TAPAS: Two-pass Approximate Adaptive Sampling for Softmax

Yu Bai* Sally Goldman Li Zhang
yub@stanford.edu sgoldman@google.com liqzhang@google.com

Google Inc.

Abstract

TAPAS is a novel adaptive sampling method for the softmax model. It uses a two pass sampling strategy where the examples used to approximate the gradient of the partition function are first sampled according to a squashed population distribution and then resampled adaptively using the context and current model. We describe an efficient distributed implementation of TAPAS. We show, on both synthetic data and a large real dataset, that TAPAS has low computational overhead and works well for minimizing the rank loss for multi-class classification problems with a very large label space.

1 Introduction

Multi-class classification problems are ubiquitous in machine learning: given empirical observations of pairs of context features $x_i \in \mathcal{X}$ and discrete label $y_i \in [V] = \{1, \ldots, V\}$, we wish to learn to predict the label y for any given x. Many tasks in computer vision, natural language processing, and recommender systems are by nature multi-class problems.

A particularly effective method for the multi-class classification task is to model the conditional probability of Prob$[y|x]$ through a neural network softmax model. In such a model, Prob$[y|x]$ is set to be proportional to $\exp(\phi(x) \cdot \psi(y))$, where ϕ, ψ are parameterized functions that map each context and label to some high dimensional space, called the context embedding and the label embedding, respectively. The model parameters of ϕ, ψ are then learned by minimizing the empirical cross entropy loss using the gradient descent method.

One challenge underlying this approach is that the vocabulary size V can be very large as we apply the method to increasingly larger tasks. For example, ImageNet (Russakovsky et al., 2015) consists of around $10^4 - 10^5$ tags for images. In a language model, the vocabulary of all words and common phrases can have $10^5 - 10^6$ entries. In a video recommendation task, V is the number of videos and is often on the order of $10^7 - 10^9$. Gradient-based training of the softmax model requires computing the partition function $Z(x) = \sum_{z \in [V]} \exp(\phi(x) \cdot \psi(z))$ at every training step. When V is large, computing $Z(x)$ becomes prohibitively expensive.

Sampling based methods, such as importance sampling (also called sampled softmax) (Bengio & Senecal, 2008) and noise contrastive estimation (Gutmann & Hyvärinen, 2012), are common techniques to address this problem. In such methods, at each training step, a small subset of samples of $[V]$ is used to approximate the gradient of $Z(x)$. The effectiveness of the sampling based method crucially depends on the sampling distribution and the sample size. It also has to be done efficiently to avoid large computational overhead. The common approach is to sample according to a pre-determined distribution, usually dependent on the empirical distribution of the labels (Chen et al., 2016; Józefowicz et al., 2016).

In this paper, we propose a **T**wo-pass **Ap**proximate **A**daptive **S**ampling method (TAPAS) for the efficient training of the softmax model with large vocabulary size. In TAPAS, the sampling is done in two passes. In the first non-adaptive pass, we sample a subset $S' \subset [V]$ according a pre-determined distribution, similar to the sampled softmax. In the second adaptive pass, we resample a smaller set S from S' which are "close" to the contexts, i.e. with higher predicted probability for the given contexts, in the training batch. We then use S for computing the gradient updates on the model parameters.

Compared to the existing approaches, TAPAS chooses the samples according to both the context and the current model parameters. The resampling reduces the size of the samples so it is more efficient to compute the gradients.

Another useful view on the sampling is that the samples serve as "negative" labels since the gradient descent would cause the context embedding and the label embeddings of the sampled classes to move away from each other. Indeed, such sampling is also called negative sampling. By focusing on a subset of the samples with higher

*The work was done during an internship at Google.

logits, the training procedure of TAPAS pays more attention to the "hard" negative labels, i.e. the classes likely to be confused with the true label given the context. This leads to more efficient training and better ranking accuracy such as the average precision score, similar to (Weston et al., 2011).

The adaptive sampling, however, does incur computational overhead. To reduce the overhead, we present a distributed approximate sampling algorithm that utilizes both the GPUs and the parallelism supported by the state-of-the-art distributed machine learning platforms. We implemented TAPAS on Tensorflow (Abadi et al., 2016) and show it has very small overhead. We demonstrate the success of TAPAS on both synthetic data and on a large scale real data set.

We provide empirical study of TAPAS in this paper. We conjecture that, similar to the analysis showing that the Warp sampling of Wsabie optimizes precision at k versus optimizing the mean rank (Weston et al., 2011), the adaptive sampling of TAPAS is closer to optimizing a rank loss versus the full softmax loss. However, theoretical analysis proving this conjecture has been surprisingly challenging and is a good direction for future work.

1.1 Related work

In this section we review related work. We list a variety of techniques that have been proposed to address the prohibitive cost of computing the negative gradient (or equivalently computing the partition function and its gradient) when the vocabulary is extremely large. For a good survey, see (Chen et al., 2016).

