NORTHWESTERN UNIVERSITY

Ensembling and Data Selection for Neural Language Models, and Analysis of F-measure

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Statistics

By

Wei Ju

EVANSTON, ILLINOIS

December 2021

ABSTRACT

Ensembling and Data Selection for Neural Language Models, and Analysis of F-measure

Wei Ju

Language models are the foundation of many natural language tasks such as machine translation, speech recognition, and dialogue systems. Modeling the probability distributions of text accurately helps capture the structures of language and extract valuable information contained in various corpora. In recent years, many advanced models have achieved state-of-the-art performance on various large, diverse language modeling benchmarks. However, training these advanced models on large benchmark corpora can be difficult due to memory constraints and prohibitive computational costs. Of these advanced models, deep ensemble model combines several individual models to obtain better generalization performance. But training standard ensemble of language models over the entire corpus may ignore the inherent characteristics of text and the specialization of ensemble model on certain subset of text.

Many natural language processing (NLP) systems are evaluated using the F-measure, which is computed as the harmonic mean of recall and precision. The F-measure is usually estimated on the sample set, and many useful approaches have been presented to estimate the standard error of the F-measure. However, no efficient approach has been proposed to compare the accuracy of the

estimated F-measures for different algorithms.

In this dissertation, we develop several statistical methodologies for more effective and efficient language modeling, and we also propose a framework for comparing different approaches of estimating F-measures. First, we correct some estimation details in Wong's approach [1], and give estimation of the covariance between two F-measures in JVESR's approach. We present an experimental framework for comparing JVESR's approach and Wong's approach in estimating F-measures of two algorithms. Second, we introduce a novel algorithm called BaMME for effective ensemble deep neural network training. By applying BaMME, each ensemble model is navigated to focus on text with certain characteristics in the training, and our experiments demonstrate that BaMME outperforms the traditional ensemble modeling. Finally, we propose two new subsampling methods, CLIS and CLISIER, for selecting text data for efficient sentence-level RNN language modeling. These two subsampling methods apply the clustering technique and the importance sampling technique to make the selected dataset as diverse and informative as possible. In the CLISIER method, we remove inefficiency sentences by building the Dataset Map. Both of CLIS and CLISIER outperform the baseline methods in training data selection for language modeling. Moreover, CLISIER provides an efficient way to build Dataset Maps for language modeling.

ACKNOWLEDGEMENTS

During the past five years, I have met many wonderful people at Northwestern University. I would like to sincerely thank them for their help and support.

I would like to express my deepest gratitude to my advisor Professor Wenxin Jiang for his valuable guidance and support. His insightful advice and consistent encouragements give me the directions and inspirations to explore various projects and knowledge. I have benefited greatly from his wealth of knowledge and experience. His advice and help also have a profound impact on my life. Without his supports and patience, this degree could not have been accomplished.

I would like to show my special gratitude to my co-advisor Professor Douglas Downey for his patient support and valuable inputs. His insightful suggestions and feedback push me to sharpen my thinking and give me lots of inspirations in my research work and life. It has been a a pleasant journey to work with Professor Downey.

I would like to thank my thesis committee member Professor Martin Tanner, who gives me professional suggestions and helpful feedback. I enjoy Professor Tanner's lectures in Bayesian Statistics, which inspire me to explore more knowledge about parameter estimations in my research study.

My thanks also go to all the people with whom I have had the chances to work during my Ph.D. studies. I am grateful to the faculty, the graduate students, and the staff at the Northwestern Statistics Department for their kind support and stimulating discussions. I would also like to thank

the researchers at Northwestern WebSAIL for their insightful discussions.

In the end, I would like to give my special thanks to my family and my friends for their unconditional love, support and encouragement.

TABLE OF CONTENTS

ABSTRACT 3

Acknowledgments 5

Table of Contents 7

List of Tables 10

List of Figures 13

Chapter 1: Introduction 15

1.1 Overview 15

1.2 Main Contributions and Outline 16

Chapter 2: Comparison of Two Methods for Estimating *F*-measure and Its Variance . . 20

2.1 Introduction 20

2.2 Related Work and Knowledge 21

2.3 Methodology 26

2.4 Experiments 30

2.5 Conclusion 34

Chapter 3: Language Models 36

3.1 *N*-gram Language Models 37

3.2 Recurrent Neural Network Language Models 39

3.3 Effective and Efficient Language Modeling 42

Chapter 4: Some Effective Modeling Methods for Ensemble Training 45

4.1 Introduction 45

4.2 Related Work and Knowledge 46

4.3 Methodology 48

4.4 Experiments 52

4.5 Further Analysis and Experiments 55

4.6 Conclusion 60

Chapter 5: Training Data Sampling for Efficient Language Modeling 61

5.1 Introduction 61

5.2 Related Work and Knowledge 63

5.3 Methodology 65

5.4 Experiments 74

5.5 Conclusion 79

Chapter 6: Conclusion and Future Work 81

References 85

Appendix A: Computation of Covariance of Two *F*-measures 93

Appendix B: More Details of Algorithm 1 95

Appendix C: Experimental Setting and Hyper-parameters 97

Appendix D: Examples to Illustrate Batch Weight Smoothing 98

LIST OF TABLES

2.1 Confusion matrix for a binary classification model on the dataset D. 22

2.2 Information of the three imbalanced datasets and some parameters for subsampling procedure. 31

2.3 Results of comparing JVESR's estimation method against Wong's estimation method on the three imbalanced datasets. JVESR's method achieves more accurate estimation results (closer to the simulated results) compared with Wong's method on all the three datasets. 32

4.1 Results of BaMME and Baseline methods on the validation and testing datasets of PTB and WT2 corpora. BaMME outperforms the Baseline method for both of the two datasets. 54

4.2 Validation and testing results (on PTB) of BaMME with different weight decaying rate. The models achieve better validation and testing perplexities when we use relatively smaller weight decaying rate. 55

4.3 Performance of generating models on the generated texts. The ith generating model performs the best on the ith generated dataset, which indicates that the datasets are properly generated. 56

4.4 Results of Baseline and Batch-wise Pre-selection of Ensembles methods on the synthetic data. The latter method produces the best testing result when 35% of the text in each batch of the synthetic data is used in the pre-training and pre-evaluation procedures. 57

4.5 Results of Baseline and Clustering-based Selection of Ensembles methods on the synthetic data. The latter method outperforms the baseline method, and produces the best result when 30% of the text in each batch of the synthetic data is used in the pre-evaluation procedure. 59

5.1 Results of CLIS and Baseline methods on Wikitext-103 corpus. CLIS outperforms the Baseline methods, and achieves lowest ppl when the number of clusters is set to 60. 76

5.2 Results of CLISIER and Baseline methods on Wikitext-103 corpus. CLISIER outperforms the Baseline methods. CLISIER achieves lowest ppl when 20% of sequences are removed from the randomly selected set. 77

5.3 Results of CLISIER and Baseline methods on the One Billion Word Benchmark corpus. CLISIER outperforms the Baseline methods. CLISIER achieves lowest ppl when 20% of sequences are removed from the randomly selected set for the 1M token and 2M token cases, and achieves the lowest ppl when 40% of sequences are removed from the random set for the 500k token case. 79

C.1 Hyper-parameters used in modeling. 97

C.2 Hyper-parameters used for batch weight smoothing. 97

D.1 Example 1 of batch weight smoothing. 98

D.2 Example 2 of batch weight smoothing. 99

LIST OF FIGURES

2.1 Illustration of the Comparative Experiments. We start with (1) shuffling the initial dataset, and then (2) split the shuffled dataset into training set and testing set, followed by (3) model training and random subsampling the testing set for (4) evaluation and simulation processes. We repeat the subsampling procedure and procedure (4) $\boldsymbol{c}$ times, and more details of the computations in (4) are described in §2.3.2. In the Figure, s_i^2 represents the estimated sample variance Var($\bar{f}_i$), and s_{12}^2 represents the sample covariance Cov($\bar{f}_1, \bar{f}_2$). 28

3.1 A single layer representation of RNN language model. The input of the RNN operation is composed of the hidden representation $\boldsymbol{h}_m$ and the embedding representation of token w_{m+1}. 40

4.1 Density plot of weights for a 4-element ensemble model. Most of the weights are extreme values before smoothing. 53

5.1 Dataset Map for RNN Model on 10M Wikitext-103. *Easy-to-learn* examples fall in the bottom-left region, while *hard-to-learn* examples lie in the top-left region. The examples fall in the upper right quadrant are referred as *ambiguous* examples. . 69

5.2 Illustration of the CLISIER method. We first (1) randomly sample a 10M set from the initial training set, and (2) use an RNN model to train it. Then, we (3) build the Dataset Map with calculated mean log-perplexity and variability, and remove $p\%$ of examples with (4) a pre-selection procedure. We (5) cluster the pre-selected set and calculate the sampling distribution. This is followed by (6) importance sampling to get the subsampled training set. Finally, we (7) train the RNN model on the subsampled training set with corrective weight. 72

CHAPTER 1
INTRODUCTION

1.1 Overview

In the past decades, machine learning algorithms get more and more attention from people and have been applied in many applications in our daily life: automatic text translation and voice recognition on the web and in our social media, commodity classification and click-through rate prediction in electronic commerce help businesses better understand customers' preferences, etc. To further develop these machine learning algorithms and make the predictions more accurate, we need to explore more about the evaluation methods and their properties. The F-measure [2] is one of the most important measures in evaluating machine learning algorithms, and it is usually estimated on large sample datasets. People are seeking more accurate methods ([1], [3]–[9]) to estimate the F-measure, so that machine learning models will get more efficient and accurate evaluations. How to compare the accuracy of two or more estimation methods in estimating F-measure and its variance is meaningful, since more accurate estimation method will lead to better evaluation of the machine learning algorithms.

Among various machine learning algorithms, computational algorithms and representations for processing natural human language have brought great impact on our life ([10]–[12]) since there are a lot of related applications: translating between different languages, answering questions, summarizing long text, etc. The goal of these applications is to produce proper word sequences as output, and people will compute the probability of the output text to ensure the generated texts are fluent. Pre-training better language models will benefit for the comparison of the output texts in

these tasks.

Language Modeling is an important task as modeling the probability distributions of text accurately will help capture the structures of language and extract useful information contained in various corpora. The pre-trained language models can be used to evaluate the output of downstream tasks such as machine translation ([10], [13]), dialogue systems ([12], [14], [15]) and summarization ([16], [17]), etc. How to effectively and efficiently train language models on various corpora is a long standing challenging problem. Especially when the corpus is pretty large, we have to seek efficient ways to subsample a training set from the original corpus which contains rich information [18], since training sophisticated models on such large corpus can be hard due to memory constraints and prohibitive computational costs. Even training language models on relatively smaller corpora, it is still difficult to utilize simple RNN models to capture the the inherent characteristics of text during training.

1.2 Main Contributions and Outline

In this dissertation, we develop a framework for comparing accuracy of estimation methods in estimating F-measure and the variance. We propose some effective modeling methods for ensemble RNN model training, which apply mixture weight to navigate the ensemble model better focus on text with certain characteristics. We also provide two efficient subsampling methods for sampling more diverse and informative text data for language modeling. We summarize our main contributions in the remainder of this section.

Framework for Comparing Two Methods in Estimating F-measure and the Variance In Chapter 2, we first give some introductions of the F-measures and the related concepts, and we also provide a literature review of some estimation methods developed in recent years for esti-

mating the F-measure and its variance. We explore some estimation details in Wong's method [1], and point out improper independence assumption of algorithms and estimations in it. We describe the correct delta method, and compare Wong's estimation method with the JVESR's estimation method, which is generalized by Jiang [7]. To compare Wong's estimation method and the JVESR's estimation method experimentally, we propose a framework of comparative experiments, which is used to compare the variances computed by the underlying methods and compute the z-statistic for pairwise comparison. One important difference between JVESR's method and Wong's method is that when utilizing JVESR's method to compute the z-statistic, the correlation between two algorithms can be naturally incorporated by our proposed covariance formula. In the experiments, we demonstrate that our proposed framework is effective in comparing the accuracy of the two estimation methods, and JVESR's method is more accurate in variance estimation and z-statistic computation.

Language Models In Chapter 3, we introduce the statistical language modeling, which is important for lots of downstream tasks such as machine translation, speech recognition, and dialogue systems. We first give the definition of the probability of sequences, and the evaluation method of the modeling. This is followed by the introduction of the n-gram language models and the recurrent neural network language models, and we point out the distinctions between these two types of models. Finally, we give a brief introduction of some problems existed in training language models, and the basic ideas of the methods we propose to solve them.

Some Effective Modeling Methods for Ensemble Training In Chapter 4, we present a novel algorithm, Batch-wise Mixture Modeling for Ensembles (BaMME), which can be utilized for ensemble deep neural network training. In BaMME, we use the mixture weight and the likelihood

we get at each time step to compute the batch weight, and multiply it with the loss function during the course of the optimization. In this way, each ensemble model will be navigated to focus on text with certain characteristics in the training. The loss function for each ensemble model is minimized in the small-batch regime to avoid the degradation in the quality of the model. We also applied the smoothing technique to avoid the extreme batch weights appeared in the optimization. Our experiments conducted on the benchmark corpora demonstrate that BaMME is effective in improving the modeling results compared with the traditional ensemble modeling. We further developed two methods –Batch-wise Pre-selection of Ensembles and Clustering-based Selection of Ensembles to select the ensemble model during the training. Both of these two methods work well on the synthetic dataset, and Clustering-based selection of ensembles achieves much better results since the text data in each cluster are more semantically related with each other after clustering.

Training Data Sampling for Efficient Language Modeling In Chapter 5, we propose two new subsampling methods, CLIS and CLISIER, for selecting text set for more efficient sentence-level RNN language modeling. In each of the subsampling method, we apply the clustering technique and the importance sampling technique to make the selected training set as diverse and informative as possible. In CLIS method, we subsample within each cluster after clustering the original corpus, and the sample size in each cluster is in proportional to the square root of cluster size. We also re-weight the corrective weights applied during the training to reflect the effect of cluster size. When applying CLISIER method to sample the training data, we first train a RNN model on a randomly sampled set to get the mean log-perplexity and variability over epochs for each sentence in the set, and plot the Dataset Map to characterize and diagnose the underlying set, followed by removing a large amount of inefficiency sentences according to the plotted Dataset Map. The remaining steps are similar as those in the CLIS method, except that we only need to cluster the pre-selected set,

rather than the original corpus. Our constructed Dataset Map is different from those presented in Swayamdipta et al. [19] and Karamcheti et al. [20]. The Dataset Maps built by Swayamdipta et al. [19] and Karamcheti et al. [20] are based on the results of classification modeling and the y-axis is the average model confidence assigned to the correct answer over epochs , while our Dataset Map is built based on the results obtained from language modeling and the y-axis is the mean log-perplexity for the sentence over epochs. The experimental results demonstrate that both of these two methods are efficient in data selection for neural language models. The CLIS method performs faster at sentence selecting owe to n-gram language models' rapid training and query time. The CLISIER method produces relatively larger reductions in average perplexity compared to the baseline methods, with the cost of using more computational resources.

Conclusion and Future Work In Chapter 6, we summarize the key findings and significance of our work, and discuss some possible future works.

CHAPTER 2
COMPARISON OF TWO METHODS FOR ESTIMATING *F*-MEASURE AND ITS VARIANCE

2.1 Introduction

The F-measure [2] is very commonly used in the field of information retrieval ([21], [22]) for document classification, query classification [23], etc. It is computed as the harmonic mean of recall and precision, and is a special case of the more generic F_β-measure, which applies weights to show the relative importance of precision or recall. F-measure is generally a better evaluation measure than other metrics like accuracy, precision, recall, etc. High value of precision usually indicates a low value of recall, and vice versa. Accuracy is not an appropriate choice when evaluating the performance of algorithms tested on the imbalanced datasets.

The F-measure is usually estimated on the sample set. To evaluate the performance of a classification model tested on a sample set using F-measure, we need to compute the precision and recall by a sample average over the predicted results of instances in the sample, followed by computing the the harmonic mean of recall and precision to get the estimated F-measure for the model. The standard error and confidence interval of the estimated F-measure are also important for us since they help us better understand how closely the set of data represents the actual population and how accurate our estimations are.

Many methods have been proposed to compute the standard error of the F-measure in the past decades ([3]–[6]), but few people know about analytic formulas for the computation of the standard error. Jiang [7] introduces Janson and Vegelius (1981) [8] & Elston, Schroeder, and Rohjan (1982)

[9]'s work on statistical formulas for computing the standard error and confidence intervals of the F-measures (which will be called the JVESR formulas), based on a property of asymptotic normality in the large sample limit. Wong [1] introduces a linear approximation approach to derive the sampling distribution of F-measure, and gives estimation formulas for standard error computations on imbalanced datasets.

With the proposed methods mentioned above, we can efficiently compute the F-measure and its variance for algorithms on different sample datasets. However, there's no approach to compare the estimation accuracy of these methods. We will propose a framework for comparing JVESR's method and Wong's method on estimating the standard error of the F-measure computed on the same dataset. The proposed framework can also be used to conduct pairwise comparison of the estimated F-measures for two different algorithms.

We will introduce the related work and concepts in §2.2, and describe the framework for comparison of the estimation methods in §2.3. Empirical results and analysis are provided in §2.4. Finally, we conclude and discuss possible future works in §2.5.

2.2 Related Work and Knowledge

2.2.1 Confusion Matrix and *F*-measure

Confusion matrix has been a very useful tool to visualize the performance of machine learning algorithms, and it is a special case of contingency table in statistics. In binary classification problems, the examples in a dataset D (usually imbalanced) can be coded as $Z = 1$ and $Z = 0$ and modeled as a random variable. The prediction results of examples in the dataset can be represented as $L = I(S > s)$, where S is the output score obtained from the classification algorithm, s is the predicting threshold, and $I(\cdot)$ is an indicator function. The classification results of the underlying dataset D can be summarized in the confusion matrix as shown in Table 2.1.

Table 2.1: Confusion matrix for a binary classification model on the dataset *D*.

		Predicted class L	
		$L = 1$	$L = 0$
Actual class Z	$Z = 1$	True Positive (*TP*)	False Negative (*FN*)
	$Z = 0$	False Positive (*FP*)	True Negative (*TN*)

In Table 2.1, each row of the matrix represents the examples in an actual class while each column represents represents the examples in a predicted class. For example, the number of examples in the class $Z = 1$ is *TP*+*FN*. The classification model performance is usually evaluated by the accuracy, which is calculated as (*TP*+*TN*)/(*TP*+*FP*+*FN*+*TN*). However, accuracy is not always a good measure for assessing the performance of a model, since the model can achieve high accuracy by claiming all the data points as negative examples for imbalanced dataset. Instead of proclaiming all the data points as negative examples, we should focus on identifying the positive cases. Some important metrics for measuring the performance of models include recall, precision and specificity ([3], [24]). Recall measures the the ability of a model to retrieve all the relevant cases within a data set, while precision measures the percentage of the truly relevant examples among all the retrieved cases, and they are defined as *TP*/(*TP*+*FN*) and *TP*/(*TP*+*FP*), respectively. For evaluating the performance of classification model on the dataset *D*, the recall and precision can be represented as $rec = P(L = 1|Z = 1) = E(ZL)/EZ$ and $prec = P(Z = 1|L = 1) = E(ZL)/EL$, respectively.

