
Biased Importance Sampling for Deep Neural Network Training

Angelos Katharopoulos François Fleuret

Idiap Research Institute, Martigny, Switzerland

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

{name.surname}@idiap.ch

Abstract

Importance sampling has been successfully used to accelerate stochastic optimization in many convex problems. However, the lack of an efficient way to calculate the importance still hinders its application to Deep Learning.

In this paper, we show that the loss value can be used as an alternative importance metric, and propose a way to efficiently approximate it for a deep model, using a small model trained for that purpose in parallel.

This method allows in particular to utilize a biased gradient estimate that implicitly optimizes a soft max-loss, and leads to better generalization performance. While such method suffers from a prohibitively high variance of the gradient estimate when using a standard stochastic optimizer, we show that when it is combined with our sampling mechanism, it results in a reliable procedure.

We showcase the generality of our method by testing it on both image classification and language modeling tasks using deep convolutional and recurrent neural networks. In particular, in case of CIFAR10 we reach 10% classification error 50 epochs faster than when using uniform sampling.

1 Introduction

The dramatic increase in available training data has made the use of Deep Neural Networks feasible, which in turn has significantly improved the state-of-the-art in many fields, in particular Computer Vision and Natural Language Processing. Due to the complexity of the resulting optimization problem, computational cost is now the core issue in training these large architectures.

When training such a model, it appears to any practitioner that not all samples are equally important; many of them are properly handled after a few epochs of training, and most could be ignored at that point without impacting the resulting final model.

For convex optimization problems, many works [Bordes et al., 2005, Zhao and Zhang, 2015, Needell et al., 2014, Canévet et al., 2016, Richtárik and Takáč, 2013] have taken advantage of the difference in importance among the samples to improve the convergence speed of stochastic optimization methods. On the contrary, important sampling is rarely used in conjunction with Deep Learning models.

Zhao and Zhang [2015] and Alain et al. [2015] prove that sampling according to the gradient norm minimizes the variance of the gradient estimates and improves the convergence speed of SGD.

However, computing the gradient norm requires to compute second-order quantities during the backward pass, which is computationally prohibitive. We propose to use the loss instead, but since computing it still requires a full forward pass, using it directly on all the samples remains intractable. So to reduce computation even more, we propose to use the prediction of a small network, trained alongside the deep model we want to eventually train, to predict an approximation of the importance

of the training samples. The complexity of this surrogate allows us to modulate the cost/accuracy trade-off.

Finally, we show a relationship between importance sampling and maximum-loss minimization, which can be used to improve the generalization ability of the trained Deep Neural Network. We evaluate the proposed method both on image and text datasets.

In summary, the contributions of this work are:

- The use of the loss instead of the gradient norm to estimate the importance of a sample
- The creation of a model able to approximate the loss for a low computational overhead
- The development of an online algorithm that minimizes a soft max-loss in the training set through importance sampling

2 Related Work

Importance sampling for convex problems has received significant attention over the years. [Bordes et al. \[2005\]](#) developed LASVM, which is an online algorithm that uses importance sampling to train kernelized support vector machines. Later [Needell et al. \[2014\]](#) and more recently [Zhao and Zhang \[2015\]](#) developed more general importance sampling methods that improve the convergence of Stochastic Gradient Descent. In particular, the latter has crucially connected the convergence speed of SGD with the variance of the gradient estimator and has shown that the target sampling distribution is the one that minimizes this variance.

[Alain et al. \[2015\]](#) are the first ones, to our knowledge, that have attempted to use importance sampling for training Deep Neural Networks. They sample according to the exact gradient norm as computed by a cluster of GPU workers. Even with a cluster of GPUs they have to constrain the networks that they use to fully connected layers in order to be able to compute the gradient norm in a reasonable time.

[Canévet et al. \[2016\]](#) worked on improving the sampling procedure for importance sampling. They imposed a prior tree structure on the weights, and use a sampling procedure inspired by the Monte Carlo Tree Search algorithm, that handles properly the exploration / exploitation dilemma and converges asymptotically to the probability distribution of the weights. However, this method fully relies on the existence of the tree structure, that should reflect the regularity of the importance on the samples, which is in itself a quite complicated embedding problem.

Finally, there is another class of methods related to importance sampling that can be perceived as using an importance metric quite antithetical to most common methods. Curriculum learning [\[Bengio et al., 2009\]](#) and its evolution self-paced learning [\[Kumar et al., 2010\]](#) present the classifier with easy samples first (samples that are likely to have a small loss) and gradually introduce harder and harder samples.

