Adaptive Sampled Softmax with Kernel Based Sampling

Guy Blanc [1] **Steffen Rendle** [2]

Abstract

Softmax is the most commonly used output function for multiclass problems and is widely used in areas such as vision, natural language processing, and recommendation. A softmax model has linear costs in the number of classes which makes it too expensive for many real-world problems. A common approach to speed up training involves sampling only some of the classes at each training step. It is known that this method is biased and that the bias increases the more the sampling distribution deviates from the output distribution. Nevertheless, almost all recent work uses simple sampling distributions that require a large sample size to mitigate the bias. In this work, we propose a new class of kernel based sampling methods and develop an efficient sampling algorithm. Kernel based sampling adapts to the model as it is trained, thus resulting in low bias. It can also be easily applied to many models because it relies only on the model's last hidden layer. We empirically study the trade-off of bias, sampling distribution and sample size and show that kernel based sampling results in low bias with few samples.

1. Introduction

Classification problems with a large number of classes are common in many language tasks (Mikolov et al., 2013; Bengio & Sénécal, 2008) and recommender systems (Covington et al., 2016). A standard and effective approach to these classification tasks is to use some model, such as a neural network, to compute a logit for each class, and assume that the class probabilities are a softmax of the logits. Computing class probabilities with softmax involves a normalization step where a *partition function* over the logits of all classes is computed. For learning the model parameters, an optimization algorithm, e.g., stochastic gradient descent, needs to compute the gradients with respect to the loss. When the number of classes, n, is large, computing the probability of each class is often too slow, as the time for each training step grows linearly with n. Sampled softmax, which creates a sample of $m < n$ classes in every update step, is commonly used when the number of classes becomes too large. It is well known that sampled softmax is biased (Bengio & Sénécal, 2008), i.e., it does not converge to the same loss as a full softmax – no matter how many update steps are taken. The only way to eliminate the bias is to sample from the softmax distribution which is not efficient. For any other sampling distribution, there are two directions to mitigate the bias: (i) choose a sampling distribution that is closer to softmax, or (ii) increase the sample size, m – which is trivial but costly. Early work (Bengio & Sénécal, 2008) has shown that a good sampling distribution should be adaptive and should depend on the model's output.

While the importance of the sampling distribution is known, surprisingly, almost all recent applications use simple sampling distributions such as uniform or global popularity, which require large sample sizes to achieve an acceptable bias. One reason for this trend could be that the models have tended to get more complex, e.g. stacked LSTMs, very deep networks, convolutional NN, etc. which makes it hard to design an efficient sampling distribution that adapts to the model.

In this work, we propose a new class of sampling distributions that approximate softmax but are efficient to compute. The proposed sampling distributions are defined over the model's output, making them adaptive to the input, the model's structure, and the current model parameters. The main idea is to sample proportionally to a non-negative kernel. We show that kernels allow us to compute the partition function efficiently in the kernel space. This result can be used in a divide and conquer algorithm that samples in $\mathcal{O}(D \log n)$ time, where D is the dimension of the kernel space. We suggest the quadratic kernel as an approximation for (absolute) softmax. See Section 3.3 for details. Kernel based sampling is generic and can be applied directly to any model where the final layer is a dot product between a hidden layer and class embeddings.

We study the bias of uniform, quadratic kernel and softmax

[1]Work done during internship at Google, Mountain View, USA [2]Google, Mountain View, USA. Correspondence to: Guy Blanc <guy.blanc@gmail.com>, Steffen Rendle <srendle@google.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

sampling empirically and show that the quadratic kernel needs one to two orders of magnitude less samples than uniform to reach the same quality as full softmax. A second observation is that once the bias is eliminated, more samples usually do not increase the convergence speed.

2. Modelling Large Multiclass Problems

In this section, we first formalize the multiclass softmax and then recap its sampling version.

Let $\boldsymbol{y} \in [0,1]^n$ with $\sum_{i=1}^n y_i = 1$ be a distribution over n classes for an input $\boldsymbol{x} \in \mathcal{X}$. The goal of supervised learning is to find a function that explains a set of observed pairs $(\boldsymbol{x}, \boldsymbol{y})$ of input $\boldsymbol{x}$ and label $\boldsymbol{y}$. Let $\boldsymbol{o} : \mathcal{X} \times \Theta \to \mathbb{R}^n$ be such a function that maps an input $\boldsymbol{x}$ to a raw score for each class. The model function $\boldsymbol{o}$ is parameterized by model parameters $\boldsymbol{\theta} \in \Theta$. To shorten notation, we drop the arguments $\boldsymbol{x}$ and $\boldsymbol{\theta}$ from $\boldsymbol{o}$ and all derived functions, whenever the dependency is clear.

2.1. Full Softmax

A *softmax* model links the model outputs $\boldsymbol{o}$ to a class probability distribution $\boldsymbol{p} \in [0,1]^n$ with $\sum_{i=1}^n p_i = 1$ by applying an exponential function

$$p_i := \frac{\exp(o_i)}{\sum_{j=1}^n \exp(o_j)} \tag{1}$$

The denominator of p_i is also known as the *partition function* and takes at least $\mathcal{O}(n)$ time to compute. For softmax, the output $\boldsymbol{o}$ is often referred to as the *logits*. The loss L of a parameter setting $\boldsymbol{\theta}$ is measured by the cross entropy between $\boldsymbol{y}$ and $\boldsymbol{p}$

$$L(\boldsymbol{y}, \boldsymbol{p}) := -\sum_{i=1}^n y_i \log p_i = \log \sum_{i=1}^n \exp(o_i) - \sum_{i=1}^n y_i \, o_i$$

This *full* softmax loss depends on *all* classes. Thus, learning a full softmax is expensive when the number of classes, n, is large.

