On the Softmax Bottleneck of Word-Level Recurrent Language Models

by

Dwarak Govind Parthiban

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for
Master of Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

For different input contexts (sequence of previous words), to predict the next word, a neural word-level language model outputs a probability distribution over all the words in the vocabulary using a softmax function. When the log of probability outputs for all such contexts are stacked together, the resulting matrix is a log probability matrix which can be denoted as $\mathbf{Q}_\theta$, where θ denotes the model parameters. When language modeling is formulated as a matrix factorization problem, the matrix to be factorized $\mathbf{Q}_\theta$ is expected to be high-rank as natural language is highly context-dependent. But existing softmax based word-level language models have a limitation of not being able to produce such matrices; this is known as the *softmax bottleneck*.

There are several works that attempted to overcome the limitations introduced by softmax bottleneck, such as the models that can produce high-rank $\mathbf{Q}_\theta$. During the process of reproducing the results of these works, we observed that the rank of $\mathbf{Q}_\theta$ does not always positively correlate with better performance (i.e., lower test perplexity). This puzzling observation triggered us to conduct a systematic investigation to check the influence of rank of $\mathbf{Q}_\theta$ on better performance of a language model.

We first introduce a new family of activation functions called the Generalized SigSoftmax (GSS). By controlling the parameters of GSS, we were able to construct language models that can produce $\mathbf{Q}_\theta$ with diverse ranks (i.e., low, medium, and high ranks). For models that use GSS with different parameters, we observe that rank does not have a strong positive correlation with perplexity on the test data, reinforcing the support of our initial observation. By inspecting the top-5 predictions made by different models for a selected set of input contexts, we observe that a high-rank $\mathbf{Q}_\theta$ does not guarantee a strong qualitative performance.

Then, we conduct experiments to check if there are any other additional benefits in having models that can produce high-rank $\mathbf{Q}_\theta$. We expose that $\mathbf{Q}_\theta$ rather suffers from the phenomenon of fast singular value decay. Additionally, we also propose an alternative metric to denote the rank of any matrix known as ϵ-effective rank, which can be useful to approximately quantify the singular value distribution when different values for ϵ are used.

We conclude by showing that it is the regularization which has played a positive role in the performance of these high-rank models in comparison to the chosen baselines, and there is no single model yet which truly gains improved expressiveness just because of breaking the softmax bottleneck.

Acknowledgements

First, I want to thank my primary supervisor Prof. Diana Inkpen for providing me this exciting research opportunity. I am immensely thankful to my co-supervisor Prof. Yongyi Mao who had guided me patiently from start till the finish. He was always available for any sort of discussion. I was always eager towards our weekly meeting which would be filled with some interesting discussions. Thank you so much Prof. Mao for giving me this wonderful and memorable experience. I consider myself lucky to have received a continued support from both of my supervisors. I must definitely thank Dr. Harry Guo because of whom I was able to get this opportunity of working with my supervisors. I am grateful to the examiners Prof. Nathalie Japkowicz and Prof. Burak Kantarci for accepting the invitation to serve as an examiner, for spending their time on reading this thesis and providing invaluable comments and suggestions, and for making themselves available for the oral defence on an earlier date despite their busy schedules.

The best part of working with two supervisors is that you get to be a part of two different labs. I always enjoyed the company of my lab mates and in particular Diana Lucaci, Ziqiao, Jelber, Ruba, Doaa, Masoumeh, Prasadith, Gerardo, Cheng, Ken, Guillaume, Boming, Runzhi, and Chenjie. We have always had long discussions on different topics ranging from food, culture, geo-politics, and job interviews, to of course research progress. Also, I had a very good company with Parth, Tahira, Gurpreet, Harish, and Varun during the course of my study.

The person who instilled into me the idea of pursuing research was Cibe, a fellow intern during my undergraduate final semester internship. Had I not met him, I am not sure whether I would have chosen this path. Thank you Cibe for your motivation. I want to express my gratitude to a list of friends, neighbours, college mates, college seniors, fellow interns, and co-workers who had helped, inspired, and also pushed me to explore further in Computer Science: Adithya Balasubramanian, Ajhay, Anand, Anirudh Challa, Barath Sriram, Bharath Santhanam, Chandrasekar, Dinesh, Dwarakesh, Ganesh Jawahar, Ganesh Prabu, Harihara Subramaniam, KPK, Krishna, Kumaran, Prabu, Prithvi, Ramnath, Satheesh, Sathyan, Sharmili, Shiv, Shyam, Srini, Srividhya, Sundar, Suvedha, Vidhoon, Vignesh, and Vinoth Sankar. A huge shoutout to Barath Sriram for spending his time towards proofreading this thesis for grammatical errors. I am grateful to him.

I was fortunate to get support from people whom I got to know only after coming to Canada as well as those who I already knew but were in Canada during the time of my graduate study. I am extremely thankful to the support and positivity from Ganesh Jawahar, Gurudev, Jai, Krishna, Lakshmanan, Martial Britto, Nishanth, Ragav, Sachin,

and Vaseetharan. I wholeheartedly thank Pooja Varshneya for her positivity and support during the last few months of my graduate study.

I am forever grateful to the unconditional love, sacrifice, and support from my Mom, Dad, Nithin, Amritha, and Amritha's Mom and Dad. I am extremely thankful to their prayers. I am blessed to have Amritha in my life and to have her constant encouragement, love, and support.

Dedication

To the God and for his/her script written for me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Intelligent computer systems are around us in our daily lives. We interact with them directly or indirectly on a day-to-day basis. Nowadays, when we want to send an email using Gmail [84], most of what we think to write is guessed correctly on most of the occasions, based on very few keystrokes from us. There was a time when we always pressed some buttons to provide our input to electronic devices, but now there are devices and interfaces to which we can just speak our language. The best part is that this works not only for English but also for several regional languages throughout the globe. All these advancements in technology are due to some recent and major developments in the areas of Machine Learning (ML), Natural Language Processing (NLP), and computer hardware especially Graphics Processing Units (GPUs), as well as the availability of large amounts of data.

Given a choice to pick just one software component which plays a major role in the above-mentioned real world applications, it would be a language model. Simply put, a language model learns to predict the probability of a sequence of words. Language models are components in several other well-known NLP applications including, but not limited to, speech recognition, machine translation, text summarization, and question answering. Language modeling, on its own, is an important task in natural language processing and understanding. Models that can identify probability distribution over sequence of words can encode complexities of language, such as grammatical structure [38]. Also, they can extract a fair amount of knowledge that a corpus contains [38].

Because of the crucial role that language models play, it has become very important to pay careful attention to the fundamental problems associated with language modeling.

For different input contexts (sequence of previous words), to predict the next word, a neural network language model outputs a probability distribution over all the words in the vocabulary by using a softmax function. When the log of the probability outputs for all such contexts are stacked together, the resulting matrix is a log probability matrix which can be denoted as $\mathbf{Q}_\theta$, where θ denotes the learnable model parameters. Let the true log probability distribution be represented as $\mathbf{Q}^*$. Yang et al. [86] hypothesized that $\mathbf{Q}^*$ should be high rank as natural language is highly context-dependent [50], and showed that existing softmax-based word-level neural language models cannot model natural language as they cannot produce high-rank $\mathbf{Q}_\theta$. They [86] called this limitation the *softmax bottleneck*. To break the bottleneck, they also proposed a solution known as Mixture of Softmaxes (MoS). By replacing softmax with MoS, they showed that the neural language models can produce a high-rank $\mathbf{Q}_\theta$, thus offering a better performance on test data. Also, they claimed "Higher the rank of $\mathbf{Q}_\theta$, better the performance of the language model on test data" Subsequently, different solutions were proposed to break the bottleneck in the works of Kanai et al. [40], Ganea et al. [22], Yang et al. [87], and Takase et al. [72].

Typically, in machine learning, in order to influence the learning process of a model, a set of adjustable parameters called *hyperparameters* are used that are fundamentally different from the learnable model parameters. The data is usually split into training data and test data. The learning process generally uses the training data and the performance of a model is evaluated on the test data. The steps undertaken to improve the performance of the model on the test data with or without trade-offs in its performance on the training data is known as *regularization*. During the process of learning, to find the best performing model parameters $\boldsymbol{\theta}^*$, iterative optimization algorithms like gradient descent are commonly used. There are different types of gradient descent algorithms, and two of them are Stochastic Gradient Descent (SGD) and Averaged Stochastic Gradient Descent (ASGD). All the solutions that were proposed to break the softmax bottleneck were built on top of the AWD-LSTM[1], a state-of-the-art recurrent neural network based language model. The learning process for the AWD-LSTM involves the use of both SGD and ASGD, starting with SGD and later switching to ASGD.

Among the solutions to break the softmax bottleneck, we looked in depth into three fundamentally-different solutions, namely SigSoftmax (SS) [40], Linear Monotonic Softmax with Piecewise Linear Increasing Functions (LMS-PLIF) [22], and MoS [86]. SS did break the bottleneck, but the rank of $\mathbf{Q}_\theta$ produced was not as high as for MoS. Also, there was no significant performance improvement on the test data compared to MoS. LMS-PLIF was shown to have a considerable improvement in performance on the test data. But there was no discussion about the rank of $\mathbf{Q}_\theta$. So, we attempted to reproduce the results

[1] AWD-LSTM - **A**veraged SGD **W**eight-**D**ropped **L**ong **S**hort-**T**erm **M**emory

for LMS-PLIF and eventually found that it is low rank. MoS and LMS-PLIF introduce additional learnable parameters which could be an advantage as they can model more complicated dependencies. Also, the hyperparameters used for LMS-PLIF and MoS were different in comparison to the baseline model with softmax function. This puzzling set of observations made us to come up with a new function that we call *Generalized SigSoftmax* (GSS) which does not introduce any additional learnable parameters like SS, but at the same time can help language models to produce $\mathbf{Q}_\theta$ with diverse ranks, including ranks in the range that MoS is able to produce, so that we can better understand the real consequences of the softmax bottleneck.

With the help of GSS and by mostly using the same set of hyperparameters for different models under comparison, we observe that the rank of $\mathbf{Q}_\theta$ does not strongly correlate with the better performance of the model on test data. We conduct a qualitative analysis for different language models, with low, medium, and high-rank $\mathbf{Q}_\theta$. We notice that the rank of $\mathbf{Q}_\theta$ does not strongly correlate with the quality of the predictions made by the language model. In a neural word-level language model, each word is represented by a real-valued vector of dimension d which is known as a word embedding. During the process of learning a language model, the word embeddings are also learned. Hence, we also evaluate the learned word embeddings from language models with different ranks for $\mathbf{Q}_\theta$ on word similarity benchmarks, and find that high-rank language models do not necessarily learn better embeddings.

As we find that there are no additional benefits in having a high-rank language model, we question the fundamental observation [86] that "Higher the rank of $\mathbf{Q}_\theta$, better the performance of a language model on test data". We conduct experiments by adjusting a few key hyperparameters of high-rank models, and observe that the positive correlation between the rank of $\mathbf{Q}_\theta$ and the performance on test data is not always true.

As singular values are typically used to calculate the rank of any matrix, we inspect the singular value distribution of $\mathbf{Q}_\theta$ and find that the distributions are very similar for models having $\mathbf{Q}_\theta$ with different ranks. All these distributions have a very fast decay rate. Hence, we question the existing approach to calculate rank from singular values of $\mathbf{Q}_\theta$, by conducting a few experiments to check for its robustness. We observe that this approach is very sensitive to tiny noise and had also failed to capture the similarity in the distributions. Therefore, we propose and suggest the use of an alternative metric which we call as ϵ-*effective rank*, which is robust and can also approximately quantify the singular value distribution when different values for ϵ are used.

Finally, we set out to find the reason why LMS-PLIF and MoS were able to perform better, and find that it is regularization which helps LMS-PLIF to perform slightly better.

For MoS, we argue that its better performance could be due to various reasons including the choice of hyperparameters and the indirect regularization, the inherent advantage because of its network structure, and the use of a mixture of probability distributions instead of one to make predictions.

1.2 Contributions

The major contributions of this thesis are:

1. We introduce a new family of parametric non-linear functions which we call the Generalized SigSoftmax (GSS) that can produce $\mathbf{Q}_\theta$ with diverse ranks.

2. We expose the unfairly used hyperparameter when training the AWD-LSTM model in some existing works, which is the trigger condition to switch the optimization algorithm from SGD to ASGD.

3. We show that the language models which can produce a high-rank $\mathbf{Q}_\theta$ do not necessarily: have better quantitative performance, make better quality predictions, or learn better embeddings.

4. We expose that the $\mathbf{Q}_\theta$ produced by both low and high-rank models rather suffer from the phenomenon of fast singular value decay. To approximately quantify the singular value distributions, we introduce and suggest the use of an alternative metric to calculate rank from singular values, namely ϵ-effective rank.

5. We show that the $\mathbf{Q}_\theta$ produced by LMS-PLIF is low-rank and the reason for LMS-PLIF's slightly better performance is regularization.

6. We argue that the high-rank $\mathbf{Q}_\theta$ produced by MoS is just a by-product and MoS's better performance could be due to: the choice of hyperparameters, the indirect regularization, its network structure, and the fact that it uses a mixture of probability distributions instead of one to make predictions.

1.3 Outline

The rest of this thesis is organized as follows:

Chapter 2: In this chapter, we discuss all the prerequisites that are needed to better understand: 1) the *softmax bottleneck* and another fundamentally similar problem called the *representation degeneration* that are observed in neural language models, and 2) the experiments that we present in later chapters.

Chapter 3: As both the above-mentioned problems were observed recently and as there are only a handful of works that had proposed solutions, we briefly discuss all the proposed solutions in this chapter.

Chpater 4: In this chapter, we introduce GSS and empirically show that the GSS function can produce $\mathbf{Q}_\theta$ with diverse ranks, by controlling its parameters. We expose one of the highly-overlooked hyperparameters in existing works. Also, we fairly compare GSS with other existing solutions and discuss the results.

Chapter 5: In this chapter, we show and discuss the results for experiments which were done to: check for any additional benefits that a high-rank language model could bring in, verify the correlation between high-rank and better performance on the test data, check the robustness of the approach used to calculate rank, and show the impact on regularization by LMS-PLIF and MoS.

Chapter 6: In this chapter, we present our conclusions and some ideas which could be pursued on in future.

Chapter 2

Background

This chapter will provide all the necessary knowledge needed to follow the upcoming chapters. First, we will briefly discuss Machine Learning (ML) with an emphasis on neural networks in general, neural networks for Natural Language Processing (NLP), and Language Modeling. Then, we will discuss Singular Value Decomposition (SVD). Finally, we will discuss two similar problems that are seen in neural language models namely Softmax Bottleneck and Representation Degeneration.

2.1 Machine Learning

Machine Learning can be defined [54] as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty.

2.1.1 Major Stages in a Machine Learning Process

2.1.1.1 Data Set Collection

For any problem to which we seek a potential machine learning based solution, the first thing to do is to collect data. There are a few key characteristics of the data set $\mathcal{D}$ that help in choosing appropriate methodologies in subsequent stages (such as modeling):

- The number of examples ($N = |\mathcal{D}|$[1]) present in the data set.

[1] $|.|$ of any set denotes the number of elements in that set.

- The nature of the data in terms of its high-level representation, i.e., image, text, audio, video, time-series.
- The presence or absence of a set of labels (discrete or scalars) for every example in the data set. With the presence of labels, each example can be described as an input-output pair $(\mathbf{x_i}, y_i)$ where y_j can be a set or a scalar, thus the data set can be represented as $\mathcal{D} = \{(\mathbf{x_i}, y_i)\}_{i=1}^{N}$. When there are multiple labels $|y_i| > 1$, otherwise $|y_i| = 1$. Whereas during the absence of labels, each example contains only inputs, i.e., $\mathbf{x_i}$, thus the data set can be denoted as $\mathcal{D} = \{\mathbf{x_i}\}_{i=1}^{N}$.

2.1.1.2 Modeling

A model can be defined [45] as a restricted family of hypothesis functions $\mathcal{H} = \{h_1, h_2, .., h_M\}$ which can map input $(\mathbf{X} = [\mathbf{x_1}\ \mathbf{x_2}\ ..\ \mathbf{x_N}]^T)$ to output (Y), discover the dependency between them, or discover the structure of input. Based on the nature of the data, the data set size, and the problem which we are trying to solve, an appropriate model needs to be chosen. There are two major classes of models:

1. *Parametric* models that have a fixed set of parameters $\boldsymbol{\theta}$ which is independent of the data set size N. We will mainly focus on this class of models.
2. *Non-parametric* models that have a total number of parameters proportional to the data set size, i.e., the number of parameters can be small or large based on the data set size.

2.1.1.3 Optimization

Given a parametric model and a data set, the model now needs to learn about the data. The process of learning (or training) is to find the best performing parameters for the model $\boldsymbol{\theta}^*$ from a parameter space $\boldsymbol{\Theta}$ such that it helps the model to better discover the dependency between the input and output or better learn about the data itself. Each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ can have an one-to-one or many-to-one correspondence with the members of $\mathcal{H}$. To find $\boldsymbol{\theta}^*$, objective functions $J(\boldsymbol{\theta})$ are used. Depending on the problem, the best parameters are obtained by either minimizing or maximizing the objective function. Note that maximizing a function is equivalent to minimizing the negative of that function. Hence, we just focus on the minimization form.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} J(\boldsymbol{\theta}) \tag{2.1}$$

The optimization problem presented in equation 2.1 can be solved by either analytical or numerical methods. Analytical solutions can be computationally expensive in many cases, or there could be no analytical solutions available. Hence, numerical methods like gradient descent are widely used to find an optimal solution. Gradient descent [66] is an iterative optimization algorithm that helps in minimizing the objective function $J(\boldsymbol{\theta})$ by updating the parameters in the opposite direction of the gradient[2] of the objective function with respect to the parameters, i.e., $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. There is a hyperparameter[3] for this algorithm, called the learning rate or step size η which determines the size of the step to take during each iteration while moving towards the minimum of the objective function.

There are three major types of gradient descent algorithms:

- *Batch Gradient Descent*: Also known as vanilla gradient descent in which gradient is computed for all examples under consideration:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta . \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{X}, Y) \tag{2.2}$$

  where $\boldsymbol{\theta}^{t}$ and $\boldsymbol{\theta}^{t+1}$ are the parameters of model before and after the optimization iteration t. To perform one iteration, gradients for all examples $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{X}, Y)$ are calculated, which can be time consuming and memory intensive.

- *Stochastic Gradient Descent*: In stochastic gradient descent (SGD), gradients are computed for each example $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}_i, y_i)$:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta . \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}_i, y_i) \tag{2.3}$$

- *Mini-Batch Gradient Descent*: In mini-batch gradient descent, gradients are computed for every mini-batch of n examples $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}_{i:i+n}, y_{i:i+n})$:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta . \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}_{i:i+n}, y_{i:i+n}) \tag{2.4}$$

  In practice, mini-batch gradient descent is the most-used form of gradient descent algorithm.

[2] A gradient is a vector of partial derivatives.

[3] In machine learning, a hyperparameter is a parameter whose value is set by users to control the training process. It is different from model parameters which are learned during training.

2.1.1.4 Regularization

The goal of machine learning is not just that the trained model performs well on already seen examples in the data set; it also has to perform well on new, unseen inputs. The ability of a model to perform well on new, unseen inputs is called *generalization* [26].

For the purpose of testing this ability of a model, a data set $\mathcal{D}$ is usually split into training set and test set. The model sees only the examples from the training set during the training process, and its generalization can be measured from its performance on test set. The gap between performance on the training set and the test set is called generalization gap. Ideally, a model has to:

1. Perform well on the training set.
2. Perform well on the test set, so that the generalization gap is small.

When a model is unable to perform well on the training set, *underfitting* happens. Contrarily, when it performs well on the training set but is unable to obtain a small generalization gap, the phenomenon is known as *overfitting*. The steps undertaken to improve the performance on the test set which also reduces the generalization gap is known as *regularization*.

2.1.2 Major Classes of Machine Learning

2.1.2.1 Supervised Learning

When we have a data set $\mathcal{D}$ which contains input-output pairs $(\mathbf{X}, Y)$, if we are unaware of the true dependency $f(\mathbf{X}) = Y$ or $P(Y|\mathbf{X})$ between them, and if we make the model to discover the dependency between them, we have supervised learning.

Based on the form of y_j, below are two types of supervised learning approaches:

1. *Classification*: Elements of $y_j \in \{c_1, c_2, .., c_K\}$, i.e., elements of y_j belonging to a finite-set of K classes or categories. If $|y_j| = 1$, it is generally known as classification. Whereas if $|y_j| > 1$, it is called multi-label classification. If $K = 2$, it is known as binary classification and if $K > 2$ it is called multi-class classification.
2. *Regression*: If y_j is a scalar label and $y_j \in \mathbb{R}$ i.e., y_j being real-valued.

2.1.2.2 Unsupervised Learning

When we have a data set which only contains inputs $\mathbf{X}$, and if we make the model to discover some patterns within $\mathbf{X}$, it is called unsupervised learning. Clustering [36] and density estimation [68] are two examples of this approach.

2.1.2.3 Reinforcement Learning

Reinforcement learning [71] is learning what to do, i.e., how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the highest reward by trying them. It is different from both supervised and unsupervised learning.

2.1.3 Loss Functions

Functions that are used to quantify errors in predictions made by the models are called *loss functions* [30]. In supervised learning, the loss function is usually a function of true output and predicted output i.e., $L(Y, f(\mathbf{X}; \boldsymbol{\theta}))$ and it outputs a scalar loss or error value. Two of the widely used loss functions are:

- *Mean Squared Error*: Mean Squared Error (MSE) loss function is the widely used loss function for regression problems. It is defined as:

$$L_{\text{MSE}}(Y, f(\mathbf{X}; \boldsymbol{\theta})) = \frac{1}{N} \sum_{i=1}^{N} \big(Y - f(\mathbf{X}; \boldsymbol{\theta})\big)^2 \tag{2.5}$$

- *Cross-Entropy*: Cross-Entropy (CE) loss is the widely used loss function for multi-class classification problems. For a single training example, it is defined as:

$$\ell_{\text{CE}}\big(g(y_i), f(\mathbf{x}_i; \boldsymbol{\theta})\big) = -\sum_{k=1}^{K} g(y_i)_k \log(f(\mathbf{x}_i; \boldsymbol{\theta})_k) \tag{2.6}$$

  where $g(y_i)$ is a function[4] that converts a class label or a set of labels y_i to a one-hot vector of dimension K. Similarly, $f(\mathbf{x}_i; \boldsymbol{\theta})$ is also a vector of dimension K which has the probability values for each class label in its components.

[4] One-hot encoding produces a vector of length equal to the number of class labels K in the data set. If y_i has only one label and if it is the label of k-th class, then the vector has its k-th component set as 1 and the rest as zero.

2.1.4 Models for Classification

According to Bishop [7], there are three distinct approaches that are widely used for solving classification problems:

1. *Generative models*: Approaches that explicitly or implicity model the probability distributions of inputs $P(\mathbf{X})$ as well as outputs $P(Y)$. As $P(\mathbf{X})$ is modeled, it is possible to generate synthetic input data points $\mathbf{x}_{s_i}$, hence getting its name generative.

2. *Discriminative models*: Approaches that directly model the posterior probabilities $P(Y|\mathbf{X})$.

3. *Discriminant functions*: Functions $f(.)$ that can map every input $\mathbf{x}_i$ directly onto a class label c_j. Some examples of this kind of functions are Fisher's linear discriminant [21] and Perceptron [64].

When the classes $\{c_1, c_2, .., c_k\}$ are disjoint, each input $\mathbf{x}_i$ is assigned to only one class c_j. In such scenarios, the input space can be divided into decision regions whose boundaries are known as decision boundaries. Models in which the decision boundaries are linear functions of the input vector $\mathbf{x}_i$, are called linear models. Contrarily, if the decision boundaries are non-linear functions of the input vector, such models are called non-linear models.

The data sets whose classes can be separated by linear decision surfaces are known as linearly separable [7] data sets.

$$f(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i + b \tag{2.7}$$

Equation 2.7 is an example for a simple linear discriminant function that could be used for a two class classification. When $f(\mathbf{x}_i) \geq 0$, $\mathbf{x}_i$ can be assigned to c_1, and when it is < 0, to c_2. Hence, the decision boundary in this case would be $f(\mathbf{x}_i) = 0$. For two and three dimensional input spaces, $f(\mathbf{x}_i) = 0$ would be a line and a plane respectively. In general, the decision boundaries are hyperplanes and hypersurfaces in the input space for linear and non-linear models, respectively.

In practice, when the original input space $\mathbf{X}$ is not linearly separable, either of the following methods can used:

- A non-linear transformation is applied to convert them into feature space where it could become linearly separable. Then, the features can be fed as inputs to linear models.

- Can be fed directly as inputs to non-linear models.

Some examples of linear models for classification include Logistic Regression [11] and and Linear Support Vector Machines (Linear SVMs) [77]. Similarly, Decision Trees [5], k-Nearest Neighbour [10], Kernel SVMs [9], and Multi-Layer Perceptrons are a few examples of non-linear models for classification. We will mainly focus on discriminative non-linear models.

2.1.5 Feedforward Neural Networks

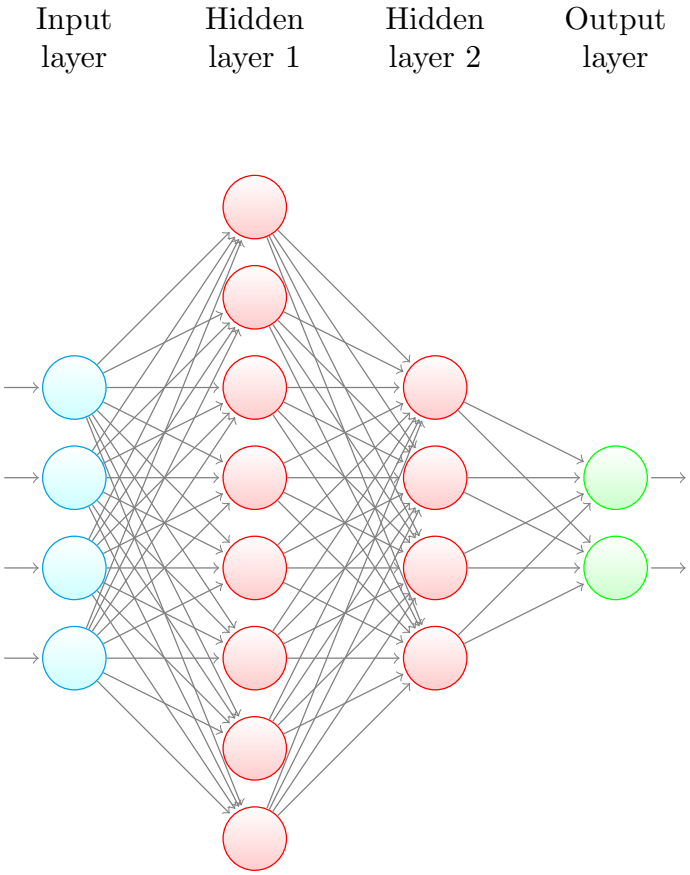


Figure 2.1: A representative MLP or 3-layer FNN. Image adapted from [45].

Inspired by early models of sensory processing by the brain [43], the concept of Artificial Neural Network (ANN) was introduced to mimic human brain's process of learning to solve

different types of problems. The core component in any ANN is an artificial neuron which models the biological neurons.

$$g(\mathbf{x_i}) = \phi(\mathbf{w}^T\mathbf{x_i} + b) \tag{2.8}$$

Equation 2.8 is the function notation for an artificial neuron whose inputs are a weight vector $\mathbf{w}$, an input data vector $\mathbf{x_i}$, and a scalar bias term b. ϕ is known as the activation function which in most cases is non-linear. Loosely, the activation functions model the firing of a biological neuron. Sometimes, the weighted sum $\mathbf{w}^T\mathbf{x_i} + b$ is called a pre-activation. An artificial neural network is constructed by connecting multiple neurons together to form a directed graph in which most of the nodes are neurons.

A Feedforward Neural Network (FNN) is a class of ANN in which the connection between nodes do not form cycles, i.e., the network is essentially a directed acyclic graph. Typically, an FNN is structured in layers, as seen in Figure 2.1. An input layer just passes on the input to subsequent hidden layers or to the output layer and is not really made up of neurons. Whereas, the hidden and the output layers are composed of one or more neurons. Hence, typically an N-layer FNN means the total number of layers in the network (minus the input layer) equals N. An important detail that needs to be mentioned about the output layer is that a single activation function can be used over the weighted sum output from all of its neurons.

Perceptrons and Multi-Layer Perceptrons (MLP) are two well-known examples of FNN. Perceptrons are the simplest kind of FNN which has only one neuron with a unit step function as its activation function. An MLP is an N-layer FNN with $N \geq 2$ that should have at least one hidden layer, one output layer, and any number of neurons within them.

2.1.6 Backpropagation

Let $\boldsymbol{\theta}$ be the parameters of an FNN which includes weights and biases of all the neurons in all the layers. When this network is trained using gradient descent algorithm to find the best performing $\boldsymbol{\theta}^*$, the gradient of the loss function[5] $\nabla_{\boldsymbol{\theta}} L$ is calculated at every optimization iteration which involves computation of several partial derivatives. When these partial derivatives are computed naively using the chain rule for larger networks with a large

[5]There is a subtle difference between minimizing a loss function and minimizing an objective function [26]. For classification problems, the objective would typically be a maximum log-likelihood function which is equivalent to minimum negative log-likelihood, which is essentially minimizing the cross-entropy loss. So, we will use these terms interchangeably depending on the context, to be consistent with the literature.

number of parameters, training can be very time consuming, as it would involve a lot of redundant intermediate computations. To fix this redundancy issue, efficient techniques to compute derivatives are necessary. *Backpropagation* is one such technique; it still makes use of chain rule but in a clever way. Once the loss value is computed, the derivatives of the scalar loss value with respect to parameters in the last layer of the network are computed. Then the derivatives of the loss value with respect to the parameters in the preceding layers are computed iteratively. While doing so, the computation of the derivatives depends only on a few other derivatives but those have been already computed during the previous iteration at the immediate successive layer which is usually saved. Those saved derivatives are reused. By doing so, the redundancy in the computation of the intermediate partial derivatives is avoided. *Automatic differentiation* refers [27] to a general way of taking a computer program which computes a value and automatically constructing a procedure for computing derivatives of that value. Backpropagation can be considered as a special case of reverse mode automatic differentiation. There are several well-known automatic differentiation frameworks such as Chainer [75], Tensorflow [1], and PyTorch [58].

