Alleviating the Low-Rank Softmax Bottleneck

Bartosz Dzionek

4th Year Project Report
Artificial Intelligence
School of Informatics
University of Edinburgh

2023

Abstract

The softmax function is the default approach to output a probability distribution in multiclass classification problems. However, recent literature on language modeling points out the shortcomings of softmax when the number of classes is large. The problem, known as the low-rank softmax bottleneck, limits the expressive capabilities of language models. In the thesis, we describe the intuition to understand the bottleneck. We provide a survey of the solutions proposed in the literature that aim to solve it. We generalize the bottleneck to any network where the last hidden layer is lower than the output. Then, for the first time, we experimentally evaluate solutions to the bottleneck in image classification. Based on experiments on handwritten digit recognition and species classification, we find that the *Mixture of Softmaxes* and *Mixture of Sigsoftmaxes* can lead to higher accuracy than softmax for very low ranks.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Bartosz Dzionek*)

Acknowledgements

Nothing in life is to be feared, it is only to be understood.

– Maria Skłodowska-Curie

First and foremost, I would like to express my deepest appreciation to my supervisor, Dr. Adam Lopez, for his invaluable guidance and support throughout this project. His insightful feedback and encouragement have been instrumental in shaping this first step of my research journey. It was great to learn from your academic and industrial experience.

I would also like to extend my gratitude to Andreas Grivas for his help and endless discussions. Andreas' expertise and willingness to lend a hand have been indispensable in overcoming many obstacles during this project. Good luck with your Ph.D.!

Additionally, I would like to thank my family for their unconditional love and support throughout my academic pursuits. Even if they mean I must leave you for most of the year. I would like to offer a special acknowledgment to my girlfriend Mariola, whose constant encouragement led me through the most arduous moments.

Finally, I want to express my gratitude to my past mathematics teachers, Elżbieta, Anna, and Teresa. They played a vital role in influencing my interests. Without their early mentoring and inspiration, I would not be where I am today.

Table of Contents

Notation

Symbol	Typical meaning
K	The number of classes
d	The dimension of the last hidden layer
M	The number of mixtures in MoS/MoSS
a, A	Regular (not bold) letters are scalars
$\mathbf{a}$, $\mathbf{v}$	Lowercase bold letters are vectors
$\mathbf{A}$, $\mathbf{B}$	Uppercase bold letters are matrices
$:=$	Is defined to
$\mathbf{AB}$	A matrix multiplication
$\mathbf{A}^\top$	A transpose of a matrix
$\mathbf{a}^\top\mathbf{b}$	A dot product
$\mathbf{a}_i$	An i-th element of a vector
$\mathbf{A}_{ij}$	An (i, j)-th entry of a matrix
$\mathbf{1}_N$	A vector of all ones of size N
$1[a = b]$	An indicator function: 1 if $a = b$, else 0
$\mathrm{img}, \mathrm{dim}$	Image of a function, dimension of a vector space
$\mathrm{P}(\cdot)$	A probability of an outcome
$\mathrm{P}(\cdot \mid \cdot)$	A conditional probability
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	A vector of real numbers of size n
$\mathbb{R}^{a\times b}$	An $a \times b$ dimensional matrix of real numbers
$\sum_{a=1}^{A}$	The sum of all elements for $a = 1, \ldots, A$
$\prod_{a=1}^{A}$	The product of all elements for $a = 1, \ldots, A$
$a \in A$	a is a member of set A
$A \subseteq B$	A is a subset of B
$\{a \mid p\}$	The set with all elements a such that p is true.
$\{a_n\}_{n=1}^{N}$	The set $\{a_1, \ldots, a_N\}$
$\forall x$	The universal quantifier meaning “for all x”

1

Introduction

1.1 Motivation

Modeling probabilistic distributions is a big part of machine learning. A common task is a multiclass classification, where an algorithm models the probability that an input example belongs to one of a set of mutually exclusive classes (Bishop, 2006). In deep learning, the input representation is learned using a neural network. However, the values of the last layer of the network are not guaranteed to satisfy the probability axioms. Hence, we need a transformation that can map the output to a valid probability distribution. The standard transformation to achieve this is softmax (Goodfellow et al., 2016).

Nevertheless, recent literature points out the shortcomings of softmax when the number of classes is large. An example of this problem is neural language modeling where a model outputs a probability distribution over a big vocabulary. Yang et al. (2018) found that in the majority of neural language models, softmax imposes a low-rank output representation that does not have enough capacity to model the complexity of a natural language. They also provided a simple solution to mitigate the problem that led to state-of-the-art results. Their paper motivated a series of follow-up papers trying to improve the expressivity along with the time and memory efficiency of the baseline solution.

On the other hand, it is not clear how much of the theory of low-rank softmax matters for practical applications. In particular, state-of-the-art models such as GPT-3 use the low-rank softmax, albeit their dimensionality has increased dramatically (Brown et al., 2020). Very often we do not need the whole distribution, but just the most likely class (referred to as argmax). An example task is machine translation in which argmax corresponds to the best translation. Grivas et al. (2022) analyzed the bottleneck from the perspective of argmax in machine translation. They argued the bottleneck affects only very infrequent tokens and is unlikely to impact the quality of practical machine translation models. Also in the paper Parthiban et al. (2021), the authors found that while the low-rank matrix is a theoretical problem, it has little impact on the performance of a model. In their empirical study, higher rank was very weakly correlated with a better perplexity of a language model.

Since the problem of a low-rank approximation induced by softmax is a novel research topic, there is a need for a review paper that would compare various solutions. In the opinion of the author, it is not clear from the literature to what extent the proposed solutions solve the bottleneck. It is also not agreed on how much of a practical problem the softmax bottleneck is. Moreover, the bottleneck was only explored in the case of language modeling. Can we generalize the bottleneck to other tasks such as image classification? Would the methods from the literature alleviate the bottleneck similarly well in these other tasks? Throughout the thesis, we aim to answer these questions.

1.2 Main contributions

The thesis addresses the problem of the low-rank softmax bottleneck. It can be regarded as a survey paper on existing solutions from the literature. The author provides the necessary background to understand the bottleneck. The methods proposed in the literature are explained using consistent mathematical notation (which is not necessarily the case in their original papers). To facilitate the discussion of the different models, we include intuitive explanations and examples. We also release all the code used in the thesis[1].

The other contributions include:

- The author generalizes the softmax bottleneck outside its initial language modeling domain into any multiclass classification problem where the number of neurons in the last hidden layer is less than the output.
- Following this general formulation, four different models from the literature are tested in image classification tasks of handwritten digit recognition and species classification. We implement and share the implementation of three models from the literature.
- Based on experiments, we demonstrate Mixture of Softmaxes and Mixture of Sigsoftmaxes can give higher accuracy in image classification in very low-rank networks.

1.3 Structure

- **Chapter 1 - Introduction:** In this chapter, we contain the motivation and goals for the thesis. We also state the main contributions.
- **Chapter 2 - Background:** We formally explain what multiclass classification is with language modeling as its example. We describe the problem of the low-rank softmax bottleneck and why simple solutions do not solve it.
- **Chapter 3 - Towards breaking the bottleneck:** The chapter contains a literature review with our theoretical findings from original papers. We also present intuition and examples to help the reader understand the literature in a broader context.

[1] https://github.com/dzionek/softmax-bottleneck

- **Chapter 4 - Experiments:** The chapter comprises our experiments on images with their findings. We share details about our implementation. Then, we experimentally evaluate models from the literature in the task of handwritten digit recognition and species classification.
- **Chapter 5 - Conclusions:** We summarize the thesis, its findings, and its limitations. We also provide recommendations for future work.
- **Appendix A - Proofs:** This appendix contains detailed mathematical proofs for the keen reader.
- **Appendix B - Additional results from experiments:** The appendix contains the detailed results we got in the experiments. Chapter 4 discusses a subset of them.

2

Background

2.1 Multiclass classification

The goal of a multiclass classification is to take an input $\mathbf{x}$ and assign it into one of K disjoint classes $\mathcal{C}_k$ where $k = 1, \ldots, K$ (Bishop, 2006). As a simple example, let us imagine a problem of assigning an article title into categories: sport $\mathcal{C}_1$, technology $\mathcal{C}_2$, or business $\mathcal{C}_3$. We would like our ideal classifier to assign the article with a title *"Lewandowski sets a goal record"* to $\mathcal{C}_1$, *"What is the best operating system?"* to $\mathcal{C}_2$, and *"The most promising stocks"* to $\mathcal{C}_3$.

We can also see this problem probabilistically as estimating the conditional distribution $\mathrm{P}(\boldsymbol{\mathcal{C}} \mid \boldsymbol{\phi}) = [\mathrm{P}(\mathcal{C}_1 \mid \boldsymbol{\phi}), \ldots, \mathrm{P}(\mathcal{C}_K \mid \boldsymbol{\phi})]^\mathsf{T}$ where the vector $\boldsymbol{\phi}$ is any transformation of the input. For example, if $\boldsymbol{\phi}$ is a transformation of the article title *"The most valuable football clubs"*, we might estimate the distribution as $\hat{\mathrm{P}}(\mathcal{C}_1 \mid \boldsymbol{\phi}) = 0.7$, $\hat{\mathrm{P}}(\mathcal{C}_2 \mid \boldsymbol{\phi}) = 0.01$, and $\hat{\mathrm{P}}(\mathcal{C}_3 \mid \boldsymbol{\phi}) = 0.29$.

The transformation $\boldsymbol{\phi}$ and the probability $\hat{\mathrm{P}}(\boldsymbol{\mathcal{C}} \mid \boldsymbol{\phi})$ can be learned using a neural network. A (feed-forward) neural network takes the input $\boldsymbol{\phi}$ and transforms it through a series of H hidden layers:

$$\mathbf{h}^{(1)} = g\left(\mathbf{W}^{(1)}\boldsymbol{\phi}\right), \quad \mathbf{h}^{(2)} = g\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)}\right), \quad \ldots \quad , \quad \mathbf{h}^{(H)} = g\left(\mathbf{W}^{(H)}\mathbf{h}^{(H-1)}\right),$$

where g is a non-linear function called an *activation function* (Goodfellow et al., 2016). An example g is a sigmoid activation that can be applied element-wise: $\sigma(x) = \frac{1}{1+\exp(-x)}$.

The network's input vector $\boldsymbol{\phi}$ is called an *embedding* and can be obtained from an article title using simple methods like one-hot encoding (Khoshgoftaar, 2020) or using learned embeddings such as Skip-Gram (Mikolov et al., 2013) and GloVe (Pennington et al., 2014).

For short, we denote the vector of the last hidden layer $\mathbf{h}^{(H)}$ as $\mathbf{h}$. We also assume $\mathbf{h} \in \mathbb{R}^d$. Then, the output layer is an affine transformation of $\mathbf{h}$,

$$\mathbf{z} = \mathbf{W}\mathbf{h},$$

such that $\mathbf{W} \in \mathbb{R}^{K \times d}$ and $\mathbf{z} \in \mathbb{R}^K$. In this output, the vector element z_i corresponds to the class C_i.

Definition 2.1 (Kolmogorov (1950))**.** Let Ω be a sample space consisting of mutually exclusive events $X_1, \ldots, X_N$. A probability measure P must satisfy:

1. $\mathrm{P}(X_i) \geq 0$ for all $i = 1, \ldots, N$.
2. $\mathrm{P}(\Omega) = \sum_{i=1}^{N} \mathrm{P}(X_i) = 1$.

In the multiclass classification, we have $\Omega = \{C_1, \ldots, C_K\}$. However, we cannot assume $\mathbf{z}$ to be a probability measure $\mathrm{P}(\cdot \mid \boldsymbol{\phi})$. The elements of $\mathbf{z}$ are not guaranteed to satisfy any of the two axioms in definition 2.1.

2.2 Softmax

To consider the set of all valid probability distributions for multiclass classification, it is worth recalling the definition of a simplex.

Definition 2.2 (Boyd and Vandenberghe (2004))**.** The $(K-1)$-dimensional probability simplex is given by

$$\Delta^{K-1} := \left\{ \mathbf{v} \in \mathbb{R}^K \mid \forall_{k=1}^{K} v_k \geq 0, \sum_{k=1}^{K} v_k = 1 \right\}.$$

Example 2.3. Every categorical distribution over K categories belongs to the $(K-1)$-dimensional probability simplex. For instance, $\mathrm{P}(\boldsymbol{C}) = [0.2, 0.3, 0.5]^{\mathsf{T}} \in \Delta^2$. We can also see $\mathrm{P}(\boldsymbol{C})$ as a point on the polytope Δ^2 as presented in figure 2.1.

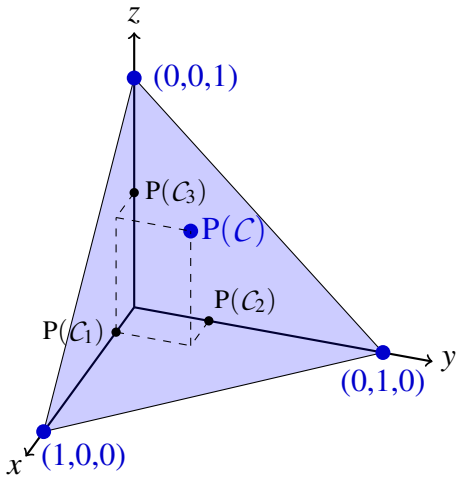


Figure 2.1: Distribution $\mathrm{P}(\boldsymbol{C}) = (0.2, 0.3, 0.5)$ and Δ^2.

Comparing the two conditions of the set in definition 2.2 with the axioms in definition 2.1, we see that $(K-1)$-dimensional probability simplex is a set of all valid probability distributions with K classes. Thus, if we can find a function $f : \mathbb{R}^K \to \Delta^{K-1}$, then $f(\mathbf{z})$ would be a valid probability distribution over K classes.

The most common choice of a function f is softmax (Goodfellow et al., 2016).

Definition 2.4 (Bridle (1989))**.** Let $\mathbf{z} \in \mathbb{R}^K$. The softmax value at the k-th element is

$$\text{softmax}(\mathbf{z})_k := \frac{\exp(z_k)}{\sum_{k'=1}^{N} \exp(z_{k'})} \qquad \text{for all } k = 1, \ldots, K.$$

As presented in theorem A.1 in the appendix, $\text{softmax}(\mathbf{z}) \in \Delta^{K-1}$. A detailed discussion of softmax as a projection onto the probability simplex was described in Blondel (2019). Most importantly, we can use softmax to define the output of a neural network to be

$$\text{softmax}(\mathbf{z}) = \text{softmax}\left(\mathbf{W}\mathbf{h}\right) = \hat{\mathrm{P}}(\mathcal{C} \,|\, \boldsymbol{\phi}).$$

The elements of the vector $\mathbf{W}\mathbf{h}$ are called *logits* (Goodfellow et al., 2016).

Example 2.5. If $\mathbf{W}\mathbf{h} = [-2.15, 1, 0]^\mathsf{T}$, softmax gives $\hat{\mathrm{P}}(\mathcal{C} \,|\, \boldsymbol{\phi}) \approx [0.03, 0.71, 0.26]^\mathsf{T}$.

We call our output layer of a neural network a softmax layer. It involves first computing the logits, and then passing them to the softmax function. In the next sections, we present why the softmax layer might not be the ideal choice when the number of classes is large.

Overall, to train the network, we can consider a training set $\mathcal{D} = \left\{ \left(\boldsymbol{\phi}^{(i)}, t^{(i)} \right) \right\}_{i=1}^{I}$, where $t^{(i)}$ is the one-hot-encoded class for the i-th example.

Example 2.6. Re-using classes from the first paragraph of this chapter, we can have

$$\begin{aligned}\mathcal{D} = \{&(\boldsymbol{\phi}^{\text{“}sport\text{”}}, \left[1,0,0\right]^\mathsf{T}), (\boldsymbol{\phi}^{\text{“}computer\text{”}}, \left[0,1,0\right]^\mathsf{T}), (\boldsymbol{\phi}^{\text{“}debt\text{”}}, \left[0,0,1\right]^\mathsf{T}),\\ &(\boldsymbol{\phi}^{\text{“}money\text{”}}, \left[0,0,1\right]^\mathsf{T}), (\boldsymbol{\phi}^{\text{“}code\text{”}}, \left[0,1,0\right]^\mathsf{T}), (\boldsymbol{\phi}^{\text{“}game\text{”}}, \left[1,0,0\right]^\mathsf{T})\}.\end{aligned}$$

The trained weights $W^* = \langle \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(H)}, \mathbf{W} \rangle$ minimize negative log-likelihood

$$-\log \hat{\mathrm{P}}(\mathcal{D} \,|\, W^*) = -\log \prod_{i=1}^{I} \prod_{k=1}^{K} \hat{\mathrm{P}}(C_k \,|\, \boldsymbol{\phi}^{(i)})^{t_k^{(i)}} = -\sum_{i=1}^{I} \sum_{k=1}^{K} t_k^{(i)} \log \hat{\mathrm{P}}(C_k \,|\, \boldsymbol{\phi}^{(i)}). \qquad (2.1)$$

The objective of equation 2.1 is called *cross-entropy* and can be optimized using backpropagation and Stochastic Gradient Descent (Bishop, 2006).