2.2. Sampled Softmax

Sampled softmax aims to approximate a full softmax during model training (Bengio & Sénécal, 2008; 2003). Rather than computing the loss over all classes, only the positive class and a sample of m negative classes are considered. Each negative class is sampled with probability q_i with replacement. For the rest of the paper, we assume w.l.o.g. that there is one positive class per training example, i.e., $\boldsymbol{y} \in \{0,1\}^n$. The vector $\boldsymbol{s} \in \{1, \dots, n\}^{m+1}$ represents a sample of classes and stores the index of the positive and the index of the m sampled negative classes. For instance, $\boldsymbol{s} = (2, 6, 7, 6, 3)$, represents a sample of size $m = 4$ with the positive class at index 2 and four negative classes, where the class at index 6 was sampled twice, and the classes at index 7 and 3 once each.

Just as $\boldsymbol{o}$, $\boldsymbol{y}$, and $\boldsymbol{p}$ with cardinality n refer to important characteristics of all the classes, $\boldsymbol{o}'$, $\boldsymbol{y}'$, and $\boldsymbol{p}'$ with cardinality $m+1$ reflect similar values for a sample $\boldsymbol{s}$. First, each index $i \in \{1, \dots, m+1\}$ of the sample $\boldsymbol{s}$ is assigned an adjusted logit o'_i.

$$o'_i := \begin{cases} o_{s_i} - \ln(m \, q_{s_i}) & \text{if } y_{s_i} = 0 \\ o_{s_i} - \ln(1) = o_{s_i} & \text{else} \end{cases} \tag{2}$$

The adjusted logit corrects the true logit o_{s_i} by the expected number of occurences of a class s_i in the sample $\boldsymbol{s}$. This correction ensures that in the limit of $m \to \infty$, sampled softmax is unbiased (Bengio & Sénécal, 2008).

Second, $\boldsymbol{p}'$ is the softmax probability distribution computed over adjusted logits $\boldsymbol{o}'$, and $\boldsymbol{y}'$ is a projection of the original labels $\boldsymbol{y}$ to the sample $\boldsymbol{s}$.

$$p'_i := \frac{\exp(o'_i)}{\sum_{j=1}^{m+1} \exp(o'_j)}, \quad y'_i := y_{s_i} \tag{3}$$

The loss of a sample $\boldsymbol{s}$ is the cross entropy $L(\boldsymbol{y}', \boldsymbol{p}')$ between predicted probabilities $\boldsymbol{p}'$ and labels $\boldsymbol{y}'$. In contrast to full softmax, the loss of sampled softmax depends only on (at most) $m+1$ different classes.

2.3. Importance of the Sampling Distribution

Sampled softmax can be viewed as an algorithm that generates an estimator for the full softmax gradient with respect to the logits. The full softmax gradient with respect to a logit o_i is

$$\frac{\partial L(\boldsymbol{p}, \boldsymbol{y})}{\partial o_i} = p_i - y_i \tag{4}$$

whereas the sampled softmax gradient with respect to an original logit o_i reads

$$\begin{aligned} \frac{\partial L(\boldsymbol{p}', \boldsymbol{y}')}{\partial o_i} &= \sum_{j=1}^{m+1} I(s_j = i)(p'_j - y'_j) \\ &= \sum_{j=1}^{m+1} I(s_j = i) p'_j - y_i \end{aligned} \tag{5}$$

Ideally, we would like to pick a sampling distribution such that sampled softmax converges to the same value as full softmax. At the very least, we would like to guarantee convergence with infinitely small step size and infinitely many steps. That is guaranteed if the sampled softmax

estimator is unbiased:

$$E\left[\frac{\partial L(\boldsymbol{p}', \boldsymbol{y}')}{\partial o_i}\right] \stackrel{?}{=} \frac{\partial L(\boldsymbol{p}, \boldsymbol{y})}{\partial o_i} \tag{6}$$

$$\Leftrightarrow E\left[\sum_{j=1}^{m+1} I(s_j = i) p'_j\right] \stackrel{?}{=} p_i \tag{7}$$

Bengio & Sénécal (2008) have shown that sampling proportional to the softmax probability, $q_i = p_i \propto \exp(o_i)$, is an unbiased estimator. In fact, $q_i = p_i \propto \exp(o_i)$ is the *only* unbiased estimator.

Theorem 2.1. *The gradient of sample softmax is an unbiased estimator of the full softmax gradient iff* $q_i = p_i \propto \exp(o_i)$.

We include a detailed proof in the supplementary material.

2.4. Properties of a Good Sampling Distribution

The last sections argued that sampled softmax is biased and the only way to mitigate the bias are (1) choose a sampling distribution q_i closer to softmax p_i or (2) increase the sample size, m. The closer the sampling distribution q_i reflects p_i, the smaller the sampling size that is needed for low bias. Finally, we highlight three properties of the softmax distribution, $p_i \propto o_i(\boldsymbol{x}, \theta)$, that a good sampling distribution, $\boldsymbol{q}$, should meet as well.

1. *Example dependent*: Every input, $\boldsymbol{x}$, has an individual sampling distribution, because the output, $\boldsymbol{o}(\boldsymbol{x})$, depends on the input, $\boldsymbol{x}$.
2. *Model structure dependent*: The sampling depends on the functional structure of $\boldsymbol{o}$. For instance, if $\boldsymbol{o}$ is an LSTM, the sampling distribution should not be represented by simple bigrams.
3. *Model parameter dependent*: The sampling distribution changes while the model is learned, because $\boldsymbol{o}$ depends on the model parameters.

Common sampling schemes such as uniform or popularity based sampling are neither example nor model dependent. In the following section, we introduce a sampling algorithm that meets these criteria and is efficient.