In general, precision and recall are not used independently as high value of precision usually indicates a low value of recall, and vice versa. Instead, precision and recall are combined into a single measure for evaluation. One of the measures that are a combination of precision and recall is the F_β measure [2], which is the weighted harmonic mean of $prec$ and rec,

$$F_\beta = (1 + \beta^2) \cdot \frac{prec \cdot rec}{\beta^2 \cdot prec + rec} \tag{2.1}$$

where β is a constant parameter. In this Chapter, we focus on the commonly used F-measure, which is a special case of F_β with $\beta = 1$.

The F-measure for a machine learning algorithm is usually calculated based on given sample dataset, and the calculated value is influenced by the sample size [4]. Naturally, a related problem – calculating the confidence interval of F-measure would be our interest. Goutte and Gaussier [3] constructed the confidence interval for F_β based on the obtained posterior distribution. Wang et al. [5] proposed a non-symmetrical confidence interval of the F-measure based on the blocked 3×2 cross validated Beta prime distribution. And Takahashi et al. [6] estimated F-measure with confidence intervals based on the large sample multivariate central limit theorem.

Wong [1] introduced a linear approximation approach to derive the sampling distribution of F-measure, and applied a parametric statistical method to compare the performance of classification algorithms on imbalanced datasets (§2.2.2). Janson and Vegelius (1981) [8] & Elston, Schroeder, and Rohjan (1982) [9] provided explicit formulas to compute the F-measure and its variance with large sample set (§2.2.3).

2.2.2 Linear Approximation of *F*-measure

Wong [1] used a linear approximation approach to derive the sampling distribution of F-measure, and designed experiments to compare the performance of classification algorithms tested on imbalanced datasets. Specifically, by assuming that the joint distribution of recall and precision follows a bivariate normal distribution, Wong represented the estimated F-measure as a combination of

recall and precision values obtained by applying *k*-fold cross validation on the dataset. When comparing the performance of algorithms tested on single or multiple datasets, Wong assumed the **independence** of different algorithms, and estimated metric values with sample mean values in different folds and datasets for different algorithms.

Wong first introduces the estimation of *F*-measure and its variance for algorithm i on single dataset. k-fold cross validation for algorithm i is performed to collect TP_{ij}, FN_{ij} and FP_{ij}, and j is the index of folders. With these collected values, they calculate the values of recall r_{ij} and precision q_{ij} for examples in fold j, and average them to get the estimated recall $\bar{r}_i$ and precision $\bar{q}_i$ for algorithm i. This is followed by the calculation of $\bar{f}_i$ and estimated value of w_i, $\bar{w}_i$. As expressed in formula,

$$\bar{f}_i = w_i \bar{r}_i + (1 - w_i)\bar{q}_i, \tag{2.2}$$

where $\bar{r}_i = \sum_{j=1}^{k} r_{ij}/k$ and $\bar{q}_i = \sum_{j=1}^{k} q_{ij}/k$ are the estimated recall and precision for algorithm i, and w_i is the weight. Since Wong [1] assumes the independence of different algorithms, they represent the variance of the point estimator $\bar{f}_1 - \bar{f}_2$ as

$$\begin{aligned} \text{Var}(\bar{f}_1 - \bar{f}_2) &= \text{Var}(\bar{f}_1) + \text{Var}(\bar{f}_2) \\ &= \sum_{i=1}^{2} [w_i^2 \sigma_{ri}^2 + (1 - w_i)^2 \sigma_{qi}^2 + 2w_i(1 - w_i)\rho_i \sigma_{ri} \sigma_{qi}] \end{aligned} \tag{2.3}$$

where σ_{ri}, σ_{qi} and ρ_i are the standard deviation of recall, standard deviation of precision and correlation of recall and precision for algorithm i. By using sample estimations for those values, they can obtain the variance of $\bar{f}_1 - \bar{f}_2$, and compute the statistic z for testing the difference of algorithm performances.

For comparing the performance of algorithms on multiple imbalanced datasets, Wong considers

the mean F-measure for each algorithm on all the datasets. With the independence assumption of different algorithms, Wong uses the similar method mentioned above to compute the estimated mean F-measure and its variance, and then calculate the statistic z to conduct the testing.

2.2.3 Statistical Formulas for F-Measures

Janson and Vegelius (1981) [8] & Elston, Schroeder, and Rohjan (1982) [9] provided statistical formulas for calculating the standard error and confidence intervals of the F-Measures, based on a property of asymptotic normality in the large sample limit. Jiang [7] generalized the analytic approach provided by Janson and Vegelius (1981) [8] & Elston, Schroeder, and Rohjan (1982) [9] for the calculation of the standard error of the F_β -measure, and the provided formulas actually are in the form of *Tversky* index. To get the value of F_β -measure and its variance, they need to apply the provided formula on large samples to ensure it converges in distribution to a normal distribution.

Since in this Chapter we only focus on the normal F-measure (β = 1), we consider a specified version of the estimated F-measure and variance based on the JVESR approach as formulated in [7]. Assume $(Z, L), (Z_1, L_1), (Z_2, L_2), ..., (Z_n, L_n), ...$ are independent and identically distributed random variables on $\{0, 1\} \times \{0, 1\}$. The sample average of all $f(Z_k, L_k)$ is denoted as $E_n f(Z, L) \triangleq n^{-1} \sum_{k=1}^{n} f(Z_k, L_k)$. For $n = 1, 2, ..., \infty$, let

$$
\begin{aligned}
\tau_n(\lambda) &\triangleq \frac{\lambda E_n(ZL)}{\lambda E_n(ZL) + E_n(L(1-Z)) + E_n(Z(1-L))} \\
\nu_n(\lambda) &\triangleq \frac{[\tau_n(\lambda^2)]^{-1} - 1 + \{[\tau_n(\lambda)]^{-1} - 1\}^2}{E_n(ZL)[\tau_n(\lambda)]^{-4}}
\end{aligned}
\tag{2.4}
$$

Then the estimated F-measure and its variance can be represented as $\tau_n(2)$ and $\nu_n(2)$, respectively.

2.3 Methodology

In §2.2.2 and §2.2.3, we have introduced the estimation methods of *F*-measure and its variance proposed by Wong [1] and JVESR as formulated in [7]. In this section, we will explore more about the details of the estimations (§2.3.1), and design experiments to compare the estimation accuracy of the two methods (§2.3.2).

2.3.1 Further Explorations of Two Estimation Methods of *F*-measure

To approximate the *F*-measure, Wong [1] represents $\bar{f}_i$ for algorithm i as a linear combination of recall and precision shown in equation (2.2). When estimating the variance of $\bar{f}_1 - \bar{f}_2$, Wong assumes classification algorithms 1 and 2 are independent, and estimates the variance as $\sigma_1^2 + \sigma_2^2$, where σ_1^2 and σ_2^2 are the estimated variances of $\bar{f}_1$ and $\bar{f}_2$, respectively. However, when applying two algorithms on the same imbalanced dataset, it is improper to assume the independence of the two algorithms since the predicted results are all related to the dataset used. For this reason, the covariance of $\bar{f}_1$ and $\bar{f}_2$ shouldn't be ignored during the variance estimation.

The approach Wong used to estimate variance of $\bar{f}_i$ may not be totally right. As shown in equation (2.3), Wong estimates the variance of $\bar{f}_i$ as

$$\begin{aligned} \mathrm{Var}\big(\bar{f}_i\big) &= \mathrm{Var}\big(\bar{w}_i\bar{r}_i + (1-\bar{w}_i)\bar{q}_i\big) \\ &\approx \bar{w}_i^2\sigma_{ri}^2 + (1-\bar{w}_i)^2\sigma_{qi}^2 + 2\bar{w}_i(1-\bar{w}_i)\rho_i\sigma_{ri}\sigma_{qi} \end{aligned} \tag{2.5}$$

However, (2.5) doesn't hold since the weight $\bar{w}_i$ is also as random as $\bar{r}_i$ and $\bar{q}_i$, and it is estimated as $\bar{w}_i = \bar{q}_i/(\bar{r}_i + \bar{q}_i)$. The proper way to find the variance of $\bar{f}_i$ is to use delta method ([25]–[27]). By applying the first order Taylor expansion with partial derivative on both sides of equation (2.2),

we have

$$\mathrm{d}\bar{f}_i = 2[w_i^2 \mathrm{d}\bar{r}_i + (1-w_i)^2 \mathrm{d}\bar{q}_i] \tag{2.6}$$

where w_i is the large sample limit of $\bar{w}_i$. Then, by delta method, it gives

$$\mathrm{Var}\big(\bar{f}_i\big) = 4\mathrm{Var}\big(w_i^2\bar{r}_i + (1-w_i)^2\bar{q}_i\big) \tag{2.7}$$

It is noteworthy that the weight w_i in (2.6) and (2.7) is non-random, while in (2.5), $\bar{w}_i$ actually is as random as $\bar{r}_i$ and $\bar{q}_i$.[1]

The correct application of the delta method will lead to the estimated variance for *F*-measure as shown in equation (2.4).

2.3.2 Design of Comparative Experiments

In §2.3.1, we have shown that the approach Wong [1] used to compute the variance of the *F*-measure is not accurate due to his improper assumption and estimations. We further design experiments to compare the accuracy of the estimation methods proposed by Wong [1] and Jiang [7]. Specifically, we will perform two different classification algorithms on the given imbalanced dataset and conduct pairwise comparison. And then employ the two underlying methods to calculate $\bar{f}_1 - \bar{f}_2$ and its variance, as well as the z-statistic to test $H_0 : F_1 - F_2 = 0$, where F_1 and F_2 are the true values of the *F*-measure. The computed results will be used to compare with the true statistics in order to evaluate the performances of the estimation methods. A high-level of the framework of our experiments is shown in Figure 2.1. We will give more detailed descriptions of

[1] In some special cases, such as when $\bar{r}_i \approx \bar{q}_i$ (so that $\bar{w}_i \approx 0.5$), (2.5) could provide approximately the same results as (2.7).

the comparative experiments in the remainder of this section.

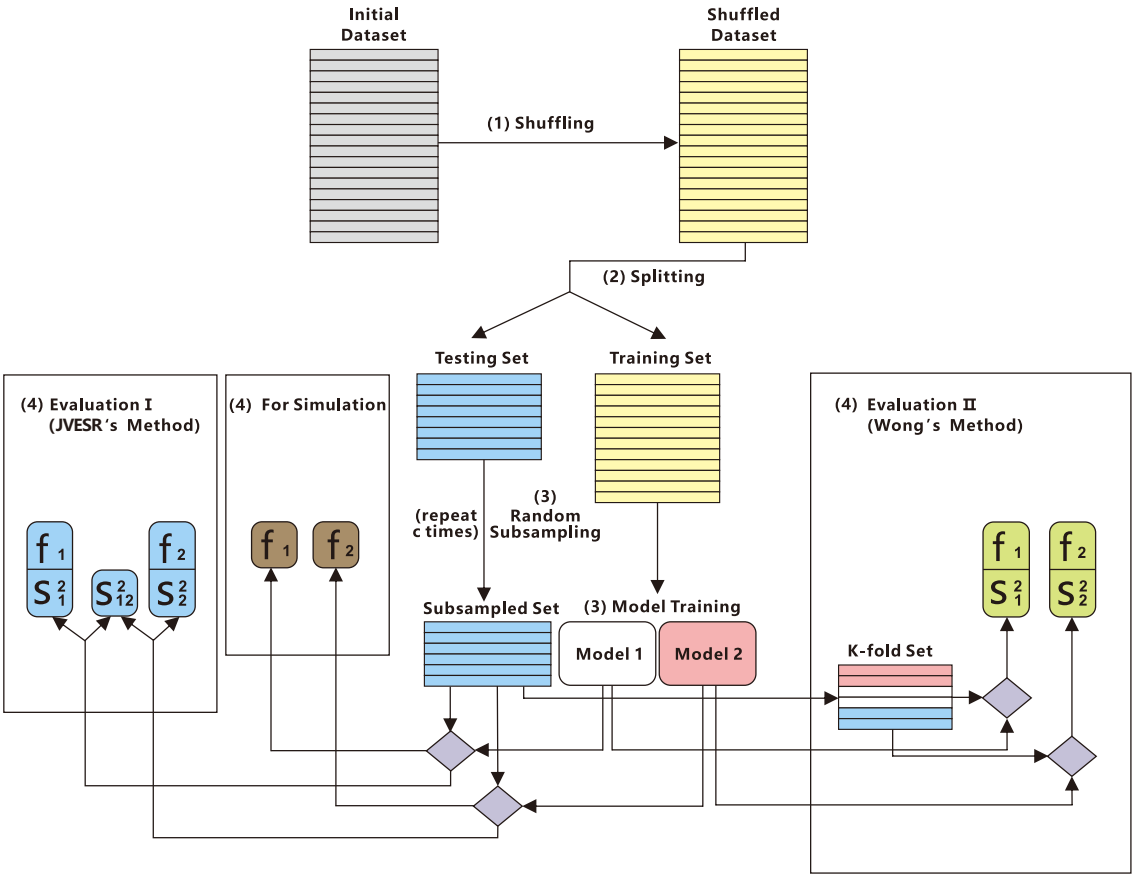

Figure 2.1: Illustration of the Comparative Experiments. We start with (1) shuffling the initial dataset, and then (2) split the shuffled dataset into training set and testing set, followed by (3) model training and random subsampling the testing set for (4) evaluation and simulation processes. We repeat the subsampling procedure and procedure (4) $\boldsymbol{c}$ times, and more details of the computations in (4) are described in §2.3.2. In the Figure, s_i^2 represents the estimated sample variance $\mathrm{Var}(\bar{f}_i)$, and s_{12}^2 represents the sample covariance $\mathrm{Cov}(\bar{f}_1, \bar{f}_2)$.

We first shuffle the original given imbalanced dataset, and split it into training and testing sets. We train two different classification models on the splitted training set, and the model details are given in §2.4.1. The trained models are performed on a set of size $\boldsymbol{n_0}$ (subsampled with replacement

from the testing set) to get the predicted results, and this is followed by calculating $\bar{f}_1 - \bar{f}_2$ and its variance using the two estimation methods. The evaluation process will be repeated $\boldsymbol{c}$ times, and we will give more descriptions of this process. The reason why we evaluate on the subsampled set is we have a simulation process introduced later, and we expect to use same data for the evaluation process and simulation process.

To apply the JVESR method to calculate $\bar{f}_1 - \bar{f}_2$ and its variance, we first perform the trained model on the subsampled set with size $\boldsymbol{n_0}$ to get predicted results. Formulas in (2.4) are used to compute the estimated *F*-measure $\bar{f}_i$ and its variance σ_i^2. The variance of $\bar{f}_1 - \bar{f}_2$ is calculated as

$$\text{Var}(\bar{f}_1 - \bar{f}_2) = \text{Var}(\bar{f}_1) + \text{Var}(\bar{f}_2) - 2\text{Cov}(\bar{f}_1, \bar{f}_2) \tag{2.8}$$

where the $\text{Cov}(\bar{f}_1, \bar{f}_2)$ can be calculated using formula (A.5) in Appendix A. With the calculated $\bar{f}_1 - \bar{f}_2$ and $\text{Var}(\bar{f}_1 - \bar{f}_2)$, the z-statistic is computed as

$$z = \frac{\bar{f}_1 - \bar{f}_2}{\sqrt{\text{Var}(\bar{f}_1 - \bar{f}_2)}} \tag{2.9}$$

After repeating this evaluation process $\boldsymbol{c}$ times, we use the averaged z-statistic to test $H_0 : F_1 - F_2 = 0$.

When utilizing Wong's method to do the calculations, we need to get the predicted results of the examples in the subsampled set (with size $\boldsymbol{n_0}$) by performing the trained model on the set. To apply k-fold cross validation for algorithms 1 and 2, we randomly and independently partition the subsampled set **twice**, each time into k disjoint folds with same number of instances in each fold when applying formula (2.10) to compute the correlation of recall and precision. We follow the detailed process described in §2.2.2 to compute the estimated *F*-measure $\bar{f}_i$, $\bar{f}_1 - \bar{f}_2$ and its

variance Var$(\bar{f}_1 - \bar{f}_2)$, with the correlation ρ_i in (2.3) estimated as

$$\bar{\rho}_i = \frac{\sum_{j=1}^{k}(r_{ij} - \bar{r}_i)(q_{ij} - \bar{q}_i)}{\sqrt{\sum_{j=1}^{k}(r_{ij} - \bar{r}_i)^2}\sqrt{\sum_{j=1}^{k}(q_{ij} - \bar{q}_i)^2}} \tag{2.10}$$

where $\bar{r}_i = \sum_{j=1}^{k} r_{ij}/k$, $\bar{q}_i = \sum_{j=1}^{k} q_{ij}/k$, and k is the number of folds used. In (2.10), r_{ij} and q_{ij} are the recall and precision for algorithm i on the data in fold j , respectively. We compute the z-statistic using formula (2.9) after getting $\bar{f}_1 - \bar{f}_2$ and its variance. Still, we repeat the process $\boldsymbol{c}$ times and compute the average of the used z-statistics to conduct the pairwise tests.

We have a simulation process to find the simulated variances of F_i and $F_1 - F_2$ [5]. By treating the testing set as a population, we draw a sample of size $\boldsymbol{n_0}$ from the testing set with replacement, and perform the trained models on the sampled set to get predicted results, followed by computing $\bar{f}_1 - \bar{f}_2$ with the predicted results. We repeat this process $\boldsymbol{c}$ times, and compute the variance of $\bar{f}_1 - \bar{f}_2$ from the distribution of these $\boldsymbol{c}$ sample values of $F_1 - F_2$. In the experiments, the $\boldsymbol{c}$ subsampled sets are the same sets we used in the evaluation process mentioned above.

2.4 Experiments

2.4.1 Experimental Settings and Datasets

We choose 3 datasets (Abalone [28], White-wine [29] and Seismic [30]) from the UCI data repository [31] for evaluating the model performance, and some information of the datasets are summarized in Table 2.2. In Wong's experiments [1], to create imbalanced datasets, some of the class values in each dataset are chosen to be the positive one. For better comparison of JVESR's method and Wong's method in estimating F-measure and its variance, we choose the same class values in the dataset as the positive one, as shown in Table 2.2.

The two classification algorithms we choose to perform on the datasets are k-nearest neighbors with $k = 1$ and the Random Forest algorithm, and we abbreviate them to 1-NN and RF, respectively. Algorithm settings are set as default in R except that the number of trees for RF is set to 100. The numbers of instances we use for training the models are listed in Table 2.2. The subsampling size $\boldsymbol{n_0}$ and the number of times ($\boldsymbol{c}$) we repeat the subsampling and calculations during evaluation and simulation processes are shown in Table 2.2 as well. When using Wong's method to compute the F-measure and the variance, the number of folds is set to be five. Our code is publicly available.[2]

Table 2.2: Information of the three imbalanced datasets and some parameters for subsampling procedure.

