Reproducing "Breaking the Softmax Bottleneck: A High-Rank RNN Language Model"

Jae Hyun Lim, Salem Lahlou, Massimo Caccia
MILA, University of Montreal
Montreal, QC H3T1J4
{lim0606, salemlahlou9, massimo.p.caccia}@gmail.com

Abstract

In this report,"Breaking the Softmax Bottleneck: A High-Rank RNN Language Model [4]" is reviewed. The paper proposes a new method to increase the rank of the joint probability matrix in Language Modeling that approximates the true (inaccessible) one, and shows two categories of experiments: language modeling on benchmark datasets and rank analysis of those matrices. This report primarily aims at duplicating the results of those two experiments conducted in the paper. Based on our analysis, we conclude that our experiments also support the increments in ranks along with the proposed method. However, we also found that the rank estimation methods in the paper should be further reviewed, and we claim that the ranks in the paper were reported using a potentially erroneous implementation.

1 Paper Summary

1.1 Motivation

Language modeling aims at estimating the density function of a sequence of tokens (words). Most successful approaches rely on an auto-regressive factorization of the joint probability of a list of tokens / sentences : Given a corpus $(X_1, \dots, X_T)$, we model the probability using the factorization $P(X) = \prod_t P(X_t|X_{<t})$ where $X_{<t}$ is usually denoted by C_t (the context).

The standard traditional approach is to use an RNN to encode the context into a hidden state vector, which is then multiplied by the word embeddings to obtain the logits. The logits consumed by Softmax give the probability distribution of the next token.

The paper tries to answer the question of whether the combination of dot product and Softmax is capable of modeling the conditional probability correctly.

1.2 Proposed approach

We want to learn a model distribution $P_\theta(X|C)$ that approximates the true data distribution (that depends only on the language) $P^*(X|C)$. Checking the capability of Softmax/dot product to correctly model the conditional probability boils down to answering the question of whether we can have a θ, such that $P_\theta(X|c) = P^*(X|c)$ for all contexts c.

In a traditional RNN-based model, $P_\theta(x|c) = \frac{exp(h_c^T w_x)}{\sum_{x'} exp(h_c^T w_{x'})}$, Where the context vector h_c and the word embedding w_x have the same dimension d. The logit is the dot product $h_c^T w_x$.

The authors show that $P_\theta(X|c) = P^*(X|c)$ is equivalent to the product of the matrices H_θ and W_θ being a matrix of rank 1 away from the matrix $A = (log(P^*(x_j|c_i)))_{i,j}$ (here, H_θ and W_θ are representations of all contexts and embeddings of all words, obtained by concatenating the previously

Submitted as course project for IFT6135 H2018 Representation Learning

defined vectors). Their main result is that if $d < rank(A) - 1$ then there exists a contexts c for which we don't have $P_\theta(X|c) = P^*(X|c)$. Proving this only requires basic linear algebra techniques.

They finish this preliminary analysis by hypothesizing that Natural language is high rank (as in: the previous matrix A is of high-rank). The rank of A can be as high as the number of tokens M (in natural language it's around 10^5). They conclude that Softmax (when using a small d) is effectively learning a low rank approximation to A, thus creating a **bottleneck**.

To fix this, the paper proposes then a **Mixture Of Softmaxes (MoS)** strategy as a way to increase expressiveness without exploding the parameter space (given that most methods that have increased expressiveness, such as N-gram models with Kneser & Ney's back-off [5], increase dramatically the number of parameters).

In MoS, instead of training one RNN model to produce the context vectors h_c, we train K RNNs to produce K different context vectors $h_{c,1}, \ldots, h_{c,K}$ for all contexts c. The model distribution is defined as $P_\theta(x|c) = \sum_{k=1}^{K} \pi_{c,k} \frac{exp(h_{c,k}^T w_x)}{\sum_{x'} exp(h_{c,k}^T w_{x'})}$, where $\sum_k \pi_{c,k} = 1$, with $\pi_{c,k}$ bein the prior or mixture weight of the k-th component, for the context c. MoS has K components, each represented by an RNN.