3. Kernel Based Sampling

Sampling directly from $q_i \propto \exp(o_i)$ requires computing the partition function and is as expensive as computing the full softmax. The motivation for sampling is to avoid that inefficiency, so sampling from $q_i \propto \exp(o_i)$ is not a good option. In this section, we propose efficient sampling distributions that depend on the example $\boldsymbol{x}$, the model structure $\boldsymbol{o}$ and the model parameter θ as highlighted in Section 2.4.

So far, we have ignored how the logits $\boldsymbol{o}$ are computed. In the following, we assume that o_i is a dot product between a context or query embedding, $\boldsymbol{h} \in \mathbb{R}^d$, and a class embedding, $\boldsymbol{w}_i \in \mathbb{R}^d$. This type of model is extremely common with many examples such as deep neural networks and factorization models. For example, $\boldsymbol{h}$ could be the last hidden layer of a deep neural network and $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ the last matrix of weights, such that $\boldsymbol{o} = \boldsymbol{W}^T \boldsymbol{h}$. The cost of computing the full softmax on a dot product model is $\mathcal{O}(nd)$.

3.1. Kernel Based Distributions

We consider sampling distributions that are proportional to some function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$. We assume that K is a kernel function for a D dimensional space, i.e., there exists a mapping $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^D$ such that $K(\boldsymbol{a}, \boldsymbol{b}) = \langle \boldsymbol{\phi}(\boldsymbol{a}), \boldsymbol{\phi}(\boldsymbol{b}) \rangle$. Thus, the sampling distribution can be written as:

$$q_i = \frac{K(\boldsymbol{h}, \boldsymbol{w}_i)}{\sum_{j=1}^n K(\boldsymbol{h}, \boldsymbol{w}_j)} = \frac{K(\boldsymbol{h}, \boldsymbol{w}_i)}{\left\langle \boldsymbol{\phi}(\boldsymbol{h}), \underbrace{\sum_{j=1}^n \boldsymbol{\phi}(\boldsymbol{w}_j)}_{=: \boldsymbol{z} \in \mathbb{R}^D} \right\rangle} \tag{8}$$

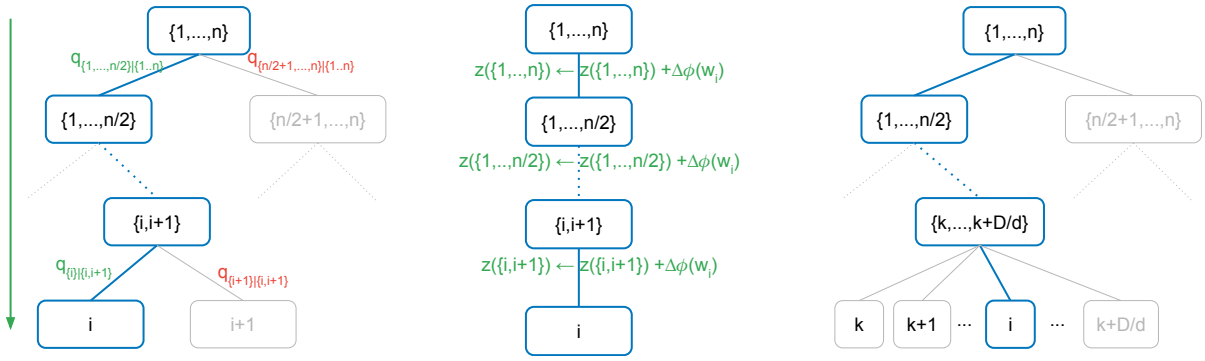
The last step shows the key property that we gain from a kernel: the summation over all classes can be isolated from the query $\boldsymbol{h}$ – i.e., the partition function becomes a simple dot product between a query vector and a summary vector $\boldsymbol{z}$. This summary vector is independent of the query and can be precomputed.

3.2. Sampling with Divide and Conquer

The kernel gives the ability to compute the probability of *one* class efficiently. Next, we discuss how this property can be used for efficient sampling from *all* classes. Instead of sampling a class directly from all the possible classes, we sample a subset of classes recursively until the subset has only one class (see Figure 1(a)). To formalize this algorithm, we introduce $C \subseteq \{1, \dots, n\}$ as a set of classes and define $\boldsymbol{z}(C) := \sum_{j \in C} \boldsymbol{\phi}(\boldsymbol{w}_j)$. Let $C' \cup C'' = C$ be a partition of C into two disjoint sets C' and $C'' = C \setminus C'$. We define the probability, $q_{C'|C}$, of sampling the set C' from C, as the sum of the probabilities of its elements:

$$\begin{aligned} q_{C'|C} &:= \sum_{j \in C'} \frac{K(\boldsymbol{h}, \boldsymbol{w}_j)}{\sum_{l \in C} K(\boldsymbol{h}, \boldsymbol{w}_l)} \\ &= \frac{\langle \phi(\boldsymbol{h}), \sum_{j \in C'} \phi(\boldsymbol{w}_j) \rangle}{\langle \phi(\boldsymbol{h}), \sum_{l \in C} \phi(\boldsymbol{w}_l) \rangle} = \frac{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C') \rangle}{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C) \rangle} \end{aligned} \tag{9}$$

If we know $\boldsymbol{z}$ for C and C', we can sample from this distribution in $\mathcal{O}(D)$ time. This scheme can be applied recursively to the sampled subset until the subset contains exactly one class. With n classes and two sets of equal size at each step, this takes $\log_2 n$ steps and in total the time for sampling a class proportional to $\boldsymbol{q}$ is $\mathcal{O}(D \log_2 n)$.



