the equations (6.1.3) and (6.1.4) to zero respectively:

$$\hat{\lambda}_1 = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial b_i}, \quad \hat{\lambda}_2 = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \widetilde{b}_j}. \tag{6.1.9}$$

Note that the estimates in equations (6.1.5) - (6.1.9) rely on the current parameter values of $\mathbf{w}$ and $\widetilde{\boldsymbol{w}}$. When $\mathbf{w}$ and $\widetilde{\boldsymbol{w}}$'s estimates are updated, these parameter estimates will need to be updated again.

To estimate other unknown quantities, we consider the second order partial derivatives

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial w_{i_2r}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial w_{i_2r}}, \quad \frac{\partial^2 H}{\partial w_{i_1k}\partial b_{i_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial b_{i_2}},$$
$$\frac{\partial^2 H}{\partial b_{i_1}\partial b_{i_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial b_{i_2}}, \quad \frac{\partial^2 H}{\partial \lambda_1^2} = \frac{\partial^2 H}{\partial \lambda_2^2} = \frac{\partial^2 H}{\partial \lambda_3^2} = 0,$$
$$\frac{\partial^2 H}{\partial u^2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2} \quad \Longrightarrow \quad -E\left[\frac{\partial^2 H}{\partial u^2}\right] = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}},$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}\widetilde{w}_{jk}\cdot I\left(i=i_1\right),$$
$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial u}\right] = \sum_{j=1}^{n}\frac{\mu_{i_1j}^{2-p_{i_1j}}}{\phi_{i_1j}}\cdot\widetilde{w}_{jk},$$
$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_1} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_1} = 0, \quad \frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 l_{ij}}{\partial w_{i_1k}\partial \lambda_2} = 0,$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_3} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_3} + \sum_{j=1}^{n}\widetilde{w}_{jk} = \sum_{j=1}^{n}\widetilde{w}_{jk},$$
$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_3}\right] = -\sum_{j=1}^{n}\widetilde{w}_{jk},$$
$$\frac{\partial^2 H}{\partial b_{i_1}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}\cdot I\left(i=i_1\right),$$
$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial u}\right] = \sum_{j=1}^{n}\frac{\mu_{i_1j}^{2-p_{i_1j}}}{\phi_{i_1j}},$$
$$\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_1} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial \lambda_1} + 1 = 1, \quad \Longrightarrow \quad -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_1}\right] = -1,$$
$$\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_2} = \frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_3} = 0.$$

When treating $\widetilde{\boldsymbol{w}}$, $\widetilde{b}$ as fixed values, we estimate $\mathbf{w}$, b, u, λ_1, and λ_3. The term that contains

λ_2 becomes a constant because this term does not involve $\mathbf{w}$, b, u and should be excluded. In this case, λ_2 cannot be estimated because $\sum_{j=1}^{n} \widetilde{b}_j$ may not be equal to zero for given $\widetilde{b}$. To derive the updating equation, we denote expectation of negative second order derivatives with respect to $\boldsymbol{\beta}$ and $\Lambda = (u, \lambda_1, \lambda_3)^\top$ as R with

$$R = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad \text{and} \quad \boldsymbol{\beta} = \left(\boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top, \cdots, \boldsymbol{\beta}_n^\top\right), \quad \boldsymbol{\beta}_i = \left(\mathbf{w}_i^\top, b_i\right)^\top,$$

where

$$A = \text{Diag}\left[A_i, i = 1, \ldots, n\right], \quad A_i = -E\left\{\sum_{i_1=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 \ell_{i_1 j}}{\partial \boldsymbol{\beta}_i \partial \boldsymbol{\beta}_i^\top}\right\}, \quad C = B^\top \text{ and}$$

$$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} \text{ is a } n(d+1) \times 3 \text{ matrix with blocks } B_i \text{ of dimension } \quad (d+1) \times 3 \quad \text{given by}$$

$$B_i = \begin{bmatrix} -E\left[\frac{\partial^2 H}{\partial \mathbf{w}_i \partial u}\right] & -E\left[\frac{\partial^2 H}{\partial \mathbf{w}_i \partial \lambda_1}\right] & -E\left[\frac{\partial^2 H}{\partial \mathbf{w}_i \partial \lambda_3}\right] \\ -E\left[\frac{\partial^2 H}{\partial b_i \partial u}\right] & -E\left[\frac{\partial^2 H}{\partial b_i \partial \lambda_1}\right] & -E\left[\frac{\partial^2 H}{\partial b_i \partial \lambda_3}\right] \end{bmatrix}, \quad D = \text{Diag}\left\{-E\left[\frac{\partial^2 H}{\partial u^2}\right], 0, 0\right\}_{3\times 3}.$$

A good trait of using this decomposition of R is that A is a block diagonal matrix whose inverse can be obtained as

$$A^{-1} = \text{Diag}\left[A_i^{-1}, i = 1, \ldots, n\right].$$

Even though the dimension of A is too large to be manageable ($n(d+1)$ dimensional square matrix.), A_i is only $(d+1)\times(d+1)$, which could be handled for reasonable vector dimension d.

Let $K = D - CA^{-1}B$. If K is invertible,

then

$$R^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BK^{-1}CA^{-1} & -A^{-1}BK^{-1} \\ -K^{-1}CA^{-1} & K^{-1} \end{bmatrix}.$$

(see Lu and Shiou 2002). Let $U = \begin{bmatrix} U_{\boldsymbol{\beta}} \\ U_{\Lambda} \end{bmatrix}$ be the score vector, where $U_{\boldsymbol{\beta}}$ is $n(d+1)$ dimensional vector, $U_{\Lambda} = \begin{bmatrix} U_u & U_{\lambda_1} & U_{\lambda_3} \end{bmatrix}^{\top}$ is 3×1 vector. Then our updating formula for the model parameter $(\boldsymbol{\beta}^{\top}, \Lambda^{\top})^{\top}$ is

$$\begin{aligned} \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_{t+1} &= \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_{t} + \frac{lr}{t} \mathrm{R}^{-1} U \\ &= \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_{t} + \frac{lr}{t} \begin{bmatrix} A^{-1} + A^{-1}BK^{-1}CA^{-1} & -A^{-1}BK^{-1} \\ -K^{-1}CA^{-1} & K^{-1} \end{bmatrix} \begin{bmatrix} U_{\boldsymbol{\beta}} \\ U_{\Lambda} \end{bmatrix} \\ &= \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_{t} + \frac{lr}{t} \begin{bmatrix} A^{-1}U_{\boldsymbol{\beta}} + A^{-1}BK^{-1}CA^{-1}U_{\boldsymbol{\beta}} - A^{-1}BK^{-1}U_{\Lambda} \\ -K^{-1}CA^{-1}U_{\boldsymbol{\beta}} + K^{-1}U_{\Lambda} \end{bmatrix}. \end{aligned}$$

The advantage of writing this way is that $A^{-1}U_{\boldsymbol{\beta}}$ and $A^{-1}B$ both have much smaller column dimensions. Hence, we could avoid storing A^{-1} but storing $A^{-1}U_{\boldsymbol{\beta}}$ and $A^{-1}B$ instead. The

terms in the updating formula can be computed as follows:

$$A^{-1}B = \begin{bmatrix} A_1^{-1}B_1 \\ A_2^{-1}B_2 \\ \vdots \\ A_n^{-1}B_n \end{bmatrix}, \quad \text{where } A_i^{-1}B_i \quad \text{is a } (d+1)\times 3 \text{ matrix},$$

$$CA^{-1} = \left(A^{-1}B\right)^{\top} = \begin{bmatrix} B_1^{\top}A_1^{-1}, & B_2^{\top}A_2^{-1}, \cdots, B_n^{\top}A_n^{-1} \end{bmatrix},$$

$$CA^{-1}B = \left[B_1^{\top}A_1^{-1}B_1 + B_2^{\top}A_2^{-1}B_2 + \cdots + B_n^{\top}A_n^{-1}B_n\right] \quad \text{is a } 3\times 3 \text{ matrix},$$

$$A^{-1}U_{\boldsymbol{\beta}} = \begin{bmatrix} A_1^{-1}U_{\boldsymbol{\beta}_1} \\ A_2^{-1}U_{\boldsymbol{\beta}_2} \\ \vdots \\ A_n^{-1}U_{\boldsymbol{\beta}_n} \end{bmatrix}, \quad A_i^{-1}U_{\boldsymbol{\beta}_i} \quad \text{is a} \quad (d+1)\times 1\text{vector},$$

$$CA^{-1}U_{\boldsymbol{\beta}} = \sum_{i=1}^{n} B_i^{\top}A_i^{-1}U_{\boldsymbol{\beta}_i} := \Delta, \quad \Delta \quad \text{is a} \quad 3\times 1 \quad \text{vector},$$

$$A^{-1}BK^{-1} = \begin{bmatrix} A_1^{-1}B_1K^{-1} \\ A_2^{-1}B_2K^{-1} \\ \vdots \\ A_n^{-1}B_nK^{-1} \end{bmatrix}, \quad A_i^{-1}B_iK^{-1} \quad \text{is a} \quad (d+1)\times 3 \quad \text{matrix},$$

$$A^{-1}BK^{-1}U_{\Lambda} = \begin{bmatrix} A_1^{-1}B_1K^{-1}U_{\Lambda} \\ A_2^{-1}B_2K^{-1}U_{\Lambda} \\ \vdots \\ A_n^{-1}B_nK^{-1}U_{\Lambda} \end{bmatrix}, \quad A_i^{-1}B_iK^{-1}U_{\Lambda} \quad \text{is a} \quad (d+1)\times 1 \quad \text{vector},$$

$$A^{-1}BK^{-1}CA^{-1}U_{\boldsymbol{\beta}} = \begin{bmatrix} A_1^{-1}B_1K^{-1}\Delta \\ A_2^{+}B_2K^{-1}\Delta \\ \vdots \\ A_n^{-1}B_nK^{-1}\Delta \end{bmatrix}, \quad A_i^{-1}B_iK^{-1}\Delta \quad \text{is} \quad (d+1)\times 1 \quad \text{vector.}$$

Therefore, the first $n(d+1)$ rows of the $R^{-1}U$ is

$$\begin{bmatrix} A_1^{-1}U_{\boldsymbol{\beta}_1} + A_1^{-1}B_1K^{-1}\Delta - A_1^{-1}B_1K^{-1}U_\Lambda \\ A_2^{-1}U_{\boldsymbol{\beta}_2} + A_2^{-1}B_2K^{-1}\Delta - A_2^{-1}B_2K^{-1}U_\Lambda \\ \vdots \\ A_n^{-1}U_{\boldsymbol{\beta}_n} + A_n^{-1}B_nK^{-1}\Delta - A_n^{-1}B_nK^{-1}U_\Lambda \end{bmatrix},$$

and the next three rows of $R^{-1}U$ is $K^{-1}\left(U_\Lambda - \Delta\right)$.

The updating formula for the parameter $\widetilde{\boldsymbol{\beta}}$ follows alternating scheme as in previous chapters. Let $\widetilde{U} = \begin{bmatrix} \widetilde{U}_{\widetilde{\boldsymbol{\beta}}} \\ \widetilde{U}_\Lambda \end{bmatrix}$ where $\widetilde{U}_\Lambda = \begin{bmatrix} \widetilde{U}_u & \widetilde{U}_{\lambda_2} & \widetilde{U}_{\lambda_3} \end{bmatrix}^\top$, $\widetilde{U}_{\widetilde{\boldsymbol{\beta}}}$ is $n(d+1)\times 1$ vector, $\widetilde{U}_\Lambda$ is 3×1 vector. Then our updating formula for the model parameter $\left(\widetilde{\boldsymbol{\beta}}^\top, \widetilde{\Lambda}^\top\right)^\top$ is

$$\begin{aligned} \begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_{t+1} &= \begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_t + \frac{lr}{t}\widetilde{R}^{-1}\widetilde{U} \\ &= \begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_t + \frac{lr}{t}\begin{bmatrix} \widetilde{A}^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} + \widetilde{A}^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{C}\widetilde{A}^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} - \widetilde{A}^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{U}_\Lambda \\ -\widetilde{K}^{-1}\widetilde{C}\widetilde{A}^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} + \widetilde{K}^{-1}\widetilde{U}_\Lambda \end{bmatrix}, \end{aligned}$$

with the first $n(d+1)$ rows given by

$$\widehat{\widetilde{\boldsymbol{\beta}}}_t + \frac{lr}{t}\begin{bmatrix} \widetilde{A}_1^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}_1} + \widetilde{A}_1^{-1}\widetilde{B}_1\widetilde{K}^{-1}\widetilde{\Delta} - \widetilde{A}_1^{-1}\widetilde{B}_1\widetilde{K}^{-1}\widetilde{U}_\Lambda \\ \widetilde{A}_2^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}_2} + \widetilde{A}_2^{-1}\widetilde{B}_2\widetilde{K}^{-1}\widetilde{\Delta} - \widetilde{A}_2^{-1}\widetilde{B}_2\widetilde{K}^{-1}\widetilde{U}_\Lambda \\ \vdots \\ \widetilde{A}_n^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}_n} + \widetilde{A}_n^{-1}\widetilde{B}_n\widetilde{K}^{-1}\widetilde{\Delta} - \widetilde{A}_n^{-1}\widetilde{B}_n\widetilde{K}^{-1}\widetilde{U}_\Lambda \end{bmatrix},$$

and the last three rows given by $\widehat{\widetilde{\Lambda}}_t + \frac{lr}{t}\widetilde{K}^{-1}\left(\widetilde{U}_\Lambda - \widetilde{\Delta}\right)$, where

$$\widetilde{A}_i^{-1} = -\left[E\left\{\sum_{i_1=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{i_1 j}}{\partial\widetilde{\boldsymbol{\beta}}_{i_1}\partial\widetilde{\boldsymbol{\beta}}_{i_1}^\top}\right\}\right]^{-1}, \quad \widetilde{B}_i = \begin{bmatrix} -E\left[\frac{\partial^2 H}{\partial\widetilde{\boldsymbol{\beta}}_i\partial u}\right] & -E\left[\frac{\partial^2 H}{\partial\widetilde{\boldsymbol{\beta}}_i\partial\lambda_1}\right] & -E\left[\frac{\partial^2 H}{\partial\widetilde{\boldsymbol{\beta}}_i\partial\lambda_3}\right] \end{bmatrix},$$

$$D = \text{Diag}\left\{-E\left[\frac{\partial^2 H}{\partial u^2}\right], 0, 0\right\}_{3\times 3}, \quad \widetilde{K} = D - \sum_{i=1}^{n}\widetilde{B}_i^\top\widetilde{A}_i^{-1}\widetilde{B}_i, \quad \widetilde{\Delta} = \sum_{i=1}^{n}\widetilde{B}_i^\top\widetilde{A}_i^{-1}\widetilde{U}_{\boldsymbol{\beta}_i}.$$

The computation of K^{-1} and Δ requires knowing the value of A_i, B_i, and $U_{\boldsymbol{\beta}_i}$ from all rows $i = 1, \ldots, n$. As a result, the parameter update for $\boldsymbol{\beta}$ and Λ couldn't be done row by row. Instead, the algorithm has to go through all rows of data before one parameter update can be conducted. Similarly, computation of $\widetilde{K}^{-1}$ and $\widetilde{\Delta}$ will need to wait until all $\widetilde{A}_j$, $\widetilde{B}_j$, and $\widetilde{U}_{\widetilde{\boldsymbol{\beta}}_i}$ from all columns are available $j = 1\ldots, n$. Consequently, $\widetilde{\boldsymbol{\beta}}$ and $\widetilde{\Lambda}$ are updated only once after all columns of the data have been visited. Due to this restriction along with the fact that data retrieval from the database has to be done one row at a time, the memory demand becomes a significant challenge in the computation.

6.2 Alternating Tweedie regression with quadratic constraints: version 19-5

In this subsection, we consider Alternating Tweedie regression with the log link function and the following quadratic constraints:

$$\begin{aligned}
&\log \mu_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j + u, \quad i,j = 1, \cdots, n; \quad \mathbf{w}_i, \widetilde{\boldsymbol{w}}_j \in \mathbb{R}^d \\
&\text{s.t.} \quad \left(\sum_{i=1}^n b_i\right)^2 = 0, \quad \left(\sum_{j=1}^n \widetilde{b}_j\right)^2 = 0, \quad \sum_{i=1}^n M_i^2 = 0, \quad \sum_{j=1}^n \widetilde{M}_j^2 = 0, \\
&\text{where} \quad M_i = \sum_{j=1}^n \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j, \quad \text{and} \quad \widetilde{M}_j = \sum_{i=1}^n \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j.
\end{aligned}$$

Mathematically, these constraints are identical to the following constraints which are identical to those used in two-way ANOVA:

$$\sum_{i=1}^n b_i = 0, \ \sum_{j=1}^n \widetilde{b}_j = 0, \ \ M_i = 0, \ \ \forall i, \ \ \widetilde{M}_j = 0 \ \ \forall j.$$

Including them in quadratic forms in the objective function will allow us to restrict the Lagrange multipliers to be negative. Specifically, using Lagrange multiplier on all constraints will lead to four Lagrange multiplier coefficients. The objective function using Lagrange multiplier to estimate these parameters is

$$\begin{aligned}
H^* &= \sum_{i=1}^n \sum_{j=1}^n \ell_{ij} + \frac{\lambda_1}{2}\left(\sum_{i=1}^n b_i\right)^2 + \frac{\lambda_2}{2}\left(\sum_{j=1}^n \widetilde{b}_j\right)^2 + \frac{\lambda_4}{2}\sum_{j=1}^n \widetilde{M}_j^2 + \frac{\lambda_5}{2}\sum_{i=1}^n M_i^2 \\
&= H_1 + H_2 + H_3 + H_5 + H_6,
\end{aligned}$$

where λ_1, λ_2, λ_4, $\lambda_5 < 0$ and H_1 to H_6 correspond to the five summations in the definition. Note that M_i and $\widetilde{M}_j$ are both functions of $\mathbf{w}$ and $\widetilde{\boldsymbol{w}}$, therefore H_5 and H_6 would both show up in the optimization process.

The first order partial derivatives from H_5 and H_6 are

$$\frac{\partial H_5}{\partial w_{i_1k}} = \frac{\partial H_5}{\partial \widetilde{M}_j} \cdot \frac{\partial \widetilde{M}_j}{\partial w_{i_1k}} = \lambda_4 \sum_{j=1}^{n} \widetilde{M}_j \cdot \widetilde{w}_{jk},$$
$$\frac{\partial H_6}{\partial w_{i_1k}} = \frac{\partial H_6}{\partial M_i} \cdot \frac{\partial M_i}{\partial w_{i_1k}} = \lambda_5 \sum_{i=1}^{n} M_i \cdot \sum_{j=1}^{n} \widetilde{w}_{jk} \cdot I\left(i = i_1\right) = \lambda_5 M_{i_1} \sum_{j=1}^{n} \widetilde{w}_{jk},$$

$$\frac{\partial H_5}{\partial \widetilde{w}_{j_1k}} = \frac{\partial H_5}{\partial \widetilde{M}_j} \cdot \frac{\partial \widetilde{M}_j}{\partial \widetilde{w}_{j_1k}} = \lambda_4 \sum_{j=1}^{n} \widetilde{M}_j \cdot \sum_{i=1}^{n} w_{ik} \cdot I\left(j = j_1\right) = \lambda_4 \widetilde{M}_{j_1} \sum_{i=1}^{n} w_{ik},$$
$$\frac{\partial H_6}{\partial \widetilde{w}_{j_1k}} = \frac{\partial H_6}{\partial M_i} \cdot \frac{\partial M_i}{\partial \widetilde{w}_{j_1k}} = \lambda_5 \sum_{i=1}^{n} M_i \cdot w_{ik}.$$

Hence the first order partial derivatives of H can be written as

$$\begin{aligned}
&\frac{\partial H}{\partial w_{i_1k}} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial w_{i_1k}} + \lambda_4 \sum_{j=1}^{n} \widetilde{M}_j \widetilde{w}_{jk} + \lambda_5 M_{i_1} \sum_{j=1}^{n} \widetilde{w}_{jk}, \\
&\frac{\partial H}{\partial b_{i_1}} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial b_{i_1}} + \lambda_1 \sum_{i=1}^{n} b_i, \\
&\frac{\partial H}{\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial u}, \\
&\frac{\partial H}{\partial \lambda_1} = \frac{1}{2}\left(\sum_{i=1}^{n} b_i\right)^2, \qquad \frac{\partial H}{\partial \lambda_4} = \frac{1}{2}\sum_{j=1}^{n} \widetilde{M}_j^2, \qquad \frac{\partial H}{\partial \lambda_5} = \frac{1}{2}\sum_{i=1}^{n} M_i^2.
\end{aligned} \tag{6.2.1}$$

$$\frac{\partial H}{\partial \widetilde{w}_{j_1k}} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \widetilde{w}_{j_1k}} + \lambda_4 \widetilde{M}_{j_1} \sum_{i=1}^{n} w_{ik} + \lambda_5 \sum_{i=1}^{n} M_i \cdot w_{ik},$$
$$\frac{\partial H}{\partial \widetilde{b}_{j_1}} = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \widetilde{b}_{j_1}} + \lambda_2 \sum_{j=1}^{n} \widetilde{b}_j.$$