3 Importance Sampling

It has already been mentioned that *Importance sampling* aims at increasing the convergence speed of SGD by reducing the variance of the gradient estimates. In the following sections, we analyze how it works and present an efficient procedure that can be used to train a Deep Learning model.

Moreover, we show that our importance sampling method has a close relation to maximum loss minimization: by using an exponentiation of the loss as importance instead of the loss itself, we can minimize a smooth max-loss, while controlling the variance.

3.1 Exact Importance Sampling

Let x_i, y_i be the i -th input-output pair from the training set, $\Psi(\cdot; \theta)$ a Deep Learning model parameterized by the vector θ , and $L(\cdot, \cdot)$ the loss function to minimize during training. The goal of training is to find

$$\theta^* = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(\Psi(x_i; \theta), y_i) \quad (1)$$

where N corresponds to the number of examples in the training set. Using Stochastic Gradient Descent with learning rate η , we iteratively update the parameters of our model, between two consecutive iterations t and $t + 1$, with

$$\theta_{t+1} = \theta_t - \eta \alpha_i \nabla_{\theta_t} L(\Psi(x_i; \theta_t), y_i) \quad (2)$$

where i is a discrete random variable sampled according to a distribution P with probabilities p_i and α_i is a sample weight. For instance, plain SGD with uniform sampling is achieved with $\alpha_i = 1$ and $p_i = \frac{1}{N}$ for all i .

If we define the convergence speed S of SGD as the reduction of the distance of the parameter vector θ from the optimal parameter vector θ^* in two consecutive iterations t and $t + 1$

$$S = -\mathbb{E}_P \left[\|\theta_{t+1} - \theta^*\|_2^2 - \|\theta_t - \theta^*\|_2^2 \right], \quad (3)$$

and if we have

$$\mathbb{E}_P [\alpha_i \nabla_{\theta_t} L(\Psi(x_i; \theta_t), y_i)] = \nabla_{\theta_t} \frac{1}{N} \sum_{i=1}^N L(\Psi(x_i; \theta_t), y_i), \quad (4)$$

and define $G_i = \alpha_i \nabla_{\theta_t} L(\Psi(x_i; \theta_t), y_i)$, then we get (this is a different derivation of the result by Wang et al. [2016])

$$S = -\mathbb{E}_P \left[(\theta_{t+1} - \theta^*)^T (\theta_{t+1} - \theta^*) - (\theta_t - \theta^*)^T (\theta_t - \theta^*) \right] \quad (5)$$

$$= -\mathbb{E}_P \left[\theta_{t+1}^T \theta_{t+1} - 2\theta_{t+1}^T \theta^* + (\theta^*)^T \theta^* - \theta_t^T \theta_t + 2\theta_t^T \theta^* - (\theta^*)^T \theta^* \right] \quad (6)$$

$$= -\mathbb{E}_P \left[(\theta_t - \eta G_i)^T (\theta_t - \eta G_i) + 2\eta G_i^T \theta^* - \theta_t^T \theta_t \right] \quad (7)$$

$$= -\mathbb{E}_P \left[-2\eta (\theta_t - \theta^*)^T G_i + \eta^2 G_i^T G_i \right] \quad (8)$$

$$= 2\eta (\theta_t - \theta^*)^T \mathbb{E}_P [G_i] - \eta^2 \mathbb{E}_P [G_i^T G_i] - \eta^2 \text{Tr}(\mathbb{V}_P [G_i]). \quad (9)$$

With this last expression, we observe that it is possible to gain a speedup by sampling from the distribution that minimizes $\text{Tr}(\mathbb{V}_P [G_i])$. Alain et al. [2015] and Zhao and Zhang [2015] show that this distribution has probabilities $p_i \propto \|\nabla_{\theta_t} L(\Psi(x_i; \theta_t), y_i)\|_2$. However, computing the norm of the gradient for each sample is computationally intensive. Alain et al. [2015] use a distributed cluster of workers and constrain their models to fully connected networks while Zhao and Zhang [2015] only consider convex problems and sample according to the Lipschitz constant of the loss of each sample, which is an upper bound of the gradient norm.

To mitigate the computational requirement, we propose to use the loss itself as the importance metric instead of the gradient norm.

Although providing the network with a larger number of confusing examples makes intuitive sense, we also provide theoretical arguments and empirical evidence that in practice the ordering of samples according to the gradient norm is reflected by their ordering according to the loss, and that this is sufficient for the loss-based importance sampling to exhibit the properties of the gradient-norm importance sampling discussed above, and perform better than the uniform sampling, as shown in the experimental results.