2.1.7 Activation Functions

Cybenko [12] proved that "a feedforward neural network having a single hidden layer with a finite number of neurons and sigmoidal activation functions can approximate any continuous function" [13]. It was also known as the universal approximation theorem, but the theorem itself was extended and got revised multiple times by others since then. A function is called sigmoidal in nature if it has the following properties [13]: 1) continuous, 2) bounded between finite lower and upper bound, 3) non-linear, 4) domain contains all real numbers, and 5) outputs have an 'S'-shaped curve. Some of the commonly-used activation functions are:

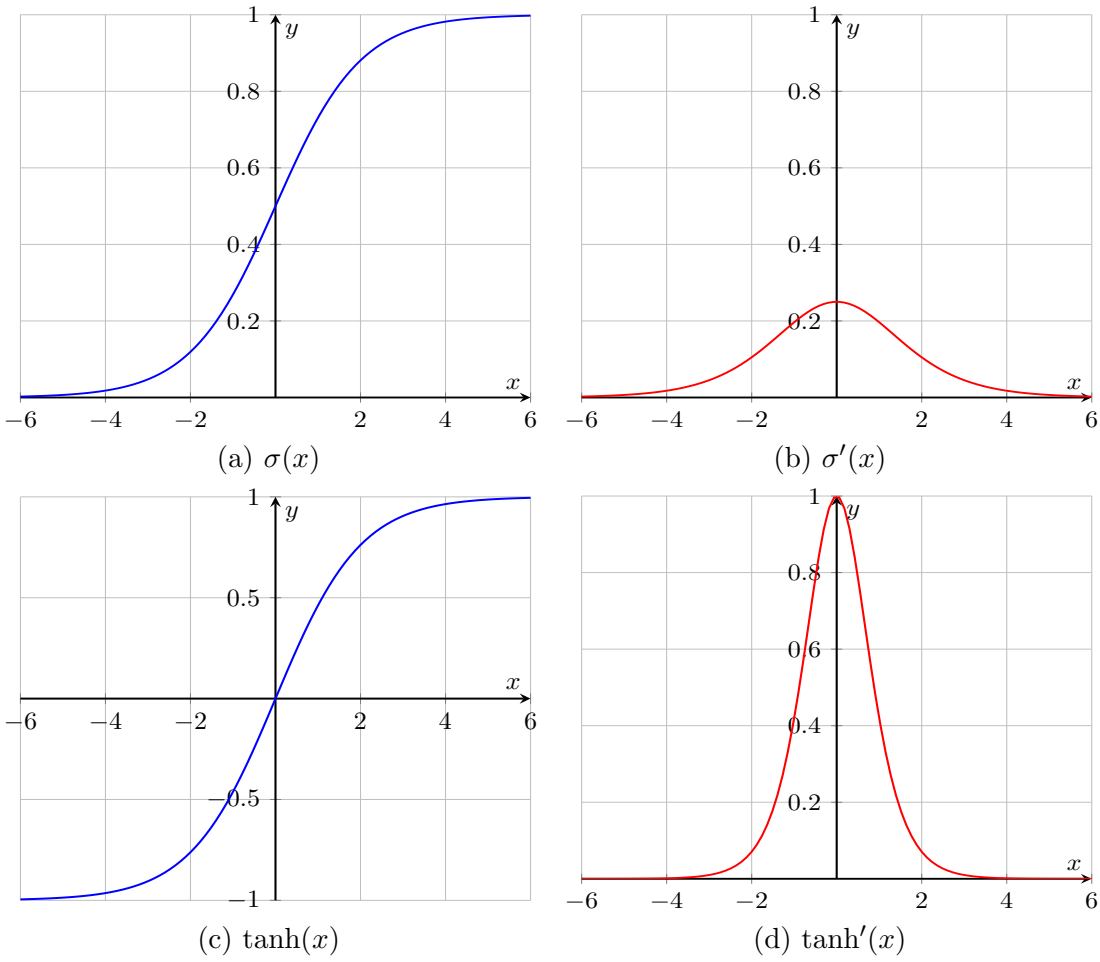
2.1.7.1 Sigmoid

The sigmoid function, also known as logistic sigmoid, is defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.9}$$

It belongs to the sigmoidal family of functions. As the range of the function 2.9 is too small compared to its domain, large changes in input values could lead to very small changes of

the output, which could potentially lead to vanishing[6] gradient problem. Also, it can never output negative values, which can be a drawback in certain scenarios.



(a) $\sigma(x)$ (b) $\sigma'(x)$

(c) $\tanh(x)$ (d) $\tanh'(x)$

Figure 2.2: Comparison between sigmoid and hyperbolic tangent and their derivatives.

[6]During each optimization iteration in a gradient descent algorithm, we know that each weight in a neural network receives an update based on the partial derivative of loss function with respect to that weight. When the activation of a weighted sum does not change much in comparison to large changes in weights, the updates during optimization iteration can be very small. This is known as the *vanishing gradient* problem. Contrarily, if the updates are extremely large in comparison to very small changes in weights, the *exploding gradient* problem happens.

2.1.7.2 Hyperbolic Tangent

The hyperbolic tangent (tanh) is another sigmoidal function which can output positive, zero, and negative values. Like sigmoid, it could also lead to vanishing gradients. The tanh function is defined as:

$$\tanh(x) = \frac{1 - \exp(-x)}{1 + \exp(x)} \tag{2.10}$$

2.1.7.3 ReLU

Nair and Hinton [55] introduced Rectified Linear Units (ReLU) which is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.11}$$

ReLU is continuous but is not bounded and can only output zero and positive values. Also, it is not differentiable at $x = 0$ (as seen in Figure 2.3b). However, it has multiple advantages over the previously discussed sigmoidal functions, as follows:

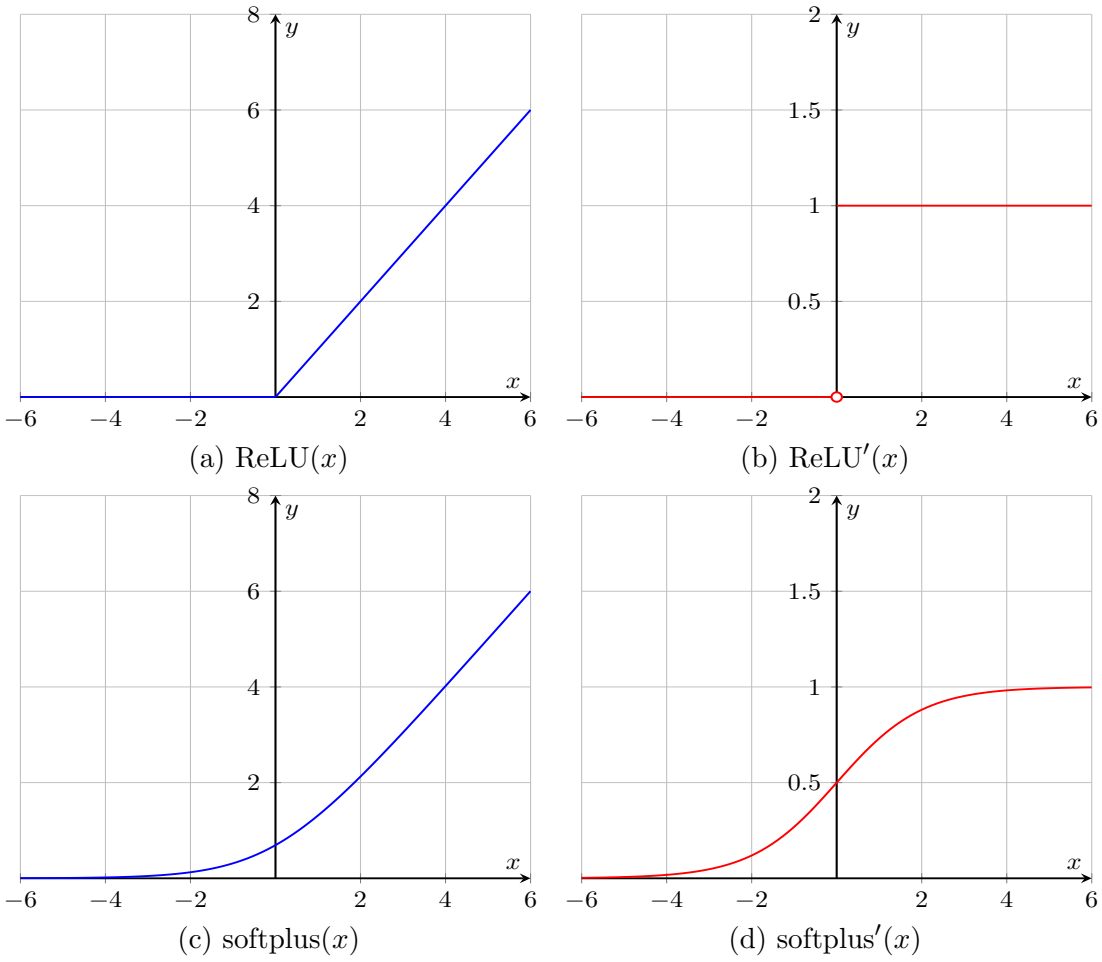
- Does not lead to vanishing gradients.
- It is faster as it does not involve exponent computations.
- Empirically, it offers better performance in neural networks.

2.1.7.4 Softplus

A smooth approximation to ReLU, known as softplus [16], was in existence even before ReLU was introduced. The softplus function is defined as:

$$\text{softplus}(x) = \log(\exp(x) + 1) \tag{2.12}$$

Unlike ReLU, it is differentiable everywhere. The derivative of softplus is the logistic sigmoid function (as seen in Figure 2.3d). However, in practice, softplus does not perform better than ReLU when used as an activation function for neurons in the hidden layers [26].



(a) ReLU(x) (b) ReLU$'(x)$

(c) softplus(x) (d) softplus$'(x)$

Figure 2.3: Comparison between ReLU and softplus and their derivatives.

2.1.7.5 Softmax

Softmax is an activation function which is especially used in the output layer of a neural network when there is a need to get a probability distribution as the output of the network. The weighted sums computed in the output layer's neurons are known as logits which are fed as inputs to the softmax function. Hence, the output layer without any activation function is informally called the logits layer. Fundamentally, both the input and the output of a softmax function are vectors. For brevity, we will always represent them in terms of just one vector component each. Assuming there are C logits (or components), the softmax

function can now be defined as:

$$\text{softmax}(l_z) = \frac{\exp(\text{l}_\text{z})}{\sum_{i=1}^{C} \exp(\text{l}_\text{i})} \tag{2.13}$$

where l_z is z-th logit. Softmax is used for multi-class classification problems.

2.1.8 Regularization Techniques for Neural Networks

There are several well-known and widely used techniques for regularizing neural networks. Also, it is a very active area of research where a lot of narrowly-focused techniques are being introduced every now and then. Here, we discuss some of the widely-known as well as less-known techniques which were beneficial to our research.

2.1.8.1 L2 Regularization

L2 regularization is one of the most commonly-used regularization technique for neural networks, as well as other machine learning models. It helps to avoid overfitting as it penalizes large weight values in neurons. It is also informally known as weight decay or L2 decay, as it forces the weights towards smaller values. The technique is to just add a regularization term to the loss function.

$$L_{reg}(.) = L(.) + \lambda ||\boldsymbol{\theta}||_2 \tag{2.14}$$

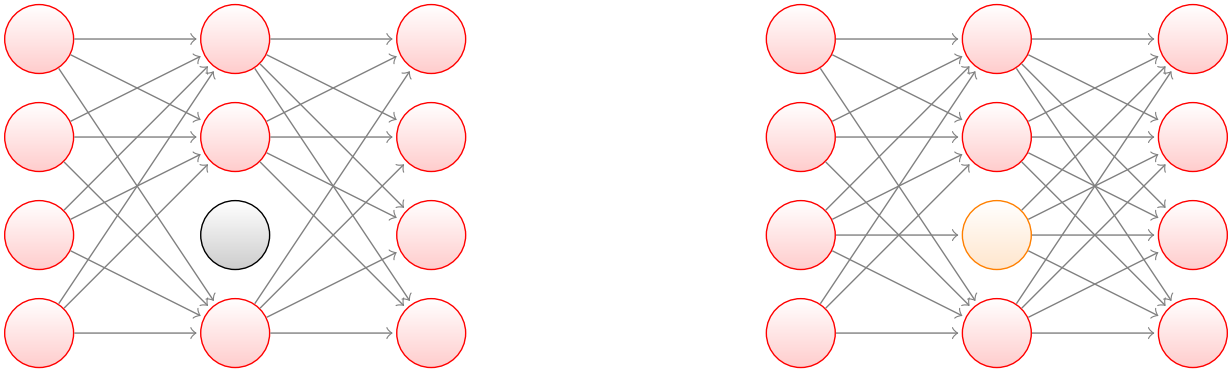
where λ is a scaling factor which is usually a hyperparameter and $||.||_2$ is the Euclidean norm.

2.1.8.2 Dropout

Srivastava et al. [69] introduced a simple yet powerful technique to regularize overfitting neural networks. The idea is to randomly drop neurons along with their input and output connections during training. In practice, it is achieved by setting the activation of those neurons to zero. As seen in Figure 2.4a, the neuron in black color is an example for a dropped-out neuron.

2.1.8.3 DropConnect

Wan et al. [80] proposed a generalization to dropout in which instead of randomly dropping neurons entirely, only a random subset of incoming connections from previous layers were dropped. In practice, it is done by setting the weights of those input connections temporarily to zero. In Figure 2.4b, the neuron in orange color is an example for a neuron on which dropconnect is applied. We could see that out of four connections, it receives only one.



(a) Dropout (b) DropConnect

Figure 2.4: Difference between Dropout and DropConnect. Image adapted from [45].

2.1.8.4 Flooding

Neural networks with sufficiently large number of parameters have the capacity to memorize examples in the training set, thus leading to a zero training loss. But, none of the widely used regularization techniques directly aim to avoid zero training loss. Ishida et al. [35] introduced a technique to prevent the reduction of training loss beyond a particular level called *flooding* level. The technique is to use gradient descent as usually for training, but when the training loss goes below the flooding level, gradient ascent[7] is used. By doing so, the training loss can be maintained around the desired flooding level.

[7] In gradient ascent, parameters are updated along the same direction as the gradient, in contrast to gradient descent where the parameters are updated along the opposite direction.

2.2 Natural Language Processing

Natural language processing (NLP) can be defined [17] as the set of methods for making human language accessible to computers. Methods in NLP are drawn from several related fields like Computational Linguistics, Machine Learning, and Artificial Intelligence.

2.2.1 Neural Networks for Natural Language Processing

2.2.1.1 Recurrent Neural Networks

Feedforward Neural Networks (FNNs) cannot persist information (i.e., they are memory-less) from a particular point in time, to be used later. They assume all its inputs and outputs are independent to each other. Also, they cannot handle variable-length input sequences. So they are not applicable to sequence processing tasks like natural language processing or generation in which the output at a particular time step depends on the outputs and inputs of previous time steps in addition to the input at the current time step. Recurrent Neural Networks (RNNs) [67] are networks with loops, allowing information to persist with the help of a memory. RNNs can be thought of as multiple copies of a network of neurons (as seen in Figure 2.5), all copies with the same parameters (this is called parameter sharing), each of them accepting inputs at different time steps, but each of those copies pass information with the help of memory to its successor copy. The network of neurons can differ among RNNs. One of the simplest form of recurrent networks is the Elman network [18] which is characterized by the following equations:

$$\mathbf{h}_t = \phi_h(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{2.15}$$

$$\mathbf{y}_t = \phi_y(\mathbf{V}\mathbf{h}_t + \mathbf{c}) \tag{2.16}$$

where $\mathbf{W}, \mathbf{U}$ and $\mathbf{V}$ are learnable weight matrices; $\mathbf{b}, \mathbf{c}$ are learnable bias vectors; and ϕ_h, ϕ_y are activation functions. $\mathbf{h}_t$ is the hidden state vector (which helps in persisting information across time steps) at time step t. $\mathbf{x}_t$ and $\mathbf{y}_t$ are the input and output vectors of the network during time step t. To train RNNs using gradient descent and backpropagation, the network is unrolled (as shown in Figure 2.5) and the loss value is computed for each time step t. For each copy of the network, gradients are calculated and parameters are updated. Doing backpropagation like this is known as *backpropagation through time* (bptt). A drawback of this approach is that it can be very time consuming for long sequences.

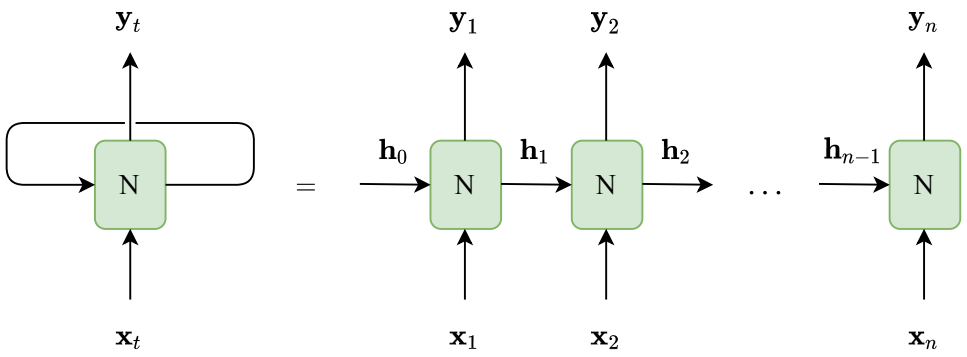


Figure 2.5: An RNN as multiple copies of network (of neurons) with information passing connections (memory state) between them. Image adapted from [56].

2.2.1.2 Long Short-Term Memory

Bengio et al. [6] and others had showed practical difficulties of training RNNs (made up of simple networks) with the gradient descent algorithms, especially when information needs to be persisted for a long time span. To overcome such difficulties, different approaches [33] were proposed along the directions of variants to gradient descent, variants to improve backpropagation over long range, alternative network topology, and so on. In particular, to remedy the problem of vanishing and exploding gradients, Long Short-Term Memory (LSTM) networks were introduced by Hochreiter and Schmidhuber [33]. The core idea behind the LSTM networks is the cell state which is capable of allowing information to flow unchanged, and gated layers to change it. There are several variations to the initially proposed LSTM by Hochreiter and Schmidhuber [33]. But the most used version of LSTMs are the ones proposed by Gers et al. [24] which have three gates, namely input, output, and forget gates. The purpose of the input gates is to protect the information stored in the cell's state from perturbation by irrelevant inputs. Similarly, the output gates filter information from the cell's state before passing it on to successive time step. The forget gates decide what information to keep and what information to throw away from the cell's state. Below are the equations that characterize an LSTM network at a single time step t:

$$\mathbf{i}_t = \sigma(\mathbf{W}^{\mathbf{i}}\mathbf{x}_t + \mathbf{U}^{\mathbf{i}}\mathbf{h}_{t-1})$$
$$\mathbf{f}_t = \sigma(\mathbf{W}^{\mathbf{f}}\mathbf{x}_t + \mathbf{U}^{\mathbf{f}}\mathbf{h}_{t-1})$$
$$\mathbf{o}_t = \sigma(\mathbf{W}^{\mathbf{o}}\mathbf{x}_t + \mathbf{U}^{\mathbf{o}}\mathbf{h}_{t-1})$$

$$\widetilde{\mathbf{c_t}} = \tanh(\mathbf{W^c x}_t + \mathbf{U^c h}_{t-1})$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \cdot \tanh(\mathbf{c}_t)$$

where $\mathbf{x}_t$ is the input to the cell at time step t, $\mathbf{c}_t, \mathbf{h}_t$ are the cell state and hidden state at time step t, and $\mathbf{W^i}, \mathbf{W^f}, \mathbf{W^o}, \mathbf{W^c}, \mathbf{U^i}, \mathbf{U^f}, \mathbf{U^o}, \mathbf{U^c}$ are the weight matrices associated with input i, forget f, and output o gates, as well as the cell's state c.

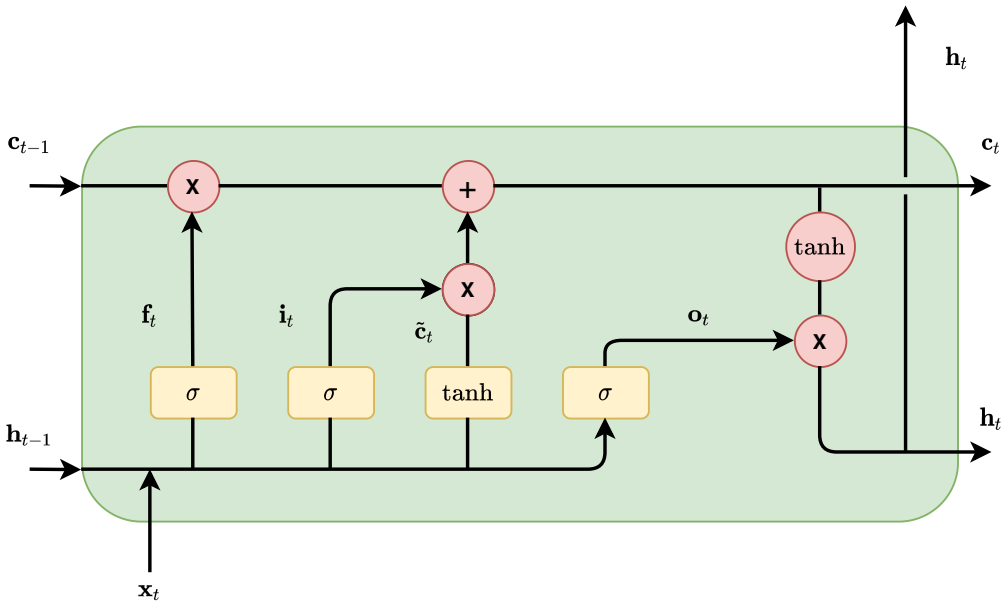


Figure 2.6: An LSTM network. Image adapted from [56].

2.2.1.3 Word Embeddings

Words can be represented as one-hot vectors for the purpose of feeding them as inputs to machine learning models for NLP. However, the vocabulary $\mathcal{V}$ of words under consideration is typically large, and representing each word by a vector of length $|\mathcal{V}|$ can be highly inefficient. Such a sparse representation is also called a local representation. In such representations, all words are considered to be equally different from each other. However, in natural language, there are a lot of semantically-similar words. Hence, the vector representations of such words should lie close to each other in their common vector space. In contrast to local

representations, there is another kind of representation called distributed representation [32], in which each word is represented as a vector in $\mathbb{R}^d$ such that a combination of its components or just one component can represent one or more aspects of the word. Such vectors are called *word embeddings*. They are dense representations. Learning word embeddings is an active area of research and one major breakthrough came with the work of Mikolov et al. [52], where neural network models were used to learn better word embeddings from a large corpus of text. Such learned representations can be used as an embedding layer in neural networks for NLP tasks. The embedding layer is a lookup function which has parameters in the form of a matrix $\mathbb{R}^{\mathcal{V} \times d}$, where each row is a vector representation of dimension d for a word in the vocabulary. This layer accepts the index of a word in the vocabulary as input and outputs the word's vector representation. Depending upon the task and its requirements, these representations (or parameter vectors) can also be further trained along with other layers in the network.

2.2.2 Language Modeling

Models that assign probabilities to a sequence of words are called word-level *language models* (LMs) [39]. In a word-level LM, the future words are predicted based on the available history of words. In the literature, the history of words is also called the *context*. Hence, we will use these two terms interchangeably.

The clouds are dark and it might _____

For the above sequence, the goal of a language model is to predict the next word w_{n+1} based on the available context or history of words $h = (w_1, w_2, .., w_n)$:

$$P(w_{n+1}|h) = P(w_{n+1}|(\text{The, clouds, are, dark, and, it, might})) \tag{2.17}$$

There are other variants of language models that operate below the word-level, namely character-level LMs and sub-word LMs. But we will focus only on word-level language models.

2.2.2.1 N-gram Language Models

A word-level n-gram model assigns probabilities to the next word based on relative frequency estimates, where n means the sequence (includes the history and the next word) length

under consideration.

$$P(w_{n+1}|w_1, w_2, .., w_n) = \frac{\text{Count}(w_1, w_2, .., w_{n+1})}{\text{Count}(w_1, w_2, .., w_n)} \tag{2.18}$$

For large sequences, estimating the probability using relative frequencies can be difficult. Hence, the history is approximated using only a few words to make n smaller. One such model is a bi-gram model ($n = 2$) where,

$$P(w_{n+1}|w_n) \approx P(w_{n+1}|w_1, w_2, .., w_n) \tag{2.19}$$

The above approximation is based on the Markov assumption: "The future state is dependent only on the present state and not on the other preceeding states." Other similar models based on the value of n are tri-grams ($n = 3$), 4-grams, and 5-grams.

2.2.2.2 Neural Language Models

Neural network models used for the task of language modeling are collectively called neural language models. As language modeling can be seen as a multi-class classification problem, a softmax activation function is typically used in the output layer of the neural language models. Regarding the embedding layer, there are two approaches that can be used in case of neural language models: 1) randomly initialize the representations for words and make them learn just like parameters in any other layer of the network, or 2) initialize with representations learned from other models, like word2vec [52], that can be further trained if needed. Neural language models have several advantages [39] over n-gram models as they can handle longer contexts, generalize similar contexts because of learnable input representations (word embeddings), and are generally believed to have a higher predictive accuracy.

Feedforward Neural Network Language Models (FNNLMs) are the simplest kind of neural language models. FNNLMs have all the limitations that are inherent to FNNs. FNNLMs use a sliding window approach over contexts to predict next word. The window size can become a bottleneck and no information is persistent between successive windows. This leads to a lot of redundant computations and becomes computationally expensive.

Recurrent Neural Network Language Models (RNNLMs) are the most commonly-used neural language models; they process a sequence word by word, and predict the next word based on the input word at the current time step t and the hidden state $\mathbf{h}_{t-1}$(or the context vector) which encodes information about the context. This makes RNNLMs perform better than n-gram models and FNNLMs, both of which are constrained by a limited context.

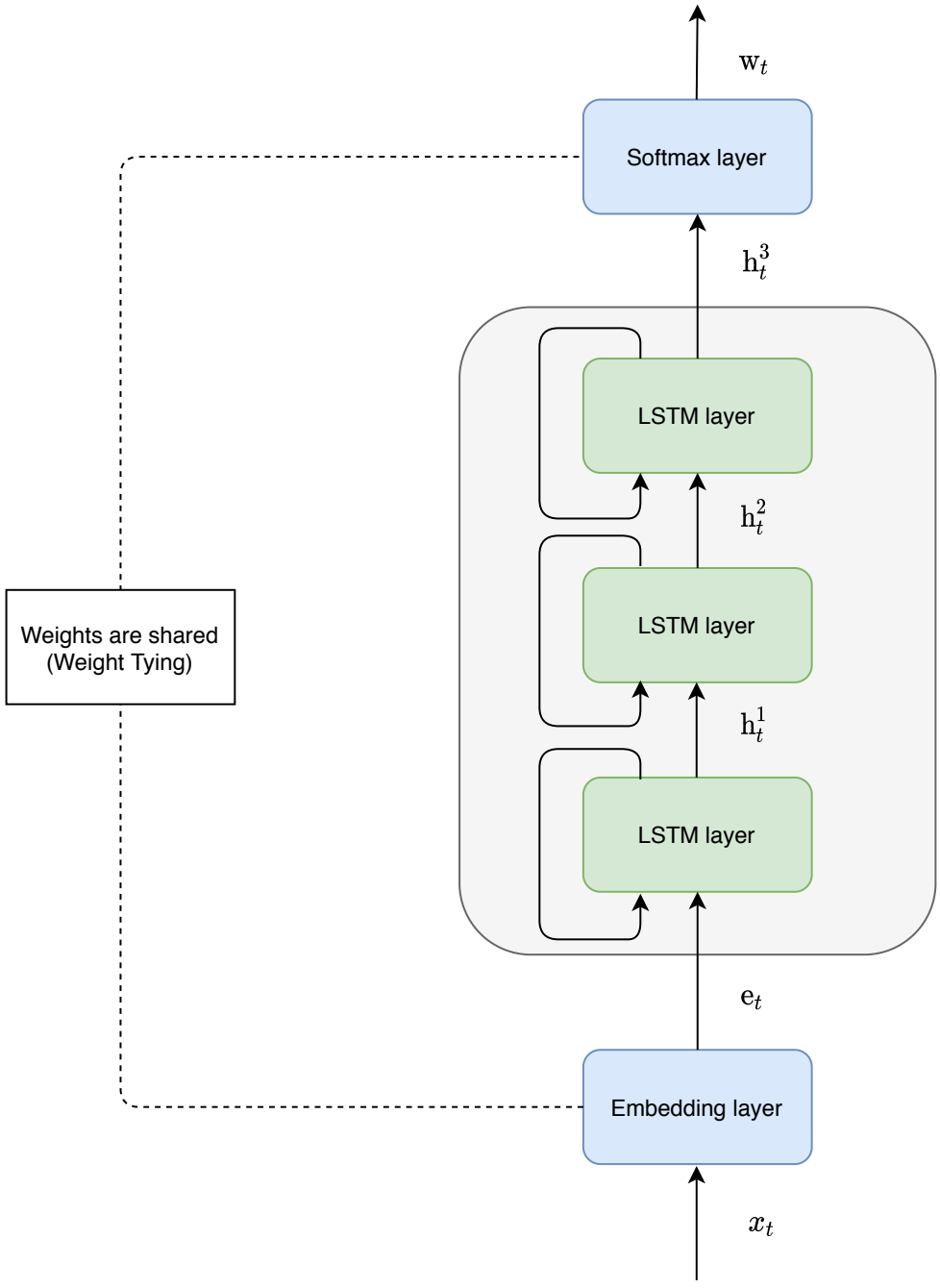


Figure 2.7: A rolled-up AWD-LSTM network operating at a particular time step t.

2.2.2.3 Averaged SGD Weight-Dropped LSTM (AWD-LSTM)

AWD-LSTM [49] is the state-of-the-art LSTM-based word-level neural language model. After its appearance, many models were introduced in the recent literature, built on top of it. AWD-LSTM (Figure 2.7) is an uni-directional stacked LSTM model with three hidden LSTM layers[8], a softmax output layer, and an input embedding layer. It is trained using an array of regularization and optimization techniques which are explained below.

1. **Averaged SGD**

   In stochastic approximation, Polyak and Juditsky [59] found that instead of using gradient resulting from a single optimization iteration, averaged gradient over all past iterations works better as its direction tends towards the global minimum. In SGD, when parameter updates happen using such gradients [59], it is known as Averaged SGD. For language modeling, SGD performed better [49] than other algorithms with momentum, like momentum SGD [70], Adam [41], Adagrad [15], and RMSProp [74]. Motivated by the above observation, Merity et al. [49] empirically investigated performance benefits that could be brought in by the usage of Averaged SGD (ASGD). They also introduced a variant of ASGD called Non-monotonically Triggered ASGD (NT-ASGD) wherein the switch from SGD to ASGD is determined by a non-monotonic condition instead of being tuned by the user. However, it has one hyperparameter which controls the switch, namely the non-monotone interval n, which essentially is a window of size n which maintains n past epochs'[9] validation[10] loss values during training. When validation perplexity fails to improve over multiple epochs, the switch to ASGD happens. The authors had reported that the best performing value for n is 5 across several data sets.

2. **Variable Sequence Lengths**

   Usually, the data set is split into multiple sequences based on a fixed sequence length. Doing so can be disadvantageous to words early in the sequence, as there might not be sufficient context for it because of the splitting; though it would actually have a context in the data set. Usually training happens for several epochs and multiple sequences are batched together and are processed in parallel when GPUs are used.