The formulae for estimating $\hat{\lambda}_1$, $\hat{\lambda}_2$, $\hat{\lambda}_4$, $\hat{\lambda}_5$ can be directly obtained from setting the first derivatives to zero as in section 6.1. Specifically, setting some of the first order derivatives

to zero gives us

$$\frac{\partial H}{\partial \mathbf{w}_i} = \sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \mathbf{w}_i} + \hat{\lambda}_4 \sum_{j=1}^{n} \widetilde{M}_j \widetilde{\boldsymbol{w}}_j + \hat{\lambda}_5 \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j M_i = 0, \tag{6.2.2}$$

$$\frac{\partial H}{\partial \widetilde{\boldsymbol{w}}_j} = \sum_{i=1}^{n} \frac{\partial \ell_{ij}}{\partial \widetilde{\boldsymbol{w}}_j} + \hat{\lambda}_4 \widetilde{M}_j \sum_{i=1}^{n} \mathbf{w}_i + \hat{\lambda}_5 \sum_{i=1}^{n} M_i \mathbf{w}_i = 0. \tag{6.2.3}$$

(6.2.2) and (6.2.3) hold for all i and j. Therefore, we can sum up all rows and columns to obtain the following equations:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \mathbf{w}_i} + \hat{\lambda}_4 n \sum_{j=1}^{n} \widetilde{M}_j \widetilde{\boldsymbol{w}}_j + \hat{\lambda}_5 \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j \sum_{i=1}^{n} M_i = 0, \tag{6.2.4}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \widetilde{\boldsymbol{w}}_j} + \hat{\lambda}_4 \sum_{j=1}^{n} \widetilde{M}_j \sum_{i=1}^{n} \mathbf{w}_i + \hat{\lambda}_5 n \sum_{i=1}^{n} M_i \mathbf{w}_i = 0. \tag{6.2.5}$$

Denote

$$a_1 = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \mathbf{w}_i}, \quad b_1 = n \sum_{j=1}^{n} \widetilde{M}_j \widetilde{\boldsymbol{w}}_j, \quad c_1 = \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j \sum_{i=1}^{n} M_i,$$

$$a_2 = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \widetilde{\boldsymbol{w}}_j}, \quad b_2 = \sum_{j=1}^{n} \widetilde{M}_j \sum_{j=1}^{n} \mathbf{w}_i, \quad c_2 = n \sum_{i=1}^{n} M_1 \cdot \mathbf{w}_i.$$

Then the equations (6.2.4) and (6.2.5) can be rewritten as

$$a_1 + b_1 \hat{\lambda}_4 + c_1 \hat{\lambda}_5 = 0$$

$$a_2 + b_2 \hat{\lambda}_4 + c_2 \hat{\lambda}_5 = 0$$

We solve the equations for $\hat{\lambda}_4$ and $\hat{\lambda}_5$ as follow.

$$b_2^\top a_1 + b_2^\top b_1 \hat{\lambda}_4 + b_2^\top c_1 \hat{\lambda}_5 = 0 \tag{6.2.6}$$

$$b_1^\top a_2 + b_1^\top b_2 \hat{\lambda}_4 + b_1^\top c_2 \hat{\lambda}_5 = 0 \tag{6.2.7}$$

Subtracting (6.2.7) from (6.2.6) provides $b_2^\top a_1 - b_1^\top a_2 = \left(b_1^\top c_2 - b_2^\top c_1\right)\hat{\lambda}_5$. So that

$$\hat{\lambda}_5 = \frac{b_2^\top a_1 - b_1^\top a_2}{b_1^\top c_2 - b_2^\top c_1}.$$

Similarly the $\hat{\lambda_4}$ can be obtained as

$$\begin{cases} c_2^\top a_1 + c_2^\top b_1 \hat{\lambda}_4 + c_2^\top c_1 \hat{\lambda}_5 = 0 \\ c_1^\top a_2 + c_1^\top b_2 \hat{\lambda}_4 + c_1^\top c_2 \hat{\lambda}_5 = 0 \end{cases} \quad \Longrightarrow \quad \hat{\lambda}_4 = \frac{c_1^\top a_2 - c_2^\top a_1}{c_2^\top b_1 - c_1^\top b_2}.$$

For $\hat{\lambda_1}$, equation (6.2.1) can be summed up for all rows. Then it can be solved for $\hat{\lambda}_1$ as

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial b_i} = -\hat{\lambda}_1 n \sum_{i=1}^{n} b_i \quad \Longrightarrow \quad \hat{\lambda}_1 = -\left(n\sum_{i=1}^{n} b_i\right)^{-1}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial b_i}.$$

In a similar way,

$$\hat{\lambda}_2 = -\left(n\sum_{i=1}^{n} \widetilde{b}_j\right)^{-1}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial \ell_{ij}}{\partial \widetilde{b}_j}.$$

The estimate of other unknown quantities will need the alternating scheme. Next we describe the details and challenges in estimating the other parameters with the alternating

Fisher scoring algorithm. Consider the second order partial derivatives

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial w_{i_2r}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial w_{i_2r}} + \lambda_4\sum_{j=1}^{n}\widetilde{w}_{jr}\widetilde{w}_{jk} + \lambda_5\sum_{j=1}^{n}\widetilde{w}_{jr}\sum_{j=1}^{n}\widetilde{w}_{jk}\cdot I(i_1=i_2),$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial w_{i_2r}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial w_{i_2r}}\right] - \lambda_4\sum_{j=1}^{n}\widetilde{w}_{jr}\widetilde{w}_{jk} - \lambda_5\sum_{j=1}^{n}\widetilde{w}_{jr}\sum_{j=1}^{n}\widetilde{w}_{jk}\cdot I(i_1=i_2).$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial b_{i_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial b_{i_2}} \quad \Longrightarrow \quad -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial b_{i_2}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial b_{i_2}}\right].$$

$$\frac{\partial^2 H}{\partial b_{i_1}\partial b_{i_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial b_{i_2}} + \lambda_1 \quad \Longrightarrow \quad -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial b_{i_2}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial b_{i_2}}\right] - \lambda_1.$$

$$\frac{\partial^2 H}{\partial u^2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2} \quad \Longrightarrow \quad -E\left[\frac{\partial^2 H}{\partial u^2}\right] = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}.$$

$$\frac{\partial^2 H}{\partial \lambda_1^2} = \frac{\partial^2 H}{\partial \lambda_2^2} = \frac{\partial^2 H}{\partial \lambda_4^2} = \frac{\partial^2 H}{\partial \lambda_5^2} = 0, \qquad \frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_2} = \frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_4} = \frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_5} = 0.$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_1} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_1} = 0, \qquad \frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 l_{ij}}{\partial w_{i_1k}\partial \lambda_2} = 0.$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}\widetilde{w}_{jk}\cdot I\left(i=i_1\right),$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial u}\right] = \sum_{j=1}^{n}\frac{\mu_{i_1j}^{2-p_{i_1j}}}{\phi_{i_1j}}\cdot\widetilde{w}_{jk}.$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_4} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_4} + \sum_{j=1}^{n}\widetilde{M}_j\widetilde{w}_{jk} = \sum_{j=1}^{n}\widetilde{M}_j\widetilde{w}_{jk},$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_4}\right] = -\sum_{j=1}^{n}\widetilde{M}_j\widetilde{w}_{jk}.$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_5} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_5} + M_{i_1}\sum_{j=1}^{n}\widetilde{w}_{jk} = M_{i_1}\sum_{j=1}^{n}\widetilde{w}_{jk},$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_5}\right] = -M_{i_1}\sum_{j=1}^{n}\widetilde{w}_{jk}.$$

$$\frac{\partial^2 H}{\partial b_{i_1}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}\cdot I\left(i=i_1\right),$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial u}\right] = \sum_{j=1}^{n}\frac{\mu_{i_1 j}^{2-p_{i_1 j}}}{\phi_{i_1 j}}.$$

$$\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_1} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial \lambda_1} + \sum_{i=1}^{n} b_i = \sum_{i=1}^{n} b_i,$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_1}\right] = -\sum_{i=1}^{n} b_i.$$

To derive the updating equations, denote

$$R = \begin{bmatrix} A_n^* & B \\ C & D \end{bmatrix}, \quad \boldsymbol{\beta}_i = \left(\mathbf{w}_i^\top, b_i\right)^\top, \quad \boldsymbol{\beta} = \left(\boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top, \cdots, \boldsymbol{\beta}_n^\top\right).$$

where

$$A_n^* = A + I_n \otimes S + \left(\mathbf{1}_n\mathbf{1}_n^\top \otimes T\right)$$

where $\mathbf{1}_n$ is a column vector of 1s, S has $(r,k)^{th}$ entry

$$s_{rk} = \begin{cases} \lambda_5 \sum_{j=1}^{n} \widetilde{w}_{jr} \sum_{j=1}^{n} \widetilde{w}_{jk} & \text{for } r = 1,\ldots,d; k = 1,\ldots,d\ , \\ 0 & \text{if } r = d+1 \text{ or } k = d+1. \end{cases}$$

T has $(r,k)^{th}$ entry

$$t_{rk} = \begin{cases} \lambda_4 \sum_{j=1}^{n} \widetilde{w}_{jr}\widetilde{w}_{jk} & \text{for } r = 1,\ldots,d; k = 1,\ldots,d, \\ 0 & \text{if } \max(r,k) = d+1 \text{ and } \min(r,k) < d+1, \\ \lambda_1 & \text{if } r = k = d+1, \end{cases}$$

with

$$
\begin{aligned}
&A = \text{Diag}\left[A_i, i = 1, \ldots, n\right] \text{ being a } n(d+1) \quad \text{dimensional square matrix, where} \\
&A_i = -E\left\{\sum_{i_1=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{i_1 j}}{\partial \boldsymbol{\beta}_i \partial \boldsymbol{\beta}_i^\top}\right\}, \qquad A^{-1} = \text{Diag}\left[A_i^{-1}, i = 1, \ldots, n\right], \\
&B = \left[B_1^T, \quad B_2^T, \quad \cdots \quad, B_n^T\right]^T \quad \text{being an } n(d+1) \times 4 \text{ matrix with} \\
&B_i = \left[-E\left[\frac{\partial^2 H}{\partial \boldsymbol{\beta}_i \partial u}\right] \qquad - E\left[\frac{\partial^2 H}{\partial \boldsymbol{\beta}_i \partial \lambda_1}\right] \qquad - E\left[\frac{\partial^2 H}{\partial \boldsymbol{\beta}_i \partial \lambda_4}\right] \qquad - E\left[\frac{\partial^2 H}{\partial \boldsymbol{\beta}_i \partial \lambda_5}\right]\right], \\
&C = B^\top \text{ and } D = \text{Diag}\left\{-E\left[\frac{\partial^2 H}{\partial u^2}\right], 0, 0\right\}_{3\times 3}.
\end{aligned}
$$

Note that A_n^* is not a block diagonal matrix because it can be written as

$$
A_n^* = \begin{bmatrix}
A_1 + S + T & T & \cdots & T \\
T & A_2 + S + T & & \vdots \\
\vdots & & \ddots & T \\
T & \cdots & T & A_n + S + T
\end{bmatrix} = \begin{bmatrix}
A_{n-1}^* & \check{T}_{n-1} \\
\check{T}_{n-1}^\top & D_n
\end{bmatrix},
$$

where $\check{T}_{n-1} = \mathbf{1}_{n-1} \otimes T$. Computing $(A_n^*)^{-1}$ is computationally challenging and infeasible because the dimension of A_n^* is $n(d+1) \times n(d+1)$.

To compute $R^{-1}U$, we will see that we need $(A_n^*)^{-1}U_{\boldsymbol{\beta}}$ and $(A_n^*)^{-1}B$ where $U_{\boldsymbol{\beta}}$ and B are defined soon. Instead of computing $(A_n^*)^{-1}$ directly, we iteratively/incrementally compute $(A_n^*)^{-1}U_{\boldsymbol{\beta}}$ and $(A_n^*)^{-1}B$ without actually storing/loading $(A_n^*)^{-1}$. That is, we loop through the rows in iterations. For the m^{th} iteration, we compute $(A_m^*)^{-1}U_m$ and $(A_m^*)^{-1}\check{T}_m$ via $(A_m^*)^{-1}U_{m-1}$ and $(A_m^*)^{-1}\check{T}_{m-1}$ obtained from the previous iteration, where $U_m = \left[U_{\boldsymbol{\beta}_1}^\top, U_{\boldsymbol{\beta}_2}^\top, \ldots, U_{\boldsymbol{\beta}_m}^\top\right]$, $\check{T}_m = \mathbf{1}_m \otimes T$, for $m \geq 2$. This computation will rely on the inverse of 2×2 block matrices.

From Lu and Shiou 2002, let $K = D - C(A_n^*)^{-1}B$. If K is invertible, then

$$R^{-1} = \begin{bmatrix} (A_n^*)^{-1} + (A_n^*)^{-1}BK^{-1}C(A_n^*)^{-1} & -(A_n^*)^{-1}BK^{-1} \\ -K^{-1}C(A_n^*)^{-1} & K^{-1} \end{bmatrix}.$$

Let $U = \begin{bmatrix} U_{\boldsymbol{\beta}} \\ U_\Lambda \end{bmatrix}$ where $U_\Lambda = \begin{bmatrix} U_u & U_{\lambda_1} & U_{\lambda_4} & U_{\lambda_5} \end{bmatrix}^\top$, $U_{\boldsymbol{\beta}}$ is $n(d+1) \times 1$ vector, U_Λ is 4×1 vector. Then our updating formula for the parameter $\left(\boldsymbol{\beta}^\top, \Lambda^\top\right)^\top$ is

$$\begin{aligned}
\begin{bmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{bmatrix}_{t+1} &= \begin{bmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{bmatrix}_t + \frac{lr}{t} \mathrm{R}^{-1} U \\
&= \begin{bmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{bmatrix}_t + \frac{lr}{t} \begin{bmatrix} (A_n^*)^{-1} + (A_n^*)^{-1}BK^{-1}C(A_n^*)^{-1} & -(A_n^*)^{-1}BK^{-1} \\ -K^{-1}C(A_n^*)^{-1} & K^{-1} \end{bmatrix} \begin{bmatrix} U_{\boldsymbol{\beta}} \\ U_\Lambda \end{bmatrix} \\
&= \begin{bmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{bmatrix}_t + \frac{lr}{t} \begin{bmatrix} (A_n^*)^{-1}U_{\boldsymbol{\beta}} + (A_n^*)^{-1}BK^{-1}C(A_n^*)^{-1}U_{\boldsymbol{\beta}} - (A_n^*)^{-1}BK^{-1}U_\Lambda \\ -K^{-1}C(A_n^*)^{-1}U_{\boldsymbol{\beta}} + K^{-1}U_\Lambda \end{bmatrix}.
\end{aligned}$$

Each component of the above updating formula can be rewritten as below counting on our incremental computation strategy. Denote

$$A_n^* = \begin{bmatrix} A_{n-1}^* & \check{T}_{n-1} \\ \check{T}_{n-1}^\top & D_n \end{bmatrix}, \quad B = \begin{bmatrix} \check{B}_{n-1} \\ B_n \end{bmatrix}, \quad U_{\boldsymbol{\beta}} = \begin{bmatrix} \check{U}_{n-1} \\ U_{\boldsymbol{\beta}_n} \end{bmatrix}, \quad \check{D}_n = \left(D_n - \check{T}_{n-1}^\top \left(A_{n-1}^*\right)^{-1} \check{T}_{n-1}\right)^{-1},$$

where $\check{T}_{n-1} = \mathbf{1}_{n-1} \otimes T$, $\check{B}_{n-1} = \left(B_1^\top, B_2^\top, \ldots, B_{n-1}^\top\right)^\top$, $\check{U}_{n-1} = \left(U_{\boldsymbol{\beta}_1}^\top, U_{\boldsymbol{\beta}_2}^\top, \ldots, U_{\boldsymbol{\beta}_{n-1}}^\top\right)^\top$. The key components in the updating formula is $(A_n^*)^{-1}B$, $(A_n^*)^{-1}U_{\boldsymbol{\beta}}$, and K. Note that K can only be computed when all the calculations up to the last row n are completed since it

involves $(A_n^*)^{-1}$ which is obtained by incrementally.

$$
\begin{aligned}
(A_n^*)^{-1}B &= \begin{bmatrix} \big(A_{n-1}^*\big)^{-1} + \big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1} & -\big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_n \\ -\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1} & \check{D}_n \end{bmatrix}\begin{bmatrix} \check{B}_{n-1} \\ B_n \end{bmatrix} \\
&= \begin{bmatrix} \big(A_{n-1}^*\big)^{-1}\check{B}_{n-1} + \big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1}\check{B}_{n-1} - \big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_nB_n \\ -\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1}\check{B}_{n-1} + \check{D}_nB_n \end{bmatrix}, \\
(A_n^*)^{-1}U_{\boldsymbol{\beta}} &= \begin{bmatrix} \big(A_{n-1}^*\big)^{-1} + \big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1} & -\big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_n \\ -\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1} & \check{D}_n \end{bmatrix}\begin{bmatrix} \check{U}_{n-1} \\ U_{\boldsymbol{\beta}_n} \end{bmatrix} \\
&= \begin{bmatrix} \big(A_{n-1}^*\big)^{-1}\check{U}_{n-1} + \big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1}\check{U}_{n-1} - \big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}\check{D}_nU_{\boldsymbol{\beta}_n} \\ -\check{D}_n\check{T}_{n-1}^\top\big(A_{n-1}^*\big)^{-1}\check{U}_{n-1} + \check{D}_nU_{\boldsymbol{\beta}_n} \end{bmatrix}.
\end{aligned}
$$

Note that in the $(A_n^*)^{-1}B$ and $(A_n^*)^{-1}U_{\boldsymbol{\beta}}$, $\big(A_{n-1}^*\big)^{-1}\check{T}_{n-1}$, $\big(A_{n-1}^*\big)^{-1}\check{B}_{n-1}$, $\big(A_{n-1}^*\big)^{-1}\check{U}_{n-1}$, and $\check{D}_n$ are computed from previous iteration incrementally. Denote

$$
\begin{aligned}
&A_m^* = \begin{bmatrix} A_{m-1}^* & \check{T}_{m-1} \\ \check{T}_{m-1}^\top & D_m \end{bmatrix}, \quad \check{B}_m = \begin{pmatrix} \check{B}_{m-1} \\ B_i \end{pmatrix}, \quad D_m = A_m + S + T, \\
&\check{T}_{m-1} = \mathbf{1}_m \otimes T, \quad \check{D}_m = \Big(D_m - \check{T}_{m-1}^\top\big(A_{m-1}^*\big)^{-1}\check{T}_{m-1}\Big)^{-1},
\end{aligned}
$$

for $m = 2, \ldots, n$ and set $\check{T}_1 = T$, $\check{B}_1 = B_1$. Then

$$
\begin{aligned}
(A^*_m)^{-1}\check{T}_m &= \begin{bmatrix} \left(A^*_{m-1}\right)^{-1} + \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1} & -\left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m \\ -\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1} & \check{D}_m \end{bmatrix}\begin{bmatrix} \check{T}_{m-1} \\ T \end{bmatrix} \\
&= \begin{bmatrix} \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1} + \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1}\check{T}_{m-1} - \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m T \\ -\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1}\check{T}_{m-1} + \check{D}_m T \end{bmatrix}, \\
(A^*_m)^{-1}\check{B}_m &= \begin{bmatrix} \left(A^*_{m-1}\right)^{-1} + \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1} & -\left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m \\ -\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1} & \check{D}_m \end{bmatrix}\begin{bmatrix} \check{B}_{m-1} \\ B_m \end{bmatrix} \\
&= \begin{bmatrix} \left(A^*_{m-1}\right)^{-1}\check{B}_{m-1} + \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1}\check{B}_{m-1} - \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m B_m \\ -\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1}\check{B}_{m-1} + \check{D}_m B_m \end{bmatrix}, \\
(A^*_m)^{-1}\check{U}_m &= \begin{bmatrix} \left(A^*_{m-1}\right)^{-1} + \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1} & -\left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m \\ -\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1} & \check{D}_m \end{bmatrix}\begin{bmatrix} \check{U}_{m-1} \\ U_{\boldsymbol{\beta}_m} \end{bmatrix} \\
&= \begin{bmatrix} \left(A^*_{m-1}\right)^{-1}\check{U}_{m-1} + \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1}\check{U}_{m-1} - \left(A^*_{m-1}\right)^{-1}\check{T}_{m-1}\check{D}_m U_{\boldsymbol{\beta}_m} \\ -\check{D}_m\check{T}^\top_{m-1}\left(A^*_{m-1}\right)^{-1}\check{U}_{m-1} + \check{D}_m U_{\boldsymbol{\beta}_m} \end{bmatrix}.
\end{aligned}
$$

Similarly, we could apply alternating and incremental scheme to update the parameter $\widetilde{\boldsymbol{\beta}}$. Let $\widetilde{U} = \begin{bmatrix} \widetilde{U}_{\widetilde{\boldsymbol{\beta}}} \\ \widetilde{U}_\Lambda \end{bmatrix}$, where $\widetilde{U}_\Lambda = \begin{bmatrix} \widetilde{U}_u & \widetilde{U}_{\lambda_2} & \widetilde{U}_{\lambda_4} & \widetilde{U}_{\lambda_5} \end{bmatrix}^\top$, $\widetilde{U}_{\widetilde{\boldsymbol{\beta}}}$ is $n(d+1) \times 1$ vector, and $\widetilde{U}_\Lambda$ is 4×1 vector. Then our updating formula for the parameter $\left(\widetilde{\boldsymbol{\beta}}^\top, \widetilde{\Lambda}^\top\right)^\top$ is

$$
\begin{bmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{bmatrix}_{t+1} = \begin{bmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{bmatrix}_t + \frac{lr}{t}\begin{bmatrix} \left(\widetilde{A}^*_n\right)^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} + \left(\widetilde{A}^*_n\right)^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{C}\left(\widetilde{A}^*_n\right)^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} - \left(\widetilde{A}^*_n\right)^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{U}_\Lambda \\ -\widetilde{K}^{-1}\widetilde{C}\left(\widetilde{A}^*_n\right)^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} + \widetilde{K}^{-1}\widetilde{U}_\Lambda \end{bmatrix}.
$$

$\left(\widetilde{A}^*_n\right)^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}}$ and $\left(\widetilde{A}^*_n\right)^{-1}\widetilde{B}$ both need to be incrementally computed. Their row dimension increases by $d+1$ for each iteration. $\widetilde{K}^{-1}$ has to be computed in the end after all columns have been visited. The incremental nature requires much large memory usage compared to the algorithm in section 6.1.