Dataset	**Total Size**	**Training Size**	**Attributes**	**Positive Class**	$\boldsymbol{n_0}$	$\boldsymbol{c}$	$(\boldsymbol{c_0}, \boldsymbol{c_1}, \boldsymbol{c_2})$
Abalone	4177	2924	8	6			(63, 8, 60)
White-wine	4898	3429	11	3,4,8,9	1000	1200	(27, 0, 27)
Seismic	2584	1250	18	1			(291, 247, 291)

The number of values we use to calculate the averaged estimated F-measure and the variance is not exactly divided by $\boldsymbol{c}$ since we disregard those evaluations where we get infinite estimated variance due to zero TP or infinite sample correlation computed from (2.10). In Table 2.2, $\boldsymbol{c_0}$, $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ represent the total number of evaluations we disregard, the number of evaluations we disregard due to infinite variance obtained with JVESR's method and the number of evaluations we disregard because of infinite variance computed with Wong's method, respectively. For datasets Abalone and White-wine, most of the disregarded evaluations are caused by the appearance of infinite variance computed using Wong's method. And for dataset Seismic, we still have extra evaluations which are abandoned solely due to Wong's method.

[2]https://github.com/teniscape/comparison-of-methods-for-estimating-F-measures

2.4.2 Results and Analysis

Our main comparison results are reported in Table 2.3. For all the datasets, the values of Var($\bar{f}_1 - \bar{f}_2$) and z-statistic computed with JVESR's approach are closer to the simulated variances and z-statistic compared with those values obtained with Wong's method. This is mainly because we could naturally modify JVESR's approach to take the convariance of $\bar{f}_1$ and $\bar{f}_2$ into consideration when computing the variance of $\bar{f}_1 - \bar{f}_2$ and also the method is more accurate in estimating the variance of $\bar{f}_i$ in general, while Wong's method ignores the dependency of algorithms tested on the same imbalanced dataset (§2.3.1).

Table 2.3: Results of comparing JVESR's estimation method against Wong's estimation method on the three imbalanced datasets. JVESR's method achieves more accurate estimation results (closer to the simulated results) compared with Wong's method on all the three datasets.

Dataset	**Metric**	**Simulation**		**JVESR's Method** [7]		**Wong's Method** [1]	
(Algorithm)		1-NN	RF	1-NN	RF	1-NN	RF
Abalone	$\bar{f}_{i}$[ℵ]	0.209	0.138	0.209	0.138	0.209	0.138
	Var($\bar{f}_i$)	0.00241	0.00278	0.00237	0.00267	0.00230	0.00235
	Corr($\bar{f}_1, \bar{f}_2$)	0.347		0.348		0	
	Var($\bar{f}_1 - \bar{f}_2$)	0.00339		0.00326		0.00465	
	z statistic	1.22		1.27		1.09	
White-wine	$\bar{f}_i$	0.347	0.445	0.347	0.445	0.347	0.445
	Var($\bar{f}_i$)	0.00276	0.00373	0.00287	0.00393	0.00273	0.00264
	Corr($\bar{f}_1, \bar{f}_2$)	0.744		0.746		0	
	Var($\bar{f}_1 - \bar{f}_2$)	0.00171		0.00180		0.00537	
	z statistic	-2.38		-2.42		-1.38	
Seismic	$\bar{f}_i$	0.145	0.052	0.145	0.052	0.145	0.052
	Var($\bar{f}_i$)	0.00164	0.00070	0.00176	0.00127	0.00173	0.00120
	Corr($\bar{f}_1, \bar{f}_2$)	0.342		0.412		0	
	Var($\bar{f}_1 - \bar{f}_2$)	0.00161		0.00179		0.00293	
	z statistic	2.33		2.16		1.75	

[ℵ] $i = 1$ if the algorithm is 1-NN. Otherwise $i = 2$.

When computing the Corr($\bar{f}_1, \bar{f}_2$) using JVESR's method, it is the averaged value of all the

$\boldsymbol{c} - \boldsymbol{c_0}$ correlations between $\bar{f}_1$ and $\bar{f}_2$ of the algorithms tested on the subsamples, each being estimated from the covariance formula proposed in Appendix A. The simulated Corr($\bar{f}_1, \bar{f}_2$) is computed as the exact correlation between the $\boldsymbol{c} - \boldsymbol{c_0}$ $\bar{f}_1$'s and $\bar{f}_2$'s of the algorithms tested on the subsamples. In Table 2.3, the Corr($\bar{f}_1, \bar{f}_2$) computed by JVESR's method is pretty close to the simulated correlation between $\bar{f}_1$ and $\bar{f}_2$ for all the three datasets, and this further verifies that the dependency of algorithms shouldn't be ignored when comparing the algorithms tested on the same dataset.

As shown in Table 2.3, we present the z-statistic values of pairwise comparison for different methods. It is possible for us to make quite opposite decision for pairwise comparison if the estimation of Var($\bar{f}_1 - \bar{f}_2$) is inaccurate. For example, for the White-wine and Seismic sets, the simulated z-statistic and and the z-statistic computed by JVESR's method both indicate that typically we have to reject the null hypothesis at the significance level 0.05 for comparing the two algorithms on both datasets. However, with the same significance level, the z-statistics computed with Wong's method indicate that we can typically accept the null hypothesis.

The ways JVESR and Wong used to compute the F-measure are essentially the same, although Wong first calculates the recall and precision for data in each fold and then average them to calculate the final F-measure for each algorithm. As a result, the F-measures calculated for the same algorithm on the same dataset shown in Table 2.3 are the same. Although for most of the cases, the variances of F-measure computed using Wong's method are not accurate, we do find that they are close to the simulated variances in some cases. For example, the Var($\bar{f}_1$) for White-wine set. Although we have pointed out in §2.3.1 that the method Wong used to compute the variance of $\bar{f}_i$ is not totally right, sometimes the values computed with formulas (2.5) and (2.7) can be pretty close.[1]

When testing the RF algorithm on Seismic dataset, we find that the number of *TP* examples

for most of the subsampled sets is less than 5, and thus the large-sample conditions is not satisfied for those cases. The precise asymptotics in the estimation of the variance in JVESR's method is probably impaired in this situation, and this makes the estimated Var($\bar{f}_2$) not so accurate as the estimations for the other two sets. The estimated variance obtained with Wong's method in this case is also larger than the simulated variance. Even the estimated variances for RF algorithm on Seismic set are not so accurate for both of the methods, we still get more accurate z statistics for pairwise comparison when applying JVESR's method since the correlation between the two algorithms can be naturally incorporated by our covariance formula (A.5) in Appendix A.

We notice that the $\bar{f}_2$ for Seismic set is close to boundary 0, and this may be another possible reason (*F*-measure being close to values 0 or 1) why it doesn't lead to a good asymptotic result when using JVESR's method to estimate the Var($\bar{f}_2$) on Seismic set . We leave for future work the more thorough understanding of this interesting phenomenon that possibly caused by small number of *TP* examples and extreme *F*-measure, and the finding of possible methods to fix it.

2.5 Conclusion

In this chapter, we compare two methods for computing the *F*-measure and its variance. We point out the improper assumption and inaccurate estimation in Wong's method [1], and describe the correct delta method. We also propose a framework of comparative experiments, which is used to conduct experiments on comparing the accuracy of JVESR's method [7] and Wong's method [1] in computing the variance of *F*-measure and the z-statistic for pairwise comparison. Our experiments demonstrate that the proposed framework is effective in comparing the accuracy of the two estimation methods, and JVESR's method is more accurate in variance estimation and z-statistic computation when we incorporate a covariance estimate naturally based on Appendix A.

Our proposed framework of comparative experiments can be extended to compare multiple

algorithms on multiple datasets in the future. Moreover, for the statistical method (Appendix A) for estimating the covariance of two F-measures computed with JVESR's method, it can be generalized to compute the covariance matrix of F-measures for multiple algorithms (private communication with W. Jiang, 2021).

When computing the F-measures of two different algorithms and the variances in Wong's experiments [1], the imbalanced dataset is "randomly and independently divided into k and l disjoint folds for classification algorithms 1 and 2, respectively". However, this description is ambiguous and it probably will not lead to two independent sets by partitioning in this way since we may have same instances falling in the ith fold both times for the two algorithms. To truly achieve the independence of the two algorithms, we may have to use two non-overlapping halves to compare the two algorithms in the future.

Finally, we comment that by comparison of two algorithms, we really mean comparison of the testing data performances of the two rules learned from the same training data, as generated by the two different classification methods. In standard error computations of both JVESR and Wong, variance of testing data is incorporated but not the variation of the classification rules learning from possibly different training data. The latter is more difficult, since its variance does not relate to standard estimation of a low dimensional parameter from training data, such as in Logistic Regression, but is involving many parameters possibly incompletely optimized (such as in Random Forest) or a completely nonparametric method (such as in Nearest Neighbor).

CHAPTER 3
LANGUAGE MODELS

Computing the probability of the text is important in many Natural Language Processing (NLP) tasks such as machine translation, text summarization, document categorization and dialogue systems. The direct or indirect goal of these tasks is to produce proper word sequences as output, and pre-training better language models will benefit for the comparison of task outputs.

Statistical language modeling estimates the probability distributions of text, and it helps capture the structures of language and extract useful information contained in various corpora. Specifically, we will consider models that assign probability to a sequence of word tokens, $p(w_1, w_2, \ldots, w_m)$, with $w_i \in \mathcal{W}$. The set $\mathcal{W}$ is known as the vocabulary set, and it is consisted of discrete vocabulary tokens. Alternatively, if we have a large corpus which is consisted of sequences $s_1, s_2, \ldots, s_n$, we can estimate joint probability distribution of the sequences: $p(s_1, s_2, \ldots, s_n)$.

The probability of a sequence $p(w_1, w_2, \ldots, w_m)$ can be factored as:

$$p(w_1, w_2, \ldots, w_m) = p(w_1) \times p(w_2|w_1) \times p(w_3|w_1, w_2) \times \ldots \times p(w_m|w_1, \ldots, w_{m-1}) \tag{3.1}$$

Or, a simpler way to represent (3.1) could be

$$p(\mathbf{w}) = \prod_{i=1}^{m} p(w_i|\mathbf{w}_{\text{sub}}^{i-1}) \tag{3.2}$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ represent a sequence consisted of m word tokens, and $\mathbf{w}_{\text{sub}}^{i-1} = (w_1, w_2, \ldots, w_{i-1})$ denotes a subsequence of $\mathbf{w}$.

To evaluate the performance of language models, instead of computing the likelihood directly using the trained models, we usually compute the **perplexity** of testing data set. It is the root inverse of the joint probability of all the sequences in the set, and the joint probability can be refactored as the product of probabilities of word tokens. The lower the perplexity, the better performance of models. In calculation, we use the exponential transformations to replace the multiplications by additions to avoid the underflow caused by vanishing of products in computer systems. The formula to compute the perplexity of the set $\mathcal{S}$ is shown as below:

$$\begin{aligned} \text{Perplexity}(\mathcal{S}) &= \sqrt[|\mathcal{S}|]{\frac{1}{p(w_1, \ldots, w_{|\mathcal{S}|})}} \\ &= \sqrt[|\mathcal{S}|]{\prod_{i=1}^{|\mathcal{S}|} \frac{1}{p(w_i|\mathbf{w}_{\text{sub}}^{i-1})}} \\ &= e^{-\frac{1}{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{S}|} \log p(w_i|\mathbf{w}_{\text{sub}}^{i-1})} \end{aligned} \tag{3.3}$$

where w_i represent the ith token in the corpus $\mathcal{S}$.

Now, the main problem we focus here is how to compute the conditional distribution of word w_t given all its predecessors. There are mainly two classes of language models to solve this problem: *N*-gram language models and neural network language models. We will discuss these two classes of language models and their applications in the remaining part of this chapter.

3.1 *N*-gram Language Models

An *n*-gram model is a type of probabilistic language model, and it makes an important approximation: the condition probability of token w_i only depends on the previous *n*-1 tokens.

$$p(w_i|\mathbf{w}_{\text{sub}}^{i-1}) = p(w_i|w_1, \ldots, w_{i-1}) \approx p(w_i|w_{i-n+1}, \ldots, w_{i-1}) \tag{3.4}$$

The *n*-gram probabilities can be computed by the following relative frequency counting,

$$p(w_i|w_{i-n+1},\ldots,w_{i-1}) = \frac{\text{number of } \{(w_{i-n+1},\ldots,w_{i-1},w_i)\}}{\sum_{w\in\mathcal{W}} \text{number of } \{(w_{i-n+1},\ldots,w_{i-1},w)\}} \tag{3.5}$$

where $\mathcal{W}$ is the vocabulary set.

Then, the probability of a sequence **w** can be approximated as

$$p(\mathbf{w}) = \prod_{i=1}^{m} p(w_i|w_1,\ldots,w_{i-1}) \approx \prod_{i=1}^{m} p(w_i|w_{i-n+1},\ldots,w_{i-1}) \tag{3.6}$$

The beginning and end sequences are usually padded with special or empty symbols.

Since the text data is usually sparse, a lot of *n*-grams have few or even zero occurrence, and this could lead to the case where $p(\mathbf{w}) = 0$. In *n*-gram models, it is not uncommon to use techniques like smoothing to solve this problem. We can apply smoothing techniques to get the estimations for those grams. For example, we can use Lidstone smoothing ([32], [33]) in language modeling.

There are many well-known smoothing methods proposed in the past research work. For example, Kneser-Ney smoothing [34], Good-Turing discounting [35] and Katz's back-off model [36], etc. The hyperparameter *n* controls the size of the past context used in conditional probabilities. If it is misspecified, the language model will probably perform poorly. Also, as *n* increases, the number of model parameter increases exponentially. For this reason, the value of *n* is usually chosen to be less than or equal to 5 in practice. The n-gram model models the text versatility by only simple counting the frequency of contexts, and the low value of hyperparameter *n* seems can't learn long term dependency of sequence well. Even with these facts, n-gram models combined with proper smoothing methods were widely used in the language modeling before the introduction of neural language models.

3.2 Recurrent Neural Network Language Models

Neural Network Language Models have been very popular in the past decade and they have become the default models in many NLP tasks, although *n*-gram models are still used in some scenarios due to its less computational cost and simple architecture. Neural networks don't take the independent assumption assumed in *n*-gram models. Instead, they can use internal states to store information and capture patterns of long text sequences during training ([37]–[39]).

In neural language models, the goal is to compute the conditional probability $p(w|u)$, where w is a word token, and u is the given context. Instead of estimating the word probabilities from relative frequencies directly, we can train the language models on the corpus, and find parameters that can maximize the log conditional probability of the corpus.

The most well-known neural network used for language modeling is the recurrent neural network (RNN) [38]. The idea behind the RNN language models is to recurrently update the context vectors while moving through the sequence. At time step m, we first get the embedding vector of the word w_m,

$$\boldsymbol{e}_m = \boldsymbol{E}_{w_m} \tag{3.7}$$

where $\boldsymbol{E}$ is the embedding matrix of input words, and $\boldsymbol{e}_m$ denotes the word embedding of w_m. Word embeddings are actually real-valued vectors, and they are initialized randomly and get updated during language model training.

With the obtained word embedding $\boldsymbol{e}_m$, we update the hidden representation from $\boldsymbol{h}_{m-1}$ to $\boldsymbol{h}_m$,

$$\boldsymbol{h}_m = f(\boldsymbol{h}_{m-1}, \boldsymbol{e}_m) \tag{3.8}$$

where $f(\cdot)$ is the recurrent function. For example, a simple recurrent function [40] is

$$f(\boldsymbol{h}_{m-1}, \boldsymbol{x}_m) = \sigma(\mathbf{U}_h \boldsymbol{h}_{m-1} + \mathbf{W}_x \boldsymbol{x}_m + \boldsymbol{b}) \tag{3.9}$$

where the input $\boldsymbol{x}_m$ in language modeling is usually the embedding of word defined as $\boldsymbol{e}_m$ in (3.7), and $\mathbf{U}_h$ and $\mathbf{W}_x$ are recurrence matrices. $\boldsymbol{b}$ is the offset vector and σ is the activation function, often defined as the hyperbolic tangent $\tanh$.

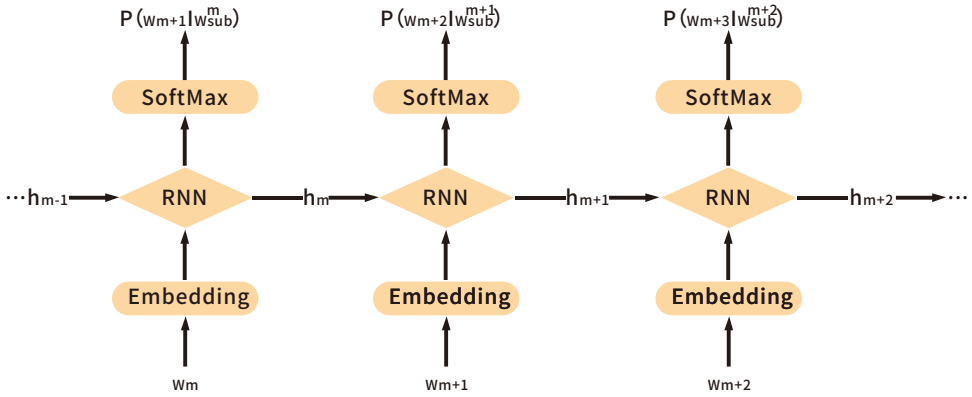

Figure 3.1: A single layer representation of RNN language model. The input of the RNN operation is composed of the hidden representation $\boldsymbol{h}_m$ and the embedding representation of token w_{m+1}.

After updating the hidden representation, we compute the probability distribution of the next word w_{m+1},

$$p(w_{m+1}|\mathbf{w}_{\text{sub}}^m) = \frac{\exp(\tau_{w_{m+1}} \cdot \boldsymbol{h}_m)}{\sum_{w \in \mathcal{W}} \exp(\tau_w \cdot \boldsymbol{h}_m)} \tag{3.10}$$

where $\tau_w' s$ are the parameters of the model. In (3.10), $\tau_{w_{m+1}} \cdot \boldsymbol{h}_m$ represents a dot product of the parameter vector $\tau_{w_{m+1}}$ and the hidden representation $\boldsymbol{h}_m$. The denominator in (3.10) ensures that the probability distribution is normalized.

Figure 3.1 demonstrates the architecture of a single layer RNN language model. RNN models can capture long-range dependencies during training since the conditional probability of w_{m+1} depends on the hidden representation $\boldsymbol{h}_m$, which is influenced by all previous tokens through the recurrence operation f. This is one of the most important distinctions from *n*-gram models. There is another important difference between the RNN models and the *n*-gram models: the RNN models incorporate the hidden embedding space that allows the model to use the distributional hypothesis– similar words tend to appear in similar contexts.

Recurrent neural network can propagate information in the long-distance training. But repeatedly using squashing function results in the depletion of the information. Similarly, applying recurrence operation over and over during back-propagation will cause the vanishing or exploding gradients problem. Variations of RNN have been proposed to alleviate this problem, and we willl introduce two types of variations of RNN.