The authors argue that the suggested method unblocks the bottleneck imposed by Softmax: MoS has an improved expressiveness because:

- When $K = 1$, it reduces to Softmax
- In the more general case, it approximates A using $\hat{A} = log \sum_k \Pi_k exp(H_{\theta,k} W_\theta^T)$, where Π_k is $N \times N$ diagonal with elements $(\pi_{c_t,k})_{t=1,\ldots,N}$ (N being the number of possible contexts).
- As $\hat{A}$ is a non linear function of the context vectors and the word embeddings, the authors mention it can be **arbitrarily** high rank with sufficient number of mixtures.

The authors propose a baseline to compare the MoS models to, which they call a **Mixture of Contexts (MoC**, which differs from MoC in that it mixes the context vectors using the prior before applying Softmax: $P_\theta(x|c) = \frac{exp(\sum_{k=1}^{K} \pi_{c,k} h_{c,k}^T w_x)}{\sum_{x'} exp(\sum_{k=1}^{K} \pi_{c,k} h_{c,k}^T w_{x'})}$. Obviously, this has the same limitations as Softmax (e.g. use $h'(c) = \sum_{k=1}^{K} \pi_{c,k} h_{c,k}$), hence, performing mixture in the feature space does not change the fact that the matrix product is upper bounded by the embedding dimension. MoC is used in the experiments as a comparative baseline only because it has the same parameters as MoS (given the same number of experts K), and it wouldn't be fair to compare MoS to Softmax otherwise.

2 Experiments

2.1 Motivation for the experiments

The paper [4] provides two categories of experiments in order to support the suggested method:

- Estimating the log-likelihoods on benchmark datasets, such as Penn Treebank (PTB) and WikiText-2 (WT2) corpus
- Estimating the ranks of the log-probability matrices of data.

They compared the results computed via different types of models, including the previous state-of-the-art, Mixture of Contexts models (MoC), Mixture of Softmaxes models (MoS), and other conventional language models.

First, it is straightforward to conduct the first class of experiments in order to demonstrate that the suggested method improves the likelihoods. The goal of language models is to estimate joint probability of all tokens in data, which is represented as auto-regressive factorizations. As the suggested method is expected to improve the ranks of the log-probability matrices of the models (which provide better approximations of the true distribution matrices), the results of the experiments for log-probabilities should be improved by the suggested method.

Second, the main argument of the paper is that MoS increases the rank of the log probability matrices, and thus the authors conducted experiments to estimate the ranks of the matrices with different types

models and demonstrated that the ranks of MoS models are much higher compared to conventional settings, and compared to MoC, even with a similar number of parameters (to make the comparison fair).

In following sections, we report the results of our reproduction of the aforementioned experiments:

- likelihood estimation on the PTB and WT2 datasets
- rank analysis on the PTB dataset

Note that the joint probabilities of likelihoods are represented as perplexities in the following sections.

2.2 Reproducing the main results

Firsly, we analyze the likelihood estimation performance of the models on PTB and WT2 datasets.

Perplexities on the test set are illustrated in Table 1. No dynamic evaluation and no finetuning was done because our main goal is rank estimation. The models were launched to train up to 1000 epochs. However, an early stopping mechanism that monitors validation error was used, as it is standard in most NLP tasks. Early stopping turned out to be crucial in all our experiments, as illustrated in Figure 1. For example, for PTB trained with MoS with 1 experts, i.e. a standard softmax output module, the model starts overfitting as early as 300 epochs.

Our results are on par with the reported numbers of the paper. For example, the reported perplexity for MoC and MoS on PTB was 57.55 and 55.97, and we obtained 57.79 and 56.07. There is however a small discrepancy for the WT2 experiment. The reported numbers are 65.98 and 63.33 for MoC and MoS respectively. We however found 68.11 and 64.68. But, we didn't run MoS 20 on WT2 because it didn't fit on a single GPU. So we have to give them the benefit of the doubt. Also, hyperparmameter tuning could be at cause here.

