Sampled Softmax with Random Fourier Features

Ankit Singh Rawat, Jiecao Chen, Felix Yu, Ananda Theertha Suresh, and Sanjiv Kumar

Google Research
New York, NY 10011
{ankitsrawat, chenjiecao, felixyu, theertha, sanjivk}@google.com.

January 1, 2020

Abstract

The computational cost of training with softmax cross entropy loss grows linearly with the number of classes. For the settings where a large number of classes are involved, a common method to speed up training is to sample a subset of classes and utilize an estimate of the loss gradient based on these classes, known as the *sampled softmax* method. However, the sampled softmax provides a biased estimate of the gradient unless the samples are drawn from the exact softmax distribution, which is again expensive to compute. Therefore, a widely employed practical approach involves sampling from a simpler distribution in the hope of approximating the exact softmax distribution. In this paper, we develop the first theoretical understanding of the role that different sampling distributions play in determining the quality of sampled softmax. Motivated by our analysis and the work on kernel-based sampling, we propose the *Random Fourier Softmax* (RF-softmax) method that utilizes the powerful Random Fourier Features to enable more efficient and accurate sampling from an approximate softmax distribution. We show that RF-softmax leads to low bias in estimation in terms of both the full softmax distribution and the full softmax gradient. Furthermore, the cost of RF-softmax scales only logarithmically with the number of classes.

1 Introduction

The cross entropy loss based on softmax function is widely used in multi-class classification tasks such as natural language processing [1], image classification [2], and recommendation systems [3]. In multi-class classification, given an input $\mathbf{x} \in \mathcal{X}$, the goal is to predict its class $t \in \{1, 2, \ldots, n\}$, where n is the number of classes. Given an input feature $\mathbf{x}$, the model (often a neural network) first computes an input embedding $\mathbf{h} \in \mathbb{R}^d$ and then the raw scores or *logits* for classes $\mathbf{o} = (o_1, \ldots, o_n)$ as the product of the input embedding $\mathbf{h}$ and the class embeddings $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^d$,

$$o_i = \tau \mathbf{h}^T \mathbf{c}_i. \tag{1}$$

Here, τ is often referred to as the (inverse) *temperature* parameter of softmax. Given the logits, the probability that the model assigns to the i-th class is computed using the *full softmax* function

$$p_i = e^{o_i}/Z, \tag{2}$$

where $Z = \sum_{i=1}^{n} e^{o_i}$ is called the *partition function*. The distribution in (2) is commonly referred to as the softmax distribution. Given a training set, the model parameters are estimated by minimizing an empirical risk over the training set, where the empirical risk is defined by the *cross-entropy loss* based on softmax function or the *full softmax loss*. Let $t \in [n]$ denote the true class for the input $\mathbf{x}$, then the full softmax loss is defined as[1]

$$\mathcal{L}(\mathbf{x}, t) := -\log p_t = -o_t + \log Z. \tag{3}$$

One typically employs first order optimization methods to train neural network models. This requires computing the gradient of the loss with respect to the model parameter $\boldsymbol{\theta}$ during each iteration

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathbf{x}, t) = -\nabla_{\boldsymbol{\theta}} o_t + \sum_{i=1}^{n} \frac{e^{o_i}}{Z} \cdot \nabla_{\boldsymbol{\theta}} o_i = -\nabla_{\boldsymbol{\theta}} o_t + \mathbb{E}_{s\sim p}\left[\nabla_{\boldsymbol{\theta}} o_s\right], \tag{4}$$

where the expectation is taken over the softmax distribution (cf. (2)). As evident from (4), computing the gradient of the full softmax loss takes $\mathcal{O}(dn)$ time due to the contributions from all n classes. Therefore, training a model using the full softmax loss becomes prohibitively expensive in the settings where a large number of classes are involved. To this end, various approaches have been proposed for efficient training. This includes different modified loss functions: hierarchical softmax [5] partitions the classes into a tree based on class similarities, allowing for $\mathcal{O}(d \log n)$ training and inference time; spherical softmax [6, 7] replaces the exponential function by a quadratic function, enabling efficient algorithm to compute the updates of the output weights irrespective of the output size. Efficient hardware-specific implementations of softmax are also being actively studied [8].