(a) sampling a class (b) updating statistics (c) large branching factor for leaves

Figure 1. Divide and conquer algorithm for sampling from a kernel distribution q. Figure 1(a) shows how to sample subsets starting from all classes $\{1, \dots, n\}$ until a single item i is reached. After the class embedding, $\boldsymbol{w}_i$, of class i changes from $\boldsymbol{w}_i^{\text{old}}$ to $\boldsymbol{w}_i^{\text{new}}$, all statistics, $\boldsymbol{z}$, on the sampling path of i are updated by $\Delta\phi(\boldsymbol{w}_i) := \phi(\boldsymbol{w}_i^{\text{new}}) - \phi(\boldsymbol{w}_i^{\text{old}})$ (Figure 1(b)). To minimize storage costs for statistics $\boldsymbol{z}$, it is beneficial to use a higher branching factor of $\mathcal{O}(\frac{D}{d})$ for the leaves (Figure 1(c)).

3.2.1. Analysis

Correctness The correctness of the divide and conquer algorithm, i.e., that it samples proportional to the kernel distribution (eq. 8), is easy to show. Assume the algorithm samples class i and the intermediate sets were C_1, C_2, …, $C_{\log n - 1}$. The probability for sampling class i with the divide and conquer algorithm is equal to q_i:

$$
\begin{aligned}
&q_{C_1|\{1,\dots,n\}}\, q_{C_2|C_1} \cdots q_{\{i\}|C_{\log n-1}} \\
=&\frac{\cancel{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C_1)\rangle}}{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(\{1,\dots,n\})\rangle} \frac{\cancel{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C_2)\rangle}}{\cancel{\langle \phi(\boldsymbol{h}), \boldsymbol{z}(C_1)\rangle}} \cdots \frac{\langle \phi(\boldsymbol{h}), \phi(\boldsymbol{w}_i)\rangle}{\cancel{\langle \phi(\boldsymbol{h}), \boldsymbol{z}_{\log n-1}\rangle}} \\
=&\frac{K(\boldsymbol{h}, \boldsymbol{w}_i)}{\langle \phi(\boldsymbol{h}), \boldsymbol{z}\rangle} = q_i \quad \square
\end{aligned}
$$

Runtime The divide and conquer algorithm assumes that $\boldsymbol{z}(C)$ is known for every set that is involved in sampling. As sampling is independent of the particular choice of the splits, we can choose any arbitrary (binary and balanced) split and keep it fixed. In total, there are n many sets that are arranged in a tree like structure and each class appears in exactly $\log_2 n$ many sets. This allows to precompute $\boldsymbol{z}(C)$ for any of the n sets. If we update an embedding, $\boldsymbol{w}_i$ during training, we can also update all sets in which i appears in, in time $O(D \log n)$ by updating $\boldsymbol{z}(C)$ for every node along the path from the root to that embedding. Figure 1(b) illustrates the update process.

3.2.2. Practical Considerations

Less Memory The structure described so far has $\mathcal{O}(n)$ nodes in total, each of which must store $\mathcal{O}(D)$ information for $\boldsymbol{z}$. This means $\mathcal{O}(nD)$ space is required to store it. Here we will describe how to reduce that to $\mathcal{O}(nd)$ space while maintaining fast sampling and updating.

Instead of splitting sets until they reach the trivial size 1, we suggest to stop splitting as soon as the size of a set is $\mathcal{O}(\frac{D}{d})$. This leads to the tree having a total of $\mathcal{O}(\frac{nd}{D})$ sets, and requires $O(nd)$ memory. Figure 1(c) sketches the sampling process with a larger branching factor for the leaves. Increasing the branching factor seems very costly for the final step because the algorithm has to sample from a set of $\mathcal{O}(\frac{D}{d})$ many classes. However, for most kernels, $K(\boldsymbol{a}, \boldsymbol{b})$ can be computed efficiently in $\mathcal{O}(d)$ time, e.g., for kernels of the form $K(\boldsymbol{a}, \boldsymbol{b}) = f(\langle \boldsymbol{a}, \boldsymbol{b}\rangle)$. Thus, performing the last step in the original space takes $\mathcal{O}(d\frac{D}{d}) = \mathcal{O}(D)$ time even with a naive implementation. The proposed modification decreases the height of the tree from $\mathcal{O}(\log_2 n)$ to $\mathcal{O}(\log_2 \frac{nd}{D})$, and adds a final step to sampling with time $\mathcal{O}(D)$. The total sampling time is thus $\mathcal{O}(D(1 + \log_2 \frac{nd}{D})$ which is still $\mathcal{O}(D \log_2 n)$.

Multiple Partial Samples Usually, we want to sample several negatives from $\boldsymbol{q}$. Instead of applying the divide and conquer algorithm m times, a single run could return all the $\frac{D}{d}$ leaf nodes. This would require an additional correction in sampled softmax to accept a weight on each sample. Then, instead of q_i being the probability of sampling a particular class, it is the probability of sampling a class multiplied by the weight given to that class when it is sampled. The drawback of this approach is that the samples are not independent and likely more total samples would be needed. We do not further investigate this approach, but in some applications, faster sampling might justify the cost of requiring a few more samples.