2.3 Language modeling

An example of a task with a large number of classes is language modeling. In language modeling the set of classes is called a vocabulary $\mathcal{V}$. Usually, $|\mathcal{V}| \approx 10^5$ (Yang et al., 2018). The goal of language modeling is to assign a probability to a sequence of word tokens (Eisenstein, 2019):

$$\mathrm{P}(X_1, \ldots, X_T) \qquad \text{where each } X_t \in \mathcal{V}.$$

Example 2.7 (Eisenstein (2019))**.** Suppose we generate a Spanish sentence $X^{(s)}$ from a language model

$$\mathrm{P}_s\left(X^{(s)}\right) = \mathrm{P}_s\left(X_1^{(s)}, \ldots, X_T^{(s)}\right).$$

Each $X_t^{(s)}$ is a token from the Spanish vocabulary $\mathcal{V}^{(s)}$. For instance, let x^* be a sentence *"El cafe negro me gusta mucho"*. Suppose we consider all possible translations of x^* into some English sentence $X^{(e)}$. The distribution over possible translations is the conditional

$$\mathrm{P}_{e\,|\,s}\left(X^{(e)}\,|\,x^*\right).$$

In this formulation, we can expect

$$\mathrm{P}_{e\,|\,s}\left(\textit{The coffee black me pleases much}\,|\,x^*\right) < \mathrm{P}_{e\,|\,s}\left(\textit{I love dark coffee}\,|\,x^*\right),$$

meaning *"I love dark coffee"* is a better English translation for x^*.

By the chain rule of probability, the probability of the whole sequence factorizes into

$$\mathrm{P}(X) = \prod_{t=1}^{T} \mathrm{P}(X_t\,|\,X_1, \ldots, X_{t-1}) = \prod_{t=1}^{T} \mathrm{P}(X_t\,|\,C_t),$$

where $C_t = X_1, \ldots, X_{t-1}$ is called a *context*. Intuitively, a context is the sequence preceding the current token.

Example 2.8. When considering a probability of a sentence, it is common to pad the sentence with the start token □ and the end token ■ (Eisenstein, 2019). These special tokens are one way that guarantees probabilities of sentences of all lengths sum to one. Then, the chain rule of probability implies

$$\mathrm{P}(\textit{A dog barks}) = \mathrm{P}(\textit{A}\,|\,\square)\,\mathrm{P}(\textit{dog}\,|\,\textit{A})\,\mathrm{P}(\textit{barks}\,|\,\textit{A dog})\,\mathrm{P}(\blacksquare\,|\,\textit{A dog barks}).$$

We can interpret $\mathrm{P}(X\,|\,C)$ as a multiclass classification with K possible values for X and a context C being the input transformation. The standard approach in language modeling research is to model this probability using softmax (Eisenstein, 2019).

2.4 Formulation of the softmax bottleneck

In this section, we reformulate language modeling as a matrix factorization problem. To do this, we distinguish two distributions. The modeled distribution $\hat{\mathrm{P}}(X\,|\,C)$ is learned to estimate the actual distribution $\mathrm{P}(X\,|\,C)$. The modeled distribution uses softmax

$$\hat{\mathrm{P}}(X\,|\,C) = \begin{bmatrix} \hat{\mathrm{P}}(X_1\,|\,C) \\ \vdots \\ \hat{\mathrm{P}}(X_K\,|\,C) \end{bmatrix} = \mathrm{softmax}\left(\begin{bmatrix} \mathbf{w}_1^\top \mathbf{h} \\ \vdots \\ \mathbf{w}_K^\top \mathbf{h} \end{bmatrix}\right) = \mathrm{softmax}\left(\begin{bmatrix} \mathbf{h}^\top \mathbf{w}_1 \\ \vdots \\ \mathbf{h}^\top \mathbf{w}_K \end{bmatrix}\right).$$

While the set of contexts can be theoretically unbounded, when training the model we use a finite number of contexts. So, for the purpose of the analysis, let us consider

the variable C captures a fixed large number of contexts $C_1, \ldots, C_N$. Then, we notice $K \ll N$, and we get a matrix

$$\hat{\mathrm{P}}(X \mid C) = \begin{bmatrix} \hat{\mathrm{P}}(X \mid C_1) & \ldots & \hat{\mathrm{P}}(X \mid C_N) \end{bmatrix} = \begin{bmatrix} \hat{\mathrm{P}}(X_1 \mid C_1) & \ldots & \hat{\mathrm{P}}(X_1 \mid C_N) \\ \vdots & \ddots & \vdots \\ \hat{\mathrm{P}}(X_K \mid C_1) & \ldots & \hat{\mathrm{P}}(X_K \mid C_N) \end{bmatrix},$$

whose (k,n)-th entry is

$$\hat{\mathrm{P}}(X \mid C)_{kn} = \hat{\mathrm{P}}(X_k \mid C_n) = \frac{\exp\left(\mathbf{h}_n^\mathsf{T} \mathbf{w}_k\right)}{\sum_{k'=1}^{K} \exp\left(\mathbf{h}_n^\mathsf{T} \mathbf{w}_{k'}\right)} := \frac{\exp\left(\mathbf{h}_n^\mathsf{T} \mathbf{w}_k\right)}{z_n}.$$

Using element-wise logarithm, we define (similarly to Ganea et al. (2019)),

$$\hat{\mathbf{A}} := \log \hat{\mathrm{P}}(X \mid C) = \mathbf{W}\mathbf{H}^\mathsf{T} - \mathbf{1}_K \log \mathbf{z}^\mathsf{T}, \tag{2.2}$$

where $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\mathsf{T} \\ \vdots \\ \mathbf{w}_K^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{K \times d}$, $\mathbf{H} = \begin{bmatrix} \mathbf{h}_1^\mathsf{T} \\ \vdots \\ \mathbf{h}_N^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{N \times d}$, $\log \mathbf{z} = \begin{bmatrix} \log z_1 \\ \vdots \\ \log z_N \end{bmatrix} \in \mathbb{R}^N$.

Yang et al. (2018) state the problem of expressivity as finding suitable parameters for $\hat{\mathbf{A}}$ that make it equal to the matrix of the actual distribution $\mathbf{A} := \log \mathrm{P}(X \mid C)$. However, we note that $\operatorname{rank}(\mathbf{W}\mathbf{H}^\mathsf{T}) \leq d$ and $\operatorname{rank}(\mathbf{1}_K \log \mathbf{z}^\mathsf{T}) = 1$. Combining with the rank-of-sum inequality (theorem A.2 in appendix), yields $\operatorname{rank}(\hat{\mathbf{A}}) \leq d + 1$. Figure 2.2 provides a visual understanding of the bounded rank.

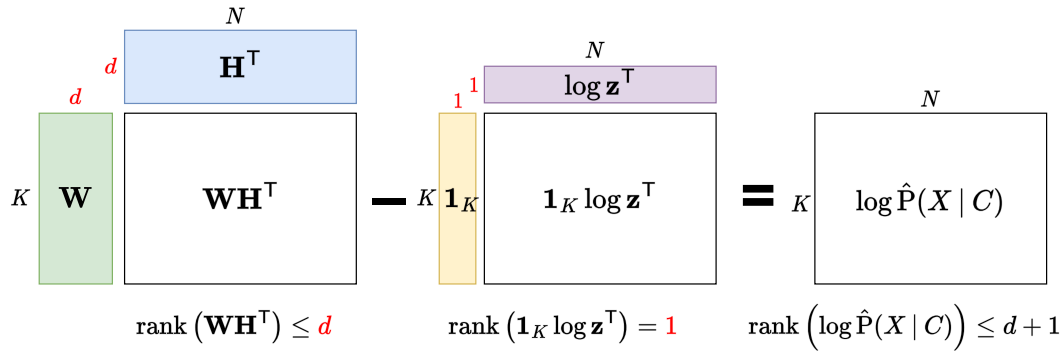


Figure 2.2: Visual representation of the equation 2.2. Consider the case $d < K, N$; we observe that d becomes the bottleneck for rank.

However, Yang et al. (2018) hypothesize the matrix of actual distributions $\mathbf{A}$ is high-rank (possibly of full rank K) due to the high context-dependency of natural languages. Since

$$d < \operatorname{rank}(\mathbf{A}) - 1 \approx K - 1 \qquad \text{implies} \qquad \operatorname{rank}(\hat{\mathbf{A}}) < \operatorname{rank}(\mathbf{A}),$$

for typical values $d \approx 10^2$ and $K \approx 10^5$, the softmax layer is a bottleneck in expressivity as $\hat{\mathbf{A}} \neq \mathbf{A}$ due to different ranks.

2.5 Naive solutions

Yang et al. (2018) mention two simple ideas that might seem to fix the problem: n-gram modeling and using sufficiently large embedding dimension d. Both of them inherently lead to overfitting according to their paper.

2.5.1 N-gram models

Definition 2.9 (Jurafsky and Martin (2008))**.** N-gram assumption is a Markov assumption stating that the sequence of tokens depends only on the previous $(n-1)$ tokens.

$$\mathrm{P}(X_t \mid X_{<t}) \approx \mathrm{P}(X_t \mid X_{t-1}, \ldots, X_{t-n+1}) \quad \forall t \text{ with appropriate padding}$$

Example 2.10. In a unigram model ($N = 1$):

$$\mathrm{P}(A\ dog\ barks) = \mathrm{P}(A)\,\mathrm{P}(dog)\,\mathrm{P}(barks),$$

whereas in a bigram model ($N = 2$):

$$\mathrm{P}(A\ dog\ barks) = \mathrm{P}(A \mid \square)\,\mathrm{P}(dog \mid A)\,\mathrm{P}(barks \mid dog)\,\mathrm{P}(\blacksquare \mid barks).$$

The advantage of N-gram is that it can be easily trained through the maximum likelihood estimation (Jurafsky and Martin, 2008):

$$\hat{\mathrm{P}}(X_t \mid X_{t-1}, \ldots, X_{t-n+1}) = \frac{\mathrm{count}(X_{t-n+1}, \ldots, X_{t-1}, X_t)}{\sum_{X \in \mathcal{V}} \mathrm{count}(X_{t-n+1}, \ldots, X_{t-1}, X)} = \frac{\mathrm{count}(X_{t-n+1}, \ldots, X_{t-1}, X_t)}{\mathrm{count}(X_{t-n+1}, \ldots, X_{t-1})}$$

Example 2.11. Suppose we have a dataset with sentences: "A dog barks", "A cat meows", "A dog sleeps". Then,

$$\hat{\mathrm{P}}(dog \mid A) = \frac{\mathrm{count}(A\ dog)}{\mathrm{count}(A)} = \frac{2}{3}, \quad \hat{\mathrm{P}}(cat \mid A) = \frac{1}{3}, \quad \hat{\mathrm{P}}(barks \mid A\ dog) = \frac{1}{2}.$$

To make sure an n-gram model can generalize to unseen data, other extensions can be added such as Laplace smoothing (Jurafsky and Martin, 2008) or back-off (Kneser and Ney, 1995). If we compute each entry of $\hat{\mathbf{A}}$ independently in this way, the matrix will be high-rank. However, we will need $K \times N$ parameters. With that many parameters and unbound N, the model is very likely to overfit (Yang et al., 2018).

2.5.2 Large embedding size

We could instead just use the previous neural model with embeddings of size $d := K$. In this case, $\mathbf{W}$ would be $K \times K$, which also leads to overfitting. It was empirically found that increasing d beyond hundreds does not improve language models (Merity et al., 2018; Melis et al., 2018; Krause et al., 2018; Yang et al., 2018). Moreover, we note that high d can be computationally infeasible.

Based on the analysis of the two simple ideas, Yang et al. (2018) claim there is a trade-off between the generalization of a model and its expressivity. They highlight the need for a model that is more expressive than the softmax neural network and does not lead to an explosion in the number of parameters.

3

Towards breaking the bottleneck

In this chapter, we critically evaluate the solutions to the softmax bottleneck proposed by the literature. We contrast the approaches with the linear softmax discussed in the previous chapter.

3.1 Mixture of Softmaxes

We start the discussion with Mixture of Softmaxes (MoS) which was the first recommendation to break the bottleneck Yang et al. (2018). The idea is to use a convex combination of M softmax activations with different context embeddings.

Definition 3.1 (Yang et al. (2018))**.** Mixture of Softmaxes (MoS) parametrizes the model as

$$\hat{\mathbf{A}}_{kn}^{(\text{MoS})} := \log \hat{\mathrm{P}}(x_k \mid c_n) := \log \sum_{m=1}^{M} \pi_{n,m} \frac{\exp\left(\mathbf{h}_{n,m}^{\mathsf{T}} \mathbf{w}_k\right)}{\sum_{k'=1}^{K} \exp\left(\mathbf{h}_{n,m}^{\mathsf{T}} \mathbf{w}_{k'}\right)} \qquad \text{s.t.} \sum_{m=1}^{M} \pi_{n,m} = 1$$

where $\pi_{n,m}$ are called *priors* (or *mixture weights*) of the m-th softmax with regards to the n-th context. The visual comparison between the MoS network and linear softmax is shown in figure 3.1. The only difference is that we have weights π_m and the final probability is a weighted average of M different logits.

Example 3.2. If $M = 1$, we have $\pi_n = 1$ for all contexts $n = 1, \ldots, N$. Hence, the MoS reduces to the linear softmax. By having $M > 1$, we will show that the MoS can theoretically attain a higher rank than softmax.