1.1 Sampled softmax

A popular approach to speed up the training of full softmax loss is using *sampled softmax*: instead of including all classes during each iteration, a small random subset of n classes is considered, where each *negative* class is sampled with some probability. Formally, let the number of sampled classes during each iteration be m, with class i being picked with probability q_i. Let $\mathcal{N}_t \triangleq [n]\backslash\{t\}$ be the set of negative classes. Assuming that $s_1, \ldots, s_m \in \mathcal{N}_t$ denote the sampled class indices, following [9], we define the adjusted logits $\mathbf{o}' = \{o'_1, o'_2, \ldots, o'_{m+1}\}$ such that $o'_1 = o_t$ and for $i \in [m]$,

$$o'_{i+1} = o_{s_i} - \log(m q_{s_i}). \tag{5}$$

Accordingly, we define the *sampled softmax distribution* as $p'_i = \frac{e^{o'_i}}{Z'}$, where $Z' = \sum_{j=1}^{m+1} e^{o'_j}$. The *sampled softmax loss* corresponds to the cross entropy loss with respect to the sampled softmax distribution:

$$\mathcal{L}'(\mathbf{x}, t) = -\log p'_t = -o_t + \log Z'. \tag{6}$$

Here, we note that adjusting the logits for the sampled negative classes using their expected number of occurrence in (5) ensures that Z' is an unbiased estimator of Z [9]. Since $\mathcal{L}'(\mathbf{x}, t)$ depends only on $m + 1$ classes, the computational cost is reduced from $\mathcal{O}(dn)$ to $\mathcal{O}(dm)$ as compared to the full softmax loss in (3).

In order to realize the training with the full softmax loss, one would like the gradient of the sampled softmax loss to be an unbiased estimator of the gradient of the full softmax loss[2], i.e.,

$$\mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \mathcal{L}'\right] = \nabla_{\boldsymbol{\theta}} \mathcal{L}, \tag{7}$$

[1]The results of this paper generalize to a multi-label setting by using multi-label to multi-class reductions [4].
[2]Since it is clear from the context, in what follows, we denote $\mathcal{L}(\mathbf{x}, t)$ and $\mathcal{L}'(\mathbf{x}, t)$ by $\mathcal{L}$ and $\mathcal{L}'$, respectively.

where the expectation is taken over the sampling distribution q. As it turns out, the sampling distribution plays a crucial role in ensuring the unbiasedness of $\nabla_{\boldsymbol{\theta}}\mathcal{L}'$. Bengio and Senécal [9] show that (7) holds if the sampling distribution is the full softmax distribution itself, i.e., $q_i \propto e^{o_i}$ (cf. (2)).

However, sampling from the softmax distribution itself is again computationally expensive: one needs to compute the partition function Z during each iteration, which is again an $\mathcal{O}(dn)$ operation since Z depends both on the current model parameters and the input. As a feasible alternative, one usually samples from a distribution which does not depend on the current model parameters and the input. Common choices are uniform, log-uniform, or the global prior of classes [10, 1]. However, since these distributions are far from the full softmax distribution, they can lead to significantly worse solutions. Various approaches have been proposed to improve negative sampling. For example, a separate model can be used to track the distribution of softmax in language modeling tasks [9]. One can also use an LSH algorithm to find the approximate nearest classes in the embedding space which in turn helps in sampling from the softmax distribution efficiently [11]. Quadratic kernel softmax [12] uses a kernel-based sampling method and quadratic approximation of the softmax function to draw each sample in sublinear time. Similarly, the Gumbel trick has been proposed to sample from the softmax distribution in sublinear time [13]. The partition function can also be written in a double-sum formulation to enable an unbiased sampling algorithm for SGD [14, 15].