[8] We know that a neural network is composed of several layers. As LSTM is a part of a bigger network with other layers, it is mentioned as a layer.

[9] During training, processing all examples in a training set once is usually called one epoch.

[10] In addition to the training and the test set, there is also another distinct split from the data set called the validation set. One of the purpose in having a validation set is to adjust the hyperparameters used by the model.

Hence, every time when a batch is created, instead of using the same sequence length, the sequence length is reduced or expanded from a base sequence length randomly.

3. **Leveraging DropConnect**

   DropConnect is applied on hidden-to-hidden weight matrices $\mathbf{U^i}, \mathbf{U^f}, \mathbf{U^o}, \mathbf{U^c}$ within the LSTM. This helps to prevent overfitting occuring on the recurrent connections of the LSTM.

4. **Same Dropout Mask**

   In a standard dropout, new binary dropout masks[11] are sampled every time when a dropout function is called. Whereas, in an AWD-LSTM, the same dropout masks are used for the inputs and the outputs of an LSTM layer for all training examples in a mini-batch.

5. **Embedding Dropout**

   Dropout is applied on the word embedding matrix in such a way that all occurrences of the dropped out words in the sequence under consideration are represented as a zero vector.

6. **Weight Tying**

   In the context of language modeling literature, weight tying means the weights of embedding and softmax layers are shared. This makes a significant reduction in the total number of learnable parameters for any model. Inan et al. [34] and Press and Wolf [60] were the ones who introduced weight tying and had also reported its performance benefits when applied to language models.

7. **Independent Embedding and Hidden Dimensions**

   Before AWD-LSTM, most stacked LSTM-based language models had the same dimension for the embedding vectors and the hidden state of all the LSTM layers. In AWD-LSTM, only the input and output dimensions of the first and last layer in the stack have the same dimension as that of the embedding.

8. **Activation Regularization and Temporal Activation Regularization**

   Activation regularization (AR) is the use of L2 regularization on individual activations. Similarly, Temporal Activation Regularization (TAR) is the use of L2 regularization

[11] Binary masks are matrices with either 0's or 1's that is multiplied elementwise with outputs of the layers to which dropout needs to be applied. The 0's ensure that certain outputs are set to zero.

on difference between outputs from LSTM layers at successive time steps.

$$\text{AR} = \alpha||\mathbf{m} \odot \mathbf{h}_t||_2 \tag{2.20}$$

$$\text{TAR} = \beta||\mathbf{h}_t - \mathbf{h}_{t-1}||_2 \tag{2.21}$$

where $\mathbf{m}$ is the dropout mask and α, β are scaling factors.

2.2.2.4 Evaluation of a Language Model

There are two kinds of evaluations to test the performance of a language model [39]:

- *Extrinsic evaluation*: An end-to-end of evaluation of a language model embedded within any application of interest. This could be the ideal way to know whether the changes made to a language model actually help the application. However, it is hard to do and depends on details of the overall system which may be irrelevant to language modeling [17]. Also, there is a huge time consumption involved in such evaluations, as neural language models themselves are very time consuming to train.
- *Intrinsic evaluation*: Metrics that quickly evaluate potential improvements in a language model. Such metrics measure the performance of a language model independent of any application. *Perplexity* (ppl) is one such metric which is the most commonly-used metric to evaluate language models. Usually, the performance of a language model is measured on the test set. Assuming that the test set $\mathcal{T}$ contains S words $w_1, w_2, .., w_S$, the perplexity of a language model on $\mathcal{T}$ is the inverse probability of $\mathcal{T}$ normalized by S [39],

$$\text{ppl}(\mathcal{T}) = P\big(w_1, w_2, .., w_S\big)^{-\frac{1}{S}} \tag{2.22}$$

$$= \sqrt[S]{\frac{1}{P(w_1, w_2, .., w_S)}} \tag{2.23}$$

$$= \sqrt[S]{\frac{1}{\prod_{i=1}^{S} P(w_i|w_1, .., w_{i-1})}} \tag{2.24}$$

Lower the perplexity on test set, better the performance of a language model.

2.3 Singular Value Decomposition

Singular Value Decomposition (SVD) [14] of any $m \times n$ real matrix $\mathbf{A}$ is the decomposition or factorization of $\mathbf{A}$ into a product of matrices $\mathbf{USV}^T$, where $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{S} \in \mathbb{R}^{m \times n}$, and $\mathbf{V}^T \in \mathbb{R}^{n \times n}$. $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{S}$ is a diagonal matrix. The diagonal elements of $\mathbf{S}$ are known as singular values and the first $\min(m, n)$ columns of $\mathbf{U}$ and $\mathbf{V}$ are known as the left and the right singular vectors of $\mathbf{A}$.

SVD [14] is used in diverse applications including image compression, facial recognition, principal component analysis, latent semantic analysis, and computing the numerical rank of a matrix. However, we are only interested in the computation of the numerical rank of a matrix using SVD.

The number of linearly-independent column vectors in a matrix $\mathbf{A}$ is known as the rank of that matrix, which is numerically equal to the number of non-zero singular values present along the diagonal of $\mathbf{S}$. The above-mentioned SVD decomposition is also called Full SVD. However, in practice it is not always required to perform a Full SVD. There are alternatives (collectively known as Reduced SVD) such as:

- *Thin SVD*: It is known that the rank of any $m \times n$ matrix can at most be equal to $\min(m, n)$. Thin SVD makes use of this fact by performing SVD as follows:

$$\mathbf{A} = \mathbf{U}_n \mathbf{S}_n \mathbf{V}^T \tag{2.25}$$

  where $\mathbf{U}_n \in \mathbb{R}^{m \times n}$ and $\mathbf{S}_n \in \mathbb{R}^{n \times n}$

- *Truncated SVD*: When we are interested in a k-rank approximation $\widetilde{\mathbf{A}}$ to the original matrix $\mathbf{A}$, we need only the top-k singular values and their corresponding left and right singular vectors. In this case, the decomposition is as follows:

$$\widetilde{\mathbf{A}} = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^T \tag{2.26}$$

  where $\mathbf{U}_k \in \mathbb{R}^{m \times k}$, $\mathbf{S}_k \in \mathbb{R}^{k \times k}$, and $\mathbf{V}^T \in \mathbb{R}^{k \times n}$

In the literature, there are several algorithms to compute SVD of a matrix. However in practice, there are a very few algorithms that are widely used to calculate SVD. The most used SVD implementation is the routine available in the Linear Algebra PACKage (LAPACK) [3]. Many popular high-level software libraries including NumPy [57] and SciPy [79] use the same LAPACK routine under their hoods. The major building blocks of the

routine are QR decomposition[12], bidiagonalization[13,14], and QR iteration[15] which results in the desired singular vectors and values. There is also a variant to this routine in which a Divide and Conquer strategy is used instead of QR iteration thus offering a significant speed-up. Halko et al. [29] introduced a randomized algorithm for the computation of Truncated SVD which is made up of two stages on a high-level: 1) Computing an approximate basis[16] for the range[17] of input matrix $\mathbf{A}$ such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A}$ and 2) Using the basis matrix $\mathbf{Q}$ to aid the standard QR decompositions that are done while performing SVD of $\mathbf{A}$.

Once we get the singular values $\mathbf{S}$ from the SVD decomposition of $\mathbf{A}$, there is a need to select a suitable threshold or tolerance ζ to detect the non-zero singular values due to floating point round-off errors. There are two commonly-used thresholds:

1. $\zeta = \max(\mathbf{S}) * \max(M, N) * eps$[18]. It is the default threshold used in the rank(.) routines available in NumPy and MATLAB [47] that is recommended for small-size matrices.

2. Based on expected round-off errors, Press et al. [61] suggested the use of an alternate threshold $\zeta = 0.5 * \sqrt{M + N + 1} * \max(\mathbf{S}) * eps$. It is used when SVD is performed for large real-valued matrices as it would more often encounter round-off errors. Also, NumPy recommends this threshold for large matrices.

2.4 Softmax Bottleneck

2.4.1 Natural Language Is High Rank

Yang et al. [86] hypothesized that "natural language is high rank" by casting language

[12]The decomposition of a matrix $\mathbf{A}$ into a product $\mathbf{A} = \mathbf{QR}$ is known as QR decomposition where $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{m \times n}$ is an upper triangular matrix

[13]A matrix that has only two diagonals with non-zero entries is known as a bidiagonal matrix. In addition to the main diagonal, the other non-zero diagonal can be either above the main diagonal or below the main diagonal.

[14]The decomposition of a matrix $\mathbf{A}$ to a bidiagonal form $\mathbf{A} = \mathbf{UBV}^T$ where $\mathbf{U}, \mathbf{V}$ are orthogonal matrices and $\mathbf{B}$ is a bidiagonal matrix.

[15]After initial QR decomposition of $\mathbf{A} = \mathbf{Q}_0\mathbf{R}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$, and so on are calculated as $\mathbf{A}_{k+1} = \mathbf{Q}_k\mathbf{R}_k$. Such an iteration is known as QR iteration. The number of iterations contributes to the quality of the decomposition.

[16]Basis is a set of linearly independent vectors that span (set of all linear combinations) a vector space.

[17]Range or column space of a matrix $\mathbf{A}$ is the span of its column vectors.

[18]It is known as machine epsilon which is an upper bound on the relative error due to rounding in floating point arithmetic.

modeling as a matrix factorization problem. Assume there is a set of m contexts (or histories) $\{c_1, c_2, .., c_m\}$, and a set of next-token[19] probabilities conditioned on any context c_i, i.e., $\{P(w_1|c_i),\ P(w_2|c_i),\ ..,\ P(w_n|c_i)\}$ which we write as $P(W|c_i)$. They considered natural language as a finite set of pairs of a context and its true conditional next-token distribution, i.e.,

$$\mathcal{L} = \{(c_1, P^*(W|c_1)), (c_2, P^*(W|c_2), .., (c_m, P^*(W|c_m))\}$$

For this formulation, the objective of a language model is to learn a model distribution $P_\theta(W|C)$ to be as close as possible to the true distribution $P^*(W|C)$.

Generally, in language models, a softmax layer is used to get $P_\theta(W|C)$. We can break down the softmax layer in terms of hidden state vectors (or context vectors) and embedding vectors:

$$P_\theta(w_z|c_j) = \text{softmax}(l_{j,z}) = \frac{\exp(l_{j,z})}{\sum_{i=i}^{N} \exp(l_{j,i})} = \frac{\exp(\mathbf{h}_{c_j}^T \mathbf{e}_z)}{\sum_{i=1}^{N} \exp(\mathbf{h}_{c_j}^T \mathbf{e}_i)}$$

where $\mathbf{h}_{c_j} = f(c_j; \theta)$ and $\mathbf{e}_z = g(\mathbf{w}_z; \theta)$. In a typical setting, $f(.)$ is the composition of recurrent layers and $g(.)$ is an indexing function to weights in softmax layer (i.e., an embedding lookup). The scalar $\mathbf{h}_{c_j}^T \mathbf{e}_z$ is also known as logit $l_{j,z}$. The dimension d is the same for both context and embedding vectors.

To bring in the matrix factorization perspective, consider the following matrices:

$$\mathbf{H}_\theta = \begin{bmatrix} \mathbf{h}_{c_1}^T \\ \mathbf{h}_{c_2}^T \\ . \\ . \\ \mathbf{h}_{c_M}^T \end{bmatrix}, \mathbf{E}_\theta = \begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \\ . \\ . \\ \mathbf{e}_N^T \end{bmatrix}, \mathbf{Q}_\theta = \begin{bmatrix} \log(P_\theta(\mathbf{w}|c_1)) \\ \log(P_\theta(\mathbf{w}|c_2)) \\ . \\ . \\ \log(P_\theta(\mathbf{w}|c_M)) \end{bmatrix}$$

where $\mathbf{H}_\theta \in \mathbb{R}^{M \times d}$, $\mathbf{E}_\theta \in \mathbb{R}^{N \times d}$, and $\mathbf{Q}_\theta \in \mathbb{R}^{M \times N}$. As cross-entropy loss is the de-facto loss function used for word-level language models, log probabilities as in $\mathbf{Q}_\theta$ are considered.

$$\mathbf{Q}_\theta = \log(P_\theta(W|C)) = \log(\text{softmax}(\mathbf{H}_\theta \mathbf{E}_\theta^\mathbf{T}))$$

Let's assume the true equivalent to $\mathbf{Q}_\theta$ as $\mathbf{Q}^* = \log(P^*(W|C))$. Yang et al. [86] hypothesize that $\mathbf{Q}^*$ has to be a high-rank matrix as natural language is highly context-dependent [50].

[19]Tokenization is the process of breaking a text into smaller units called tokens. In our case, as we focus only on word-level language models, the tokens include: words, punctuation, numbers, symbols, and other special tokens. Hence, we use the general term "token" instead of "word" to be consistent with the literature wherever it is needed.

2.4.2 The Limitation of Softmax Function

As said in Section 2.4.1, according to Yang et al. [86]'s hypothesis, $\mathbf{Q}^*$ has to be high-rank. Hence, the question now is whether the rank of $\mathbf{Q}_\theta$ can be as high as that of $\mathbf{Q}^*$. Below are the steps that help in determining the rank of $\mathbf{Q}_\theta$.

$$\text{softmax}(\mathbf{H}_\theta \mathbf{E}_\theta^{\mathbf{T}}) = \begin{bmatrix} \frac{\exp(l_{1,1})}{\sum_{i=i}^{N}\exp(l_{1,i})} & \frac{\exp(l_{1,2})}{\sum_{i=i}^{N}\exp(l_{1,i})} & . & . & \frac{\exp(l_{1,N})}{\sum_{i=i}^{N}\exp(l_{1,i})} \\ \frac{\exp(l_{2,1})}{\sum_{i=i}^{N}\exp(l_{2,i})} & \frac{\exp(l_{2,2})}{\sum_{i=i}^{N}\exp(l_{2,i})} & . & . & \frac{\exp(l_{2,N})}{\sum_{i=i}^{N}\exp(l_{2,i})} \\ . & . & . & . & . \\ . & . & . & . & . \\ \frac{\exp(l_{M,1})}{\sum_{i=i}^{N}\exp(l_{M,i})} & \frac{\exp(l_{M,2})}{\sum_{i=i}^{N}\exp(l_{M,i})} & . & . & \frac{\exp(l_{M,N})}{\sum_{i=i}^{N}\exp(l_{M,i})} \end{bmatrix}$$

By replacing the normalization constant as nc_j, we get:

$$= \begin{bmatrix} \frac{\exp(l_{1,1})}{nc_1} & \frac{\exp(l_{1,2})}{nc_1} & . & . & \frac{\exp(l_{1,N})}{nc_1} \\ \frac{\exp(l_{2,1})}{nc_2} & \frac{\exp(l_{2,2})}{nc_2} & . & . & \frac{\exp(l_{2,N})}{nc_2} \\ . & . & . & . & . \\ . & . & . & . & . \\ \frac{\exp(l_{M,1})}{nc_M} & \frac{\exp(l_{M,2})}{nc_M} & . & . & \frac{\exp(l_{M,N})}{nc_M} \end{bmatrix}$$

Now, when we take an elementwise log, we get:

$$\log(\text{softmax}(\mathbf{H}_\theta \mathbf{E}_\theta^{\mathbf{T}})) = \mathbf{Q}_\theta = \begin{bmatrix} l_{1,1}+nk_1 & l_{1,2}+nk_1 & . & . & l_{1,N}+nk_1 \\ l_{2,1}+nk_2 & l_{2,2}+nk_2 & . & . & l_{2,N}+nk_2 \\ . & . & . & . & . \\ . & . & . & . & . \\ l_{M,1}+nk_M & l_{M,2}+nk_M & . & . & l_{M,N}+nk_M \end{bmatrix}$$

where $nk_j = -\log(nc_j)$. We get the above formulation because of $\log(\frac{a}{b}) = \log(a) - \log(b)$ and $\log(\exp(a)) = a$

Expressing $\mathbf{Q}_\theta$ as a sum of two matrices:

$$\mathbf{Q}_\theta = \begin{bmatrix} l_{1,1} & l_{1,2} & . & . & l_{1,N} \\ l_{2,1} & l_{2,2} & . & . & l_{2,N} \\ . & . & . & . & . \\ . & . & . & . & . \\ l_{M,1} & l_{M,2} & . & . & l_{M,N} \end{bmatrix} + \begin{bmatrix} nk_1 & nk_1 & . & . & nk_1 \\ nk_2 & nk_2 & . & . & nk_2 \\ . & . & . & . & . \\ . & . & . & . & . \\ nk_M & nk_M & . & . & nk_M \end{bmatrix}$$

For any two matrices $\mathbf{A}, \mathbf{B}$ of same dimension, $\text{rank}(\mathbf{A}+\mathbf{B}) \leq \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B})$. Also, for any two matrices $\mathbf{A}$ of dimension $m \times n$ and $\mathbf{B}$ of dimension $n \times k$, $\text{rank}(\mathbf{AB}) \leq \min(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B}))$. By using these two inequalities, we get:

$$\text{rank}(\mathbf{Q}_\theta) \leq d + 1$$

The rank of $\mathbf{Q}_\theta$ is bottlenecked by the embedding dimension d when softmax is used to compute probabilities as it supposedly turns out that $\log(\text{softmax}(.))$ is a linear transformation. Hence, the rank of $\mathbf{Q}_\theta$ cannot be as high as $\mathbf{Q}^*$. This phenomenon is known as the *softmax bottleneck*. The authors of [86] claimed that such softmax based models lack enough expressivity to model a highly context-dependent natural language.

2.4.3 Revisiting the Hypothesis

As the true distribution P^* is generally unknown, it is difficult for anyone (including Yang et al. [86]) to rigorously prove the hypothesis that $\mathbf{Q}^*$ has to be a high-rank matrix. However, Yang et al. [86] intuitively made a few supporting justifications as follows:

- Natural language $\mathcal{L}$ is highly context-dependent [50]. The token "north" is likely to be followed by "korea" or "korean" in news articles on international politics which however is unlikely in a textbook on U.S. domestic history.
- If $\mathbf{Q}^*$ is low rank, it means humans need only a limited number of semantic bases, and all context-dependent semantic meanings can be spanned from them. However, it is hard to find a natural concept in linguistics and cognitive science that corresponds to such bases, which questions the existence of such bases.

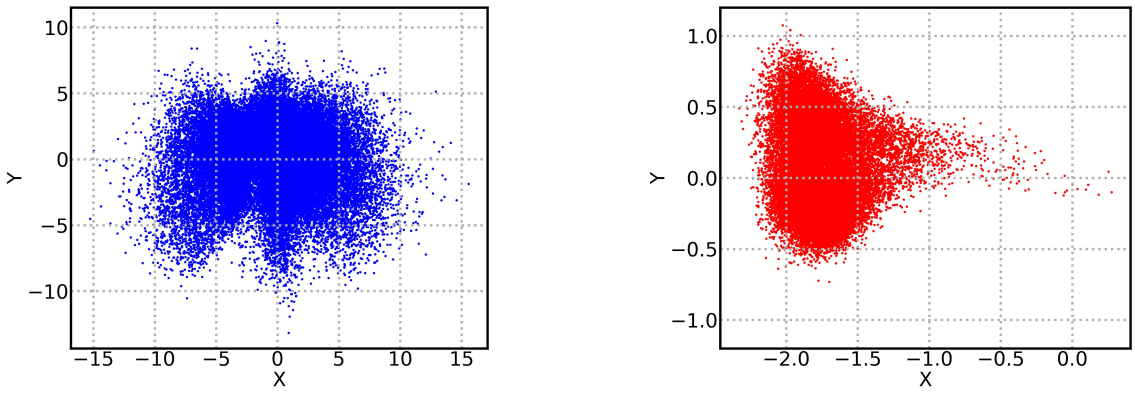
Though the above-mentioned justifications might seem acceptable to a certain degree, we think that it is not convincing enough, because, there could be other implementation-specific details which may deteriorate these justifications. An example is the length of context under consideration. If only a local context[20] is considered, there can be several local contexts which are similar, for which only a few bases would suffice, even though the global context surrounding those local contexts might be different.

Having expressed our concerns, we still accept the hypothesis as we are more focused on an in-depth empirical study of the softmax bottleneck and its suggested connections with the performance of a word-level recurrent language model.

[20]Local vs Global context: For example, a bigger scope of text like a paragraph or a document can be thought of as a *global* context. Similarly, one or few sentences within a global context can be thought of as a *local* context.

2.5 Representation Degeneration

There is another problem in the broad context of language generation tasks (machine translation, language modeling, etc.) that is similar to the problem of *softmax bottleneck* in terms of the basic fundamentals. Gao et al. [23] studied learned models for language generation by looking at the 2D projection of the weights of the embedding layer. They found that the learned word embeddings in a Transformer model[21] degenerated into a narrow cone (Figure 2.8b) in contrast to the embeddings learned in a word2vec model [52] which were diversely distributed around the origin (Figure 2.8a). This problem is known as the *representation degeneration*.



(a) word2vec

(b) Transformer

Figure 2.8: 2D projection of learned word embeddings from different models. Image extracted from [23].

A strong connection between the decay rate of singular values of the embedding matrix and the above described cone-shaped embedding projections was observed in the works of [23, 82]. Specifically, when the singular values tend to drop very fast, it has been believed that this could lead to such a shape. Wang et al. [82] described it as the phenomenon of *fast singular value decay*.

[21] Transformer [78, 73] is a neural network model mainly introduced for sequence processing tasks that is completely different from RNN-type models. It is based on the idea of self-attention which is the ability to directly look at different positions in the input sequence to compute a representation for the sequence. In the context of a language sequence, it does not operate word by word like an RNN but operates on a set of words at a time.

Chapter 3

Related Work

3.1 Breaking the Softmax Bottleneck

Almost all the existing works to break the softmax bottleneck in language models, which we will discuss in this section have used the Penn TreeBank (PTB) [46] and WikiText-2 (WT2) [48] data sets for their experiments. We will discuss these data sets in detail in the next chapter (Section 4.3.1). For now, we will just report the results of existing works on these two data sets.

3.1.1 The Two Main Directions

As discussed in Section 2.4.2, the log(softmax(.)) function can at most increase the rank of any matrix by 1. Hence, the two main directions to break the bottleneck, based on the above-mentioned fact, are:

1. Replacing log(softmax(.)) with log(f(.)), where f(.) can be any function that can output a probability distribution, so that $\mathbf{Q}_\theta$ can arbitrarily be high rank. Yang et al. [86] and Kanai et al. [40] took this direction.

2. Introducing non-linear activation functions which can increase the rank of $\mathbf{H}_\theta \mathbf{E}_\theta^T$ before log(softmax(.)) is applied. Ganea et al. [22] followed this direction.

3.1.2 Mixture of Softmaxes

Model	#Param	Validation ppl	Test ppl	Rank
		PTB		
AWD-LSTM	24M	60.70	58.80	400
AWD-LSTM-MoS	22M	58.08	55.97	9,981
AWD-LSTM-MoC	22M	59.82	57.55	280
		WT2		
AWD-LSTM	33M	69.10	66.00	—
AWD-LSTM-MoS	35M	66.01	63.33	—
AWD-LSTM-MoC	35M	68.76	65.98	—

Table 3.1: Reported results by the authors of MoS [86].

Yang et al. [86], who initially observed the phenomenon of the softmax bottleneck, had also proposed a solution to break the bottleneck. It is known as *Mixture of Softmaxes* (MoS), in which the conditional next-token distribution is expressed as:

$$P_\theta(w_z|\mathbf{c}_j) = \sum_{k=1}^{K} \pi_{c_j,k} \frac{\exp(\mathbf{h}_{c_j,k}^T \mathbf{e}_z)}{\sum_{i=1}^{N} \exp(\mathbf{h}_{c_j,k}^T \mathbf{e}_i)} \quad s.t \quad \sum_{k=1}^{K} \pi_{c_j,k} = 1 \tag{3.1}$$

where K is the total number of mixture components, $\mathbf{h}_{c_j,k}$ is the k-th transformed context vector, and $\pi_{c_j,k}$ is the mixing ratio for $\mathbf{h}_{c_j,k}$ given a context $\mathbf{c}_j$. The context vector $\mathbf{h}_{c_j}$ from the composition of recurrent layers is further transformed into k context vectors by a fully-connected layer $\mathbb{R}^{d_1} \rightarrow \mathbb{R}^{K \times d}$ with a non-linear activation function, tanh. The mixing ratios $\boldsymbol{\pi}$ are also learned. For language modeling experiments, the authors of [86] had replaced the softmax function in the AWD-LSTM with MoS. A small but an important component in the network topology that compromises one of the characteristics of the AWD-LSTM is the above-mentioned fully-connected layer. Note that $\mathbf{h}_{c_j}$ does not necessarily have to be of dimension d anymore. It is only the dimension of $\mathbf{h}_{c_j,k}$ that should be d.

Because of the formulation of MoS, as seen in equation 3.1, $\mathbf{Q}_\theta$ is modeled as:

$$\mathbf{Q}_\theta = \log \sum_{k=1}^{K} \boldsymbol{\Pi}_k \exp(\mathbf{H}_{\theta,k} \mathbf{E}_\theta^T) \tag{3.2}$$

where $\mathbf{\Pi}_k \in \mathbb{R}^{n \times n}$ is a diagonal matrix that contains mixture weights $\pi_{c_j,k}$ along its diagonal. Because of the fact that log(MoS(.)) is a non-linear function (log-sum-exp), $\mathbf{Q}_\theta$ can be high rank.

The authors of [86] had also introduced another model called *Mixture of Contexts* (MoC), which has the same parameterization as that of MoS, but the mixing is done in the context vector space instead of the probability space. Hence, $\mathbf{Q}_\theta$ is still bottlenecked by the embedding dimension d. In MoC, the next-token conditional distribution is expressed as:

$$P_\theta(w_z|\mathbf{c}_j) = \frac{\exp((\sum_{k=1}^{K} \pi_{c_j,k}\, \mathbf{h}_{c_j,k}^T)\, \mathbf{e}_z)}{\sum_{i=1}^{N} \exp((\sum_{k=1}^{K} \pi_{c_j,k}\, \mathbf{h}_{c_j,k}^T)\, \mathbf{e}_i)} \;\; s.t \;\; \sum_{k=1}^{K} \pi_{c_j,k} = 1 \tag{3.3}$$

3.1.3 SigSoftmax

Model	#Param	Validation ppl	Test ppl	Rank
		PTB		
AWD-LSTM	24M	61.01±0.40	58.80±0.40	402
AWD-LSTM-SS	24M	61.01±0.20	58.40±0.20	4,640
		WT2		
AWD-LSTM	33M	68.00±0.20	65.20±0.20	402
AWD-LSTM-SS	33M	67.80±0.10	65.00±0.20	5,465

Table 3.2: Reported results by the authors of SS [40].

Kanai et al. [40] had proposed alternatives to the exponential function (remember that the softmax function has the exponential function under its hood) to get a probability distribution. They had experimented with other well-known non-linear functions such as sigmoid and ReLU. They proposed a general function definition $f(.)$ as an alternative to softmax as follows:

$$f(l_z) = \frac{g(l_z)}{\sum_{i=1}^{N} g(l_i)} \tag{3.4}$$

such that the chosen $g(.)$ and $f(.)$ should possess the following properties:

1. log($g(.)$) should be non-linear.

2. Numerically-stable gradients for optimization when $f(.)$ is used.

3. The range of $f(.)$ should be $[0, 1]$.

4. $g(.)$ should be monotonically increasing.

Also, the authors of [40] proposed a novel function which is a product of two well-known non-linear functions, i.e., exponential and sigmoid. It is known as *SigSoftmax* (SS) which is defined as follows:

$$\text{SS}(l_z) = \frac{\exp(l_z)\sigma(l_z)}{\sum_{i=1}^{N} \exp(l_i)\sigma(l_i)} \tag{3.5}$$

They had also showed that SS(.) possess the properties of $f(.)$, and $\exp(.)\sigma(.)$ satisfies the properties of $g(.)$. Instead of reporting the results for only one random initialization of model parameters, the model was trained five times using different initialization, and the mean and the standard deviation of validation and test perplexities were reported as shown in Table 3.2.

3.1.4 Linear Monotonic Softmax with Piecewise Linear Increasing Functions

Model	**#Param**	**Validation ppl**	**Test ppl**	**Rank**
		PTB		
AWD-LSTM	24.2M	60.83	58.37	—
AWD-LSTM-LMS-PLIF	24.4M	59.45	57.25	—
		WT2		
AWD-LSTM	33M	68.11	65.22	—
AWD-LSTM-LMS-PLIF	33.2M	67.87	64.86	—

Table 3.3: Reported results by the authors of LMS-PLIF [22].

Ganea et al. [22] defined Linear Monotonic Softmax (LMS) layer as follows:

$$P_\theta(w_z|\mathbf{c}_j) = \frac{\exp(f(\mathbf{h}_{c_j}^T \mathbf{e}_z))}{\sum_{i=1}^{N} \exp(f(\mathbf{h}_{c_j}^T \mathbf{e}_i))} \tag{3.6}$$

In equation 3.6, the main contribution was to learn the pointwise non-linear function $f(.)$ to alleviate softmax bottleneck. The authors restricted $f(.)$ to possess the following properties:

1. Non-linearity: to not limit the rank to d.
2. Increasing: to preserve the ordering of logits l_z.
3. Continuous and piecewise differentiable (almost everywhere).
4. Fast and memory efficient.

They constructed a learnable pointwise function called Piecewise Linear Increasing Functions (PLIF) as follows:

- First, they fix a large enough interval $[-T, T]$ that can accommodate the range of l_z during the process of training and testing, and K equidistant intervals in $[-T, T]$ such that $k_i = -T + \frac{2Ti}{K}$; $1 \leq i \leq K + 1$
- Then, they defined a piecewise linear function:

$$f(l_z) = s_i l_z + b_i \quad \forall l_z \in [k_i, k_{i+1}] \tag{3.7}$$

  where s_i is the slope of the linear function on the interval $[k_i, k_{i+1}]$. To enforce $s_i > 0$ for the purpose of preserving the ordering of logits, they made $s_i = \text{softplus}(v_i)$ where v_i is an unconstrained parameter.

- Finally, to make the function continuous, i.e., $f(.)$ to have same value at k_i in adjacent sub-intervals $[k_{i-1}, k_i]$ and $[k_i, k_{i+1}]$, they used the following formulation $b_i = b_0 + s_0 k_0 - s_i k_i + \frac{2T}{K} \sum_{j=0}^{i-1} s_j$; $\forall i > 0$

3.1.5 Mixtape

Model	Test ppl	Rank
PTB		
AWD-LSTM	59.19	—
AWD-LSTM-MoS	56.14	—
AWD-LSTM-Mixtape	56.37	—

Table 3.4: Reported results by the authors of Mixtape [87].