The priors can be modeled using a recurrent neural network (RNN) (Rumelhart et al., 1986). In particular, Yang et al. (2018) in their paper use the state-of-the-art architecture at that time called AWD-LSTM (Merity et al., 2018). In general, an RNN is composed of T hidden states $(\mathbf{g}_1, \ldots, \mathbf{g}_T)$. Each context is a transformation of a hidden state $\mathbf{h}_{n_t,m} = \tanh\left(\mathbf{W}_{h,m}\mathbf{g}_t\right)$. To make sure the priors sum to one, the RNN normalizes them using softmax. Thus, the prior at time t takes the form

$$\pi_{n_t,m} = \frac{\exp\left(\mathbf{w}_{\pi,m}^{\mathsf{T}}\mathbf{g}_t\right)}{\sum_{m'=1}^{M} \exp\left(\mathbf{w}_{\pi,m'}^{\mathsf{T}}\mathbf{g}_t\right)},$$

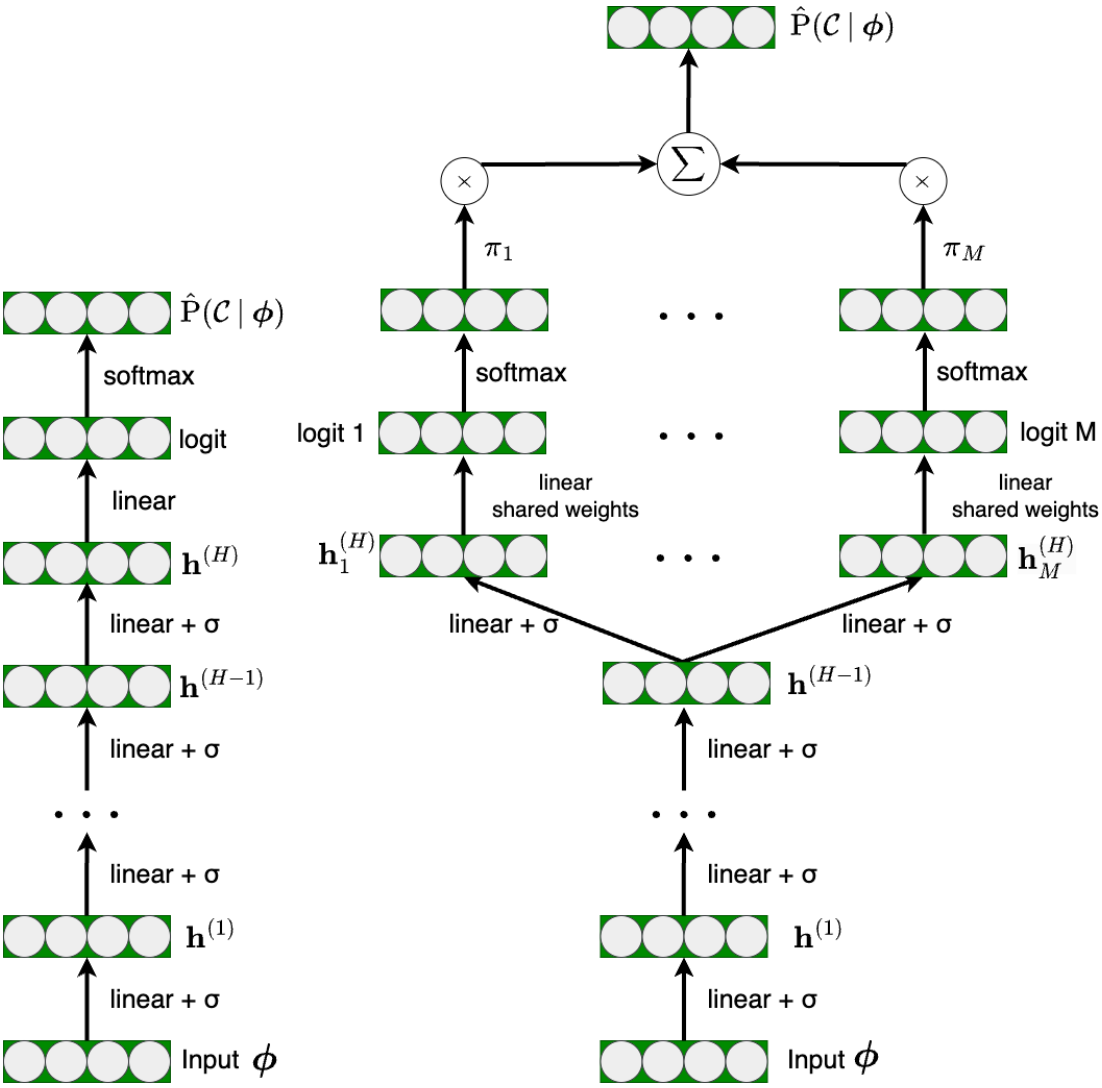


Figure 3.1: The linear softmax network (left) and the MoS network (right). Contexts were omitted for simplicity; σ can be any non-linearity; products and sums are element-wise. All linear layers from $h_m^{(H)}$ to logit m share the same weights.

where $\mathbf{W}_{h,m}$ and $\mathbf{w}_{\pi,m}$ are parameters of the model.

Compared to the linear softmax, we do not have the previous matrix factorization (equation 2.2). In fact, the log probability matrix is

$$\hat{\mathbf{A}}^{(\mathrm{MoS})} = \log \sum_{m=1}^{M} \exp\left(\mathbf{W}\mathbf{H}_m^{\mathsf{T}} - \mathbf{1}_K \log \mathbf{z}^{\mathsf{T}}\right) \mathbf{\Pi}_m,$$

where $\mathbf{\Pi}_m$ is a $N \times N$ diagonal matrix with entries $\pi_{n,m}$. In other words, the matrix $\mathbf{\Pi}_m$ contains the mixture coefficients for a given mixture component across N timesteps. Yang et al. (2018) argue that due to the non-linear log-sum-exp transformation, the matrix can be arbitrarily high rank. Furthermore, they claim the model would generalize well (unlike the naive solutions discussed in section 2.5). The model does not need to

have many parameters, as we can decrease dimension d to compensate for additional mixture parameters.

In opposition to the optimism of Yang et al. (2018), the author notes the $\hat{\mathbf{A}}^{(\text{MoS})}$ matrix might not necessarily be high-rank. Yang et al. (2018) did not provide any theoretical guarantees for the rank, except the intuition with non-linear structure. Let us think about the corner case when all context embeddings are equal, i.e. $\mathbf{H}_m = \mathbf{H}$ for all m. Then, it holds that

$$\begin{aligned}
\hat{\mathbf{A}}^{(\text{MoS})} &= \log \sum_{m=1}^{M} \exp\left(\mathbf{W}\mathbf{H}^\top - \mathbf{1}_K \log \mathbf{z}^\top\right) \boldsymbol{\Pi}_m \\
&= \log \exp\left(\mathbf{W}\mathbf{H}^\top - \mathbf{1}_K \log \mathbf{z}^\top\right) \sum_{m=1}^{M} \boldsymbol{\Pi}_m \\
&= \log \exp\left(\mathbf{W}\mathbf{H}^\top - \mathbf{1}_K \log \mathbf{z}^\top\right) \mathbb{I}_N \\
&= \mathbf{W}\mathbf{H}^\top - \mathbf{1}_K \log \mathbf{z}^\top.
\end{aligned}$$

In this case, we have the same factorization as in equation 2.2, so $\text{rank}\left(\hat{\mathbf{A}}^{(\text{MoS})}\right) \leq d+1$.

On the other hand, when $\mathbf{H}_m$ are not the same, the $\hat{\mathbf{A}}^{(\text{MoS})}$ matrix can possibly be high rank. This is because entries of $\hat{\mathbf{A}}^{(\text{MoS})}$ are computed by a non-linear element-wise function. Even though the rank of $\mathbf{W}\mathbf{H}_m^\top - \mathbf{1}_K \log \mathbf{z}^\top$ is bounded by $d+1$, its entries are transformed by a non-linear log-sum-exp function. To better understand why a non-linear element-wise function can make the rank higher, let us consider two examples.

Example 3.3. (A non-linear element-wise function can increase rank). Let us consider

$$\mathbf{A} := \begin{bmatrix} 1 & 2 & 8 \\ 2 & 4 & 16 \\ 4 & 8 & 32 \end{bmatrix}, \quad \log_2 \mathbf{A} = \begin{bmatrix} 0 & 1 & 3 \\ 1 & 2 & 4 \\ 2 & 3 & 5 \end{bmatrix}, \quad \mathbf{B} := \begin{bmatrix} 1 & 4 & 1 \\ 2 & 2 & 1 \\ 3 & 0 & 1 \end{bmatrix}, \quad 2^{\mathbf{B}} = \begin{bmatrix} 2 & 16 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 2 \end{bmatrix}.$$

We note $\text{rank}(\mathbf{A}) = 1$ as the columns of $\mathbf{A}$ are multiples of each other. However, $\text{rank}\left(\log_2 \mathbf{A}\right) = 2$ because two times the second row minus the third row gives the first row. Also, while $\text{rank}(\mathbf{B}) = 2$ due to the middle row being an average of the other rows, we have $\text{rank}\left(2^{\mathbf{B}}\right) = 3$.

Example 3.4. (Log-sum-exp is a non-linear element-wise function). Let us define

$$\text{LSE}(x, y) := \log(\exp(x) + \exp(y)).$$

By definition, the LSE function is an element-wise function. The plots in figures 3.2 & 3.3 illustrate the log-sum-exp function is non-linear. In contrast, for a linear function, the contour plot would present parallel lines.

We need to note that, in practice, there are no guarantees the MoS training leads to distinct column/row spaces whose sum could span all K dimensions. This is evident in the example 3.3, where taking log did not provide a full-rank matrix. Furthermore, the cross-entropy objective used in multiclass classification might not cause the rank to be maximized.

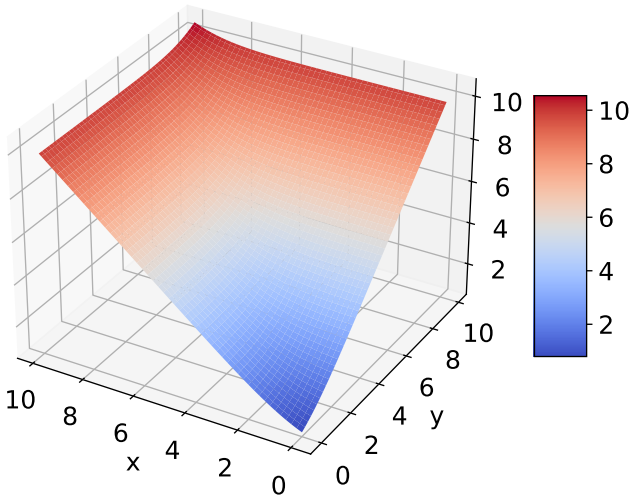


Figure 3.2: 3D plot of LSE(x, y)

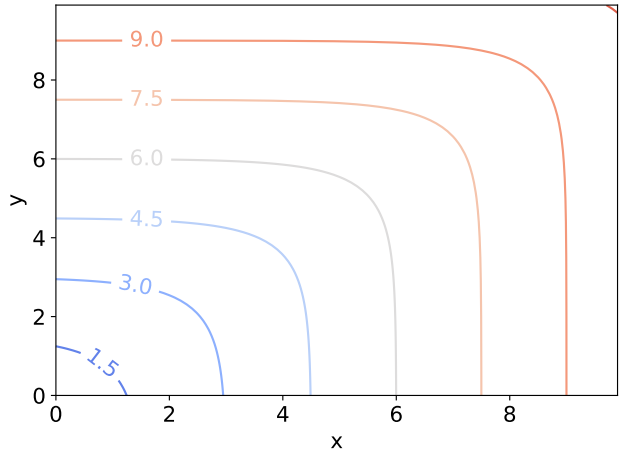


Figure 3.3: Contour plot of LSE(x, y)

3.1.1 Mixture of Contexts

Definition 3.5 (Yang et al. (2018))**.** As a naive alternative to the MoS, one could also consider Mixture of Contexts (MoC),

$$\hat{\mathbf{A}}_{kn}^{(\text{MoC})} := \log \frac{\exp\left(\left(\sum_{m=1}^{M} \pi_{n,m}\mathbf{h}_{n,m}\right)^{\mathsf{T}} \mathbf{w}_k\right)}{\sum_{k'=1}^{K} \exp\left(\left(\sum_{m=1}^{M} \pi_{n,m}\mathbf{h}_{n,m}\right)^{\mathsf{T}} \mathbf{w}_{k'}\right)} \qquad \text{s.t.} \sum_{m=1}^{M} \pi_{n,m} = 1.$$

However, by setting $\mathbf{h}'_n := \sum_{m=1}^{M} \pi_{n,m}\mathbf{h}_{n,m}$, Yang et al. (2018) observe that $\hat{\mathbf{A}}_{MoC}$ could be factorized as in equation 2.2, leading again to the softmax bottleneck.

3.1.2 Mixtape

Despite the ability to break the bottleneck, the MoS model can be much more expensive in terms of memory and time requirements. Yang et al. (2019) iterate on the mixture of contexts idea in the Mixtape model. The authors argue that using element-wise matrix multiplication would make the matrix high-rank. However, the evidence is mostly empirical. According to their results, Mixtape is much more efficient at the cost of slightly lower performance.

We note that claims about MoS being prohibitively expensive might be exaggerated. The authors were able to train both Mixtape and the better-performing but slower MoS. We also observe MoS can be always scaled down by taking a lower number of mixtures M. Therefore, Mixtape might only be applicable when time and memory are very important constraints and picking small M is impossible. We will not assume such constraints in this paper and thus we will not evaluate Mixtape in our experiments.

3.2 Sigsoftmax

Kanai et al. (2018) take a different approach than the Mixture of Softmaxes. The authors note that the MoS is *de facto* an additional layer or a mixing technique instead of a different output activation function. The MoS also requires tuning the additional hyperparameter M (number of mixtures) and adding additional weights $\mathbf{W}_{h,m}$ and $\mathbf{w}_{\pi,m}$.

3.2.1 Necessary conditions

To remediate the aforementioned problems, Kanai et al. (2018) go back to the properties that must hold for a valid output function $f : \mathbb{R}^K \to \Delta^{K-1}$. They state the function must be of the normalizing form

$$f(\mathbf{z})_k = \frac{g(\mathbf{z})_k}{\sum_{k'=1}^{K} g(\mathbf{z})_{k'}},$$

and satisfy conditions:

- **Non-linearity of $\log g(\mathbf{z})$:** If $\log g(\mathbf{z})$ is a linear function of $\mathbf{z}$, we would have a factorization similar to equation 2.2, leading to the rank bottleneck.
- **Numerical stability:** Training with gradient-based methods requires computing the gradient of log probabilities

  $$\frac{\partial \log f(\mathbf{z})_i}{\partial z_j} = \frac{1}{f(\mathbf{z})_i} \frac{\partial f(\mathbf{z})_i}{\partial z_j}.$$

  One needs to make sure $f(\mathbf{z})_i \neq 0$ throughout training to avoid division by zero.
- **Non-negative:** Recall that Δ^{K-1} is a subset of the non-negative orthant (definition 2.2). This requires $g(\mathbf{z})_i \geq 0$ for all $i = 1, \ldots, K$. The condition also follows directly from definition 2.1 condition 1.

- **Monotonically increasing:** f should be a smoothed argmax of $\mathbf{z}$. The higher the z_i value, the higher probability f should output. Given a constant denominator, it necessitates g to also be monotonically increasing.

The authors note that softmax does not satisfy all of these constraints.

Example 3.6. For softmax,

- log softmax is numerically stable:

$$\frac{\partial \log \operatorname{softmax}(\mathbf{z})_i}{\partial z_j} = 1[i=j] - \operatorname{softmax}(\mathbf{z})_j.$$

  The activation does not involve division (by zero).

- g is non-negative: $\exp(\mathbf{z})_i > 0$.
- g is monotonically increasing: $\forall z_i \geq z_j$, $\exp(z_i) \geq \exp(z_j)$.
- $\log g$ is linear: $\log \exp \mathbf{z} = \mathbf{z}$. Thus, softmax suffers from the rank bottleneck.

3.2.2 Sigsoftmax and its mixture

The solution for the softmax bottleneck proposed by their paper is replacing exp with a product of exp and sigmoid.

Definition 3.7 (Kanai et al. (2018))**.** Sigsoftmax uses $g(z) = \exp(z)\sigma(z)$ where σ is a sigmoid function,

$$\hat{\mathbf{A}}^{(\mathrm{SS})}_{kn} := \log \frac{\exp\left(\mathbf{h}_n^\top \mathbf{w}_k\right)\sigma\left(\mathbf{h}_n^\top \mathbf{w}_k\right)}{\sum_{k'=1}^{K} \exp\left(\mathbf{h}_n^\top \mathbf{w}_{k'}\right)\sigma\left(\mathbf{h}_n^\top \mathbf{w}_{k'}\right)}.$$

Unlike softmax, sigsoftmax satisfies all of the aforementioned constraints.

Theorem 3.8 (Kanai et al. (2018))**.** Sigsoftmax satisfies the necessary conditions from section 3.2.1.

Proof.

- non-linearity: $\log\left(\exp(\mathbf{z})\sigma(\mathbf{z})\right) = 2\mathbf{z} - \log(\mathbf{1} + \exp(\mathbf{z}))$.
- numerically stable: $\frac{\partial \log f(\mathbf{z})_i}{\partial z_j} = (1[i=j] - f(\mathbf{z})_j)(2 - \sigma(z_j))$.
- non-negative: both $\exp(z) \geq 0$ and $\sigma(z) \geq 0$.
- monotonically increasing: both exp and σ are monotonically increasing.

□

Again, we should accentuate these are necessary conditions, not sufficient conditions. The non-linear property of sigsoftmax can help to increase the rank but it is not guaranteed. To increase the chance of high rank we can consider taking a mixture of sigsoftmaxes as we did before in MoS.

Definition 3.9 (Kanai et al. (2018))**.** Similarly as before, one can define Mixture of Sigsoftmaxes (MoSS). Note that the priors can also be computed using sigsoftmax.

$$\hat{\mathbf{A}}^{(\mathrm{MoSS})}_{kn} := \log \sum_{m=1}^{M} \pi_{n,m} \frac{\exp\left(\mathbf{h}_{n,m}^{\mathsf{T}} \mathbf{w}_k\right) \sigma\left(\mathbf{h}_{n,m}^{\mathsf{T}} \mathbf{w}_k\right)}{\sum_{k'=1}^{K} \exp\left(\mathbf{h}_{n,m}^{\mathsf{T}} \mathbf{w}_{k'}\right) \sigma\left(\mathbf{h}_{n,m}^{\mathsf{T}} \mathbf{w}_{k'}\right)},$$
$$\pi_{n_t,m} := \frac{\exp\left(\mathbf{w}_{\pi,m}^{\mathsf{T}} \mathbf{g}_t\right) \sigma\left(\mathbf{w}_{\pi,m}^{\mathsf{T}} \mathbf{g}_t\right)}{\sum_{m'=1}^{M} \exp\left(\mathbf{w}_{\pi,m'}^{\mathsf{T}} \mathbf{g}_t\right) \sigma\left(\mathbf{w}_{\pi,m'}^{\mathsf{T}} \mathbf{g}_t\right)}.$$

We note that sigsoftmax has the same number of parameters as softmax, and the MoSS has the same as the MoS. However, both will require more time to train. The evidence for a possible improvement of MoSS over MoS is purely empirical.