6.3 Alternating Tweedie regression with quadratic constraints: version 19-6

In previous section, one serious obstacle is that the T submatrix appears in all blocks, including diagonal and off-diagonal blocks. Here we consider different objective functions as the subtasks of the alternating scheme under the same settings as in previous section 6.2. That is, assuming the log link function and the following constraints:

$$\begin{aligned}
&\log \mu_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j + u, \quad i,j = 1, \cdots, n; \quad \mathbf{w}_i, \widetilde{\boldsymbol{w}}_j \in \mathbb{R}^d \\
&\text{s.t.} \quad \left(\sum_{i=1}^n b_i\right)^2 = 0, \quad \left(\sum_{j=1}^n \widetilde{b}_j\right)^2 = 0, \quad \sum_{i=1}^n M_i^2 = 0, \quad \sum_{j=1}^n \widetilde{M}_j^2 = 0, \\
&\text{where} \quad M_i = \sum_{j=1}^n \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j, \quad \text{and} \quad \widetilde{M}_j = \sum_{i=1}^n \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j.
\end{aligned}$$

Instead of using Lagrange multiplier on all constraints in the overall objective function

$$\begin{aligned}
H^* &= \sum_{i=1}^n \sum_{j=1}^n \ell_{ij} + \frac{\lambda_1}{2}\left(\sum_{i=1}^n b_i\right)^2 + \frac{\lambda_2}{2}\left(\sum_{j=1}^n \widetilde{b}_j\right)^2 + \frac{\lambda_3}{2}\sum_{j=1}^n M_i^2 + \frac{\lambda_4}{2}\sum_{i=1}^n \widetilde{M}_j^2 \\
&= H_0 + H_1 + H_2 + H_3 + H_4,
\end{aligned}$$

where λ_1, λ_2, λ_3, $\lambda_4 < 0$, we consider only part of the objective function during each stage of the alternating scheme. Specifically, let

$$H = H_0 + H_1 + H_3 \quad \text{and} \quad \widetilde{H} = H_0 + H_2 + H_4.$$

Then, we alternately optimize H with respect to $\mathbf{w}_i, b_i, u, \lambda_1, \lambda_3$ and $\widetilde{H}$ with respect to

$\widetilde{\boldsymbol{w}}_j, \widetilde{b}_j, u, \lambda_2, \lambda_4$. For computing the score vector,

$$\frac{\partial H_3}{\partial w_{i_1 k}} = \frac{\partial H_3}{\partial M_i} \cdot \frac{\partial M_i}{\partial w_{i_1 k}} = \lambda_3 \sum_{j=1}^{n} M_i \sum_{j=1}^{n} \widetilde{w}_{jk} \cdot I(i = i_1) = \lambda_3 M_{i_1} \sum_{j=1}^{n} \widetilde{w}_{jk},$$
$$\frac{\partial H_4}{\partial \widetilde{w}_{j_1 k}} = \frac{\partial H_4}{\partial \widetilde{M}_j} \cdot \frac{\partial \widetilde{M}_j}{\partial \widetilde{w}_{j_1 k}} = \lambda_4 \sum_{i=1}^{n} \widetilde{M}_j \cdot \sum_{j=1}^{n} w_{ik} \cdot I\left(j = j_1\right) = \lambda_4 \widetilde{M}_{j_1} \sum_{j=1}^{n} w_{ik}.$$

The first order partial derivatives are

$$\frac{\partial H}{\partial w_{i_1 k}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial w_{i_1 k}} + \lambda_3 M_{i_1} \sum_{j=1}^{n} \widetilde{w}_{jk}, \tag{6.3.1}$$
$$\frac{\partial H}{\partial b_{i_1}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial b_{i_1}} + \lambda_1 \sum_{i=1}^{n} b_i, \tag{6.3.2}$$
$$\frac{\partial H}{\partial u} = \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial u},$$
$$\frac{\partial H}{\partial \lambda_1} = \frac{1}{2} \left(\sum_{i=1}^{n} b_i \right)^2, \quad \frac{\partial H}{\partial \lambda_3} = \frac{1}{2} \sum_{i=1}^{n} M_i^2.$$

We estimate $\hat{\lambda_1}$ and $\hat{\lambda_3}$ as in previous sections 6.1 and 6.2 by setting equations (6.3.1) and (6.3.2) to zero respectively for all rows. Then we obtain

$$\hat{\lambda}_1 = - \left(n \sum_{i=1}^{n} b_i \right)^{-1} \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial b_i},$$
$$\hat{\lambda}_3 = - \left(\sum_{i=1}^{n} M_i \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j^\top \right)^{-1} \sum_{j=1}^{n} \widetilde{\boldsymbol{w}}_j^\top \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \ell_{ij}}{\partial \mathbf{w}_i}.$$

To estimate other unknown quantities with alternating scheme, consider the second order

partial derivatives

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial w_{i_2r}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial w_{i_2r}} + \lambda_3 \sum_{j=1}^{n}\widetilde{w}_{jr}\sum_{j=1}^{n}\widetilde{w}_{jk}\cdot I(i_1 = i_2),$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial w_{i_2r}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial w_{i_2r}}\right] - \lambda_3 \sum_{j=1}^{n}\widetilde{w}_{jr}\sum_{j=1}^{n}\widetilde{w}_{jk}\cdot I(i_1 = i_2).$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial b_{i_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial b_{i_2}}, \qquad \frac{\partial^2 H}{\partial \lambda_1^2} = \frac{\partial^2 \widetilde{H}}{\partial \lambda_2^2} = \frac{\partial^2 H}{\partial \lambda_3^2} = \frac{\partial^2 \widetilde{H}}{\partial \lambda_4^2} = 0,$$

$$\frac{\partial^2 H}{\partial b_{i_1}\partial b_{i_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial b_{i_2}} + \lambda_1, \quad \Longrightarrow \quad -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial b_{i_2}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial b_{i_2}}\right] - \lambda_1.$$

$$\frac{\partial^2 H}{\partial u^2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}, \qquad \Longrightarrow \qquad -E\left[\frac{\partial^2 H}{\partial u^2}\right] = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\mu_{ij}^{2-p_{ij}}}{\phi_{ij}}.$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}\widetilde{w}_{jk}\cdot I\left(i = i_1\right),$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial u}\right] = \sum_{j=1}^{n}\frac{\mu_{i_1j}^{2-p_{i_1j}}}{\phi_{i_1j}}\cdot\widetilde{w}_{jk}.$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_1} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_1} = 0, qquad\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_3} = 0,$$

$$\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_3} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial w_{i_1k}\partial \lambda_3} + M_{i_1}\sum_{j=1}^{n}\widetilde{w}_{jk} = M_{i_1}\sum_{j=1}^{n}\widetilde{w}_{jk},$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial w_{i_1k}\partial \lambda_3}\right] = -M_{i_1}\sum_{j=1}^{n}\widetilde{w}_{jk}.$$

$$\frac{\partial^2 H}{\partial b_{i_1}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial u} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial u^2}\cdot I\left(i = i_1\right),$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial u}\right] = \sum_{j=1}^{n}\frac{\mu_{i_1j}^{2-p_{i_1j}}}{\phi_{i_1j}}.$$

$$\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_1} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1}\partial \lambda_1} + \sum_{i=1}^{n} b_i = \sum_{i=1}^{n} b_i,$$

$$\Longrightarrow -E\left[\frac{\partial^2 H}{\partial b_{i_1}\partial \lambda_1}\right] = -\sum_{i=1}^{n} b_i.$$

Let

$$R = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad \boldsymbol{\beta}_i = \left(\mathbf{w}_i^\top, b_i\right)^\top, \quad \boldsymbol{\beta} = \left(\boldsymbol{\beta}_1^\top, \boldsymbol{\beta}_2^\top, \cdots, \boldsymbol{\beta}_n^\top\right)$$

where $A = \text{Diag}\left[A_i, i = 1, \ldots, n\right]$ is $n(d+1)$ dimensional square matrix with

$$A_i = -E\left\{\sum_{i_1=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 \ell_{i_1 j}}{\partial \boldsymbol{\beta}_i \partial \boldsymbol{\beta}_i^\top}\right\} + S, \tag{6.3.3}$$

and S has $(r,k)^{th}$ entry s_{rk} as

$$s_{rk} = \begin{cases} \lambda_3 \sum_{j=1}^{n} \widetilde{w}_{jr} \sum_{j=1}^{n} \widetilde{w}_{jk} & \text{for} \quad r = 1, \ldots, d; k = 1, \ldots, d \\ 0, & \text{if} \quad \max(r,k) = d+1, \min(r,k) < d+1 \cdot \\ \lambda_1, & \text{if} \quad r = k = d+1 \end{cases}$$

$A^{-1} = \text{Diag}\left[A_i^{-1}, i = 1, \ldots, n\right], \quad A \quad \text{is} \quad n(d+1) \quad \text{dimensional square matrix.}$

$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix}, \quad B_i \quad \text{is} \quad (d+1) \times 3 \quad \text{matrix. Then} \quad B \quad \text{is} \quad n(d+1) \times 3 \quad \text{matrix.}$

$$B_i = \begin{bmatrix} -E\left[\frac{\partial^2 H}{\partial \mathbf{w}_i \partial u}\right] & -E\left[\frac{\partial^2 H}{\partial \mathbf{w}_i \partial \lambda_1}\right] & -E\left[\frac{\partial^2 H}{\partial \mathbf{w}_i \partial \lambda_3}\right] \\ -E\left[\frac{\partial^2 H}{\partial b_i \partial u}\right] & -E\left[\frac{\partial^2 H}{\partial b_i \partial \lambda_1}\right] & -E\left[\frac{\partial^2 H}{\partial b_i \partial \lambda_3}\right] \end{bmatrix}.$$

$$C = B^\top, \qquad D = \text{Diag}\left\{-E\left[\frac{\partial^2 H}{\partial u^2}\right], 0, 0\right\}_{3\times 3}. \tag{6.3.4}$$

Let $K = D - CA^{-1}B$. If K is invertible, then from Lu and Shiou 2002

$$R^{-1} = \begin{bmatrix} A^{-1} + A^{-1}BK^{-1}CA^{-1} & -A^{-1}BK^{-1} \\ -K^{-1}CA^{-1} & K^{-1} \end{bmatrix}.$$

Let $U = \begin{bmatrix} U_{\boldsymbol{\beta}} \\ U_\Lambda \end{bmatrix}$ where $U_\Lambda = \begin{bmatrix} U_u & U_{\lambda_1} & U_{\lambda_3} \end{bmatrix}^\top$, $U_{\boldsymbol{\beta}}$ is $n(d+1) \times 1$ vector, U_Λ is 3×1 vector. Then our updating formula for parameter $\left(\boldsymbol{\beta}^\top, \Lambda^\top\right)^\top$ is

$$\begin{aligned}
\begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_{t+1} &= \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_t + \frac{lr}{t} \mathrm{R}^{-1} U \\
&= \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_t + \frac{lr}{t} \begin{bmatrix} A^{-1} + A^{-1}BK^{-1}CA^{-1} & -A^{-1}BK^{-1} \\ -K^{-1}CA^{-1} & K^{-1} \end{bmatrix} \begin{bmatrix} U_{\boldsymbol{\beta}} \\ U_\Lambda \end{bmatrix} \\
&= \begin{pmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\Lambda} \end{pmatrix}_t + \frac{lr}{t} \begin{bmatrix} A^{-1}U_{\boldsymbol{\beta}} + A^{-1}BK^{-1}CA^{-1}U_{\boldsymbol{\beta}} - A^{-1}BK^{-1}U_\Lambda \\ -K^{-1}CA^{-1}U_{\boldsymbol{\beta}} + K^{-1}U_\Lambda \end{bmatrix}.
\end{aligned}$$

Each term in the updating formula is exactly the same as their counterpart in section 6.1 except the components of the matrix A is augmented as in (6.3.3).

Working with the column data while treating $\mathbf{w}$ and $\mathbf{b}$ as fixed, the partial derivatives are

$$\begin{aligned}
\frac{\partial \widetilde{H}}{\partial \widetilde{w}_{j_1 k}} &= \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \widetilde{\ell}_{ij}}{\partial \widetilde{w}_{j_1 k}} + \lambda_4 \widetilde{M}_{j_1} \sum_{i=1}^{n} w_{ik}, \\
\frac{\partial \widetilde{H}}{\partial \widetilde{b}_{j_1}} &= \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial \widetilde{\ell}_{ij}}{\partial \widetilde{b}_{j_1}} + \lambda_2 \sum_{i=1}^{n} \widetilde{b}_j, \\
\frac{\partial \widetilde{H}}{\partial \lambda_2} &= \frac{1}{2} \left(\sum_{j=1}^{n} \widetilde{b}_j \right)^2, \quad \frac{\partial \widetilde{H}}{\partial \lambda_4} = \frac{1}{2} \sum_{j=1}^{n} \widetilde{M}_j^2.
\end{aligned}$$

Setting the first two equations to zero for all j_1 to solve for the estimate of λ_2 and λ_4 yields

$$\hat{\lambda}_2 = -\left(n\sum_{j=1}^{n}\widetilde{b}_j\right)^{-1}\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial\widetilde{\ell}_{ij}}{\partial\widetilde{b}_j},$$
$$\hat{\lambda}_4 = -\left(\sum_{j=1}^{n}\widetilde{M}_j\sum_{i=1}^{n}\mathbf{w}_i^\top\right)^{-1}\sum_{i=1}^{n}\mathbf{w}_i^\top\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial\widetilde{\ell}_{ij}}{\partial\widetilde{\boldsymbol{w}}_j}.$$

The following second order partial derivatives are needed to compute the updated parameter estimates:

$$\frac{\partial^2\widetilde{H}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{w}_{j_2r}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{w}_{j_2r}} + \lambda_4\sum_{i=1}^{n}w_{ir}\sum_{i=1}^{n}w_{ik}\cdot I(j_1=j_2),$$
$$\implies -E\left[\frac{\partial^2\widetilde{H}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{w}_{j_2r}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{w}_{j_2r}}\right] - \lambda_4\sum_{i=1}^{n}w_{ir}\sum_{i=1}^{n}w_{ik}\cdot I(j_1=j_2).$$
$$\frac{\partial^2\widetilde{H}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{b}_{j_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{w}_{j_1k}\partial\widetilde{b}_{j_2}},$$
$$\frac{\partial^2\widetilde{H}}{\partial\widetilde{b}_{j_1}\partial\widetilde{b}_{j_2}} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{b}_{j_1}\partial\widetilde{b}_{j_2}}!+\lambda_2, \quad \implies \quad -E\left[\frac{\partial^2\widetilde{H}}{\partial\widetilde{b}_{j_1}\partial\widetilde{b}_{j_2}}\right] = -E\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{b}_{j_1}\partial\widetilde{b}_{j_2}}\right] - \lambda_2.$$

$$\frac{\partial^2\widetilde{H}}{\partial\widetilde{w}_{j_1k}\partial\lambda_2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{w}_{j_1k}\partial\lambda_2} = 0, \qquad \frac{\partial^2\widetilde{H}}{\partial\widetilde{b}_{j_1}\partial\lambda_4} = 0.$$
$$\frac{\partial^2\widetilde{H}}{\partial\widetilde{w}_{j_1k}\partial\lambda_4} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{w}_{j_1k}\partial\lambda_4} + \widetilde{M}_{j_1}\sum_{i=1}^{n}w_{ik} = \widetilde{M}_{j_1}\sum_{j=1}^{n}w_{ik},$$
$$\implies -E\left[\frac{\partial^2\widetilde{H}}{\partial\widetilde{w}_{j_1k}\partial\lambda_4}\right] = -\widetilde{M}_{j_1}\sum_{j=1}^{n}w_{ik}.$$
$$\frac{\partial^2\widetilde{H}}{\partial\widetilde{b}_{j_1}\partial\lambda_2} = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2\widetilde{\ell}_{ij}}{\partial\widetilde{b}_{j_1}\partial\lambda_2} + \sum_{j=1}^{n}\widetilde{b}_j = \sum_{j=1}^{n}\widetilde{b}_j, \quad \implies \quad -E\left[\frac{\partial^2\widetilde{H}}{\partial\widetilde{b}_{j_1}\partial\lambda_2}\right] = -\sum_{j=1}^{n}\widetilde{b}_j.$$

Let $\widetilde{U} = \begin{bmatrix} \widetilde{U}_{\widetilde{\boldsymbol{\beta}}} \\ \widetilde{U}_{\widetilde{\Lambda}} \end{bmatrix}$ where $\widetilde{U}_{\Lambda} = \begin{bmatrix} U_u & \widetilde{U}_{\lambda_2} & \widetilde{U}_{\lambda_4} \end{bmatrix}^{\top}$. The updating equation is

$$\begin{aligned}
\begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_{t+1} &= \begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_{t} + \frac{lr}{t} \mathrm{R}^{-1} \widetilde{U} \\
&= \begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_{t} + \frac{lr}{t} \begin{bmatrix} \widetilde{A}^{-1} + \widetilde{A}^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{C}\widetilde{A}^{-1} & -\widetilde{A}^{-1}\widetilde{B}\widetilde{K}^{-1} \\ -\widetilde{K}^{-1}\widetilde{C}\widetilde{A}^{-1} & \widetilde{K}^{-1} \end{bmatrix} \begin{bmatrix} \widetilde{U}_{\widetilde{\boldsymbol{\beta}}} \\ \widetilde{U}_{\widetilde{\Lambda}} \end{bmatrix} \\
&= \begin{pmatrix} \widehat{\widetilde{\boldsymbol{\beta}}} \\ \widehat{\widetilde{\Lambda}} \end{pmatrix}_{t} + \frac{lr}{t} \begin{bmatrix} \widetilde{A}^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} + \widetilde{A}^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{C}\widetilde{A}^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} - \widetilde{A}^{-1}\widetilde{B}\widetilde{K}^{-1}\widetilde{U}_{\widetilde{\Lambda}} \\ -\widetilde{K}^{-1}\widetilde{C}\widetilde{A}^{-1}\widetilde{U}_{\widetilde{\boldsymbol{\beta}}} + \widetilde{K}^{-1}\widetilde{U}_{\widetilde{\Lambda}} \end{bmatrix},
\end{aligned}$$

where $\widetilde{A} = \mathrm{Diag}\left[\widetilde{A}_i, i = 1, \ldots, n\right]$ is $n(d+1)$

$$\widetilde{A}_i = -E\left\{\sum_{i_1=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 \ell_{i_1 j}}{\partial \widetilde{\boldsymbol{\beta}}_{i_1} \partial \widetilde{\boldsymbol{\beta}}_{i_1}^{\top}}\right\} + \widetilde{S}, \tag{6.3.5}$$

and $\widetilde{S}$ has $(r,k)^{th}$ entry $\widetilde{s}_{rk}$ as

$$\widetilde{s}_{rk} = \begin{cases} \lambda_4 \sum_{i=1}^{n} w_{ir} \sum_{i=1}^{n} w_{ik} & \text{for} \quad r = 1, \ldots, d; k = 1, \ldots, d, \\ 0, & \text{if} \quad \max(r,k) = d+1, \min(r,k) < d+1, \cdot \\ \lambda_2, & \text{if} \quad r = k = d+1 \end{cases}$$

$$\begin{aligned}
&\widetilde{B} = \left[\widetilde{B}_1^{\top}, \quad \widetilde{B}_2^{\top}, \quad \cdots \quad , \widetilde{B}_n^{\top}\right]^{\top} \quad \text{is an } n(d+1) \times 3 \text{ matrix with} \\
&\widetilde{B}_i = \left[-E\left[\frac{\partial^2 H}{\partial \widetilde{\boldsymbol{\beta}}_i \partial u}\right] \qquad - E\left[\frac{\partial^2 H}{\partial \widetilde{\boldsymbol{\beta}}_i \partial \lambda_2}\right] \qquad - E\left[\frac{\partial^2 H}{\partial \widetilde{\boldsymbol{\beta}}_i \partial \lambda_4}\right]\right],
\end{aligned}$$

and $\widetilde{C} = \widetilde{B}^{\top}$, $\widetilde{K} = D - \widetilde{C}\widetilde{A}^{-1}\widetilde{B}$, with D was defined in (6.3.4).

Compared to the version in section 6.2, the A^{-1} and $\widetilde{A}^{-1}$ here can be computed directly

because A and $\widetilde{A}$ are block-diagonal matrices.

Numerical performance of these different versions of algorithms will be discussed in the forthcoming chapter.