Among other training approaches based on sampled losses, Noise Contrastive Estimation (NCE) and its variants avoid computing the partition function [16], and (semi-)hard negative sampling [17, 4, 18] selects the negatives that most violate the current objective function. Hyvärinen [19] proposes minimization of Fisher divergence (a.k.a. score matching) to avoid computation of the partition function Z. However, in our setting, the partition function depends on the input embedding $\mathbf{h}$, which changes during the training. Thus, while calculating the score function (taking derivative of Z with respect to $(\mathbf{h}, \mathbf{c})$), the partition function has a non-trivial contribution which makes this approach inapplicable to our setting. We also note the existence of MCMC based approaches in the literature (see, e.g., [20]) for sampling classes with a distribution that is close to the softmax distribution. Such methods do not come with precise computational complexity guarantees.

1.2 Our contributions

Theory. Despite a large body of work on improving the quality of sampled softmax, developing a theoretical understanding of the performance of sampled softmax has not received much attention. Blanc and Rendle [12] show that the full softmax distribution is the *only* distribution that provides an unbiased estimate of the true gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$. However, it is not clear *how different sampling distributions affect the bias* $\nabla_{\boldsymbol{\theta}}\mathcal{L} - \mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\mathcal{L}'\right]$. In this paper, we address this issue and characterize the bias of the gradient for a generic sampling distribution (cf. Section 2).

Algorithm. In Section 3, guided by our analysis and recognizing the practical appeal of kernel-based sampling [12], we propose Random Fourier softmax (RF-softmax), a new kernel-based sampling method for the settings with normalized embeddings. RF-softmax employs the powerful Random Fourier Features [21] and guarantees small bias of the gradient estimate. Furthermore, the complexity of sampling one class for RF-softmax is $\mathcal{O}(D \log n)$, where D denotes the number of random features used in RF-softmax. In contrast, assuming that d denotes the embedding dimension, the full softmax and the prior kernel-based sampling method (Quadratic-softmax [12]) incur $\mathcal{O}(dn)$ and $\mathcal{O}(d^2 \log n)$ computational cost to generate one sample, respectively. In practice, D can be two orders of magnitudes smaller than d^2 to achieve similar or better performance. As a result, RF-softmax has two desirable features: 1) better accuracy due to lower bias and 2) computational efficiency due to low sampling cost.

Experiments. We conduct experiments on widely used NLP and extreme classification datasets to demonstrate the utility of the proposed RF-softmax method (cf. Section 4).

2 Gradient bias of sampled softmax

The goal of sampled softmax is to obtain a computationally efficient estimate of the *true gradient* $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ (cf. (4)) of the full softmax loss (cf. (3)) with small bias. In this section we develop a theoretical understanding of how different sampling distributions affect the bias of the gradient. To the best of our knowledge, this is the first result of this kind.

For the cross entropy loss based on the sampled softmax (cf. (6)), the training algorithm employs the following estimate of $\nabla_{\boldsymbol{\theta}}\mathcal{L}$.

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}' = -\nabla_{\boldsymbol{\theta}} o_t + \frac{e^{o_t}\nabla_{\boldsymbol{\theta}} o_t + \sum_{i\in[m]}\frac{e^{o_{s_i}}}{mq_{s_i}}\nabla_{\boldsymbol{\theta}} o_{s_i}}{e^{o_t} + \sum_{i\in[m]}\frac{e^{o_{s_i}}}{mq_{s_i}}}. \tag{8}$$

The following result bounds the bias of the estimate $\nabla_{\boldsymbol{\theta}}\mathcal{L}'$. Without loss of generality, we work with the sampling distributions that assign strictly positive probability to each negative class, i.e., $q_i > 0 \;\; \forall\, i \in \mathcal{N}_t$.