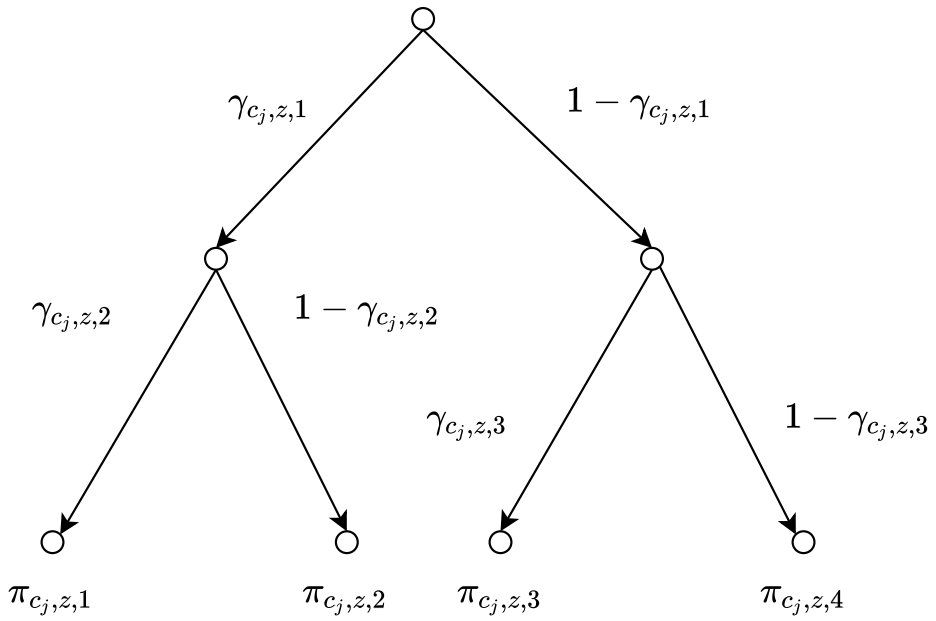


Figure 3.1: Sigmoid tree decomposition. Image adapted from [87].

Generally, softmax(.) is considered to be a computationally-intensive operation as it involves applying exp(.) to each token in the vocabulary. The intensiveness is directly proportional to the size of the vocabulary. MoS performs even worse as it has K softmax(.) computations. Also, it is highly memory-intensive on GPUs.

To overcome the above-mentioned efficiency issues, Yang et al. [87] introduced a novel method called Mixtape which is able to produce a high-rank $\mathbf{Q}_\theta$ like MoS but is better than MoS in terms of efficiency. In Mixtape, the next-token conditional distribution is expressed as:

$$P_\theta(w_z|\mathbf{c}_j) = \frac{\exp(\sum_{k=1}^{K} \pi_{c_j,z,k}\, \mathbf{h}_{c_j,k}^T\, \mathbf{e}_z)}{\sum_{i=1}^{N} \exp(\sum_{k=1}^{K} \pi_{c_j,i,k}\, \mathbf{h}_{c_j,k}^T\, \mathbf{e}_i)} \;\; s.t \;\; \sum_{k=1}^{K} \pi_{c_j,i,k} = 1, \;\; 1 \leq i \leq N \tag{3.8}$$

We can see a similarity between MoC, as seen in the equation 3.3, and Mixtape, as seen in the equation above. The key difference is the use of mixture weights (or priors) $\pi_{c_j,k,i}$ for every token in the vocabulary. Because of the above formulation $\mathbf{Q}_\theta$ is modeled as:

$$\mathbf{Q}_\theta = \sum_{k=1}^{K} \mathbf{\Pi}_k \odot (\mathbf{H}_{\theta,k}\mathbf{E}_\theta^T) \tag{3.9}$$

It is due to the element-wise multiplication $\odot$ as seen in the equation 3.9, $\mathbf{Q}_\theta$ can be high-rank. However, a naive implementation of the equation 3.8 will result in longer training and prediction time than that of MoS. This is mainly due to the calculation of K priors for every token in the vocabulary, i.e., K softmax(.) for every token in the vocabulary. To address this inefficiency, the authors of [87] had also introduced a technique called *sigmoid tree decomposition* (as seen in Figure 3.1) to compute the priors efficiently. The technique is explained below:

Let $a_{c_j,z,k}$ be the pre-activation priors on which softmax(.) is applied i.e.,

$$\pi_{c_j,z,k} = \frac{\exp(a_{c_j,z,k})}{\sum_{k'=1}^{K} \exp(a_{c_j,z,k})} \tag{3.10}$$

Instead of using softmax(.), an alternative approach using sigmoid(.) was introduced by the authors of [87]:

$$\begin{aligned}
\gamma_{c_j,z,k} &= \sigma(a_{c_j,z,k}) \text{ for } 1 \leq k \leq K-1 \\
\pi_{c_j,z,1} &= \gamma_{c_j,z,1}\gamma_{c_j,z,2} \\
\pi_{c_j,z,2} &= \gamma_{c_j,z,1}(1-\gamma_{c_j,z,2}) \\
\pi_{c_j,z,3} &= (1-\gamma_{c_j,z,1})\gamma_{c_j,z,3} \\
\pi_{c_j,z,4} &= (1-\gamma_{c_j,z,1})(1-\gamma_{c_j,z,3})
\end{aligned} \tag{3.11}$$

The pre-activation priors $a_{c_j,z,k}$ are computed as:

$$a_{c_j,z,k} = \mathbf{v}_z^T \tanh(\mathbf{U}_k \mathbf{h}_{c_j}) + \mathbf{u}_k^T \mathbf{h}_{c_j} + b_{z,k} \tag{3.12}$$

where $\mathbf{v}_z \in \mathbb{R}^{d_2}, \mathbf{U}_k \in \mathbb{R}^{d_2 \times d_1}, \mathbf{u}_k \in \mathbb{R}^{d_1}, b_{z,k} \in \mathbb{R}, dim(\mathbf{h}_{c_j})$[1] $= d_1$, and, $d_2 << d$

Similar to MoS, $\mathbf{h}_{c_j,k}$ is computed from $\mathbf{h}_{c_j}$ as follows:

$$\mathbf{h}_{c_j,k} = \tanh(\mathbf{W}_k \mathbf{h}_{c_j}) \tag{3.13}$$

where $\mathbf{W}_k \in \mathbb{R}^{d_1 \times d}$

[1] $dim(.)$ denotes the dimension of a vector

3.1.6 Direct Output Connection

Model	#Param	Validation ppl	Test ppl	Rank
		PTB		
AWD-LSTM	24M	60.00	57.30	401
AWD-LSTM-DOC	23M	54.62	52.87	10,000
		WT2		
AWD-LSTM	33M	68.60	65.80	—
AWD-LSTM-DOC	37M	60.97	58.55	—

Table 3.5: Reported results by the authors of DOC [22].

Takase et al. [72] had proposed a generalization to MoS called Direct Output Connection (DOC). As we know that AWD-LSTM has mutliple LSTM layers in a stacked manner, DOC takes advantage of this architecture. In terms of computing the next-token conditional distribution, the key difference between MoS and DOC is that the former takes into account only the output of the top-most LSTM layer, i.e., $\mathbf{h}_t^3$ as seen in Figure 2.7, whereas DOC uses the outputs from all the hidden layers and embedding layer, i.e., $\mathbf{h}_t^3, \mathbf{h}_t^2, \mathbf{h}_t^1, \mathbf{e}_t$. With all these outputs, DOC computes R probability distributions and takes a weighted sum over them. In DOC, the next-token conditional distribution is expressed as:

$$P_\theta(w_z|\mathbf{c}_j) = \sum_{r=1}^{R} \pi_{c_j,r} \frac{\exp(\mathbf{g}_{c_j,r}^T \mathbf{e}_z)}{\sum_{i=1}^{N} \exp(\mathbf{g}_{c_j,r}^T \mathbf{e}_i)} \text{ s.t. } \sum_{r=1}^{R} \pi_{c_j,r} = 1 \tag{3.14}$$

$$\mathbf{g}_{c_j,r} = \mathbf{W}_{r,i}\, \mathbf{o}_{c_j,i} \text{ for } 0 \leq i \leq 3 \tag{3.15}$$

where $\mathbf{o}_{c_j,0} = \mathbf{e}_t, \mathbf{o}_{c_j,k} = \mathbf{h}_t^k$ for $1 \leq k \leq 3$ and $\mathbf{W}_{r,i} \in \mathbb{R}^{d \times dim(\mathbf{o}_{c_j,i})}$

However, for the calculation of mixture weights $\boldsymbol{\pi}_{c_j}$, only the output from the top-most LSTM layer is used:

$$\boldsymbol{\pi}_{c_j} = \text{softmax}(\mathbf{U}\, \mathbf{o}_{c_j,3}) \tag{3.16}$$

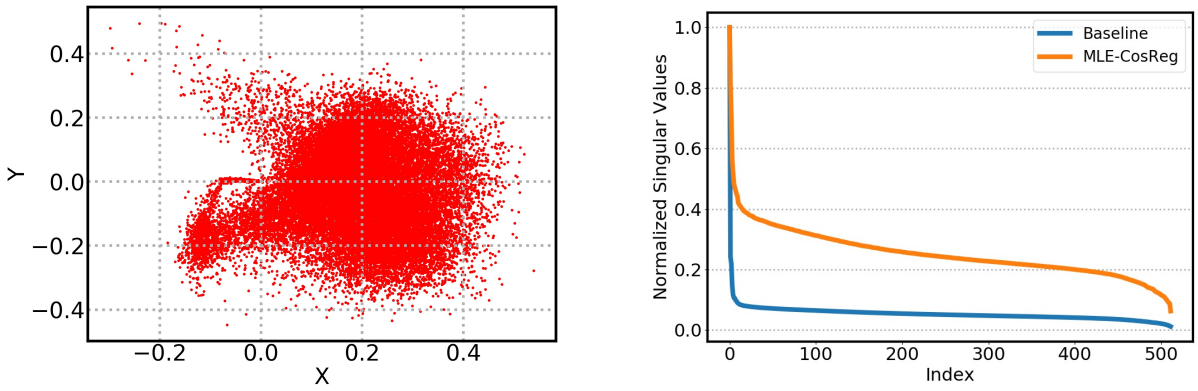
where $\mathbf{U} \in \mathbb{R}^{r \times dim(\mathbf{o}_{c_j,3})}$. In addition, let v_i be the number of $\mathbf{g}_{c_j}$ vectors computed from $\mathbf{o}_{c_j,i}$. Then R is actually defined as the sum of v_i for all such i, i.e., $R = \sum_{i=0}^{3} v_i$. When

$R = v_3$, DOC is equivalent to MoS. After experimenting with different combinations of values for each v_i, the authors of [72] reported $R = 20$ with $v_3 = 15, v_2 = 5$, and $v_1 = v_0 = 0$ as the best performing configuration.

3.2 Alleviating Representation Degeneration

Though existing works report results on both machine translation and language modeling, we mostly focus only on their language modeling experiments using the AWD-LSTM model. We look at their machine translation experiments, but only in the context of representation degeneration, and not on any task-specific performance improvements. Also, we report their language modeling results in terms of performance improvements only on PTB and/or WT2 datasets, as few had also considered other data sets.

3.2.1 Cosine Regularization



(a) Diversely distributed embeddings around the origin.

(b) Slow vs Fast singular value decay.

Figure 3.2: The effect of using MLE-CosReg. Images extracted from [23].

After observing the 2D projections of embedding vectors degenerating into narrow cone (Figure 2.8b), Gao et al. [23] introduced a straightforward approach to improve the angle between boundaries of the cone.

Model	#Param	Validation ppl	Test ppl
	WT2		
AWD-LSTM	33M	69.10	66.00
AWD-LSTM + MLE-CosReg	33M	68.20	65.20

Table 3.6: Reported results by the authors of MLE-CosReg [23].

To do so, they relied on the idea of cosine similarities between any two embeddings. For any embedding vector $\mathbf{e}_i$, let $\hat{\mathbf{e}}_i$ be its unit vector i.e., $\hat{\mathbf{e}}_i = \frac{\mathbf{e}_i}{||\mathbf{e}_i||}$. They introduced a regularization term in addition to the original maximum likelihood (MLE) objective:

$$J_{\text{MLE-CosReg}}(\boldsymbol{\theta}) = J_{\text{MLE}}(\boldsymbol{\theta}) + \gamma \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \hat{\mathbf{e}}_i \hat{\mathbf{e}}_j \tag{3.17}$$

where γ is a scaling factor which is a hyperparameter.

With this new objective function called MLE-CosReg, they were able to avoid the degeneration in machine translation as the projections were diversely distributed around the origin (Figure 3.2a). They had also noticed that the singular values of embedding matrix decayed slowly than before (Figure 3.2b). However, it is unclear that a similar behavior can also be seen in language modeling as they did not report any similar figures.

3.2.2 Spectrum Control

Model	#Param	Validation ppl	Test ppl
	WT2		
AWD-LSTM	33M	69.10	66.00
AWD-LSTM + SC	33M	66.30	63.70

Table 3.7: Reported results by the authors of SC [82].

Based on the observations [23, 82] that the decay rate of singular values is correlated with the degeneration of embeddings, Wang et al. [82] introduced an approach that explicitly controls the decay rate of the singular values. To do so, they used an SVD reparameterization trick,

in which the embedding matrix $\mathbf{E}$ is represented by its factorized matrices $\mathbf{USV}^T$. Then, the next-token conditional distribution is expressed as:

$$P_\theta(w_z|\mathbf{c}_j) = \frac{\exp(\mathbf{h}_{c_j,k}^T\,(\mathbf{USV}^T)_z)}{\sum_{i=1}^{N}\exp(\mathbf{h}_{c_j,k}^T(\mathbf{USV}^T)_i)} \tag{3.18}$$

where $(\mathbf{USV}^T)_z$ is read as the z-th row of the matrix product $\mathbf{USV}^T$ which is actually $\mathbf{e}_z$.

To enforce orthonormality constraints, they introduced the following regularization terms:

$$\lambda_r||\mathbf{U}^T\mathbf{U}-\mathbf{I}||_F^2 + \lambda_r||\mathbf{V}^T\mathbf{V}-\mathbf{I}||_F^2 + \lambda_r||\mathbf{U}^T\mathbf{U}-\mathbf{I}||_2^2 + \lambda_r||\mathbf{V}^T\mathbf{V}-\mathbf{I}||_2^2 \tag{3.19}$$

where λ_r for $1 \leq r \leq 4$ are scaling factors of the regularization terms, $||.||_F^2$ is the Frobenius norm, $||.||_2^2$ is spectral norm, and $\mathbf{I}$ is an identity matrix.

Also, to control the decay rate of the singular values (also known as "Spectrum Control" (SC) [37, 82]), they used two types of decay functions: exponential decay and polynomial decay. With any chosen decay function, they make the singular values s_{kk} in $\mathbf{S}$ to have a decay rate similar to that function by introducing an additional regularization term. For an exponential decay function:

$$\lambda_e \sum_{k=1}^{d}(s_{kk} - c_1\exp(-c_2 k^\gamma))^2 \tag{3.20}$$

Similarly, if a polynomial decay function is chosen:

$$\lambda_p \sum_{k=1}^{d}(s_{kk} - c_1 k^\gamma)^2 \tag{3.21}$$

where λ_e, λ_p are scaling factors of the respective regularization terms, c_1, c_2 are some standard constants, and γ is the decay rate.

They had experimented with both exponential and polynomial decay by using different decay rates, and had reported that an exponential decay worked well for the AWD-LSTM model and a polynomial decay worked well for the Transformer model. Though they have shown improvements in test perplexity (Table 3.7) for the AWD-LSTM model, they did not provide any evidence like differences in the decay rate of the singular values before and after (similar to Figure 3.2b) the use of their spectrum control technique.

3.2.3 Adversarial Softmax

Model	#Param	Validation ppl	Test ppl
	PTB		
AWD-LSTM	24M	60.00	57.30
AWD-LSTM + AdvSoft	24M	57.15	55.01
	WT2		
AWD-LSTM	33M	68.60	65.80
AWD-LSTM + AdvSoft	33M	64.01	61.56

Table 3.8: Reported results by the authors of AdvSoft [81].

Without adding any computational overhead like the regularization terms introduced in MLE-CosReg [23] and SC [82], Wang et al. [81] proposed an adversarial perturbation on the embedding vectors, to prevent their degeneration by promoting their diversity.

Let the parameter vector $\boldsymbol{\theta}$, which includes all the parameters in the network, get split into two as $\boldsymbol{\Omega}$ and $\boldsymbol{\varpi}$, where $\boldsymbol{\varpi}$ contains the embedding matrix flattened as a single vector and $\boldsymbol{\Omega}$ contains the rest of the parameters in the network. The authors of [81] had proposed a modified training objective as follows:

$$\max_{\boldsymbol{\Omega},\boldsymbol{\varpi}} \min_{\boldsymbol{\delta}_{z;j}} \sum_{j} \log P(w_z|c_j;\ \boldsymbol{\Omega}, \mathbf{e}_z + \boldsymbol{\delta}_{z;j})\ \ s.t.\ \ ||\boldsymbol{\delta}_{z;j}|| \leq \epsilon \tag{3.22}$$

where w_z is the word that should be predicted given a context c_j, and $\boldsymbol{\delta}_{z;j}$ is the adversarial perturbation applied to embedding vector $\mathbf{e}_z$ of the word to be predicted. The above objective has an outer maximization (for better log-likelihood) and inner minimization (for worse log-likelihood), unlike a maximum log-likelihood objective which only has the maximization part. Usually, this kind of objective functions (equation 3.22) require outer and inner loops during training. But the authors of [81] had showed that the inner minimization problem in equation 3.22 has a closed form solution:

$$\boldsymbol{\delta}^*_{z;j} = -\epsilon \frac{\mathbf{h}_{c_j}}{||\mathbf{h}_{c_j}||} \tag{3.23}$$

where $\mathbf{h}_{c_j}$ is the context vector which would usually be the output of recurrent layers given a context c_j. The above solution can directly be added to $\mathbf{e}_z$ thus eliminating the need for

a computationally expensive outer and inner loops. When softmax is calculated using such perturbed embeddings, it is known as Adversarial Softmax (AdvSoft).

Though the proposed method is theoretically believed to help in alleviating the representation degeneration problem, the main motivation of the authors was to propose a new regularization technique for neural language models.

Chapter 4

Generalized SigSoftmax

4.1 Motivation

Among several solutions to break the softmax bottleneck, we focus particularly on three fundamentally-different solutions namely Mixture of Softmaxes (MoS), SigSoftmax (SS), and Linear Monotonic Softmax with Piecewise Linear Increasing Functions (LMS-PLIF).

Metrics	**Function**		
	SS	LMS-PLIF	MoS
rank (reported)	4640	—	9980
rank (ours)	4979	580	9979
perplexity difference (reported)	0.40	1.11	2.88
perplexity difference (ours)	−0.06	0.43	1.28

Table 4.1: Perplexity difference (subtracted from the baseline AWD-LSTM's perplexity; positive means better than the baseline and negative means worse) for SS, LMS-PLIF, and MoS models and their respective ranks of the log probability matrix $\mathbf{Q}_\theta$ on the test set of PTB. The reported results in their respective original papers are compared with our reproduced results.

In Table 4.1, if we look at the reported results for perplexity difference and rank of $\mathbf{Q}_\theta$ on test set of PTB data set, SS breaks the softmax bottleneck but its rank is not as high as MoS. Its performance on the test set is also only slightly better than the baseline (Softmax), unlike for MoS. LMS-PLIF, which performs better than SS and Softmax, was believed to produce a high-rank $\mathbf{Q}_\theta$, but its rank was not reported. Hence, we trained all these models, including the baseline Softmax model, using the respective authors' implementations; but we made sure that all the models have same values for their common hyperparameters. Looking at our results, LMS-PLIF actually produces a low-rank $\mathbf{Q}_\theta$, contrary to its purpose; but it still performs better than Softmax and SS. It seems that having a high-rank $\mathbf{Q}_\theta$ does not correlate with better perplexity on the test set, which contradicts the understanding that learning a high-rank language model is necessary to model the context dependencies in natural language. But as the correlation is still present in the case of MoS, we need another function which can produce $\mathbf{Q}_\theta$ with diverse ranks, including ranks in the range that MoS is able to produce. Only then could we make a fair comparison and become clear whether learning a high-rank language model is important or not.

Hence, in this chapter, we introduce Generalized SigSoftmax (GSS), a parameterized function that can produce $\mathbf{Q}_\theta$ with diverse ranks when its parameters are adjusted. Also, as we noticed unfair comparisons that were made in some of the existing works, we attempt to make a fair comparison between models using different functions wherever it is possible. We also perform statistical significance tests wherever it is needed.

4.2 Construction of Generalized SigSoftmax from SigSoftmax

4.2.1 SigSoftmax in Terms of Softmax

A careful observation that needs to be made in the general function definition $f(.)$, which can be an alternative to softmax, as suggested by the authors of SS [40] is $g(.)$ as discussed in Section 3.1.3. It was said that $\log(g(.))$ should be non-linear. We know that for SS(l_z):

$$g(l_z) = \exp(l_z)\,\sigma(l_z) \tag{4.1}$$

$$\log(g(l_z)) = \log(\exp(l_z)) + \log\left(\frac{\exp(l_z)}{1+\exp(l_z)}\right) \tag{4.2}$$

$$= l_z + l_z - \log(1+\exp(l_z)) \tag{4.3}$$

$$= 2l_z - \log(1+\exp(l_z)) \tag{4.4}$$

As discussed in Section 2.4.2, $\log(\text{softmax}(l_z))$ does a constant addition to its input, i.e., $l_z + nk$, thus being linear. Hence, we can pass any function of l_z, i.e., $h(l_z)$ as an input to $\log(\text{softmax}(.))$ and the output that we will get is $h(l_z) + nk$. If $h(.)$ is an identity function, $\log(\text{softmax}(h(l_z))$ is the same as $\log(\text{softmax}(l_z))$. However, if $h(.)$ is a non-linear function, $\log(\text{softmax}(h(l_z))$ turns out to be non-linear. Now, we can represent sigsoftmax in terms of softmax as follows:

$$\text{SS}(l_z) = \text{softmax}(2l_z - \log(1 + \exp(l_z))) \tag{4.5}$$

With the above formulation of sigsoftmax, it can be also considered as an example for the second direction to break the softmax bottleneck as discussed in Section 3.1.1, i.e., explicitly introducing non-linear activation functions to increase the rank of $\mathbf{H}_\theta \mathbf{E}_\theta^T$.

4.2.2 Formulation of Generalized SigSoftmax

For any input x, we call equation 4.4 as Decomposed SigSoftmax (DSS):

$$\text{DSS}(x) = 2x - \log(1 + \exp(x)) \tag{4.6}$$

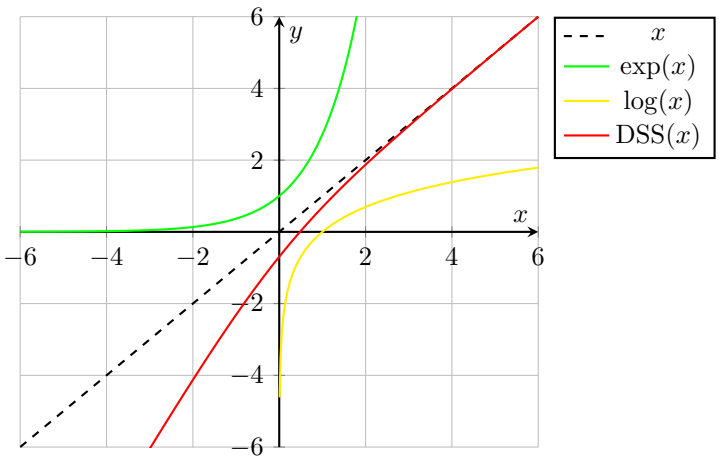


Figure 4.1: Graph of $\text{DSS}(x), \log(x)$, and $\exp(x)$.

The graph of equation 4.6, as seen in Figure 4.1, is the inspiration for our approach.

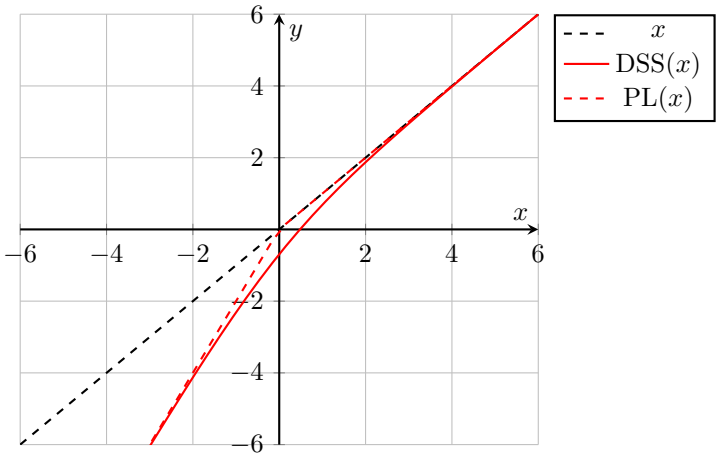


Figure 4.2: PL(x) vs DSS(x).

First, we introduce a piecewise linear function PL(x) (Figure 4.2) with two pieces which is a rough approximation to DSS(x):

$$\mathrm{PL}(x) = \begin{cases} x & x > 0 \\ 2x & x \leq 0 \end{cases} \tag{4.7}$$

Initially, we generalize equation 4.7 by making the slope of the second piece, i.e., $2x$ parameterized as kx.

$$\mathrm{PL}(x;k) = \begin{cases} x & x > 0 \\ kx & x \leq 0 \end{cases} \tag{4.8}$$

Then, we introduce $\widetilde{\mathrm{PL}}(x;k)$:

$$\widetilde{\mathrm{PL}}(x;k) = kx - (k-1)\log(\exp(x)+1) \tag{4.9}$$

which is a smooth approximation to PL($x;k$) because:

$$\begin{aligned} \text{when} \quad & x \to \infty, \quad \widetilde{\mathrm{PL}}(x;k) \to x \\ \text{similarly, when} \quad & x \to -\infty, \quad \widetilde{\mathrm{PL}}(x;k) \to kx \end{aligned} \tag{4.10}$$

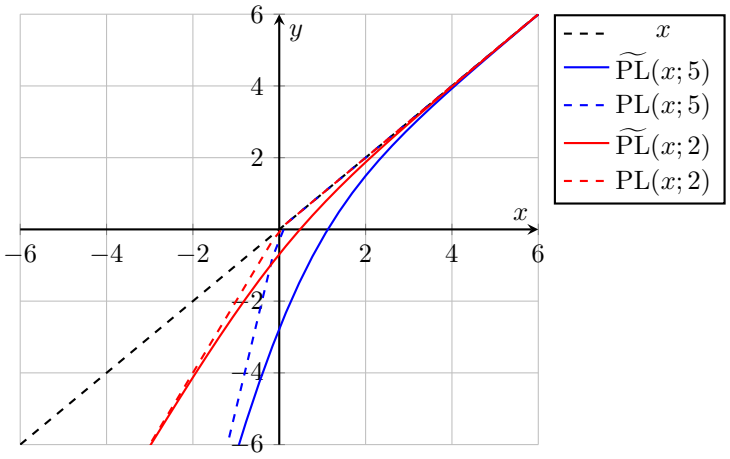


Figure 4.3: $\widetilde{\mathrm{PL}}(x;k)$ vs $\mathrm{PL}(x;k)$ for different values of k.

By looking at equations 4.8 and 4.6 as well as the Figure 4.3, it becomes obvious that:

$$\widetilde{\mathrm{PL}}(x;2) = \mathrm{DSS}(x) \tag{4.11}$$

In equation 4.7, the condition to determine the piece is actually based on a fixed constant i.e., 0. Now, we parameterize it as c. Also, we plug in the general line equation instead of kx, but we keep the other piece x as it is:

$$\mathrm{PL}(x;c,k,b) = \begin{cases} x & x > c \\ kx + b & x \leq c \end{cases} \tag{4.12}$$

We make use of the concept of intersection of two lines, to reduce the number of parameters for $\mathrm{PL}(x;c,k,b)$ from three to two by getting rid of b. Instead of using the intersection point, we can also make use of x-intercept or y-intercept as shown in Figure 4.4. We know that the two lines $y = x$ and $y = kx + b$ intersect at $x = y = c$. So:

$$\begin{aligned} kx + b = x &= c \\ kc + b &= c \\ b &= c(1-k) \end{aligned}$$

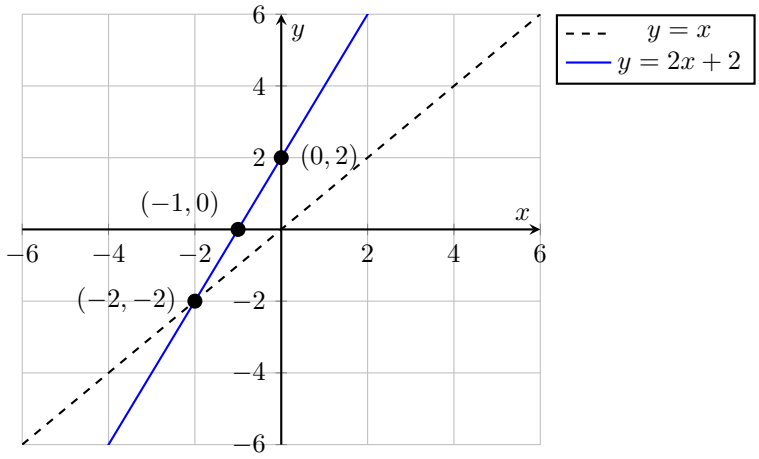


Figure 4.4: A representative graph to show the three possible control points.

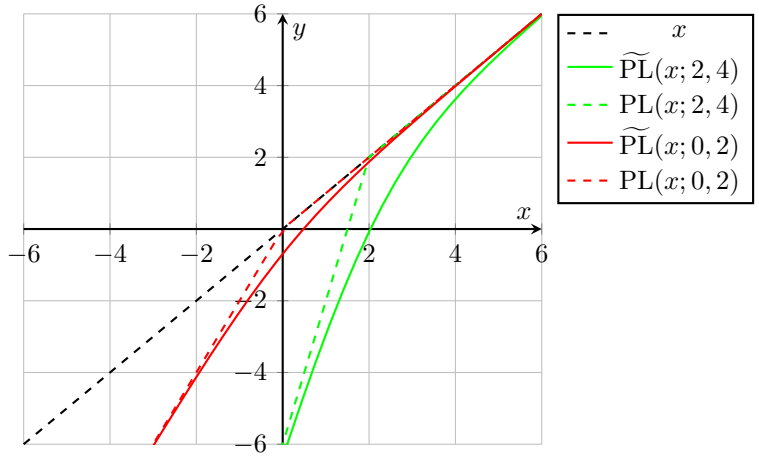


Figure 4.5: $\widetilde{\mathrm{PL}}(x; c, k)$ vs $\mathrm{PL}(x; c, k)$ for different values of c, k.