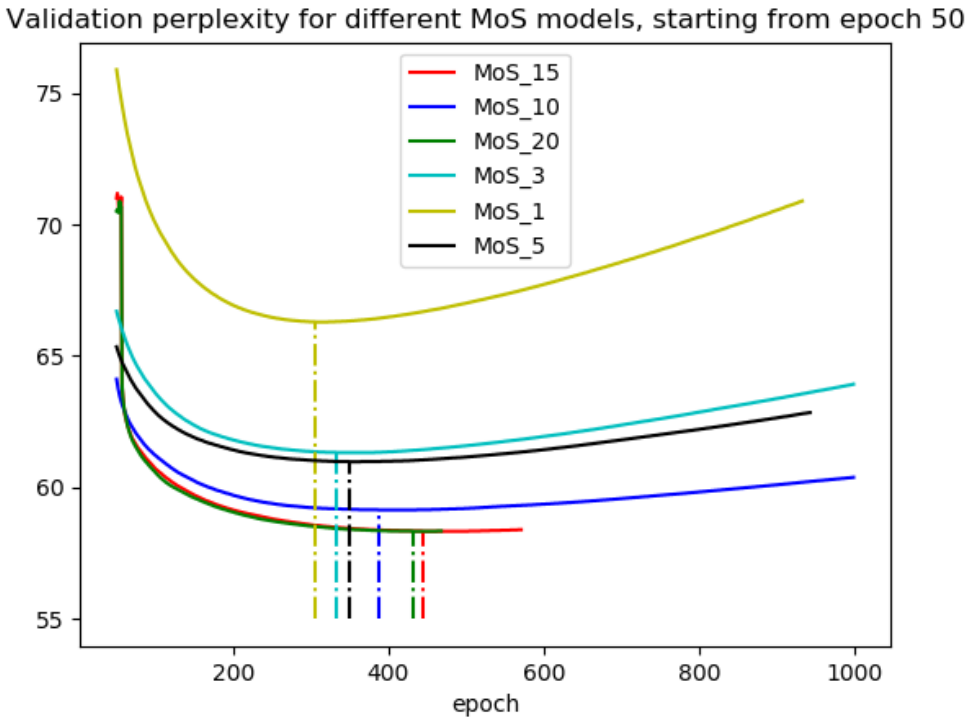

Figure 1: Training curve for Penn Treebank corpus

2.3 Rank Analysis

The main hypothesis of the paper is that MoS increases the rank of the log probabilities matrix of the true data distribution. As a result, the author conducted a series of experiments to approximate the ranks of the trained models. In this section, we report the results of reproducing rank analysis experiments for our experiments on the PTB dataset.

In order to estimate the ranks of the log probabilities matrices (on the test corpus), we used both a conventional SVD algorithm [2] [1] and a randomized SVD algorithm [1] [2]. We confirmed that both libraries provided approximately the same results. It is important to mention that in all SVD experiments, **numerical errors due to rounding of floating-point** number systems should be taken into account. As a result, we came out with two different methods, each of which is indicated as "rank1" and "rank2" in the results. The detailed discussions of the two different rank estimations will follow at the end of this section. Table 1 illustrates the rank analysis results.

In general, we found that our experiments support the hypothesis suggested by the authors. More specifically, the results demonstrated that the ranks of MoC matrices are consistent in both "rank1" and "rank2" regardless of the number of mixtures. On the other hand, the ranks of MoS matrices increase as the number of mixtures increases. Moreover, the ranks of MoS experiments are higher than the one of MoC experiments regardless of rank estimation methods. Note that all MoC experiments are equivalent to a Softmax on a rank basis. Thus the rank should always be 280, in theory.