3.3 Piecewise linear increasing function

Ganea et al. (2019) continue the search of a function g discussed by Kanai et al. (2018). The authors consider functions of the form $g(z) = \exp(\psi(z))$. To make sure ψ is expressive (can model any distribution), the authors add an additional constraint that ψ needs to be surjective on $\mathbb{R}$. Since $\log g(z) = \psi(z)$, ψ must be a non-linear function to avoid the low-rank bottleneck.

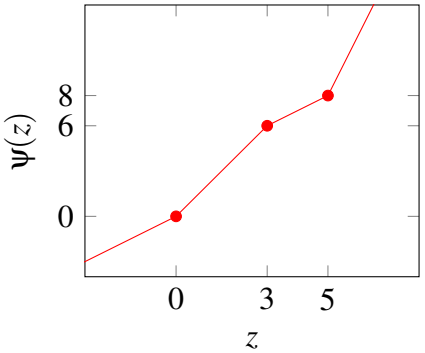
Example 3.10. A simple choice of a non-linear function could be sigmoid or rectified linear unit (ReLU). But the image of sigmoid is $(0,1)$ and of ReLU is $[0,\infty)$. Hence, they are not surjective on $\mathbb{R}$ and cannot be considered for ψ.

Additionally, to preserve the argmax property, one needs to ensure ψ is increasing. Instead of picking one specific $\psi(z)$, Ganea et al. (2019) decided to consider the whole family of continuous piecewise linear functions.

Definition 3.11 (Chua and Kang (1977))**.** A continuous piecewise linear function is a function whose graph is composed of line segments that consecutively intersect at their endpoints.

Example 3.12. We give an example of an increasing continuous piecewise linear function $\psi(z)$ comprising 4 line segments.

$$\psi(z) := \begin{cases} z & \text{if } z < 0, \\ 2z & \text{if } 0 \le z < 3, \\ z+3 & \text{if } 3 \le z < 5, \\ 4z-12 & \text{if } z \ge 5. \end{cases}$$



In the approach of Ganea et al. (2019), the segments are learned together with the model. A large enough interval $[-T,T]$ is split into $M+1$ equally-distanced knots

$$l_i := -T + \frac{2Ti}{M} \quad \text{for } i = 0,\dots,M.$$

Here, T and M are hyperparameters (e.g. $T = 20$, $M = 10^5$). The function becomes

$$\psi(x) := s_i x + b_i \quad \forall x \in [l_i, l_{i+1}],$$

with parameters s_i, b_i.

It is necessary to constrain the parameters s_i and b_i to make sure the function is increasing and continuous. Fortunately, we can map this constrained optimization problem into unconstrained optimization. First, to ensure ψ is increasing, the slope s_i must be positive. To attain this, the authors set $s_i := \log(1+\exp(v_i))$ (i.e. softplus function) and find first unconstrained v_i. The last part is ensuring the segments are intersecting at their ends. The condition holds when

$$b_i := b_0 + s_0 l_0 - l_i s_i + \frac{2T}{M} \sum_{j=0}^{i-1} s_j.$$

Thus, we can only learn the unconstrained bias term b_0 and then we are able to compute all b_i.

Definition 3.13 (Ganea et al. (2019))**.** For a learned ψ being a piecewise linear increasing function (PLIF),

$$\hat{\mathbf{A}}^{(\text{PLIF})}_{kn} := \log \frac{\exp\big(\psi(\mathbf{h}_n^\top \mathbf{w}_k)\big)}{\sum_{k'=1}^{K} \exp\big(\psi(\mathbf{h}_n^\top \mathbf{w}_{k'})\big)}.$$

The advantage of the PLIF model is the computational cost of training is only slightly higher than softmax. It allows using a high M (many different segments) leading to a highly non-linear function that means a more flexible model. Then, we can hope a highly non-linear model would lead to an increase in rank.

3.4 Generalization of the bottleneck

While all the approaches presented in this chapter aim to increase the rank of the matrix $\hat{\mathbf{A}}$, we can notice the main point why someone would use them is to increase performance (like perplexity for language models). We do not need to think about the rank of any matrix to check if they are able to improve a model. Therefore, we propose to generalize the problem of the low-rank softmax bottleneck to any network where $d < K$. Having a very small $d \ll K$ is very likely to negatively impact the performance as the very low-dimensional vector needs to be mapped into a high-dimensional vector. Having such a generalization, we are able to test solutions from this chapter outside language modeling.

4

Experiments

In the previous chapter, we discussed the various approaches from the literature to solve the problem of low-rank softmax bottleneck. In this chapter, the goal is to experimentally evaluate these models. In particular, for the first time, we will test the models outside the language-modeling domain. We aim to assess if the improvements claimed by language modeling researchers can transfer into better performance in image classification.

4.1 Implementation highlights

The experiments are conducted using PyTorch version 2.0 (Paszke et al., 2019). The hardware used is a cluster of NVIDIA GTX 1060 6GB GPUs. The implementation used in the experiments is made publicly available on Github[1].

4.1.1 Code

In the experiments, we will compare Softmax with the alternative output layers: Mixture of Softmaxes (MoS), Sigsoftmax, Mixture of Sigsoftmaxes (MoSS), and Piecewise Linear Increasing Function model (PLIF). Since the bottleneck is identified in the standard Softmax, it is going to be our baseline. Henceforth, we will commonly refer to MoS and MoSS as the mixture models. The others are called non-mixture models. This should not be confused with the general family of mixture models in statistics.

The code for PLIF is courtesy of Ganea et al. (2019) and publicly available as part of experiments from Parthiban et al. (2021). For the other models, the code was written by the author by following the definitions in their original papers.

In the context of language modeling, Yang et al. (2018) modeled priors $\pi_{n,m}$ using a recurrent neural network. However, in our experiments, we will deal with images that do not have a natural equivalent of context. Thus, our priors need to be non-contextual. To do this, we use $\boldsymbol{\pi}' \in \mathbb{R}^M$ as the parameters. The priors are enforced to be non-negative and sum to one by transformation $\pi_m := \text{softmax}(\boldsymbol{\pi}')_m$.

[1] https://github.com/dzionek/softmax-bottleneck

4.1.2 Numerical stability

When evaluating softmax and its alternatives, it is important to avoid errors due to the fixed precision of floating-point numbers (IEEE, 2019).

Whenever possible, we use the built-in PyTorch functions that do not have problems with numerical stability, such as `torch.logsumexp`. However, sometimes we need to create our own numerically-stable functions. For instance, to evaluate $\log\left(\sum \exp(x)\sigma(x)\right)$.

The first type of problems we encountered is due to arithmetic overflow. The problem appears when raising a number to a very high power, e.g. evaluating $\exp(100)$ for softmax. To avoid this problem, a common strategy is to subtract the maximum logit (Blanchard et al., 2020):

$$\frac{\exp(x_i - \max_k x_k)}{\sum_j \exp\left(x_j - \max_k x_k\right)} = \frac{\exp(-\max_k x_k)\exp(x_i)}{\exp(-\max_k x_k)\sum_j \exp\left(x_j\right)} = \frac{\exp(x_i)}{\sum_j \exp\left(x_j\right)}.$$

The subtraction guarantees each subtracted logit is less than or equal to zero, and cannot overflow.

Example 4.1. $\text{softmax}([1, 10, 100]^\mathsf{T}) = \text{softmax}([-99, -90, 0]^\mathsf{T})$.

The other problem is arithmetic underflow. When we have a very small positive number it is stored as zero due to limited precision. This can cause problems with the logarithms of zero as the logarithm of zero is undefined. To fix this, the most common approach is to add a very small ε before taking the logarithm. The very small number is unlikely to change the final results while helping with the underflow. Following the approach from the code of Yang et al. (2018), we use $\varepsilon = 10^{-8}$.

4.1.3 Efficient implementation of mixtures

Note that in contrast to the figure presenting MoS (fig. 3.1), in the implementation we can learn all the last hidden layers and logits through just two linear layers. We are also able to vectorize both the multiplication by prior coefficients π_m and the final sum. In the thesis, we present the implementation of the most interesting example, MoSS, as figure 4.1. All M embeddings are learned using the `self.fc1` layer. Then, we obtain logits with the `self.fc2` layer. Priors are ensured to be valid with softmax and then multiplied element-wise with logits. Finally, by summing we get the output probabilities. The code snippet also employs the techniques to avoid underflow and overflow as discussed in section 4.1.2. For the remaining implementation, we refer the reader to the code repository.

```
class MixtureOfSigSoftmaxesNetwork(nn.Module):
    def __init__(self, d, M):
        super().__init__()
        self.fc1 = nn.Linear(PREVIOUS_HIDDEN_LAYER, d * M)
        self.fc2 = nn.Linear(d, K)
        self.prior = nn.Parameter(torch.randn(M), requires_grad=True)
        self.d = d
        self.M = M

    def sigsoftmax(self, logits, dim):
        stable_logits = logits - torch.max(logits)
        unnormalized = torch.exp(stable_logits) * torch.sigmoid(logits)
        return unnormalized / (
                    torch.sum(unnormalized, dim=dim, keepdim=True) + EPSILON)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = x.reshape((x.shape[0], self.M, self.d))

        prior = F.softmax(self.prior, dim=-1)
        output = self.sigsoftmax(self.fc2(x), dim=2)
        output = output * prior.view((1, self.M, 1)).repeat((x.shape[0], 1, K))
        output = torch.sum(output, dim=1)
        output = torch.log(output + EPSILON)
        return output
```

Figure 4.1: Our PyTorch implementation of MoSS.

4.2 Research question

In the following set of experiments, we want to answer the question: *Can the other types of output layers improve the standard softmax image classifier when $d < K$ while having comparable performance when $d \geq K$?* The performance of the classifier will be measured in terms of (average) cross-entropy (equation 2.1) and accuracy on the test set. Accuracy is simply defined as the ratio of the number of correctly classified examples to the total number of examples.

4.3 Handwritten digit recognition

4.3.1 Dataset

We start our evaluation with one of the most significant datasets in the history of deep learning (Wang and Raj, 2017): MNIST (LeCun et al., 1998b). The MNIST dataset is composed of 70 thousand 28x28 pixel images corresponding to handwritten digits. Each pixel takes a grayscale value between 0 and 255. We split the dataset into a training set with 60 thousand images and a test set with the remaining 10 thousand. A sample of the dataset is shown in figure 4.2.

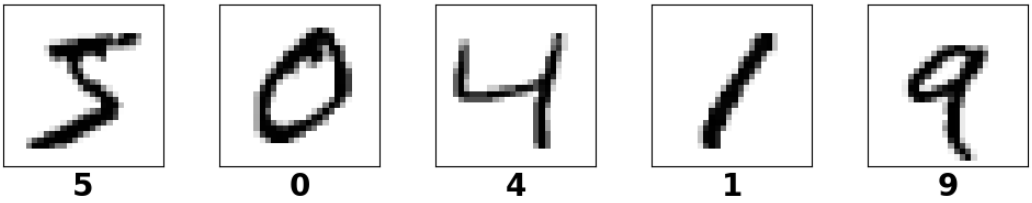


Figure 4.2: Example images from MNIST with their labels written under the images.

4.3.2 Methodology

One can notice that in the context of MNIST the number of classes is small ($K = 10$). It is easy to train a model with a high dimension of the last hidden layer and get a full-rank solution. In fact, the popular PyTorch example[2] for MNIST uses $d = 128$ and achieves almost 100% accuracy. However, we can still investigate the effect of different output layers by setting d to less than 10. In this way, the low-dimensional vector of the last hidden layer needs to be transformed into a higher-dimensional vector of the probabilities.

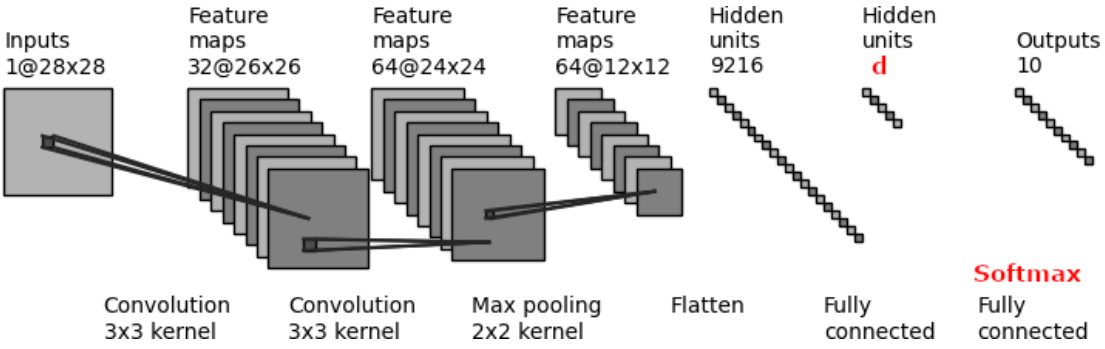


Figure 4.3: The network used for training on MNIST. We vary the size of the last hidden layer d and the activation for the output layer (in red). Using $d < 10$ imposes low-rank output.

Every compared model will be the same network except for the final layers. The baseline Softmax network is based on the PyTorch example[3]. To extract features from the images, we use convolutional layers with max-pooling (LeCun et al., 1998a). The inputs are then flattened and processed through a feed-forward neural network with two linear layers. The output is the probability of the image representing a particular digit. The visual representation of the network is given as figure 4.3. To improve the training process, we use the AdamW optimizer (Loshchilov and Hutter, 2017) with its default PyTorch learning rate 10^{-3}.

We run 50 different experiments. For each 5 different output layers, we train 10 different architectures with varying d:

- The first architecture with the dimension of the last hidden layer $d = 128$. This is a standard architecture from the PyTorch example. Since this network would

[2] Accessed on 15 March 2023: https://github.com/pytorch/examples/tree/main/mnist
[3] See footnote 2

overfit due to a high number of parameters, we use dropout for regularization (Srivastava et al., 2014). In detail, we use dropout with $p = 0.25$ after max pooling, and $p = 0.5$ after the first fully connected linear layer.

- We consider 9 different low-rank architectures where the penultimate layer has fewer neurons than the output: from $d = 9$ down to $d = 1$. These architectures do not use dropout because a lower number of parameters is a natural regularizer.

Each network is trained for 40 epochs because this is the number that leads to convergence in test accuracy, according to the manual examination of training curves by the author. To make sure the results do not depend on initial weight initialization, each experiment is re-run on 10 different seeds randomly sampled by the author. Each train and test loss and accuracy will be reported as the average of the different seeds plus-minus the standard deviation.

4.3.3 Results

d	Model	Test loss	Test accuracy (%)
128	Softmax	0.03 ± 0.0	99.37 ± 0.05
	Sigsoftmax	0.03 ± 0.0	99.37 ± 0.06
	PLIF	0.03 ± 0.0	99.38 ± 0.04
	MoS	0.03 ± 0.0	99.44 ± 0.05
	MoSS	$\mathbf{0.03 \pm 0.0}$	$\mathbf{99.48 \pm 0.04}$
5	Softmax	0.44 ± 0.56	84.84 ± 23.23
	Sigsoftmax	0.45 ± 0.42	87.36 ± 17.41
	PLIF	0.24 ± 0.29	92.92 ± 12.95
	MoS	$\mathbf{0.1 \pm 0.06}$	97.82 ± 1.68
	MoSS	0.12 ± 0.05	$\mathbf{98.3 \pm 0.52}$
3	Softmax	0.89 ± 0.81	69.58 ± 33.69
	Sigsoftmax	0.95 ± 0.8	68.95 ± 34.12
	PLIF	0.6 ± 0.69	78.56 ± 28.14
	MoS	$\mathbf{0.2 \pm 0.14}$	95.07 ± 3.58
	MoSS	0.26 ± 0.22	$\mathbf{96.78 \pm 0.65}$
1	Softmax	1.78 ± 0.43	33.79 ± 18.6
	Sigsoftmax	1.74 ± 0.47	35.4 ± 19.91
	PLIF	1.75 ± 0.56	32.13 ± 21.16
	MoS	1.47 ± 0.29	41.2 ± 11.0
	MoSS	$\mathbf{1.3 \pm 0.36}$	$\mathbf{48.22 \pm 13.55}$

Table 4.1: Comparison of test cross-entropy and accuracy on MNIST by varying d. Reported values are means and standard deviations based on 10 different random seeds. In bold, we present the best means for each d. Full results are presented in tables B.1 & B.2.