Now, equation 4.12 becomes:

$$\mathrm{PL}(x; c, k) = \begin{cases} x & x > c \\ kx + c(1 - k) & x \leq c \end{cases} \tag{4.13}$$

The smooth approximation of equation 4.12 by making use of the properties mentioned in 4.10 is:

$$\widetilde{\mathrm{PL}}(x; c, k) = k(x - c) + c - (k - 1)\log(\exp(x - c) + 1) \tag{4.14}$$

As we know that $\text{softplus}(x) = \log(\exp(x) + 1)$, we get:

$$\widetilde{\text{PL}}(x; c, k) = k(x - c) + c - (k - 1)\text{softplus}(x - c) \tag{4.15}$$

Note that $\widetilde{\text{PL}}(x; 0, 0) = \text{softplus}(x)$. Finally, we define Generalized SigSoftmax (GSS) as:

$$\text{GSS}(l_z; c, k) = \text{softmax}(\widetilde{\text{PL}}(l_z; c, k)) \tag{4.16}$$

We could immediately see the connection between $\text{SS}(l_z)$ as seen in equation 4.5 and equation 4.16 as follows:

$$\text{SS}(l_z) = \text{GSS}(l_z; 0, 2) \tag{4.17}$$

Similarly:

$$\text{softmax}(l_z) = \text{GSS}(l_z; c, 1) \tag{4.18}$$

Thus, we have introduced the GSS function. We will be using it as a replacement for the softmax function in the AWD-LSTM model. In the upcoming sections, we will empirically show that the GSS function can produce $\mathbf{Q}_\theta$ with diverse ranks, and we will compare the GSS function with existing works.

4.3 Experimental Setup for Fair Comparison

All the existing solutions [40, 22, 86, 72, 87] to break the softmax bottleneck were built on top of the AWD-LSTM [49] model. Hence, we replace the softmax function in the AWD-LSTM model with GSS and we obtain a model that we call the AWD-LSTM-GSS model. In most of our experiments, we compare AWD-LSTM-GSS with other AWD-LSTM models that use different functions such as softmax, SS, LMS-PLIF, and MoS.

4.3.1 Data Description and Preprocessing

All our experiments were conducted on two popular and commonly-used benchmark data sets for language modeling - Penn TreeBank (PTB) [46] and WikiText-2 (WT2) [48]. The PTB data set was created from a collection of Wall Street Journal (WSJ) news articles, and a preprocessed version of it was released by Mikolov et al. [51]. The WikiText (WT) language modeling data set is a collection of over 100 million tokens, extracted from the set of verified Good and Featured articles on Wikipedia.

4.3.1.1 Necessary Information About Preprocessing

1. On PTB, below are some notable details about the additional preprocessing done by Mikolov et al. [51]:
    - All words were lower-cased.
    - Numbers were replaced with the letter N.
    - All punctuation tokens were removed.
2. Similarly, to construct WT from 23,805 Good articles and 4,790 Featured articles on Wikipedia:
    - Tokenization was performed using Moses tokenizer [42].
    - Mathematical formulae and LaTeXcode were replaced with <formula> token.
    - Numbers, punctuations, and cases were preserved.
    - Two variants of the data set were released namely WikiText-103 (WT103) and WikiText-2 (WT2). Both have the same validation and test sets, but differ in training set with former being larger than the latter.

In both PTB and WT2, the new line characters were replaced with the <eos> token, and the top $N-1$ frequent tokens along with the <unk> token were chosen as the vocabulary of size N. Rest of the tokens (i.e., not frequent) in the data set were replaced with the <unk> token, which is also known as the out-of-vocabulary token. Merity et al. [49], in addition to releasing their code for AWD-LSTM as an open source software in GitHub, had also released the preprocessed versions of PTB and WT2 data sets. Several works [82, 81, 87, 86, 40, 22, 23, 72], which were built on top of the AWD-LSTM for language modeling, had used the same preprocessed data sets whose statistics are shown in Table 4.2.

4.3.2 Hyperparameter Configuration

As AWD-LSTM is a highly regularized model, it is critical not to overlook any of its hyperparameters. The authors of SS and LMS-PLIF did not change much the hyperparameters of the AWD-LSTM with an exception for one or two. But the authors of MoC and MoS had used different values for most of the hyperparameters, including the dimensions of the context and embedding vectors. For now, we use them as it is for our experiments. We will discuss the influence of this different set of hyperparameters in the next chapter.

Statistics	Data set	
	PTB	WT2
Number of tokens in training set	929,589	2,088,628
Number of tokens in validation set	73,760	217,646
Number of tokens in testing set	82,430	245,569
Vocabulary size	10,000	33,278
Percentage of <unk> token	4.91%	3.20%
Percentage of <eos> token	4.53%	1.76%

Table 4.2: Statistics for Penn TreeBank (PTB) and WikiText-2 (WT2) data sets.

However, for other AWD-LSTM models with different functions including softmax, SS, GSS, and LMS-PLIF, we use the same hyperparameters that were used by the authors of AWD-LSTM. There are definitely a considerable number of hyperparameters that have the same value for all models under comparison across both the PTB and WT2 data sets. Notable among them are 0.5[1] for dropconnect on LSTM weights, 0.1 for embedding lookup dropout, 1.2 $\times 10^{-6}$ for scaling factor of L2 regularization, 2 and 1 for scaling factors of AR and TAR, 0.4 for dropout on the context vector, i.e., output vector from the top-most LSTM layer $\mathbf{h}_t^3$. Hyperparameters that are different for these models are shown in Table 4.3.

4.3.2.1 Hyperparameters specific to MoS, MoC, and LMS-PLIF

MoS, MoC, and LMS-PLIF introduce extra trainable parameters in addition to those present in the original AWD-LSTM model. In case of MoS and MoC, a fully-connected layer was introduced in between the top-most LSTM layer and the output layer. A dropout rate of 0.29 was applied to that layer, in both MoS and MoC, across both data sets. The number of mixtures K was set to 15.

In the case of LMS-PLIF, there are a few hyperparameters that are specific to the PLIF layer which are responsible for learning the pointwise linear functions. On both data sets,

[1]The values for dropconnect and dropout are the probabilities of dropping connection and dropping neurons respectively.

[2]The random seed used for random initialization of parameters in the network.

Hyperparameter	For Softmax, SS, GSS, and LMS-PLIF		For MoS and MoC	
	Data set		**Data set**	
	PTB	WT2	PTB	WT2
Dropout for $\mathbf{e}_t$	0.4	0.65	0.4	0.55
Dropout for $\mathbf{h}_t^1, \mathbf{h}_t^2$	0.3	0.2	0.225	0.2
Learning rate	20.0	30.0	20.0	15.0
Batch size	20	80	12	15
Seed[2]	141	1,881	28	1,881
$dim(\mathbf{e}_t)$	400	400	280	300
$dim(\mathbf{h}_t^1), dim(\mathbf{h}_t^2)$	1,150	1,150	960	1,150
$dim(\mathbf{h}_t^3)$	400	400	620	650

Table 4.3: Differences in hyperparameters. Please refer Figure 2.7 to know about $\mathbf{e}_t$, $\mathbf{h}_t^1$, $\mathbf{h}_t^2$, $\mathbf{h}_t^3$.

the number of intervals K was set to 10^5, so that the slopes of 10^5 linear functions (lines) can be learned. The range of l_z under consideration was [-20, 20] within which 10^5 different linear functions are learned. Also, a layer-specific learning rate of 0.02 was used.

4.3.2.2 Trigger condition for switching to ASGD

There are several works [72, 22, 81, 82] that were built on top of AWD-LSTM language models wherein the authors had tried slightly different approaches to handle the triggering for switching to ASGD, different from how it has been done in the original AWD-LSTM model. Takase et al. [72] had experimented with different non-monotone interval n and had reported that $n = 60$ performed better than the original configuration of $n = 5$. In LMS-PLIF [22] and a few other works [81, 82], the authors had made the trigger at a particular epoch to switch to ASGD. Specifically, they had made the trigger when the epoch number is 200. We call it as Epoch Triggered ASGD (ET-ASGD). In both the above-mentioned approaches, it is notable that the baseline AWD-LSTM performs better than its original hyperparameter configuration. However, except for [72], others had not explicitly

acknowledged that we could get a better performance on the test set because of this change in approach. We emphasize that for a fair comparison, all models under comparison should follow the same approach. Another important clarification that is needed here is, though we talk about ASGD that stands for Averaged Stochastic Gradient Descent, in practice, we use Mini-Batch Gradient Descent and Mini-Batch Averaged Gradient Descent, which are essentially the mini-batch counterparts of SGD and ASGD.

4.3.3 Rank Calculation

As we are interested in the rank of the log probability matrix $\mathbf{Q}_\theta$, we first need to construct the matrix. To get $\mathbf{Q}_\theta$, the model is evaluated on the test set and the log probability distributions over the vocabulary, which the model outputs at every time step t, are stacked together.

$$\mathbf{Q}_\theta = \begin{bmatrix} \log(P_\theta(\mathbf{w}|c_1)) \\ \log(P_\theta(\mathbf{w}|c_2)) \\ . \\ . \\ \log(P_\theta(\mathbf{w}|c_T)) \end{bmatrix} \tag{4.19}$$

where $\mathbf{Q}_\theta \in \mathbb{R}^{T \times N}$, T is the total number of time steps which is essentially the total number of contexts M. Hence, $\mathbf{Q}_\theta \in \mathbb{R}^{M \times N}$

In the PTB data set, there are $10,000$ tokens in the vocabulary and $82,430$ contexts in its test set. Hence, $\mathbf{Q}_\theta \in \mathbb{R}^{82,430 \times 10,000}$. The size of $\mathbf{Q}_\theta$ in GigaBytes (GB) is approximately 3.3 GB as single precision floating point (4 Bytes) is typically used to represent the parameters of the network. Thus, the inputs and outputs of the network also use the same data type.

Similarly, in the WT2 data set, there are $33,278$ tokens in the vocabulary and $245,370$ contexts in the test set. Hence, if all the contexts are considered, the size of $\mathbf{Q}_\theta \in \mathbb{R}^{245,370 \times 33,278}$ would approximately be 33 GB. Due to memory and time constraints of the LAPACK routine used for SVD calculation, we consider only the top $33,320$ contexts so that $\mathbf{Q}_\theta \in \mathbb{R}^{33,320 \times 33,278}$. The size is now around 4.5 GB.

Existing solutions to break the softmax bottleneck such as SigSoftmax (SS) and Mixture of Softmaxes (MoS) use the threshold proposed by Press et al. [61] to identify non-zero singular values. To be consistent with them, we use the same threshold.

As the authors of SS and MoS did not release their code for calculating the rank of $\mathbf{Q}_\theta$, we were not sure about the choice of algorithm used for SVD calculation. However, after

discussing with the authors of SS, we came to know that they had used the randomized algorithm [29] to compute Truncated SVD. But we use Thin SVD, as we do not want to introduce additional randomness in the computation of SVD. Also, Truncated SVD is typically meant for sparse matrices and not for dense matrices. Because of this difference in choice of algorithm, we are able to see a difference in the value of rank between the reported results for SS and our reproduced results for SS (shown in Table 4.1). In the same table, we can also see that the difference in the rank values between the reported results for MoS and our reproduced results is only 1. Hence, we suspect that the authors of MoS should have also used Thin SVD.

4.3.4 Hardware and Software Configuration

All our experiments were conducted on Compute Canada's cedar cluster of GPU servers. The experiments were run on same configuration of hardware to ensure a consistent environment for training the models. The hardware configuration was 8 cores of CPU, 32GB of memory (RAM), and one V100 GPU (16GB or 32GB depending on its availability and our needs). We did not train any model on multiple GPUs as it would usually result in a slight performance reduction. Also, requesting multiple GPUs for a job on Compute Canada's clusters could result in longer wait times. The neural networks related code was written using PyTorch 0.4.1 [58] framework. Python 3.7 [76] with well-known libraries like NumPy were used for other analyses. For better managing hundreds of model training that we did, we relied upon comet.ml, a cloud-based platform to track and compare models.

4.3.5 Statistical Significance Test

According to Redman [63], "Statistical significance helps quantify whether a result is likely due to chance or to some factor of interest". When any two models are compared against each other, the ideal way to make a claim that one model is better than the other is by doing a statistical significance test between the samples of results from both models. For neural language models, to get a sample of perplexities on test set, the model has to be trained using different random seeds for network initialization and evaluate each of them on the test set. Until recently, not many in the literature undertook such efforts, as training a model just for a single seed is very time consuming. But now, such efforts are gaining importance for model comparisons. Among the related works that we discussed in Chapter 3, it is worth noting that only in the work of SigSoftmax we were able to see such attempts.

We define a null hypothesis which says that there is no statistical significance between the sample results of models A and B, i.e., results of model B is likely due to chance and not due to any other factor of interest, which in our case is the use of a function other than Softmax. The p-value is the probability that the null hypothesis is true. Conventionally, a p-value greater than 0.05 or 5×10^{-2} is required to say that the null hypothesis is true. If it is less than 5×10^{-2}, we can say that a model B performs statistically significantly better than a model A. To compute the p-value, we perform a two-sided Wilcoxon Rank-Sum (WRS) test, which is a non-parametric two-sample test [83]. Assume there are two different populations U and V. Let S_U and S_V be the samples of observations from U and B. The WRS test is solely based on the order in which the observations from the two samples fall [83]. More formally, the null hypothesis is defined as $H_0 : U = V$, which is that the distribution of measurements in U is same as that in V. The two-sided alternative hypothesis is defined as $H_1 : U \neq V$, in which the distributions are not same. The WRS test is based upon ranking all observations in the combined sample $S_U + S_V$ resulting in a test statistic W which is then used to compute the p-value [83].

4.4 Experiments and Results

4.4.1 Generalized SigSoftmax's Ability to Produce Diverse Ranks

Here, we verify GSS's ability to produce ranks in the low, medium, and high ranges as an outcome of our motivation, as discussed in Section 4.1. To do so, we control the parameters c and k of GSS(.) by performing a finite grid search.

For the PTB data set, we grid search over the cross product of $c = \{0.5\, x \mid x \in [-4, 4]\}$ and $k = \{0.25\, x \mid x \in [5, 12]\}$. Similarly for the WT2 data set, we grid search over the cross product of $c = \{0.5\, x \mid x \in [-4, 4]\}$ and $k = \{0.5\, x \mid x \in [3, 6]\}$.

We say that a rank is low if it is in the range of $[0, 2500)$, medium if it is in $[2500, 7500)$, and high if it is in $[7500, N]$ where N is the vocabulary size of the data set.

We categorize the actual ranks into these classes only for explanatory purposes and fixing the range for each category (low, medium, and high) is arbitrary. We could see (in tables 4.4 and 4.5) that GSS indeed is able to produce $\mathbf{Q}_\theta$ with diverse ranks. However, the key observation from these tables and figures 4.6 and 4.7 is that having a high-rank $\mathbf{Q}_\theta$ does not necessarily mean better performance (i.e., lower perplexity) on the test set.

Rank range	c	k	Validation ppl	Test ppl	Rank
Low	-2.00	1.25	59.57	57.22	1,694
	2.00	2.50	60.75	58.04	2,052
	2.00	1.25	59.36	56.94	2,169
	1.50	1.25	59.49	57.07	2,381
	1.00	2.75	61.63	58.26	2,498
Medium	0.00	3.00	61.92	58.48	2,969
	1.50	1.50	59.46	57.14	3,363
	0.50	1.50	59.41	56.88	4,525
	0.00	1.75	59.26	56.83	5,798
	-1.00	2.50	59.73	56.99	6,798
High	-1.50	2.00	59.64	57.07	7,507
	-1.00	2.25	59.47	57.21	8,064
	-1.50	2.50	59.36	56.78	8,989
	-1.50	2.75	59.32	56.92	9,257
	-2.00	3.00	59.44	57.04	9,473

Table 4.4: AWD-LSTM-GSS models (trained on PTB) with diverse ranks when different values are used for parameters c and k of GSS.

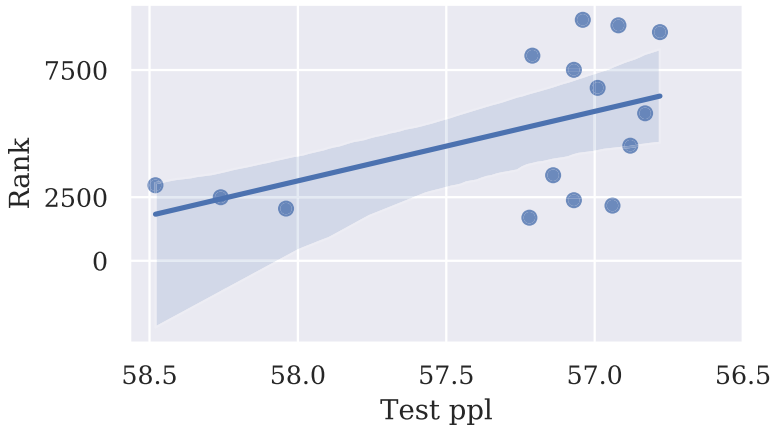


Figure 4.6: Correlation between test perplexity and rank of $\mathbf{Q}_\theta$ for several AWD-LSTM-GSS models (Table 4.4) with different values for parameters c and k of GSS.

Rank range	c	k	Validation ppl	Test ppl	Rank
Low	2.00	2.25	68.53	65.63	1,747
	2.00	3.00	69.59	66.82	1,855
	1.50	2.50	68.86	65.73	1,936
	1.50	3.00	69.70	66.50	2,047
	2.00	2.00	67.37	64.30	2,272
Medium	0.00	3.00	70.23	66.89	2,857
	-1.50	1.50	67.32	64.54	3,365
	1.00	2.00	67.13	64.27	4,622
	-2.00	2.00	67.46	64.74	5,603
	0.00	2.50	67.29	64.51	6,989
High	-2.00	2.50	67.59	64.68	7,620
	-1.50	2.50	67.12	64.43	8,479
	-2.00	3.00	67.15	64.61	9,359
	-1.50	3.00	66.96	64.37	10,145
	-1.00	3.00	67.48	64.55	10,285

Table 4.5: AWD-LSTM-GSS models (trained on WT2) with diverse ranks when different values are used for parameters c and k of GSS.

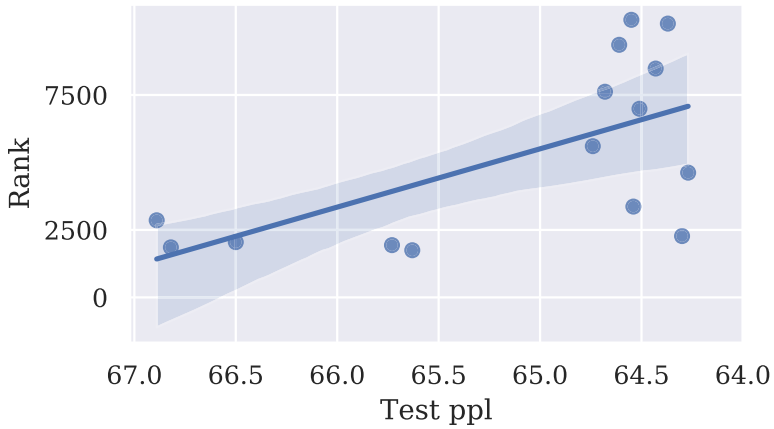


Figure 4.7: Correlation between test perplexity and rank of $\mathbf{Q}_\theta$ for several AWD-LSTM-GSS models (Table 4.5) with different values for parameters c and k of GSS.

Contrarily, having a low rank does not mean worse performance (i.e., higher perplexity) on the test set. In other words, there is no strong positive correlation between the rank of $\mathbf{Q}_\theta$ and the performance of a language model in terms of its perplexity on the test set.

For both data sets, we select one pair of values for (c, k). The selection is based on whether GSS with chosen (c, k) can produce a high-rank $\mathbf{Q}_\theta$ as well as have a lower perplexity on the test set when compared to other pairs of parameters. We use the same pair of (c, k) for all our further experiments and analyses. For the PTB data set, we choose c and k as -1.50 and 2.50. Similarly, for WT2, we choose them as -1.50 and 3.00.

4.4.2 Comparison of Generalized SigSoftmax with Other Functions

We compare AWD-LSTM-GSS against other AWD-LSTM models with Softmax, SS, LMS-PLIF, and MoS functions. The comparison is made on both the PTB and WT2 data sets.

Function	#Param	Train ppl	Validation ppl	Test ppl	Rank
NT-ASGD					
Softmax	24.22M	34.05	60.35	58.07	402
SS	24.22M	33.68	60.45	57.75	4,906
GSS	24.22M	34.24	59.95	57.60	8,276
LMS-PLIF	24.32M	37.19	60.86	58.45	510
MoS	21.50M	33.08	58.21	56.07	9,979
MoC	21.50M	33.73	59.84	57.40	282
ET-ASGD					
Softmax	24.22M	34.03	59.48	57.10	402
SS	24.22M	32.83	59.95	57.16	4,979
GSS	24.22M	34.21	59.37	56.78	8,989
LMS-PLIF	24.32M	37.07	59.08	56.67	580
MoS	21.50M	31.62	57.12	55.11	9,983
MoC	21.50M	31.37	58.38	55.81	282

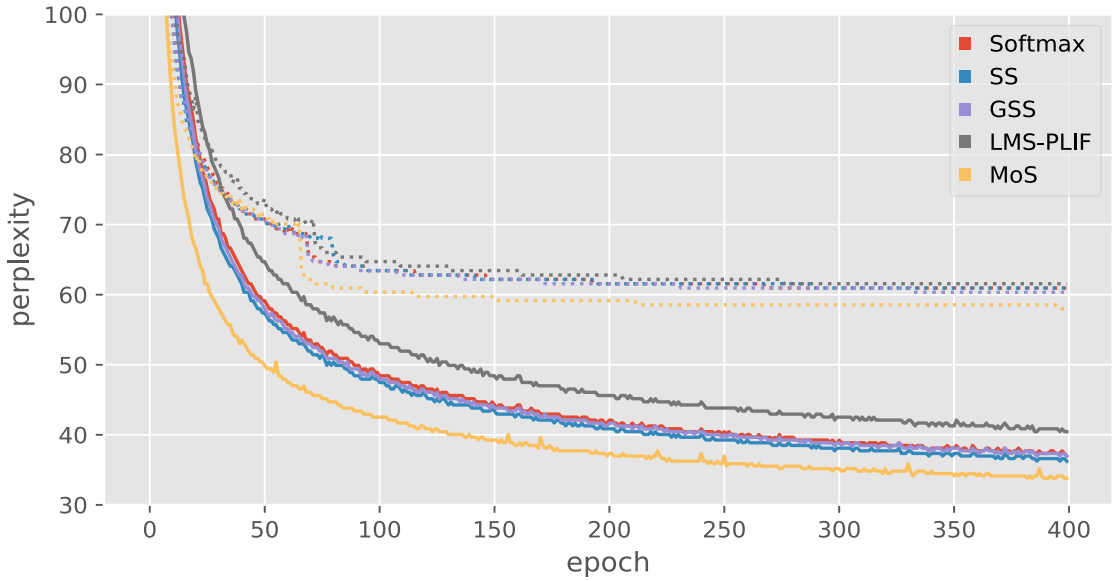
Table 4.6: Performance comparison for NT-ASGD vs ET-ASGD on PTB.

First, we conduct experiments on both data sets to check the influence of NT-ASGD and ET-ASGD on the performance of these models. We also include Mixture of Contexts (MoC), the low-rank baseline model introduced by the authors of MoS for comparison, because we use MoC for some analyses in the next chapter. The results are shown in tables 4.6 and 4.7. It is very clear that ET-ASGD definitely has a positive influence in the training of all these models across the two data sets. The test perplexity gets lowered approximately by 1.00 for all models trained on PTB, and by 0.75 for models on WT2. For the model with LMS-PLIF that is trained on PTB, the test perplexity gets reduced by 1.78, the biggest gain among all models. Similarly, MoC's test perplexity gets lowered by 1.59 when ET-ASGD is used for training on PTB. It is not just the test performance of these models, in terms of perplexity, that gets impacted by the use of ET-ASGD, but also the rank of $\mathbf{Q}_\theta$ that gets improved except for AWD-LSTM with the softmax function.

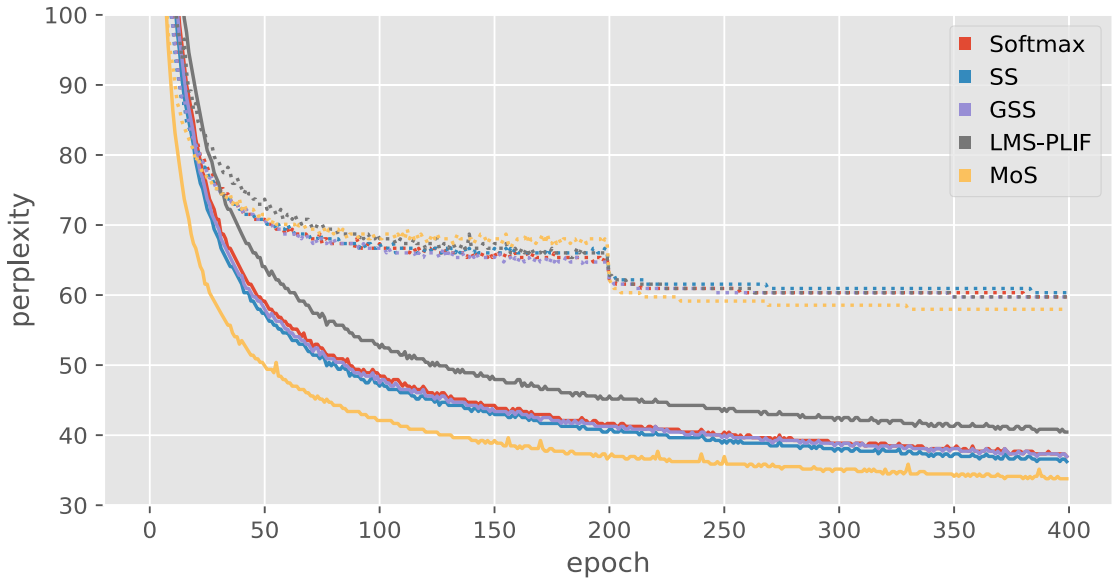
Function	**#Param**	**Train ppl**	**Validation ppl**	**Test ppl**	**Rank**
			NT-ASGD		
Softmax	33.55M	39.07	68.35	65.28	402
SS	33.55M	39.21	67.84	65.08	5,879
GSS	33.55M	39.05	67.72	65.07	9,130
LMS-PLIF	33.65M	41.11	68.54	65.59	479
MoS	34.90M	35.92	65.93	63.06	13,215
MoC	34.90M	37.21	69.08	66.42	302
			ET-ASGD		
Softmax	33.55M	39.09	67.59	64.56	402
SS	33.55M	39.19	67.19	64.33	6,590
GSS	33.55M	39.12	66.97	64.38	10,145
LMS-PLIF	33.65M	41.19	67.19	64.32	513
MoS	34.90M	35.99	64.58	61.90	15,738
MoC	34.90M	37.23	68.19	65.83	302

Table 4.7: Performance comparison for NT-ASGD vs ET-ASGD on WT2.

The learning curves for different models trained on PTB and WT2 using NT-ASGD and ET-ASGD are shown in figures 4.8a, 4.8b, 4.9a and 4.9b. When ET-ASGD is used, we can notice a significant drop in the validation perplexity, exactly after the 200th epoch for all models because of the switch from mini-batch gradient descent to mini-batch averaged gradient descent, as seen in figures 4.8b and 4.9b.

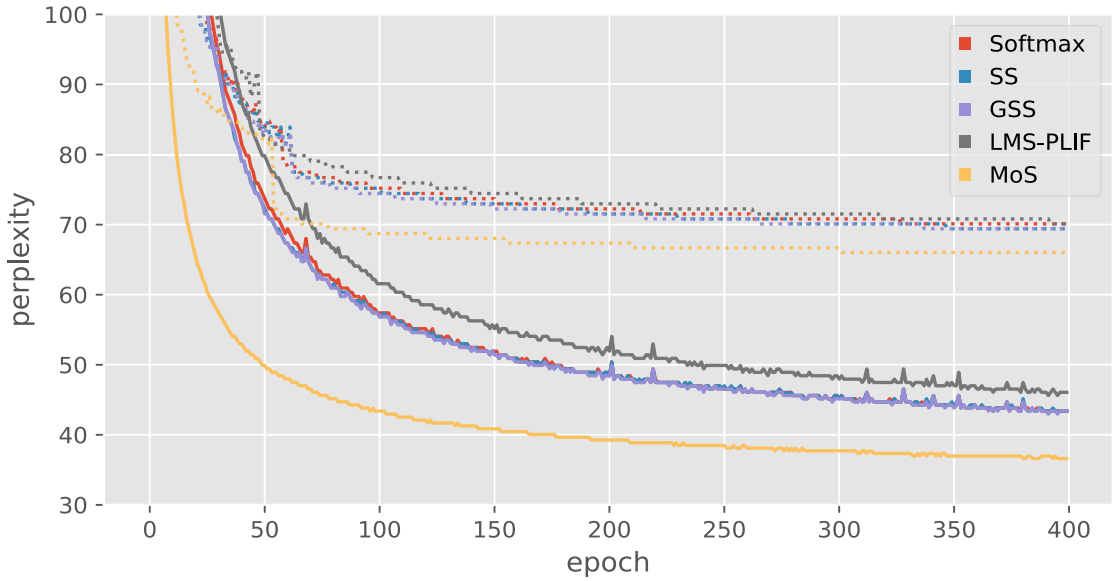


(a) Using NT-ASGD

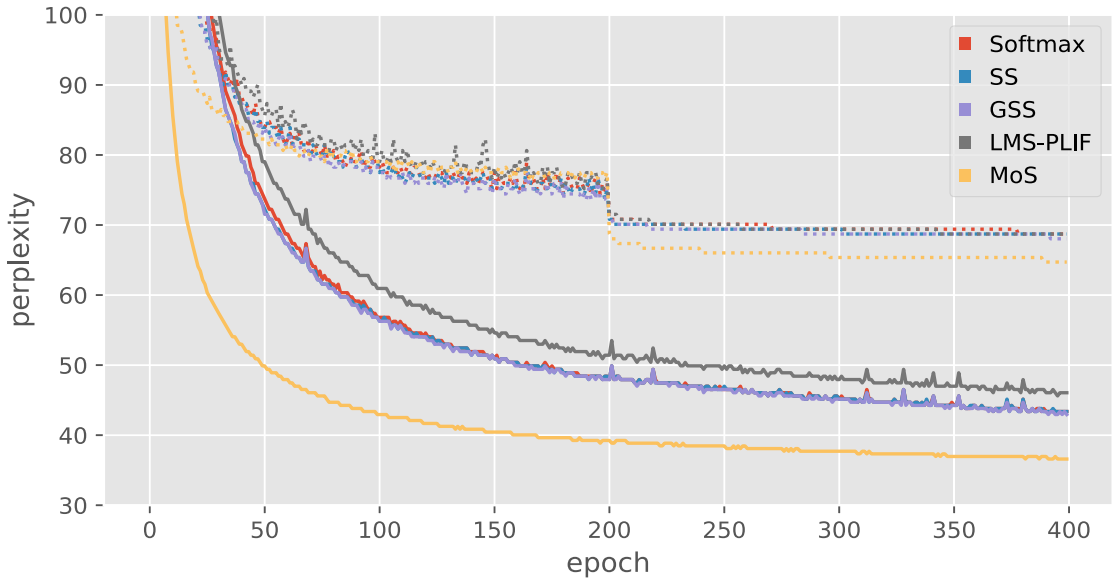


(b) Using ET-ASGD

Figure 4.8: Training AWD-LSTM with different functions on PTB. Solid lines denote training perplexity and dotted lines denote validation perplexity.