3.3. Quadratic Kernel

One obvious choice for a kernel is a quadratic function $K(\boldsymbol{h}, \boldsymbol{w}_i) = \alpha\langle \boldsymbol{h}, \boldsymbol{w}_i\rangle^2 + 1$. This function is conveniently

always positive. Its feature representation is

$$\phi(\boldsymbol{a}) = \left[\sqrt{\alpha}\,\mathrm{vec}(\boldsymbol{a} \otimes \boldsymbol{a}), 1\right] \tag{10}$$

with $D = O(d^2)$, allowing for $O(d^2 \log n)$ sampling. It is also a reasonably good approximation of $\exp$ near the origin, where many logits tend to be. However, a quadratic function is a poor approximation for negative logits and would oversample classes with negative logits. To align the sampling distribution $\boldsymbol{q}$ better with the prediction distribution $\boldsymbol{p}$, we suggest a modification of the softmax probability, $\boldsymbol{p}$, in eq. (1) to an *absolute softmax*

$$p_i = \frac{\exp(|o_i|)}{\sum_{j=1}^{n} \exp(|o_j|)} \tag{11}$$

This modified prediction distribution does not negatively impact the expressiveness because softmax is shift invariant, i.e., $q_i \propto \exp(o_i) \propto \exp(o_i)\exp(c) = \exp(o_i + c)$ for any constant $c \in \mathbb{R}$. In particular, any *softmax* solution has a corresponding *absolute softmax* solution by shifting the logits, $\boldsymbol{o}$, of the softmax solution by any c large enough to make all the logits nonnegative. We investigated also empirically the quality of *softmax* and *absolute softmax* as prediction distribution when learning without sampling, i.e., full softmax, and both performed very similarly[1] on the datasets of Section 4.1.1. Finally, analogous to Section 2.3, for absolute softmax as the prediction distribution, the only unbiased sampling distribution is absolute softmax. This follows directly from Theorem 2.1 in the supplementary material, because the analysis was shown for $p_i \propto \exp(o_i)$ and *any* output o_i, so it also holds for the modified output $|o_i|$. Therefore, we suggest to use an absolute softmax as prediction distribution when sampling from a symmetric kernel like the quadratic kernel and a standard softmax in other cases.

Another way to look at absolute softmax is to add an additional layer to $\boldsymbol{o}$ that performs $|\boldsymbol{o}|$ and then passing the result to a standard softmax.

4. Experiments

In this section, we empirically investigate the trade-off between bias, sampling distribution, and number of samples.

4.1. Experimental Setup

4.1.1. Datasets and Models

We study sampled softmax on a natural language processing (NLP) problem and a recommender system dataset.

Penn Tree Bank For the NLP problem, we learn a language model on the Penn Tree Bank dataset (Marcus et al., 1999), a dataset with approximately 1 million training words and a vocabulary of size 10,000. We use the well-studied "medium regularized LSTM" implementation[2] of Zaremba et al. (2014). We made one minor modification, and changed the units per layer from 650 to 200. Doing so ensures that the expressiveness of the model is small enough that we do not need to worry about early-stopping, and dropout on its own is a sufficient regularizer. We report the perplexity loss as in (Zaremba et al., 2014).

YouTube In this recommendation dataset, we predict which video a user will watch next based upon various user features and the three previously watched videos. We train a deep neural network where the user features and previous videos are the input and the output is the watch probability over all videos. To study the effect on sampling, we created two versions of the dataset: YouTube10k, and YouTube100k with 10,000, and 100,000 videos (=classes) respectively. The 10k dataset has about 113 million training examples, and the 100k dataset about 187 million examples. For recommender systems, a common evaluation protocol is to rank videos by their scores and then use some ranking metric (e.g. mean average precision) to measure the quality of a model. Here, we only wish to measure how well sampled softmax approximates a full softmax. Thus, we measure the cross-entropy loss of full softmax. In our YouTube experiments, the cross-entropy loss was also highly correlated with ranking metrics such as mean average precision.

4.1.2. Sampling Distributions

We test the performance of three sampling distributions:

1. Uniform distribution, $q_i \propto 1$, where every class is sampled with the same probability. This provides a convenient baseline.
2. Softmax distribution, $q_i \propto \exp(o_i)$, which is the ideal sampling distribution as shown in Theorem 2.1, but is very expensive to sample from.
3. Quadratic distribution, $q_i \propto 100(o_i)^2 + 1$, as proposed in Section 3.3

4.2. Results and Analysis

4.2.1. Bias of Sampling

First, we study the bias of sampled softmax empirically. According to Section 2.3, any sampled softmax is biased unless softmax is chosen as the sampling distribution, and this bias decreases as the sample size, m, increases. We visualize the bias by learning models with different sampling

[1] Similar empirical findings were obtained by Brébisson & Vincent (2015) on various tasks.

[2] https://www.tensorflow.org/tutorials/recurrent

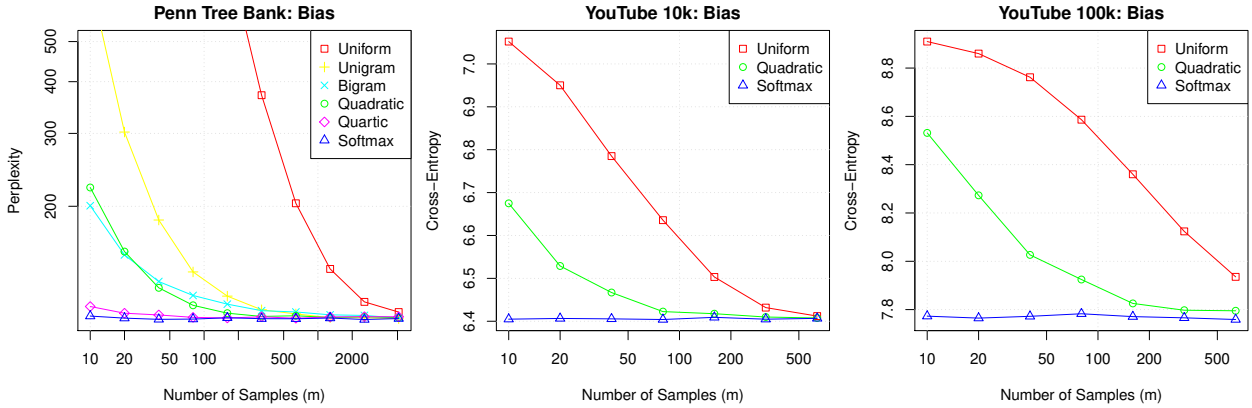


Figure 2. Final model quality when training a sampled softmax with different sampling distributions (*uniform*, *quadratic*, *softmax*) and number of samples, m. The quadratic distribution needs one to two orders of magnitude less samples than uniform sampling to learn a low bias model. Penn Tree Bank includes additional results for a *unigram* and a *bigram* sampler which are common sampling distributions in NLP sequence tasks. The results for Penn Tree Bank also include a *quartic* sampler which is a 4-th degree polynomial kernel with $q_i \propto o_i^4 + 1$.