Cross-entropy and accuracy

The full results of the experiments are available in the appendix as tables B.1 & B.2. Here, we will focus on the insights from selected groups of experiments.

First, let us consider a subset of the results as given in table 4.1. It is apparent that in a high-rank setting $d = 128$, all of the models achieved high accuracy and low cross-entropy. The differences in performance of different layers are minimal. Interestingly, MoS and MoSS give the highest accuracies. It shows that even though the alternative layers are designed for low-rank cases, they do similarly well as Softmax when the rank is high. Moreover, when $d = 128$, the standard deviations are very low, which shows we can get a consistent performance irrespective of the random seed used.

When we decrease d to be less than K, we notice bigger differences between the output layers. When $d = 5$, we see a big increase in variance for the non-mixture models. Softmax gives the lowest accuracy and highest variance. Sigsoftmax seems to give performance similar to Softmax. PLIF outperforms Softmax and Sigsoftmax in accuracy but gives worse accuracy than MoS and the best MoSS. The reported losses behave differently than accuracies but generally point to the same superiority of the mixture models.

If we decrease d even further to $d = 3$, the effect observed in decreasing d to 5 seems to be even more pronounced. MoS and MoSS are able to retain high accuracy. Notably, MoSS continues to give remarkably low standard deviation of accuracies. PLIF is still between the mixture models and Softmax/Sigsoftmax with loss and accuracy, and Softmax/Sigsoftmax are still the worst with similar scores.

In the most extreme case, when $d = 1$, all models report high variance. Non-mixture models have both the lowest accuracy and the highest variance. This time PLIF does not outperform Softmax and Sigsoftmax. Mixture models are the best but the difference in percentage points is smaller. However, the difference in accuracy between MoSS and MoS is even bigger in favor of MoSS, despite its higher variance.

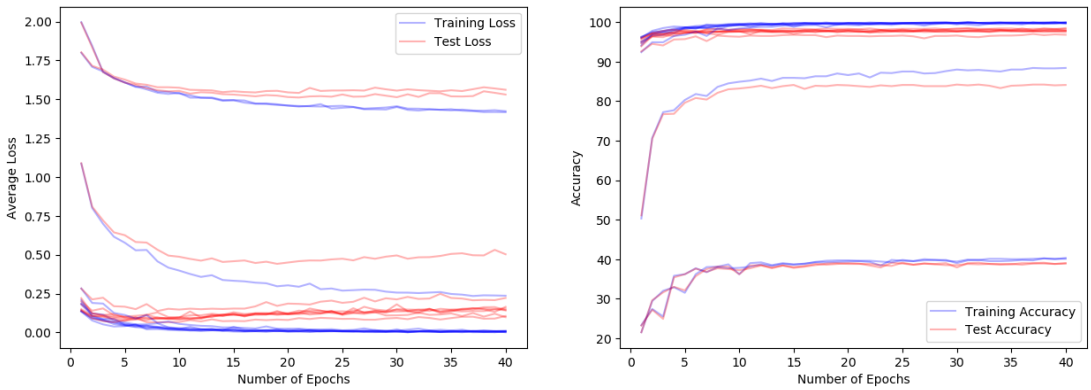


Figure 4.4: Training curves with cross entropy and accuracy for Softmax and $d = 5$.

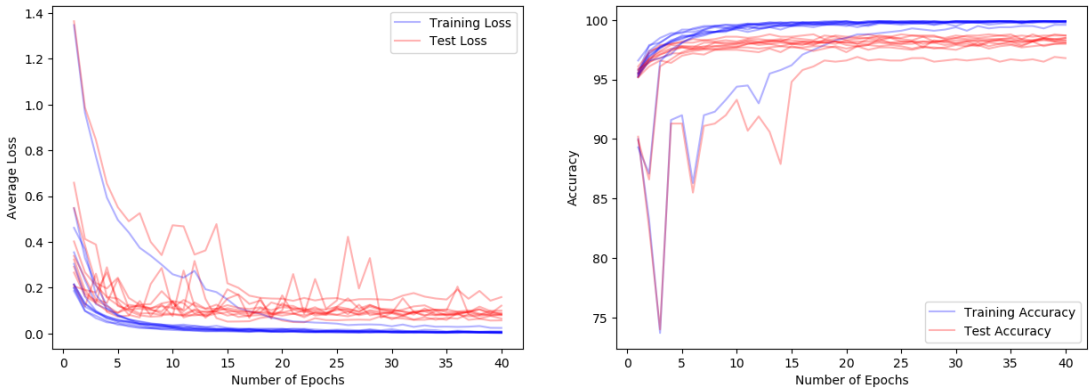


Figure 4.5: Training curves with cross entropy and accuracy for MoSS and $d = 5$.

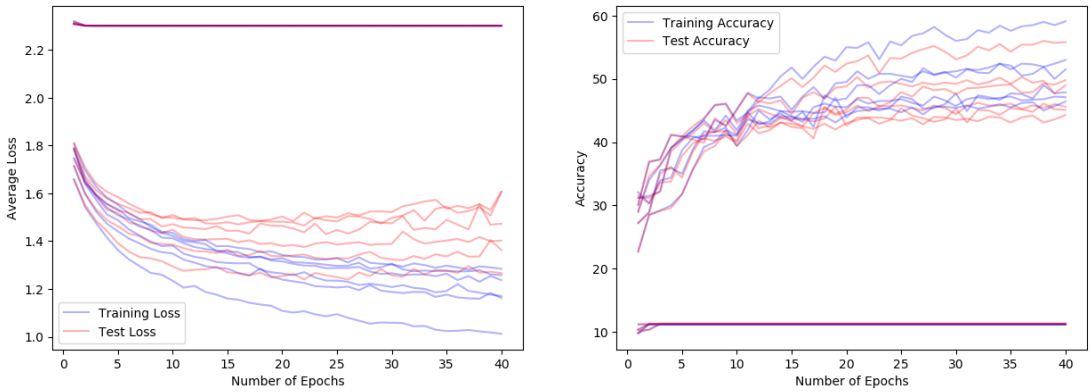


Figure 4.6: Training curves with cross entropy and accuracy for Softmax and $d = 1$.

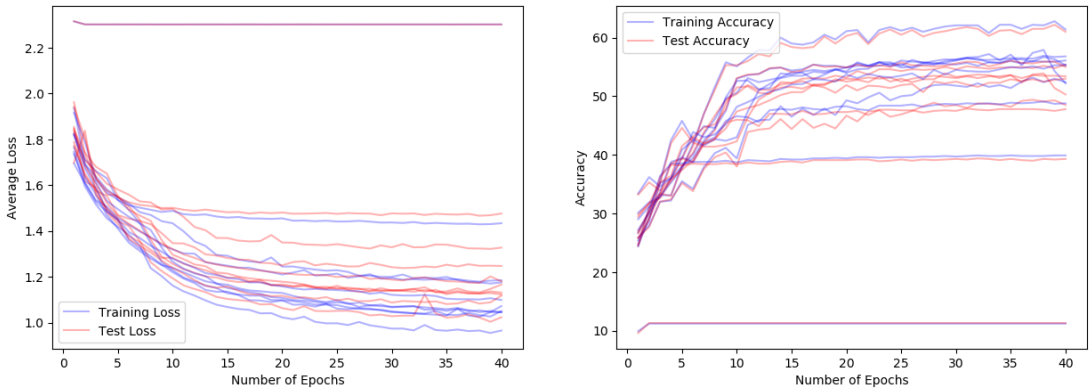


Figure 4.7: Training curves with cross entropy and accuracy for MoSS and $d = 1$.

Training curves and unlucky seeds

We note that the results in the experiments are heavily influenced by the high variability between different seeds. To better understand what happened, we can look at training curves. In the training curves we present, each line corresponds to a different seed. There are two types of patterns we can identify.

The first pattern is when the network is able to learn and occasional lower accuracies

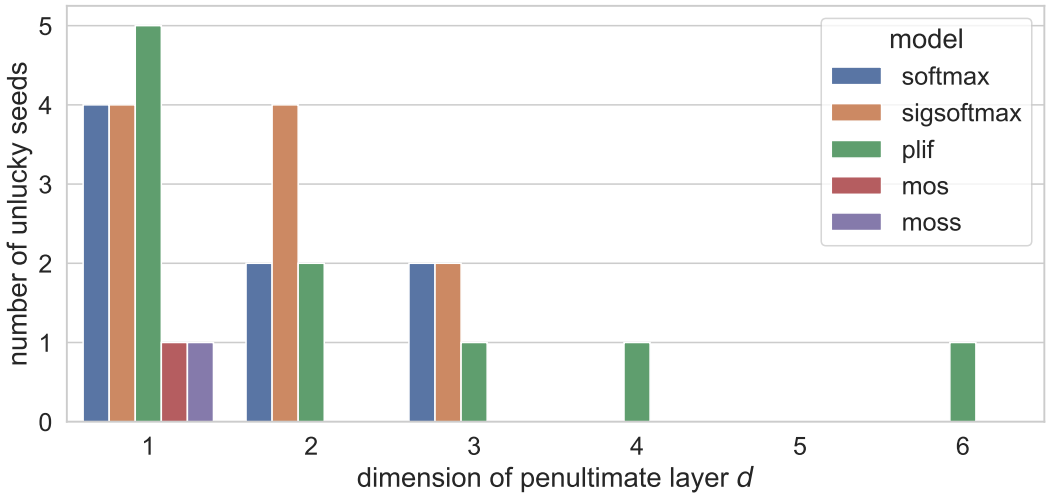


Figure 4.8: The number of unlucky seeds that led to test accuracy below 13%.

might be due to the low-rank weakness of the architecture. This is the case when $d = 5$. Figure 4.4 depicts Softmax and one can see that two seeds led to the accuracy of about 80% and 40%. However, for MoSS, as shown in figure 4.5, all seeds gave over 95% accuracy.

On the other hand, the second pattern is when the network was not able to learn properly. In the figure 4.6, we see that for Softmax at $d = 1$ four seeds were stuck at about 11% accuracy and 2.3 average cross-entropy. For MoSS, only one seed led to such low performance (fig. 4.7). In this thesis, we refer to such seeds that lead to accuracy below 13% as "unlucky". We plot a histogram of those unlucky seeds by each d and model, and present it in figure 4.8. The highest d for which we found an unlucky seed was $d = 6$ for PLIF. Nonetheless, we can observe the unlucky seeds are mainly visible for very low values of d. It is likely that this is related to more difficult training with ultra-small d. Comparing different models, we see the mixture models had only one unlucky seed each at $d = 1$, compared to many more unlucky seeds for non-mixture models. It is disappointing to see Sigsoftmax and PLIF having the same number of unlucky seeds or bigger than Softmax. We hypothesize this is the case because Softmax is well-optimized in PyTorch, compared to our custom activations.

We note that for unlucky seeds the classifier has a performance similar to a random classifier. For a random classifier over 10 classes, we have $\hat{\mathrm{P}}(C_k \mid \boldsymbol{\phi}) = 0.1$ for each class k. With perfectly balanced dataset we can expect the accuracy to be 10% and the average cross-entropy to be $-\log \hat{\mathrm{P}}(C_k \mid \boldsymbol{\phi}) = -\log(0.1) \approx 2.3$.

During designing the experiments, we found that selecting an optimizer had a significant impact on the number of unlucky seeds. In particular, when we earlier used the Adam optimizer (Kingma and Ba, 2014) instead of AdamW (Loshchilov and Hutter, 2017), we witnessed more unlucky seeds across all activations.

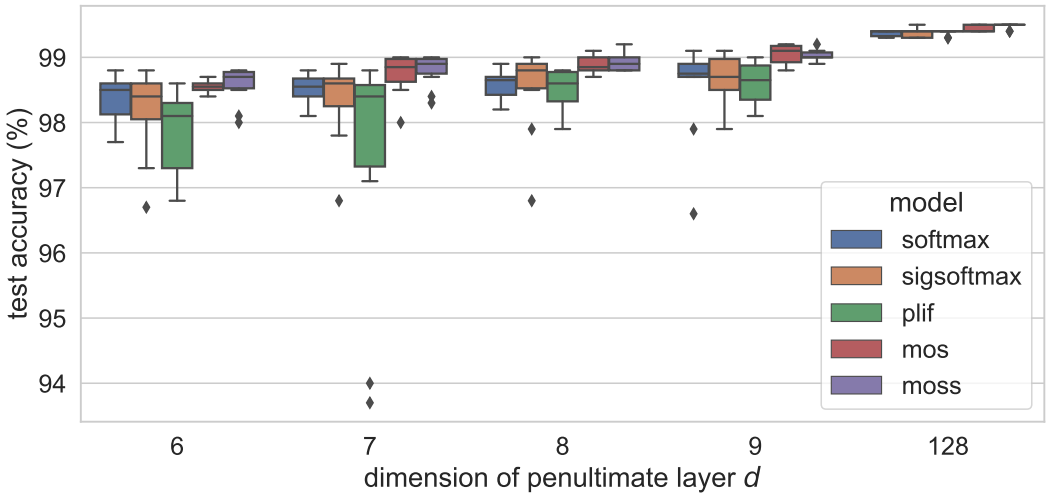


Figure 4.9: Test accuracy after discarding unlucky seeds for high values of d.

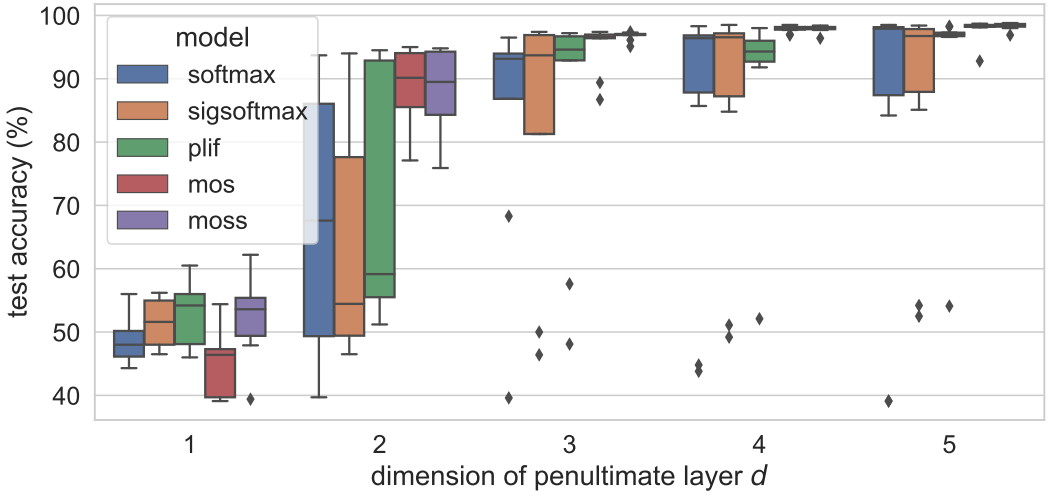


Figure 4.10: Test accuracy after discarding unlucky seeds for low values of d.

Analysis after discarding unlucky seeds

We think it is worth noting that mixture models led to better training across more seeds. In spite of this, we can also analyze potential improvements from a different perspective. Let us consider it is not a problem to run an experiment on multiple seeds and discard results from unlucky seeds (by a simple test of accuracy below 13%).

If we discard unlucky seeds for $d = 6, 7, 8, 9, 128$ (only one unlucky seed in this group, for PLIF and $d = 6$), we get distributions as in figure 4.9. While all models perform relatively well for these values of d, we notice PLIF has the most significant drops in accuracy (especially $d = 7$ and unlucky seed at $d = 6$). We can also say MoS and MoSS are slightly better performing than non-mixture models.

For lower values of d, the distributions after removing unlucky seeds for each model are given in figure 4.10. We can say that MoS and MoSS had much less variance in their test accuracy than non-mixture models for $d \geq 2$. The most interesting example is $d = 2$ where the variances of non-mixture models are very large. Consequently, their mean accuracy is much lower. But if we look at the top-performing seeds, all models are able to attain a very high accuracy of approximately 93%. Surprisingly, if we check $d = 1$, MoS was actually the worst model in both the lowest, mean, and highest reported accuracy. MoSS had the highest-scoring seeds, confirming its superiority across all values of d.

