# Doubly Sparse: Sparse Mixture of Sparse Experts for Efficient Softmax Inference

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Computations for the softmax function in neural network models are expensive when the number of output classes is large. This can become a significant issue in both training and inference for such models. In this paper, we present Doubly Sparse Softmax (DS-Softmax), *Sparse Mixture of Sparse of Sparse Experts*, to improve the efficiency for softmax inference. During training, our method learns a two-level class hierarchy by dividing entire output class space into several partially overlapping experts. Each expert is responsible for a learned subset of the output class space and each output class only belongs to a small number of those experts. During inference, our method quickly locates the most probable expert to compute small-scale softmax. Our method is learning-based and requires no knowledge of the output class partition space a priori. We empirically evaluate our method on several real-world tasks and demonstrate that we can achieve significant computation reductions without loss of performance.

## 1 Introduction

Deep learning models have demonstrated impressive performance in many classification problems (LeCun et al., 2015). In many of these models, softmax function/layer is commonly used to produce categorical distributions over the output space. Due to its linear complexity, computation for softmax layer can become a bottleneck with large output dimensions, such as language modelling (Bengio et al., 2003), neural machine translation (Bahdanau et al., 2014) and face recognition (Sun et al., 2014). In some models, softmax contributes to more than 95% computation. This becomes more of an issue when computational resource is limited, like mobile devices (Howard et al., 2017). Many methods have been proposed to reduce softmax complexity for both training and inference phases. In terms of inference, our goal is not to computing the exact categorical distribution over the whole vocabulary, but rather to search for top-K classes accurately and efficiently.

Our work aims to improve the inference efficiency of the softmax layer. We propose a novel Doubly Sparse softmax (DS-Softmax) layer. The proposed method is motivated by (Shazeer et al., 2017), and it learns a two-level overlapping hierarchy using *sparse* mixture of *sparse* experts. Each expert is trained to only contain a small subset of entire output class space, while each class is permitted to belong to more than one expert. Given a set of experts and an input vector, the DS-Softmax first selects the top expert that is most related to the input (in contrast to a dense mixture of experts), and then the chosen expert could return a scored list of most probable classes in it sparse subset. This method can reduce the linear complexity in original softmax significantly since it does not need to consider the whole vocabulary.

We conduct experiments in different real tasks, ranging from language modeling to neural machine translation. We demonstrate our method can reduce softmax computation dramatically without loss of prediction performance. For example, we achieved more than 23x speedup in language modelling and

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

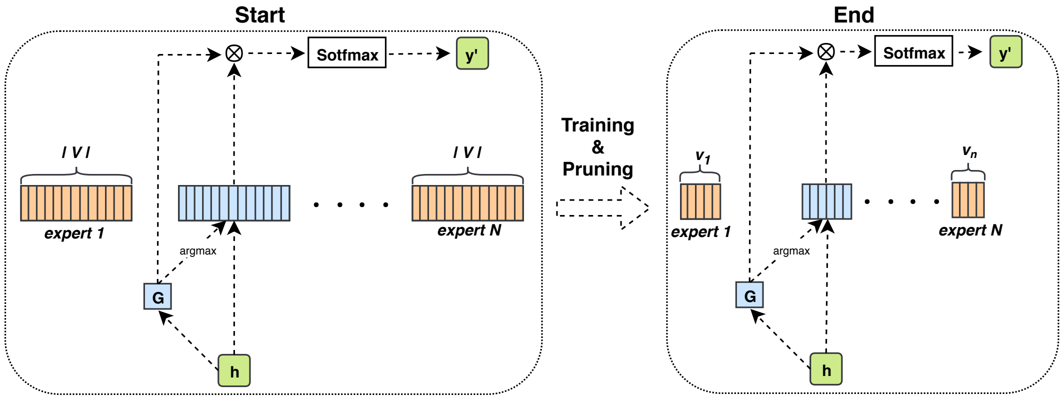


Figure 1: Overview of DS-Softmax. Initial model is similar to sparsly gating mixture of experts model. After pruning, each expert will only consists partial outputs $v_n$ instead of $|V|$.

15x speedup in translation with similar performances. In real device, our method also demonstrates similar speedup with theoretic one.