An important distinction which is relevant to our work is the extent to which the methods depend on the *vocabulary* (e.g. the label frequencies), the *context* (e.g., words before the word to be predicted in a language model) and the *model* (e.g. current weights) itself.

Sampled softmax. TAPAS builds on the idea of sampling-based approximations of the softmax loss and its gradients. These sampled softmax strategies specify a sampling distribution Q from which they draw a subset of the label space $[V]$. Popular sampling distributions include the naive uniform distribution, frequency-based unigram (sample frequency) or bigram distributions (Bengio & Senecal, 2003; 2008), or a power-raised distribution of the unigram (Mikolov et al., 2013; Ji et al., 2015). These distributions are specified beforehand and do not adapt to the training process. TAPAS can be applied on top of any of such sampling schemes to add an adaptive layer to provide harder negatives.

More similar to TAPAS are the many variants of *Adaptive Sampling* that adapt the sampling distribution Q to the model training process. One work closely related to ours is the pioneering method of *Adaptive Importance Sampling* (Bengio & Senecal, 2008). Bengio & Senecal observe that sampling from the exponentiated logits $P(y|x) \propto \exp(\phi(x) \cdot \psi(y))$ will give us unbiased estimates of the full softmax gradient. To overcome the inefficiency of such a distribution, they define an approximate Q using an n-gram model that is a mixture of a set of k-gram models which can be efficiently adapted during training so that fewer examples are needed to approximate the gradient. In addition they introduce effective sample size (ESS) which adaptively selects the size of the sample to use for the negative sampling. An important limitation of their work is that their Q has an n-gram structure that is most appropriate for language models. In contrast, TAPAS utilizes context and model information without imposing structural assumptions so is suitable for more tasks. Also ESS, while using the variance of the prediction probability to determine the sample size, does not subsample it to use harder negatives and thus does not lend itself as well for ranking tasks.

(Jean et al., 2014) introduce sampled softmax for neural machine translation with very large vocabularies where the negative sampling is performed in mini-batches. However, the sampling method does not depend on the current model, which is a key aspect of our work.

Tree-based methods. Hierarchical Softmax (HSM) (Morin & Bengio, 2005; Goodman, 2001) is another popular technique that organizes the labels into a tree where the leaves are the labels and the intermediate nodes are latent variables. The probability of a label is the product of the probabilities of the latent variables along the path from the root to the leaf. This decomposition allows a sequential computation of the probabilities and saves the cost of computing the full partition function $Z(x)$. The most common use of this is a two-level HSM such as in (Mikolov et al., 2013). HSM is most suitable when the labels naturally forms a concept tree such as language models and is able to achieve state-of-the-art perplexities on such tasks (Józefowicz et al., 2016). However, inferring a tree structure for a general-purpose multi-class task might be highly non-trivial, and it is also hard to adjust the tree structure during training.

Efficient implementations. Similar to our work which provides a very efficient implantation on a distributed architecture such as Tensorflow (Abadi et al., 2016), (Grave et al., 2016) introduced an efficient softmax approximation that is appropriate for distribution on GPUs. They define a strategy to produce an approximate hierarchical model that is well suited to efficient computation

by GPUs. Again, here the sampling does not depend on the current model.

Other loss functions. Many other loss functions prove successful in multi-class problems. One alternative approach is Noise Contrastive Estimation (NCE) (Gutmann & Hyvärinen, 2012). These methods do not compute the negative gradient but instead learn to discriminate between true labels and samples from a noise distribution. It essentially relates a multi-class problem to a binary problem. This is very suitable in a multi-label scenario, i.e. each context having multiple true labels. Another approach are *Infrequent Normalization (Self Normalization)* that perform infrequent updates of the negative gradient (Andreas & Klein, 2015).

Although a very different direction, there is an interesting relationship between Wsabie (Weston et al., 2011) that uses stochastic gradient descent to optimize a ranking loss. The interesting aspect about Wsabie is that the selection of the negative samples is very tightly linked to the current model and it has been shown to improve the loss of the top ranked items as compared to optimizing the AUC. We show that the second phase of TAPAS achieves a similar goal.

1.2 Outline

The rest of this paper is organized as follows. In Section 2 we review preliminaries on softmax regression, sampled softmax, rank losses, and motivate adaptive sampling strategies. Section 3 describes the TAPAS algorithm in detail. We further discuss issues on its computational cost and provide an efficient implementation in Section 4. We demonstrate the success of TAPAS on synthetic datasets and a large-scale real dataset in Section 5.