Chapter 7

Numerical evaluation of the proposed algorithms for word embedding and applications to NLP tasks

Previously we have planned to use the same vocabulary size of 400,000 as GloVe to learn word embeddings using a Wikipedia dump data. The January 2022 Wikipedia dump has around 4 billion tokens. We encountered more trouble than we expected during preprocessing of the compressed text and converting the raw text data into co-occurrence counts. We exploited the computational techniques described in Chapter 5. At the stage of combining the co-occurrence counts from 62 enormous sized databases into smaller number of databases, we had serious trouble because Beocat constantly preempted some of the running processes and restarted them as new processes. The trouble happens when each submitted job incrementally takes a large chunk of data from a database to summarize and write into another databases. Many processes ran parallelly together on one task. When preemption happened, the re-run processes were restarted to continue writing into the new databases. The content includes both the already completed part before the preemption and those that have not been summarized and written out previously. We tried many times to re-run the code but

never had any of the submission executed successfully to complete without preemption. This leads to repeated co-occurrence counts to part of the data. Each time of resubmitting the job, it takes weeks to run. In the end, we feel that we can not trust the final combined database. Instead, we decided to use smaller vocabulary size. Rather than using words to form the vocabulary, we chose to use WordPiece tokens, which has the potential to alleviate problems with rare or unseen words.

The WordPiece tokenizer of Wu et al. 2016 divides words into a limited set of common subword units referred as wordpieces. This tokenizer was used in BERT (Devlin et al. 2018) to tokenize words into subword tokens. We adopted the same tokenizer. The most commonly used BERT base uncased version has 30,522 WordPiece tokens. Removing some international characters leads to 26,531 tokens for us to use. With WordPiece tokenization, the word 'cucumber' is broken into the sequence of characters 'cu', '##cum', '##ber', where ## indicates that the character is a middle part of the word. We tokenized the entire training corpus with this tokenizer and obtained token-token co-occurrence counts (X_{ij}) based on our definition

$$X_{ij} = \begin{cases} \sum_{\text{s} \in \text{ all sentences}} 1/d_{ij}(s), & \text{if } d_{ij}(s) \leq \text{k} \\ 0, & otherwise \end{cases}$$

where $d_{ij}(s)$ = separation between token i and token j in sentence s, k is pre-determined window size 10. The sparse entries of the co-occurrence counts were stored in a sqlite database for model training.

With these 26,531 tokens in our vocabulary, we are able to carry out the model estimation for the SA-Tweedie models described in Chapters 4 and 6. The model in Chapter 4 are referred as version 19-3 in further discussions. Those in Chapter 6 are referred as versions 19-4, 19-5, and 19-6, respectively. The calculation details are similar to those described in Chapter 5 except that the units are WordPiece tokens instead of words.

7.1 Numerical performance on algorithm convergence behavior

In this subsection, we present the result of SA-Tweedie model fitting and estimation using the Wikipedia dump data. The dataset and preprocessing were described in Chapter 5.

The SA-Tweedie model was applied to the log(co-occurrence count+1) of token-token pairs in the entire Wikipedia dump data. The reason that we decide to use log count is because the skewness of the raw counts are too big to be modeled well. From the co-occurrence matrix, we explored the raw count data in each row to compute the sample skewness. The histogram of the sample skewnesses from different rows are given in the left panel in Figure 7.1. We can see that most of the skewness values are much larger than 40. It is very difficult to model the data if the skewness is over six. The log-counts seems to have sample skewness values mostly within 6. Even after log-transformation, the data in most of the rows are still very skewed (see Figure 7.2 for histogram of some randomly selected rows) and have plenty of zero observations. For this reason, we decided to model the log-counts using the alternating Tweedie regression model we proposed in Chapters 4 and 6.

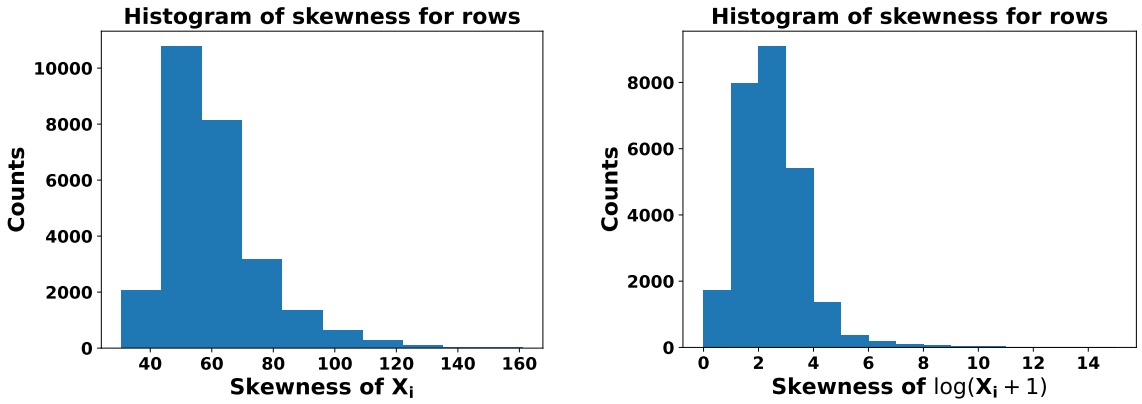

Figure 7.1: *Histogram of skewness for each row in raw co-occurrence count matrix (left panel) and the log co-occurrence count (right panel) constructed from Wikipedia dump.*

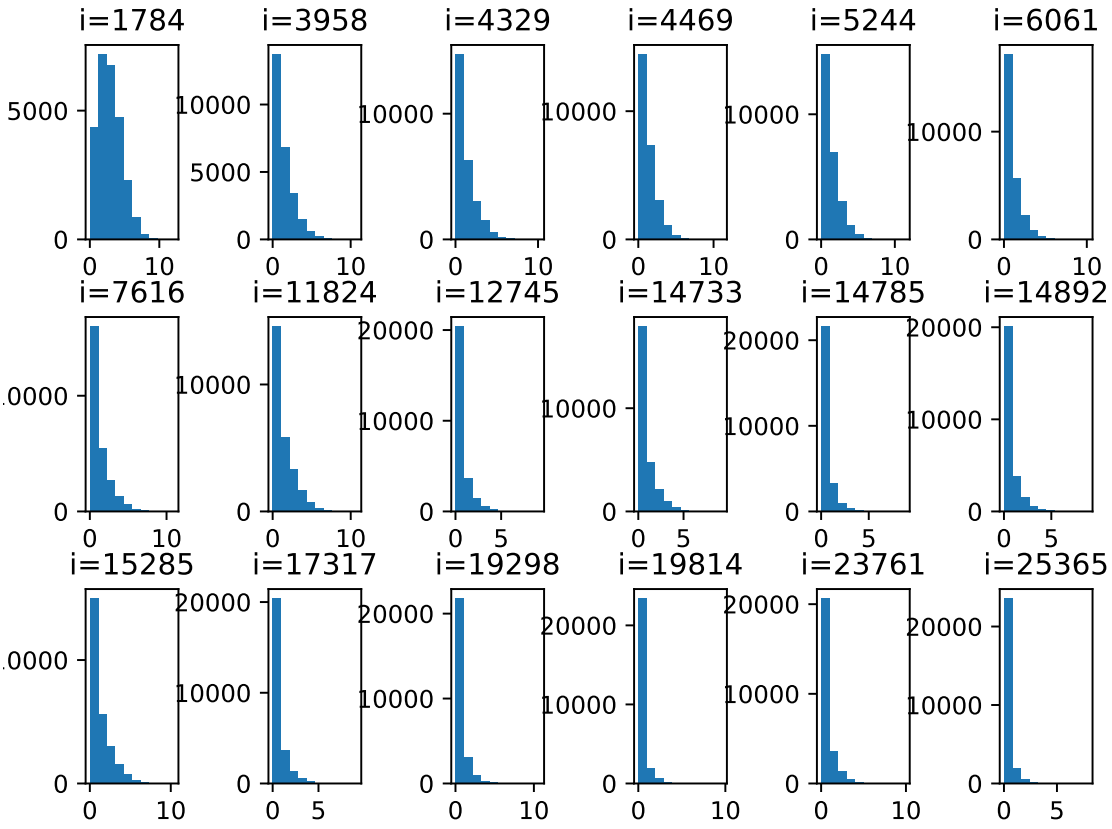

Figure 7.2: *Histogram of co-occurrence count after log transformation. The index i indicates row index in the co-occurrence matrix. 18 rows are randomly selected.*

Recall that in Chapter 4, we defined our objective loss function as

$$Loss(\boldsymbol{\beta}, \widetilde{\boldsymbol{\beta}}) = -\sum_{i=1}^{n}\sum_{j=1}^{n}(y_{ij}\theta_{ij} - \kappa(\theta_{ij})). \tag{7.1.1}$$

This loss is in fact the main part of the deviance loss function in Tweedie regression since it changes with the model parameter $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$ while the remaining term in the deviance is fixed for given p_{ij}'s and observed y_{ij}'s. Note that the deviance loss is given by

$$\text{Deviance loss} = 2\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{y_{ij}^{2-p_{ij}}}{(1-p_{ij})(2-p_{ij})} - y_{ij}\theta_{ij} + \kappa(\theta_{ij})\right).$$

The value of the constant term $\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{y_{ij}^{2-p_{ij}}}{(1-p_{ij})(2-p_{ij})} = 8.9043 \times 10^8$ stays fixed for fixed

p_{ij}'s and observed y_{ij}'s. This value represents the major part in the loglikelihood function under the saturated model that each observation has its own mean. Its value gives us an idea of the magnitude of our loss function defined in equation (7.1.1).

We use relative convergence criteria on the loss function by checking if the magnitude of change in the loss function compared to the absolute value of current loss function is less than a predefined threshold. In computation, we set the convergence threshold as $\epsilon = 10^{-4}$ and the maximum number of iterations ($maxit$) was set to be 100. That is, the model is said to be converged if the relative change in Loss below is less than 10^{-4}.

$$\text{Relative change in Loss } = \frac{|Loss(\boldsymbol{\beta}^{(t+1)}, \widetilde{\boldsymbol{\beta}}^{(t+1)}) - Loss(\boldsymbol{\beta}^{(t)}, \widetilde{\boldsymbol{\beta}}^{(t)})|}{|Loss(\boldsymbol{\beta}^{(t+1)}, \widetilde{\boldsymbol{\beta}}^{(t+1)})| + 0.1}. \tag{7.1.2}$$

For the models with constraints given in Chapter 6, we add the parameter constraints in the loss function in (7.1.1). Then the relative convergence criteria is used to assess whether the model has converged.

Figures 7.3, 7.4, and 7.5 present the loss curves from training the 19-3, 19-4, and 19-6 versions of the algorithms. The loss function of the 19-3 version behave nicely in that it drastically goes down in very few iterations. The loss functions maintained a monotone decreasing trend for all embedding dimensions. The algorithm for $d = 100$ and $d = 300$ converged based on the relative convergence criterion. Upon convergence, the values of the loss function are -8.2186×10^8 and -8.3284×10^8 for $d = 100$ and $d = 300$, respectively. Their relative loss change values are 9.7895×10^{-5} and 9.4375×10^{-5}, respectively. The $d = 768$ case has not converged yet during the writing. The current loss value is -8.3812×10^8, which gives the relative loss change value 0.0011.

The version 19-4 of the algorithm in Figure 7.4 oscillates back and forth like the trajectory of a pendulum. The magnitude of the loss decreases as the iteration increases slowly. This happens because the linear constraints in the objective function does not have a control of the direction. Consequently, the algorithm repeatedly corrects the change in the previous

iteration. Currently the initial learning rate was set to be 0.5, which is the same as all the other versions of the algorithms. With this learning rate, this version 19-4 algorithm might converge over hundreds or thousands of iterations. For this algorithm, reducing the learning rate may help the algorithm to converge faster.

The 19-5 version's quadratic constraints were carried into both parts of the alternating regression process. Even though it was a good intention but the greedy nature exhausted the available memory allocated to the process. Consequently, there was no iteration ever finished. The machine was busy swapping data between RAM and hard drive all the time. Therefore, we were not able to proceed the training process of version 19-5.

The 19-6 version shown in Figure 7.5 shows drastic oscillation in the beginning of the training process for both embedding dimensions 50 and 100. The embedding dimension 100 case quickly finds the right direction and starts to optimize the objective function monotonically. For embedding dimension 50, however, the curve kept oscillates as the training process moves on. But the magnitude of the oscillation reduces. This might be again due to initial learning rate 0.5 being too large for the 50-dimensional case. One interesting point is that we expected the higher dimensional vector to give lower loss. In this scenario, however, the $d = 50$ case reached lower loss value than the $d = 100$ case after the 20th iteration even though its loss curve is not monotone.

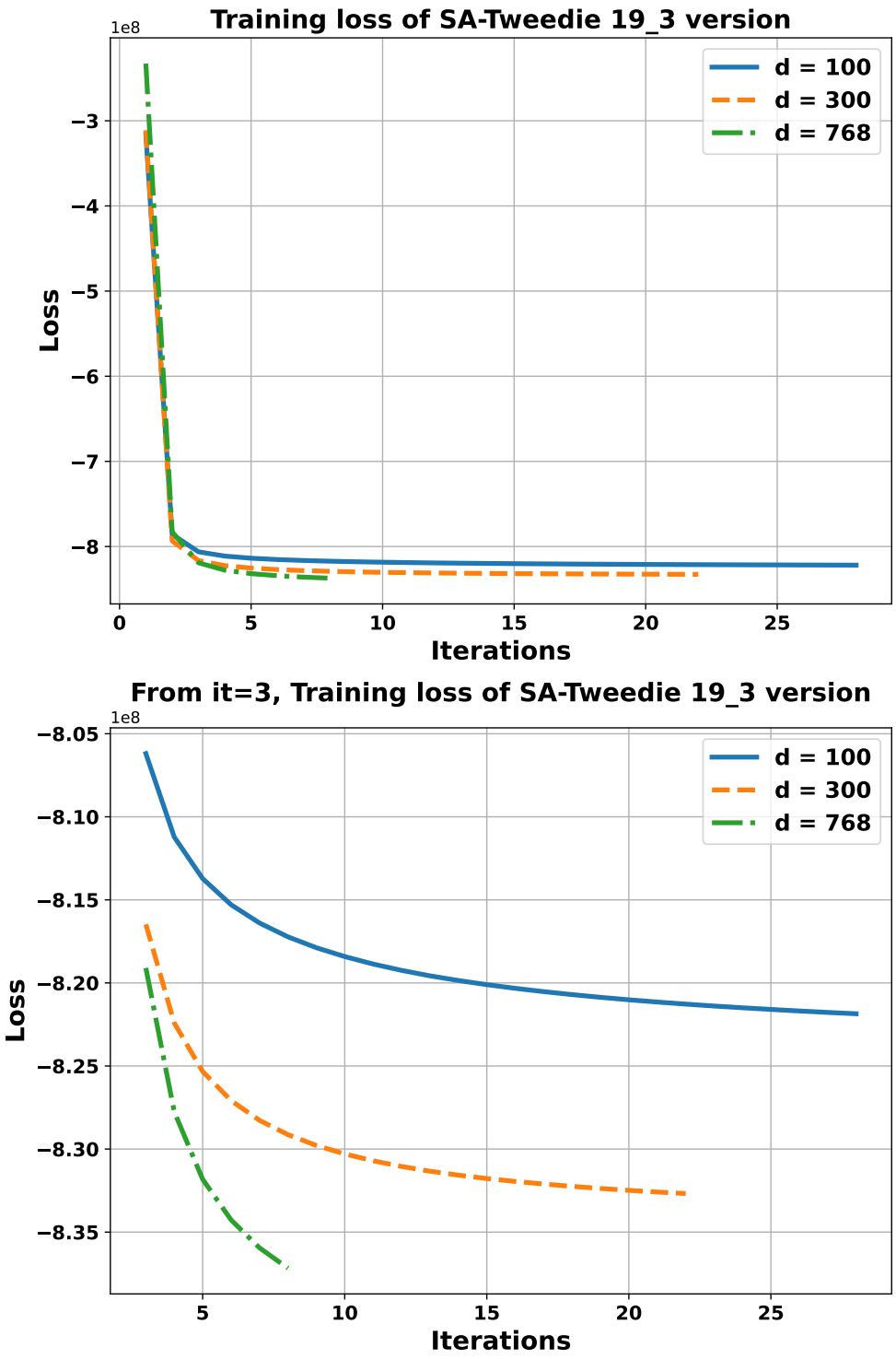

Figure 7.3: *Trajectory of the training process of version 19-3 of SA-Tweedie. Three different embedding dimensions, 100, 300, and 768, are considered. The left panel shows the entire training process. The right panel shows the details after the third iteration. The model achieved much lower loss with higher embedding dimensions.*

Figure 7.4: *Trajectory of the training process of the version 19-4 of the SA-Tweedie algorithm. Two different embedding dimensions 50 and 100 were considered.*

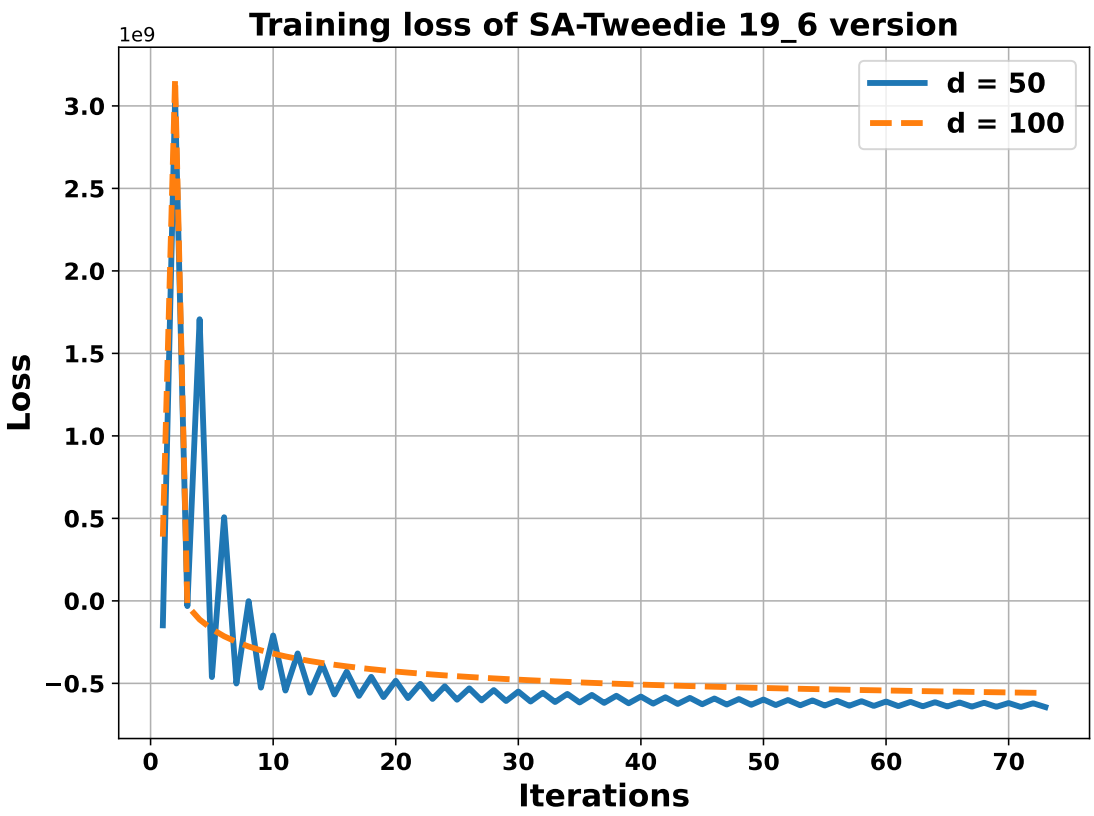

Figure 7.5: *Trajectory of the training process of version 19-6 of the SA-Tweedie algorithm. Two embedding dimensions, 50 and 100, were considered.*

7.2 Real data analysis using SA-Tweedie

7.2.1 Word similarity and word analogy

In this subsection, we illustrate applications of our pretrained word embeddings from Wikipedia dump in some NLP tasks. We would like to use the cosine similarity between word embedding vectors to find most similar words for a given word. For example, given a query word 'billion', we would like to return the top 10 words that have the highest cosine similarity. We will compare the words returned by using SA-Tweedie and GloVe embeddings. The 10 words were chosen by the algorithm from the vocabulary that each algorithm was trained on. The SA-Tweedie version 19-3 only has 26531 tokens. Even though it has the potential

ability to give vector representation for unseen words by combining those from sub-word tokens, we have not figured out an effective way to combine them yet. Some of the tokens of SA-Tweedie's vocabulary are sub-word tokens added with ##. For example, the word 'cucumber' does not exist in SA-Tweedie's vocabulary. Instead, 'cu', '##cum', '##ber' are in its vocabulary. Unfortunately, none of these tokens carry the meaning of cucumber. These tokens won't be helpful in word similarity or word analogy task since they are not full words. Hence searching within the vocabulary would lead to no useful information. In this regard, the small vocabulary size put SA-Tweedie in disadvantage for word analogy tasks compared to those that has huge vocabulary sizes such as GloVe that has 400,000 tokens or Word2Vec that has one million tokens. For this reason, we do not pursue large scale word analogy analysis here. What we find in our experiment is that SA-Tweedie's vector representations could accurately find semantically similar words if they are in SA-Tweedie's vocabulary. For syntactic tasks such as "*like* is to *unlike* as *certain* is to ___?", SA-Tweedie may not do as well because SA-Tweedie was trained with WordPiece tokens instead of extended collection of words. For this particular example, SA-Tweedie returned 'whereas' and GloVe returned 'furthermore' as the closest answer. Neither of them is close to the correct answer 'uncertain'. More examples like these are listed in Table 7.1. Both GloVe and SA-Tweedie answered correctly for questions regarding city-nation, family, comparative, and past tense sections with high accuracy. However, for opposite, plural, and superlative sections, both embeddings answer incorrectly with relatively low accuracy. In general, both embeddings show good performance at semantic related questions than those from syntactic in word analogy task.