The long short-term memory (LSTM) ([39], [41]) is a variation of RNNs, and it can efficiently avoid vanishing or exploding gradients issue in back-propagation. In LSTM, the hidden state $\boldsymbol{h}_m$ is augmented with a memory cell $\boldsymbol{c}_m$, which is a gated sum of its previous value $\boldsymbol{c}_{m-1}$ and the input activation $\boldsymbol{g}_m$. The update equations of LSTM are shown as follows:

$$
\begin{aligned}
\boldsymbol{f}_m &= \sigma(\mathbf{U}_f \boldsymbol{h}_{m-1} + \mathbf{W}_f \boldsymbol{x}_m + \boldsymbol{b}_f) \\
\boldsymbol{i}_m &= \sigma(\mathbf{U}_i \boldsymbol{h}_{m-1} + \mathbf{W}_i \boldsymbol{x}_m + \boldsymbol{b}_i) \\
\boldsymbol{g}_m &= \tanh(\mathbf{U}_g \boldsymbol{h}_{m-1} + \mathbf{W}_g \boldsymbol{x}_m + \boldsymbol{b}_g) \\
\boldsymbol{c}_m &= \boldsymbol{f}_m \circ \boldsymbol{c}_{m-1} + \boldsymbol{i}_m \circ \boldsymbol{g}_m \\
\boldsymbol{o}_m &= \sigma(\mathbf{U}_o \boldsymbol{h}_{m-1} + \mathbf{W}_o \boldsymbol{x}_m + \boldsymbol{b}_o) \\
\boldsymbol{h}_m &= \boldsymbol{o}_m \circ \tanh(\boldsymbol{c}_m)
\end{aligned}
\tag{3.11}
$$

where $\boldsymbol{x}_m$ is the input vector, $\boldsymbol{h}_m$ is the hidden representation, $\mathbf{U}$ and $\mathbf{W}$ are recurrence matrices, σ is the sigmoid function, $\circ$ denotes the Hadamard product, and $\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{o}$ represent the outputs of forget gate, input gate and output gate, respectively. The overall update operation can be represented as

$$(\boldsymbol{h}_m, \boldsymbol{c}_m) = f((\boldsymbol{h}_{m-1}, \boldsymbol{c}_{m-1}), \boldsymbol{x}_m) \tag{3.12}$$

When utilizing LSTMs in language modeling, equation (3.8) should be replaced with

$$(\boldsymbol{h}_m, \boldsymbol{c}_m) = f((\boldsymbol{h}_{m-1}, \boldsymbol{c}_{m-1}), \boldsymbol{e}_m) \tag{3.13}$$

Gated recurrent unit (GRU) was introduced by Cho et al. [42] in 2014, and it is also a variation of RNNs. Like LSTMs, GRUs use regulatory gates to control information flow, but they have fewer parameters than LSTMs as they don't have output gate. The performance of GRUs on certain NLP tasks is similar to that of LSTMs, and GRUs have been shown to have even better performance on smaller datasets [43].

When estimating the model parameters, gradient based methods like Stochastic gradient descent (SGD) [44] or Adam [45] are usually used for the negative log-likelihood minimization. And the gradient can be calculated by back-propagation algorithm [46] during the optimization procedure.

3.3 Effective and Efficient Language Modeling

In the previous sections, we have given a brief introduction of the importance of language modeling, its evaluation method and the related applications. We also introduced the *n*-gram language models, the recurrent neural network language models and their two types of variations: LSTM and GRU. How to pre-train RNN language models effectively and efficiently given a corpus has

always been important topics in Natural Language Processing (NLP) as the pre-trained language models will affect the performance of downstream tasks directly.

In language model pre-training, we usually have pretty huge corpus that we probably can't train one single model well on it. One effective way to solve this problem is to utilize ensemble modeling techniques [47] since ensemble model combines several individual models to obtain better generalization performance. In Chapter 4, we will propose an algorithm called Batch-wise Mixture Modeling for Ensembles (BaMME), which applies ensemble modeling techniques with mixture weights to pre-train language models on the large corpus. Our experimental results will demonstrate the effectiveness of our newly-developed algorithm. We will further create some synthetic datasets, and develop Batch-wise Pre-selection of Ensembles and Clustering-based Selection of Ensembles methods to investigate how to better navigate ensemble model to focus on text with certain characteristics.

To achieve state-of-art results on some benchmark corpora or real text corpora requires training very large and sophisticated models, which can be difficult to train due to memory constraints and prohibitive computational costs. Even using RNN models with relatively small capacities to do the training, the time costs could still be an issue as the sizes of the corpora such as Wikitext-103 [48] and the Billion Word Corpus [49] are usually pretty large. The RNN models for these large corpora can be learned by training on a set of sentences subsampled from the original corpus. In Chapter 5, we will propose two subsampling methods which apply the clustering technique and the importance sampling technique to make the selected training set as diverse and informative as possible. These two methods make the model training more efficient, and the experimental results show that our methods yield lower perplexities than the baseline results. The first sampling method performs faster owe to n-gram language models' rapid training and query time. In the second sampling method, we employ the dataset cartography technique [19] to identify those inefficiency

training examples [50], and remove varying numbers of those examples during the pre-selection process. Although the second method uses more time and computational resources compared with the first sampling method and the importance sampling method [18], it helps remove the *hard-to-learn* sentences and yields relatively better evaluation results.

CHAPTER 4
SOME EFFECTIVE MODELING METHODS FOR ENSEMBLE TRAINING

4.1 Introduction

In Natural Language Processing (NLP) and Natural Language Generation (NLG), Language Modeling (LM) is an important task as modeling the distributions of text accurately will help capture the structures of language and extract useful information contained in various corpora. In many NLP tasks such as machine translation, speech recognition, or dialogue systems, the goal is to produce word sequences as output. Pretraining better language models will benefit for the comparison of task outputs.

Large pretrained language models such as BERT [51] and RoBERTa [52] are the foundation of NLP today. But we usually have more text than we can possible train one single model on well. Ensemble modeling techniques in machine learning combine the results from several models to improve the overall performance, and they have been employed in a number of NLP and NLG tasks to achieve better results. Ensemble of Recurrent neural network models of language (RNNLMs) combine the predictions of multiple RNNLMs to reduce the predicted perplexity of corpus.

However, training standard ensemble of RNNLMs over the whole corpus may ignore the inherent characteristics of text and the specialization of ensemble element on certain subset of text. And simply averaging the predictions of the ensemble models may result in incoherent utterances in NLG [47]. A natural idea is to investigate partitioning strategies to learn the best way to specialize our models.

Inspired by the ideas in domain-adaptive and task-adaptive pretraining [53] for language mod-

els, we try to investigate whether having different ensemble elements focus on soft-clustered subsets of the text with different characteristics (varies of sentence lengths, differences of types of words, etc.) will augment the training and reduce the predicted perplexity. Specifically, we propose a new algorithm, Batch-wise Mixture Modeling for Ensembles (BaMME), which can be utilized for ensemble deep neural network training. In BaMME, we use the mixture weight and the likelihood we get at each time step to compute the batch weight, and multiply it with the loss function during the course of the optimization. When combining the predictions of the models, we compute the sum of weighted likelihoods.

We apply our newly-created algorithm to train ensemble RNNLMs, and we show that our method outperforms the standard method used for training ensemble RNNLMs. Training big pre-trained models is expensive, so we work in a simulated setting with small RNNLMs and datasets like Wikitext-2 (WT2) [48] and Penn Tree Bank (PTB) [33], which simulates the case where the models are too small to learn the entire corpus and we can instead use an ensemble to get better performance.

4.2 Related Work and Knowledge

4.2.1 Ensemble of Recurrent Neural Networks

As introduced in §3.2, RNNs have shown significant promise for sequential text training as they use their internal state to store information and capture patterns of text sequences during modeling.

Ensemble of RNNs further reduce the generalization error of the prediction. Juraska et al. [47] utilize three neural models with different encoders and combine the predictions of models to obtain higher BLEU scores of utterances in NLG. A class-specific weighted voting mechanism was introduced by Aniol et al. [54], who propose an ensemble model based on three selected architecture for SQuAD Task [55]. Webb et al. [56] introduce a new family of loss functions to

study the properties of joint training for neural network ensembles, and their method show the promise for joint training in resource limited scenarios by using appropriate regularization.

Our method has two distinctions from traditional Ensembles of RNNs. First, we guide ensemble element to specialize on a subset of the batches, by multiplying the loss function of each batch with smoothed weight. Second, loss function for each ensemble element is minimized in the small-batch regime to avoid the degradation in the quality of the model [57].

4.2.2 Importance Sampling

Importance sampling techniques have raised increased attention these years since (adaptive) sampling distributions can capture characteristics of data points that contribute to the improvement of performance and they offers improved convergence through variance reduction [58]. Adaptive strategies are adopted to update the parameters of distributions during the optimization process ([59], [60]). Adaptive mixture weights were presented by Borsos [61], who propose a framework for variance reduction where sampling distributions are combined into a mixture and loss function is multiplied with the importance weight during the course of the optimization.

In our project, we also use mixture weights, and they are adapted during the training process. Instead of multiplying importance weight, we multiply the loss with the computed batch weight during the optimization. The method to choose mixture weights is also different. Borsos et al. [61] use a two-stage projection procedure to find mixture weighs, while we update the mixture weights by setting them to be equal to the smoothed batch weights in the last epoch (see more details in Algorithm 1 and Appendix B).

4.2.3 Expectation–maximization (EM) Algorithm

EM algorithm [62] has been shown to be a promising iterative method to find maximum likelihood estimates (MLE) of parameters in many statistical models, where the model depends on unobserved latent variables and it's difficult to explicitly find the MLE of the parameters.

In the traditional EM algorithm, we first define the conditional expectation of the log likelihood function of the parameter given the data and the current estimates in E-step, and then we maximize it in M-step. Usually, we will find the lower bound of the expectation of the log likelihood in the E-step, and maximize the lower bound instead. The lower bound is often depending on the distribution of the latent variable.

The differences of our method and the EM method lie in several aspects. First, we are minimizing the upper bound of the negative batch loss (see Appendix B). Second, the reason we multiply the batch loss with the batch weight is not simply to formulate the upper bound. Instead, we are trying to guide each ensemble element to specialize on a subset of the batches. Also, we smooth the batch weights before multiplying them with the batch losses.

4.3 Methodology

In this project, our goal is to train ensemble RNNLMs on the given datasets and minimize the loss function. We are also trying to investigate how the different elements of the ensemble specialize on certain text domains or sentence structures. Typically, we minimize the average negative log-likelihood of training corpus ,

$$\min_{\theta \in \mathbf{\Theta}} \mathcal{L}(\theta) = \min_{\theta \in \mathbf{\Theta}} \frac{1}{|\mathcal{C}|} \sum_{m=1}^{|\mathcal{C}|} l(w_m; \theta) \tag{4.1}$$

where $\mathcal{C}$ is the training corpus, and the loss function $l(w_m;\theta)$ can be expressed as

$$l(w_m;\theta) = -\log p(w_m|\mathbf{w}_{\text{sub}}^{m-1}) \tag{4.2}$$

where $\mathbf{w}_{\text{sub}}^{m-1} = (w_1, w_2, \ldots, w_{m-1})$ denotes a subsequence of $\mathbf{w}$.

In practice, instead of minimizing the loss function in equation (4.1) globally, it's not uncommon to utilize SGD [63] to optimize the batch losses [64] described as below:

$$l(x_i;\theta_i) = -\frac{1}{|\mathcal{C}_i|}\sum_{m=1}^{|\mathcal{C}_i|}\log p(x_{i,m};\theta_i) \tag{4.3}$$

$$\triangleq -\frac{1}{|\mathcal{C}_i|}\sum_{m=1}^{|\mathcal{C}_i|}\log p(x_{i,m}|x_{i,1}, x_{i,2}, \ldots, x_{i,m-1}), \quad i = 1, \ldots, N \tag{4.4}$$

where $\mathcal{C}_i$ is the set of tokenized words in the ith batch, and $x_{i,m}$ is the mth token in $\mathcal{C}_i$.

Intuitively, it's hard to create the perfect clusters for most of the text documents. Accordingly, we utilize the soft clustering of corpus in the modeling. On the one hand, let each ensemble element specialize on a distinct subset of the batches will force the ensemble element to capture the structure and characteristic of text in those batches. On the other hand, standard ensemble models improve the modeling results by reducing the generalization errors. The soft clustering balance the effects of these two architectures by assigning batch weight to each batch, which enables our mixture models to gain the beneficial effects from the two different configurations.

The algorithm we use to train ensemble models on the corpus is presented in Algorithm 1. Assume we have K ensemble elements. We first initialize the batch weights $v_{i,k}^{(0)}$ randomly, such that $\sum_{k=1}^{K} v_{i,k}^{(0)} = 1$ and $v_{i,k}^{(0)} > 0$. In the training epoch $c+1$, if we are considering smoothing batch weights and given the initial smoothing threshold batch weights, then we should smooth $\mathbf{v}^{(c)}$ and this will be discussed later in §4.4.

Algorithm 1 Batch-wise Mixture Modeling for Ensembles

procedure BaMME($\mathbf{x}, epochs, s_0$) ▷ s_0 is the initial smoothing threshold of weights, and the i^{th} element in $\mathbf{x}$ is the tokenized text in batch i
  Initialize weights $v_{i,k}^{(0)}$ for $i = 1, \ldots, N$ and $k = 1, \ldots, K$, such that $\sum_{k=1}^{K} v_{i,k}^{(0)} = 1$.
  Initialize deep neural network parameters $\mathbf{\Theta}^{(0)}$.
  $s = s_0/(1-\eta)$ ▷ η is the weight decaying rate
  for c in $epochs$ **do**
    if $s > 0$ **then**
      $s = s \times (1-\eta)$
      Use Algorithm 2 to smooth weights $\mathbf{v}^{(c-1)}$ with smoothing threshold s.
    end if
    $\mathbf{\Theta}^{(c)} = \mathbf{\Theta}^{(c-1)}$
    $\pi_{i,k}^{(c)} = v_{i,k}^{(c-1)}$ for $i = 1, \ldots, N$ and $k = 1, \ldots, K$.
    $\mathbf{v}_k^{(c-1)} = \left(v_{1,k}^{(c-1)}, \ldots, v_{N,k}^{(c-1)}\right)$ for $k = 1, \ldots, K$.
    Compute likelihood $\phi_k^{(c)}(x_i)$ for $i = 1, \ldots, N$ and $k = 1, \ldots, K$.
    Batch-wise minimize weighted negative log-loss(using batch weight $\mathbf{v}_k^{(c-1)}$ for ensemble element $k, k = 1, ..., K$), and update parameters $\mathbf{\Theta}^{(c)}$ batch-wisely.
    Compute validation loss and save updated models if necessary.
    $v_{i,k}^{(c)} = \pi_{i,k}^{(c)}\phi_k^{(c)}(x_i) / \sum_{k=1}^{K} \pi_{i,k}^{(c)}\phi_k^{(c)}(x_i)$ for $i = 1, \ldots, N$ and $k = 1, \ldots, K$.
  end for
end procedure

Let's denote the computed likelihood vector for batch i in the training epoch c as $\phi^{(c)}(x_i) = \left[\phi_1^{(c)}(x_i), \ldots, \phi_K^{(c)}(x_i)\right]$. To minimize the batch loss shown in (4.4), it's enough to minimize the following weighted losses for batch i (see in Appendix B):

$$v_{i,k}^{(c)} \cdot l^{(c+1)}(x_i; \theta_{i,k}), \quad k = 1, \ldots, K \tag{4.5}$$

where $v_{i,k}^{(c)} = \frac{\pi_{i,k}^{(c)}\phi_k^{(c)}(x_i)}{\pi_i^{(c)\intercal}\phi^{(c)}(x_i)}$ is the batch weight for batch i, and $\pi_i^{(c)} = \left[\pi_{i,1}^{(c)}, \ldots, \pi_{i,K}^{(c)}\right]$ is the mixture weight vector. Mixture weight $\pi_{i,k}^{(c)}$ is set to be the batch weight $v_{i,k}^{(c-1)}$ as shown in Appendix A.

Algorithm 2 Weight Smoothing in Proportion

procedure WT_SIP($\mathbf{v}$, s) ▷ s is the smoothing threshold of weights
  for i in N **do** ▷ N is the number of batches
    $\mathbf{v}_i = (v_{i1}, \ldots, v_{iK})$
    $\mathbf{u}_i = (v_{i1}, \ldots, v_{iK})$
    $c = \mathrm{argmin}(\mathbf{v}_i)$
    $l = \mathrm{argmax}(\mathbf{u}_i)$
    $\delta = 0$
    while $\mathbf{v}_{ic} < s$ **do**
      $\mu = s - \mathbf{v}_{ic}$
      $\Delta = \left\{k \in [K] : \mathbf{v}_{ik} > s + \delta \text{ for } \mathbf{v}_{ik} \text{ in } \mathbf{v}_i\right\}$
      $\lambda = \sum_{k \in \Delta} \mathbf{v}_{ik}$
      for k in Δ **do**
        $\mathbf{v}_{ic} = \mathbf{v}_{ic} + \mu \times \mathbf{v}_{ik}/\lambda$
        $\mathbf{v}_{ik} = \mathbf{v}_{ik} - \mu \times \mathbf{v}_{ik}/\lambda$
      end for
      $\delta = \min\left(s/K, \mathbf{u}_{ic}/\mathbf{u}_{il}\right)$
      if $\mathbf{v}_{il} \geq s + \delta$ **then**
        $\mathbf{v}_{ic} = \mathbf{v}_{ic} + \delta$
        $\mathbf{v}_{il} = \mathbf{v}_{il} - \delta$
      end if
      $c = \mathrm{argmin}(\mathbf{v}_i)$
    end while
  end for
end procedure

4.4 Experiments

In this section, we introduce our experimental settings and show the comparison results of batch-wise mixture modeling and standard ensemble modeling.

4.4.1 Experimental Settings and Datasets

We evaluate two modeling methods on two widely used corpora: Penn Treebank (PTB) [33] and Wikitext-2 (WT2) [48] based on perplexity. And we will conduct additional experiments on synthetic text data to further show the effectiveness of our modeling method. We follow the optimization techniques introduced by Merity et al. [65] and Yang et al. [66]. Some of the hyper-parameters used in our experiments are adopted from Yang et al. [66]. Other hyper-parameters, such as the initial smoothing threshold s_0 and weight decaying rate η for BaMME, are tuned mannully based on the validation performance. Currently, we have tuned hyper-parameters for 3-element ensemble models, and all these models are trained by SGD and accelerated SGD (ASGD) for 40 epochs. Training big pretrained models on large dataset is expensive, so we work in a simulated setting with small RNNLMs, which simulate the case where the models are too small to learn the entire corpus and we can instead use an ensemble to get better performance. The model size for PTB is smaller than that for WT2 due to the fact that the data size of WT2 is larger than PTB (see Appendix C for our hyper-parameters).

4.4.2 Smoothing of Batch Weight

During the training process, most of the batch weights we get are close to 0 or 1. Figure 4.1 shows the density plot of ensemble weights in the first five training epochs for a 4-element ensemble model. Except for the randomly initialized batch weights in the first epoch, most of the batch

weights are extreme values for the 4 ensemble elements. Extreme batch weights lead the hard clustering of training corpus, and this will neglect the effects that the ensemble training has on our dataset. Hence we have to smooth the batch weights.