2 Preliminaries

Notation. We use $[V]$ to denote the set $\{1, 2, \cdots, V\}$. Given a finite set S and a function $f : S \to \mathbb{R}$, we use $\operatorname{argmax}^n_{x \in S} f(x)$ to denote the n element subset of S that has the largest f value. For two vectors $u, v \in \mathbb{R}^d$, let $u \cdot v$ denote their dot product.

Multi-class classification. We consider the multi-class classification problem of predicting the label[1] given a context where the label comes from vocabulary set $[V]$. One classical example is the language model where we predict a word from the context surrounding the word in a sentence. It can also be used to model a recommendation system where the context represents the user features, such as demographic information and the past user activities, and each label represents an item, for example a song or a video, that the user might like.

Softmax regression. In the softmax regression, each context x is mapped to a real feature vector $\phi(x) \in \mathbb{R}^d$, and each label y is mapped to $\psi(y) \in \mathbb{R}^d$. Here ϕ is a neural network with multiple (non-linear) layers, and ψ maps each id to a vector in $\mathbb{R}^d$. We model the conditional probability $\text{Prob}[y|x]$ as $\text{Prob}[y|x] \propto \exp(\phi(x) \cdot \psi(y))$, i.e.

$$\text{Prob}[y|x] = \exp(\phi(x) \cdot \psi(y))/Z_{\phi,\psi}(x)\,,$$

where $Z_{\phi,\psi}(x) = \sum_{z \in [V]} \exp(\phi(x) \cdot \psi(z))$ is the *partition function* at x. We omit ϕ, ψ from the notation when it is clear from the context. Given the data set D containing empirical observation of (x, y) pairs, the cross-entropy loss (or softmax loss) of a model (ϕ, ψ) is defined as

$$L(\phi, \psi) = \sum_{(x,y) \in D} -\log \text{Prob}[y|x]\,.$$

The model parameters are then learned by minimizing the the above loss L over (ϕ, ψ) using the stochastic gradient descent method. The point-wise gradient of the loss on an example $(x, y) \in D$ is:

$$\begin{aligned}
&\nabla_\theta(-\log \text{Prob}[y|x]) \\
=&\nabla_\theta[-(\phi(x) \cdot \psi(y)) + \log \sum_{z \in [V]} \exp(\phi(x) \cdot \psi(z))] \\
=&-\nabla_\theta(\phi(x) \cdot \psi(y)) + \sum_{z \in [V]} \text{Prob}[z|x] \nabla_\theta(\phi(x) \cdot \psi(z))\,.
\end{aligned}$$

Applying to the context and the label embeddings, respectively, we have

$$\nabla_{\phi(x)} = -\psi(y) + \sum_{z \in [V]} \text{Prob}[z|x]\psi(z)\,, \qquad (1)$$

$$\nabla_{\psi(z)} = (-\delta_{yz} + \text{Prob}[z|x])\phi(x)\,. \qquad (2)$$

Here δ_{yz} denotes the Kronecker delta. By stochastic gradient descent, we compute the mean of the point-wise gradient on a random batch of examples and then apply the gradient descent.

With the trained model, the inference is done by computing

$$\widehat{y}(x) = \operatorname{argmax}_{y \in [V]}\{\phi(x) \cdot \psi(y)\}\,.$$

It is common to compute the top k labels if multiple candidates are allowed.

[1] All the discussion in the paper directly generalizes to the case when each context may receive multiple labels or a distribution of labels by using the cross-entropy loss with respect to the distribution of label classes. For the simplicity of presentation, we focus on the case when there is a single label for each context.

Sampled softmax. By (1), computing the gradient on $\phi(x)$ requires to compute $\sum_{z\in[V]} \text{Prob}[z|x]\psi(z)$. This computation can be prohibitively expensive for a large V. One solution is to sample a subset $S \subset [V]$ to approximate $\nabla_{\phi(x)}$. In sampled softmax, each label z in $[V]$ is assigned a probability q_z for being selected in the sample. At each training step, a random subset S of $[V]$ is sampled according to q, and the subset is used to approximate ∇L. For computing q_z, one popular method is to use squashed empirical frequency. Suppose f_z is the empirical frequency of the class z. We set $p_z \propto \max(f_z^\alpha, \beta)$ where $0 \le \alpha \le 1$ is a squash exponent, and $\beta > 0$ is a lower bound to guarantee a non-vanishing sampling. Sampled softmax and its variants have shown to produce the best accuracy for many multi-class classification problems (Józefowicz et al., 2016; Chen et al., 2016). Typically the sampled softmax uses the same sampling distribution for all the contexts. In (Bengio & Senecal, 2008), it is suggested to use adaptive sampling according to a separate model which makes rough estimation of the label probability.

From (1), we can also see that the gradient descent would move $\phi(x)$ towards $\psi(y)$ but away from $\psi(z)$ for $z \neq y$. This is also why such sampling is commonly called negative sampling as the sampled z has the effect similar to a negative class label.