4.3.4 Conclusions & limitations

Based on the MNIST experiments we can say that:

- The alternative output layers do offer similarly high performance to Softmax when $d > K$. They can be also equally good for higher ranks, such as in our network when $d \geq 6$.
- We can notice there are two groups based on the analysis of performance: mixture models (MoS and MoSS) perform similarly to each other, and the second group is non-mixture models. In general, mixture models gave higher accuracy than non-mixture models for all values of d.
- MoSS was the best model in our tests. It was better than the second-best MoS, having both higher mean and lower standard deviation across different seeds.
- In the low-rank case, our MNIST network suffered from unlucky seeds that made the classifier unable to learn anything beyond random choice.
- The success of MoS and MoSS is largely due to being more resistant to unlucky seeds. We hypothesize that since they have more parameters, they can be easier to optimize for very low values of d. Sigsoftmax and PLIF were more susceptible to unlucky seeds than Softmax. Especially PLIF has shown big dependence on appropriate seeds.
- We cannot conclude any model is always better than the rest. Even with small $d = 2$, there were seeds that gave similarly high 93% accuracy. We can however say, it was the easiest with MoSS to get high accuracy. On the other hand, Sigsoftmax and PLIF did not make a meaningful difference compared to the commonly used Softmax.

It is also worth pointing out the limitations of our MNIST experiments:

- The original formulation of the softmax bottleneck is related to a high number of classes and high context-dependence in natural languages. None of the conditions apply to handwritten digit recognition. In particular, the standard approach is to use $d > K$, in contrast to language modeling (see section 2.5.2).
- When evaluating MoS and MoSS, one needs to consider the number of mixtures M as an additional hyperparameter. Also for PLIF, the number of knots and the size of the interval can impact the final performance. Due to the limited

time and computational resources, the author run all experiments with the default parameters based on the code from papers: $M = 10$ for mixtures, and 100 thousand knots with the interval $[-20, 20]$ for PLIF.

- The sensitivity to weight initialization shows that running the experiments on 10 seeds might not be enough to get clear results. Additionally, the influence of the choice of an optimizer, we identified, shows one should spend more time adjusting optimizers and their hyperparameters.

4.4 Species classification

The experiments on MNIST were made on a small dataset with a low number of classes. Let us now consider a dataset with a higher number of classes.

4.4.1 Dataset

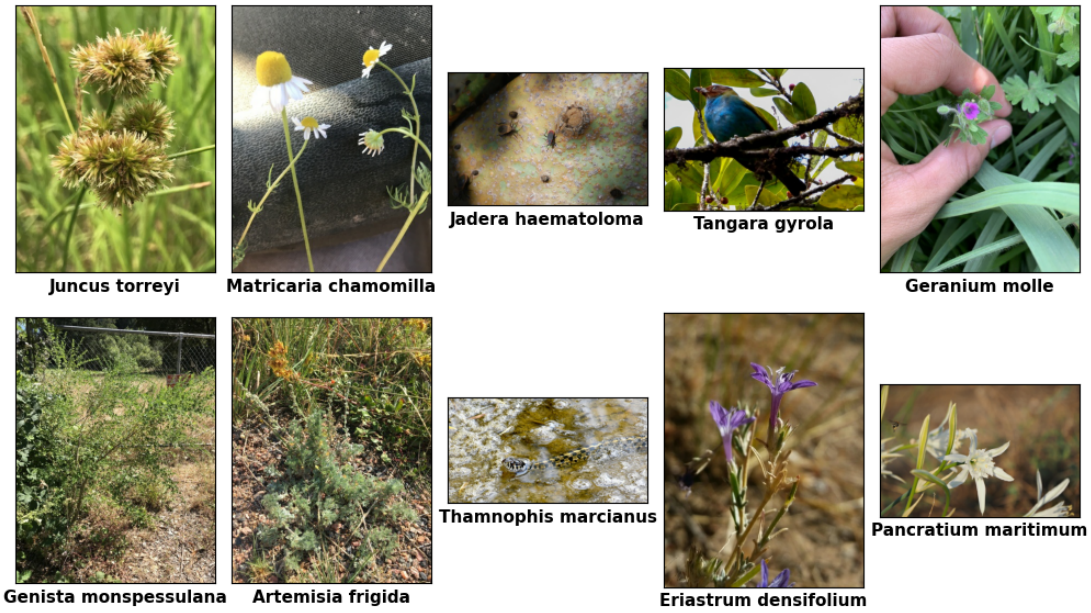


Figure 4.11: Representatives of our smaller subset of iNaturalist with 100 classes. Decoded labels are given under the images.

In this section, we will consider the task of species classification. Given a picture of an animal or a plant, the classifier needs to recognize the class the picture belongs to. The dataset we will use comes from iNaturalist (Horn et al., 2017). In particular, we will use the dataset from the 2021 challenge[4]. This dataset comprises images of 10 thousand species. In the mini version of the dataset, the training set has 50 images for each species. The test set comprises 10 images per species. However, we note that even the mini dataset is 50GB in size. It significantly exceeds the computational resources of the computer cluster offered to undergraduate students. Therefore, in the thesis, we

[4]Accessed on 19 March 2023: https://www.kaggle.com/competitions/inaturalist-2021

are forced to limit our analysis to a smaller subset of 100 classes. We randomly sample these 100 classes. We present some examples of images from our smaller test set in figure 4.11.

4.4.2 Methodology

Instead of training all parts of a model, a common approach in classifying images is transfer learning. The method consists of using a large pre-trained model that is later fine-tuned on a downstream task. Transfer learning is especially helpful when one can leverage a model trained on a big dataset in training a new model on a small dataset (Hussain et al., 2019). We note that this is the case in our experiment.

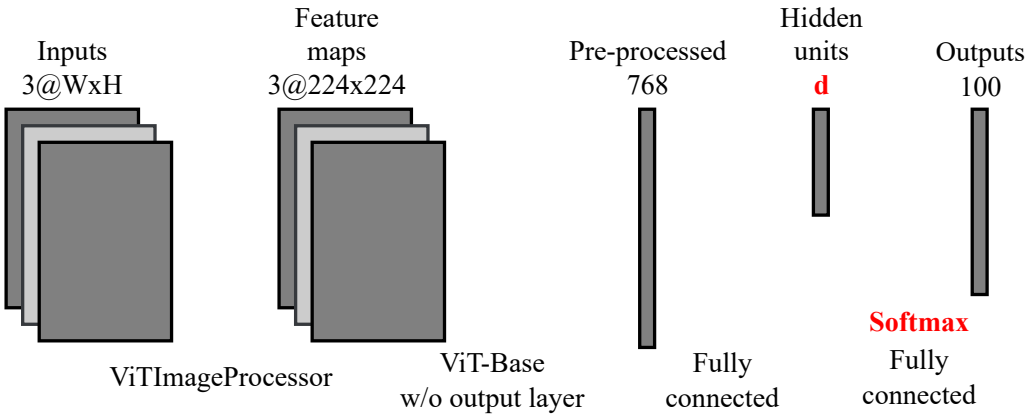


Figure 4.12: The network used for training on our subset of iNaturalist 2021. The initial image can be of arbitrary dimensions, it will be scaled appropriately to 224x224 by ViTImageProcessor. Then, the image is passed through ViT-Base without the output layer. We treat ViT-Base as a pre-processing step and do not update its weights during training. Such pre-processed images are then trained on two linear layers with varying $d < 100$ and output activations (in red), similarly as for MNIST.

Our pre-trained model is going to be Vision Transformer (ViT) which gives state-of-the-art scores on image classification benchmarks (Dosovitskiy et al., 2020). Vision Transformer splits images into patches and uses them as an input to a transformer (Vaswani et al., 2017). Hence, we can think of the patches as equivalents of tokens in language modeling. The ViT model was pre-trained on the ImageNet dataset (Deng et al., 2009). Since ImageNet contains a large collection of species, we think ViT will be a reasonable choice for transfer learning on iNaturalist.

In the experiments, we use the base version of ViT (ViT-Base has 86 million parameters). To make sure the images are of the correct dimensions, we will first pass them through the model's pre-processing tool. In our approach to transfer learning, we freeze all the weights of ViT. We remove the output layer of ViT and instead connect its last hidden layer with 2 new linear layers that will be trained to give output corresponding to 100 classes. The visualization of our approach is shown in figure 4.12. Our linear layers are trained for 200 epochs and optimized using the AdamW optimizer (Loshchilov and Hutter, 2017) with the default learning rate 10^{-3}.

Similarly to MNIST, we will vary d in our experiments to compare the training performance of different output layers. For experiments, we pick $d \in \{32, 16, 8, 4, 2, 1\}$. For statistical consistency, every experiment is run with 10 different seeds and is reported with the mean and standard deviation.

4.4.3 Results

d	Model	Test loss	Test accuracy (%)
32	Softmax	0.62 ± 0.07	86.2 ± 0.5
	Sigsoftmax	0.72 ± 0.11	85.87 ± 0.69
	PLIF	0.62 ± 0.11	85.76 ± 0.62
	MoS	$\mathbf{0.6 \pm 0.07}$	$\mathbf{86.68 \pm 0.52}$
	MoSS	0.68 ± 0.13	86.39 ± 0.64
16	Softmax	0.86 ± 0.18	81.8 ± 0.91
	Sigsoftmax	0.92 ± 0.23	80.86 ± 0.9
	PLIF	$\mathbf{0.78 \pm 0.02}$	80.88 ± 0.85
	MoS	0.83 ± 0.11	$\mathbf{82.03 \pm 0.57}$
	MoSS	0.96 ± 0.15	81.92 ± 0.52
8	Softmax	$\mathbf{1.23 \pm 0.07}$	71.6 ± 1.03
	Sigsoftmax	1.45 ± 0.1	69.72 ± 0.95
	PLIF	1.24 ± 0.06	70.6 ± 1.01
	MoS	1.26 ± 0.09	$\mathbf{73.79 \pm 1.39}$
	MoSS	1.37 ± 0.05	72.65 ± 1.1
4	Softmax	2.34 ± 0.12	45.79 ± 1.75
	Sigsoftmax	2.64 ± 0.18	44.59 ± 2.2
	PLIF	2.44 ± 0.14	45.49 ± 1.6
	MoS	$\mathbf{2.29 \pm 0.11}$	$\mathbf{50.78 \pm 1.19}$
	MoSS	2.69 ± 0.18	46.12 ± 1.51
2	Softmax	3.69 ± 0.11	11.37 ± 0.89
	Sigsoftmax	3.82 ± 0.16	11.46 ± 0.86
	PLIF	3.65 ± 0.11	$\mathbf{13.39 \pm 0.84}$
	MoS	$\mathbf{3.52 \pm 0.06}$	12.61 ± 0.84
	MoSS	3.9 ± 0.35	12.64 ± 0.97

Table 4.2: Test cross entropy and accuracy for various d. Reported values are means and standard deviations based on 10 different random seeds. In bold, we present the best means for each d. Full results are in table B.3.

Cross-entropy and accuracy

The full results of our experiment are given in table B.3. Table 4.2 contains the main results we discuss in this section.

Our first observation is transfer learning from ViT is a good choice to classify images in our sample of iNaturalist. Using small $d = 32$, we are already able to get high test accuracy ($> 85\%$) and low cross-entropy. We can expect that with higher d, regularization, and tuning hyperparameters the metrics would be even higher.

In terms of the different models we used, we see that for $d = 32$ and $d = 16$, all models have similar test accuracy. However, MoS seems to be slightly better in accuracy than the other models.

For $d = 8$, we go more low-rank and both accuracy and loss scores are already much worse than for higher values of d. MoS is still slightly better, but the difference between the accuracy of MoS and other models increases. Sigsoftmax is the worst in both loss and accuracy, in particular, it performs worryingly worse than Softmax.

For $d = 4$, MoS is clearly better than other models in terms of accuracy. All other models perform very similarly to each other in accuracy. However, the difference between MoS and Softmax measured in loss is not as big as in accuracy.

With $d = 2$, the network becomes too low-rank and all models perform very badly. Interestingly, PLIF is the best (or rather the least bad) in accuracy.

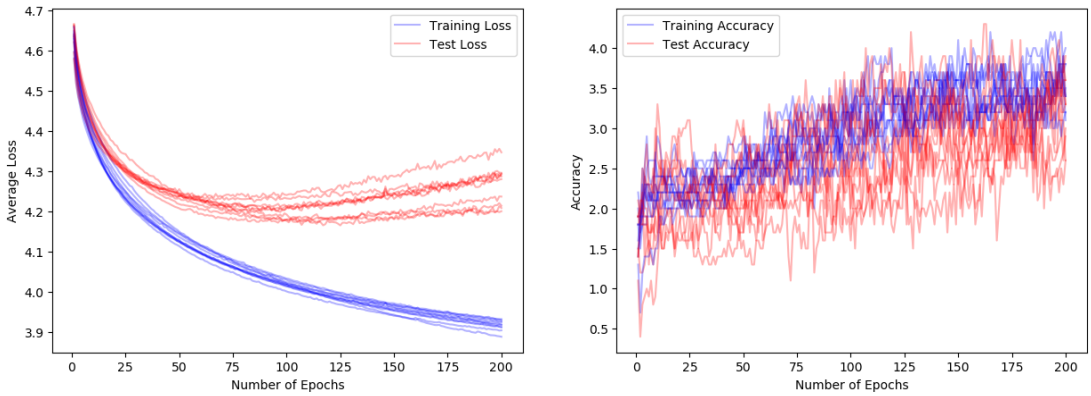


Figure 4.13: Training curves with cross-entropy and accuracy for Softmax and $d = 1$.

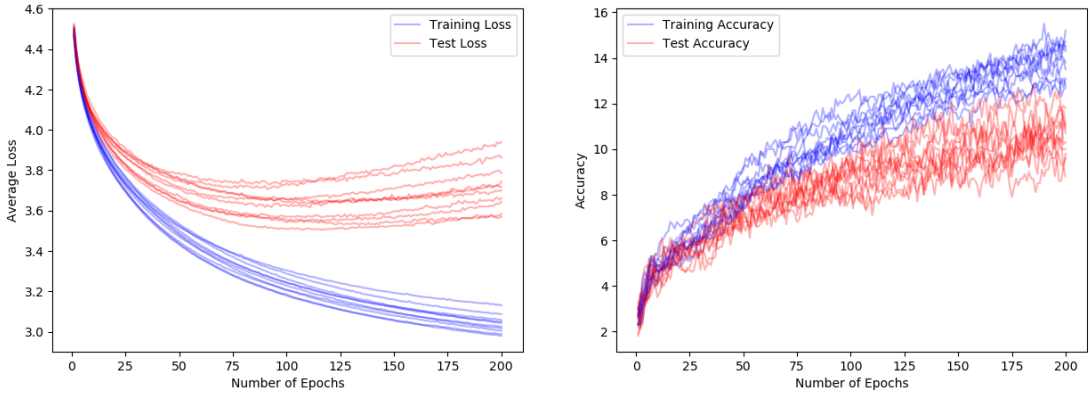


Figure 4.14: Training curves with cross-entropy and accuracy for Softmax and $d = 2$.

Training curves

In contrast to MNIST, this time we did not experience unlucky seeds that would make the network unable to learn. Overall, the variance between different seeds was very low as shown in table 4.2. Even for very small values of d and using Softmax, we get very

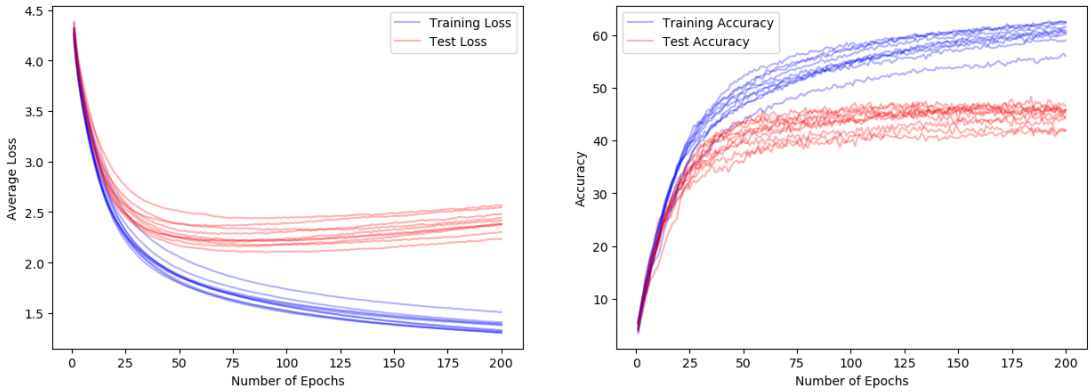


Figure 4.15: Training curves with cross-entropy and accuracy for Softmax and $d = 4$.

similar training curves in every 10 seeds. Figures 4.13, 4.14 and 4.15 depict training curves for Softmax on $d = 1, 2, 4$ respectively. We notice based on the loss curves that training proceeds as expected without unlucky seeds. We also note that as we increase d, the training curves become less and less wiggly.