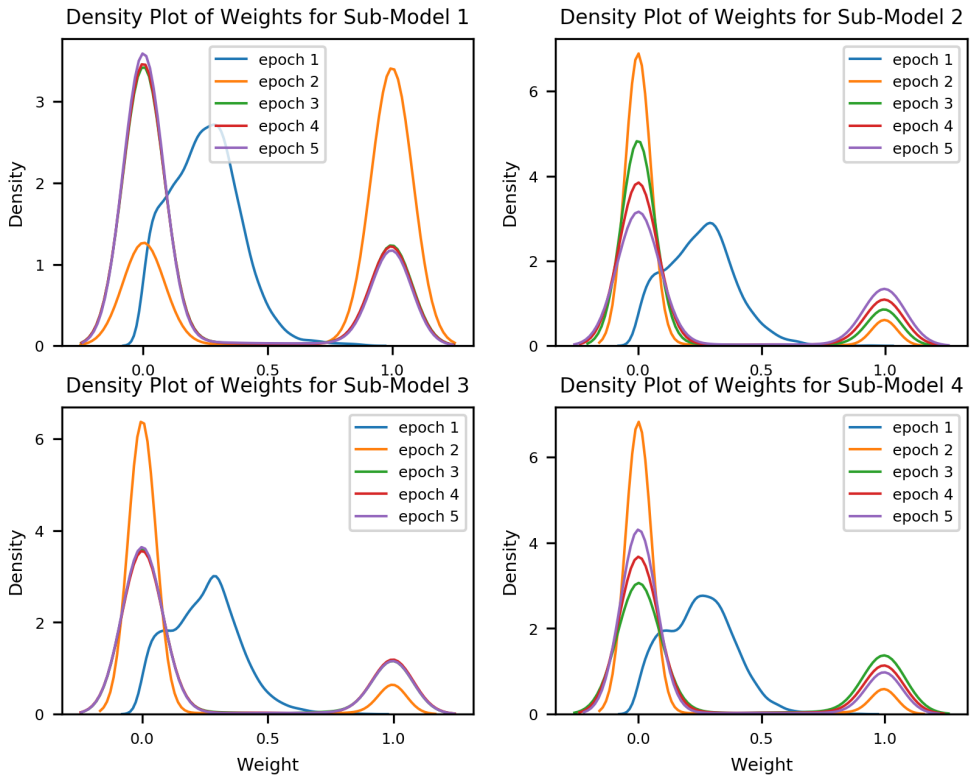

Figure 4.1: Density plot of weights for a 4-element ensemble model. Most of the weights are extreme values before smoothing.

We can smooth batch weight $w_{ik}^{(c-1)}$ in epoch c by employing Algorithm 2, which is called *Weight Smoothing in Proportion (WT_SIP)*. The basic idea behind this smoothing method is simple. With the given smoothing threshold s, we increase the small batch weights to ensure that all the weights are greater or equal to s. We keep the record of set Δ where each weight is greater than s, and add the weight smaller than s with values in proportional to the weights in Δ to fill the gap. We further augment the small weight with δ, which is the restricted relative ratio of small

weight to the largest ensemble weight, to reflect the relative difference of original extreme weights. Examples to illustrate how Algorithm 2 works can be found in Appendix D.

4.4.3 Results and Analysis

In the experiments, we implement the code for standard ensemble modeling, which is the baseline. And we compare it experimentally to BaMME. In Table 4.1, we show the comparison results of our proposed method and the baseline. Under its best settings, BaMME outperforms the baseline, and substantially improves the results on test datasets of PTB and WT2, by 13.73 points and 4.18 points in perplexity, respectively.

Table 4.1: Results of BaMME and Baseline methods on the validation and testing datasets of PTB and WT2 corpora. BaMME outperforms the Baseline method for both of the two datasets.

Method	**PTB**		**WT2**	
	Validation	**Testing**	**Validation**	**Testing**
Baseline	220.98	208.93	205.19	186.87
BaMME	204.57	195.20	199.14	182.69

When smoothing the batch weights in BaMME, we employ a geometric series function to describe the effect of smoothing threshold have on smoothing the batch weights. As the training step increases, the smoothing threshold shrinks. This leads the increment of divergence in batch weights, which enables the ensemble element to specialize on a subset of batches. But at the same time we don't want the smoothing threshold to decay too fast since moving too fast to hard clustering will weaken the beneficial effects from ensemble training. We further conduct experiments to compare the effects of weight decaying rate η. In Table 4.2, we show the validation and testing perplexities on PTB when using BaMME with different decaying rate, and it achieves better

validation/testing perplexity when we use relatively smaller weight decaying rate.

Table 4.2: Validation and testing results (on PTB) of BaMME with different weight decaying rate. The models achieve better validation and testing perplexities when we use relatively smaller weight decaying rate.

Decaying rate η	0.010	0.008	0.005	0.004	0.003	0.002	0.001
Validation	206.94	206.41	206.14	204.63	204.64	204.59	204.57
Testing	197.50	196.96	196.65	195.25	195.25	195.20	195.20

4.5 Further Analysis and Experiments

From previous sections, we have seen the effectiveness of BaMME algorithm and the smoothing method. However, we are still not so clear how the soft-clustering mechanism employed during the training contribute to the overall improvement when applying BaMME algorithm.

In this section, we will generate synthetic text data given sampled text sequences in different topics, and try to navigate each ensemble model to focus on certain part of the synthetic data set during the training by applying a pre-evaluation procedure and clustering method.

4.5.1 Synthetic Data Generation

To investigate whether having different ensemble elements focus on different subsets of the text with different characteristics will augment the training, we want to generate a text data with specific distributions of types of vocabulary. Specifically, we first selected three clusters of sentences in music, energy and mathematics from Penn Treebank [33] dataset, and there are 20 sentences in each cluster. After the selection procedure, we utilize three light RNN models (generating models) to generate three cluster of batches of sentences.

Before fusing the three generated datasets into synthetic training, validation and testing sets for ensemble modeling, we evaluate the performance of generating model on the corresponding

generated set. We expect the ith generating model performs the best on the ith generated dataset (i = 1, 2, 3). Otherwise, the datasets are not properly generated, and we have to check the generating models we used.

Table 4.3: Performance of generating models on the generated texts. The ith generating model performs the best on the ith generated dataset, which indicates that the datasets are properly generated.

Model\Text	1	2	3
1	177.09	457.94	415.18
2	609.60	119.68	580.94
3	441.24	465.08	171.45

As expected, the diagonal perplexities in Table 4.3 are significantly lower than the perplexities in its vertical and horizontal lines, i.e., the i th generating model achieves the best performance on the i th generated dataset (i = 1, 2, 3). We then split the generated dataset i into training set i, validation set i and testing set i (i = 1, 2, 3), and fuse training sets 1-3 into the final synthetic training dataset with size 72K. Similarly, we can fuse the validation sets and testing sets into final validation set and testing set, respectively. The sizes of synthetic validation set and testing set are all 9K.

4.5.2 Batch-wise Pre-selection of Ensembles

With the synthetic dataset, we can train the ensemble models on it. In BaMME, we navigate each ensemble model to focus on certain part of the training text by weighting the loss for each batch. Although we see the effectiveness of BaMME in §4.4, we are still seeking more accurate ways to do the navigation. One idea here is to employ a pre-evaluation procedure.

Based on the fact that the text sequences in the same batch of our synthetic data usually share

similar vocabulary or are consisted of semantically related tokens, we use the ensemble models to train the $p\%$ of the text data and use each trained ensemble model to compute the log-likelihood in the pre-evaluation procedure, and then we choose the ensemble model which achieves the highest log-likelihood to train the remaining part of the text data within that batch. In this way, we can find the most appropriate ensemble model for each batch of text data. The initial learning rate and batch size we used during the training are 12 and 6, respectively. The experimental results on synthetic data are presented in Table 4.4.

Baseline Train the ensemble models. Computed likelihoods are averaged, and then transformed to perplexity.

Batch-wise Pre-selection of Ensembles Train the ensemble models on $p\%$ of the text data in each batch, and select the most appropriate model to train the remaining text data in the batch. During evaluation, the log-likelihood of the first $p\%$ of data in each batch is computed by the ensemble models, and the log-likelihood of remaining data in that batch is computed by the model selected during the pre-evaluation procedure.

Table 4.4: Results of Baseline and Batch-wise Pre-selection of Ensembles methods on the synthetic data. The latter method produces the best testing result when 35% of the text in each batch of the synthetic data is used in the pre-training and pre-evaluation procedures.

Method	Baseline	Batch-wise Pre-selection of Ensembles				
p %	-	15%	20%	25%	30%	35%
Validation	140.42	144.20	145.04	139.30	147.83	143.90
Testing	149.28	147.43	148.09	148.73	160.06	146.14

In Table 4.4, we summarize the evaluation results on synthetic validation and testing datasets. We note that the best validation result achieved when we use 25% of text data to do pre-evaluation in each batch, and it is only comparable with the baseline validation result. The testing perplexities get improved, and it get reduced by 3 units when we use 35% of text data to do pre-evaluation.

The pre-evaluation method seems good as we're trying to find the most appropriate model to train the batch of data. However, even with the pre-evaluation procedure, we still can not guaranteen that we choose the best ensemble model to train the entire text data in each batch. Remember that the synthetic training set is consisted of three parts: text in music, energy and mathematics. What if we manually force each ensemble model to train each part of the text data directly, and keep using the pre-evaluation procedure when evaluating the validation and testing datasets? The evaluation results got further improved! Actually, the perplexities on validation and testing datasets are reduced to 131.27 and 137.84, respectively. Navigating each ensemble model to focus on texts sharing similar vocabulary or are semantically related seems to be helpful for the training. Inspired by this idea, we can cluster the text data first, and then use the ensemble models to train the clustered dataset.

4.5.3 Clustering-based Selection of Ensembles

In order to navigate each ensemble model to better focus on certain part of the training dataset, we can first cluster the initial training dataset into several clusters based on the text semantic relationships. Specifically, for the synthetic training data created in §4.5.2, we can use clustering method like KMeans method [67] to cluster the bachified text data into three clusters, and then each ensemble model will be responsible for training the batch of text data within the same cluster.

When clustering the synthetic text data, most (more than 90%) of the batch of text data in the first 1/3 part and last 1/3 part of the entire text data are clustered into same cluster, however, the first 1/3 part and last 1/3 part of data are actually generated from sentences in two different topics! One possible reason for this phenomenon is the text data in these two parts are not distinguishable enough during the clustering process. Therefore, we re-select 20 sentences in sports from Wikitext-2 [48], and use the light RNN model to generate a 30K set. We split the new generated set into

training, validation and testing sets, and fuse them with the previous corresponding sets generated from sentences in music and energy to form the new synthetic training, validation and testing sets.

After obtaining the new synthetic training set, we use the KMeans algorithm to cluster it, and set the size limit of each cluster to be 90 (we have 255 batches of text data in total). In this way, about 95% of the text data got clustered correctly. During the ensemble training, we select one ensemble model to train each cluster of batch data, and the details of modeling are described below. And the **Baseline** is the same as in §4.5.2.

Clustering-based Selection of Ensembles Cluster the original bachified dataset into n clusters, and navigate each ensemble model to train one cluster of data during the training process. During evaluation, the log-likelihood of the first $p\%$ of data in each batch is computed by the n-ensemble model, and the log-likelihood of remaining data in that batch is computed by the model selected during the pre-evaluation procedure.

Table 4.5: Results of Baseline and Clustering-based Selection of Ensembles methods on the synthetic data. The latter method outperforms the baseline method, and produces the best result when 30% of the text in each batch of the synthetic data is used in the pre-evaluation procedure.

Method	Baseline	Clustering-based Selection of Ensembles			
p %	-	15%	20%	25%	30%
Validation	141.86	132.49	133.01	134.06	133.40
Testing	164.66	157.29	156.56	156.47	154.64

We summarize our evaluation results in Table 4.5, and we can see that both of the validation and testing results get improved. Clustering the training set helps better navigate each ensemble model focus on a proportion of the text data. Actually, most of the text data within a cluster are generated from the same text sequences, and they are semantically related. When each ensemble model focus on the text data sharing similar characteristic or even vocabulary, it can be trained

better as the training of former batch of text data will benefit the training of latter ones, and it can help capture the inherent patterns/characteristics of the text sequences or phrases in the subsequent batches.

4.6 Conclusion

In this chapter, we have proposed some effective modeling methods for language ensemble training. Our experiments demonstrate that the proposed modeling methods are effective in improving the evaluation results compared with the traditional ensemble modeling. BaMME utilizes loss weighting to effectively navigate the ensemble model focus on certain part of the text data, and smoothing technique is applied to avoid getting extreme weights. Batch-wise Pre-selection of Ensembles aims to navigate each ensemble better focus on part of the data without using weighting of losses, and it gives some insights to Clustering-based Selection of Ensembles method.

When working on the synthetic dataset, Clustering-based Selection of Ensembles achieves much better results since after clustering, the text data in each cluster are more semantically related with each other. However, real datasets are usually difficult to perfectly clustered using hard clustering algorithm. In the future, we would like to incorporate soft-clustering method in clustering-based selection of ensembles method to better train the real datasets.

CHAPTER 5
TRAINING DATA SAMPLING FOR EFFICIENT LANGUAGE MODELING

5.1 Introduction

Language modeling has been an important task in Natural Language Processing (NLP) as it helps better understand the output of many downstream tasks such as machine translation ([10], [13]), summarization ([16], [17]) and dialogue systems ([12], [14], [15]), etc. The input of language models is usually a text corpus containing sequences of word tokens, and the model learns the probability distributions of text, which will help capture the structures of language and extract useful information contained in the corpus. In recent years, many language modeling work ([68]–[70]) have achieved state-of-art results on various large, diverse benchmark corpora such as Penn Treebank (PTB) Corpus [33], Wikitext-103 [48] and the Billion Word Corpus [49].

Training on large corpora mentioned above usually gives models which can evaluate text data as accurate as possible. However, to achieve state-of-art results on those corpora requires training very large and sophisticated models, which can be difficult to train due to memory constraints and prohibitive computational costs. Even using models with relatively small capacities to do the training, the time costs could still be an issue as the sizes of the benchmark corpora or real text corpora are usually much larger than imagined. We can subsample a small portion of the sentences in the original corpus, and train the sentence level language models. By effectively subsampling the corpus, the language model training could be more efficient since less memory are needed and the computational costs can be largely reduced.

On the one hand, we aims at selecting a set of sentences that can be as diverse as the original

corpus; on the other hand, the sentences we sampled in the set should be more informative than a uniformly sampled training set. To keep the diversity of the subsampled set, we will first cluster the corpus so that when sampling in different clusters, the sampled sentences are in different topics or semantically different with each other. We also employ the importance sampling technique [18] to select the training data that is as informative as possible.

We propose two novel subsampling methods for selecting text set for more efficient sentence-level RNN language modeling. The first method combines the clustering technique and the importance sampling technique to make the selected training set as diverse and informative as possible. More specifically, we apply the Latent Dirichlet Allocation (LDA) clustering algorithm to cluster the original corpus into c clusters, and then sample sentences in each cluster according to the probability score calculated with n-gram perplexity of sentence. It's noteworthy that the sample size in each cluster is in proportional to the square root of cluster size, and we re-weight the corrective weights to reflect the effect of cluster size. The experiments on Wikitext-103 [48] Corpus show the effectiveness of our newly developed subsampling method as the heldout perplexities are lower than the baselines. By utilizing the new developed subsampling method, the additional computational costs are limited as clustering text corpus, training and querying n-gram language models are fast.

In the large benchmark corpus or real text corpus, it is common to see some hard-to-learn sentences during language model training. For this reason, we employ the dataset cartography technique [19] to identify those collective outliers [50], and remove varying numbers of outliers during the pre-selection process in our second sampling method. Specifically, we randomly select a text set which has a much smaller size than the original corpus, and we use RNN model to pre-train the selected text set. Then a 2D Dataset Map is plotted, and we systematically remove $p\%$ of the selected data pool. After obtaining the pre-selected text set, we continue to cluster it and

employ importance sampling technique to further subsample a training dataset with fixed token size. We experimentally evaluate the effectiveness of the second subsampling method, and the heldout perplexities of RNN models trained with the subsampled training set are much lower than the baseline results. The heldout perplexities we get here are also lower than those we get using the first subsampling method, with the fact that additional computational costs are used for the second subsampling method.

5.2 Related Work and Knowledge

5.2.1 Importance Sampling

As introduced in §4.2.2, importance sampling has been an useful technique applied in optimization process and deep neural network training to accelerate the training process and improve the modeling accuracy ([58]–[61], [71], [72]). For example, adaptive mixture weights were presented by Borsos [61], who propose a framework for variance reduction where sampling distributions are combined into a mixture and loss function is multiplied with the importance weight during the course of the optimization. Importance sampling has also been used to sample data for domain specific language modeling [42].

Fernandez et al. [18] developed multiple importance sampling distributions which bias the training set to select sentences that can increase the training set's information content. They also used weights during RNN language model training, and the weights were set to the reciprocal of the sentence's selection probability. We propose an generalized form of the importance sampling distributions developed by Fernandez et al. And we also use corrective weights during RNN language model training. But instead of setting the corrective weights to the reciprocal of the sentence's selection probability directly, we re-weight the weights to reflect the effect of cluster size.

5.2.2 Interpreting and Analyzing Datasets

It is commonplace for researchers to interpret and analyze large datasets in modern machine learning studies. Properly Assessing the quality of data and removing redundancies [73] for better model training has been an important and challenging problem.

Dataset Maps [19] has been an very useful tool to characterize and diagnose datasets. It is first introduced by Swayamdipta et al., who propose a model-specific graph for visualizing the learnability of individual training examples. Two model-specific measures, model confidence and prediction variability, are used in the graph to indicate the "learnability" of training examples ([19], [74]).

Karamcheti et al. [20] use Dataset Maps to profile the learnability of training examples in active learning, and they find that active learning methods prefer sampling *hard-to-learn* examples, which leads to poor performance. By removing a large proportion of those examples, the performance of active learning gets improved.

Unlike previous datasets analyzed by Dataset Maps that are usually used in classification problems, the datasets we consider are used in language modeling, and we treat each sentence as an individual example during training. Instead of using the model confidence as one of the indicators of an training example's "learnability", we compute the average log-perplexity of each sentence over training epochs, and use it as one of the two measures. The variability is also computed based on the log-perplexity of sentences. We discover that by filtering out a proportion of *hard-to-learn* examples in the pre-selected text set, we get lower helout perplexities of RNN models trained with the final subsampled training set.

5.3 Methodology

RNNLMs are trained on a given training dataset to minimize to loss function defined in (4.1). As mentioned in §5.1, it is usually difficult to train very large and sophisticated models on large corpora due to memory constraints and prohibitive computational costs. Sentence-level language models for large corpora can be learned by training on sets of sentences subsampled from the original corpus. We propose two new subsampling methods for selecting sentence set for more efficient sentence-level RNN language modeling. Both of these two methods apply the clustering technique and the importance sampling technique to make the selected training set as diverse and informative as possible. The second subsampling method has a pre-selection procedure, in which we narrow the text set used for clustering and importance sampling by initial uniform sampling from the original corpus and removing some *hard-to-learn* sentences according to Dataset Maps.

In this section, we first generalize the importance sampling distributions developed by Fernandez et al. (§5.3.1), and then present our two subsampling methods (§5.3.2, §5.3.4).