Rank loss. While the softmax loss is smooth and suitable for minimization using gradient descent, in practice, the rank loss is often used for evaluating the model quality since usually only top model predictions are relevant to the applications. There are various variants of rank losses. In this paper we consider the common metrics of the precision and the mean average precision (MAP) metrics (KaggleWiki), defined as follows. Suppose that the model produces a ranked list of predictions $z_1, z_2, \cdots$, and the true labels is a set $Y = \{y_1, \cdots, y_m\}$. Write $Z_k = z_1, z_2, \cdots, z_k$. Then precision@k is defined as $|Z_k \cap Y|/k$, the fraction of true labels among the top k model predictions, and MAP@k is defined as the average of the precision@k' for each position $k' \le k$ where $z_{k'} \in Y$.

While the softmax loss is a good surrogate to the rank loss (Zhang, 2004), they are not identical. Indeed, the main contribution of this paper is to design an efficient adaptive sampling method for softmax model which achieves low rank loss (but not necessarily softmax loss).

3 Algorithm

Our sampling algorithm works with the mini-batch stochastic gradient descent method and is carried out for each batch. Consider a batch $B = \{(x_i, y_i)\}$ of training examples. Algorithm 1 describes the two pass sampling algorithm. In the first pass, we sample a subset S' using the sampling distribution Q just like in the sampled softmax, and in the second pass we resample $S \subset S'$ adaptively dependent on the batch B and the current model parameters. We then use S for computing the gradient as done in sampled softmax.

Algorithm 1 Two-pass adaptive sampling algorithm.

Parameters: n: number of samples, r: presmaple factor, τ: sampling temperature.
Input: A batch B.
Output: $S \subset V$ where $|S| = n$.
Non-adaptive pass
Sample $\min(r \cdot n, |V|)$ classes S' according to sampling distribution Q.
Adaptive pass
Compute and return
$S = \text{argmax}^n_{y \in S'} \sum_{i \in B} \exp(\phi(x_i) \cdot \psi(y))/\tau)$.

The algorithm takes three parameters, where n is the number of output samples, $r \ge 1$ is the pre-sample factor, and τ is the sampling temperature. Note that when $r = 1$, TAPAS is identical to sampled softmax.

3.1 Non-adaptive sampling pass

The first pass is non-adaptive sampling and can use any existing sampling method Q. In our implementation, we use the standard squashed empirical distribution as described in Section 2. The main purpose of the non-adaptive sampling pass is to obtain a sample S' with smaller size so the adaptive pass can be done efficiently. The sampling also helps to reduce over-fitting as explained later. In our experience, it seems a good tradeoff by choosing the size of S' in the order of 1% to 10% of the total number of classes.

3.2 Adaptive sampling pass

In the adaptive sampling pass, a smaller set of samples S are chosen from S' dependent on $\{x_i\}_{i \in B}$ and the current model parameters ϕ, ψ. Intuitively we choose the samples that are "close" to x_i, i.e. those classes y such that $\phi(x_i) \cdot \psi(y)$ is large. For efficiency, the sampling is done at the batch level. Intuitively we include a label y in S' if it is close to x_i for some $i \in B$. We use temperature τ to control the adaptivity — when τ is smaller, the sampling is more adaptive as there is an increasing chance for y to be selected if it is close to any of x_i. During the training, we reduce the temperature over time. This is similar to the intuition of decreased temperature in training softmax model (Cesa-Bianchi & Fischer, 1998).

The adaptive sampling pass has a few benefits. First, it reduces the sample size further by a factor of r. This reduces the gradient computation cost significantly, and with the adaptive sampling, hopefully does not lose much accuracy on the gradient estimation. Secondly, by using only the examples that are close to the context, we focus on the "hard negatives" and can obtain lower rank loss, similar to the intuition in Wsabie (Weston et al., 2011). Thirdly, since the gradient update is on a smaller set of classes, there is less chance for the gradient to become stale, which is useful with asynchronous training. Since the adaptive sampling deterministically chooses the n classes close to the context, the first pass is important for introducing randomness into the sampling. Otherwise it may cause over-fitting of the model. For example, at the extreme, if we always choose the top n labels from the entire V, it would cause the model not to generalize well as it may "push away" the correct labels. The adaptive sampling, while reducing the rank loss, actually leads to higher cross-entropy loss. This is due to that the adaptive sampling skews the sampling distribution and hence results in a more biased gradient estimation. However, in our algorithm, we intentionally does not correct for this skewness since we would like to emphasize on the "hard negatives" to improve the rank loss.

实际上使用适应性采样会导致整体有更高的交叉熵。算法本身的目的还是在于降低 Rank Loss

The adaptive sampling does come with a price. It requires to compute the dot product $\phi(x_i) \cdot \psi(y)$ for all the pairs $i \in B$ and $y \in S'$. This computation can be expensive if the size of S' is large. In the following, we describe a distributed approximation to Algorithm 1 that utilizes GPUs and the parallelism supported by the distributed machine learning systems. With our implementation, we show that TAPAS incurs a very low overhead.