Theorem 1. *Let $\nabla_{\boldsymbol{\theta}}\mathcal{L}'$ (cf. (8)) be the estimate of $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ based on m negative classes $s_1, \ldots, s_m$, drawn according to the sampling distribution q. We further assume that the gradients of the logits $\nabla_{\boldsymbol{\theta}} o_i$ have their coordinates bounded[3] by M. Then, the bias of $\nabla_{\boldsymbol{\theta}}\mathcal{L}'$ satisfies*

$$\mathrm{LB} \le \mathbb{E}\left[\nabla_{\boldsymbol{\theta}}\mathcal{L}'\right] - \nabla_{\boldsymbol{\theta}}\mathcal{L} \le \mathrm{UB} \tag{9}$$

with

$$\mathrm{LB} \triangleq -\frac{M\sum_{k\in\mathcal{N}_t} e^{o_k}\left|Z_t - \frac{e^{o_k}}{q_k}\right|}{mZ^2}\left(1 - o\Big(\frac{1}{m}\Big)\right)\cdot \mathbf{1}, \tag{10}$$

$$\mathrm{UB} \triangleq \Bigg(\underbrace{\frac{\sum_{j\in\mathcal{N}_t}\frac{e^{2o_j}}{q_j} - Z_t^2}{mZ^3}}_{\mathrm{UB}_1} + o\Big(\frac{1}{m}\Big)\Bigg)\cdot \mathbf{g} \,+\, \Bigg(\frac{2M}{m}\underbrace{\frac{\max_{i,i'\in\mathcal{N}_t}\left|\frac{e^{o_i}}{q_i} - \frac{e^{o_{i'}}}{q_{i'}}\right| Z_t}{Z^2 + \sum_{j\in\mathcal{N}_t}\frac{e^{2o_j}}{q_j}}}_{\mathrm{UB}_2} + o\Big(\frac{1}{m}\Big)\Bigg)\cdot \mathbf{1}, \tag{11}$$

where $Z_t \triangleq \sum_{j\in\mathcal{N}_t} e^{o_j}$, $\mathbf{g} \triangleq \sum_{j\in\mathcal{N}_t} e^{o_j}\nabla_{\boldsymbol{\theta}} o_j$ and $\mathbf{1}$ is the all one vector.

The proof of Theorem 1 is presented in Appendix A. Theorem 1 captures the effect of the underlying sampling distribution q on the bias of gradient estimate in terms of three (closely related) quantities:

$$\sum_{j\in\mathcal{N}_t}\frac{e^{2o_j}}{q_j}, \quad \max_{j,j'\in\mathcal{N}_t}\left|\frac{e^{o_j}}{q_j} - \frac{e^{o_{j'}}}{q_{j'}}\right|, \text{ and } \left|\sum_{j\in\mathcal{N}_t} e^{o_j} - \frac{e^{o_k}}{q_k}\right|. \tag{12}$$

Ideally, we would like to pick a sampling distribution for which all these quantities are as small as possible. Since q is a probability distribution, it follows from Cauchy-Schwarz inequality that

$$\sum_j \frac{e^{2o_j}}{q_j} = \Big(\sum_j q_j\Big)\cdot\Big(\sum_j \frac{e^{2o_j}}{q_j}\Big) \ge \Big(\sum_j e^{o_j}\Big)^2. \tag{13}$$

[3]This assumption naturally holds in most of the practical implementations, where each of the gradient coordinates or norm of the gradient is clipped by a threshold.

If $q_j \propto e^{o_j}$, then (13) is attained (equivalently, $\sum_j \frac{e^{2o_j}}{q_j}$ is minimized). In particular, this implies that UB_1 in (11) disappears. Furthermore, for such a distribution we have $q_j = \frac{e^{o_j}}{\sum_{i \in \mathcal{N}_t} e^{o_i}}$. This implies that both UB_2 and LB disappear for such a distribution as well. This guarantees a small bias of the gradient estimate $\nabla_{\boldsymbol{\theta}} \mathcal{L}'$.