5.3.1 Generalized Sampling Distributions

In Fernandez et al. [18], several sampling distributions are proposed for selecting training sentences for an RNN language model, and they are formed based on the perplexity of sentences, mean perplexity and standard deviation of perplexities. In addition to the Z-score Sampling distribution, Fernandez et al. also proposed Limited Z-score Sampling distribution and Squared Z-score Sampling distribution in order to smooth the distribution in the weight space. We generalize the distributions in the following form:

$$p(s) = \begin{cases} k\Big(\alpha\big(\frac{ppl(s)-\mu_{ppl}}{\sigma_{ppl}}\big)^{\tau} + \beta\Big), & \text{if } ppl(s) > \mu_{ppl} \\ k, & \text{otherwise} \end{cases} \tag{5.1}$$

where $ppl(s)$ is the perplexity of sentence s, μ_{ppl} is the mean perplexity, σ_{ppl} is the standard deviation of the perplexities, and k is a normalizing constant to make the whole distribution a proper probability distribution.

There are several constant parameters in 5.1. α is a constant weight that determines the importance of the z-score in calculating the sampling distribution. τ is the order of the z-score, and it describes the shrinkage rate of the z-score in the distribution. And β determines the displacement of the z-score in the probability. These constant parameters are chosen based on the evaluation results of validation set in the experiments.

5.3.2 Clustering-based Importance Subsampling (CLIS)

To make the subsampled training set as diverse as possible, we first apply an unsupervised clustering algorithm to cluster the original corpus, and then sample sentences in each cluster. In this way, the sentences sampled from different clusters are in different topics or semantically different with each other, and this makes the subsampled set of sentences more diverse in content than the randomly sampled set. Before applying the clustering algorithm, we use the *CountVectorizer* class from sklearn [75] to create a document-term matrix, and remove all the stop words as they do not really contribute to the modeling.

Instead of utilizing K-means algorithm to partition the corpus, we utilize Latent Dirichlet Allocation (LDA) algorithm [76] to assign sentences in the corpus to a mixture of clusters as this gives more realistic results. After getting the probabilities of each sentence belonging to the clusters, we

choose the cluster with largest probability as the cluster that the sentence belongs to.

With clustered set of sentences, we need to calculate the selection probabilities of sentences using (5.1). To calculate the selection probabilities, an n-gram model was trained on a pre-sampled heldout set with the same number of tokens used in the final RNN model training. For example, if we want to subsample a 2 million text set for training an RNN model, we first train the n-gram model on a 2 million heldout text set and use the trained n-gram model to calculate sentence perplexities in (5.1). The n-gram models are trained as 5-gram models with Kneser-Ney discounting [34] using ***SRILM*** [77]. And we query the models using ***KenLM*** [78] when calculating sentence perplexities.

After getting the n-gram perplexities of sentences and the calculated selection probabilities, we sample sentences in each cluster with the selection probabilities. If the number of sentences in the i^{th} cluster is r_i, the number of tokens we subsample from the i^{th} cluster, m_i, satisfies

$$m_i \propto \sqrt{r_i/\mu_r}, \quad i = 1, \ldots, c \tag{5.2}$$

where μ_r is the average of the cluster sizes, and c is the number of clusters.

Fernandez et al. [18] has verified the effectiveness of corrective weights used during RNN model training. In our work, we continue applying corrective weights during final model training. To reflect the effect of cluster size, we re-weight the corrective weight for each sentence s_k as

$$\sqrt{r_{k_i}/\mu_r}/p(s_k), \quad k = 1, \ldots, n \tag{5.3}$$

where $p(s_k)$ is the selection probability of sentence s_k, and here we assume sentence s_k belongs to the ${k_i}^{\text{th}}$ cluster.

5.3.3 Analysis via Dataset Maps

In §5.3.2, we have introduced our first subsampling method (CLIS) for selecting set of sentences for RNN language modeling, and we will show the effectiveness of this method in §5.4. We are still wondering if all the sentences we sampled are efficient for the RNN model training. In other words, we would like to know which sentences are inefficiency ones, and we shouldn't sample them.

Dataset Maps [19] is a very useful tool to characterize and diagnose the underlying dataset. A Dataset Map is a model-specific graph which profiles the learnability of sentences in the training dataset. As an RNN model trains multiple epochs and sees the same sentence repeatedly, we want to characterize how well the RNN model can learn the sentence across all the epochs.

Swayamdipta et al. [19] define the confidence as the mean model probability of the true label across training epochs to capture how confidently the model assigns the true label to the observation when analyzing classification datasets. Inspired by this idea, we use **mean log-perplexity** of the sentence across epochs to characterize how well the RNN model can learn the sentence in the training set, and it is defined as,

$$\hat{\mu}_k = \frac{1}{T}\sum_{t=1}^{T} \log ppl(s_k; \boldsymbol{\theta}^{(t)}), \quad k = 1, \ldots, n \tag{5.4}$$

where $\boldsymbol{\theta}^{(t)}$ denotes the model parameters at the end of t^{th} epoch, and T is the total number of epochs the model trains. A sentence with relatively low **mean log-perplexity** can be treated as "easier" for the learner to learn as our goal is to minimize the average negative log-likelihood of training set defined in (4.1).

Another important statistic which measures spread of $\log ppl(s_k; \boldsymbol{\theta}^{(t)})$ is **variability**, and it is

defined as the standard deviation of log perplexity:

$$\hat{\sigma}_k = \sqrt{\frac{\sum_{t=1}^{T} \left(\log ppl(s_k; \boldsymbol{\theta}^{(t)}) - \hat{\mu}_k \right)^2}{T}}, \quad k = 1, \ldots, n \tag{5.5}$$

When using the trained RNN model to evaluate a given sentence across epochs, if the evaluated log perplexities are pretty close, the sentence will have low **variability**. In the contrary, if the evaluated log perplexities differ a lot across epochs, the sentence is considered to be *ambiguous* to the model no matter how high or low its **mean log-perplexity** is.

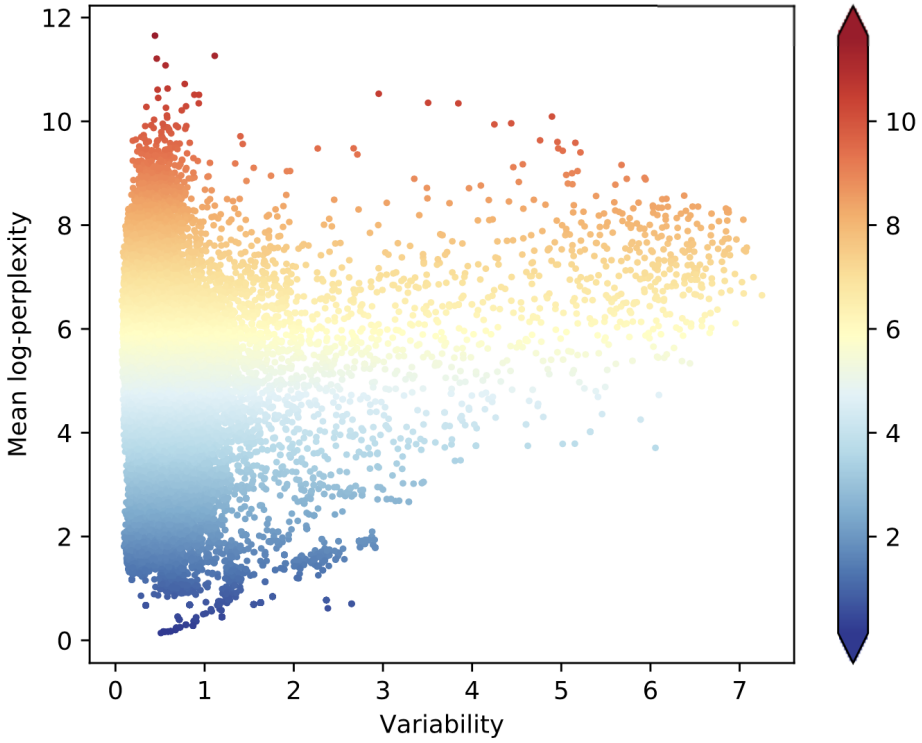

Figure 5.1: Dataset Map for RNN Model on 10M Wikitext-103. *Easy-to-learn* examples fall in the bottom-left region, while *hard-to-learn* examples lie in the top-left region. The examples fall in the upper right quadrant are referred as *ambiguous* examples.

With the two introduced measures, we can construct the dataset map for a 10M training set randomly sampled from the Wikitext-103 dataset. We show the Dataset Map built with LSTM language model in Figure 5.1.

The model we used to built the Dataset Map is the same model we used for training subsampled set of sentences, and more details of the model are in §5.4. When building the Dataset Map, we remove the sentences with top 0.2% **variability** since those sentences are generally nonsensical, and selecting those sentences has minimal contribution to the model performance improvement. Those sentences are also removed before the pre-selection procedure introduced in §5.3.4. There are several distinctions between the Dataset Map we built and the Dataset Maps built by Swayamdipta et al. [19]:

1. Instead of plotting the average model confidence assigned to the correct answer over epochs, the y-axis plots the **mean log-perplexity** of sentences across training epochs on our map.

2. Examples are colored by their **mean log-perplexity** to describe their "learnability" on our map, while instances are placed by a coarse statistic, correctness, to describe their "learnability" on the map plotted by Swayamdipta et al.

3. Swayamdipta et al. build the maps by training the models on the entire datasets, while our map is built by training the RNN model on a randomly pre-selected subset of the training set for saving the computational costs when applying the sencond subsampling method (§5.3.4).

In Figure 5.1, a large proportion of the examples lie in the bottom-left part (low **mean log-perplexity** and low **variability**) of the graph, and they are referred as *easy-to-learn* examples for the model. About 30% of the training examples lie in the top-left part of the graph, and we refer them as *hard-to-learn* (for the model) as they get high log-perplexities during model evaluations and the **variability** of log-perplexity across epochs is low for those examples. The golden region

separate the *easy-to-learn* and *hard-to-learn* examples. In the right-hand side, there is a small group of training examples presenting in the upper quadrant. These examples have relatively high **variability**, and it's hard to predict whether they get relatively high or low log-perplexities during model evaluations. This type of training examples are referred as *ambiguous* (to the model) by Swayamdipta et al. [19].

Ambiguous and *hard-to-learn* examples could be the most informative data for model learning [19], since they bring challenges to the modeling ([79], [80]). Swayamdipta et al. [19] show that training on the set containing large amount of *ambiguous* data achieves much better result compared with the random selection baseline, providing that a proper amount of *easy-to-learn* data is included in the training set. Karamcheti et al. [20] find that *hard-to-learn* examples degrade the performance of active learning, and by removing a large proportion of those examples, the performance of active learning gets improved. We hypothesize that *hard-to-learn* examples have negative effect on the performance of language modeling since it's difficult to get the high log-perplexities of those examples reduced during pre-training. To verify this, we will introduce a way to remove a certain amount of *hard-to-learn* examples in the second subsampling method (§5.3.4).

5.3.4 Clustering-based Importance Subsampling with Inefficiency Examples Removed (CLISIER)

We now describe the framework of our second subsampling method (CLISIER) for subsampling a diverse and informative set for sentence-level RNN language model training. The high-level overview of this approach is shown in Figure 5.2. In CLISIER, we first characterize the dataset using Dataset Map (§5.3.3), and remove a proportion of inefficiency examples to get a pre-selected set. After getting the pre-selected set, we apply an unsupervised clustering algorithm to cluster it, and sample sentences in each cluster with selection probabilities defined in 5.1. Finally, we utilize

the RNN model to train the subsampled training set with corrective weights defined in (5.2).

Our final goal is to learn a sentence-level language model for large corpora by training it on a subsampled set of sentences. This will save a lot of memory and computational costs, and makes

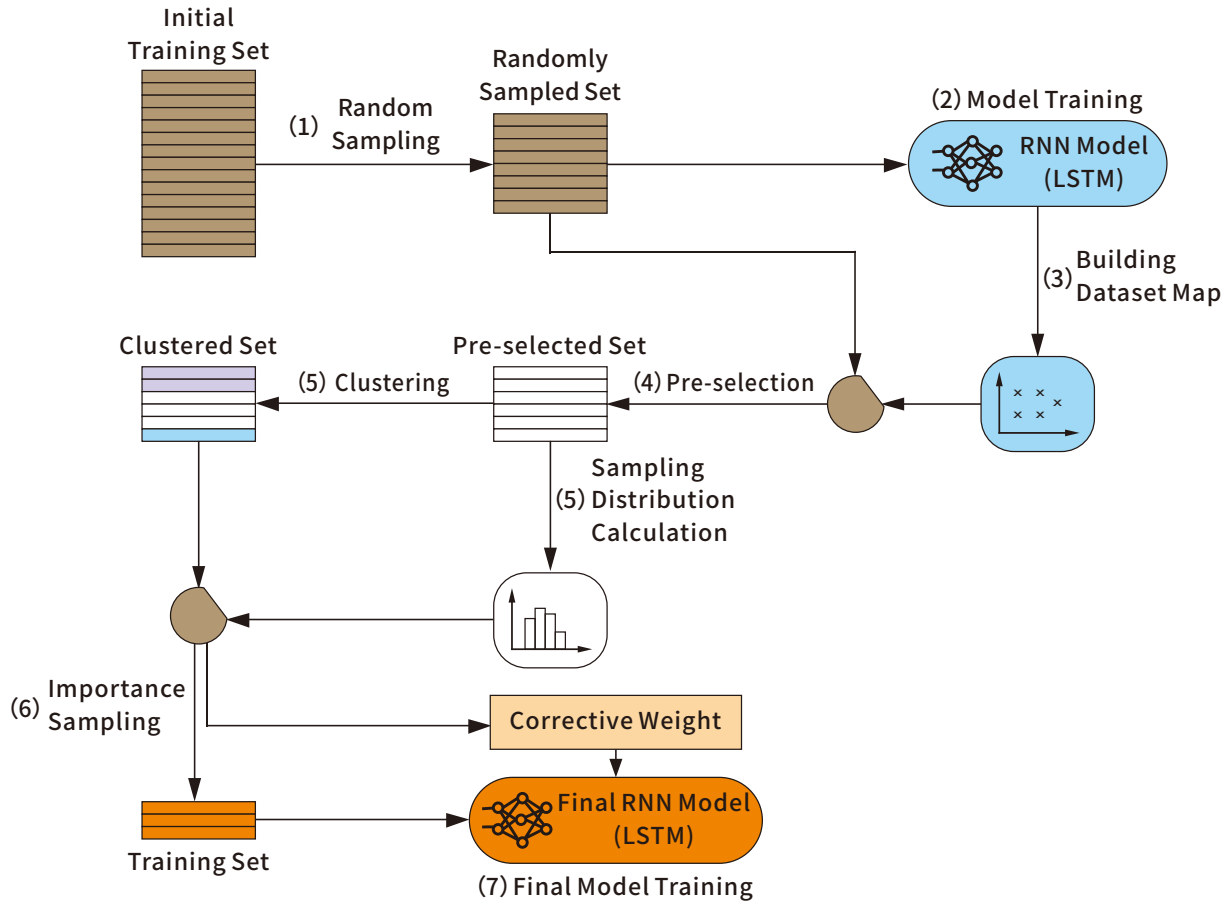

Figure 5.2: Illustration of the CLISIER method. We first (1) randomly sample a 10M set from the initial training set, and (2) use an RNN model to train it. Then, we (3) build the Dataset Map with calculated mean log-perplexity and variability, and remove $p\%$ of examples with (4) a pre-selection procedure. We (5) cluster the pre-selected set and calculate the sampling distribution. This is followed by (6) importance sampling to get the subsampled training set. Finally, we (7) train the RNN model on the subsampled training set with corrective weight.

the model training more efficient and easier. To align with this goal, when building the Dataset Map for the underlying corpus (e.g., the Wikitext-103 corpus [48]), we randomly sample a 10M

token set from the original corpus, and train the RNN model on it to get the **mean log-perplexity** and **variability** for each sentence.

With the randomly sampled set of sentences and pre-trained RNN model, we need to create a pre-selected set with varying numbers of training examples removed. Basically, we want to remove different amount of *hard-to-learn* examples to see if it can improve the performance of final RNN modeling on subsampled set. Instead of removing *hard-to-learn* examples directly, we calculate and sort the quotient of the **variability** and **mean log-perplexity** of example in initial randomly sampled set, and remove examples with a low quotient value. There are two possible cases if we remove in this way:

1. If the number of examples we consider to remove is smaller than the number of *hard-to-learn* examples (top-left region in Figure 5.1), the examples we will remove are almost all *hard-to-learn* ones.

2. If the number of examples we consider to remove is close or larger than the number of *hard-to-learn* examples, the examples in the golden region (if their **variability** are large, they should be considered as *ambiguous* examples) and *easy-to-learn* examples are removed alternately after *hard-to-learn* examples are all removed.

Examples fall in golden region in Figure 5.1 are hard to distinguished as *hard-to-learn* or *easy-to-learn* examples. Thus, in addition to removing a small number of examples, we may also try removing relatively larger amount of examples. The latter case results in the removal of a small amount of *easy-to-learn* examples as described in case 2. This is fine as long as we keep a certain amount of *easy-to-learn* examples in the pool [19].

After obtaining the pre-selected set of sentences, we utilize the Latent Dirichlet Allocation (LDA) algorithm to cluster the pre-selected set of sentences into c clusters, which ensures the

higher diversity of the final subsampled training set. We use the same method and setting mentioned in §5.3.2 to train an n-gram model on a pre-sampled heldout set with the same number of tokens used in the final RNN model training. With the pre-trained n-gram model, we calculate the perplexities and selection probabilities of sentences in the pre-selected set.

We sample sentences in each cluster with the calculated selection probabilities, and the number of sentences we sample in each cluster is determined by (5.2). Finally, we train an RNN model on the subsampled training set, and we apply the corrective weights defined in (5.3) during the model training. The RNN model we train on the subsampled set has the same architecture with the RNN model we train on the initial randomly sampled set to build the Dataseet Map.

5.4 Experiments

In this section, we experimentally evaluate the effectiveness of our two subsampling methods. We first give some introductions of the dataset details (§5.4.1), followed by our experimental settings (§5.4.2), and finally we show the comparison results of our methods and the baselines (§5.4.3).

5.4.1 Dataset Details

We test our two subsampling methods and baseline methods by subsampling from the Wikitext-103 corpus [48] and the One Billion Word Benchmark corpus [81], and training sentence-level RNN models on the subsampled datasets. When processing Wikitext-103 corpus, header of each topic was removed and the corpus was parsed to create individual sentences. We shuffled the sentences and split them into initial_training, validation and testing sets. The validation and testing sets contained 250k tokens. The initial_training set had about 99 million tokens, and the final training set were subsampled from the initial_training set. For Billion Word experiments, we subsampled a 500 million token set from the released training set, and used it as the initial_training set. The

validation and testing sets were randomly sampled from the released heldout set, each with 250k tokens in it. When building the Dataset Maps in our second subsampling method (CLISIER, §5.3.4), the 10 million token sets was sampled from the initial_training sets.