4 Implementation

We implemented TAPAS on Tensorflow (Abadi et al., 2016), but the same algorithm can be easily adapted to the other distributed machine learning platforms such as (Collobert et al., 2011; Bergstra et al., 2010). In Tensorflow, the machines are organized as *workers* and *parameter servers* where the parameter server hosts the parameters such as the embeddings (in our case $\psi(y)$ for $y \in [V]$) and neural network parameters, and the worker performs the gradient computation and parameter update by communicating with the parameter server. In the typical setup, the parameter servers are hosted on CPUs, and the workers on GPUs for large scale training.

In the adaptive sampling pass, we need to compute $\text{argmax}^n_{y \in S'} \sum_{i \in B} \exp(\phi(x_i) \cdot \psi(y))/\tau)$, which in turn requires to compute $\phi(x_i) \cdot \psi(y)$ for each $i \in B$ and $y \in S'$. If we use the standard setup of "sample at worker," then we would need to fetch the parameters

Algorithm 2 TAPAS.

Worker
  Compute $\phi(x_i)$ for each $i \in B$.
  Sample S' non-adaptively.
  for $j = 1, \cdots, m$ **do**
    Compute $S'_j = S' \cap V_j$
    Send $\{\phi(x_i)\}_{i \in B}$ and S'_j to parameter server j.
  end for
PS$_j$
  Compute
    $S_j = \text{argmax}^{n/m}_{y \in S'_j} \sum_{i \in B} \exp(\phi(x_i) \cdot \psi(y))/\tau)$.
  Send S_j and corresponding parameters to the worker.
Worker
  Compute the gradient with negative samples $S = \cup_{j=1}^m S_j$.
  Update the parameters using the gradients.

$\psi(y)$ for $y \in S'$ from the parameter server to the worker and performs the sampling at the worker. When S' is large, such method would incur large network communication and cause significant slowdown of the training. There is then less benefit from adaptive sampling. Instead, in our implementation, we take the "sample at the parameter server" approach by hosting the parameter servers on the GPUs and sampling on the parameter servers. In addition, to facilitate efficient distributed sampling, we only approximately sample the top n elements. More specifically, suppose there are m parameter servers, and parameter server PS$_j$ hosts the embeddings of a random subset $V_j \subseteq V$. After the pre-sample S' is obtained, PS$_j$ will only look at $S'_j = S' \cap V_j$ and select the top n/m negative samples S_j. The union of S_j will be an approximate top n choice from S'. Our implementation is described in Algorithm 2.

In the implementation, $\{\phi(x_i)\}_{i \in B}$ is broadcast to all the parameter servers. Compared to sampling at the worker, this does incur some cost. On the other hand, only those samples in S are sent from the parameter servers to the worker. Since the size of S' (in the order of $100,000$) is typically much larger than the size of B (in the order of 1000), Algorithm 2 has significantly lower communication cost compared to sampling at the worker. In addition, the sampling are distributed over the parameter servers and can be done efficiently. One additional optimization is that we do not compute the exact top n elements in S'. Instead, we take the union of the top n/m elements from each of m parameter servers. So what we obtain is the approximate top n samples. However, when the labels are randomly partitioned, the approximation is fairly good as n (in the order of $10,000$) is much larger than m (in the order of 10 to 100).

n: number of resampled
r: 预采样因子，预采样数=n*r

精度与预采样数有很强的依赖，同预采样数的情况下，TAPAS和Sampled Softmax的精度很接近，然而TAPAS的速度快r倍

With the above implementation, the overhead of TAPAS is very small. In our experiments, we do not observe much performance difference of TAPAS with n samples for $r = 1$ (same as the sampled softmax) and for $r = 10$. But the rank loss for $r = 10$ is significantly lower than $r = 1$, hence giving us a significant quality gain with similar training time.

5 Experiments

5.1 Synthetic data

We perform experiments on synthetic multi-class classification tasks to test TAPAS on various combinations of parameters. We begin by learning a linear classifier to approximate a standard Gaussian mixture with 1000 classes. We then experiment with a large-scale synthetic dataset with 10000 classes where the data is generated by a non-linear model. The second setting seeks to imitate real-world modeling tasks.

5.1.1 Linear Classifiers

Data. Our first experiment train linear classifiers on a Gaussian mixture dataset. We generate $V = 1000$ random centroids $\mu_j \in \mathbb{R}^d$ from a Gaussian prior $\mathsf{N}(0, \frac{c^2}{d} I_d)$, where we set $d = 50$ and $c = 3$. To generate data point i, we uniformly choose a label $y_i \in [V]$ and generate the associated position x_i from the Gaussian distribution $\mathsf{N}(\mu_{y_i}, I_d)$. The goal is to learn a classifier that predicts y given x.