Table 1: Results

		MoC				MoS			
	experts	param.	perp.	rank1*	rank2**	param.	perp.	rank1*	rank2**
PTB	1	19,057,620	65.34	282	282	19,057,620	64.75	**2884**‡	280
	3	19,406,620	62.63	282	282	19,406,620	59.13	**6097**	645
	5	19,755,620	60.07	282	282	19,755,620	58.66	**8375**	924
	10	20,628,120	58.98	282	282	20,628,120	56.92	**9957**	1274
	15	21,500,620	57.49	282	282	21,500,620	56.07	**9980**	1445
	20	22,373,120	57.79	282	282	22,373,120	56.07	**9981**	1625
WT2†	1	32,166,228	70.63	–	–	32,166,228	71.32	–	–
	3	32,558,128	68.78	–	–	32,558,128	66.37	–	–
	5	32,950,028	68.56	–	–	32,950,028	65.48	–	–
	10	33,929,778	68.67	–	–	33,929,778	65.00	–	–
	15	34,909,528	68.11	–	–	34,909,528	64.68	–	–
	20	35,889,278	68.19	–	–	35,889,278	64.82	–	–

Number of parameters for each model trained, along with the test perplexity obtained with an early-stopped model (using validation perplexity as a metric). We also show the ranks of the approximating matrices using the two different thresholds discussed in this section.

*: rank estimated with the expected roundoff error bound as in [3].

**: rank estimated with the empirical bound as shown in Figure 2.

† The number of epochs for training was 150, instead of 1000.

‡ Note that MoS = 1 is supposed to have a rank of 280. For MoS > 1, the estimated ranks approximately match the reported results in the paper [4].

Reproducing the rank analysis involved two major issues:

- requirement of huge computation resources for SVD of large matrices
- numerical errors in SVD results due to errors in floating-point arithmetics

The first major obstacle in the rank analysis was the necessity of huge computation resources in SVD computation. The randomized SVD algorithm has a computational complexity for a given matrix $A \in \mathbb{R}^{N \times M}$ and a target truncated rank K equal to $T = O(K^2(M+N))$ [1]. Note that conventional SVD algorithms have larger computational complexity than the randomized one.

The test corpus of Penn Treebank (PTB) dataset has 10,000 words, and the number of tokens in the dataset is 78,669; thus, the log probability matrix A is in $\mathbb{R}^{N \times M}$, with $N = 78,669$ and

[1] https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.linalg.svd.html

[2] https://github.com/facebook/fbpca

$M = 10,000$. As a result, the corresponding computational complexity required to estimate the rank is $T_{PTB} = O(9 \times 10^{12})$ if $K = M = 10,000$. In our experiments, we used a CPU cluster, in which 36 CPUs are connected and each CPU has 64 Gb memory, which is the largest configuration currently available for us.

The rank analysis of language models on the WikiText-2 (WT2) corpus couldn't be conducted because of the limitations of the computational resources. The size of vocabulary in WikiText-2 is 33,728, and the numbers of tokens of valid and test corpus in WikiText-2 are 217,646 and 245,569, respectively. With $K = M = 33,728$, the computational complexities for the rank estimations would be $T_{WT2} = O(2.25 \times 10^{14})$. This requires a cluster that is 25 times larger, which was not available to us. We thought of estimating the rank of the matrix evaluated on a subset of the test corpus, but that was still impossible: as the number of contexts should include at least one occurrence of each word, the number of contexts is supposed to be larger than the vocabulary size, i.e. $N > M = K$. In this case, the lower-bound on the complexity would be $T_{WT2,subset} = O(3.6 \times 10^{13})$. This still requires 4 times larger memory than the one for PTB, which is again infeasible.