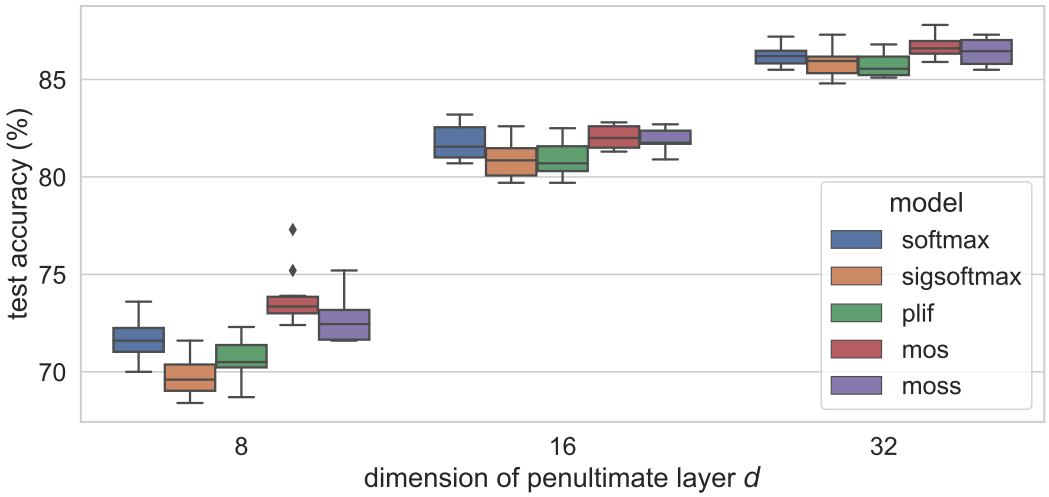


Figure 4.16: Test accuracy across 10 seeds for high values of d.

Reliable training between seeds

To better see that training gave a very similar performance on different seeds, let us again looks at boxplots. This time, however, we do not discard any seeds as the plots are readable even with all seeds.

Figure 4.16 presents the distribution of accuracies for high values of d. For $d = 16$ and $d = 32$, we see very similar distributions across models. For $d = 8$, we start seeing MoS is the best. The only unexpected outlier points on this plot come from MoS and are due to even higher accuracy than for most of its seeds. Overall, based on this plot, we might be disappointed with Sigsoftmax and PLIF. They perform worse than Softmax across different d-s, even though they were invented to outperform it.

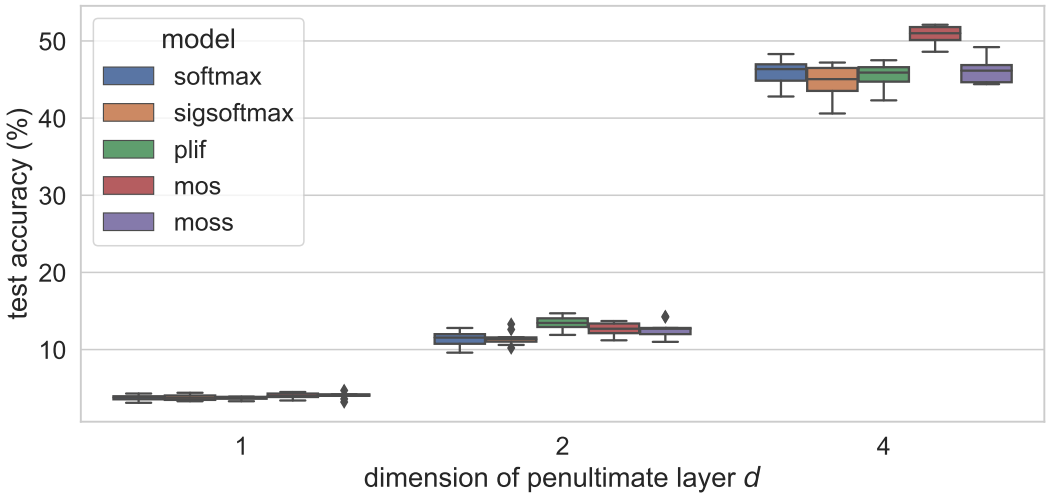


Figure 4.17: Test accuracy across 10 seeds for low values of d.

Figure 4.17 illustrates accuracies for lower values of d. For $d = 4$, we see the dominance of MoS even more clearly. Even the worst-performing seed for MoS was better than the best-performing seed for Softmax. Since the values $d = 2$ and $d = 1$ are very low, none of the models can retain high accuracy and the differences between models are minimal.

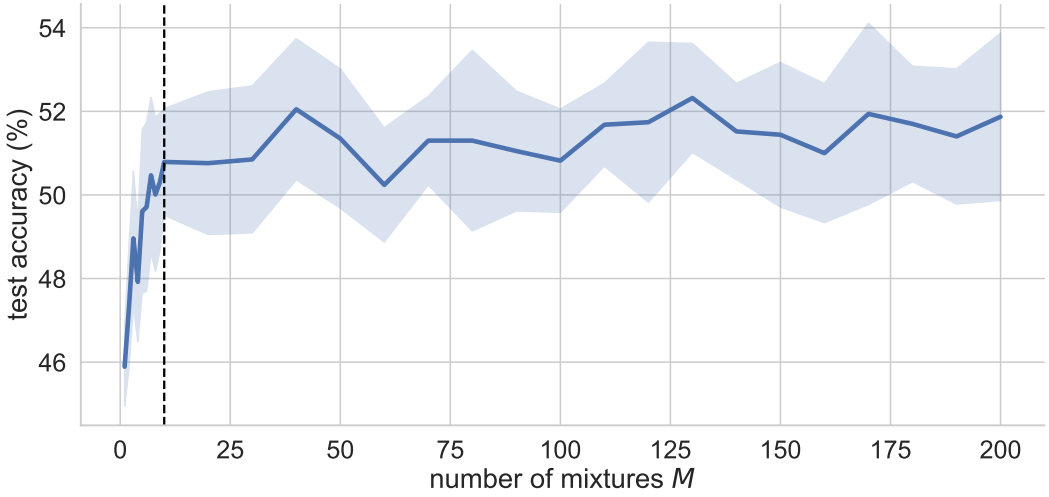


Figure 4.18: Test accuracy of MoS on iNaturalist when $d = 4$ and with varying M. The line represents the mean and the shadow standard deviation based on 10 different seeds. The dashed line shows the previously evaluated $M = 10$.

Impact of the number of mixtures

Based on our results, we might think that MoS was the best choice for low-rank d. Nevertheless, it is not obvious how much improvement can we expect from MoS because it adds an additional hyperparameter to tune, the number of mixtures M. As far

as we are aware, there does not exist any literature about picking the optimal M. Thus, in the follow-up experiment, we pick MoS as our activation and keep $d = 4$ because for it the MoS brought the biggest improvement in accuracy. We test 29 different values for M, in the set $\{1, 2, 3, \ldots, 9, 10, 20, 30, \ldots, 200\}$ to measure how sensitive the model is to M. The network architecture and hyperparameters are the same as before.

We present our result as figure 4.18. When $M = 1$, MoS reduces to Softmax, so has the same accuracy as presented in table 4.2. As M increases to 10, we see an increase in accuracy. However, with values of M higher than 10, we get to the point of diminishing returns. The optimal accuracy was found for $M = 130$ but it was not much better than for $M = 10$. The experiment also shows that even with very high M we will not be able to get high accuracy when the rank is very low.

4.4.4 Conclusions & limitations

The species classification experiments again demonstrate mixture models are able to improve accuracy in the low-rank case while being equally good for high rank. The higher accuracy was more pronounced in the case of MoS, MoSS only slightly outperformed Softmax. Sigsoftmax and PLIF were on par with Softmax, with PLIF being only slightly better for ultra-low d. However, it is worth mentioning we did not engage in tuning hyperparameters of PLIF and used the same as for MNIST.

Based on the analysis of tuning the number of mixtures M, we can say MoS is not particularly sensitive to M. Even a small value of M can lead to better accuracy and there is no need to increase the computational time and memory with high M. The limitation of this experiment is having a fixed d. It is possible that for even smaller d it would be necessary to have higher M. When d is very low, for example, $d = 2$, MoS was not able to yield significant improvement. It would be interesting to see if having higher M would allow it to bring improvement.

While our study provides interesting results for 100 classes, ideally the experiments should be done on the whole dataset with 10 thousand classes. When dealing with 10 thousand classes, the network would have a natural rank bottleneck. This is due to transfer learning as the dimension of the penultimate layer for the Vision Transformer ranges from 768 (ViT-Base) to 1280 (ViT-Huge). The fine-tuning network would need to transform such lower-dimensional vectors into 10000 dimensions. However, this was not possible with the servers of the author, and we leave these experiments for future work.

5

Conclusions

5.1 Summary and findings

In this thesis, we managed to explain the theory of the softmax bottleneck to make it easier to understand to an undergraduate-level student. We included the necessary background on multiclass classification and language modeling and kept the notation consistent throughout the thesis. Since we discussed the main solutions proposed in the literature, this paper can serve as a survey of the low-rank softmax bottleneck literature. Our contribution encompasses presenting examples and figures to deepen understanding of the theory and providing open-source implementation to foster future research.

The main contribution is generalizing the concept of bottleneck to any neural multiclass classification with the dimension of the penultimate layer lower than the number of classes. For the first time ever, we evaluated the solutions to the low-rank softmax problem in image classification. To do this, we wrote our own implementation of Sigsoftmax, MoS, and MoSS.

First, we looked at a small number of classes in the handwritten digit recognition task. We implemented a convolutional neural network and looked at the impact of changing d and output activation on accuracy and cross-entropy. All models were equally good if $d \geq K$. However, when $d < K$, we noticed MoS and MoSS gave higher accuracy than Softmax and were much more reliable across different seeds. They suffered much less from the problem of "unlucky" seeds.

Then, we analyzed the more challenging task of species classification. We demonstrated that transfer learning from Vision Transformer is an effective way to train a classifier on iNaturalist. Once again, we provided evidence that MoS and MoSS can yield higher accuracy than Softmax for $d \ll K$.

Therefore, we can recommend trying MoS and MoSS as an alternative to Softmax in neural networks when the penultimate layer dimension is significantly less than the output layer dimension. We cannot say which one is the better of the two activations. MoSS seemed to be better on MNIST, but MoS outperformed it on iNaturalist.

Our iNaturalist experiments show that even a small number of mixtures (e.g. $M = 10$)

can be close to optimal, so there is no need for a significant increase in computational time with higher M. There is also no reason for an extensive search for M.

Based on our results, we can also suggest trying PLIF as it was sometimes better than Softmax. On the other hand, we did not observe any meaningful improvements from (single) Sigsoftmax.

Overall, the low-rank softmax bottleneck is far from being solved. When d decreases to small values, the performance of MoS and MoSS also decreases. For extremely small d, their accuracy was much lower than for high d and on par with Softmax. Even with a very high number of mixtures M, the mixture models are not able to get the same accuracy as when d is higher. The question remains whether it is possible to always attain a high rank, and if a high rank would lead to high performance.

5.2 Limitations

It is worth reiterating the main limitations we identified in this thesis.

Our experiments are based on a different domain that does not exhibit the same problems as language modeling. In particular, the main hypothesis motivating the low-rank softmax bottleneck is based on two assumptions of Yang et al. (2018):

- The number of classes K is very large, e.g. $K \approx 10^5$ for language modeling. Having very high d, for instance, $d := K$, would lead to an explosion in the number of parameters and overfitting. However, from the perspective of an undergraduate student, we were not able to access hardware that would make such experiments possible. The experiments we conducted had low K of 10 and 100. The standard approach on those datasets would be to use d higher than K, add regularization, and consequently get high accuracy.
- The output is expected to form a high-rank matrix. This assumption was based on the hypothesis that a natural language is highly context-dependent, so the matrix $\mathbf{A}$ should have uncorrelated entries and have a high rank. This was not the case in our experiments. There was no concept of a context for images, all images were independent of each other.

Besides, our experiments did not involve extensive hyperparameter tuning. While the goal was to fairly evaluate different output layers on the same configurations, it is possible that different models can perform optimally for different hyperparameters. In particular, we experienced unlucky seeds on MNIST but not on iNaturalist. We think more work should be done to explain why this happened.

5.3 Future work

The most natural line of future research is trying the solutions to the bottleneck problem on different datasets in image classification, especially those with a high number of classes. Besides, evaluation in yet another type of multiclass classification (beyond

images and language) would allow us to better assess the general applicability of models as MoS.

Due to limited time, the author was not able to cover all papers on the low-rank softmax bottleneck. In future work, it would be interesting to see a broader discussion. For example, Chang and McCallum (2022) continue the topic by addressing a limitation of Mixture of Softmaxes when the embedding space distribution has multiple modes. Demeter et al. (2020), on the other hand, analyze the problem geometrically. They show softmax can limit the expressiveness of a language model when the words are on the interior of a convex hull of the embedding space.

It is also necessary to look into the literature suggesting the bottleneck might not be a problem in practice. Parthiban et al. (2021) experimentally demonstrated that rank is neither a necessary nor sufficient condition for the good perplexity of language models. Grivas et al. (2022) evaluated many large language models and machine translation models and also concluded the low-rank softmax had little impact on the performance.

Finally, while the evaluation of RNN-based language models motivated the early research on the softmax bottleneck, it is vital to assess its impact on modern transformer architectures (Vaswani et al., 2017). The state-of-the-art language models like the GPT family (Radford et al., 2018) or BERT (Devlin et al., 2019) employ linear softmax over vocabulary in their output layer. The current research points out to low usefulness of MoS in transformers. In particular, Narang et al. (2021) experimentally show using softmax and MoS in transformers led to a comparable performance on different tasks. Moreover, Tay et al. (2022) analyze how the performance of different transformer architectures scales with the number of parameters and computation time. Their findings show the linear softmax transformer gives better performance than MoS transformer when using the same computational time. However, if time is not an issue, a simple change of softmax to MoS was able to slightly increase performance. So, overall, we think the validity of using MoS and other solutions presented in this thesis in transformers is not yet resolved and requires further research.

Bibliography

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Pierre Blanchard, Desmond J Higham, and Nicholas J Higham. 2020. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4):2311–2330.

Mathieu Blondel. 2019. Structured prediction with projection oracles. *Advances in neural information processing systems*, 32.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

John Bridle. 1989. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Haw-Shiuan Chang and Andrew McCallum. 2022. Softmax bottleneck makes language models unable to represent multi-mode word distributions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8048–8073, Dublin, Ireland. Association for Computational Linguistics.

L.O. Chua and Sung Mo Kang. 1977. Section-wise piecewise-linear functions: Canonical representation, properties, and applications. *Proceedings of the IEEE*, 65(6):915–929.

David Demeter, Gregory Kimmel, and Doug Downey. 2020. Stolen probability: A structural weakness of neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2191–2197, Online. Association for Computational Linguistics.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929.

J. Eisenstein. 2019. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning series. MIT Press.

Octavian Ganea, Sylvain Gelly, Gary Becigneul, and Aliaksei Severyn. 2019. Breaking the softmax bottleneck via learnable monotonic pointwise non-linearities. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2073–2082. PMLR.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Andreas Grivas, Nikolay Bogoychev, and Adam Lopez. 2022. Low-rank softmax can have unargmaxable classes in theory but rarely in practice. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6738–6758, Dublin, Ireland. Association for Computational Linguistics.

Grant Van Horn, Oisin Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. 2017. The inaturalist challenge 2017 dataset. *CoRR*, abs/1707.06642.

Mahbub Hussain, Jordan J Bird, and Diego R Faria. 2019. A study on cnn transfer learning for image classification. In *Advances in Computational Intelligence Systems: Contributions Presented at the 18th UK Workshop on Computational Intelligence, September 5-7, 2018, Nottingham, UK*, pages 191–202. Springer.

IEEE. 2019. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84.