RNN models models were trained on 500 thousand, 1 million and 2 million token training sets. For CLIS (§5.3.2), these training sets were subsampled from the initial_training set directly; while in CLISIER (§5.3.4), they were subsampled from the pre-selected set. Rare words were replaced with <unk> tokens during dataset pre-processing.

5.4.2 Experimental Settings

To calculate the selection probabilities of sequences, we use the method and setting mentioned in §5.3.2 to train an n-gram model on a pre-sampled heldout set with the same number of tokens used in the final RNN model training. The constant parameters α and τ in the sampling distribution (5.1) are chosen based on the evaluation results of validation set in the experiments, and β is set to be 1 for convenient comparison with the baseline results.

We use a two-layer long short-term memory (LSTM) neural network to build the RNN model, and the dimensions of hidden and embedding layers are set to be 200. We choose Adam optimizer [82] with default settings to train the models for 10 epochs with a batch size of 12. For testing CLISIER, the LSTM neural network we train on the subsampled training set has the same architecture with the one we train on the initial randomly sampled set to build the Dataset Map [19]. When clustering the randomly selected 10M token sets sampled from Wikitext-103 and the One Billion Word Benchmark corpus, the number of clusters we set is 20 as the size of this set is smaller than the size of initial training set.

Baselines We have two baselines. The random baseline is training the LSTM neural network on the set randomly sampled from the corpus. We also compare our methods with the existing work

on selecting RNN language model training data using an importance sampling (IS) scheme [18].

5.4.3 Results and Analysis

In Table 5.1, we summarize the model performances of RNN models trained on the sets sampled from Wikitext-103 using CLIS (§5.3.2) and the baseline methods. All the experiments are repeated 3 times with different random seeds, and we report the mean perplexity and the standard deviation

Table 5.1: Results of CLIS and Baseline methods on Wikitext-103 corpus. CLIS outperforms the Baseline methods, and achieves lowest ppl when the number of clusters is set to 60.

Tokens	**Method**	$(\alpha, \tau)^{\ddagger}$	**Validation ppl**	**Testing ppl**
500k	Random	-	582.92±4.17	576.68±4.11
	IS [18]	(1,1)	552.58±4.08	544.71±3.81
	$\text{CLIS}_{7^{\dagger}}$	(2,1)	551.57±3.79	542.70±4.40
	CLIS_{20}	(2,1)	549.63±5.90	540.92±5.87
	CLIS_{60}	(1,2)	548.84±2.62	540.23±2.78
	CLIS_{100}	(2,1)	549.96±1.82	542.49±1.73
1M	Random	-	447.03±1.82	440.04±1.94
	IS	(0.5,1)	416.27±2.37	412.52±2.13
	CLIS_{7}	(0.5,1)	410.74±2.70	406.50±2.59
	CLIS_{20}	(0.5,1)	411.24±4.63	405.81±4.08
	CLIS_{60}	(2,1)	410.66±1.41	405.39±1.22
	CLIS_{100}	(1,1)	412.98±1.66	408.10±1.54
2M	Random	-	341.59±2.83	337.80±2.94
	IS	(2,1)	298.30±1.54	295.22±2.15
	CLIS_{7}	(2,1)	296.13±1.07	293.28±1.08
	CLIS_{20}	(1,1)	294.51±5.02	291.69±5.08
	CLIS_{60}	(2,1)	293.45±2.75	290.37±3.02
	CLIS_{100}	(1,1)	295.89±2.25	291.81±2.39

† Number of clusters
‡ Constant parameters defined in (5.1)

of perplexity for validation and testing sets. RNN models trained using our CLIS method outperforms those trained on the sets sampled using baseline methods, and the models achieve the best performance when the number of clusters is set to be 60 during sentences clustering.

RNN models trained with the CLIS subsampling method all obtained lower average perplexities than those trained with baselines, and the CLIS_{60} method produced the largest reductions in average testing perplexity of 39.51 and 5.49 compared to the Random and IS baselines, respectively. As expected, clustering the original corpus helps improve the model performance since sampling sentences from different clusters makes the training set more diverse and sentences in different topics or with different structures are included in it. With larger number of clusters, our evaluations results are more stable as the standard deviations of perplexity are usually smaller than the cases using relatively fewer clusters.

Table 5.2: Results of CLISIER and Baseline methods on Wikitext-103 corpus. CLISIER outperforms the Baseline methods. CLISIER achieves lowest ppl when 20% of sequences are removed from the randomly selected set.

Tokens	**Method**	$(\alpha, \tau)^{\ddagger}$	**Validation ppl**	**Testing ppl**
500k	Random	-	582.92±4.17	576.68±4.11
	IS [18]	(1,1)	552.58±4.08	544.71±3.81
	$\text{CLISIER}_{10\%}{}^{\diamond}$	(2,1)	546.32±2.85	538.89±2.54
	$\text{CLISIER}_{20\%}$	(2,1)	544.76±2.97	537.21±3.15
	$\text{CLISIER}_{40\%}$	(1,2)	547.43±3.39	540.52±3.46
1M	Random	-	447.03±1.82	440.04±1.94
	IS	(0.5,1)	416.27±2.37	412.52±2.13
	$\text{CLISIER}_{10\%}$	(0.5,1)	411.32±2.10	404.17±2.27
	$\text{CLISIER}_{20\%}$	(0.5,1)	406.54±1.79	401.61±1.43
	$\text{CLISIER}_{40\%}$	(0.5,1)	414.97±2.03	409.75±1.91
2M	Random	-	341.59±2.83	337.80±2.94
	IS	(2,1)	298.30±1.54	295.22±2.15
	$\text{CLISIER}_{10\%}$	(2,1)	290.40±3.42	287.94±3.30
	$\text{CLISIER}_{20\%}$	(1,1)	289.72±3.09	287.05±3.17
	$\text{CLISIER}_{40\%}$	(1,1)	294.26±3.34	291.43±3.69

$^{\diamond}$ Percentage of sequences removed

‡ Constant parameters defined in (5.1)

The results of training RNN language models on the text sets sampled from Wikitext-103 using the CLISIER (§5.3.4) and the baseline methods are reported in Table 5.2. With a number of ineffi-

ciency training examples removed, RNN models trained using the CLISIER methods yield a much lower model perplexity as compared to the models trained on the similar sized sets subsampled using the baseline methods. The CLISIER$_{20\%}$ method produced the most significant reductions in average testing perplexity of 42.88 and 8.86 compared to the Random and IS baselines.

In the work presented by Karamcheti et al. [20], they find that the more *hard-to-learn* examples they remove from the entire data pool, the better the performance of active learning methods. And their models achieve the best results when removing 50% of the full pool. While in our case, the RNN models yield the best results when we remove only 20% of the training examples from the dataset pool. One of the possible reasons for this interesting observation is there are fewer *hard-to-learn* examples in our dataset pool, and removing too many examples result in the removal of a proportion of *easy-to-learn* and *ambiguous* examples, which will degrade the performance of the trained models. For example, in Table 5.2, we see that the CLISIER$_{40\%}$ method produced relatively higher mean perplexities compared with using the methods CLISIER$_{10\%}$ and CLISIER$_{20\%}$. This is possibly because after all the *hard-to-learn* examples are removed, *easy-to-learn* examples and examples in the golden region / *ambiguous* region (Figure 5.1) are removed alternately. Although removing a small number of *easy-to-learn* examples usually will not degrade the performance of models, finding the optimal balance of *easy-to-learn* and *ambiguous* examples in low data regimes is still an open problem.

Instead of applying CLIS approach on the One Billion Word Benchmark corpus, we test the effectiveness of CLISIER approach working on it since clustering the randomly selected 10M token set will save more computational resources than clustering the initial 500 million token set. The results of training RNN language models on the text sets sampled from the One Billion Word Benchmark corpus using the CLISIER (§5.3.4) and the baseline methods are presented in Table 5.3. Still, the CLISIER methods outperformed the baseline methods, and the CLISIER$_{20\%}$

method achieved the most significant reductions in average testing perplexity of 77.30 and 14.37 compared to the Random baseline and the IS baseline, respectively.

Table 5.3: Results of CLISIER and Baseline methods on the One Billion Word Benchmark corpus. CLISIER outperforms the Baseline methods. CLISIER achieves lowest ppl when 20% of sequences are removed from the randomly selected set for the 1M token and 2M token cases, and achieves the lowest ppl when 40% of sequences are removed from the random set for the 500k token case.

Tokens	**Method**	$(\alpha, \tau)^{\ddagger}$	**Validation ppl**	**Testing ppl**
500k	Random	-	650.62±4.85	651.39±4.43
	IS [18]	(2,1)	573.25±3.78	573.64±3.51
	CLISIER$_{10\%}{}^{\diamond}$	(2,1)	562.90±3.05	562.23±2.82
	CLISIER$_{20\%}$	(0.5,1)	560.67±2.96	559.58±3.46
	CLISIER$_{40\%}$	(2,1)	559.37±3.53	558.94±3.22
1M	Random	-	489.29±2.16	490.07±2.65
	IS	(1,1)	431.34±2.75	431.79±3.04
	CLISIER$_{10\%}$	(4,1)	419.25±2.11	418.55±2.48
	CLISIER$_{20\%}$	(4,1)	415.17±2.97	414.62±2.35
	CLISIER$_{40\%}$	(2,1)	416.35±3.41	417.03±2.92
2M	Random	-	358.91±2.26	359.88±2.14
	IS	(1,2)	307.61±1.02	307.12±1.89
	CLISIER$_{10\%}$	(1,1)	297.47±2.15	296.67±1.93
	CLISIER$_{20\%}$	(1,1)	295.74±2.31	295.24±2.55
	CLISIER$_{40\%}$	(1,1)	297.69±1.82	296.98±2.23

$^{\diamond}$ Percentage of sequences removed
‡ Constant parameters defined in (5.1)

5.5 Conclusion

In this Chapter, we propose two training data subsampling methods for efficient language modeling: Clustering-based Importance Subsampling (CLIS) and Clustering-based Importance Subsampling with Inefficiency Examples Removed (CLISIER). By utilizing the proposed subsampling methods, we can more efficiently learn sentence-level language models for large corpora by training them on a set of sentences subsampled from the original corpus. We experimentally showed

the effectiveness of the two proposed methods at selecting training set for RNN language models.

Both of CLIS and CLISIER methods apply the clustering technique and the importance sampling technique to make the selected training set as diverse and informative as possible. We also re-weight the corrective weights used in the model training to reflect the effect of cluster size. The CLIS method performs faster at sentence selecting owe to n-gram language models' rapid training and query time. However, even with importance sampling, we still have the chance to sample a number of inefficiency sentences. The CLISIER method has a pre-selection procedure, in which we first uniformly sample a set of sentences from the original corpus and remove a large amount of inefficiency sentences according to Dataset Maps. From the experimental results, CLISIER method produces relatively larger reductions in average perplexity compared to the baseline methods, with the cost of using more time and computational resources. We defer the exploration of a more efficient pre-selection procedure for CLISIER to the future work.

CHAPTER 6
CONCLUSION AND FUTURE WORK

In this dissertation, we presented a framework for comparing accuracy of estimation methods in estimating F-measure and the variance, some effective modeling methods for ensemble RNN model training, and two efficient subsampling methods for sampling more diverse and informative text data for language modeling. We summarize the key findings of our work, and discuss some possible future works in this chapter.

In Chapter 2, we present a framework of comparative experiments, which is used to compare the variances computed by the Wong's estimation method [1] and the JVESR's estimation method [7], and compute the z-statistic for pairwise comparison. In JVESR's method, the correlation between two algorithms can be naturally incorporated by our proposed covariance formula when computing the z-statistic, while Wong's method assumes the independence of two algorithms, which doesn't hold in current experimental settings of Wong's method. The estimation of variance of F-measure in Wong's method is not totally correct, and we describe the correct delta method. Our experiments on three datasets chosen from the UCI data repository [31] demonstrated that the proposed framework is effective in comparing the accuracy of the two underlying estimation methods, and JVESR's method is more accurate in variance estimation and z-statistic computation when we incorporate a covariance estimate naturally based on Appendix A.

In our proposed framework of comparative experiments, we only compared two algorithms on the single dataset. However, machine learning algorithms are generally tested on multiple datasets to determine which one has the best performance. For this reason, we can generate the framework to compare multiple algorithms on multiple datasets in the future. Actually, we have generalized

the statistical method (Appendix A) to compute the covariance matrix of F-measures for multiple algorithms, and Wong [1] also present the estimation method for computing the F-measure and variance on multiple datasets. These will bring convenience to the generalization work of our framework. For the seek of independence of the algorithms in Wong's method, we can use multiple non-overlapping subsets to evaluate the algorithms in the future.

In Chapter 3, we described the definition of the probability of sequences, and the evaluation method of the statistical language modeling. We introduced the n-gram language models and the recurrent neural network language models, and made some comparisons between these two types of language models.

In Chapter 4, we proposed some effective modeling methods for ensemble training. The first novel algorithm we presented is Batch-wise Mixture Modeling for Ensembles (BaMME), in which we effectively navigate the ensemble model to focus on certain part of the text data by weighting the loss, and apply smoothing technique to avoid getting extreme weights during the training process. The loss function for each ensemble model is minimized in the small-batch regime to avoid the degradation in the quality of the model. We experiment on the two benchmark corpora: Penn Treebank (PTB) and Wikitext-2 (WT2), and the results show that BaMME is effective in improving the performance of ensemble models compared with the baseline method.

In BaMME, we soft-clustering the original batchified text data by assigning mixture weights to each batch of text data. To further explore how the soft-clustering technique applied during the training contribute to the overall improvement of model performance, we developed two methods –Batch-wise Pre-selection of Ensembles and Clustering-based Selection of Ensembles to select the ensemble model during the training. Batch-wise Pre-selection of Ensembles aims to navigate each ensemble better focus on part of the data without using weighting of losses. Clustering-based Selection of Ensembles method produces much better results on the synthetic dataset than baseline

method since after clustering, the text data in each cluster are more semantically related with each other. However, real datasets are generally hard to perfectly clustered using hard clustering algorithm. We would like to incorporate soft-clustering technique in Clustering-based Selection of Ensembles method in the future work to better train the real datasets.

In Chapter 5, we presented two novel subsampling methods called CLIS and CLISIER for subsampling text data for efficient RNN language modeling. Both of these two methods apply the clustering technique and the importance sampling technique to make the selected training set as diverse and informative as possible. In CLIS method, we first apply an unsupervised clustering algorithm to cluster the original corpus. Then, we utilize n-gram model to socre the sentences in the initial dataset, and compute the selection probabilities of sentences, followed by sampling within each cluster with sample size in proportional to the square root of cluster size. Finally, we train a RNN language model with the pre-claculated corrective weights. We re-weight the corrective weights applied during the training to reflect the effect of cluster size.

For the CLISIER method, we incorporate a pre-selection procedure, where we randomly sample a text set from the original corpus, and remove a certain amount of inefficiency sentences from the sampled set. To remove inefficiency sentences, we train a RNN model on the sampled set, and calculate the mean log-perplexity and variability over epochs for each sentence in the set, followed by plotting the Dataset Map to characterize and diagnose the underlying set. We calculate and sort the quotient of the variability and mean log-perplexity of sentence in initial randomly sampled set, and remove examples with low quotient values to get the pre-selected text set. The remaining procedures are similar as those in the CLIS method, except that we only need to cluster the pre-selected set.

Our experiments demonstrate that both of these two subsampling methods are efficient in data selection for neural language models, and the performance of trained RNN models on the subsam-

pled set are better than those trained on the set sampled using baseline methods. The CLISIER method produces relatively larger reductions in perplexities compared to the baseline methods, with the cost of using more computational resources to pre-train a RNN model in the pre-selection step. We defer the development of a more efficient pre-selection method for CLISIER to the future work.

REFERENCES

[1] T. Wong, "Linear approximation of F-measure for the performance evaluation of classification algorithms on imbalanced data sets," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[2] C. J. van Rijsbergen, "Information retrieval," *Butterworths, 2nd edition*, 1979.

[3] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and F-score, with implication for evaluation," *Advances in Information Retrieval, ECIR*, pp. 345–359, 2005.

[4] J. Uttley, "Power analysis, sample size, and assessment of statistical assumptions—improving the evidential value of lighting research," *LEUKOS*, vol. 15, no. 2-3, pp. 143–162, 2019.

[5] Y. Wang, J. Li, Y. Li, R. Wang, and X. Yang, "Confidence interval for F_1 measure of algorithm performance based on blocked 3×2 cross-validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 651–659, 2015.

[6] K. Takahashi, K. Yamamoto, A. Kuchiba, and T. Koyama, "Confidence interval for micro-averaged F_1 and macro-averaged F_1 scores," *Applied Intelligence*, 2021.

[7] W. Jiang, "Statistical formulas for F measures," *ArXiv*, abs/2012.14894, 2020.

[8] S. Janson and J. Vegelius, "Measures of ecological association," *Oecologia*, vol. 49, pp. 371–376, 1981.

[9] R. C. Elston, S. R. Schroeder, and J. Rohjan, "Measures of observer agreement when binomial data are collected in free operant situations," *Journal of Behavioral Assessment*, vol. 4, pp. 299–310, 1982.

[10] K. Knight, "Decoding complexity in word-replacement translation models," *Computational Linguistics*, vol. 25, no. 4, pp. 607–615, 1999.

[11] V. Chunwijitra and C. Wutiwiwatchai, "Classification-based spoken text selection for LVCSR language modeling," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 24, 2017.

[12] Y. Yang, C. Malaviya, J. Fernandez, S. Swayamdipta, R. Le Bras, J.-P. Wang, C. Bhagavatula, Y. Choi, and D. Downey, "Generative data augmentation for commonsense reasoning," *Association for Computational Linguistics (ACL)*, pp. 1008–1025, 2020.

[13] S. Shen, Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu, "Minimum risk training for neural machine translation," *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1683–1692, 2016.

[14] A. Rudnicky and W. Xu, "An agenda-based dialog management architecture for spoken language systems," *IEEE Automatic Speech Recognition and Understanding Workshop*, vol. 13, 1999.

[15] L. Shang, Z. Lu, and H. Li, "Neural responding machine for short-text conversation," *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1577–1586, 2015.

[16] D. Marcu, "From discourse structures to text summaries," *Proceedings of the workshop on Intelligent Scalable Text Summarization*, 1997.

[17] A. See, P. Liu, and C. Manning, "Get to the point: Summarization with pointer- generator networks," *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1073–1083, 2017.

[18] J. Fernandez and D. Downey, "Sampling informative training data for RNN language models," *Proceedings of Association for Computational Linguistics (ACL) 2018, Student Research Workshop*, pp. 9–13, 2018.

[19] S. Swayamdipta, R. Schwartz, N. Lourie, Y. Wang, H. Hajishirzi, N. A. Smith, and Y. Choi, "Dataset cartography: Mapping and diagnosing datasets with training dynamics," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9275–9293, 2020.

[20] S. Karamcheti, R. Krishna, F. Li, and C. D. Manning, "Mind your outliers! investigating the negative impact of outliers on active learning for visual question answering," *CoRR*, abs/2107.02331, 2021.

[21] G. Hripcsak and A. S. Rothschild, "Agreement, the f-measure, and reliability in information retrieval," *Journal of the American Medical Informatics Association*, vol. 12, no. 3, pp. 296–298, 2005.