(a) Using NT-ASGD



(b) Using ET-ASGD

Figure 4.9: Training AWD-LSTM with different functions on WT2. Solid lines denote training perplexity and dotted lines denote validation perplexity.

Whereas, when using NT-ASGD, there is a high chance that the switch could have happened earlier than the desired epoch that works well for a particular model, thus negatively impacting the performance.

A neural language model that is able to achieve a test perplexity of x does not always achieve the perplexity x every time, as the trainable parameters of the model are randomly initialized. Even though, most of the existing works that try to break the softmax bottleneck have reported the results for only one initialization. It is common in the neural networks literature for the authors to report the seed used for random initialization, so that it is reproducible; but most do not repeat the training for different seeds as it is time consuming. Inspired by SS, we repeat the training using different seeds and report the mean and the standard deviation (SD) of the results as mean $\pm$ one SD. For $\mathbf{Q}_\theta$'s rank, the mean and SD are rounded-off to the nearest integer. Specifically, we train all the models on both data sets 11 times, using 11 different seeds for random initialization, which includes 10 randomly sampled (for each model) integers in addition to the seed values mentioned in Table 4.3. The results are shown in Table 4.8. All models are trained for 1,000 epochs on both data sets, except for MoS on WT2, as it is very time consuming. All models were able to converge before 1,000 epochs. Fortunately, when trained on WT2, MoS was able to converge around the 500th epoch. So it is trained only for 500 epochs. Also, it roughly takes around 600 GPU seconds[3] to complete one epoch when MoS is trained on the WT2 data set using a single NVIDIA V100 GPU. For 11 different seeds, it takes approximately 38.5 GPU days. Similarly, when MoS is trained on PTB data set for 1,000 epochs, it takes approximately 12.65 GPU days for 11 different seeds. To train AWD-LSTM models using other functions on PTB and WT2, it takes approximately 7.25 and 10.56 GPU days respectively for 11 different seeds.

By looking at the p-values from two-sided Wilcoxon Rank-Sum tests between the samples of test perplexities of AWD-LSTM models using different functions and the sample of test perplexities of AWD-LSTM model using Softmax on both data sets (as shown in Table 4.9), we can see that GSS is not statistically significantly better than Softmax on the PTB and WT2 data sets, despite significant differences in the ranks of $\mathbf{Q}_\theta$. The training and validation perplexities are also similar on both data sets for Softmax, SS, and GSS as seen in Table 4.8. However, LMS-PLIF, a low-rank model seems to perform slightly better than Softmax, SS, and GSS. MoS performs the best. It seems that the high rank of $\mathbf{Q}_\theta$ is just a by-product, and it does not have any major influence on the performance of a model. To further support or disprove this observation, we conduct several experiments in the next chapter to check if the linear independence in $\mathbf{Q}_\theta$ helps in any way or not.

[3]GPU seconds/days mean the number of seconds/days taken when trained on a single GPU

Function	#Param	Time	Train ppl	Validation ppl	Test ppl	Rank
			PTB			
Softmax	24.22M	∼55s	33.91±0.25	59.55±0.12	57.08±0.09	402±0
SS	24.22M	∼57s	32.87±0.19	59.69±0.15	57.05±0.15	5,113±85
GSS	24.22M	∼58s	33.68±0.32	59.51±0.13	57.00±0.14	8,904±57
LMS-PLIF	24.32M	∼59s	36.67±0.30	59.08±0.09	56.80±0.11	496±83
MoS	21.50M	∼100s	31.84±0.23	57.14±0.26	54.91±0.26	9,981±2
			WT2			
Softmax	33.55M	∼72s	39.32±0.14	67.57±0.14	64.63±0.08	402±0
SS	33.55M	∼83s	39.29±0.17	67.28±0.16	64.35±0.17	6,634±46
GSS	33.55M	∼87s	39.42±0.41	67.35±0.22	64.50±0.13	10,122±72
LMS-PLIF	33.65M	∼93s	41.13±0.15	67.15±0.21	64.16±0.17	524±74
MoS	34.90M	∼600s	36.17±0.30	64.54±0.36	61.96±0.40	15,734±188

Table 4.8: Comparison of AWD-LSTM models with different functions after each model was trained 11 times with different seeds for random initialization of parameters in the network. The column Time denotes the time taken to complete one epoch.

Function	Test ppl	p-value
SS	57.03 ± 0.15	3.25×10^{-1}
GSS	57.02 ± 0.13	2.56×10^{-1}
LMS-PLIF	56.81 ± 0.11	5.06×10^{-4}
MoS	54.88 ± 0.26	1.57×10^{-4}

(a) Trained on PTB. The sample statistic[4] of test perplexities of Softmax is 57.08 ± 0.10

Function	Test ppl	p-value
SS	64.35 ± 0.18	1.93×10^{-3}
GSS	64.51 ± 0.13	5.87×10^{-2}
LMS-PLIF	64.15 ± 0.17	1.57×10^{-4}
MoS	61.97 ± 0.43	2.38×10^{-4}

(b) Trained on WT2. The sample statistic of test perplexities of Softmax is 64.63 ± 0.09

Table 4.9: p-values from two-sided Wilcoxon Rank-Sum tests between sample of test perplexities of AWD-LSTM models with different functions and sample of test perplexities of baseline AWD-LSTM with Softmax. Each model was trained 10 times with randomly sampled seeds (for each of them) for random initialization of parameters.

[4]Throughout this thesis, the sample statistic of test perplexities denote the mean ± one standard deviation of the sample of test perplexities.

4.5 Summary

In this chapter:

- We generalized SigSoftmax (SS) and introduced a new family of output layer specific, non-linear, and parametric activation functions called Generalized SigSoftmax (GSS).
- We empirically showed that GSS can help language models to produce $\mathbf{Q}_\theta$ with diverse ranks, by controlling the parameters of GSS.
- We discussed the differences in hyperparameters among existing works and exposed the importance of using the same trigger condition for switching to ASGD. Specifically, we showed that using epoch number as the trigger condition for switch to ASGD (ET-ASGD) was beneficial to all models under comparison. When ET-ASGD is used, we showed that the performance improvements for Linear Monotonic Softmax with Piecewise Increasing Functions (LMS-PLIF) is not as high as it was reported by its authors, when compared with Softmax.
- On both PTB and WT2 data sets, we showed that GSS is not statistically significantly better than Softmax in terms of its sample test perplexities, irrespective of their differences in the rank of $\mathbf{Q}_\theta$. With the help of GSS, we showed that a high-rank language model does not guarantee a better quantitative performance than a low-rank model.

Chapter 5

Reanalysis of High-Rank Models

5.1 Motivation

In the previous chapter, we showed that a high-rank language model does not necessarily offer better performance in terms of perplexity on test set. First, in this chapter, we conduct experiments to check if there are any additional benefits in having a high-rank language model. In these experiments, we consider AWD-LSTM models with different functions such as Softmax, SigSoftmax (SS), Generalized SigSoftmax (GSS), Linear Monotonic Softmax with Piecewise Linear Increasing Functions (LMS-PLIF), Mixture of Softmaxes (MoS), and Mixture of Contexts (MoC). We perform a qualitative analysis by looking at the actual predictions for the next token made by different models for different input contexts. As the word embeddings are jointly learned during the process of training a language model, we evaluate the quality of embeddings on several word similarity benchmarks. Also, we inspect the word embedding matrix from these models for the presence of the representation degeneration problem. We find that there are no significant additional benefits. Second, we question the fundamental observation made by the authors of MoS, i.e., "Higher the rank of $\mathbf{Q}_\theta$, better the performance in terms of test perplexity", by varying a few hyperparameters. We observe that such a correlation is not always true. We also find that the singular value distributions of $\mathbf{Q}_\theta$ from these models are similar, irrespective of the differences in the calculated rank. Third, we check for the robustness of the approach used to calculate the rank of a matrix from singular values. We observe that this approach to calculate rank is not able to identify the similarity in singular value distributions and is sensitive to tiny gaussian additive noise. Fourth, we introduce an alternative metric to calculate the rank of a matrix which we call ϵ-effective rank. Finally, we conduct experiments to check for the

impact on the regularization of the models when LMS-PLIF and MoS are used.

5.2 Qualitative Analysis

Yang et al. [86] had conducted a qualitative analysis by actually looking at the probabilities of the next token for different contexts in the PTB data set. They compared MoS with MoC and had showed that MoS makes better context-dependent predictions because of its ability to produce a high-rank $\mathbf{Q}_\theta$. We redo the analysis, by considering the same set of contexts as they did, but we also include models using functions other than MoS and MoC.

Context #1	managed properly and with a long-term outlook these can become investment-grade quality properties <eos> canadian <unk> production totaled N metric tons in the week ended oct. N up N N from the preceding week's total of N _____				
Softmax	**tons** **0.90**	million 0.04	metric 0.02	trillion 0.01	billion 0.01
SS	**tons** **0.85**	million 0.04	metric 0.02	billion 0.02	units 0.01
GSS	**tons** **0.68**	units 0.09	million 0.04	billion 0.03	trillion 0.03
LMS-PLIF	**tons** **0.83**	metric 0.11	million 0.02	units 0.01	trillion 0.01
MoS	**tons** **0.40**	million 0.26	billion 0.12	<eos> 0.05	units 0.05
MoC	**tons** **0.36**	million 0.27	billion 0.13	units 0.05	metric 0.04
MoS†	million 0.28	billion 0.23	**tons** **0.19**	trillion 0.10	<eos> 0.05
MoC†	million 0.30	**tons** **0.30**	billion 0.17	<eos> 0.04	trillion 0.03
MoS‡	million 0.38	**tons** **0.24**	billion 0.09	barrels 0.06	ounces 0.04
MoC‡	billion 0.39	million 0.36	trillion 0.05	<eos> 0.04	N 0.03

Reference #1	canadian <unk> production totaled N metric **tons** in the week ended oct. N up N N from the preceding week 's total of N tons statistics canada a federal agency said <eos>				
Context #2	the thriving <unk> street area offers <unk> of about $ N a square foot as do <unk> locations along lower fifth avenue <eos> by contrast <unk> in the best retail locations in boston san fran- cisco and chicago rarely top $ N _____				
Softmax	million 0.32	billion 0.30	<eos> 0.04	to 0.03	in 0.03
SS	<eos> 0.28	**a** **0.11**	million 0.11	to 0.07	far 0.05
GSS	<eos> 0.18	and 0.13	million 0.10	to 0.08	**a** **0.06**
LMS-PLIF	**a** **0.17**	<eos> 0.14	million 0.07	to 0.07	far 0.06
MoS	million 0.28	billion 0.15	<eos> 0.14	**a** **0.10**	to 0.05
MoC	<eos> 0.12	to 0.11	million 0.10	in 0.08	**a** **0.05**
MoS†	million 0.22	**a** **0.13**	<eos> 0.12	billion 0.11	to 0.07
MoC†	million 0.22	to 0.11	<eos> 0.08	in 0.07	billion 0.06
MoS‡	<eos> 0.36	**a** **0.13**	to 0.07	for 0.07	and 0.06
MoC‡	million 0.39	billion 0.36	<eos> 0.05	to 0.04	of 0.03
Reference #2	by contrast <unk> in the best retail locations in boston san francisco and chicago rarely top $ N **a** square foot <eos>				
Context #3	as other <unk> governments particularly poland and the soviet union have recently discovered initial steps to open up society can create a momentum for radical change that becomes difficult if not impossible to control <eos> as the days go by the south _____				
Softmax	africa 0.20	**african** **0.11**	to 0.08	korea 0.05	korean 0.05

SS	africa 0.15	**african** **0.14**	korea 0.08	korean 0.05	<unk> 0.05
GSS	africa 0.18	korean 0.10	**african** **0.09**	and 0.06	korea 0.04
LMS-PLIF	africa 0.19	korea 0.05	**african** **0.05**	and 0.05	korean 0.04
MoS	africa 0.15	**african** **0.11**	korea 0.08	of 0.06	and 0.05
MoC	bloc 0.19	africa 0.14	and 0.07	korea 0.06	**african** **0.04**
MoS†	**african** **0.16**	africa 0.13	the 0.08	korea 0.06	<unk> 0.04
MoC†	and 0.11	africa 0.10	bloc 0.07	korea 0.06	<unk> 0.05
MoS‡	africa 0.15	**african** **0.15**	<eos> 0.14	korea 0.08	korean 0.05
MoC‡	<eos> 0.38	and 0.08	of 0.06	or 0.05	<unk> 0.04
Reference #3	as the days go by the south **african** government will be ever more hard pressed to justify the continued <unk> of mr. <unk> as well as the continued banning of the anc and enforcement of the state of emergency <eos>				
Context #4	shares of ual the parent of united airlines were extremely active all day friday reacting to news and rumors about the proposed $ N billion buy-out of the airline by an <unk> group <eos> wall street 's takeover-stock speculators or risk arbitragers had placed unusually large bets that a takeover would succeed and _____				
Softmax	the 0.17	<unk> 0.05	that 0.04	they 0.03	it 0.02
SS	the 0.12	that 0.06	they 0.05	<unk> 0.03	then 0.03
GSS	the 0.17	that 0.04	they 0.04	it 0.03	<unk> 0.03
LMS-PLIF	the 0.10	<unk> 0.05	that 0.03	they 0.02	even 0.02

MoS	the 0.12	<unk> 0.08	**ual** **0.08**	that 0.03	coniston 0.02
MoC	the 0.22	<unk> 0.03	they 0.03	a 0.03	then 0.02
MoS†	the 0.10	<unk> 0.07	**ual** **0.06**	that 0.03	they 0.02
MoC†	the 0.23	<unk> 0.03	mr . 0.02	**ual** **0.03**	that 0.02
MoS‡	the 0.14	that 0.07	**ual** **0.07**	<unk> 0.03	it 0.02
MoC‡	the 0.10	<unk> 0.06	that 0.05	in 0.02	it 0.02
Reference #4	wall street 's takeover-stock speculators or risk arbitragers had placed unusually large bets that a takeover would succeed and **ual** stock would rise <eos>				
Context #5	the government is watching closely to see if their presence in the <unk> leads to increased <unk> protests and violence if it does pretoria will use this as a reason to keep mr. <unk> behind bars <eos> pretoria has n't forgotten why they were all sentenced to life <unk> in the first place for sabotage and ____				
Softmax	<unk> 0.54	political 0.02	violence 0.01	peace 0.01	**conspiracy** **0.01**
SS	<unk> 0.45	political 0.02	other 0.01	violence 0.01	incest 0.01
GSS	<unk> 0.45	other 0.01	political 0.01	incest 0.01	civil 0.01
LMS-PLIF	<unk> 0.50	political 0.01	a 0.01	the 0.01	incest 0.01
MoS	<unk> 0.26	violence 0.03	other 0.03	the 0.03	a 0.02
MoC	<unk> 0.65	other 0.02	incest 0.01	acts 0.01	the 0.01
MoS†	<unk> 0.21	acts 0.03	the 0.03	other 0.03	incest 0.02
MoC†	<unk> 0.38	other 0.04	the 0.03	in 0.01	that 0.01

MoS‡	<unk> 0.47	violence 0.11	**conspiracy 0.03**	incest 0.03	civil 0.03
MoC‡	<unk> 0.41	the 0.03	a 0.02	other 0.02	in 0.01
Reference #5	pretoria has n't forgotten why they were all sentenced to life <unk> in the first place for sabotage and **conspiracy** to <unk> the government <eos>				
Context #6	china's <unk> <unk> program has achieved some successes in <unk> runaway economic growth and stabilizing prices but has failed to eliminate serious defects in state planning and an <unk> drain on state budgets <eos> the official china daily said retail prices of <unk> foods have n't risen since last december but acknowledged that huge government ____				
Softmax	spending 0.09	costs 0.07	payments 0.04	orders 0.04	sales 0.04
SS	spending 0.10	costs 0.04	<unk> 0.04	orders 0.03	payments 0.03
GSS	spending 0.13	sales 0.07	<unk> 0.04	exports 0.03	officials 0.03
LMS-PLIF	officials 0.10	<unk> 0.05	spending 0.05	and 0.03	contracts 0.03
MoS	spending 0.12	**subsidies 0.09**	payments 0.08	costs 0.03	sales 0.03
MoC	spending 0.09	debt 0.08	payments 0.08	orders 0.04	<unk> 0.03
MoS†	**subsidies 0.13**	spending 0.10	benefits 0.04	costs 0.04	orders 0.03
MoC†	spending 0.09	officials 0.05	debt 0.04	**subsidies 0.03**	<unk> 0.03
MoS‡	**subsidies 0.15**	spending 0.08	officials 0.04	costs 0.04	<unk> 0.04
MoC‡	officials 0.04	figures 0.03	efforts 0.03	<unk> 0.03	costs 0.03

Reference #6	the official china daily said retail prices of <unk> foods have n't risen since last december but acknowledged that huge government **subsidies** were a main factor in keeping prices down <eos>

Table 5.1: Next token prediction for different contexts in the PTB's test set. Top-5 predictions from AWD-LSTM models using different functions are shown. Results for MoS‡ and MoC‡ are those reported in the work of Yang et al. [86]. As MoS‡ and MoC‡ use NT-ASGD, our reproduced versions MoS† and MoC† also use NT-ASGD to be comparable with them. The rest of the models use ET-ASGD.

Here, we discuss the results from the qualitative analysis, based on Table 5.1:

- Yang et al. [86] had showed that MoS‡ makes better context-dependent predictions than MoC‡ in the chosen set of contexts as MoS‡ was able to get the true next token among its top-5 predictions. Whereas, MoC‡ was not able to get the true next token for all of the chosen contexts.
- However, our reproduced version MoC†, which is slightly better in terms of test perplexity (57.40 as shown in Table 4.6) than MoC‡ (57.55 as shown in Table 3.1), was able to get the true next token among its top-5 predictions for contexts 1, 4, and 6.
- Because of the use of ET-ASGD, MoC has an even lower test perplexity (55.81) compared to MoC† (57.30) as seen in Table 4.6. It was able to get the true next token for contexts 1, 2, and 3. However it was not able to get for contexts 4 and 6 that MoC† was able to get.
- Among the chosen set of contexts, not only were the high-rank models[1] GSS, MoS, MoS†, and MoS‡ able to get the true next token, but also the low-rank models such as Softmax (for context 1, 3, 5), LMS-PLIF (for contexts 1, 2, 3), MoC (for contexts 1, 2, 3), and MoC† (for contexts 1, 4, 6) were able to get it.
- As the prediction probabilities for the true next token conditioned on most of the chosen contexts, computed by all the models are very small, arriving at a conclusion like attributing it to lower test perplexity or to high rank is not fair. An example is context 5 where MoS‡ was able to get the true next token, but MoS, which is far better in terms of test perplexity and has a similar rank, did not get it.

[1]By model X, we actually mean an AWD-LSTM using function X. For brevity, we will use mentions like this in several places.

It is now evident that there is no strong connection between the qualitative performance of a model and its ability to produce a high-rank $\mathbf{Q}_\theta$.

5.3 Analysis on Embedding Matrix

In a standard AWD-LSTM model, we know that the output layer (or the softmax layer) consists of two main components:

1. A trainable word embedding matrix, as softmax and embedding layers are tied.
2. A softmax function that outputs a probability distribution.

Because of differences in the rank of log probability matrices produced by AWD-LSTM models with different functions, there would be differences in the gradients that flow back through the network. Among the trainable parameters which could get impacted in such scenarios, word embeddings are the ones that can be evaluated using different well-known methods.

5.3.1 Word Similarity

Though most words do not have many synonyms, they do have a lot of similar words [39]. For example, *black* is not a synonym of *white*. However they are similar words, as both are colors. Word similarity is a task wherein the representations of words are evaluated on similar pair of words to check how well they were able to capture the similarity, i.e., whether the representations are close enough.

5.3.1.1 Benchmark Data Sets

Inspired by the work of Faruqui and Dyer [19] on the evaluation of word embeddings, we evaluate the word embeddings (also known as word vectors) that are present within our learned language models on 13 different word similarity benchmark data sets. WS-353 [20] contains 353 pairs of words with their similarity scores assigned by humans. Agirre et al. [2] divided WS-353 into two such that one is focussed on similarity (WS-353-SIM [2]) and the other is focused on relatedness [2] (WS-353-REL [2]). WS-353 contain words of different

[2]For example, *coffee* and *cup* are not similar but they are related because of their co-participation most of the times [39]

types whereas RG-65 [65], MC-30 [53] contain only noun pairs and YP-130 [85], Verb-143 [4], SimVerb-3500 [25] contain only verb pairs. Hill et al. [31] introduced SimLex-999 which has a much stricter notion for similarity wherein relatedness and associations are ignored or have a lower similarity score. MTurk-287 [62], MTurk-777 [28], and MEN [8] are crowdsourced data sets constructed using the Amazon Mechanical Turk. Luong et al. [44] introduced the RW-STANFORD data set which contains rarely used words.

As the vocabularies constructed from the PTB and WT2 data sets contain only the top-N frequent tokens (words and other symbols), a lot of words from the similarity benchmark data sets are absent in these vocabularies. If a word from any pair present in a similarity benchmark data set is absent (as shown in Table 5.2) in these vocabularies, the words in the pair are not considered during the evaluation.

Word similarity benchmark	**#Word pairs in benchmark**	**#Word pairs not in PTB**	**#Word pairs not in WT2**
WS-353	353	116	48
WS-353-SIM	203	66	28
WS-353-REL	252	80	28
RG-65	65	55	20
MC-30	30	21	4
MTurk-287	287	146	106
MTurk-771	771	346	99
MEN	3,000	1,952	863
YP-130	130	65	43
VERB-143	144	9	0
RW-STANFORD	2,034	1,889	1,605
SimVerb-3500	3,500	1,746	1,080
SimLex-999	999	424	106

Table 5.2: Comparison of word pairs in the similarity benchmark data sets and vocabularies constructed from the language modeling data sets.

5.3.1.2 Performance Measure

The benchmark data sets have similarity scores assigned by humans, which usually are in different ranges say [1, 10], [1, 5] etc. However, the similarity between vector representations of pairs of words is calculated using cosine similarity which is in the range [0, 1]. Similar to

[19], we report the Spearman's rank correlation coefficient ρ between the rankings of word vector similarity scores computed by us and the rankings of similarity scores assigned by humans in the similarity benchmark data sets.

Spearman's rank correlation coefficient is a non-parametric measure of the monotonicity of the relationship between the two sets of observations X and Y of same length n. Let $R(X)$ be the ordering (ranking) of observations in X from smallest to largest. Similarly, let $R(Y)$ be the ordered observations of Y. Then Spearman's rank correlation coefficient ρ is calculated as:

$$\rho = \frac{\frac{1}{n}\sum_{i=1}^{n}(R(x_i) - \overline{R(X)})(R(y_i) - \overline{R(Y)})}{\sqrt{\left(\frac{1}{n}\sum_{i=1}^{n}\left(R(x_i) - \overline{R(X)}\right)^2\right)\left(\frac{1}{n}\sum_{i=1}^{n}\left(R(y_i) - \overline{R(Y)}\right)^2\right)}} \tag{5.1}$$

where $\overline{R(X)}$ and $\overline{R(Y)}$ are the means of ordered observations of X and Y. In our case, X is the word vector similarity scores computed using cosine similarity and Y is the human assigned similarity scores. The value of ρ varies between -1 and 1 with 0 meaning no correlation, -1 meaning perfect negative correlation, and +1 meaning perfect positive correlation.

5.3.1.3 Experimental Results and Discussion

Word embeddings from the AWD-LSTM models with different functions such as Softmax, SS, GSS, LMS-PLIF, MoS, and MoC trained on the PTB and WT2 data sets are evaluated on 13 word similarity benchmark data sets. The evaluation results of the embeddings from these language models trained on PTB and WT2 are shown in tables 5.3 and 5.4, respectively.

As MoS and MoC use a smaller embedding dimension d on both PTB and WT2, they are compared separately (as seen in figures 5.3a and 5.3e) in addition to their comparison with models having the embedding dimension of $d = 400$ (as seen in figures 5.3c and 5.3d). Though there is no single model whose embeddings perform the best on all word similarity benchmarks, there are definitely a few models that had performed better than the rest in most of the benchmarks. We discuss them as follows:

- Embeddings from MoC trained on both PTB and WT2 fared better than those from MoS. When trained on PTB, its embeddings performed better than those of MoS by having better correlation coefficients on 8 word similarity benchmarks whereas MoS's

Benchmark	Softmax	SS	GSS	LMS-PLIF	MoS	MoC
WS-353	0.4160	0.3968	0.3949	0.4167	0.3609	0.4025
WS-353-SIM	0.4550	0.4462	0.4507	0.4710	0.3846	0.4451
WS-353-REL	0.3774	0.3491	0.3470	0.3714	0.3399	0.3361
RG-65	0.3697	0.5030	0.5152	0.5152	0.2485	0.6121
MC-30	0.3833	0.4667	0.3833	0.3500	0.1333	0.4167
MTurk-287	0.6086	0.6153	0.5918	0.5843	0.6171	0.5857
MTurk-771	0.4273	0.4341	0.4378	0.4199	0.3985	0.4186
MEN	0.4299	0.4460	0.4355	0.4298	0.3789	0.4337
YP-130	0.1734	0.1657	0.1190	0.1279	0.2817	0.2780
VERB-143	0.4388	0.4350	0.4599	0.4534	0.4672	0.4358
RW-STANFORD	0.4787	0.4676	0.4527	0.4819	0.4904	0.4603
SimVerb-3500	0.1185	0.1212	0.1260	0.1161	0.1133	0.1331
SimLex-999	0.2273	0.2067	0.2361	0.2060	0.1887	0.1950

Table 5.3: Spearman's rank correlation coefficient ρ values on different word similarity benchmarks for embeddings from language models trained on PTB.

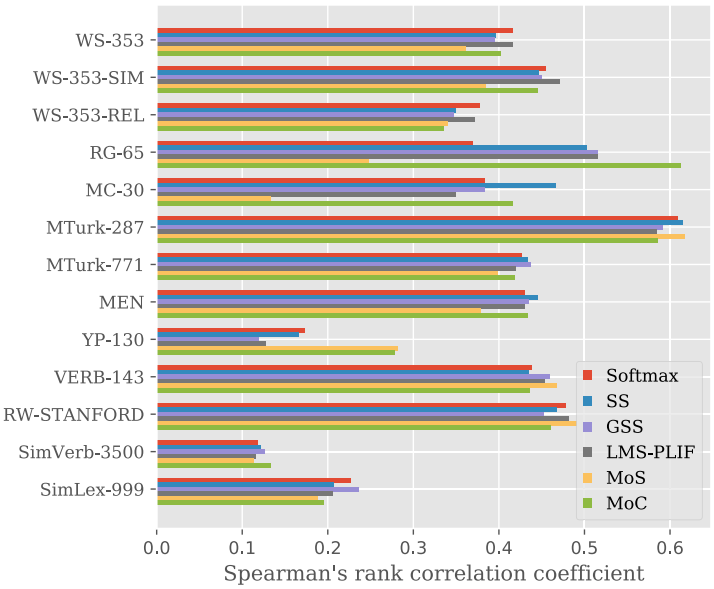


Figure 5.1: A grouped bar chart for the results in Table 5.3.

Benchmark	Softmax	SS	GSS	LMS-PLIF	MoS	MoC
WS-353	0.4658	0.4691	0.4799	0.4657	0.4155	0.4676
WS-353-SIM	0.5925	0.6007	0.6077	0.6022	0.5551	0.5872
WS-353-REL	0.3759	0.3905	0.3933	0.3654	0.3238	0.3777
RG-65	0.5701	0.5368	0.5547	0.5231	0.4868	0.5426
MC-30	0.7308	0.7627	0.7442	0.7247	0.6050	0.7490
MTurk-287	0.5405	0.5682	0.5634	0.5485	0.5685	0.5068
MTurk-771	0.4483	0.4559	0.4581	0.4450	0.4129	0.4425
MEN	0.5895	0.5883	0.5965	0.5830	0.5399	0.5659
YP-130	0.1889	0.2127	0.2388	0.2272	0.1665	0.2117
VERB-143	0.4268	0.4306	0.4401	0.4253	0.4541	0.4646
RW-STANFORD	0.4565	0.4698	0.4582	0.4521	0.4487	0.4781
SimVerb-3500	0.1243	0.1283	0.1288	0.1283	0.1438	0.1515
SimLex-999	0.2432	0.2337	0.2276	0.2325	0.1783	0.2175

Table 5.4: Spearman's rank correlation coefficient ρ values on different word similarity benchmarks for embeddings from language models trained on WT2.

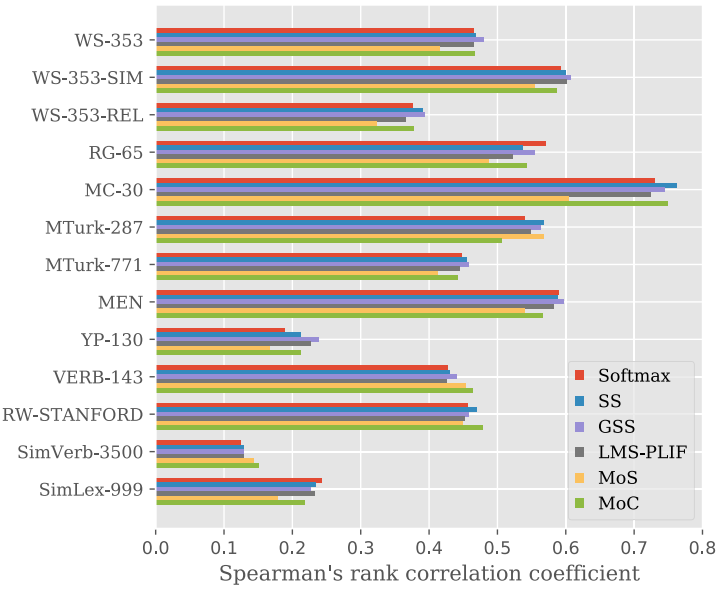
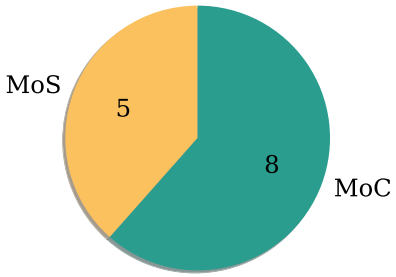
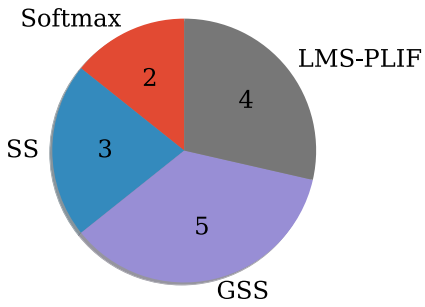


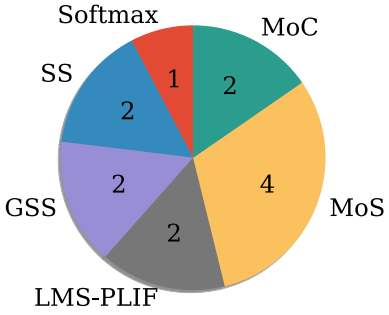
Figure 5.2: A grouped bar chart for the results in Table 5.4.