The second major issue for reproducing the rank analysis experiments was understanding the numerical errors in the estimated eigenvalues of SVD. In the computation of SVD, and in order to estimate the eigenvalues with floating-point arithmetics, we have to take care of rounding errors in the floating-point number system, and the effect of theses errors is referred to in the paper of interest [4]. For example, imagine a matrix $A \in \mathbb{R}^{N \times M}$ filled with constant value c. Even though A is of rank 1 (one eigenvalue, and the corresponding eigenspace is of dimension 1), A can be estimated to be full-rank because of the errors stemming from rounding errors. However, the errors in our estimations of the eigenvalues that are due to the rounding errors should be smaller than those due to actual variations in data. For numerical errors in SVD, Press et al. [3] suggested the **expected roundoff error** as a threshold below which the estimated eigenvalues can be considered as being zero.The bound λ_{noise}, is defined as follows;

$$\lambda_{noise} = 0.5 \times \sqrt{M + N + 1} \times \lambda_{\max} \times \epsilon \tag{1}$$

where $\lambda_{\max}$ is the largest eigenvalue of a given matrix with size $N \times M$, and ϵ is the machine epsilon (or rounding error) of the floating-point number system. In practice, $\epsilon = 1.19209 \times 10^{-7}$. For the PTB dataset, $\lambda_{noise} \approx 6.5$. We used the expected roundoff error as a bound to estimate the ranks of the matrices of log-probabilities, and treated **the eigenvalues smaller than the bound as zero**. **The ranks estimated with this bound are indicated as "rank1"** in Table 1.

Using the expected roundoff error resulted in the ranks of all MoC are to be very close to 280 (the theoretical rank - it's the dimension of the word embeddings). Moreover, **the error bound-based rank of MoS with a single mixture is 2884, which much higher than 280**. Hypothetically, MoS with 1 expert and MoC with 1 expert are supposed to be the same model, and their ranks are supposed to be the same as well. **This implies the rank analysis results described in the paper [4] could potentially be wrongly reported, as the results of MoS (with ≥ 3 experts)are consistent with the ones in the paper.**

Regarding this, we hypothesize that the current implementation of MoS potentially increases rounding error effects, and that is why it is resulting in a log-probability matrix for (MoS with 1 expert) of a rank larger than 280. Our best bet is that it is because the current code includes a treatment to handle underflow in the logarithmic scale, whose input is a multiplication of a mixture probability and softmaxes; MoC doesn't need this treatment as a single softmax is estimated with log-softmax function. We believe that it is because of the conventional treatment (i.e. adding a small number such as 10^{-8} in the current MoS implementation) that there are amplifications in the roundoff errors, that resulted in MoS with 1 expert having a much higher rank than the one we were expecting (280). As a result, and in order to take into account the authors' handling of underflow in the logarthmic scale, we concluded that the the threshold of the error-driven eigenvalues should be larger than the one described in [3]. We thus set this threshold as 281-th eigenvalue of MoS with 1 expert, we will refer to it hereinafter as the **empirical bound**.The ranks estimated with the empirical bound indicated as "rank2" in Table 1.

The disparity between the theoretical bound of the error-driven eigenvalues and the empirical ones are shown in Figure 2. Note that the ranks reported in the paper [4] use what we refer to as 'rank1'; however, the 'rank1' for the MoS with 1 expert is higher accordingly. Too bad they did not report it in the paper.

Based on our analysis, we conclude that the rank estimation methods in the paper should be **further reviewed**. It is possible to claim that the reported ranks for MoS in the original paper were higher than reality, which we claim is what we refer to as the empirical rank ("rank2"). On the contrary, the hypothesis itself, a.k.a. advancements of ranks via MoS, is still supported even with the empirical rank ("rank2"), as the increments in ranks along with the increments in the number of mixtures are clear.