## 2 DS-Softmax: Sparse Mixture of Sparse Experts

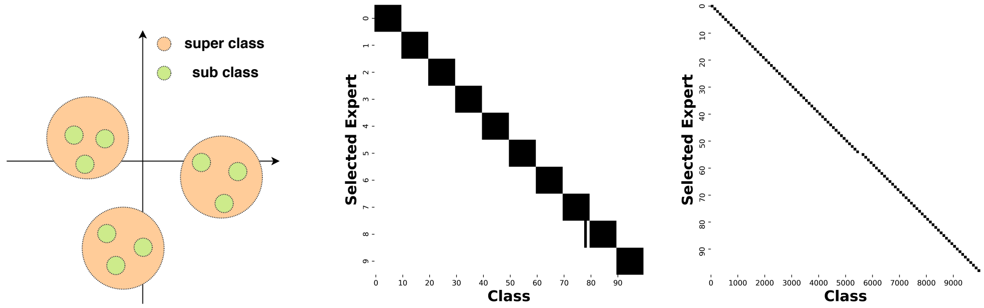
Goodman (2001) studied a two-level hierarchy for language modeling, where each word belongs to one unique cluster. (A "cluster" here refers to a cluster of words.) From this perspective, our method can be as an extension of their method to allow overlapping hierarchy. This is because, in language modeling, it is often difficult to exactly assign a word to a single cluster. For example, if we want to predict next word of "I want to eat ___" and one possible correct answer is "cookie", we can quickly notice that possible answer belongs to something eatable. So if we only search right answer inside words with the eatable property, we can dramatically increase the efficiency. On the other, though "cookie" is one of the correct answers, it might also like appear under some non-eatable context, such as "a piece of data" in computer science. Thus, a two-level overlapping hierarchy can naturally accommodate word homonyms like this by allowing each word to belong to more than one cluster. We believe this observation is likely to be true in other applications besides language modeling.

**Overview** Doubly Sparse softmax (DS-Softmax) is designed to capture such overlapped two-level hierarchy among output classes. In DS-Softmax, the first level is the *sparse* mixture and second level contains several *sparse* experts. (Here an expert can be thought as a similar concept as cluster.) The sparse mixture is to choose the right expert/cluster while sparse experts are responsible to separate full output space into multiple, overlapped and small class clusters. The design of mixture gating is inspired by Shazeer et al. (2017) but each expert in their model needs to search whole output space, while DS-Softmax only searches a small subset. This becomes much faster given large output space.

The first level of sparsification is a sparse gating mechanism inspired by Sparsely-Gated Mixture of Experts (Shazeer et al., 2017), where only partial experts are activated. For faster inference purpose, only top-one expert is chosen, which is corresponding to choose the right experts. The second level sparsification is the sparse experts, which output a categorical distribution for only a subset output classes. To sparsify each expert, we apply group lasso loss to restrain the weights inside softmax. Furthermore, the utilization of each experts is balanced with additional losses. The detail of our method can be found in Appendix.

## 3 Experiments

We evaluate the proposed method on both synthetic and real tasks. For the synthetic task, our goal is to demonstrate that our learning method could discover the hidden two-level hierarchy automatically. We also evaluate both theoretical speedup (FLOPs) and real device speedup (latency on CPU) on three different real tasks: natural language modelling, neural machine translation and Chinese handwritten




(a) Synthetic data (b) Results on 10 * 10 (c) Results on 100 * 100

Figure 2: (a) Illustration of data generation. (b) and (c) Results on discovered sparse experts on 10x10 and 100x100 datasets. The x-axis indicates class and y-axis shows the selected expert for handling this class. The order of x-axis is arranged through their super class information. For example, each 10 sub classes are belonged to one super classes in (b).

character recognition. In those real tasks, all layers except the DS-Softmax layer are pre-trained in all tasks.

### 3.1 Synthetic task

One two-level hierarchy synthetic dataset is illustrated Fig 2a. Each super class contain multiple sub classes. Two different sizes are evaluated, 10x10 (super classes x sub classes) and 100x100. The result is illustrated in Fig. 2b and Fig. 2c. We found our DS-Softmax can perfectly capture the hierarchy. For sanity check and visualization purposes, the ground-truth two hierachy in the synthetic data does not have overlappings.