Next we will illustrate the delicate difference between GloVe and SA-Tweedie with a few more examples. Tables 7.2 and 7.3 list the top ten most similar words to 'billion', 'kansas', 'south', and 'china' returned from using GloVe and SA-Tweedie embeddings.

- For 'billion', most of the top words returned from both embeddings align with common sense. SA-Tweedie embedding captured more words in financial sectors such as 'estimated', 'investment', and 'assets' which frequently appear with the word 'billion'.

	Given triplets			Answers GloVe	Answers SA-Tweedie
Correct	athens	greece	berlin	germany	germany
	man	king	woman	queen	queen
	strong	stronger	soft	softer	softer
	jumping	jumped	running	ran	ran
Incorrect	like	unlike	certain	furthermore	whereas
	banana	bananas	lion	lambs	beasts
	big	biggest	tall	tallest	largest
	big	biggest	tallest	second-largest	skyscraper

Table 7.1: *Word analogy triplets (in first three columns) and the top answer return from GloVe and SA-Tweedie. The first four examples received correct answers from both but the last four examples received incorrect answers from both embedding methods except for 'tallest' from GloVe.*

The two numbers 1.2 and 1.3 returned by GloVe are weird.

- For 'kansas', the words returned from both embeddings are mostly neighbor states in the midwest. In GloVe's result, four out of 10 are not states in the midwest (texas, arizona, kan., colorado). Only one of those from SA-Tweedie is not a state in the midwest. The similarity scores are very different. SA-Tweedie's similarities are all above 0.9, but GloVe's are at most 0.7.

- For 'south', the top four returned words from SA-Tweedie are 'south', 'north', 'west', and 'east', which are all directions. On the other hand, the top four returned words from GloVe are 'south', 'north', 'africa', 'korea'. Even though South Africa and South Korea may appear together frequently, 'africa' and 'korea' do not represent the semantic category of the word 'south'.

- For 'china', the top two words returned from both embeddings are identical. The next few top words from GloVe are a mix of cities and provinces. SA-Tweedie's returned top words are mostly at the same level, i.e. countries. Additionally, the similarities for SA-Tweedie's returned words are at least 0.8 while those from GloVe are all less than 0.8 except for the word china itself.

Rank	Top 10 most similar words to 'billion'				Top 10 most similar words to 'kansas'			
	GloVe		SA-Tweedie		GloVe		SA-Tweedie	
	Words	Similarity	Words	Similarity	Words	Similarity	Words	Similarity
1	billion	1.00	billion	1.00	kansas	1.00	kansas	1.00
2	million	0.80	million	0.84	missouri	0.70	oklahoma	0.94
3	trillion	0.77	dollars	0.80	oklahoma	0.68	missouri	0.93
4	dollars	0.76	estimated	0.77	texas	0.64	iowa	0.92
5	$	0.70	percent	0.76	nebraska	0.63	minnesota	0.92
6	dlrs	0.68	investment	0.76	minnesota	0.61	illinois	0.92
7	euros	0.68	millions	0.75	arizona	0.58	nebraska	0.92
8	worth	0.66	$	0.74	iowa	0.57	indiana	0.91
9	1.2	0.66	assets	0.72	kan.	0.56	arizona	0.90
10	1.3	0.66	cost	0.72	colorado	0.56	ohio	0.90

Table 7.2: *Top 10 most similar words to 'billion' and 'kansas' based on cosine similarity of word vectors from GloVe and SA-Tweedie.*

Rank	Top 10 most similar words to 'south'				Top 10 most similar words to 'china'			
	GloVe		SA-Tweedie		GloVe		SA-Tweedie	
	Words	Similarity	Words	Similarity	Words	Similarity	Words	Similarity
1	south	1.00	south	1.00	china	1.00	china	1.00
2	north	0.81	north	0.97	chinese	0.79	chinese	0.89
3	africa	0.69	west	0.96	beijing	0.77	japan	0.86
4	korea	0.66	east	0.94	taiwan	0.68	korea	0.86
5	southern	0.63	western	0.88	shanghai	0.62	india	0.85
6	east	0.62	united	0.88	mainland	0.62	hong	0.84
7	west	0.62	central	0.87	guangdong	0.61	taiwan	0.83
8	korean	0.61	northern	0.87	tibet	0.58	asia	0.83
9	african	0.60	southern	0.87	hong	0.58	europe	0.83
10	southeast	0.58	city	0.86	kong	0.57	america	0.81

Table 7.3: *Top 10 most similar words to 'south' and 'china' based on cosine similarity of word vectors from GloVe and SA-Tweedie.*

For these returned words, we projected the word vectors onto the first two principal components. They are shown in Figure 7.6 for GloVe and Figure 7.7 for SA-Tweedie. The cosine similarity between words is reflected by the angle between the two vectors formed with the origin, which is shown as the intersection point of the two blue lines. Those returned by SA-Tweedie are aligned tightly on similar vector directions. On the other hand, those from GloVe have bigger angles for its returned words similar to 'china' and 'south'. This shows that GloVe has trouble to capture the relationship among these words.

7.2.2 IMDB sentiment analysis

In this subsection, we present an analysis of binary text classification task on famous benchmark movie review data (IMDB data) from Stanford. They constructed a collection of 50,000

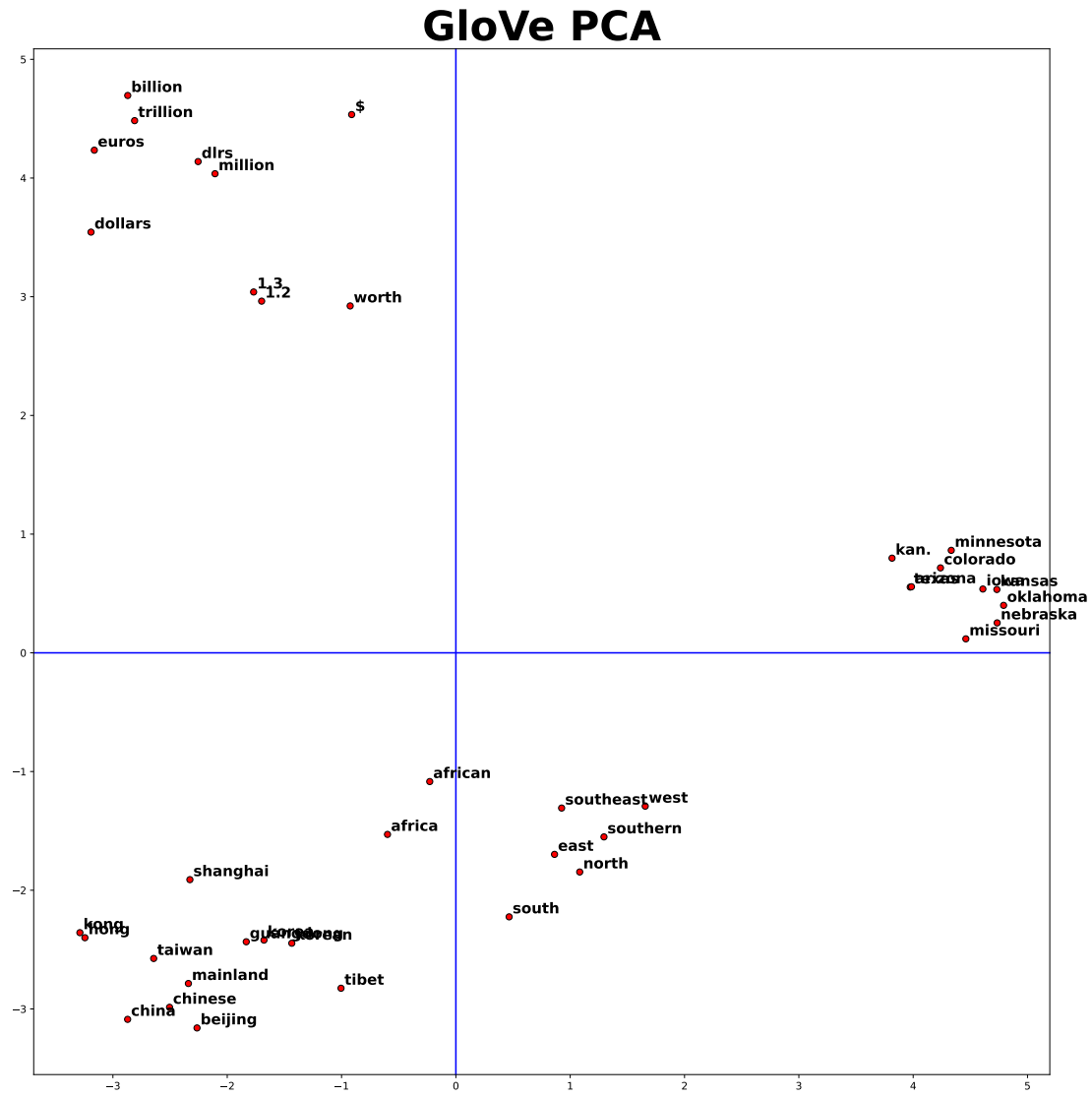

Figure 7.6: *The top 10 most similar words to 'billion', 'kansas', 'south', and 'china' returned by GloVe are projected on to the first two principal components.*

reviews from IMDB. Each movie review has no more than 30 reviews. The dataset consist of even number of positive and negative reviews for both training and test set. The training and test set both have 25,000 reviews respectively. They considered only highly polarized reviews: a negative review has a score below 4 out of 10, and a positive review has a score above 7 out of 10. The dataset does not contain neutral reviews.

To analyze the data, we first conducted data cleaning. Common abbreviations were replaced by plain words such as 'omg' to 'oh my god', 'lol' to 'laugh out loud'. HTML tags,

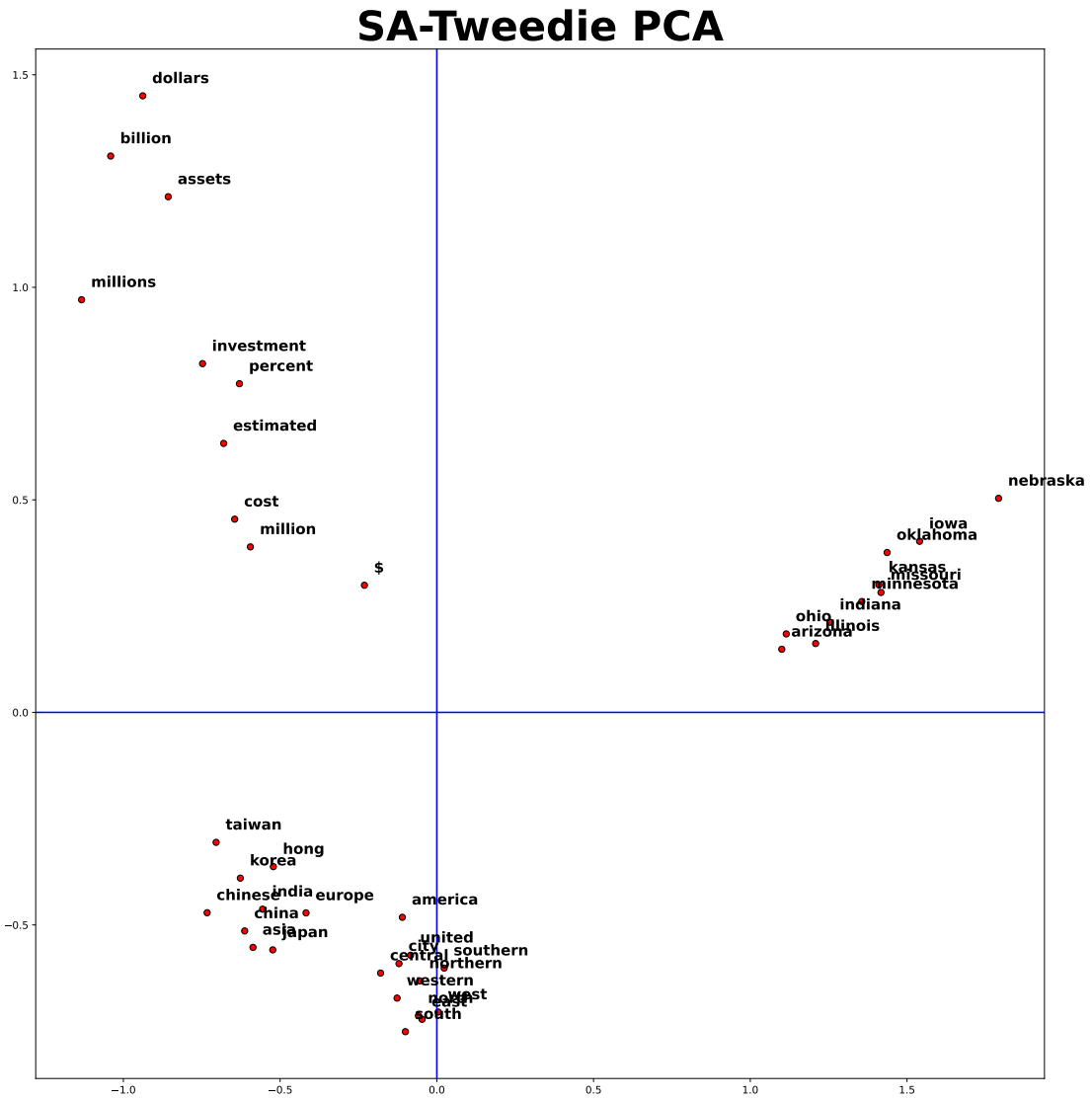

Figure 7.7: *The top 10 most similar words to 'billion', 'kansas', 'south', and 'china' returned by SA-Tweedie are projected on to the first two principal components.*

emojis, and punctuations are removed from raw reviews to get better quality of sentiment from text reviews. After cleaning the reviews, we split the given training set to training and validation set with the ratio of 7 to 3, which are for fitting and optimizing a model via parameter tuning, respectively. While splitting, we tried to preserve equal number of positive and negative reviews. Consequently, we have a training set consists of total 17,500 reviews with 8,748 negative and 8,752 positive reviews. The validation set has total 7,500 reviews with 3,752 negative and 3,748 positive reviews.

We further preprocessed the texts. The *keras.preprocessing.text.Tokenizer* is used to tokenize the reviews. Afterward, the distribution of the number of words in every review was obtained to determine how many tokens will be fed into a model during the training process. The number of word counts has a highly skewed distribution (see figure 7.8). We adopted 80^{th} percentile 318 as maximum input length in a model. Majority of reviews have less than 318 word counts. Such reviews were padded with zeros to reach the common length. This is because each review will be represented by a matrix of dimension 318 by 300. Carrying such big matrices around is memory inefficient. Instead, a vocabulary is created and each movie review text is converted to a sequence of indices that point to the locations of the tokens in the vocabulary. The vocabulary for this specific task was defined as the most frequent 10,000 words from the reviews in the training dataset.

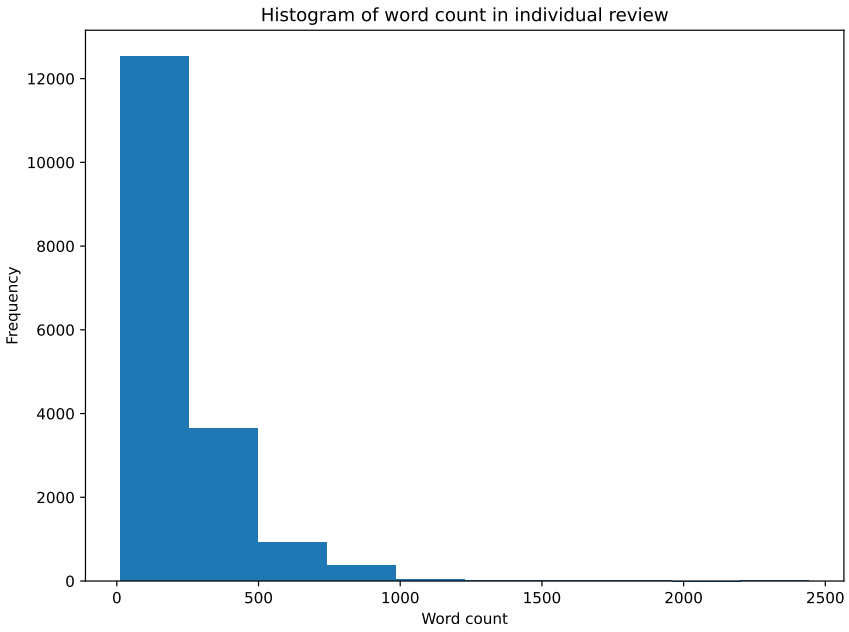

Figure 7.8: *The histogram of word count in individual review.*

We adopted the Bidirectional Long Short-Term Memory (BiLSTM) model as a classifier. BiLSTM model was popularly used in NLP field before Transformer-based models came out since it could capture the context of the given text sequence by utilizing long and short term

memory in a bidirectional way. Unlike its predecessor RNN model which uses only short term memory, BiLSTM could more efficiently capture the context of a long sequence. BiLSTM is a relatively compact model compare to Transformer based models, such as BERT, which have a lot of layers in them. Hence, BiLSTM is frequently used in token based language models. The architecture of the model contains the following components sequentially:

- embedding layer (10,000 by d): d-dimensional random embedding with data from uniform (-0.05, 0.05), or pretrained SA-Tweedie or GloVe embedding, where d=100, 300, or 768 (if available);
- one layer of bidirectional LSTM (hidden dimension 512);
- GlobalMaxPool1D layer for global max pooling operation to downsample by taking the maximum value within each row;
- Densely-connected linear layer followed by a rectified linear unit (ReLU) activation (output dimension=256);
- Densely-connected linear layer with softmax activation (output dimension=2).

The model parameters were updated using the Adam optimizer with default learning rate 0.001. The learning rate was adjusted by a factor of 0.2 using *ReduceLROnPlateau* when there is no improvement in validation loss for consecutive two epochs. The training loop will check at the end of every epoch whether the loss is no longer decreasing. When the number of epochs with no improvement reaches two, the training process stops. The best estimates of model parameters are taken to be those that give the minimum validation loss. Then the resulting model is applied to the test data to predict review sentiment.

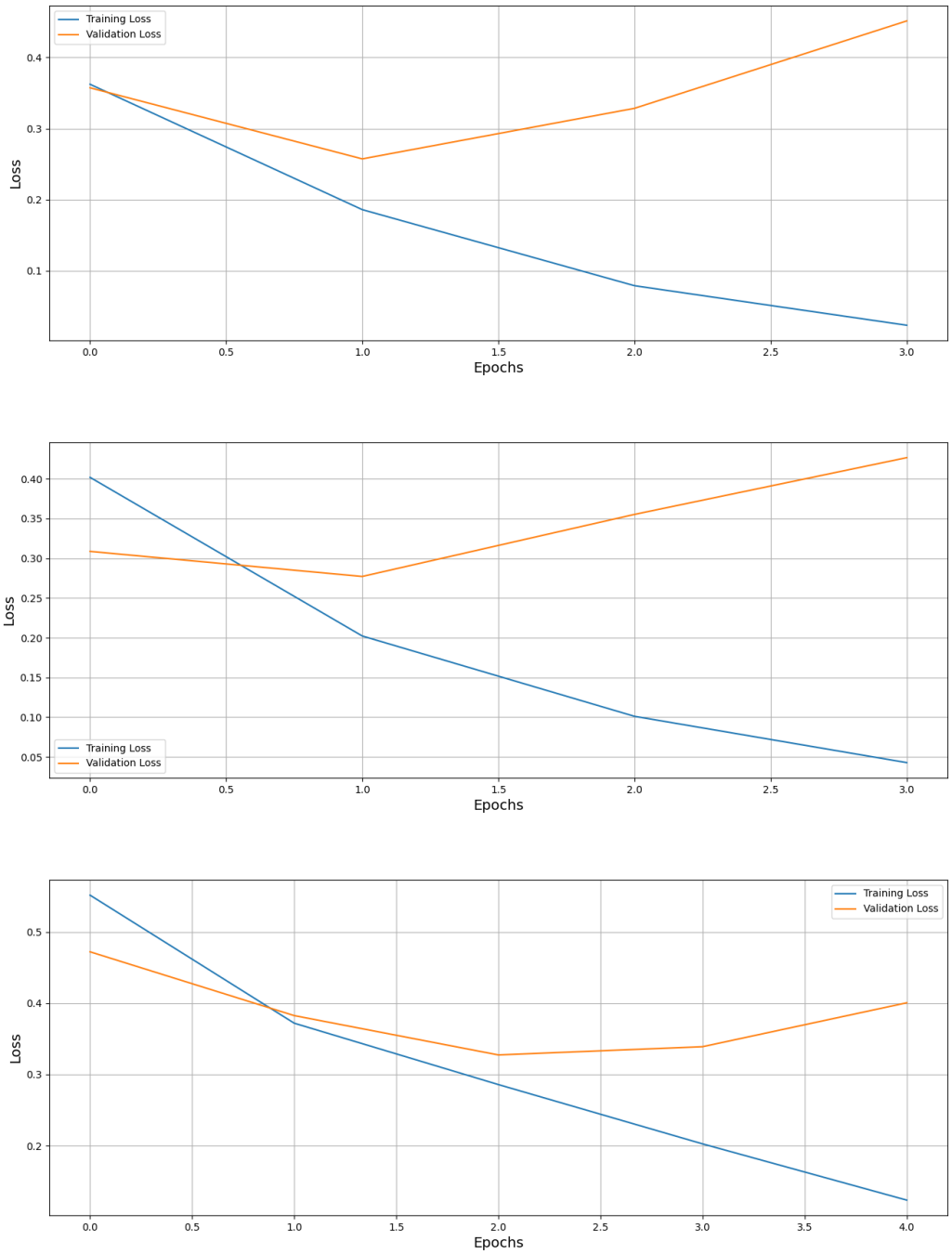

Figure 7.9: *The typical trajectory of training and validation loss from text classification task on IMDB dataset. Top panel: GloVe embedding with fine-tuning; middle panel: SA-Tweedie embedding with fine-tuning; bottom panel: random embedding with fine-tuning.*

Figure 7.9 shows the typical trajectory of the training process. The training loss reduces

until the model stops training according to the stopping criterion for all three cases. The BiLSTM model captures a lot of information about the data and takes very few epochs to show overfitting if the training process does not stop. In Figure 7.9, the minimum of validation loss is achieved at epoch one for GloVe and SA-Tweedie embeddings, but the random embedding achieved the minimum at epoch 2 with larger loss value.