Since sampling exactly from the distribution q such that $q_j \propto e^{o_j}$ is computationally expensive, one has to resort to other distributions that incur smaller computational cost. However, to ensure small bias of the gradient estimate $\nabla_{\boldsymbol{\theta}} \mathcal{L}'$, Theorem 1 and the accompanying discussion suggest that it is desirable to employ a distribution that ensures that $\frac{e^{o_j}}{q_j}$ is as close to 1 as possible for each $j \in \mathcal{N}_t$ and all possible values of the logit o_j. In other words, we are interested in those sampling distributions that provide a tight *uniform multiplicative approximation* of e^{o_j} in a computationally efficient manner.

This motivates our main contribution in the next section, where we rely on kernel-based sampling methods to efficiently implement a distribution that uniformly approximates the softmax distribution.

3 Random Fourier Softmax (RF-Softmax)

In this section, guided by the conclusion in Section 2, we propose Random Fourier Softmax (RF-softmax), as a new sampling method that employs Random Fourier Features to tightly approximate the full softmax distribution. RF-softmax falls under the broader class of kernel-based sampling methods which are amenable to efficient implementation. Before presenting RF-softmax, we briefly describe the kernel-based sampling and an existing method based on quadratic kernels [12].

3.1 Kernel-based sampling and Quadratic-softmax

Given a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, the input embedding $\mathbf{h} \in \mathbb{R}^d$, and the class embeddings $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^d$, kernel-based sampling selects the class i with probability $q_i = \frac{K(\mathbf{h}, \mathbf{c}_i)}{\sum_{j=1}^n K(\mathbf{h}, \mathbf{c}_j)}$. Note that if $K(\mathbf{h}, \mathbf{c}_i) = \exp(o_i) = \exp(\tau \mathbf{h}^T \mathbf{c}_i)$, this amounts to directly sampling from the softmax distribution. Blanc and Steffen [12] show that if the kernel can be linearized by a mapping $\phi : \mathbb{R}^d \to \mathbb{R}^D$ such that $K(\mathbf{h}, \mathbf{c}_i) = \phi(\mathbf{h})^T \phi(\mathbf{c}_i)$, sampling one point from the distribution takes only $\mathcal{O}(D \log n)$ time by a divide-and-conquer algorithm. We briefly review the algorithm in this section.

Under the linearization assumption, the sampling distribution takes the following form.

$$q_i = \frac{K(\mathbf{h}, \mathbf{c}_i)}{\sum_{j=1}^n \phi(\mathbf{h})^T \phi(\mathbf{c}_j)} = \frac{\phi(\mathbf{h})^T \phi(\mathbf{c}_i)}{\phi(\mathbf{h})^T \sum_{j=1}^n \phi(\mathbf{c}_j)}.$$

The idea is to organize the classes in a binary tree with individual classes at the leaves. We then sample along a path on this tree recursively until we reach a single class. Each sampling step takes $\mathcal{O}(D)$ time as we can pre-compute $\sum_{j \in S} \phi(\mathbf{c}_j)$ where S is any subset of classes. Similarly, when the embedding of a class changes, the cost of updating all $\sum_{j \in S} \phi(\mathbf{c}_j)$ along the path between the root and this class is again $\mathcal{O}(D \log n)$.

Note that we pre-compute $\sum_{j \in [n]} \phi(\mathbf{c}_j)$ for the root node. Now, suppose the left neighbor and the right neighbor of the root node divide all the classes into two disjoint set S_1 and S_2, respectively. In this case, we pre-compute $\sum_{j \in S_1} \phi(\mathbf{c}_j)$ and $\sum_{j \in S_2} \phi(\mathbf{c}_j)$ for the left neighbor and the right neighbor of the root node, respectively. First, the probability of the sampled class being in S_1 is

$$q_{S_1} = \frac{\phi(\mathbf{h})^T \sum_{j \in S_1} \phi(\mathbf{c}_j)}{\phi(\mathbf{h})^T \sum_{j \in S_1} \phi(\mathbf{c}_j) + \phi(\mathbf{h})^T \sum_{j \in S_2} \phi(\mathbf{c}_j)} \tag{14}$$

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.