Daniel Jurafsky and James H Martin. 2008. Speech and language processing: An introduction to speech recognition, computational linguistics and natural language processing. *Upper Saddle River, NJ: Prentice Hall*.

Sekitoshi Kanai, Yasuhiro Fujiwara, Yuki Yamanaka, and Shuichi Adachi. 2018. Sigsoftmax: Reanalysis of the softmax bottleneck. NIPS'18, page 284–294, Red Hook, NY, USA. Curran Associates Inc.

Taghi Khoshgoftaar. 2020. Survey on categorical data for neural networks. *Journal of Big Data*, 7.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. *1995 International Conference on Acoustics, Speech, and Signal Processing*, 1:181–184 vol.1.

Andrey Kolmogorov. 1950. *Foundations of the theory of probability*. Chelsea Publishing Company, New York, USA.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2018. Dynamic evaluation of neural sequence models. In *International Conference on Machine Learning*, pages 2766–2775. PMLR.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998a. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998b. MNIST handwritten digit database.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Sharan Narang, Hyung Won Chung, Yi Tay, Liam Fedus, Thibault Fevry, Michael Matena, Karishma Malkan, Noah Fiedel, Noam Shazeer, Zhenzhong Lan, Yanqi Zhou, Wei Li, Nan Ding, Jake Marcus, Adam Roberts, and Colin Raffel. 2021. Do transformer modifications transfer across implementations and applications? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5758–5773, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dwarak Govind Parthiban, Yongyi Mao, and Diana Inkpen. 2021. On the softmax bottleneck of recurrent language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13640–13647.

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. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q. Tran, Dani Yogatama, and Donald Metzler. 2022. Scaling laws vs model architectures: How does inductive bias influence scaling?

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Haohan Wang and Bhiksha Raj. 2017. On the origin of deep learning on the origin of deep learning.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2018. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*.

Zhilin Yang, Thang Luong, Ruslan Salakhutdinov, and Quoc Le. 2019. Mixtape: Breaking the softmax bottleneck efficiently. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

Appendix A

Proofs

Theorem A.1. For all $\mathbf{z} \in \mathbb{R}^K$, $\text{softmax}(\mathbf{z}) \in \Delta^{K-1}$.

Proof. Let $\mathbf{z} \in \mathbb{R}^K$. We prove two conditions of the set in definition 2.2:

- Since $\exp(a) \geq 0$ for all $a \in \mathbb{R}$, $\frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})} \geq 0$ for all $k = 1, \ldots, K$.
- $\sum_{k=1}^{K} \text{softmax}(\mathbf{z})_k = \sum_{k=1}^{K} \frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})} = \frac{\sum_{k=1}^{K} \exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})} = 1.$

As both $\forall_{k=1}^{K} \text{softmax}(\mathbf{z})_k \geq 0$ and $\sum_{k=1}^{K} \text{softmax}(\mathbf{z})_k = 1$, $\text{softmax}(\mathbf{z}) \in \Delta^{K-1}$. □

Theorem A.2. For all $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, $\quad \text{rank}(\mathbf{A} + \mathbf{B}) \leq \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B})$.

Proof. Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, i.e. $\mathbf{A}$ and $\mathbf{B}$ are matrices of linear transformations T_A and T_B respectively. If we take any $\mathbf{v} \in \text{img}(\mathrm{T}_A + \mathrm{T}_B)$, it means there exists some $\mathbf{x}$ such that $(\mathrm{T}_A + \mathrm{T}_B)(\mathbf{x}) = \mathbf{v}$. By definition of linear operator, $(\mathrm{T}_A + \mathrm{T}_B)(\mathbf{x}) = \mathrm{T}_A(\mathbf{x}) + \mathrm{T}_B(\mathbf{x})$, so $\mathbf{v} \in \text{img}(\mathrm{T}_A) + \text{img}(\mathrm{T}_B)$. Thus, $\text{img}(\mathrm{T}_A + \mathrm{T}_B) \subseteq \text{img}(\mathrm{T}_A) + \text{img}(\mathrm{T}_B)$, and the rank is

$$\begin{aligned} \text{rank}(\mathbf{A} + \mathbf{B}) &:= \dim(\text{img}(\mathrm{T}_A + \mathrm{T}_B)) \leq \dim(\text{img}(\mathrm{T}_A) + \text{img}(\mathrm{T}_B)) \\ &\leq \dim(\text{img}(\mathrm{T}_A)) + \dim(\text{img}(\mathrm{T}_B)) = \text{rank}(\mathbf{A}) + \text{rank}(\mathbf{B}). \end{aligned}$$

□

Appendix B

Additional results from experiments

B.1 Handwritten digit recognition

d	model	train loss	test loss	train accuracy	test accuracy
128	softmax	0.0 ± 0.0	0.03 ± 0.0	99.96 ± 0.07	99.37 ± 0.05
128	sigsoftmax	$\mathbf{0.0 \pm 0.0}$	0.03 ± 0.0	$\mathbf{99.96 \pm 0.05}$	99.37 ± 0.06
128	plif	0.0 ± 0.0	0.03 ± 0.0	$\mathbf{99.96 \pm 0.05}$	99.38 ± 0.04
128	mos	0.0 ± 0.0	0.03 ± 0.0	99.86 ± 0.05	99.44 ± 0.05
128	moss	0.0 ± 0.0	$\mathbf{0.03 \pm 0.0}$	99.93 ± 0.06	$\mathbf{99.48 \pm 0.04}$
9	softmax	0.0 ± 0.0	0.09 ± 0.04	99.91 ± 0.09	98.53 ± 0.71
9	sigsoftmax	0.0 ± 0.0	0.09 ± 0.03	$\mathbf{99.94 \pm 0.1}$	98.68 ± 0.34
9	plif	0.0 ± 0.0	0.08 ± 0.02	99.89 ± 0.11	98.6 ± 0.31
9	mos	0.0 ± 0.0	0.05 ± 0.01	99.87 ± 0.05	$\mathbf{99.05 \pm 0.14}$
9	moss	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.05 \pm 0.01}$	99.92 ± 0.04	99.02 ± 0.09
8	softmax	0.0 ± 0.0	0.08 ± 0.01	99.84 ± 0.13	98.59 ± 0.2
8	sigsoftmax	0.0 ± 0.01	0.14 ± 0.16	$\mathbf{99.92 \pm 0.21}$	98.52 ± 0.65
8	plif	$\mathbf{0.0 \pm 0.0}$	0.08 ± 0.01	99.9 ± 0.12	98.52 ± 0.28
8	mos	0.01 ± 0.0	$\mathbf{0.05 \pm 0.0}$	99.81 ± 0.09	98.89 ± 0.14
8	moss	0.0 ± 0.0	0.06 ± 0.01	99.88 ± 0.09	$\mathbf{98.94 \pm 0.15}$
7	softmax	$\mathbf{0.0 \pm 0.0}$	0.09 ± 0.02	$\mathbf{99.9 \pm 0.06}$	98.52 ± 0.21
7	sigsoftmax	0.01 ± 0.01	0.11 ± 0.04	99.89 ± 0.23	98.34 ± 0.6
7	plif	0.02 ± 0.04	0.13 ± 0.07	99.33 ± 1.07	97.42 ± 1.84
7	mos	0.01 ± 0.0	$\mathbf{0.06 \pm 0.01}$	99.78 ± 0.07	98.74 ± 0.3
7	moss	0.01 ± 0.0	0.06 ± 0.02	99.88 ± 0.09	$\mathbf{98.8 \pm 0.24}$
6	softmax	$\mathbf{0.0 \pm 0.0}$	0.1 ± 0.02	$\mathbf{99.89 \pm 0.05}$	98.34 ± 0.35
6	sigsoftmax	0.01 ± 0.01	0.13 ± 0.08	99.88 ± 0.15	98.17 ± 0.65
6	plif	0.24 ± 0.69	0.33 ± 0.66	90.88 ± 26.56	89.24 ± 25.99
6	mos	0.01 ± 0.0	$\mathbf{0.06 \pm 0.01}$	99.74 ± 0.08	98.56 ± 0.09
6	moss	0.01 ± 0.0	0.1 ± 0.08	99.88 ± 0.07	$\mathbf{98.57 \pm 0.28}$

Table B.1: Cross entropy and accuracy for $d = 6$ to 9, and 128. Reported values are means and standard deviations based on 10 different random seeds.

d	model	train loss	test loss	train accuracy	test accuracy
5	softmax	0.32 ± 0.57	0.44 ± 0.56	86.67 ± 23.7	84.84 ± 23.23
5	sigsoftmax	0.25 ± 0.41	0.45 ± 0.42	89.95 ± 17.05	87.36 ± 17.41
5	plif	0.12 ± 0.3	0.24 ± 0.29	95.2 ± 13.0	92.92 ± 12.95
5	mos	0.04 ± 0.05	$\mathbf{0.1 \pm 0.06}$	99.34 ± 0.99	97.82 ± 1.68
5	moss	$\mathbf{0.01 \pm 0.01}$	0.12 ± 0.05	$\mathbf{99.76 \pm 0.32}$	$\mathbf{98.3 \pm 0.52}$
4	softmax	0.3 ± 0.5	0.48 ± 0.49	87.75 ± 20.77	85.09 ± 20.68
4	sigsoftmax	0.28 ± 0.42	0.46 ± 0.45	88.89 ± 17.79	86.35 ± 18.48
4	plif	0.4 ± 0.7	0.52 ± 0.66	84.91 ± 27.72	82.35 ± 26.98
4	mos	0.02 ± 0.01	$\mathbf{0.09 \pm 0.02}$	99.47 ± 0.32	$\mathbf{97.87 \pm 0.5}$
4	moss	$\mathbf{0.02 \pm 0.01}$	0.11 ± 0.03	$\mathbf{99.61 \pm 0.25}$	$\mathbf{97.87 \pm 0.53}$
3	softmax	0.73 ± 0.89	0.89 ± 0.81	71.81 ± 35.13	69.58 ± 33.69
3	sigsoftmax	0.75 ± 0.89	0.95 ± 0.8	71.08 ± 35.24	68.95 ± 34.12
3	plif	0.5 ± 0.73	0.6 ± 0.69	80.75 ± 28.98	78.56 ± 28.14
3	mos	0.11 ± 0.16	$\mathbf{0.2 \pm 0.14}$	96.95 ± 3.98	95.07 ± 3.58
3	moss	$\mathbf{0.04 \pm 0.03}$	0.26 ± 0.22	$\mathbf{99.03 \pm 0.72}$	$\mathbf{96.78 \pm 0.65}$
2	softmax	1.04 ± 0.79	1.2 ± 0.71	58.57 ± 30.89	56.46 ± 29.57
2	sigsoftmax	1.4 ± 0.81	1.53 ± 0.71	44.4 ± 30.88	42.77 ± 29.5
2	plif	0.98 ± 0.76	1.08 ± 0.71	60.72 ± 29.92	58.35 ± 28.64
2	mos	$\mathbf{0.2 \pm 0.09}$	$\mathbf{0.32 \pm 0.11}$	$\mathbf{91.49 \pm 5.66}$	$\mathbf{88.97 \pm 5.77}$
2	moss	0.23 ± 0.1	0.44 ± 0.08	90.96 ± 6.2	88.53 ± 6.24
1	softmax	1.64 ± 0.54	1.78 ± 0.43	35.05 ± 19.75	33.79 ± 18.6
1	sigsoftmax	1.59 ± 0.59	1.74 ± 0.47	36.97 ± 21.33	35.4 ± 19.91
1	plif	1.69 ± 0.62	1.75 ± 0.56	32.97 ± 22.2	32.13 ± 21.16
1	mos	1.44 ± 0.3	1.47 ± 0.29	41.68 ± 11.21	41.2 ± 11.0
1	moss	$\mathbf{1.23 \pm 0.38}$	$\mathbf{1.3 \pm 0.36}$	$\mathbf{49.63 \pm 14.03}$	$\mathbf{48.22 \pm 13.55}$

Table B.2: Cross entropy and accuracy for $d = 1$ to 5. Reported values are means and standard deviations based on 10 different random seeds. In bold, we present the best means for each d.

B.2 Species classification

d	model	train loss	test loss	train accuracy	test accuracy
32	softmax	0.07 ± 0.08	0.62 ± 0.07	99.34 ± 1.12	86.2 ± 0.5
32	sigsoftmax	$\mathbf{0.02 \pm 0.02}$	0.72 ± 0.11	$\mathbf{99.99 \pm 0.03}$	85.87 ± 0.69
32	plif	0.06 ± 0.06	0.62 ± 0.11	99.49 ± 0.88	85.76 ± 0.62
32	mos	0.06 ± 0.05	$\mathbf{0.6 \pm 0.07}$	99.01 ± 0.51	$\mathbf{86.68 \pm 0.52}$
32	moss	0.06 ± 0.05	0.68 ± 0.13	99.0 ± 0.62	86.39 ± 0.64
16	softmax	0.18 ± 0.13	0.86 ± 0.18	97.17 ± 2.77	81.8 ± 0.91
16	sigsoftmax	0.22 ± 0.1	0.92 ± 0.23	96.4 ± 2.22	80.86 ± 0.9
16	plif	0.27 ± 0.09	$\mathbf{0.78 \pm 0.02}$	94.66 ± 2.33	80.88 ± 0.85
16	mos	0.15 ± 0.08	0.83 ± 0.11	96.74 ± 0.94	$\mathbf{82.03 \pm 0.57}$
16	moss	$\mathbf{0.13 \pm 0.07}$	0.96 ± 0.15	$\mathbf{97.19 \pm 0.98}$	81.92 ± 0.52
8	softmax	0.59 ± 0.12	$\mathbf{1.23 \pm 0.07}$	86.04 ± 3.12	71.6 ± 1.03
8	sigsoftmax	0.58 ± 0.13	1.45 ± 0.1	86.77 ± 3.54	69.72 ± 0.95
8	plif	0.64 ± 0.06	1.24 ± 0.06	84.12 ± 1.7	70.6 ± 1.01
8	mos	$\mathbf{0.48 \pm 0.19}$	1.26 ± 0.09	$\mathbf{90.08 \pm 2.66}$	$\mathbf{73.79 \pm 1.39}$
8	moss	0.55 ± 0.03	1.37 ± 0.05	90.05 ± 1.0	72.65 ± 1.1
4	softmax	1.43 ± 0.06	2.34 ± 0.12	59.28 ± 1.73	45.79 ± 1.75
4	sigsoftmax	1.49 ± 0.08	2.64 ± 0.18	58.05 ± 2.12	44.59 ± 2.2
4	plif	1.27 ± 0.12	2.44 ± 0.14	64.09 ± 3.42	45.49 ± 1.6
4	mos	$\mathbf{0.95 \pm 0.08}$	$\mathbf{2.29 \pm 0.11}$	$\mathbf{81.55 \pm 2.28}$	$\mathbf{50.78 \pm 1.19}$
4	moss	1.4 ± 0.11	2.69 ± 0.18	70.63 ± 3.94	46.12 ± 1.51
2	softmax	3.07 ± 0.05	3.69 ± 0.11	13.59 ± 0.7	11.37 ± 0.89
2	sigsoftmax	3.03 ± 0.05	3.82 ± 0.16	13.69 ± 0.73	11.46 ± 0.86
2	plif	2.92 ± 0.07	3.65 ± 0.11	16.04 ± 1.03	$\mathbf{13.39 \pm 0.84}$
2	mos	2.89 ± 0.06	$\mathbf{3.52 \pm 0.06}$	20.71 ± 1.25	12.61 ± 0.84
2	moss	$\mathbf{2.86 \pm 0.03}$	3.9 ± 0.35	$\mathbf{21.78 \pm 1.16}$	12.64 ± 0.97
1	softmax	3.95 ± 0.04	4.24 ± 0.05	3.44 ± 0.25	3.76 ± 0.34
1	sigsoftmax	3.94 ± 0.06	4.26 ± 0.07	3.55 ± 0.31	3.8 ± 0.36
1	plif	3.91 ± 0.08	4.26 ± 0.06	3.55 ± 0.4	3.73 ± 0.18
1	mos	3.88 ± 0.04	$\mathbf{4.12 \pm 0.04}$	4.46 ± 0.47	4.01 ± 0.37
1	moss	$\mathbf{3.86 \pm 0.04}$	4.17 ± 0.09	$\mathbf{4.54 \pm 0.53}$	$\mathbf{4.03 \pm 0.38}$

Table B.3: Cross entropy and accuracy for various d. Reported values are means and standard deviations based on 10 different random seeds. In bold, we present the best means for each d.