There are negative and positive reviews. For each category of the reviews, positive (pos) or negative (neg), we compute precision and recall rate for each algorithm applied to the test data. The precision and recall are defined as follows. Consider any category of interest, for example, the positive review category, the precision and recall for this category are computed as

$$\text{precision} = \frac{TP}{TP + FP}, \quad \text{recall} = \frac{TP}{TP + FN},$$

where TP refers to the number of true positive reviews, FP refers to the number of negative reviews predicted to be positive, and FN refers to the number of positive reviews predicted to be negative. The F1 score is the harmonic mean of the precision and recall:

$$\text{F1 score} = \left(\frac{precision^{-1} + recall^{-1}}{2} \right)^{-1}.$$

Table 7.4 presents the prediction results for the test data after each model has been trained with the training dataset. It can be seen that the random embedding has lower prediction performance in precision, recall and F1-scores. This data set is a well balanced binary classification data. Therefore, the F1 score is equal to the classification accuracy. The GloVe and SA-Tweedie have comparable performance, with GloVe having slightly higher accuracy. Figure 7.10 shows the loss and F1 scores for the three top performing settings under each embedding. The training loss and training F-1 score curves exhibit similar monotone pattern in all three panels as epoch increases. The validation loss curves have simple pattern for the random and SA-Tweedie embedding but changed direction for the model with GloVe

	Category	Precision	Recall	F1 score	Support
Random	neg	87.03	88.11	87.57	12500
	pos	87.96	86.86	87.41	12500
	avg	87.49	87.49	87.49	25000
SA-Tweedie	neg	89.81	88.17	88.98	12500
	pos	88.38	89.99	89.18	12500
	avg	89.09	89.08	89.08	25000
GloVe	neg	89.43	89.44	89.44	12500
	pos	89.44	89.43	89.44	12500
	avg	89.44	89.44	89.44	25000

Table 7.4: *Text classification performance on the IMDB test dataset. The global seed was set to be 1111. These are the best performance for each embedding chosen from results with hidden dim = 256, 512, 768, and embedding dimension d=100, 300, 768 with the exception that GloVe does not have 768 dimensional embeddings available.*

embedding. The F1 scores for GloVe and SA-Tweedie here are slightly higher than the best results reported in the original paper (Maas et al. 2011) whose authors made the IMDB dataset available to the public. Modern transformers and many models with complicated architecture may have achieved even better results. But we refrain from resorting to those complicated models because our purpose is to see how our proposed embedding can be utilized in practice.

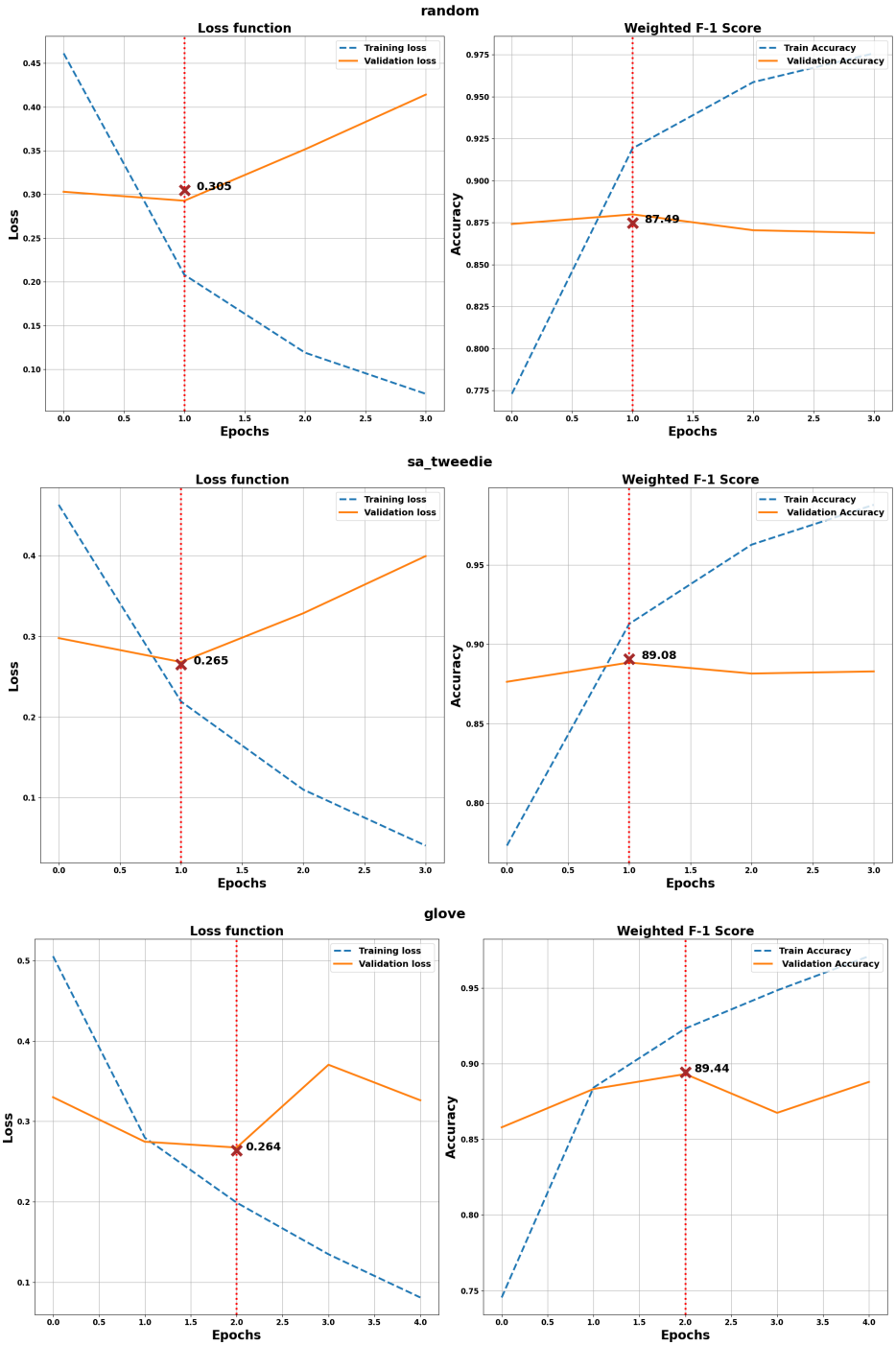

Figure 7.10: *Training trajectory of the best performed models in IMDB text classification task. Top panel: random embedding achieved with embedding dimension 300 and hidden dimension 256; middle panel: SA-Tweedie embedding achieved with embedding dimension 768 concatenation and hidden dimension 512 ; bottom panel: GloVe embedding achieved with embedding dimension 100 and hidden dimension 768. Test loss and F1 score are marked along with their value.*

7.2.3 Application to NER task on CoNLL-2003 data

In this section, we consider Named Entity Recognition (NER) task using the CoNLL-2003 English benchmark dataset. CoNLL-2003 is a collection of documents from Reuters newswire articles. Each sentence in the dataset is annotated with nine categories representing the beginning or inside of a named entity from four entity types: person, location, organization, and miscellaneous. The nine categories are B-LOC, B-MISC, B-ORG, B-PER, I-LOC, I-MISC, I-ORG, I-PER, O. The 'O' category is used for cases that do not belong to any of the four entity types.

The authors of Pennington et al. 2014 presented their result of analysis for this dataset. They used a comprehensive set of 437,905 discrete features that comes with the standard distribution of the Stanford NER model (Finkel et al. 2005). In addition, they added 50-dimensional vectors for each word of a five-word context as continuous features. With these features as input, they trained a conditional random field (CRF) with exactly the same setup as the CRFjoin model of Wang and Manning 2013. They reported F1-weighted score 93.2% and 88.3% on the validation and test set performance. In this subsection, we present our result of using a simple BiLSTM model with GloVe, random embedding, or SA-Tweedie embedding with 100 or 300 dimensional word vector representations. The GloVe performance in this simple model (around 88% on test set) concurs with the numbers reported by Wang and Manning 2013 as described above. The simple model with SA-Tweedie embedding achieved over 91% in F1-weighted score for the test dataset.

For our analysis, the vocabulary was generated from the training data. It contains all tokens with at least one count. The training and test data were both converted to lower case. Note that lower or upper case letters do contribute to the NER task since many named entities are in upper case letters. By converting to lower case letters, we increased the difficulty level of the NER task.

We tried with a similar model architecture as used in the previous subsection (BiLSTM + GlobalMaxPool1D + Dense layer with ReLU + Dense layer with softmax activation).

Unfortunately, these two datasets have very different nature. The IMDB data has more tokens in each movie review and only two possible outcomes (pos or neg). This translated to more information in the input variable and relatively easier task for the output prediction since it is binary case. The NER task on CoNLL-2003 data, on the other hand, has generally very short input text compared to the IMDB data. The output variable has 9 categories which are highly unbalanced. The O class is the majority and many other classes have very small number of observations. Such unbalanced multi-class classification is much more challenging than the binary classification case.

For this task, we used a simpler version of bidirectional LSTM model using our token embedding to perform the classification. The model architecture starts with an embedding layer using the pre-trained token embedding from either SA-Tweedie or GloVe, followed by a bidirectional dynamic RNN layer with variable input size padded to common length and 512 hidden units, a max pooling layer with output dimension 512, a linear layer with output dimension 256 and relu activation, and then a linear activation layer at the end with output dimension 9. The loss function was taken to be the cross entropy.

The model training used dataloader with batchsize = 256 to load the data. The tokens and labels are padded to the same maximum length for all text data in the same batch. Adam with lr =0.001 was used to conduct parameter update. No weight decay was used (i.e. weight decay =0) and amsgrad = False. The maximum L_2 norm of gradients w.r.t all parameters were truncated (i.e. clipped) at 10% of the total norm.

15 epochs were trained to fine tune the embedding vectors and other model parameters. Within the 15 epochs, the best model parameter estimates were chosen to be those that leads to the smallest validation loss. GloVe often took 1 to 2 epochs to reach its best validation loss. Random embedding took 7 to 8 epochs to reach its lowest validation loss. SA-Tweedie took 4 to 5 epochs to reach its best validation loss. The model with the lowest validation loss was then selected and applied to the test data.

The results with max pooling dimension 512 are presented in Table 7.5. The random

Embedding	Random			GloVe			SA-Tweedie		
Seed	12	42	111	12	42	111	12	42	111
Epoch	5	8	7	2	2	2	5	5	4
Valid loss	0.231	0.234	0.214	0.932	0.910	0.870	0.194	0.183	0.189
Valid F1 B-LOC	32.51	33.41	33.08	33.01	33.09	32.14	33.69	33.40	33.01
Valid F1 B-MISC	15.64	16.15	15.75	15.10	14.54	14.28	16.20	15.71	16.05
Valid F1 B-ORG	20.37	20.16	21.19	19.62	17.55	19.57	19.47	21.2	21.33
Valid F1 B-PER	24.57	23.96	24.39	22.44	21.96	22.37	24.45	24.66	24.39
Valid F1 I-LOC	3.71	4.84	4.45	4.43	4.55	4.85	5.05	5.04	4.85
Valid F1 I-MISC	4.90	4.97	4.42	3.44	3.72	3.91	4.29	4.82	4.75
Valid F1 I-ORG	7.27	7.65	8.01	5.64	5.57	5.98	6.78	8.31	7.86
Valid F1 I-PER	17.18	18.69	18.76	11.73	11.88	11.99	17.92	19.00	17.81
Valid F1 O	93.36	93.27	93.76	95.47	95.40	95.50	96.38	96.23	96.41
Valid F1-weighted	93.22	93.61	94.09	93.13	92.87	93.13	**94.85**	**95.14**	**95.01**
Test loss	0.390	0.448	0.404	1.565	1.530	1.469	0.335	0.334	0.339
Test F1 B-LOC	28.07	27.75	27.92	27.92	28.15	27.49	27.46	27.16	26.91
Test F1 B-MISC	10.58	10.90	11.03	10.15	10.05	9.43	11.28	11.08	11.10
Test F1 B-ORG	22.73	23.06	24.11	20.68	18.21	20.26	22.64	24.48	24.89
Test F1 B-PER	17.73	17.28	17.48	13.81	13.64	13.90	17.00	17.30	16.40
Test F1 I-LOC	2.85	4.15	3.47	3.57	3.52	3.93	3.69	3.82	3.44
Test F1 I-MISC	2.53	2.57	2.43	2.04	2.20	2.21	2.46	2.65	2.75
Test F1 I-ORG	8.70	9.04	9.13	6.75	6.65	7.19	8.70	9.58	9.58
Test F1 I-PER	13.21	14.82	14.69	4.88	4.96	5.08	12.82	13.94	11.64
Test F1 O	89.00	88.28	89.18	93.72	93.75	93.67	95.15	94.33	95.33
Test F1-weighted	88.18	87.78	88.72	88.67	88.48	88.66	**91.00**	**91.06**	**91.05**

Table 7.5: *Validation and test metrics using random, GloVe, or SA-Tweedie embedding for NER task. Embedding dimension was 300 for all three methods, and used BiLSTM model with max pooling output dimension 512. Seed: global seed set for random number generation on both CPU and GPU. Epoch: the epoch number that validation loss reached the minimum. Random embedding were generated from the uniform (-0.05, 0.05) as is the default setting in Tensorflow Keras.*

embedding and GloVe both achieved less than 94% in weighted F1 score on the validation data and less than 89% in weighted F1 score on the test data. SA-Tweedie achieved around 95% on weighted F1 score on the validation data and 91% weighted F1 score on the test data.

To get a sense of how good SA-Tweedie performs, we can compare it with the result from BERT models. In the BERT models on the NER task, case-preserving WordPiece models along with the maximal document context provided by the data were used to learn

	Dev F1	Test F1
Fine-tuning approach		
BERT large	96.6	92.8
BERT base	96.4	92.4
Feature-based approach (BERT base)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7.6: *F1 score reported in Devlin et al. 2018 from various BERT model on NER task of CoNLL-2003 data (Table 7 of Devlin et al. 2018). The fine-tuning approach takes the representation of the first cased WordPiece sub-token of each word as the input and fine tune the pretrained BERT as a classification task. The feature-based approach froze the parameters from one or more layers of the pretrained BERT and used them as input to 768-dimensional BiLSTM for the tagging task.*

the context specific representation of words. The F1 score of various BERT models on the validation and test set is given in table (7.6). Fine-tuning approach for both BERT large and base model achieved F1 score on test set as 92.8 and 92.4 respectively. On the validation set, BERT large and base showed F1 score 96.6 and 96.4 respectively. Note that BERT large model has 24 hidden layers with 1024-dimensional embedding, and BERT base model has 12 hidden layers with 768-dimensional embedding. The total number of parameters is 345 million for BERT large model and 110 million for BERT base mode. In their feature-based approach, which estimate only BiLSTM model parameters while keeping BERT parameters fixed, the 768-dimensional BiLSTM used in BERT still has a lot more parameters than our 512-dimensional BiLSTM. Our simple one-layer BiLSTM model using SA-Tweedie embedding with a max pooling layer has much less number of parameters. Keeping these model complexity in mind, BiLSTM with pretrained SA-Tweedie embedding offers a shortcut toward better performance while using a simpler model. Given the small architecture of our model, both validation and test performance of SA-Tweedie are encouraging.

What we presented in this section are for one setting of the analysis. We also experimented on some other model settings. In particular, we considered varying the model

architecture a little bit by changing the max pooling output dimension. When max pooling is used, we considered max pooling output dimension 256 and 512. When no max pooling is used, we denote the max pooling dimension as 1024, which is equal to the output dimension from the previous RNN layer. Further settings include differentiating capital versus lower case letters by adding an additional dimension in the vector representation, for which the value was taken to be the indicator of whether capital letters are present in the word. The result varies for all embedding methods.

For SA-Tweedie, we also considered using the representation of only first piece of the WordPiece tokens from any word as an alternative. For some words, the WordPiece tokenizer splits one word into multiple pieces of tokens. The SA-Tweedie gives vector representation for all pieces separately. In the end, the representation of the word need to either combine the representation from different pieces or take just the representation of the first piece alone. The authors of Devlin et al. 2018 presented results of NER analysis with BERT using first piece token only. To combine the representation vectors from pieces to form a single representation of the word, many methods have been explored in the literature (cf. Wu et al. 2016 and the references therein). In general, some form of positional encoding is used to preserve the relative location of the token in the word. The positional encoding is then included along with the token representations throughout the model parameter estimation process. What form of positional encoding to use is also an active research area in the NLP community. See Dufter et al. 2021 for a survey of different positional encoding methods. We used the simple form of positional encoding provided by Vaswani et al. 2017. That is, the positional encoding is a matrix that has number of columns equal to the dimension of the word representation. The number of rows is equal to the number of tokens from the word. Suppose there are k WordPiece tokens and let pos enumerate the positions of the tokens in

the word. The positional encoding for the i^{th} column in the *pos* row is given by

$$\begin{aligned} \text{PE}_{(\text{pos},2i)} &= \sin\left(\frac{\text{pos}}{10000^{2i/d}}\right), \\ \text{PE}_{(\text{pos},2i+1)} &= \cos\left(\frac{\text{pos}}{10000^{2i/d}}\right), \end{aligned}$$

where $\text{pos} = 0, \ldots, k-1$, $i = 0, \ldots, d/2$. Generally, we see that the SA-Tweedie with only first piece token performs better than using combination of the token pieces.

Figure 7.11 shows the comparison of performance from all three embedding methods. For the NER task, the SA-Tweedie embedding gives the best results in that the overall F1-weighted scores are consistently higher than those from GloVe and random embedding. This pattern can be seen by directly comparing performance under the same seed and the same lower case condition shown in Figure 7.11.

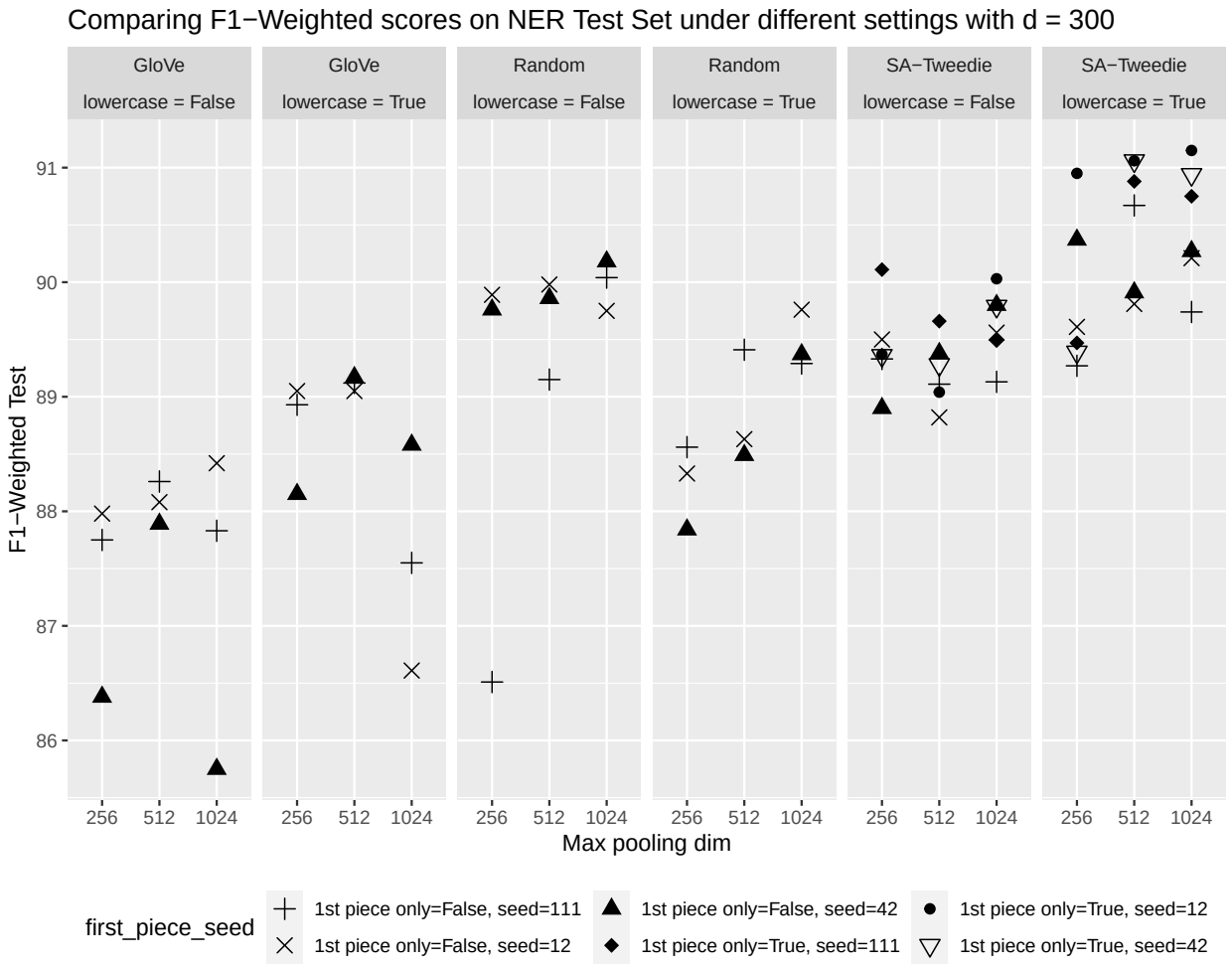

Figure 7.11: *Weighted F1 score on NER test set for different settings. All embeddings used 300-dimensional representations.*

The random embedding even outperformed the GloVe embedding in this task. Further investigation show that the model using GloVe embedding tend to quickly overfit (see Figure 7.12) in just a couple of epochs.