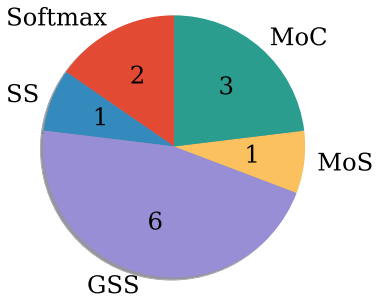
(a) Models (PTB) with $d = 280$



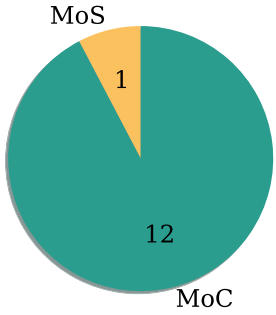
(b) Models (PTB) with $d = 400$



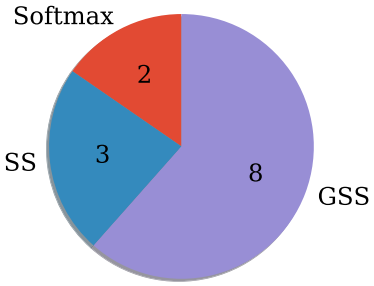
(c) All models (PTB)



(d) All models (WT2)



(e) Models (WT2) with $d = 300$



(f) Models (WT2) with $d = 400$

Figure 5.3: Number of word similarity benchmark data sets against which the similarity scores of embeddings from different language models (trained on PTB and WT2) have the highest Spearman's rank correlation ρ values. d denotes the embedding dimension.

embeddings had better correlation coefficients only on 5 benchmarks (as seen in Table 5.3 and Figure 5.3a). Similarly on WT2, MoC's embeddings topped on 12 benchmarks leaving only 1 to MoS's (as seen in Table 5.4 and Figure 5.3e).

- Among the models whose embedding dimension is $d = 400$, GSS fared better than the rest. When trained on PTB, GSS's embeddings had the highest correlation on 5 benchmarks (as seen in Figure 5.3b). Similarly on WT2, its embeddings had the highest correlation on 8 benchmarks (as seen in Figure 5.3f).
- When embeddings from all models (ignoring their embedding dimensions) are compared against each other, MoS's embeddings had the highest correlation on 4 benchmarks when trained on PTB and GSS's embeddings had the highest correlation on 6 benchmarks when trained on WT2.

On a surface level, when all models are compared (as seen in figures 5.3c and 5.3d), it might look that high-rank models such as MoS and GSS have better embeddings in terms of their performances on word similarity benchmarks. However, this is not always true. The comparison between MoS and MoC (as seen in figures 5.3a and 5.3e) tells us a different story. MoC, a low-rank model with a higher test perplexity than that of MoS on both data sets, has better embeddings.

5.3.2 Representation Degeneration

Though the authors of cosine regularization (MLE-CosReg) and spectrum control (SC) had claimed that their techniques help in solving the fast singular value decay phenomenon, the relative improvements in terms of test perplexity were not large for AWD-LSTM models. Also, in both the works, figures depicting the spectrum (i.e., distribution of singular values) before and after the use of their techniques were not shown for the AWD-LSTM models. Because of the above-mentioned unconvincing observations, we attempt to answer the following two questions,

1. Does the AWD-LSTM model really have a fast singular value decay phenomenon for its embedding matrix in the first place?
2. If it has, do the solutions to break the softmax bottleneck indirectly impact the decay rate of the embedding matrix's singular values?

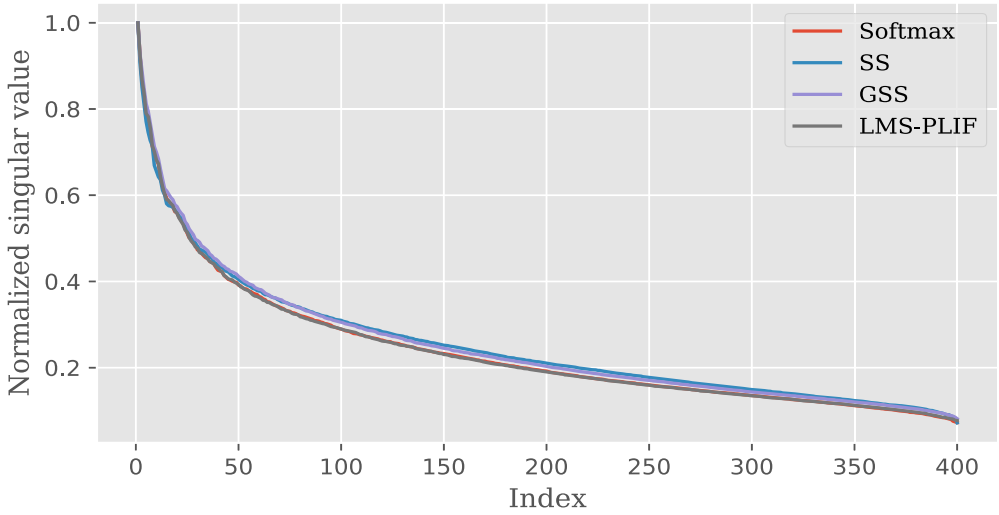


Figure 5.4: Spectrum of embedding matrix for AWD-LSTM with Softmax, SS, GSS, and LMS-PLIF trained on PTB.

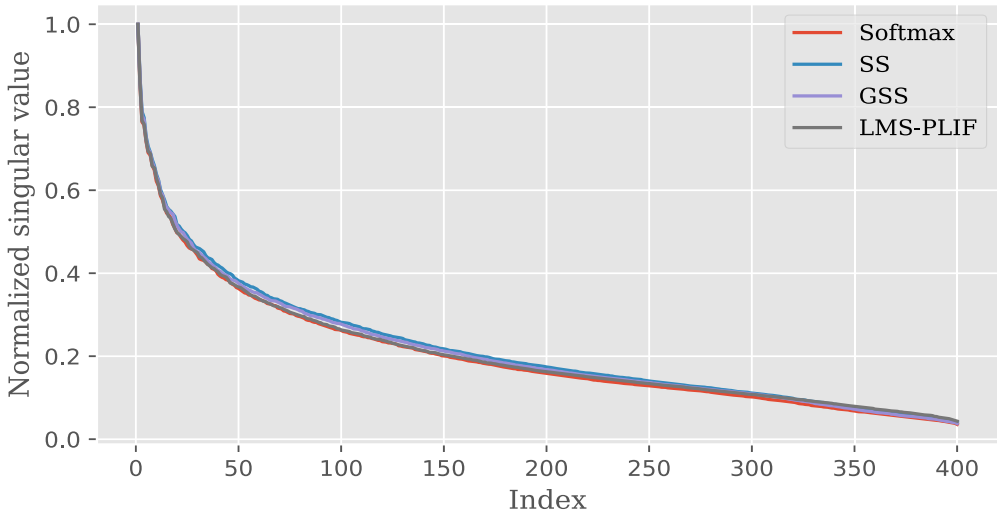


Figure 5.5: Spectrum of embedding matrix for AWD-LSTM with Softmax, SS, GSS, and LMS-PLIF trained on WT2.

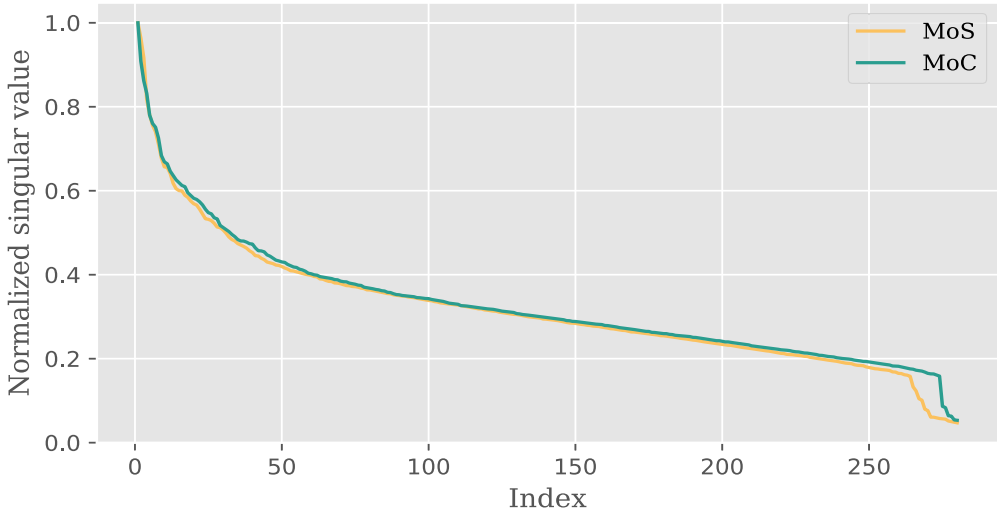


Figure 5.6: Spectrum of embedding matrix for AWD-LSTM with MoS and MoC trained on PTB.

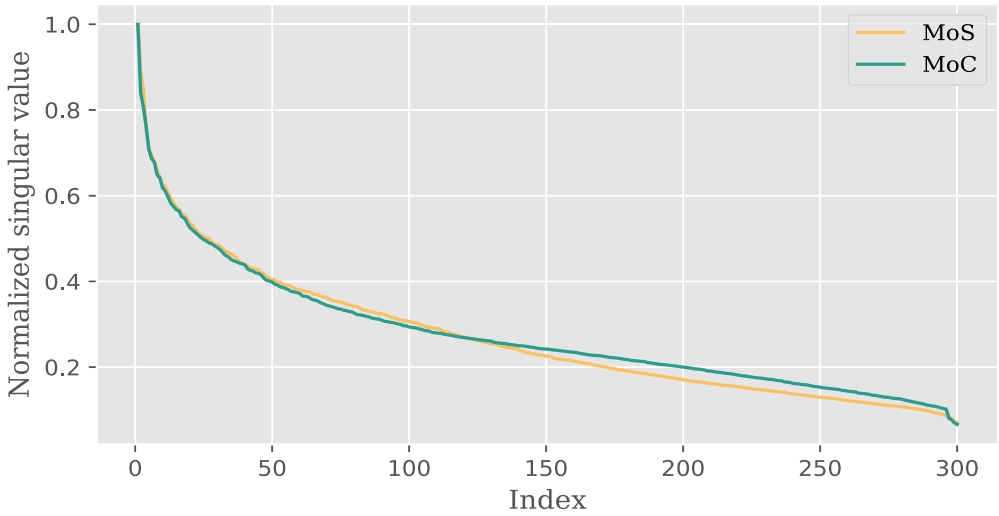
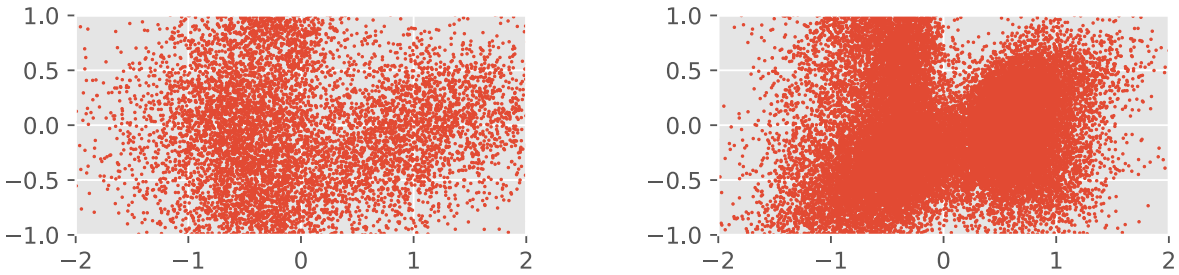


Figure 5.7: Spectrum of embedding matrix for AWD-LSTM with MoS and MoC trained on WT2.

As seen in figures 5.4, 5.6, 5.5, and 5.7, AWD-LSTM language models with different functions do not seem to suffer from the phenomenon of fast singular value decay, unlike the Transformer models for language generation tasks [23, 82].



(a) Trained on PTB (b) Trained on WT2

Figure 5.8: 2D projections of embeddings from an AWD-LSTM with softmax.

However, there are slight differences in the normalized distribution of singular values for AWD-LSTM models using different functions. But they do not lead to representation degeneration, as the 2D projections of the embeddings are well distributed around the origin of the system of coordinates, as seen in Figure 5.8, unlike Transformer models as seen in Figure 2.8b.

Hence, we suspect that the techniques [23, 81, 82], which are believed to alleviate the representation degeneration for AWD-LSTM models, are rather just regularizing the model, directly or indirectly. To investigate further, we conducted an ablation study on Adversarial Softmax and found some supporting evidence that are discussed in appendix .1. As the complete set of hyperparameter configuration for MLE-CosReg and Spectrum Control were not available publicly, we restricted our study to only the Adversarial Softmax.

5.4 Correlation between Rank and Other Metrics

To verify the role of rank, Yang et al. [86] had conducted an experiment on the PTB data set and claimed (Table 5.5) that "When all other hyperparameters are fixed for MoS, increasing the number of mixture components increases the rank of log probablity matrix $\mathbf{Q}_\theta$ thus improving the test performance (lower test perplexity). Also, this positive correlation between rank and performance is what makes MoS better in terms of context-dependent predictions." As this was the key and the only experiment conducted by the authors of

MoS to justify that their model is able to get better test performance because of high-rank $\mathbf{Q}_\theta$, we attempt to verify the above-mentioned claim by conducting rigorous experiments on the PTB and WT2 data sets.

#Mixture	**Test ppl**	**Rank**
3	58.62	6,467
5	57.36	8,930
10	56.33	9,973
15	55.97	9,981
20	56.17	9,981

Table 5.5: Reported [86] correlation between rank and perplexity on PTB's test set for AWD-LSTM-MoS model.

#Mixture	**#Param**	**Train ppl**	**Test ppl**	**Rank**
		PTB		
1	19.05M	56.42	64.50	282
3	19.40M	41.77	59.25	5,575
5	19.75M	38.09	58.38	8,057
10	20.62M	35.48	56.21	9,976
15	21.50M	33.08	56.07	9,979
20	22.37M	32.19	56.19	9,980
		WT2		
10	33.92M	36.94	62.65	12,198
15	34.90M	35.92	63.06	13,229
20	35.88M	35.35	62.76	13,998

Table 5.6: AWD-LSTM-MoS for different number of mixtures.

To reproduce the results reported in [86], we fixed all the hyperparameters of the AWD-LSTM-MoS model, except for the the number of mixtures. The reproduced results are shown in Table 5.6. The table shows that there is a positive correlation between test performance and the rank of $\mathbf{Q}_\theta$ on test sets of both PTB and WT2. Another important detail that needs to be noted is that increasing the number of mixture of components increases the total number of parameters in the model. This could make the model to

overfit, which is evident as there is a decrease in training perplexity with an increasing generalization gap. Note that there is also another positive correlation between training performance and the rank of $\mathbf{Q}_\theta$ on the test set.

Now, the question is whether the positive correlation between the test performance and the rank of $\mathbf{Q}_\theta$ on the test set is always true. To answer this question, we conduct a few more experiments. We empirically found that the AWD-LSTM-MoS model is highly sensitive to the dropout applied on the hidden layer introduced by MoS, which we call as the MoS dropout. Hence, we did an experiment by varying the MoS dropout, but fixing rest of the hyperparameters. In Table 5.7, what we observe is contradictory. We see that the test performance is not positively correlated with the rank of $\mathbf{Q}_\theta$ on the test set, unlike the performance on the training set.

#Mixture = 15			
MoS dropout	**Train ppl**	**Test ppl**	**Rank**
PTB			
0.29	33.08	56.07	9,979
0.145	29.21	59.09	9,985
0.00	23.81	64.82	9,992
WT2			
0.29	39.11	63.06	13,215
0.145	32.19	64.38	17,256
0.00	27.51	68.49	19,427

Table 5.7: AWD-LSTM-MoS for different MoS dropout rates.

Though we are not sure about the cause of the positive correlation between the training performance and the rank of $\mathbf{Q}_\theta$ on the test set, we wanted to check whether such a correlation occurs in other scenarios too. As discussed in Section 2.1.8.4, flooding is a regularization technique. However, there is also another use case of flooding, where we can use it to make different models under comparison to stay around a particular training loss (flooding level) during the process of training, so that they can be fairly compared against each other, irrespective of their individual capacities. We compare MoS with different number of mixtures on the PTB data set by setting the flooding level to 1.0 (i.e., the training loss per batch should stay around 1.0). We turned off all other regularization techniques.

#Mixture	Train ppl	Test ppl	Rank
3	2.90	3153.96	10,000
5	2.87	2937.25	10,000
10	2.90	3183.86	10,000
15	2.88	3071.22	10,000
20	2.89	2997.62	10,000

Table 5.8: Highly overfitting AWD-LSTM-MoS on PTB for different number of mixtures.

c	k	Train ppl	Test ppl	Rank
		PTB		
-2.00	1.25	33.38	57.22	1,694
0.00	1.50	33.41	56.87	4,876
-2.00	3.00	33.72	57.04	9,473
		WT2		
1.00	2.00	38.87	64.27	4,622
-1.50	3.00	39.12	64.38	10,145

Table 5.9: Regularized AWD-LSTM-GSS for different c and k.

c	k	Train ppl	Test ppl	Rank
		PTB		
-2.00	1.25	2.86	3568.03	7,488
0.00	1.50	2.88	4250.78	9,976
-2.00	3.00	2.86	5027.36	10,000
		WT2		
1.00	2.00	2.86	41782.87	17,059
-1.50	3.00	2.90	37246.44	24,006

Table 5.10: Highly overfitting AWD-LSTM-GSS for different c and k.

As shown in Table 5.8, once again, we observe a positive correlation between the training performance and the rank of $\mathbf{Q}_\theta$ on the test set and a negative correlation between the test performance and the rank. We conducted a similar experiment on GSS as well, and observed the same. Comparing the results shown in Table 5.10 with those in Table 5.9, we see a positive correlation between the training performance and the rank for chosen c and k. The models with different c's and k's in Table 5.9 were trained with all the regularization techniques required for better generalization. Whereas the ones in Table 5.10 have only the flooding technique enabled and the rest of the techniques turned off, to make the model overfit.

From all these experiments, there are two things that look clear:

1. Positive correlation between rank and test performance is not always true.
2. Calculated rank values seem to be volatile as they exhibit puzzling correlations with other metrics.

Instead of relying only on the value of the rank, we decided to inspect the actual singular value distribution of $\mathbf{Q}_\theta$ for different models. A similar inspection was done earlier by the authors of SigSoftmax (SS) [40]. We plotted the singular value distribution resulting from the SVD of $\mathbf{Q}_\theta$ for different models trained on both the PTB (Figure 5.9) and the WT2 (Figure 5.11) data sets.

The plots seen in figures 5.9 and 5.11 are similar to those reported in the work of SS [40]. The authors of SS and MoS [40, 86] had argued that slower the decay of singular values, better the performance. However, an important information that was poorly emphasized in the work of SS [40] was the use of log scale for y-axis in both plots. So we refine the plots by using a linear scale for y-axis. As the singular values are different for different $\mathbf{Q}_\theta$'s, we normalized the singular values to be in the range [0, 1]. The construction of this kind of plot is similar to the plots that were used to check for the problem of representation degeneration. The refined version of the plots for both PTB and WT2 data sets are shown in figures 5.10 and 5.12, respectively.

It looks like that it is actually the log probability matrices $\mathbf{Q}_\theta$'s of different AWD-LSTM models that experience the fast singular value decay phenomenon. All AWD-LSTM models with different functions have more or less the same distribution of normalized singular values. The differences in the distribution of the singular values are only visible in log scale. Therefore, we argue that such small differences cannot be the cause for performance improvements, contrary to the belief of authors of SS and MoS.

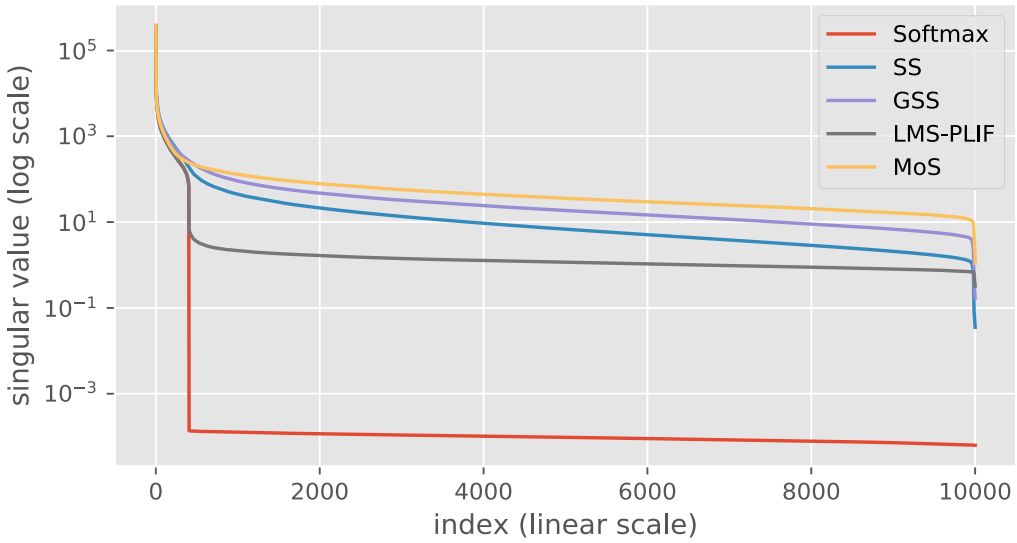


Figure 5.9: Singular values of $\mathbf{Q}_\theta$ on PTB's test set.

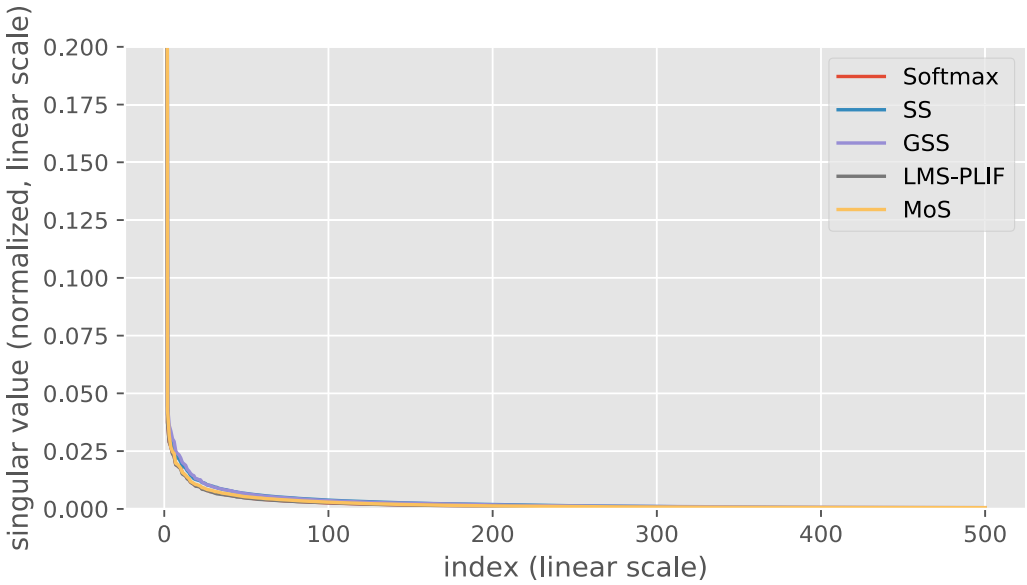


Figure 5.10: Normalized singular values [0,1] of $\mathbf{Q}_\theta$ on PTB's test set. For better visibility, x-axis limited to show first 500 indices and y-axis limited to show [0, 0.2].

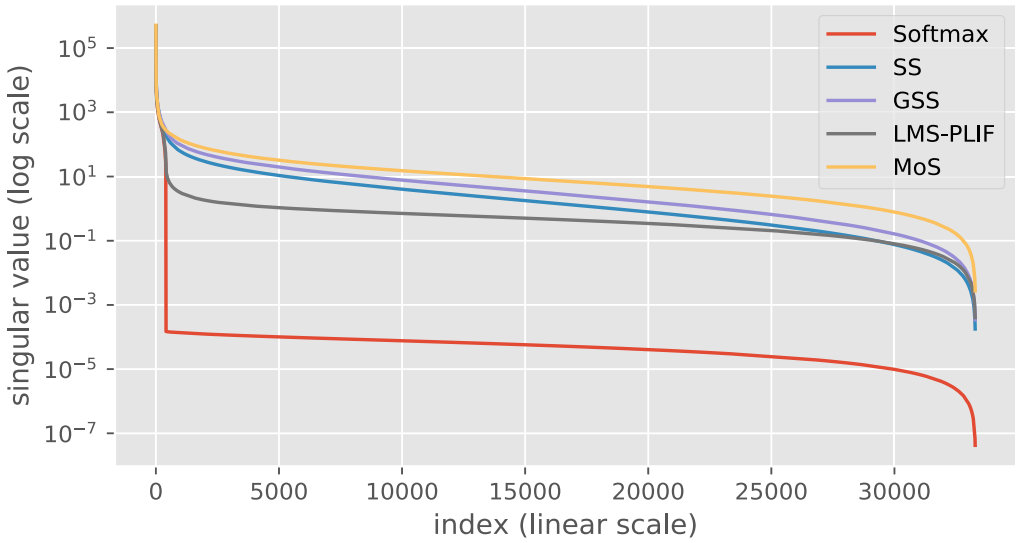


Figure 5.11: Singular values of $\mathbf{Q}_\theta$ on WT2's test set.

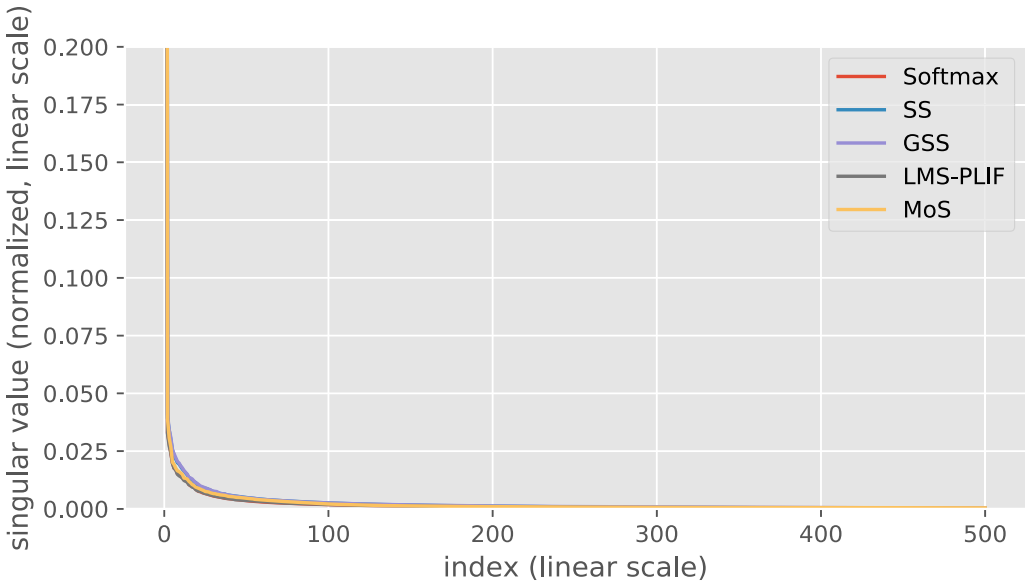


Figure 5.12: Normalized singular values [0,1] of $\mathbf{Q}_\theta$ on WT2's test set. For better visibility, x-axis limited to show first 500 indices and y-axis limited to show the interval [0, 0.2].

5.5 Robustness of Press' Rank

As the rank values for large real-valued matrices are typically calculated using Press et al. [61]'s approach (which we call as Press' rank), we wanted to check how robust is it to tiny noises. We conducted two experiments to check the sensitivity of Press' rank. For both experiments we use MoS's $\mathbf{Q}_\theta$ constructed from PTB's test set. We know that $\mathbf{Q}_\theta \in \mathbb{R}^{82{,}430 \times 10{,}000}$ as there are 82,430 contexts in the test set and 10,000 words in the vocabulary of PTB. We need both $\mathbf{U}$ and $\mathbf{V}$ in addition to $\mathbf{S}$ as outputs from SVD routine for our experiments. However, in practice, computing $\mathbf{U}$ and $\mathbf{V}$ in addition to $\mathbf{S}$ from Thin SVD routines is very time consuming when compared to only computing $\mathbf{S}$. Hence, we consider only the top 10,000 contexts instead of 82,430 instances for these experiments, i.e., $\mathbf{Q}_\theta \in \mathbb{R}^{10{,}000 \times 10{,}000}$. For this truncated $\mathbf{Q}_\theta$, the rank is 8,933 when calculated using Press et al. [61]'s approach. Remember that the rank of $\mathbf{Q}_\theta$ for MoS was 9,979 (as seen in Table 4.6) when all the contexts from the test set were used to construct $\mathbf{Q}_\theta$. But this difference in rank does not impact our experiments as we just need a $\mathbf{Q}_\theta$ that occurs in practice rather than a randomly generated matrix.

5.5.1 To Reconstruction Errors

Algorithm 5.1

Input:
$\mathbf{Q}_\theta \in \mathbb{R}^{m \times n}, m \geq n$

$\mathbf{U}, \mathbf{S}, \mathbf{V}^T \leftarrow svd(\mathbf{Q}_\theta)$
$\mathbf{S}_p \leftarrow prune(\mathbf{S}, 10)$ ▷ retains only top-10 values. puts the rest to 0.
$i \leftarrow 10$
while $i \neq 0$ **do**
    $\mathbf{U}_q, \mathbf{S}_q, \mathbf{V}_q^T \leftarrow svd(\mathbf{S}_p)$
    $\mathbf{S}_p = \mathbf{U}_q \mathbf{S}_q \mathbf{V}_q^T$
    $print(rank(\mathbf{S}_p))$ ▷ $rank(.)$ uses Press et. al [61]'s approach.
    $i \leftarrow i - 1$
end while

As we want the noise to be something that is unavoidable and that occurs naturally, we check for the robustness of Press' rank to errors that arise when reconstructing the original matrix from SVD factorized matrices. First, we do an SVD for $\mathbf{Q}_\theta$ to calculate the matrices

$\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}^T$. We keep the top-10 singular values of $\mathbf{S}$ and set the rest to zero. Now, the product $\mathbf{USV}^T$ matrix is of rank 10. We repeatedly factorize and reconstruct this low-rank matrix and check if its rank increases when calculated using Press et al. [61]'s approach. The steps of the algorithm are outlined in Algorithm 5.1. The results for this experiment are shown in Table 5.11. Though the reconstruction error increases after every iteration, Press' rank seems to be robust.