### 3.2 Comparisons on FLOPs and Real Device

Real device experiments were conducted on machine with Two Intel(R) Xeon(R) CPU @ 2.20GHz and 16G memory. All tested models are re-implemented using numpy. Two configurations of SVD-Softmax Shim et al. (2017) are evaluated, SVD-5 and SVD-10. They use top 5% and 10% dimension for final evaluation in their preview window and window width is 16. Latency of each sample is shown in table 1. According to the result, our DS-Softmax can achieve not only better FLOPs speedup but also much better performance on latency.

| Task | Full | | DS-64 (Ours) | | | SVD-5 | | | SVD-10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | Value | ms | Value | FLOPs | ms | Value | FLOPs | ms | Value | FLOPs | ms |
| PTB | 0.252 | 0.73 | **0.258** | **15.99x** | **0.05** | 0.249 | 6.67x | 0.12 | 0.251 | 5.00x | 0.18 |
| Wiki-2 | 0.257 | 3.07 | **0.259** | **23.86x** | **0.12** | 0.253 | 7.35x | 0.43 | 0.255 | 5.38x | 0.63 |
| En-Ve | **25.2** | 1.91 | 25.0 | **15.08x** | **0.12** | 25.0 | 6.77x | 0.39 | 25.1 | 5.06x | 0.42 |
| CASIA | **0.906** | 1.61 | 0.901 | **6.91x** | **0.25** | 0.899 | 3.00x | 0.59 | 0.902 | 2.61x | 0.68 |

Table 1: Comparison with SVD-softmax on real device latency. The "ms" indicates the latency in microseconds. Bold fonts indicate better results.

# 4 Conclusion

In this paper, we present *doubly sparse: sparse mixture of sparse experts* for efficient softmax inference. Our method is trained end-to-end. It learns a two-level overlapping class hierarchy. Each expert is learned to be only responsible for a small subset of the output class space. During inference, our method first identifies the responsible expert and then perform a small scale softmax computation just for that expert. Our experiments on several real-world tasks have demonstrated the efficacy of our proposed method.

## References


Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Joshua Goodman. Classes for fast maximum entropy training. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pp. 561–564. IEEE, 2001.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Kyuhong Shim, Minjae Lee, Iksoo Choi, Yoonho Boo, and Wonyong Sung. Svd-softmax: Fast softmax approximation on large vocabulary neural networks. In *Advances in Neural Information Processing Systems*, pp. 5463–5473, 2017.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1891–1898, 2014.

# A Detail of Methods

**Sparse gating.** The first level of sparsification is a sparse gating mechanism inspired by Shazeer et al. (2017), which is to design to choose the right experts. The sparse gating outputs a sparse activation over a set of experts. For faster inference purpose, only the top-one expert is chosen here. One major difference comparing to Shazeer et al. (2017) is described as follows. Suppose we have $K$ experts . Given input activation vector $h \in \mathbb{R}^d$, gating values $G_k(h)$, $k = 1, ..., K$, are normalized prior to the selection as shown in Eq. 1 and then we choose the gate with the largest value $g_k = \max_i G_i(h)$ and set all other gates to be zero. Also, corresponding $k$-th expert is chosen.

$$\begin{aligned} G_k(h) &= \frac{\exp(W_k^g h)}{\sum_{k'} \exp(W_{k'}^g h)}, \\ g_k &= \begin{cases} G_k(h), & \text{if } k =_i G_i(h), \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{1}$$

This allows gradient to be back-propagated to whole $W^g$ instead of $W_k^g$ only, $W^g \in \mathbb{R}^{K \times d}$. In Shazeer et al. (2017), normalization is done after top-K experts are selected. We can not do that since we only choose top-1 expert since it will carry no gradient information since it becomes constant 1. Given the sparse gate, we compute the probability of class $c$ as,

$$O(h) = p(c|h) = \frac{\exp(\sum_k g_k W^e_{(c,k)} h)}{\sum_{c'} \exp(\sum_k g_k W^e_{(c',k)} h)}, \tag{2}$$

where $W^e_{(c,k)} \in \mathbb{R}^d$ is softmax embedding weight vector for class $c$ in expert $k$. Note that only one $g_k$ (the chosen expert) is nonzero in the formulation above. The gating values can be interpreted as an inverse temperature term for final categorical distribution produced by the chosen expert $k$ Hinton et al. (2015), shown in Eq. 2. A smaller $g_k$ gives a more uniform distribution and larger $g_k$ corresponds to a sharper one.