Model. Under this generative model, the true posterior of Prob$[y|x]$ has an exact softmax form

$$\text{Prob}[y = j|x] = \frac{\exp(w_j \cdot x + b_j)}{\sum_{j=1}^{V} \exp(w_j \cdot x + b_j)},$$

where $w_j = \frac{\mu_j}{\sigma^2}$ and $b_j = -\frac{\|\mu_j\|^2}{2\sigma^2}$. For this simple problem, we can compute the sample mean $\widehat{\mu}_j = \frac{1}{|\{i:y_i=j\}|} \sum_{i:y_i=j} x_i$ and use it to compute w_j, b_j. With a moderate number of samples, we could achieve accuracy close to the information theoretical bound. However, that requires seeing the whole dataset. Here we are interested in evaluating the performance of stochastic gradient based methods. We note even in this simple setting, the convergence properties of the sampled softmax are not fully understood.

In all the experiments we generate the training and test sets of size 10^5 and 10^4, respectively, as described above. The model is a linear softmax model. We use the AdaGrad optimizer (Duchi et al., 2011) with mini-batches of size $B = 16$. We use precision at 1 to measure the quality of each experiment. We carry out two sets of experiments, the first varying the number of samples n, and the second varying the pre-sample factor r.

(1) Fix $n = 16$ and let $r = 1, 2, 4, 8$.

(2) Fix pre-sample size $n \times r = 128$, and let $n = 16, 32, 64, 128$.

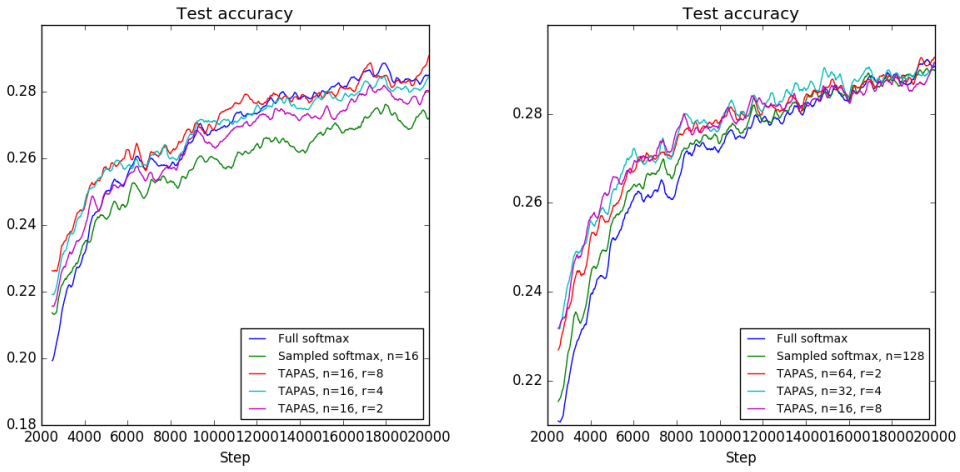
We also run the full softmax computation for comparison.

Results. Figure 1 plots the test prediction accuracy of the classifiers of the two sets of experiments. Since the linear problem is relatively easy, we do not observe big differences between different experiments. For example, there is less than 10% difference between the full softmax and the sampled softmax with $n = 16$. However, even for such a simple problem, there is visible quality difference with different sampling strategies. In Figure 1(a), we can see for fixed n, increasing r increases the accuracy, especially for smaller r's. From Figure 1(b) we observe that the accuracy of different experiments are getting close over time, verifying that the accuracy has strong dependency on $n \cdot r$ so using TAPAS with n samples and pre-sample factor r performs similarly to the sampled softmax with $n \cdot r$ samples. However, the latter is much more expensive to run as it needs r times more samples in the gradient computation.

5.1.2 Neural networks

We create a more challenging task by adding more classes and using a non-linear generative model.

Data. We use a non-linear generative model to generate the data points as follows. First, generate $V = 10000$ "centroids" $\mu_j \in \mathbb{R}^{d_c}$ similar to Section 5.1.1. To generate a data point x_i, we first choose the label $y_i \in [V]$ uniformly at random and form a input vector $\widetilde{x}_i = [\mu_{y_i}, z_i] \in \mathbb{R}^{d_c + d_n}$, where $z_i \sim \mathsf{N}(0, \sigma^2 I_{d_n})$ is a random Gaussian. This vector is then passed through a neural network to get x_i. The generative network has two layers where the first layer is a d_h-unit layer with ReLU activation and the second layer is a linear layer with d units. Both layers are fully connected. This network imitates the generative neural networks (Goodfellow et al., 2014). The weights of the generative networks are random Gaussians with proper scaling. We use parameters $d_c = d_n = 10$, $d_h = 50$, and $d = 25$. For illustration, we plot two dimensional examples generated by such a method in Figure 2(a). As can be seen, the generated clusters have significant overlapping and are more challenging to classify.