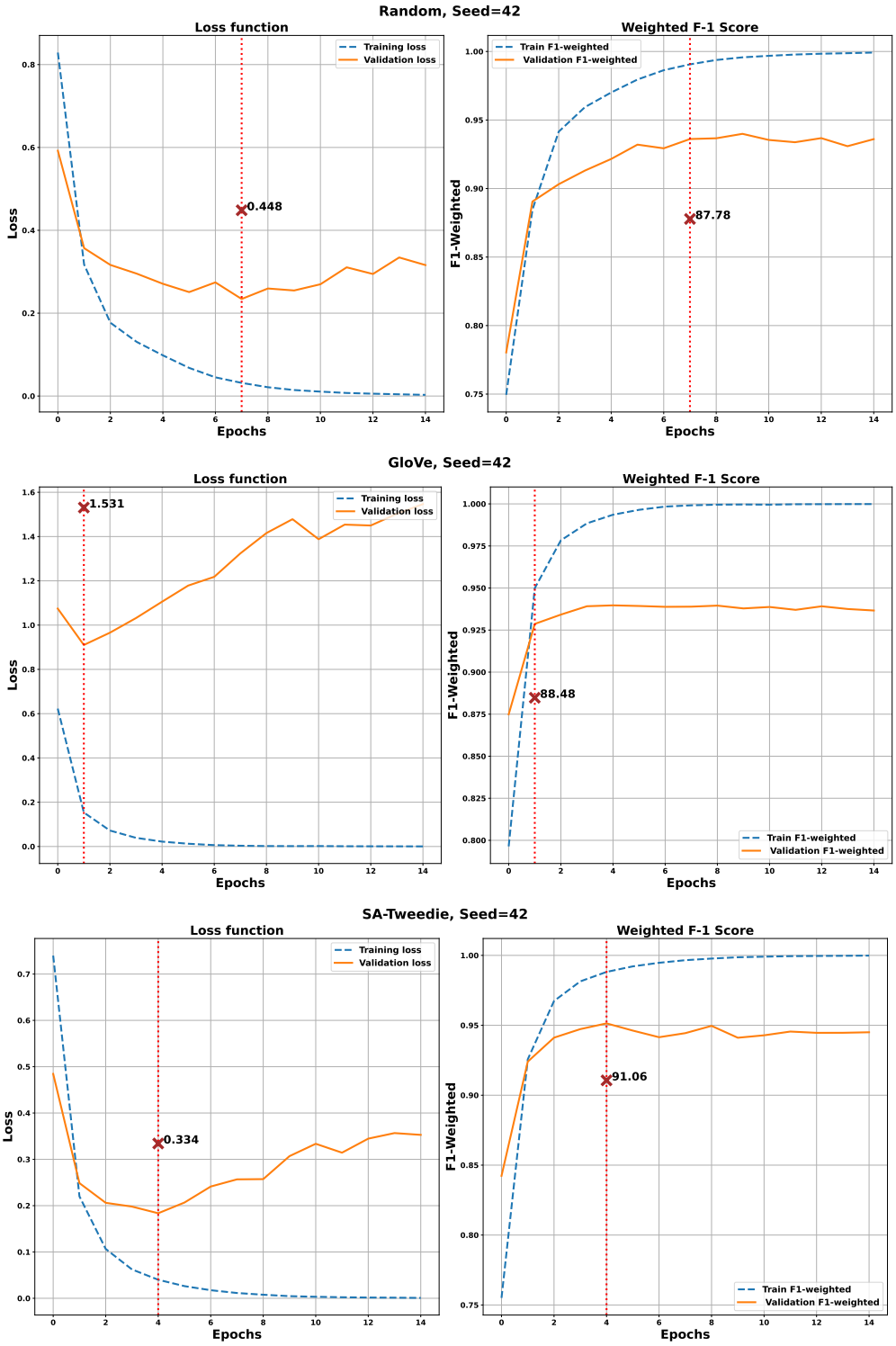

Figure 7.12: *Training and validation loss along with training and validation weighted F1 score for 15 epochs. Top row: random embedding with global seed 42. Middle row: GloVe embedding with global seed 42. Bottom row: SA-Tweedie embedding with global seed 42. The loss and weighted F1 score for test data are marked with cross marks with value given beside them.*

In fact, majority of the GloVe cases show drastic overfit starting from the second epoch. The random embedding starts with no information and then progresses little by little to reduce the training loss until a certain point. The SA-Tweedie is in between the the two. A few epochs of training and parameter updates still benefits the model with SA-Tweedie embedding.

A slight surprise happens when we compare the models that convert all words to lower-case letters with those that are cased. It is often thought that named entities tend to use capital letters and therefore keeping capital letters may improve the performance. This is the case with the random embedding. The middle two panels in Figure 7.11 show that the extra information of capital versus lower case letters provides an additional boost to the model using random embedding. On the other hand, the trend is opposite for GloVe or SA-Tweedie embedding. We see a slight decay here in the performance of using capital letters compared to the lower cases for both embedding methods. This tells that the embeddings themselves already carried more information than what was provided in the extra column of indicator variable of capital versus lower case letters. Instead of being beneficial, the single extra dimension of indicator variable we added in the model served as a nuisance. To make capital letters helpful in prediction, the embedding of the words with capital letters should be learned during the whole training process instead of compensating at the end. We could not take this direction for the dissertation due to time limitation. A more suitable approach is to learn word embeddings for both capital and lower case words from the start but this significantly increases the computational demand which requires access to many TPUs or pods of GPUs for extended time access. In the current settings, SA-Tweedie using only first piece token and lower case letters outperforms all other methods and settings.

We also examined different options of vector representations including using $\boldsymbol{\beta}$ alone, using $\widetilde{\boldsymbol{\beta}}$ alone, concatenation of the $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$, and average of $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$. These settings were solely experimented on SA-Tweedie embeddings since they are not available for the GloVe or random embedding. Our experiments show that using $\boldsymbol{\beta}$ or $\widetilde{\boldsymbol{\beta}}$ alone appear to be sufficient

and they give better results than using the average embeddings (results not shown). Figure 7.13 shows the weighted F1 score for the test set using embedding dimension $d = 100$. Compared to the results with $d = 300$, all models exhibited slightly lower performance in the $d = 100$ case. In this figure, we presented the results of using $\boldsymbol{\beta}$ or $\widetilde{\boldsymbol{\beta}}$ alone from three different versions of SA-Tweedie. The 19_3_$june$17 and 19_3_$june$30 versions refer to SA-Tweedie embedding obtained from Chapter 4. The dates refer to when the embeddings were retrieved from the training process. In both cases, the model training process is close to convergence but not converged yet. There were no constraints in those model parameters. As we know the model without constraints on parameters could have many different solutions. The 19_3_$june$17 and 19_3_$june$30 versions are just two of the many possible approximation to the solutions. Their performance appear to be similar in some cases but each also has its own value. For example, the 19_3_$june$17 version with $\boldsymbol{\beta}$ seems to have slightly lower performance when the max pooling dimension is 512 when compared to the 19_3_$june$30 version, which improved the performance for max pooling dimension 512 but at the expense of slightly increasing the variation in performance with max pooling dimension 256. The difference is not very significant in Figure 7.13. The 19_6_$june$30 version refers to the SA-Tweedie model described in subsection 6.3 of Chapter 6, in which quadratic constraints are included in the objective function. The constraints help with model identifiability and also lead to slightly better performance. In the top right corner panel of Figure 7.13, it achieved the best performance. Unfortunately, we only trained on $d = 100$. The constraints add considerable computational cost. We will have to defer further investigation of the 19-6 version in future study.

In terms of how the various model embeddings perform under different max pooling dimensions, we see that SA-Tweedie and random embedding both show better performance when no max pooling layer (corresponds to max pooling dim = 1024) was included and the same seed is used. The bidirectional RNN layer in the architecture has output dimension 1024. The max pooling reduces the output dimension to a lower number which corresponds

to summarizing the information from a higher dimension to a lower dimensional space. Such max pooling seems to be helpful for GloVe embedding as we see that the model with GloVe embedding showed better prediction in weighted F1 score when the max pooling dimension is 512 compared to the GloVe embedding model without max pooling. The SA-Tweedie or random embedding did not enjoy such contribution. Instead, both the SA-Tweedie or random embedding models showed better performance when no max pooling layer was included compared to their counterparts with a max pooling layer.

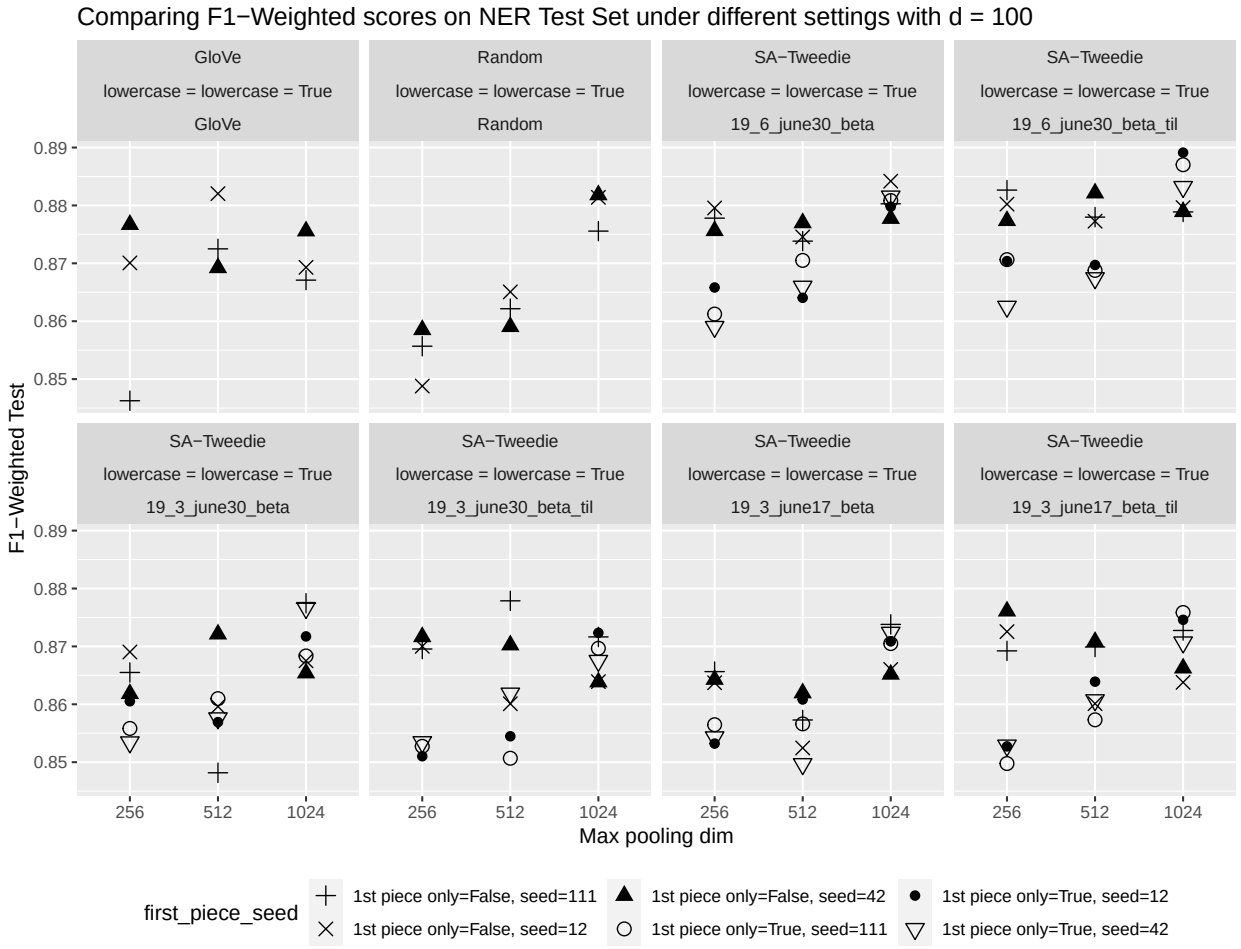

Figure 7.13: *F1 wieghted score on NER test set for different settings. All embeddings used 100-dimensional representations.*

It is still possible to adjust some of the tuning parameters to achieve better performance of

prediction. For example, with SA-Tweedie embedding of dimension $d = 768$ from 19_3_june30 version, the concatenation of $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$ leads to weighted F1 score of 94.83% for the validation data and weighted F1 score of 91.24% on the test data. Figure 7.14 depicts the training trajectory. The validation loss increases slowly as the number of epochs increases. The validation F1-weighted score quickly rises to the near 94% and stayed around there for a while.

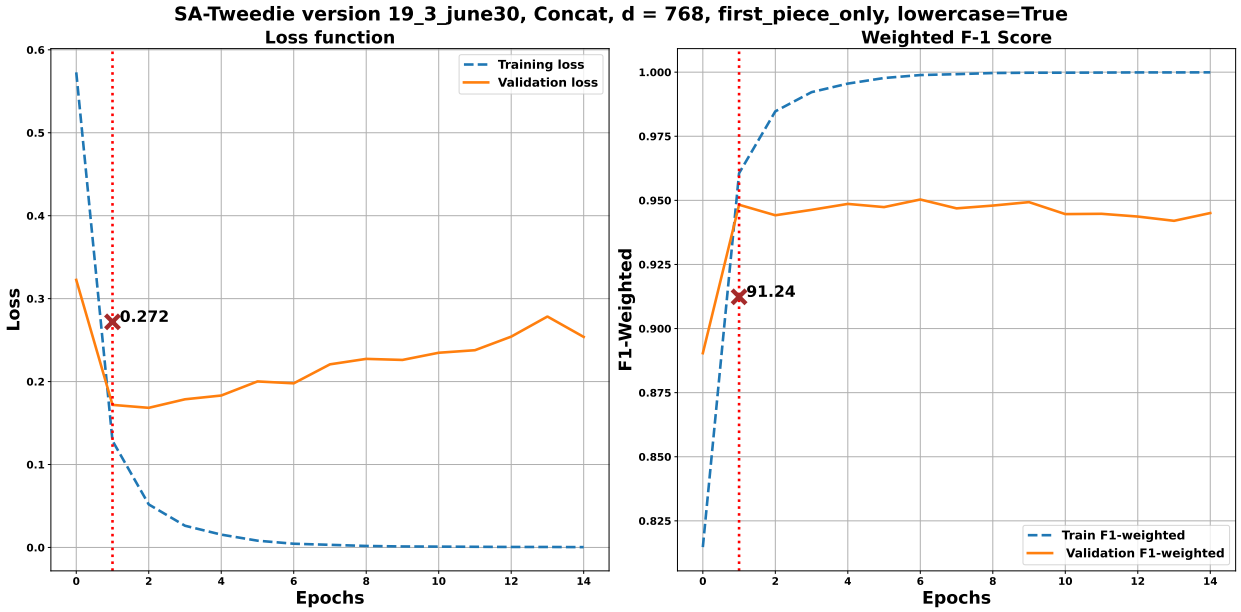

Figure 7.14: *Training and validation loss curve and F1-weighted score on NER task from SA-Tweedie embedding using concatenated $\boldsymbol{\beta}$ and $\widetilde{\boldsymbol{\beta}}$ with 768-dimensional representation. The loss and F1-weighted score on test set is marked along with their value.*

Discussion and Summary

In this dissertation, we studied methods for analyzing the co-occurrence count data from unstructured text data. We extracted numerical data by defining windows of co-occurrence of word-word pairs using weighted count on the continuous scale. Positive probability mass is allowed for zero counts.

We proposed several likelihood-based mixture models with shared parameters to accommodate large amount of zero observations. We modeled the relevance of user-item or word-word pairs by cosine similarity of their unknown dense vector representations. Due to no covariates observed, we have to alternately use part of the model parameters as fixed values to update the remaining part of the parameters. The unknown values are estimated via the shared parameter alternating regression models. Under this framework, we presented detailed study of Shared parameter Alternating Zero Inflated Gamma (SA-ZIG) regression and Shared parameter Alternating Tweedie (SA-Tweedie) regression, along with some applications in the Natural Language Processing (NLP) domain.

For the SA-ZIG regression, canonical link and log link models were considered. Two parameter updating schemes are proposed along with an algorithm to estimate the unknown parameters. Convergence analysis is presented analytically. Numerical studies showed that SA-ZIG with learning rate adjustment has satisfactory performance but the SA-ZIG using Fisher scoring without learning rate adjustment may fail to find the maximum likelihood estimate.

For the SA-Tweedie regression, multiple versions of an algorithm with different objective functions are proposed to estimate the parameters. Both models with and without constraints on parameters were studied. Updating formulae were given to obtain parameter estimates iteratively under each setting. A learning rate adjustment was used along with

the Fisher scoring method to help the algorithm stay on track of optimizing direction. The model without constraints on parameters (19-3 version) reduces the loss function quickly as the number of iteration increases even though the model parameters are not uniquely identifiable. The model with linear constraints has very slow convergence. The model with 19-6 version of the quadratic constraints had huge up and downs initially but was able to stabilize as the iterations increase. Simulation studies showed that our algorithm with Fisher scoring and learning rate adjustment outperforms the one without learning rate adjustment and gradient descent with Adam update. Pseudo-likelihood approach with alternating parameter update was also studied but found to be unsuitable for our shared parameter alternating regression models with unobserved covariates.

The proposed 19-3 and 19-6 versions of the algorithms were applied to Wikipedia dump data to obtain dense token/word vector representations, which are used in some NLP tasks such as finding the most similar words, sentiment analysis, and Named Entity Recognition (NER) task. Through experiments, we found that the word vectors trained from SA-Tweedie could find semantically similar words if the words are in its training vocabulary. Due to WordPiece tokenization used in forming the vocabulary, the SA-Tweedie embeddings are not very good as syntactic tasks. Comparisons with GloVe and random embeddings were made in sentiment analysis and in NER task. The SA-Tweedie had slightly lower or comparable performance to GloVe in a large scale moview review sentiment analysis but clearly outperformed GloVe in the CoNLL-2003 NER task. Both GloVe and SA-Tweedie pre-trained embeddings showed advantage in giving improved prediction accuracy and F1 scores compared to random embeddings.

Current results of our NLP analysis pertains to words in the vocabulary. Future study could be extended to obtaining vector representation of unseen words via combining WordPiece token representations with positional encoding. Context-based word embeddings may be obtained by fine-tuning a simple BiLSTM or Attention based Transformer in a language model with the pre-trained token embedding plus positional encoding. This requires exten-

sive further experiments.

A popular subject in statistical modeling is the inference on model parameters. Here, we would like to clarify that such inference may not be feasible and not of practical interest in matrix factorization problem. Considering only the i^{th} row of data $y_{i1}, y_{i2}, \ldots, y_{in}$, and we are in the 19-3 version framework. Denote $I_i(\boldsymbol{\beta}_i)$, $U_i(\boldsymbol{\beta}_i)$ to be the negative expectation of the Hessian matrix and the Score vector for i^{th} row, where $\boldsymbol{\beta}_i = (\mathbf{w}_i^\top, b_i)^\top$. The parameter updating equation is equivalent to the iteratively reweighted least square formula. That is,

$$\begin{aligned}
&\hat{\boldsymbol{\beta}}_{i,t+1} = \hat{\boldsymbol{\beta}}_{i,t} + \mathrm{lr} \cdot I^{-1}\left(\hat{\boldsymbol{\beta}}_{i,t}\right) U\left(\hat{\boldsymbol{\beta}}_{i,t}\right), \\
&\Leftrightarrow I\left(\hat{\boldsymbol{\beta}}_{i,t}\right)\hat{\boldsymbol{\beta}}_{i,t+1} = I\left(\hat{\boldsymbol{\beta}}_{i,t}\right)\hat{\boldsymbol{\beta}}_{i,t} + \mathrm{lr}\, U\left(\hat{\boldsymbol{\beta}}_{i,t}\right) \\
&\Leftrightarrow \hat{\boldsymbol{\beta}}_{i,t+1} = \left(\widetilde{X}^\top \widetilde{M}_{it} \widetilde{X}\right)^{-1} \widetilde{X}^\top \widetilde{M}_{it} \widetilde{z}_{it}.
\end{aligned}$$

where $\widetilde{M}_{i,t} = \mathrm{diag}\left(\widetilde{M}_{i1,t}, \ldots, \widetilde{M}_{in,t}\right)$, $\quad \widetilde{z}_{i,t} = (\widetilde{z}_{i1,t}, \cdots, \widetilde{z}_{in,t})$, and

$$\widetilde{M}_{ij,t} = \frac{1}{V_{ij}\left(\frac{\partial \eta_{ij}}{\partial \mu_{ij}}\right)^2\Big|_{\hat{\boldsymbol{\beta}}_{i,t}}}, \quad \text{with } \eta_{ij} = \log \mu_{ij} = \widetilde{\boldsymbol{w}}_j^\top \cdot \mathbf{w}_i + \widetilde{b}_j + b_i,$$

$$\widetilde{z}_{ij,t} = \hat{\eta}_{ij,t} + lr\left[(y_{ij} - \mu_{ij})\frac{\partial \eta_{ij}}{\partial \mu_{ij}}\right]\Big|_{\boldsymbol{\beta}_i = \hat{\boldsymbol{\beta}}_{i,t}}, \quad \widetilde{X} = \begin{pmatrix} 1 & \widetilde{w}_{11} & \cdots & \widetilde{w}_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \widetilde{w}_{n1} & \cdots & \widetilde{w}_{nd} \end{pmatrix}.$$

Given $\widetilde{w}_j, \widetilde{b}_j, j = 1, 2, \cdots, n, \hat{\boldsymbol{\beta}}_i$ has limiting distribution $N\left(\boldsymbol{\beta}_i, \widetilde{G}_i(\boldsymbol{\beta}_i, \widetilde{\boldsymbol{\beta}})\right)$ where $\widetilde{G}_i(\boldsymbol{\beta}_i, \widetilde{\boldsymbol{\beta}}) = \left(\widetilde{X}^\top \widetilde{M}_i \widetilde{X}\right)^{-1} \widetilde{X}' \widetilde{M}_i \,\mathrm{Var}(\widetilde{\mathbf{z}}_i) \widetilde{M}_i \widetilde{X} \left(\widetilde{X}^\top \widetilde{M}_i \widetilde{X}\right)^{-1}$ (Assume certain regularity conditions hold).