**Sparse experts with group lasso.** The second level sparsification is the sparse experts, which output a categorical distribution for only a subset output classes. To sparsify each expert, we apply group lasso loss to restrain the $W^e_{(c,k)}$, shown in Eq. 3. Then, pruning is carried out for $W^e_{(c,k)}$ during training with $\gamma$ is a lasso threshold according to Eq. 4.

$$\mathcal{L}_{lasso} = \sum_k \sum_c \|W^e_{(c,k)}\|_2, \tag{3}$$

$$W^e_{(c,k)} = \begin{cases} W^e_{(c,k)}, & \text{if } \|W^e_{(c,k)}\|_2 > \gamma, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

**Loading Balance.** We denote the sparsity percentage out of full softmax in $k$-th expert as sparsity$_k$ and proportion of $k$-th expert activated as utilization$_k$. Then, the overall speedup compared to the full softmax can be calculated as as $1/\sum_k(\text{utilization}_k * \text{sparsity}_k)$. Thus, better utilization is essential for speedup as well. For example, there is no speedup if the expert with full output space is always chosen. We borrow a similar loading balance function from Shazeer et al. (2017) in Eq. 5. It encourages the utilization percentage of each expert to be balanced by maximizing the coefficient of variation (CV) for gating outputs. In addition, to encourage the exclusiveness of classes, we incorporate group lasso loss on expert level where each class should only exist in only one expert as shown in Eq. 6.

$$\mathcal{L}_{load} = -\text{CV}\left(\sum_{h \in H(x)} G(h)\right), \tag{5}$$

$$\mathcal{L}_{expert} = \sum_k \sqrt{\sum_c \|W^e_{(c,k)}\|_2^2}. \tag{6}$$

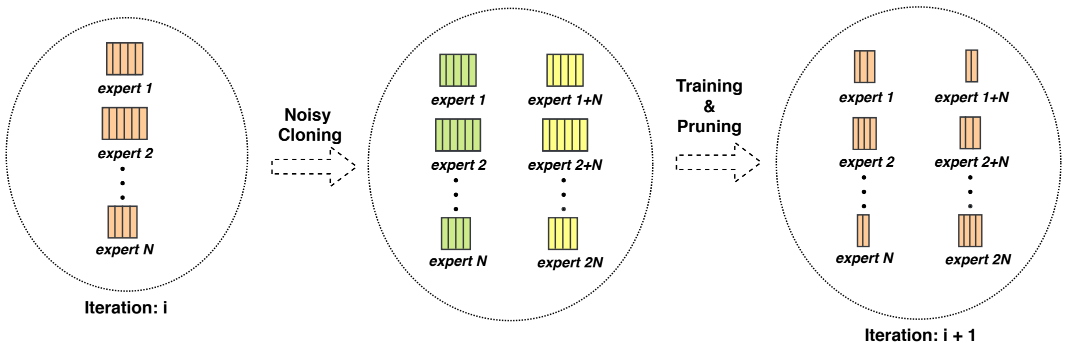



Figure 3: The mitosis training scheme: the sparsity is inherited when parent experts produce offspring, reducing the memory requirements for training with more experts.

**Algorithm 1**

**Initialization:** Let $x$ be the input, $y$ be the corresponding label, $H$ be the pretrained function, $V$ be the output dimension and $D(y', y)$ be an arbitrarily distance function. Set $W^e \leftarrow$ parameters for experts and $W^g \leftarrow$ parameters for the gating network. The hyper-parameter $t$ denotes target performance.

**while** $epoch < \text{Max}$ **do**

  $epoch = epoch + 1$

  $\mathcal{L}_{task} = D(O(H(x)), y)$

  $\mathcal{L}_{all} = \mathcal{L}_{task} + \lambda_{load}\,\mathcal{L}_{load} + \lambda_{lasso}\,\mathcal{L}_{lasso} + \lambda_{expert}\,\mathcal{L}_{expert}$

  $W^e = W^e - \alpha\frac{\partial}{\partial W^e}\mathcal{L}_{all}(x, y; W^e, W^g)$

  $W^g = W^g - \alpha\frac{\partial}{\partial W^g}\mathcal{L}_{all}(x, y; W^e, W^g)$

  **if** $L_{task} < t$ **then**

    **for all** $W^e_{(c,k)} \in W^e$ **do**

      $W^e_{(c,k)} = 0$, if $\|W^e_{(c,k)}\|_2 < \gamma$

**Mitosis training.** Memory might become a bottleneck during training if we initialize all experts with full softmax. Therefore, we design one training scheme, called mitosis training, to reduce memory requirement. The method is to initialize with a smaller model (fewer number of experts) and then gradually breed to a bigger one after noisy cloning shown in Fig. 3. For each cloning, the sparsity is inherited so that less memory is required. For example, in one of our experiments, we only need 3.25x memory with 64 experts compared to a full softmax implementation.

**The final training algorithm.** Our final training objective, $\mathcal{L}_{all}$, consists of a combination of the related contributions discussed above. We describe our training procedure in Algorithm 1.