(a) (b)

Figure 1: Results on linear classifiers. Left: fixing $n = 16$, varying r. Right: fixing $nr = 128$, varying (n, r). We add a ℓ_2-regularization of level 0.001 in all cases to prevent over-fitting. We truncate the first 2500 steps and smooth the curves by a moving average filter.

Model. The classifier is a neural network with one 50-dimensional hidden layer, so both x and y gets embedded in 50 dimensional space.

We create a train set of size 10^6 and test set of size 10^5. We carry out the experiments with similar set up as in Section 5.1.1 with slightly larger parameters: the batch size is set to 32, and for the first set of experiments $n = 64$ and for the second set $n \cdot r = 512$.

Results. Figure 2 shows the results of the two set of experiments. For fixed n as shown in Figure 2(b), we observe the similar phenomenon that the accuracy increases when we increase r, except that the effect is much more visible in this more challenging task. In Figure 2(c), with large sample size, there is a gain of accuracy. However the accuracies of $n = 512, r = 1$ and $n = 128, r = 4$ are almost the same, which still shows a significant gain as the latter is 4 times faster in the gradient computation.

5.2 Real data

We apply TAPAS to a large scale classification problem with a real data set. The data set consists of sequences of users' consumption of items on a popular video site. Our training data consists of 200 million sequences with average length of 30, and the item comes from a dictionary of size $500,000$. We build a softmax sequence model for predicting the next five items in the sequence from the prefix of the sequence. The scoring is done using the mean average precision (MAP) at 20. The testing is done on 20 million holdout sequences. This is a fairly challenging task as the items have a rather long tail distribution, for example, the top 20% most popular items only occurs about 60% of times. Our model is a complex neural network model that achieved the highest precision result compared to multiple internal implementations of the state-of-the-art methods.

In our experiments, we fix the model architecture and training hyper-parameters but vary the pre-sample factor and the number of samples. We experiment the combination of $n = 1000, 8000$ and $r = 1, 8$. In all the experiments, the training is done using 6 workers and 6 parameter servers, all hosted on GPUs.

Table 1: Number of steps per second.

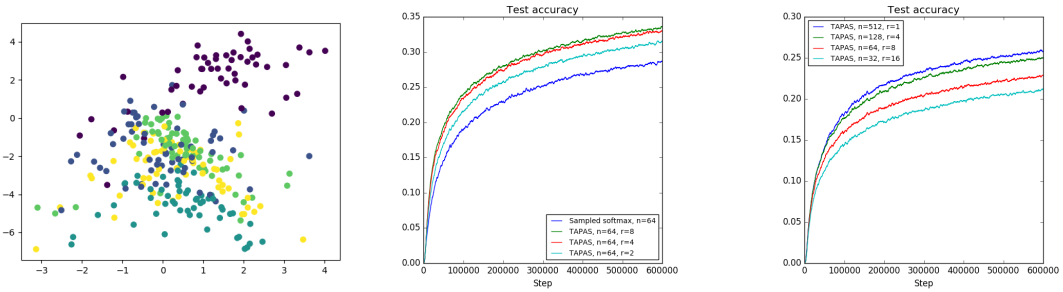
Experiment	Steps/Sec
$n = 1000, r = 1$	181
$n = 1000, r = 8$	173
$n = 8000, r = 1$	92
$n = 8000, r = 8$	81

n*r=8000

Efficiency. Table 1 shows the number of training steps per second for each experiment. We observe that for the same n, increasing r from 1 to 8 only causes a small, about 10%, overhead. Increasing the value n however does slow down the training significantly.

MAP score. Figure 3 shows the MAP@20 scores. As can be seen from the plot, with TAPAS, the MAP score

n一样的情况下，提高r并没有增加多少耗时，但是效果提升不少



(a) (b) (c)

Figure 2: Results on two-layer networks. Left: an illustration of the generative model in two dimensions with 300 points, 5 classes, and generative hidden dimensions $(d_c, d_n, d_h) = (5, 5, 10)$. Middle and right: test accuracies of classifiers trained with TAPAS. Middle: fixing $n = 64$, varying r. Right: fixing $nr = 512$, varying (n, r). We smooth the curves by a moving average filter.

is greatly improved. When $n = 1000$, the MAP score is improved from 0.050 to 0.068 by increasing r from 1 to 8, representing almost 30% improvement, and when $n = 8000$, the improvement is smaller, but still about 12% from 0.067 to 0.075. Interestingly, TAPAS with $n = 1000, r = 8$ has slightly higher accuracy than $n = 8000, r = 1$.