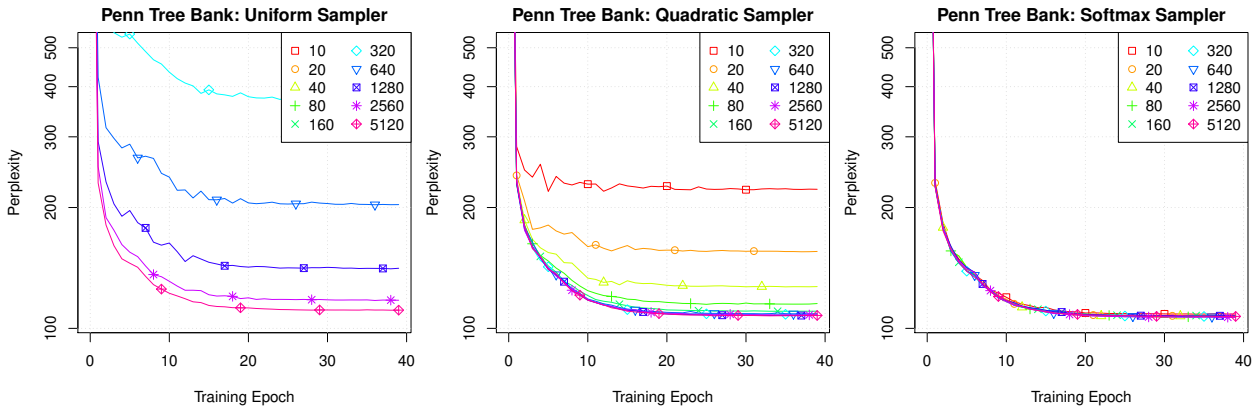


Figure 3. Convergence speed for a varying sample size $m \in \{10, 20, 40, \ldots\}$. Once enough samples are taken to remove the bias, adding more samples does not increase convergence speed considerably. The results for YouTube10k and YouTube100k show a similar behavior an can be found in the supplementary material.

strategies until convergence and reporting the final accuracy. Very biased models perform poorly even when they are run until convergence.

The results are shown in Figure 2. As expected from theorem 2.1, the quality of softmax sampling, i.e., $\boldsymbol{q} \propto \exp(\boldsymbol{o})$, is independent of the number of samples m. This verifies that a "good" sampling distribution does not need many samples. On the other hand, uniform and quadratic sampling are both biased and their final quality improves with increasing sample size, m. Again, it is important to note that training for more epochs does not solve this issue because the loss that sampled softmax optimized is biased when sampling uniformly or according to a quadratic kernel for any fixed size m. On all datasets, quadratic has a much lower bias than uniform sampling and approaches the loss of softmax with 10s to 100s of samples.

4.2.2. Convergence Speed

Second, we study the speed of convergence by measuring the progress of the loss against the number of training epochs. Every update step consists of reading a batch of training examples, sampling m negative classes per example and performing the update with sampled softmax. We plot loss against epochs instead of wall runtime to eliminate any implementation specific artifacts. Please note that the larger the sample size m, the more computationally expensive an epoch.

Sample Size First, we study how the sample size, m, influences convergence speed. Figure 3 shows the convergence for the three sampling strategies. As already discussed, we see that the number of samples has a large effect on the accuracy of the model for the uniform and quadratic sam-

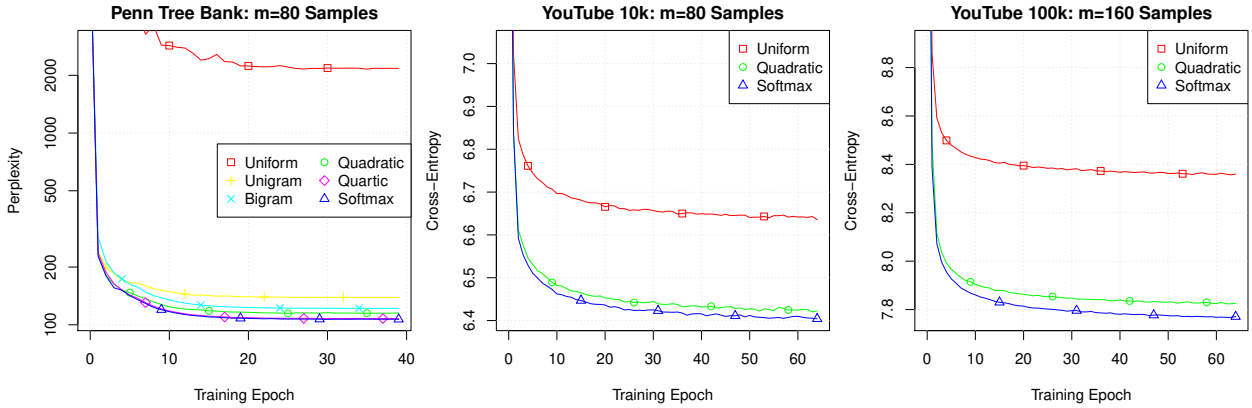


Figure 4. Convergence speed of different sampling distributions for a fixed sampling size. The convergence speed of all distributions is similar only the bias is different. The supplementary material shows additional graphs for $m \in \{80, 160, 320\}$ for Penn Tree Bank, YouTube 10k, and YouTube 100k.