Similarly, given $\mathbf{w}_i, b_i, i = 1, 2, \cdots, n$, $\widehat{\widetilde{\boldsymbol{\beta}}}_j$ has limiting distribution $N\left(\widetilde{\boldsymbol{\beta}}_j, G_j(\boldsymbol{\beta}, \widetilde{\boldsymbol{\beta}}_j)\right)$

where $G_j(\boldsymbol{\beta}, \widetilde{\boldsymbol{\beta}}_j) = \left(X^\top M_j X\right)^{-1} X^\top M_j \operatorname{Var}(\mathbf{z}_j) M_j X \left(X^\top M_j X\right)^{-1}$,

$$\text{with } M_j = \frac{1}{V_{ij}\left(\frac{\partial \eta_{ij}}{\partial \mu_{ij}}\right)^2\Big|_{\hat{\widetilde{\boldsymbol{\beta}}}_{j,t}}}, \quad \text{with } \eta_{ij} = \log \mu_{ij} = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j$$

$$X = \begin{pmatrix} 1 & w_{11} & \cdots & w_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & w_{n1} & \cdots & w_{nd} \end{pmatrix}$$

$$z_{j,t} = (z_{j1,t}, \cdots, z_{jn,t}), \quad z_{kj,t} = \hat{\eta}_{kj,t} + lr \left[(y_{kj} - \mu_{kj}) \frac{\partial \eta_{kj}}{\partial \mu_{kj}}\right]\Big|_{\widetilde{\boldsymbol{\beta}}_j = \hat{\widetilde{\boldsymbol{\beta}}}_{j,t}},$$

Even though $\hat{\boldsymbol{\beta}}$ and $\hat{\widetilde{\boldsymbol{\beta}}}$ each is approximately normal given the other one, the circular dependence on each other make it impossible to consider inferences. Additionally, it may not be meaningful to consider the inference of $\boldsymbol{\beta}$ or $\widetilde{\boldsymbol{\beta}}$. This is because what we need is the relative position of all word vectors in some space. There are many such spaces available. Below is an example.

Consider two different parameterizations for $\log(\mu_{ij}) = \mathbf{w}_i^\top \widetilde{\boldsymbol{w}}_j + b_i + \widetilde{b}_j$:

$$\mathbf{w}_i, \widetilde{\boldsymbol{w}}_j, b_i, \widetilde{b}_j \tag{7.2.1}$$

$$c\mathbf{w}_i, \frac{\widetilde{\boldsymbol{w}}_j}{c}, b_i, \widetilde{b}_j \tag{7.2.2}$$

That is, in parameterization (7.2.1), $\mathbf{w}_i$ is vector representation for token i, $\widetilde{\boldsymbol{w}}_j$ is vector representation for token j. In parameterization (7.2.2), $c\mathbf{w}_i$ is vector representation for token i, $\widetilde{\boldsymbol{w}}_j/c$ is vector representation for token j, where c is any nonzero constant in $\mathbb{R}$. Both parameterizations give the same $\log(\mu_{ij})$ and the same cosine similarity between the two tokens. Therefore, testing if $\mathbf{w}_i$ or $\widetilde{\boldsymbol{w}}_j$ is equal to specific given value is not of practical interest. In fact, reparameterization with any scaling or permutation in elements in $\mathbf{w}_i$ and $\widetilde{\boldsymbol{w}}_j$ could still lead to the same $\log(\mu_{ij})$. Clearly the model parameters are not uniquely defined. Instead, what we care is the dot product between the two vectors. Instead of

inference on $\boldsymbol{\beta}$ or $\widetilde{\boldsymbol{\beta}}$, there are research interested in studying the identifiability after allowing permutation and scaling. See for example, Fu et al. 2019 and Gillis and Rajkó 2023 and the references therein. Model identifiability is beyond the scope of this dissertation.

Bibliography

A. Albert and J. A. Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1–10, 1984. ISSN 00063444. URL http://www.jstor.org/stable/2336390.

Paul S. Albert, Dean A. Follmann, and Huiman X. Barnhart. A generalized estimating equation approach for modeling random length binary vector data. *Biometrics*, 53(3): 1116–1124, 1997. URL http://www.jstor.org/stable/2533568.

Edgar Altszyler, Sidarta Ribeiro, Mariano Sigman, and Diego Fernández Slezak. The interpretation of dream meaning: Resolving ambiguity using latent semantic analysis in a small corpus of text. *Consciousness and Cognition*, 56:178–187, 2017. ISSN 1053-8100. doi: https://doi.org/10.1016/j.concog.2017.09.004. URL https://www.sciencedirect.com/science/article/pii/S1053810017301034.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Wagner Hugo Bonat and Célestin C. Kokonendji. Flexible tweedie regression models for continuous data. *Journal of Statistical Computation and Simulation*, 87(11):2138–2152, 2017. URL https://doi.org/10.1080/00949655.2017.1318876.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Dan Ciresan, Alessandro Giusti, Luca Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy im-

ages. volume 25, 2012. URL https://proceedings.neurips.cc/paper/2012/file/459a4ddcb586f24efd9395aa7662bc7c-Paper.pdf.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. *ArXiv*, abs/2003.10555, 2020.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, page 191–198. Association for Computing Machinery, 2016. ISBN 9781450340359. doi: 10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

David Roxbee Cox and David V. Hinkley. *Theoretical statistics*. CRC Press, 2017. ISBN 9781482214925.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Juan Du. Which estimator of the dispersion parameter for the gamma family generalized linear models is to be chosen? *Master thesis, Dalarna University*, 2007. URL http://www.statistics.du.se/essays/D07C.Du.pdf.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61): 2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. Position information in transformers: An overview. *CoRR*, abs/2102.11090, 2021. URL https://arxiv.org/abs/2102.11090.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the*

43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), pages 363–370, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219885. URL https://aclanthology.org/P05-1045.

Xiao Fu, Kejun Huang, Nicholas D. Sidiropoulos, and Wing-Kin Ma. Nonnegative matrix factorization for signal and data analytics: Identifiability, algorithms, and applications. *IEEE Signal Processing Magazine*, 36(2):59–80, mar 2019. doi: 10.1109/msp.2018.2877582. URL https://doi.org/10.1109%2Fmsp.2018.2877582.

Nicolas Gillis and Ró bert Rajkó. Partial identifiability for nonnegative matrix factorization. *SIAM Journal on Matrix Analysis and Applications*, 44(1):27–52, jan 2023. doi: 10.1137/22m1507553. URL https://doi.org/10.1137%2F22m1507553.

Geof H. Givens and Jennifer A. Hoeting. *Computational Statistics, Second Edition*. John Wiley & Sons, Inc., 2 edition, 2013. ISBN 9780470533314; 0470533315; 9781118555552; 1118555554.

S.J. Haberman. *The Analysis of Frequency Data*. Midway reprint. University of Chicago Press, 1977. ISBN 9780226311852. URL https://books.google.com/books?id=TQvvAAAAMAAJ.

Thomas R. Ten Have, Allen R. Kunselman, Erik P. Pulkstenis, and J. Richard Landis. Mixed effects logistic regression models for longitudinal binary response data with informative drop-out. *Biometrics*, 54(1):367–383, 1998. URL http://www.jstor.org/stable/2534023.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2019. URL https://arxiv.org/abs/1909.10351.

Bent Jorgensen. *The Theory of Dispersion Models*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1997.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Ro{bert}a: A robustly optimized {bert} pretraining approach, 2020. URL https://openreview.net/forum?id=SyxS0T4tvS.

Tzon-Tzer Lu and Sheng-Hua Shiou. Inverses of 2×2 block matrices. *Computers & Mathematics with Applications*, 43(1):119–129, 2002. ISSN 0898-1221. doi: https://doi.org/10.1016/S0898-1221(01)00278-4. URL https://www.sciencedirect.com/science/article/pii/S0898122101002784.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL https://aclanthology.org/P11-1015.

Ian C. Marschner. glm2: Fitting generalized linear models with convergence problems. *The R Journal*, 3:12–15, 2011.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013a. URL https://arxiv.org/abs/1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013b. URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Tomas Mikolov, Scott Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013c. URL https://www.microsoft.com/en-us/research/publication/linguistic-regularities-in-continuous-space-word-representations/.

Elizabeth Dastrup Mills. Adjusting for covariates in zero-inflated gamma and zero-inflated log-normal models for semicontinuous data. *PhD dissertation, University of Iowa*, 2013. doi: 10.17077/etd.7v3bafbd.

Lawrence H Moulton, Frank C Curriero, and Paulo F Barroso. Mixture models for quantitative hiv rna data. *Statistical Methods in Medical Research*, 11(4):317–325, 2002. URL https://doi.org/10.1191/0962280202sm292ra.

J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7:308–313, 1965. URL https://doi.org/10.1093/comjnl/7.4.308.

Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *ArXiv*, 2011. URL https://doi.org/10.48550/arXiv.1106.5730.

Aline Araújo Nobre, Marilia Sá Carvalho, Rosane Härter Griep, Maria de Jesus Mendes da Fonseca, Enirtes Caetano Prates Melo, Itamar de Souza Santos, and Dora Chor. Multinomial model and zero-inflated gamma model to study time spent on leisure time physical activity: an example of elsa-brasil. *Revista de Saúde Pública*, 51:76, Jan. 2017. doi: 10.11606/s1518-8787.2017051006882. URL https://www.revistas.usp.br/rsp/article/view/138333.

M. R. Osborne. Fisher's method of scoring. *International Statistical Review / Revue Internationale de Statistique*, 60(1):99–117, 1992. ISSN 03067734, 17515823. URL http://www.jstor.org/stable/1403504.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

S. Ravichandiran. *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT*. Packt Publishing, 2021. ISBN 9781838826239. URL https://books.google.com/books?id=CvsWEAAAQBAJ.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019. URL https://arxiv.org/abs/1910.01108.

Mervyn Joseph Silvapulle. On the existence of maximum likelihood estimators for the binomial response models. *Journal of the royal statistical society series b-methodological*, 43: 310–313, 1981.

Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. Deep learning for recommender systems: A netflix case study. *AI Magazine*, 42(3):7–18, Nov. 2021. doi: 10.1609/aimag.v42i3.18140. URL https://ojs.aaai.org/index.php/aimagazine/article/view/18140.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Mengqiu Wang and Christopher D. Manning. Effect of non-linear deep architecture in sequence labeling. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1285–1291, Nagoya, Japan, October 2013. Asian Federation of Natural Language Processing. URL https://aclanthology.org/I13-1183.

Xue-Xin Wei, Ding Zhou, Andres D. Grosmark, Zaki Ajabi, Fraser T. Sparks, Pengcheng Zhou, Mark P. Brandon, Attila Losonczy, and Liam Paninski. A zero-inflated gamma model for post-deconvolved calcium imaging traces. *Neural data science / analysis*, 3(2), 2020.

Margaret C. Wu and Raymond J. Carroll. Estimation and comparison of changes in the presence of informative right censoring by modeling the censoring process. *Biometrics*, 44 (1):175–188, 1988. URL http://www.jstor.org/stable/2531905.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the

gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

Appendix A

Appendix A: Second order partial derivatives of log likelihood of alternating Tweedie regression

The second order partial derivatives for $\ell_{ij}^{(0)}$ and $\widetilde{\ell}_{ij}^{(0)}$ are

$$
\begin{aligned}
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial w_{i_1 k} \partial w_{i_2 r}} &= -(2-p_{ij}) \frac{\mu_{ij}^{1-p_{ij}}}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial w_{i_2 r}} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}=0) \\
&= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot \widetilde{w}_{jr} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}=0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{w}_{j_2 r}} &= -(2-p_{ij}) \frac{\mu_{ij}^{1-p_{ij}}}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial \widetilde{w}_{j_2 r}} \cdot w_{ik} \cdot I(j=j_1=j_2) I(y_{ij}=0) \\
&= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot w_{ir} \cdot w_{jk} \cdot I(j=j_1=j_2) I(y_{ij}=0), \\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial w_{i_1 k} \partial b_{i_2}} &= -(2-p_{ij}) \frac{\mu_{ij}^{1-p_{ij}}}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial b_{i_2}} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}=0) \\
&= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}=0), \\
\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{b}_{j_2}} &= -(2-p_{j}) \frac{\mu_{ij}^{1-p_{ij}}}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial \widetilde{b}_{j_2}} \cdot w_{ik} \cdot I(j=j_1=j_2) I(y_{ij}=0) \\
&= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{jj}} \cdot w_{ik} \cdot I(j=j_1=j_2) I(y_{ij}=0), \\
\frac{\partial^2 \ell_{ij}^{(0)}}{\partial b_{i_1} \partial b_{i_2}} &= -(2-p_{ij}) \frac{\mu_{ij}^{1-p_{ij}}}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial b_{i_2}} \cdot I(i=i_1=i_2) I(y_{ij}=0) \\
&= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot I(i=i_1=i_2) I(y_{ij}=0),
\end{aligned}
$$

$$\begin{aligned}\frac{\partial^2 \widetilde{\ell}_{ij}^{(0)}}{\partial \widetilde{b}_{j_1} \partial \widetilde{b}_{j_2}} &= -(2-p_{jj}) \frac{\mu_{ij}^{1-p_{ij}}}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial \widetilde{b}_{j_2}} \cdot I(j=j_1=j_2) I(y_{ij}=0) \\ &= -\frac{(2-p_{ij})}{\phi_{ij}} \cdot \mu_{ij}^{2-p_{ij}} \cdot I(j=j_1=j_2) I(y_{ij}=0).\end{aligned}$$

Note that above second order derivatives and their expectations are equal since the canonical link is used.

Next, we will consider the second order partial derivatives and their expectations for the case that $y_{ij} > 0$:

$$\begin{aligned}&\frac{\partial^2 \ell_{ij}^{(1)}}{\partial w_{i_1 k} \partial w_{i_2 r}} = \frac{\partial \left(\partial \ell_{ij}^{(1)} / \partial w_{ik} \right)}{\partial w_{i_2 r}} \\ = &\left[\frac{\partial \left(\frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \right)}{\partial w_{i_2 r}} \frac{\mu_{ij}}{V_{ij}} \widetilde{w}_{jk} I(i=i_1=i_2) + \frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \frac{\partial \left(\frac{\mu_{ij}}{V_{ij}} \widetilde{w}_{jk} \right)}{\partial w_{i_2 r}} I(i=i_1=i_2) \right] I(y_{ij}>0).\end{aligned}$$

Taking negative expectation, we get

$$\begin{aligned}-E\left[\frac{\partial^2 \ell_{ij}^{(1)}}{\partial w_{i_1 k} \partial w_{i_2 r}} \middle| y_{ij} > 0 \right] &= \frac{1}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial w_{i_2 r}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}>0) \\ &= \frac{1}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial w_{i_2 r}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}>0) \\ &= \frac{1}{\phi_{ij}} \cdot \mu_{ij} \cdot \widetilde{w}_{jr} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot \widetilde{w}_{jk} \cdot I(i=i_1=i_2) I(y_{ij}>0) \\ &= \frac{\mu_{ij}^2}{\phi_{ij} V_{ij}} \cdot \widetilde{w}_{jk} \cdot \widetilde{w}_{jr} \cdot I(i=i_1=i_2) I(y_{ij}>0).\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 \ell_{ij}}{\partial b_{i_1} \partial b_{i_2}} &= \frac{\partial \left(\partial \ell_{ij} / \partial b_{i_1} \right)}{\partial b_{i_2}} \\ &= \frac{\partial \left(\frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \right)}{\partial b_{i_2}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot 1 \cdot I(i=i_1=i_2) + \frac{y_{ij}-\mu_{ij}}{\phi_{ij}} \cdot \frac{\partial \left(\frac{1}{V_{ij}} \cdot \mu_{ij} \right)}{\partial b_{i_1}} \cdot I(i=i_1=i_2)\end{aligned}$$

and the negative expectation is

$$\begin{aligned}
-E\left[\frac{\partial^2 \ell_{ij}}{\partial b_{i_1} b_{i_2}}\right] &= \frac{1}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial b_{i_2}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot I(i = i_1 = i_2) \qquad (\because E\left(y_{ij} - \mu_{ij}\right) = 0) \\
&= \frac{1}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial b_{i_2}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot I(i = i_1 = i_2) \\
&= \frac{1}{\phi_{ij}} \cdot \mu_{ij} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot I(i = i_1 = i_2) \\
&= \frac{\mu_{ij}^2}{\phi_{ij} V_{ij}} \cdot I(i = i_1 = i_2).
\end{aligned}$$

Similarly, we compute the second order partial derivatives and their expectations for

$$\begin{aligned}
\frac{\partial^2 \ell_{ij}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{w}_{j_2 r}} &= \frac{\partial \left(\partial \ell_{ij} / \partial \widetilde{w}_{j_1 k}\right)}{\partial \widetilde{w}_{j_2 r}} \\
&= \frac{\partial \left(\frac{y_{ij} - \mu_{ij}}{\phi_{ij}}\right)}{\partial \widetilde{w}_{j_2 r}} \frac{\mu_{ij}}{V_{ij}} w_{ik} \, I(j = j_1 = j_2) + \frac{y_{ij} - \mu_{ij}}{\phi_{ij}} \frac{\partial \left(\frac{\mu_{ij}}{V_{ij}} w_{ik}\right)}{\partial \widetilde{w}_{jr}} I(j = j_1 = j_2)
\end{aligned}$$

Taking negative expectation

$$\begin{aligned}
-E\left[\frac{\partial^2 \ell_{ij}}{\partial \widetilde{w}_{j_1 k} \partial \widetilde{w}_{j_2 r}}\right] &= \frac{1}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \widetilde{w}_{j_2 r}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot w_{ik} \cdot I(j = j_1 = j_2), \qquad (\because E\left(y_{ij} - \mu_{ij}\right) = 0) \\
&= \frac{1}{\phi_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial \eta_{ij}} \cdot \frac{\partial \eta_{ij}}{\partial \widetilde{w}_{j_2 r}} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot w_{ik} \cdot I(j = j_1 = j_2) \\
&= \frac{1}{\phi_{ij}} \cdot \mu_{ij} \cdot w_{ir} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot w_{ik} \cdot I(j = j_1 = j_2) \\
&= \frac{\mu_{ij}^2}{\phi_{ij} V_{ij}} \cdot w_{ik} \cdot w_{ir} \cdot I(j = j_1 = j_2).
\end{aligned}$$

Similarly,

$$-E\left[\frac{\partial^2 \ell_{ij}}{\partial \widetilde{b}_{j_1} \partial \widetilde{b}_{j_2}}\right] = \frac{1}{\phi_{ij}} \cdot \mu_{ij} \cdot \frac{1}{V_{ij}} \cdot \mu_{ij} \cdot I(j = j_1 = j_2) = \frac{\mu_{ij}^2}{\phi_{ij} V_{ij}} \cdot I(j = j_1 = j_2)$$

Below is about the off diagnol elements in the Information matrix which is non zero.

$$
\begin{aligned}
-E\left[\frac{\partial^2 \ell_{ij}}{\partial w_{i_1 k}\partial b_{i_2}}\right] &= -E\left[\frac{\partial}{\partial b_{i_2}}\left(\frac{y_{ij}-\mu_{ij}}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot\widetilde{w}_{jk}\cdot I(i=i_1)\right)\right] \\
&= \frac{\partial \mu_{ij}}{\partial b_{i_2}}\frac{1}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot\widetilde{w}_{jk}\cdot I(i=i_1) \\
&= \frac{\partial \mu_{ij}}{\partial \eta_{ij}}\cdot\frac{\partial \eta_{ij}}{\partial b_{i2}}\cdot\frac{1}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot\widetilde{w}_{jk}\cdot I(i=i_1) \\
&= \mu_{ij}\cdot\frac{1}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\mu_{ij}\cdot\widetilde{w}_{jk}\cdot I(i=i_1=i_2) \\
&= \mu_{ij}^2\cdot\frac{1}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot\widetilde{w}_{jk}\cdot I(i=i_1=i_2), \\
-E\left[\frac{\partial^2 \ell_{ij}}{\partial \widetilde{w}_{j_1 k}\partial \widetilde{b}_{j_2}}\right] &= \mu_{ij}^2\cdot\frac{1}{\phi_{ij}}\cdot\frac{1}{V_{ij}}\cdot w_{ik}\cdot I(j=j_1=j_2).
\end{aligned}
$$