Iteration	1	2	3	4	5	6	7	8	9	10
Rank	10	10	10	10	10	10	10	10	10	10
Error $\times 10^{-28}$	0.97	0.99	1.01	1.05	1.08	1.10	1.13	1.17	1.23	1.28

Table 5.11: Robustness to SVD reconstruction errors.

5.5.2 To Gaussian Additive Noise

Algorithm 5.2

Input:

$\mathbf{Q}_\theta \in \mathbb{R}^{m \times n}, m \geq n$
$c, s \in \mathbb{R}$
$j \in \mathbb{Z}$

1: $\mathbf{U}, \mathbf{S}, \mathbf{V}^T \leftarrow svd(\mathbf{Q}_\theta)$ ▷ $\mathbf{S} \in \mathbb{R}^{n \times n}$
2: $\mathbf{S}_p \leftarrow prune(\mathbf{S}, 10)$ ▷ retains only top-10 values. makes the rest as 0.
3: **while** $j \neq 0$ **do**
4: $\quad \mathbf{Z} \leftarrow gaussian\_random(0, c/n^2)$
5: $\quad \mathbf{S}_g = \mathbf{S}_p + \mathbf{Z}$
6: $\quad print(rank(\mathbf{S}_g))$ ▷ $rank(.)$ uses Press et. al [61]'s approach.
7: $\quad c \leftarrow c + s$
8: $\quad j \leftarrow j - 1$
9: **end while**

Now, we check for the robustness of Press' rank to very small gaussian additive noises. The steps of the algorithm are outlined in Algorithm 5.2. The procedure is same as our previous experiment till the step of making the rank of the product matrix $\mathbf{USV}^T$ equal to 10. The parameter c which is fed as an input to the algorithm controls the standard

deviation of the zero-mean gaussian distribution from which the additive noise is sampled. Iteratively, c is changed according to the step size s which is another parameter of the algorithm. For different additive noises, we look for change in Press' rank for the matrix which has an actual rank of 10.

We inspect the behavior of Press' rank by using different step sizes. The results for different c's are shown in Table 5.12. Results for step sizes in the power of 2 are shown in the first sub-table, for step size of 2 in the second sub-table, and for step size of 1 in the third sub-table. We can see that for c's lower than 10^{-25}, the rank of the matrix is computed correctly using Press et al. [61]'s approach. However, for just a few tens of higher magnitudes, the value of Press' rank becomes close to full-rank.

c	10^{-2}	10^{-4}	10^{-8}	10^{-16}	10^{-32}
Rank	10,000	10,000	10,000	9,998	10

c	10^{-20}	10^{-22}	10^{-24}	10^{-26}	10^{-28}
Rank	9,899	9,001	1,159	10	10

c	10^{-21}	10^{-22}	10^{-23}	10^{-24}	10^{-25}
Rank	9,681	9,001	6,869	1,159	10

Table 5.12: Robustness to gaussian additive noise.

5.6 Epsilon Effective Rank

We know that the singular values are present along the main diagonal of the matrix $\mathbf{S}$, that is obtained from the SVD of any real matrix $\mathbf{A}$. Assuming $\mathbf{S}$ to be a square matrix of dimension $n \times n$, then $s_{11}, s_{22}, .., s_{nn}$ are the singular values. The LAPACK routine that is used for the computation of SVD, conventionally, returns $\mathbf{S}$ with singular values along its diagonal sorted in decreasing order. Hence, we represent them as a sequence $(s_{11}, s_{22}, .., s_{nn})$. In the need for an alternative metric which solely depends on the singular values to calculate the rank, we define ϵ-effective rank as follows:

$$\sum_{i=1}^{k} s_{ii}^2 \geq (1-\epsilon) \sum_{i=1}^{n} s_{ii}^2 \quad \forall \epsilon \in [0,1] \tag{5.2}$$

The smallest value for k, to which the inequality mentioned in equation 5.2 holds true, is defined as the ϵ-effective rank. In words, "the smallest number of singular values whose squares sum to equal or more than $1 - \epsilon$ fraction of the total sum of squares of singular values is called the ϵ-effective rank". One major advantage of this metric is that we can approximately quantify the singular value distribution (and how fast the singular values drop) when different values for ϵ are used. We repeat the same experiment for checking robustness of Press' rank to gaussian noise, but this time we also compute ϵ-effective rank for different ϵ's in addition to Press' rank to make a comparison between the two different metrics for rank calculation. The results are shown in Table 5.13.

c	**Press' rank**	**Effective rank** for various ϵ		
		10^{-3}	10^{-4}	10^{-5}
10^{-20}	9,899	7	10	10
10^{-21}	9,681	7	10	10
10^{-22}	9,001	7	10	10
10^{-23}	6,869	7	10	10
10^{-24}	1,159	7	10	10
10^{-25}	10	7	10	10

Table 5.13: Comparison between Press' rank and ϵ-effective rank.

Function	**Press' rank**	**Effective rank** for various ϵ		
		10^{-3}	10^{-4}	10^{-5}
Softmax	402	52	201	306
SS	4,979	100	297	1,038
GSS	8,989	100	559	3,456
LMS-PLIF	580	50	198	335
MoS	9,983	81	1,521	6,428

Table 5.14: Rank of $\mathbf{Q}_\theta$ constructed from the test set of PTB.

Function	**Press' rank**	**Effective rank** for various ϵ		
		10^{-3}	10^{-4}	10^{-5}
Softmax	402	27	141	274
SS	6,590	54	249	1,201
GSS	10,145	60	391	2,988
LMS-PLIF	513	29	150	287
MoS	15,738	49	773	5,982

Table 5.15: Rank of $\mathbf{Q}_\theta$ constructed from the test set of WT2.

From Table 5.13, we see that the use of 0.001 for ϵ seems to be on the stricter side as the actual rank of matrix under consideration is 10. Similarly, 0.0001 seems to be just right

or can also be thought of being on the lenient side. Instead of using only one value for ϵ, using a couple of values starting from 0.001 and further smaller values can more or less present a clear information about the singular value distribution and the effective value of rank that is calculated from it.

We report Press' rank and ϵ-effective rank of $\mathbf{Q}_\theta$ for AWD-LSTM models with different functions trained on PTB and WT2 data sets in tables 5.14 and 5.15 respectively. The effective ranks for the ϵ values of 0.001 and 0.0001 are still low for all AWD-LSTM models with different functions, though the Press' rank is high. Therefore, we argue that Press' rank cannot be the main reason for better performance of these models on the test set.

5.7 Impact on Regularization

We always suspected that regularization could have been the major influencer for models with LMS-PLIF and MoS functions, because they introduce additional trainable parameters and/or have a completely different hyperparameter configurations. Contrarily, models with SS and GSS functions do not introduce any additional trainable parameters and have the same hyperparameter configuration as that of the baseline AWD-LSTM with the softmax function. Here, we conduct experiments to show how regularization actually helps models using LMS-PLIF and MoS functions to perform better.

5.7.1 By Linear Monotonic Softmax with Piecewise Linear Increasing Functions

AWD-LSTM-LMS-PLIF introduces 10^5 additional trainable parameters in comparison to the baseline AWD-LSTM model with softmax function, to learn the slopes of 10^5 lines. Before training, all these 10^5 parameters are initialized with random positive values. Assume that these parameters are not trained, because of which all 10^5 lines would just have a fixed set of randomly initialized slope values that is not changed throughout the training. In such a scenario, what happens is that the logits are transformed by a fixed set of piecewise linear functions. This is technically equivalent to the baseline AWD-LSTM model with softmax, except for the constant perturbations to the logits. In practice, making a layer's parameters not be learned during training and also not participating in backpropagation is known as *freezing* a layer. We compare LMS-PLIF with another version where its PLIF layer is frozen (denoted as LMS-PLIF†) and the baseline model with the softmax function. Each model was trained 11 times using 10 randomly sampled seeds and the reported seed

by its respective authors for random initialization of the parameters in the network. The results are shown in Table 5.16. As we just mentioned that LMS-PLIF† is equivalent to baseline model with Softmax in terms of parameterization, we can see that the total number of trainable parameters is the same for both models. They use the same hyperparameter configuration. By looking at the training and out-of-training (validation, test) performance of LMS-PLIF† in comparison to the softmax function on both the PTB and WT2 data sets, it is evident that it has traded its performance on the training set for small improvements in the performance on the validation and test sets which is essentially a form of regularization. This trade-off can be attributed to the perturbations that are made to the logits.

Function	**#Param**	**Time (epoch)**	**Train ppl**	**Validation ppl**	**Test ppl**
			PTB		
Softmax	24.22M	∼55s	33.91±0.25	59.55±0.12	57.08±0.09
LMS-PLIF	24.32M	∼59s	36.67±0.30	59.08±0.09	56.80±0.11
LMS-PLIF†	24.22M	∼56s	37.16±0.24	59.23±0.11	56.83±0.10
			WT2		
Softmax	33.55M	∼72s	39.32±0.14	67.57±0.14	64.63±0.08
LMS-PLIF	33.65M	∼93s	41.13±0.15	67.15±0.21	64.16±0.17
LMS-PLIF†	33.55M	∼77s	41.67±0.15	67.22±0.13	64.28±0.17

Table 5.16: Effect of freezing the PLIF layer. In LMS-PLIF†, the PLIF layer is frozen.

In order to compute the p-value from a two-sided Wilcoxon Rank-Sum test between samples of test perplexities of LMS-PLIF and that of LMS-PLIF†, we only consider the test perplexities of model instances whose parameters were initialized using 10 randomly sampled seeds (for each model). From Table 5.17, we can see that the performance improvement on test set offered by LMS-PLIF is not statistically significantly better than that of LMS-PLIF†, because the p-values are higher than 5×10^{-2}. In other words, we could also infer that the learned slopes are not better than the randomly initialized slopes for a better test performance, even though they help in increasing the rank of $\mathbf{Q}_\theta$.

Data set	Test ppl		p-value
	LMS-PLIF	LMS-PLIF†	
PTB	56.81 ± 0.11	56.82 ± 0.09	5.45×10^{-1}
WT2	64.15 ± 0.17	64.30 ± 0.17	6.96×10^{-2}

Table 5.17: p-value from a two-sided Wilcoxon Rank-Sum test between sample of test perplexities of LMS-PLIF† and that of LMS-PLIF.

5.7.2 By Mixture of Softmaxes

Among the hyperparameters used for MoS on the PTB data set, two particular hyperparameters that stood out were the dimensions of the embedding $\mathbf{e}_t$ and context $\mathbf{h}_t^3$ vectors which were 280 and 620, respectively. All other models that were built on top of AWD-LSTM had used the same dimension for context and embedding vectors. When AWD-LSTM-MoS has such a high context vector dimension, they could encode much more context information than other AWD-LSTM models which have a lower dimension. Another important hyperparameter difference between the two was the training batch size. MoS used 12 whereas Softmax used 20 for the training batch size. We set out to make a fair comparison between AWD-LSTM-MoS and AWD-LSTM with softmax. However, it is not a straight-forward process. Hence, we do it systematically, as follows:

- First, we make the training batch size for both the models as 12.
- Can we make $dim(\mathbf{e}_t) = dim(\mathbf{h}_t^3)$ for both MoS and Softmax models[3]?
  - For both models, we make the dimension of both the embedding and the context vectors to be 280.

Model	#Param	Train ppl	Validation ppl	Test ppl
MoS†	17.53M	40.82	58.76	56.62
Softmax†	16.35M	47.77	60.84	58.44
Softmax‡	16.35M	36.67	59.31	57.37

[3]By MoS and Softmax model, we mean AWD-LSTM-MoS and AWD-LSTM with softmax, respectively.

Note that MoS† has more number of parameters than that of Softmax†. It is because MoS† has an additional hidden layer in between the top-most LSTM layer and the output layer of the AWD-LSTM network. MoS† and Softmax† use values used by the authors of MoS for rest of their hyperparameters. In the case of Softmax†, if we look at its training perplexity, it looks like the model is underfitting. Hence, instead of using MoS's hyperparameters, we proportionally[4] reduced the values of some of the hyperparameters of the baseline Softmax model and used it for Softmax‡.

- Now with a better set of hyperparameters for Softmax‡, we looked out for even better hyperparameters by searching around this set, in the pursuit to make the model better fit on the test set. In our limited search, we were able to find only one hyperparameter set which was slightly better. Softmax⋄ uses that new set. Similarly, we search around the hyperparameter set of MoS† that resulted in another set which was used in MoS‡.

Model	**#Param**	**Train ppl**	**Validation ppl**	**Test ppl**
MoS†	17.53M	40.82	58.76	56.62
MoS‡	17.53M	42.45	58.29	56.23
Softmax‡	16.35M	36.67	59.31	57.37
Softmax⋄	16.35M	38.81	59.36	57.25

- We realized that with our small-scale hyperparameter search and the Softmax model having a lower number of parameters than that of MoS (even when $dim(\mathbf{e}_t) = dim(\mathbf{h}_t^3) = 280$ for both of them), we could never find a hyperparameter set that could make the performance of the Softmax model to be similar to that of MoS.

- Can we make MoS and Softmax model to have comparable number of parameters?
  - The only feasible way to do this is to increase the embedding and the context vector dimensions in the Softmax model. Hence, we increase them from 280 to 340, resulting in Softmax♣.

[4] We know that Softmax‡ has a lower number of model parameters (16.35M) than that of the original AWD-LSTM model (24.22M). Hence, we proportionally reduce the dropout rates used for different layers to train the original AWD-LSTM model, based on the reduction in the number of parameters in each layer, to make them suitable for Softmax‡.

Model	#Param	Train ppl	Validation ppl	Test ppl
MoS‡	17.53M	42.45	58.29	56.23
Softmax♣	17.56M	33.15	59.45	56.98

– We further regularize Softmax♣ to make it have a similar generalization gap as that of MoS† resulting in Softmax★.

Model	#Param	Train ppl	Validation ppl	Test ppl
MoS‡	17.53M	42.45	58.29	56.23
Softmax♣	17.56M	33.15	59.45	56.98
Softmax★	17.56M	41.17	58.52	56.37

- MoS‡ and Softmax★ were trained using 11 different seeds for random initialization of the parameters in the network which includes 10 randomly sampled integers (for each of them) in addition to the seed value of 28 (that was used to do a hyperparameter search for models using both MoS and Softmax) which was the reported seed for MoS. The p-value is 6.50×10^{-1} when the sample statistic of test perplexities of MoS‡ is 56.41 ± 0.20 and that of Softmax★ is 56.45 ± 0.15. The samples do not include the test perplexity of model instance whose parameters were initialized using the seed 28, hence the difference in the sample statistic of test perplexities when compared to those mentioned in Table 5.18. As the p-value is greater than 5×10^{-2}, MoS‡ is not statistically significantly better than Softmax★.

Model	#Param	Train ppl	Validation ppl	Test ppl
MoS‡	17.53M	42.45±1.03	58.53±0.17	56.39±0.19
Softmax★	17.56M	40.87±0.18	58.61±0.10	56.45±0.10

Table 5.18: Fair comparison between MoS and Softmax on PTB.

With our limited search, we made the performance of AWD-LSTM with softmax as good as that of AWD-LSTM with MoS, and showed the impact on regularization because of the differences in hyperparameters and network structure. However, we are aware that

this is not conclusive, as there could be another set of hyperparameters which can make MoS perform better than softmax, or vice-versa. As this is a laborious and at times a never-ending process, we confine our hyperparameter search to only the PTB data set. Also, in case of the hyperparameters used to train MoS on the WT2 data set, there are a lot more differences than the ones used to train the original AWD-LSTM model with the softmax function. Hence, it is really difficult to even do some sort of systematic search, as we were able to do on the PTB data set.

Another key aspect of MoS is that it uses a mixture of probability distributions instead of only one distribution to predict the next token, which can inherently be advantageous. Also, remember that MoS introduces a fully-connected hidden layer in between the top-most LSTM layer and the output layer. The regularization techniques applied for that layer like MoS dropout (as shown in Table 5.7), can also be a factor for the better performance of MoS. Hence, we argue that the attribution of the better performance of MoS to the high rank of $\mathbf{Q}_\theta$ is not fair.

5.8 Summary

In this chapter:

- We conducted a qualitative analysis for AWD-LSTM models with different functions such as Softmax, SS, GSS, LMS-PLIF, MoS, and MoC, and showed that the rank of $\mathbf{Q}_\theta$ is not correlated with the quality of predictions made by the language model.
- We analyzed the quality of learned embeddings present in different language models, by evaluating them on 13 word similarity benchmark data sets, and showed that the rank of $\mathbf{Q}_\theta$ is not correlated with the quality of the embeddings.
- We showed that the embeddings from the original AWD-LSTM model with the softmax function, do not suffer from the phenomenon of fast singular value decay and the problem of representation generation, unlike the Transformer model.
- We showed that the correlation between rank of $\mathbf{Q}_\theta$ and the performance of a language model in terms of perplexity on the test set is not always true for the AWD-LSTM model with MoS.
- We exposed that the singular value distributions of both low-rank and high-rank $\mathbf{Q}_\theta$'s are similar, and showed that all these distributions exhibit the phenomenon of fast

singular value decay. We argued that the differences in these distributions that are visible only in log scale cannot be responsible for the better performance of a model.

- We questioned the approach of Press et al. [61] that was used to calculate the rank of a matrix, and showed that it was: sensitive to tiny noise, and not able to quantify the similarity in singular value distribution.

- We introduced an alternative approach to calculate the rank of a matrix, which we call the ϵ-effective rank, which is robust than Press' rank and can approximately quantify the singular value distribution when different values for ϵ are used.

- We showed that it is regularization which had helped the AWD-LSTM-LMS-PLIF model to perform slightly better than the AWD-LSTM with the softmax function. Also, we showed that the PLIF layer present within LMS-PLIF does not learn slopes for linear functions that are better than randomly initialized slopes.

- On the PTB data set, we showed that under a fair comparison, when the AWD-LSTM model with the softmax function is regularized well enough, the performance difference between MoS and softmax is not significant. We also argued that MoS was able to perform better because of the hyperparameters used and its network structure, and not because of the high rank of $\mathbf{Q}_\theta$.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, our initial goal was to come up with a non-linear output layer activation function which can make language models produce log probability matrices $\mathbf{Q}_\theta$ with diverse ranks, so that we can better understand about the problem of the softmax bottleneck and its consequences. We introduced Generalized SigSoftmax (GSS) for that purpose, and empirically showed that it can produce $\mathbf{Q}_\theta$ with diverse ranks. With the help of GSS, we were able to show that the ability of language models to produce high-rank $\mathbf{Q}_\theta$ does not guarantee a better performance in terms of perplexity on test set.

We emphasized the importance of using the same hyperparameter configuration when comparing different AWD-LSTM models by exposing the hyperparameter overlooked by many which is the trigger condition to switch from SGD to ASGD. We showed that under fair comparison, existing works like LMS-PLIF, which was earlier believed to have a significant performance improvement, no longer bring that much improvement.

We showed that there is no correlation between a language model's qualitative performance and its ability to produce a high-rank $\mathbf{Q}_\theta$. Also, we showed that a high-rank language model does not necessarily learn better embeddings, by evaluating their learned embeddings on word similarity benchmark data sets.

We exposed that it is $\mathbf{Q}_\theta$ that suffers from the phenomenon of fast singular value decay, and not the embedding matrix as it was believed, with respect to the AWD-LSTM language models. To approximately capture such phenomenon quantitatively and to be more robust to tiny noises, we introduced and suggested the use of ϵ-effective rank.

Finally, we identified that regularization is the cause for the small performance improvement in LMS-PLIF. Regarding MoS, we showed that $\mathbf{Q}_\theta$ being high-rank is most likely a by-product. We argued that its better performance should be attributed to regularization, the inherent advantages that it could have because of its network structure, the choice of hyperparameters, and the fact that the predictions are made using a mixture of probability distributions instead of one distribution.

6.2 Future Work

- As GSS is a generic output layer activation function, its use in other neural network models for multi-class classification problems needs to be explored.
- Better understanding of the connection between the phenomenon of fast singular value decay observed in $\mathbf{Q}_\theta$ and the performance of a language model. If it does degrade the performance, then solving it using approaches similar to spectrum control for embedding matrix could really be worth pursuing.
- Though we found there is no additional benefit that a high-rank language model could have, it will be interesting to see if it actually aids in the better performance for other NLP applications.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27, Boulder, Colorado, June 2009. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N09-1003.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

[4] Simon Baker, Roi Reichart, and Anna Korhonen. An unsupervised model for instance level subcategorization acquisition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 278–289, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1034. URL https://www.aclweb.org/anthology/D14-1034.

[5] William A Belson. Matching and prediction on the principle of biological classification. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 8(2):65–75, 1959.

[6] Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 1994.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145, 2012.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.

[10] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13:21–27, 1967.

[11] Jan Salomon Cramer. The origins of logistic regression. In *Tinbergen Institute Working Paper*, 2002.

[12] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[13] Leonid Datta. A survey on activation functions and their relation with xavier and he normal initialization, 2020.

[14] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. *The Singular Value Decomposition: anatomy of optimizing an algorithm for extreme scale*, 2017 (accessed July 25, 2020). URL http://www.netlib.org/utk/people/JackDongarra/PAPERS/svd-sirev-M111773R.pdf.

[15] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 2010.

[16] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *Advances in neural information processing systems*, pages 472–478, 2001.

[17] Jacob Eisenstein. Natural language processing, 2018.

[18] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[19] Manaal Faruqui and Chris Dyer. Community evaluation and exchange of word vectors at wordvectors.org. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 19–24, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-5004. URL https://www.aclweb.org/anthology/P14-5004.

[20] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414, 2001.

[21] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[22] Octavian Ganea, Sylvain Gelly, Gary Bécigneul, and Aliaksei Severyn. Breaking the softmax bottleneck via learnable monotonic pointwise non-linearities. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97, pages 2073–2082, 2019. URL http://proceedings.mlr.press/v97/ganea19a.html.

[23] Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tieyan Liu. Representation degeneration problem in training natural language generation models. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkEYojRqtm.

[24] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 2000.

[25] Daniela Gerz, Ivan Vulic, Felix Hill, Roi Reichart, and Anna Korhonen. Simverb-3500: A large-scale evaluation set of verb similarity. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2173–2182, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1235. URL https://www.aclweb.org/anthology/D16-1235.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[27] Roger Grosse. Lecture notes on intro to neural networks and machine learning, 2018.

[28] Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1406–1414, 2012.

[29] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[30] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[31] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, December 2015. doi: 10.1162/COLI_a_00237. URL https://www.aclweb.org/anthology/J15-4004.

[32] Geoffrey E Hinton. Distributed representations. 1984.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[34] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *ArXiv*, abs/1611.01462, 2017.

[35] Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Do we need zero training loss after achieving zero training error? *arXiv preprint arXiv:2002.08709*, 2020.

[36] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., USA, 1988. ISBN 013022278X.

[37] Haoming Jiang, Zhehui Chen, Minshuo Chen, Feng Liu, Dingding Wang, and Tuo Zhao. On computation and generalization of generative adversarial networks under spectrum control. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJNH6sAqY7.

[38] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling, 2016. URL https://arxiv.org/pdf/1602.02410.pdf.

[39] Dan Jurafsky and James H.Martin. Speech and language processing (3rd edition), 2019. URL https://web.stanford.edu/~jurafsky/slp3.

[40] Sekitoshi Kanai, Yasuhiro Fujiwara, Yuki Yamanaka, and Shuichi Adachi. Sigsoftmax: Reanalysis of the softmax bottleneck, 2018.

[41] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, 2015.

[42] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[43] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26(2): 195–197, 2008.

[44] Minh-Thang Luong, Richard Socher, and Christopher D. Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, Sofia, Bulgaria, 2013.

[45] Yongyi Mao. Introduction Deep Learning and Reinforcement Learning Lecture 01 : Machine Learning Basics Machine Learning : A Definition. 2018.

[46] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): 313–330, 1993. URL https://www.aclweb.org/anthology/J93-2004.

[47] MATLAB. *9.7.0.1190202 (R2019b)*. The MathWorks Inc., Natick, Massachusetts, 2018.

[48] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

[49] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *ArXiv*, 2018.

[50] T. Mikolov and G. Zweig. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239, 2012.

[51] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

[52] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[53] George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991.

[54] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020.

[55] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[56] Christopher Olah. *Understanding LSTM Networks*, 2015 (accessed July 20, 2020). URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[57] Travis Oliphant. *Guide to NumPy*. 01 2006.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[59] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[60] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *ArXiv*, abs/1608.05859, 2017.

[61] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[62] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In

Proceedings of the 20th international conference on World wide web, pages 337–346, 2011.

[63] T.C. Redman. *Data Driven: Profiting from Your Most Important Business Asset*. Harvard Business Review Press, 2008. ISBN 9781422163641. URL https://books.google.ca/books?id=Q5CJJ2wVkYAC.

[64] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[65] Herbert Rubenstein and John Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8:627–633, 10 1965. doi: 10.1145/365628.365657.

[66] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[67] D. E. Rumelhart and J. L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. The MIT Press, 1987.

[68] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[69] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[70] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[71] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[72] Sho Takase, Jun Suzuki, and Masaaki Nagata. Direct output connection for a high-rank language model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4599–4609, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1489. URL https://www.aclweb.org/anthology/D18-1489.

[73] Tensorflow.org. *Transformer model for language understanding*, 2020 (accessed July 25, 2020). URL https://www.tensorflow.org/tutorials/text/transformer.

[74] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[75] Seiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shuji Suzuki, Kota Uenishi, Brian Vogel, and Hiroyuki Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2002–2011, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330756. URL https://doi.org/10.1145/3292500.3330756.

[76] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[77] Vladimir Vapnik. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[79] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

[80] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/wan13.html.

[81] Dilin Wang, Chengyue Gong, and Qiang Liu. Improving neural language modeling via adversarial training. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors,

Proceedings of the 36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, pages 6555–6565, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/wang19f.html.

[82] Lingxiao Wang, Jing Huang, Kevin Huang, Ziniu Hu, Guangtao Wang, and Quanquan Gu. Improving neural language generation with spectrum control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxY8CNtvr.

[83] Christopher J. Wild and George A.F. Seber. Chance encounters: A first course in data analysis and inference, 1997. https://www.stat.auckland.ac.nz/~wild/ChanceEnc/Ch10.wilcoxon.pdf.

[84] Yonghui Wu. *Smart Compose: Using Neural Networks to Help Write Emails*, 2018 (accessed July 25, 2020). URL https://ai.googleblog.com/2018/05/smart-compose-using-neural-networks-to.html.

[85] Dongqiang Yang and David M. W. Powers. Verb similarity on the taxonomy of wordnet. In *In the 3rd International WordNet Conference (GWC-06), Jeju Island, Korea*, 2006.

[86] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkwZSG-CZ.

[87] Zhilin Yang, Thang Luong, Russ R Salakhutdinov, and Quoc V Le. Mixtape: Breaking the softmax bottleneck efficiently. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5775–5783. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9723-mixtape-breaking-the-softmax-bottleneck-efficiently.pdf.

Appendix

.1 Ablation Study for Adversarial Softmax

In the work of Adversarial Softmax (AdvSoft), Wang et al. [81] had used two techniques which should have been emphasized more. The first is the use of ET-ASGD similar to what the authors of LMS-PLIF [22] did. The second is the use of a gaussian noise that is added to the embeddings before passing them as inputs to the LSTM layers in the AWD-LSTM, which we call as the embedding noise. The main technique proposed by Wang et al. [81] was AdvSoft that involves adversarial perturbation to the embeddings before the softmax function is applied. Note that this perturbation is different from the embedding noise. Hence, we conduct an ablation study for AdvSoft on both the PTB and the WT2 data sets. The hyperparameters and the techniques that are different from the AWD-LSTM are the ones that are focused in the ablation study. The results for the PTB and the WT2 data sets are shown in tables A.2 and A.3 respectively.

Data set	**Switch**	**Train ppl**	**Validation ppl**	**Test ppl**
PTB	NT-ASGD	34.05	60.35	58.07
PTB	ET-ASGD	34.03	59.48	57.10
WT2	NT-ASGD	39.07	68.35	65.28
WT2	ET-ASGD	39.09	67.59	64.56

Table A.1: AWD-LSTM

Compared to the results shown in table A.1, there is no performance improvement in terms of test perplexity on the PTB data set, when AdvSoft is used without embedding noise. The significant improvement is mainly because of the embedding noise and ET-ASGD, and not because of AdvSoft. On the WT2 data set, in addition to the switch and embeddding

Switch	Embedding noise	Train ppl	Validation ppl	Test ppl
NT-ASGD	No	37.02	60.48	57.82
NT-ASGD	Yes	38.02	59.15	56.88
ET-ASGD	No	36.60	59.68	57.43
ET-ASGD	Yes	37.90	58.17	55.97

Table A.2: AWD-LSTM + AdvSoft on PTB

noise, the authors of AdvSoft had also used a dropout rate of 0.55 for the embedding vectors, different from the one used by the original AWD-LSTM which is 0.65. The results shown in table A.3 tells that the performance improvement is because of the switch, embedding noise, and the dropout rate, and not entirely because of the use of AdvSoft.

Switch	Dropout for $\mathbf{e}_t$	Embedding noise	Train ppl	Validation ppl	Test ppl
NT-ASGD	0.55	No	38.23	69.06	66.12
NT-ASGD	0.55	Yes	42.48	66.70	63.97
NT-ASGD	0.65	No	41.56	68.44	65.47
NT-ASGD	0.65	Yes	45.70	67.86	65.40
ET-ASGD	0.55	No	38.23	68.70	65.92
ET-ASGD	0.55	Yes	42.59	65.77	63.26
ET-ASGD	0.65	No	41.61	67.66	64.67
ET-ASGD	0.65	Yes	45.61	66.96	64.06

Table A.3: AWD-LSTM + AdvSoft on WT2

.2 lmkit

As the AWD-LSTM codebase was publicly released by Merity et al. [49] on GitHub platform, several others had cloned it and had added their improvements to it. However, it is difficult to compare different works that are present in multiple GitHub repositories. So, we release a single private repository named as *lmkit* which has the AWD-LSTM, and also the implementations for all other works which were built on top of AWD-LSTM as a solution for softmax bottleneck and representation generation. It is well-modularized and adding

new functionalities can be easier. It currently has the implementations for SigSoftmax (SS), Generalized SigSoftmax (GSS), Linear Monotonic Softmax with Piecewise Linear Increasing Functions (LMS-PLIF), Mixture of Softmaxes (MoS), Mixture of Contexts (MoC), Adversarial Softmax (AdvSoft), and Spectrum Control (SC). For LMS-PLIF and SC, as the authors had not released their code publicly, we implemented them based on the details provided in their respective papers and had also double checked our implementation with the authors by requesting their code. We are thankful to Wang et al. [82] and Ganea et al. [22] for sharing their code with us. We would soon be converting *lmkit* from being a private repository to a public repository on GitHub. Once it is made public, it should be available under the repositories of yottabytt.