[22] A. Mosquera and P. Moreda, "The use of metrics for measuring informality levels in web 2.0 texts," *Proceedings of the 8th Brazilian Symposium in Information and Human Language Technology*, pp. 184–188, 2011.

[23] S. M. Beitzel, "On understanding and classifying web queries," 2006.

[24] Y. Wang and J. Li, "Credible intervals for precision and recall based on a k-fold cross-validated beta distribution," *Neural Computation*, vol. 28, no. 8, pp. 1694–1722, 2016.

[25] A. Vaart, "Asymptotic statistics (cambridge series in statistical and probabilistic mathematics)," *Cambridge: Cambridge University Press*, 1998.

[26] A. Vaart and J. Wellner, "Weak convergence and empirical processes," *Springer, New York*, 1996.

[27] W. Jiang and Y. Zhao, "On asymptotic distributions and confidence intervals for LIFT measures in data mining," *Journal of the American Statistical Association*, vol. 110, no. 512, pp. 1717–1725, 2015.

[28] W. J. Nash, T. L. Sellers, S. R. Talbot, A. J. Cawthorn, and W. B. Ford, "The population biology of abalone (_haliotis_species) in tasmania. i. blacklip abalone (_h. rubra_) from the north coast and islands of bass strait," *Sea Fisheries Division, Technical Report*, no. 48, 1994.

[29] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems, Elsevier*, vol. 47, no. 4, pp. 547–553, 2009.

[30] M. Sikora and Ł. Wróbel, "Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines," *Archives of Mining Sciences*, vol. 55, no. 1, pp. 91–114, 2010.

[31] M. Lichman, "UCI machine learning repository, http://archive.ics.uci.edu/ml, University of California, Irvine, School of Information and Computer Sciences," 2013.

[32] G. J. Lidstone, "Note on the general case of the Bayes–Laplace formula for inductive or a posteriori probabilities," *Transactions of the Faculty of Actuaries*, vol. 8, pp. 182–192, 1920.

[33] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[34] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 181–184, 1995.

[35] W. A. Gale, "Good-turing smoothing without tears," *Journal of Quantitative Linguistics*, vol. 2, 1995.

[36] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 400–401, 1987.

[37] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *JMLR*, pp. 1137–1155, 2003.

[38] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," *INTERSPEECH*, pp. 1045–1048, 2010.

[39] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," *Interspeech*, pp. 194–197, 2012.

[40] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[42] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," *ArXiv*, abs/1406.1078, 2014.

[43] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv*, abs/1412.3555, 2014.

[44] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, abs/1412.6980, 2015.

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[47] J. Juraska, P. Karagiannis, K. K. Bowden, and M. A. Walker, "A deep ensemble model with slot alignment for sequence-to-sequence natural language generation," *Proceedings of NAACL-HLT 2018*, pp. 152–162, 2018.

[48] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *CoRR*, abs/1609.07843, 2016.

[49] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn, "One billion word benchmark for measuring progress in statistical language modeling," *CoRR*, abs/1312.3005, 2013.

[50] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2000.

[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *CoRR*, abs/1810.04805, 2018.

[52] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *CoRR*, abs/1907.11692, 2019.

[53] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: adapt language models to domains and tasks," *CoRR*, abs/2004.10964, 2020.

[54] A. Aniol, M. Pietron, and J. Duda, "Ensemble approach for natural language question answering problem," *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 180–183, 2019.

[55] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016.

[56] A. M. Webb, C. Reynolds, D.-A. Iliescu, H. Reeve, M. Luján, and G. Brown, "Joint training of neural network ensembles," *CoRR*, abs/1902.04422, 2019.

[57] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *The International Conference on Learning Representations (ICLR)*, 2017.

[58] H. Namkoong, A. Sinha, S. Yadlowsky, and J. C. Duchi, "Adaptive sampling probabilities for non-smooth optimization," *Proceedings of the 34th International Conference on Machine Learning*, vol. PMLR 70, pp. 2574–2583, 2017.

[59] G. Bouchard, T. Trouillon, J. Perez, and A. Gaidon, "Online learning to sample," *CoRR*, abs/1506.09016, 2015.

[60] S. U. Stich, A. Raj, and M. Jaggi, "Safe adaptive importance sampling," *Advances in Neural Information Processing Systems 30*, pp. 4384–4394, 2017.

[61] Z. Borsos, S. Curi, K. Y. Levy, and A. Krause, "Online variance reduction with mixtures," *Proceedings of the International Conference on Machine learning (ICML)*, 2019.

[62] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[63] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, pp. 223–311, 2018.

[64] D. Lei, Z. Sun, Y. Xiao, and W. Y. Wang, "Implicit regularization of stochastic gradient descent in natural language processing: Observations and implications," *CoRR*, abs/1811.00659, 2018.

[65] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *CoRR*, abs/1708.02182, 2017.

[66] Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen, "Breaking the softmax bottleneck: A high-rank RNN language model," *Proceedings of ICLR 2018*, 2018.

[67] J. MacQueen, "Some methods for classification and analysis of multivariate observations," *Berkeley Symposium on Mathematical Statistics and Probability*, vol. 5.1, pp. 281–297, 1967.

[68] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan,

R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, abs/2005.14165, 2020.

[69] Y. Tay, M. Dehghani, V. Aribandi, J. P. Gupta, P. Pham, Z. Qin, D. Bahri, D.-C. Juan, and D. Metzler, "Omninet: Omnidirectional representations from transformers," *CoRR*, abs/2103.01075, 2021.

[70] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *CoRR*, abs/1909.08053, 2019.

[71] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," *ArXiv*, abs/1803.00942, 2018.

[72] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, "Variance reduction in sgd by distributed importance sampling," *CoRR*, abs/1511.06481, 2015.

[73] S. Gururangan, S. Swayamdipta, O. Levy, R. Schwartz, S. Bowman, and N. A. Smith, "Annotation artifacts in natural language inference data," *Association for Computational Linguistics (ACL)*, pp. 107–112, 2018.

[74] H.-S. Chang, E. Learned-Miller, and A. McCallum, "Active bias: Training more accurate neural networks by emphasizing high variance samples," *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1002–1012, 2017.

[75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[76] D. Zhao, J. He, and J. Liu, "An improved lda algorithm for text classification," *2014 International Conference on Information Science, Electronics and Electrical Engineering*, pp. 217–221, 2014.

[77] A. Stolcke, "Srilm-an extensible language modeling toolkit," *Seventh International Conference on Spoken Language Processing*, 2002.

[78] K. Heafield, "KenLM: Faster and smaller language model queries," *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pp. 187–197, 2011.

[79] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," *Association for Computational Linguistics (ACL)*, pp. 1112–1122, 2018.

[80] A. Shrivastava, A. Gupta, and R. Girshick, "Training region-based object detectors with online hard example mining," *CVPR*, pp. 761–769, 2016.

[81] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robin- son, "One billion word benchmark for measuring progress in statistical language modeling," *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[82] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, abs/1412.6980, 2014.

[83] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *Advances in Neural Information Processing Systems*, pp. 1019–1027, 2016.

APPENDIX A

COMPUTATION OF COVARIANCE OF TWO *F*-MEASURES

[1] When computing Var($\bar{f}_1 - \bar{f}_2$) in equation (2.8), we need to know the covariance of $\bar{f}_1$ and $\bar{f}_2$. In JVESR's method [7], $\bar{f}_i$ is computed by $\tau_n(\lambda)$ in (2.4) with $\lambda = 2$. We rewrite $\tau_n(\lambda)$ as

$$\tau_{ni}(\lambda) \triangleq \frac{\lambda E_n(ZL_i)}{\lambda E_n(ZL_i) + E_n(L_i(1-Z)) + E_n(Z(1-L_i))} \tag{A.1}$$

where Z, L_i are the same random variables defined in §2.2.3, and $E_n f(Z, L_i) = n^{-1} \sum_{k=1}^{n} f(Z_k, L_{ki})$.

Let's flip the numerator and denominator of $\tau_{ni}(\lambda)$, and denote $\kappa_{ni}(\lambda)$ as

$$\kappa_{ni}(\lambda) \triangleq \tau_{ni}^{-1}(\lambda) - 1 = \frac{E_n(L_i(1-Z) + Z(1-L_i))}{\lambda E_n(ZL_i)} \tag{A.2}$$

In order to apply delta method, we need to differentiate $\kappa_{ni}(\lambda)$,

$$\begin{aligned} \mathrm{d}\kappa_{ni}(\lambda) &= \frac{\mathrm{d}E_n(L_i(1-Z) + Z(1-L_i))}{\lambda E_n(ZL_i)} - \frac{E_n(L_i(1-Z) + Z(1-L_i)) \cdot \mathrm{d}E_n(ZL_i)}{\lambda E_n^2(ZL_i)} \\ &\approx \frac{\mathrm{d}E_n(L_i(1-Z) + Z(1-L_i)) - \kappa_{ni}(\lambda) \cdot \mathrm{d}E_n(ZL_i)}{\lambda E(ZL_i)} \end{aligned} \tag{A.3}$$

where $E = E_\infty$.

[1] This material is adapted from a private communication with W. Jiang in 03/2021.

With (A.3) and delta method, we have

$$\begin{aligned}\mathrm{Cov}\big(\kappa_{n1}(\lambda),\kappa_{n2}(\lambda)\big) = &\frac{\mathrm{Cov}(L_1(1-Z)+Z(1-L_1),L_2(1-Z)+Z(1-L_2))}{n\lambda^2E(ZL_1)E(ZL_2)}\\ &-\frac{\kappa_{n1}(\lambda)\cdot\mathrm{Cov}(ZL_1,L_2(1-Z)+Z(1-L_2))}{n\lambda^2E(ZL_1)E(ZL_2)}\\ &-\frac{\kappa_{n2}(\lambda)\cdot\mathrm{Cov}(ZL_2,L_1(1-Z)+Z(1-L_1))}{n\lambda^2E(ZL_1)E(ZL_2)}\\ &+\frac{\kappa_{n1}(\lambda)\kappa_{n2}(\lambda)\cdot\mathrm{Cov}(ZL_1,ZL_2)}{n\lambda^2E(ZL_1)E(ZL_2)}\end{aligned} \tag{A.4}$$

$\kappa_{ni}(\lambda)$ in (A.2) can also be differentiated as $\mathrm{d}\kappa_{ni}(\lambda) \approx -\tau_{ni}^{-2}(\lambda)\mathrm{d}\tau_{ni}(\lambda)$, and thus we have $\mathrm{d}\tau_{ni}(\lambda) \approx -\tau_{ni}^{2}(\lambda)\mathrm{d}\kappa_{ni}(\lambda)$. By applying delta method again, the covariance of $\bar{f}_1$ and $\bar{f}_2$ can be calculated as

$$\begin{aligned}\mathrm{Cov}(\bar{f}_1,\bar{f}_2) &= \mathrm{Cov}\big(\tau_{n1}(\lambda),\tau_{n2}(\lambda)\big)\big|_{\lambda=2}\\ &\approx \tau_{n1}^2(\lambda)\tau_{n2}^2(\lambda)\cdot\mathrm{Cov}\big(\kappa_{n1}(\lambda),\kappa_{n2}(\lambda)\big)\big|_{\lambda=2}\end{aligned} \tag{A.5}$$

With the computed covariance of $\bar{f}_1$ and $\bar{f}_2$, we can use (2.8) to compute the variance of $\bar{f}_1 - \bar{f}_2$ and conduct pairwise comparison of two algorithms.

APPENDIX B

MORE DETAILS OF ALGORITHM 1

To minimize batch loss (4.4), we will show that it's enough to optimize (4.5). Consider the following:

$$
\begin{aligned}
l(x_i;\theta_i) &= -\frac{1}{|\mathcal{C}_i|}\sum_{m=1}^{|\mathcal{C}_i|}\log p(x_{i,m};\theta_i) \\
&= -\frac{1}{|\mathcal{C}_i|}\log\prod_{m=1}^{|\mathcal{C}_i|} p(x_{i,m};\theta_i) \\
&= -\frac{1}{|\mathcal{C}_i|}\log\sum_{k=1}^{K} v_{i,k}\frac{\prod_{m=1}^{|\mathcal{C}_i|} p(x_{i,m};\theta_{i,k})}{v_{i,k}} \qquad \text{(B.1)} \\
&\leq -\frac{1}{|\mathcal{C}_i|}\sum_{k=1}^{K} v_{i,k}\log\frac{\prod_{m=1}^{|\mathcal{C}_i|} p(x_{i,m};\theta_{i,k})}{v_{i,k}} \qquad \text{(B.2)} \\
&= -\frac{1}{|\mathcal{C}_i|}\sum_{k=1}^{K} v_{i,k}\Big(\sum_{m=1}^{|\mathcal{C}_i|}\log p(x_{i,m};\theta_{i,k}) - \log v_{i,k}\Big) \\
&= \sum_{k=1}^{K} v_{i,k} l(x_i;\theta_{i,k}) + \frac{1}{|\mathcal{C}_i|}\sum_{k=1}^{K} v_{i,k}\log v_{i,k} \qquad \text{(B.3)}
\end{aligned}
$$

In (B.2), we used Jensen's inequality. (B.3) is an upper bound of the batch loss $l(x_i;\theta_i)$. It's enough for us to make the inequality in (B.2) hold with equality at the particular values of parameters $\theta_{i,k}$ that minimize the weighted loss in (B.3).

To make the inequality in (B.2) hold with equality, it's sufficient to set the fraction term in (B.1) to be some constant that does not depend on the ensemble index k. With the fact that $\sum_{k=1}^{K} v_{i,k} = 1$,

batch weight $v_{i,k}$ should be set as

$$\begin{aligned} v_{i,k} &= \frac{\prod_{m=1}^{|\mathcal{C}_i|} p(x_{i,m}; \theta_{i,k})}{\sum_{k=1}^{K} \prod_{m=1}^{|\mathcal{C}_i|} p(x_{i,m}; \theta_{i,k})} \\ &= \frac{\pi_{i,k} \phi_k(x_i)}{\sum_{k=1}^{K} \pi_{i,k} \phi_k(x_i)} \end{aligned}$$

where $\pi_{i,k}$ is the mixture weight to be determined later and $\phi_k(x_i)$ is the likelihood of batch data x_i and it is computed by the kth ensemble element.

To derive the mixture weight $w_{i,k}$, let's construct the Lagrangian

$$\mathcal{L}(\pi) = -\frac{1}{|\mathcal{C}_i|} \sum_{k=1}^{K} v_{i,k} \log \pi_{i,k} \phi_k(x_i) + \beta \Big(\sum_{k=1}^{K} \pi_{i,k} - 1 \Big)$$

Taking derivatives, setting them to zeros and solving, we get

$$\pi_{i,k} = \frac{1}{|\mathcal{C}_i|} \frac{v_{i,k}}{\beta}$$

Considering the constraint that $\sum_k \pi_{i,k} = 1$, we have $\beta = \sum_k v_{i,k}/|\mathcal{C}_i| = 1/|\mathcal{C}_i|$. Finally, to update parameters $\pi_{i,k}$, we simply set them to be $\pi_{i,k} = v_{i,k}$.

APPENDIX C

EXPERIMENTAL SETTING AND HYPER-PARAMETERS

The hyper-parameters used in language modeling experiments is shown in Table C.1 below.

Table C.1: Hyper-parameters used in modeling.

Hyper-parameter	**PTB**	**WT2 (Baseline)**	**WT2 (Mixture)**
Batch size	12	15	12
Small batch size	-	5	2
Learning rate	20	15	
Embedding size	27	42	
Sequence length	65	70	
LSTM hidden sizes	[37, 30]	[60, 48]	
Word-level VT-dropout*	0.10	0.10	
Embedding VT-dropout	0.55	0.40	
Hidden state VT-dropout	0.20	0.225	
Recurrent weight dropout	0.50	0.50	

* VT-dropout abbreviates variational dropout ([83]).

The hyper-parameters used for batch weight smoothing in mixture modeling is summarized in the Table C.2 below.

Table C.2: Hyper-parameters used for batch weight smoothing.

Hyper-parameter	**PTB**	**WT2**
Decaying rate η	0.001	0.001
Initial smoothing threshold	1/3	0.31

APPENDIX D

EXAMPLES TO ILLUSTRATE BATCH WEIGHT SMOOTHING

We will use two examples to illustrate how to employ Algorithm 2 to smooth batch weight. Assume we're modeling a mixture model with three ensemble elements, and the smoothing threshold is 0.15 in the current epoch. In the first example, the batch weights before smoothing are $0.01, 1.0 \times 10^{-8}$ and $0.99 - 10^{-8}$, respectively. The operation and values of batch weights in each step are listed below.

Table D.1: Example 1 of batch weight smoothing.

Step	Operation	Batch Weight Values
0	Initial values $\mathbf{w} = (w_1, w_2, w_3)$	$(0.01, 1.0 \times 10^{-8}$ and $0.99 - 10^{-8})$
1	$\mu = 0.15 - 10^{-8}, \Delta = \{3\}, \lambda = w_3$ $w_2 = w_2 + (0.15 - 10^{-8}) \times w_3/w_3$ $w_3 = w_3 - (0.15 - 10^{-8}) \times w_3/w_3$	$(0.01, 0.15$ and $0.84)$
2	$\delta = 10^{-8}/(0.99 - 10^{-8}) = 1.01 \times 10^{-8}$ $w_2 = w_2 + 1.01 \times 10^{-8}$ $w_3 = w_3 - 1.01 \times 10^{-8}$	$(0.01, 0.15 + 1.01 \times 10^{-8}$ and $0.84 - 1.01 \times 10^{-8})$
3	$\mu = 0.14, \Delta = \{3\}, \lambda = w_3$ $w_1 = w_1 + 0.14 \times w_3/w_3$ $w_3 = w_3 - 0.14 \times w_3/w_3$	$(0.15, 0.15 + 1.01 \times 10^{-8}$ and $0.7 - 1.01 \times 10^{-8})$
4	$\delta = 0.01/(0.99 - 10^{-8}) = 0.0101$ $w_1 = w_1 + 0.0101$ $w_3 = w_3 - 0.0101$	$(0.1601, 0.15 + 1.01 \times 10^{-8}$ and $0.6899 - 1.01 \times 10^{-8})$

In our second example, the unsmoothed batch weights are $0.001, 0.354$ and 0.645, respectively. The operations and values of batch weights are listed in the table below.

Table D.2: Example 2 of batch weight smoothing.

Step	**Operation**	**Batch Weight Values**
0	Initial values $\mathbf{w} = (w_1, w_2, w_3)$	$(0.001, 0.354$ and $0.645)$
1	$\mu = 0.149, \Delta = \{2, 3\}, \lambda = 0.999$ $w_1 = w_1 + 0.149 \times w_2/0.999$ $w_2 = w_2 - 0.149 \times w_2/0.999$	$(0.0538, 0.3012$ and $0.645)$
2	$w_1 = w_1 + 0.149 \times w_3/0.999$ $w_3 = w_3 - 0.149 \times w_3/0.999$	$(0.15, 0.3012$ and $0.5488)$
3	$\delta = 0.001/0.645 = 0.0016$ $w_1 = w_1 + 0.0016$ $w_3 = w_3 - 0.0016$	$(0.1516, 0.3012$ and $0.5472)$