pler. Interestingly, once enough samples are taken to remove the bias, adding more samples appears to have a small and mostly unobservable effect on convergence speed. We previously discussed how bias of the sampled softmax estimator affects the final optimum achieved. The variance of this estimator affects how many steps we need to converge. The variance has two sources: (i) The gradient computed on a batch is a noisy (but unbiased) estimator of the gradient on the entire training set and (ii) the gradient given a set of sampled classes is an estimator of the gradient on that batch. While taking a larger sample size can reduce the variance from source (ii), if the variance from source (i) is the dominate source, doing so will not appreciably increase convergence speed. For our data sets, we found that once we take a reasonable number of samples (only 10s), adding more does not noticeably increase convergence speed. This is likely because the variance from source (i) dominates that from source (ii). For instance on Penn Tree Bank, quadratic sampling with $m \in \{160, 320, \ldots\}$ samples does not show any difference in convergence speed.

To summarize, the sample size m influences the bias but the influence on the convergence speed is small and often not noticeable.

Sampling Distribution Finally, we fix the number of samples m and vary the sampling distribution. Figure 4 shows that all three sampling distributions have a comparable speed of convergence, however, uniform converges to a much worse loss due to its high bias. Quadratic and softmax converge similarly although quadratic has a slightly worse loss throughout the whole training process due to its bias.

5. Related Work

In this section, we summarize the main approaches for training classification models over many classes. All of them make some approximation of the full softmax to lower the computational complexity.

5.1. Sampled Softmax

Other works on sampled softmax have noted that a good sampling distribution can boost performance and attempted to come up with such distributions. Bengio & Sénécal (2008) propose an adaptive sampler for language models. They argue that the sampling distribution should track the model distribution as closely as possible. They propose to learn a mixture of unigrams, bigrams, trigrams, etc. that is adapted while training. While the work of Bengio & Sénécal (2008) needs a second model to track the trained model, our work uses the trained model directly for sampling. This makes our approach much easier to apply. Secondly, kernel based sampling is more appealing for sophisticated model structures where it is hard to come up with a simple model that can track the trained model well. Labeau & Allauzen (2017) study sampling distributions for noise contrastive estimation (NCE) (Gutmann & Hyvrinen, 2010). Their experiments highlight the issues of simple sampling distributions such as uniform, or unigram. Another idea to improve the sampling distribution is the Two-Pass Approximate Adaptive Sampling for Softmax (TAPAS). In that work, Bai et al. (2017) propose taking one large sample of classes, which might be in the order of 100,000 (20% of all classes in their case) and computing the logits from that sample. Then, a smaller number of classes, e.g., 1,000, is chosen from those 100,000 classes based on the computed logits. This second sample of 1,000 classes is used for the sampled softmax. By using a distributed implementation and GPUs, it is possible

to compute the logits of the larger sample quickly. While the TAPAS sampler is adaptive and depends on the current model's output as in our work, it is computationally much more expensive. Bakhtiary et al. (2015) also explore selectively computing logits using hashing to obtain faster training steps for large batch sizes.

5.2. Hierarchical Softmax and Its Variations

Hierachical Softmax (HSM) is an approximation of a full softmax introduced in (Goodman, 2001) that is quickly computable. It involves grouping the classes into clusters, where each cluster is a latent variable.[3] If c_j is the j^{th} cluster and class i is in c_j, then we factor p_i as $p(y_i|x) = p(c_j|x)\, p(y_i|c_j)$. If we set the number of classes in each cluster to be $O(\sqrt{n})$ and the cluster probabilities can be computed in time $O(d)$, then this version of hierarchical softmax can be done in $O(d\sqrt{n})$.

Morin & Bengio (2005) extend this structure to a tree. Instead of having one layer of clusters they use a binary tree where each internal node is a cluster and the leaf nodes are the classes. The probability of a class is then the product of the conditional probability of each node along the path from the root to that class. Such a structure allows for $O(d \log n)$ training time.

While hierarchical softmax can be much faster than a full softmax, it often performs worse at convergence. For instance, Chen et al. (2015) found full softmax to achieve a perplexity more than 10% better than hierarchical softmax. They also note that while hierarchical softmax can speed up training, it slows down inference if the goal is to compute the class or classes with the highest logits. In particular, both a full softmax and sampled softmax can treat inference as a maximum inner product search, which can be done in sublinear time with methods such as locally sensitive hashing (Shrivastava & Li, 2014) or clustering (Auvolat & Vincent, 2015). The tree structure has a large effect both on the final performance and the efficiency of each training step, so there is much work on modifying that structure. Various approaches have been used to build this tree, such as by class similarity (Le et al., 2011), by frequency binning (Mikolov et al., 2011), or to optimize the speed of the model (Grave et al., 2017). See Zweig & Makarychev (2013) for experimental results showing the effects of some common tree structures.