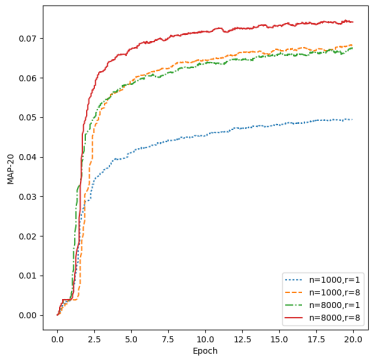


Figure 3: Results on MAP score. n: number of samples; r: pre-sample factor.

Softmax loss. As we described earlier, TAPAS suits well on rank loss but it may not work so well on full softmax loss. Indeed, Figure 4 shows that while the model trained with $n = 8000, r = 8$ has much higher MAP20 score than the combination of $n = 1000, r = 1$ (0.075 vs 0.050), it actually has a slightly higher full softmax loss (9.34 vs 9.10).

These experiments demonstrate that TAPAS can make large improvement on the rank loss with fairly low overhead. On the other hand, it does not improve on the full softmax loss.

虽然n=8，r=8的MAP结果更好，但是softmax loss却更高

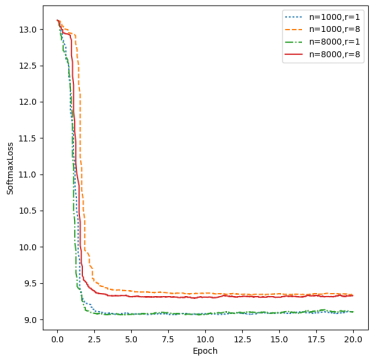


Figure 4: Results on softmax loss. n: number of samples; r: pre-sample factor.

6 Conclusion

We present TAPAS as an effective sampling strategy for softmax model. We have implemented the algorithm and plan to release the code for open-source use. We are particularly thrilled by its effectiveness on the rank loss. It remains an interesting question to establish a formal connection between TAPAS sampling strategy and the rank loss, probably under suitable assumptions.

Acknowledgments

We would like to thank Zhifeng Chen for the idea of co-locating sampling with the parameter sever and the help to make it work; Kunal Talwar for useful discus-

sion; Chris Colby and George Roumpos for the open source tensorflow implementation; and Walid Krichene for many useful comments on the paper.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Gregory S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian J., Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Józefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek Gordon, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul A., Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda B., Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695, 2016.

Andreas, Jacob and Klein, Dan. When and why are log-linear models self-normalizing? In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 244–249, 2015.

Bengio, Yoshua and Senecal, Jean-Sébastien. Quick training of probabilistic neural nets by importance sampling. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics, AISTATS 2003, Key West, Florida, USA, January 3-6, 2003*, 2003.

Bengio, Yoshua and Senecal, Jean-Sébastien. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Trans. Neural Networks*, 19(4):713–722, 2008.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010. Oral Presentation.

Cesa-Bianchi, Nicolo and Fischer, Paul. Finite-time regret bounds for the multiarmed bandit problem. In *ICML*, pp. 100–108, 1998.

Chen, Wenlin, Grangier, David, and Auli, Michael. Strategies for training large vocabulary neural language models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Goodman, Joshua. Classes for fast maximum entropy training. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2001, 7-11 May, 2001, Salt Palace Convention Center, Salt Lake City, Utah, USA, Proceedings*, pp. 561–564, 2001.

Grave, Edouard, Joulin, Armand, Cissé, Moustapha, Grangier, David, and Jégou, Hervé. Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309, 2016.

Gutmann, Michael and Hyvärinen, Aapo. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, 2012.

Jean, Sébastien, Cho, Kyunghyun, Memisevic, Roland, and Bengio, Yoshua. On using very large target vocabulary for neural machine translation. *CoRR*, abs/1412.2007, 2014.

Ji, Shihao, Vishwanathan, S. V. N., Satish, Nadathur, Anderson, Michael J., and Dubey, Pradeep. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *CoRR*, abs/1511.06909, 2015.

Józefowicz, Rafal, Vinyals, Oriol, Schuster, Mike, Shazeer, Noam, and Wu, Yonghui. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016. URL `http://arxiv.org/abs/1602.02410`.

KaggleWiki. Mean average precision. https://www.kaggle.com/wiki/MeanAveragePrecision.

Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

Morin, Frederic and Bengio, Yoshua. Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, AISTATS 2005, Bridgetown, Barbados, January 6-8, 2005*, 2005.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael S., Berg, Alexander C., and Li, Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Weston, Jason, Bengio, Samy, and Usunier, Nicolas. WSABIE: scaling up to large vocabulary image annotation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2764–2770, 2011.

Zhang, Tong. Statistical analysis of some multi-category large margin classification methods. *Journal of Machine Learning Research*, 5(Oct):1225–1251, 2004.