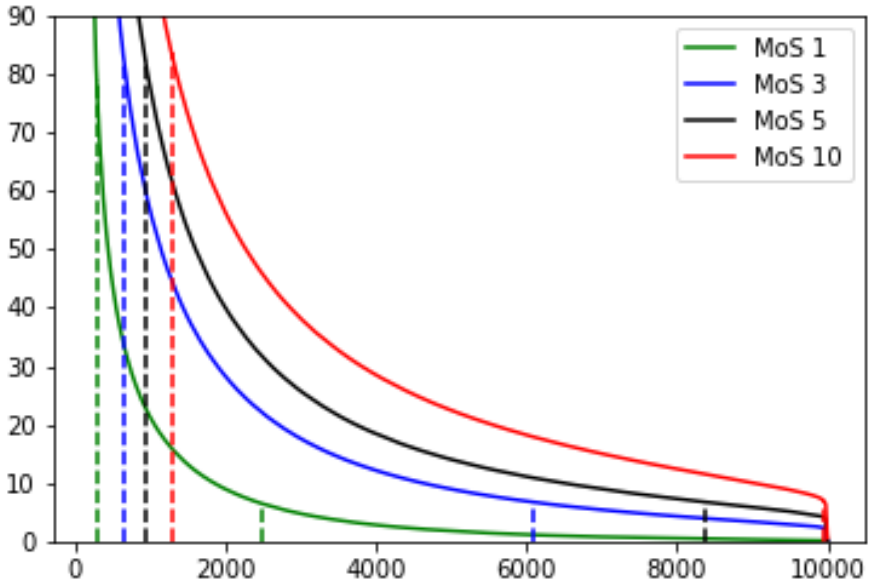

Figure 2: Eigenvalue curves on PTB corpus in decreasing. Rank 1 and rank 2 are depicted by the dotted lines. As discussed, ranks 2 are much higher than ranks 1. Futher investigation is required.

3 Discussion - Conclusion

We believe the paper is extremely well written, and easy to follow. The authors provided mathematical proofs for why their suggested method works better than previous methods, which is always very enjoyable. The experiments they conducted are all relevant given that they not only show that their method achieves state of the art results on usual datasets, but also support the hypothesis for why it is the case (the increasing rank).

Nonetheless, we strongly believe that, to the contrary of what the authors claim, the bottleneck is not due to the Softmax function, but rather because of the simple fact that the product of two matrices has a rank that is upper-bounded by the lowest rank of the two matrices. We prefer to call it the dot-product bottleneck, but maybe "Softmax bottleneck" is more catchy.

Finally, when trying to reproduce the rank analysis they conducted in the original paper, we stumbled upon what we think is a major flow in their rank-evaluating algorithm. Their handling of underflow made the ranks they reported higher than what they are in reality as discussed in **2.3**. This observation is motivated by our study of MoS with 1 expert, for which we obtain a much higher rank than the one we should have in reality (the embedding size: 280). We wish the authors took the time to estimate the rank for MoS with 1 expert, as this would have probably made them question their rank analysis algorithm. We are highly confident about this conclusion given that we are able to obtain the same rank (approximately) as they did for MoS with 3, 5, 10, 15 and 20 experts. To tackle this, we chose another threshold for the eigenvalues (the value below which the evaluated eigenvalues should be considered as equal to zero), by setting it to the 281-st eigenvalue of MoS with 1 expert *with the underflow handling*. By definition, this threshold, that we referred to as the empirical bound, would

yield the correct rank for MoS with 1 expert (i.e. 280). However, it resulted in ranks lower than the ones they reported in the original paper for MoS with 3, 5, 10, 15 and 20 experts. This means that by using 15 experts, we are still far from what they call "full rank", *undermining* their claim in section 3.3 that "further increasing the number of components after we reach full rank degrades the performance due to overfitting, especially that the perplexities we report (without dynamic evaluation and finetuning) don't really confirm this overfitting hypothesis. We agree with the authors however in their main argument that increasing the number of experts increases the rank.

Acknowledgments

This report was enabled in part by support provided by Calcul Québec (www.calculquebec.ca) and Compute Canada (www.computecanada.ca).

References

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

[2] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[3] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[4] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. *CoRR*, abs/1711.03953, 2017.

[5] Hui Zhang and David Chiang. Kneser-ney smoothing on expected counts. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 765–774, 2014.