5.3. Spherical Softmax and Kernels

Vincent et al. (2015) propose to optimize a variation of softmax referred to as spherical softmax. In spherical softmax, the prediction distribution is changed by replacing the $\exp$ in eq. (1) with a quadratic function. This alternative formulation allows exact gradient computations without computing all the logits, needing only $\mathcal{O}(d^2)$ time. This time is thus independent of the number of classes, resulting in a significant speedup. Brébisson & Vincent (2015) note that on some problems, spherical softmax produces models with comparable quality as full exponential softmax (eq. 1), but on other problems the quality is considerably worse. Especially for problems with many classes, optimizing spherical softmax seems to produce low quality results. We also found the spherical formulation not to be as effective as an exponential softmax on our datasets. Finally, our approach of kernel based sampled softmax with a quadratic kernel can be viewed as using the spherical softmax for sampling, and then the normal exponential softmax formulation for computation of the loss. As in our approach the quadratic kernel is only used for sampling, the high quality of exponential softmax is preserved. Rastogi & Durme (2015) propose kernel feature maps to approximate the softmax partition function during inference after a model has been learned. In contrast to this, our work focuses on using kernels during training which requires dealing with parameter updates.

6. Conclusion

This work shows the importance of the sampling distribution when learning a sampled softmax model. In particular, any sampling distribution besides softmax is biased and converges to a worse quality than full softmax – no matter how many learning steps are taken. The only way to mitigate the bias is to increase the sample size or to use a better sampling distribution. A good sampling distribution should depend on the model output, which requires the distribution to be example dependent, model structure dependent and model parameter dependent. We have introduced the new class of kernel based sampling methods that sample based on the model output. Kernels allow efficient sampling even for a large number of classes as they depend only on the dimension of the kernel space and with a divide and conquer algorithm on the logarithm of the number of classes. On several experiments, a quadratic kernel showed a one to two order better sample efficiency than uniform sampling.

Both the sampling algorithm as well as the kernel function offer several directions for future work. Besides other analytical kernels such as polynomial kernels, random feature maps (Rahimi & Recht, 2008) could provide another rich class for constructing kernels. For particular classes of kernels, more efficient sampling algorithms might exist, e.g., for non-negative maps, ϕ, a clever use of Alias sampling (Walker, 1977) could provide an $\mathcal{O}(D)$ time sampling algorithm.

[3] Some of the literature refers to what we call classes as simply words and uses classes to refer to what we call clusters. We chose the term *classes* instead of *words* to stress that hierarchical softmax can apply to contexts outside of NLP.

References

Auvolat, A. and Vincent, P. Clustering is efficient for approximate maximum inner product search. *CoRR*, abs/1507.05910, 2015. URL http://arxiv.org/abs/1507.05910.

Bai, Y., Goldman, S., and Zhang, L. TAPAS: two-pass approximate adaptive sampling for softmax. *CoRR*, abs/1707.03073, 2017. URL http://arxiv.org/abs/1707.03073.

Bakhtiary, A. H., Lapedriza, À., and Masip, D. Speeding up neural networks for large scale classification using WTA hashing. *CoRR*, abs/1504.07488, 2015. URL http://arxiv.org/abs/1504.07488.

Bengio, Y. and Sénécal, J.-S. Quick training of probabilistic neural nets by importance sampling. In *Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

Bengio, Y. and Sénécal, J.-S. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Transactions on Neural Networks*, 19(4):713–722, April 2008. ISSN 1045-9227.

Brébisson, A. d. and Vincent, P. An exploration of softmax alternatives belonging to the spherical loss family. *CoRR*, abs/1511.05042, 2015. URL http://arxiv.org/abs/1511.05042.

Chen, W., Grangier, D., and Auli, M. Strategies for training large vocabulary neural language models. *CoRR*, abs/1512.04906, 2015. URL http://arxiv.org/abs/1512.04906.

Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 191–198, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4035-9. doi: 10.1145/2959100.2959190.

Goodman, J. Classes for fast maximum entropy training. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*. IEEE, 2001. ISBN 0-7803-7041-4.

Grave, É., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. Efficient softmax approximation for GPUs. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1302–1310, 2017.

Gutmann, M. and Hyvrinen, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Teh, Y. W. and Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

Labeau, M. and Allauzen, A. An experimental analysis of noise-contrastive estimation: the noise distribution matters. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 15–20, Valencia, Spain, April 2017. Association for Computational Linguistics.

Le, H.-S., Oparin, I., Allauzen, A., Gauvain, J.-L., and Yvon, F. Structured output layer neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '11)*, pp. 5524 – 5527. IEEE, 2011. ISBN 978-1-4577-0539-7.

Marcus, M., Santorini, B., Marcinkiewicz, M. A., and Taylor, A. Treebank-3 ldc99t42, 1999.

Mikolov, T., Kombrink, S., Burget, L., Cernock, J., and Khudanpur, S. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '11)*, pp. 5528–5531. IEEE, 2011. ISBN 978-1-4577-0539-7.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pp. 3111–3119, USA, 2013. Curran Associates Inc.

Morin, F. and Bengio, Y. Hierarchical probabilistic neural network language model. In Cowell, R. G. and Ghahramani, Z. (eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pp. 246–252. Society for Artificial Intelligence and Statistics, 2005.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T. (eds.), *Advances in Neural Information Processing Systems 20*, pp. 1177–1184. Curran Associates, Inc., 2008.

Rastogi, P. and Durme, B. V. Sublinear partition estimation. abs/1508.01596, 2015. URL http://arxiv.org/abs/1508.01596.

Shrivastava, A. and Li, P. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 2321–2329, Cambridge, MA, USA, 2014. MIT Press.

Vincent, P., Brébisson, A. d., and Bouthillier, X. Efficient exact gradient update for training deep networks with very large sparse targets. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1108–1116, Cambridge, MA, USA, 2015. MIT Press.

Walker, A. J. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, September 1977. ISSN 0098-3500. doi: 10.1145/355744.355749.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL http://arxiv.org/abs/1409.2329.

Zweig, G. and Makarychev, K. Speed regularization and optimality in word classing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '13)*, pp. 8237–8241. IEEE, 2013. ISBN 978-1-4799-0356-6.