To sum up, we propose the following importance sampling scheme that creates an unbiased estimator of the gradient vector of Batch Gradient Descent with lower variance:

$$p_i \propto L(\Psi(x_i; \theta_t), y_i) \quad (10)$$

$$\alpha_i = \frac{1}{N p_i}. \quad (11)$$

3.2 Relation to Max Loss Minimization

Minimizing the average loss over the training set does not necessarily result in the best model for classification. Shalev-Shwartz and Wexler [2016] argue that minimizing the maximum loss can lead to better generalization performance, especially if there exist a few “rare” samples in the training set.

In this section, we show that introducing a minor bias inducing modification in the sample weights α_i , we are able to focus on the high-loss samples with a variable intensity up to the point of minimizing the maximum loss.

Instead of choosing the sample weights α_i such that we get an unbiased estimator of the gradient, we define them according to

$$\alpha_i = \frac{1}{N p_i^k}, \text{ with } k \in (-\infty, 1], \quad (12)$$

and with $L_i = L(\Psi(x_i; \theta), y_i)$ and $p_i \propto L_i$, we get

$$\mathbb{E}_P [\alpha_i \nabla_\theta L_i] = \sum_{i=1}^N p_i \alpha_i \nabla_\theta L_i \quad (13)$$

$$= \sum_{i=1}^N \frac{p_i^{1-k}}{N} \nabla_\theta L_i \quad (14)$$

$$= C \sum_{i=1}^N \frac{L_i^{1-k}}{N} \nabla_\theta L_i \quad (15)$$

$$= C' \sum_{i=1}^N \frac{1}{N} \nabla_\theta L_i^{2-k}. \quad (16)$$

We show that we have an unbiased estimator of the gradient of the original loss function raised to the power $2 - k \geq 1$. When $2 - k \gg 1$ we are essentially minimizing the maximum loss but as it will be analyzed in the experiments, smaller values can be helpful both in increasing the convergence speed and improving the generalization error.

3.3 Approximate Importance Sampling

Although by using the loss instead of the gradient norm, we simplify the problem and make it straightforward for use with Deep Learning models, calculating the loss for a portion of the training set is still prohibitively resource intensive. To alleviate this problem and make importance sampling practical we propose approximating the loss with another model, which we train alongside our Deep Learning model.

Let $\mathcal{H}_t = \{(j, \tau, L_j^\tau) \mid j \in \{0, 1, \dots, N\}, \tau \leq t\}$ be the history of the losses where the triplet (j, τ, L_j^τ) denotes the sample index j , the iteration index τ and the value of the loss for sample j at iteration τ (samples seen in the same mini-batch appear in the history as triplets with the same τ).

Our goal is to learn a model $M(x_i, y_i, \mathcal{H}_{t-1}) \approx L(\Psi(x_i; \theta_t), y_i)$ that has negligible computational complexity compared to $\Psi(x_i; \theta_t)$. The above formulation defines a large number of models including approximations of the original neural network $\Psi(\cdot)$. In order to create lightweight models that do not impact the performance of a single forward-backward pass (or impact it minimally) we focus on models that use the class information and the history of the losses. Specifically, we consider models that map the history and the class to two separate representations that are then combined with a simple linear projection.

To generate a representation for the history, we run an LSTM over the previous losses of a sample and return the hidden state. The use of an LSTM allows the history of other samples to influence the representation through the shared weights. Let this mapping be represented by $M_h(j, \mathcal{H}_{t-1}; \pi_h)$ parameterized by π_h . Regarding the class mapping, we use a simple embedding, namely we map each class to a specific vector in \mathbb{R}^D . Let this mapping be $M_y(y_j; \pi_y)$ parameterized by π_y .

Finally, we solve the following optimization problem and learn a function that predicts the importance of each sample for the next training iteration.

$$\pi^*, \pi_h^*, \pi_y^* = \arg \min_{\pi, \pi_h, \pi_y} \sum_{i=1}^N \frac{1}{N} \left(M(M_h(i, \mathcal{H}_{t-1}; \pi_h), M_y(y_i; \pi_y); \pi) - L(\Psi(x_i; \theta_t), y_i) \right)^2 \quad (17)$$

The precise training procedure is described in pseudocode in algorithm 1 where, for succinctness, we use $M(\cdot; \pi)$ to denote the composition and the parameters of the models $M(\cdot)$, $M_h(\cdot)$ and $M_y(\cdot)$.

Algorithm 1 Approximate importance sampling

```
1: Assume inputs  $\eta, \pi_0, \theta_0, k \in (-\infty, 1], X = \{x_1, x_2, \dots, x_N\}$  and  $Y = \{y_1, y_2, \dots, y_N\}$ 
2:  $t \leftarrow 0$ 
3: repeat
4:    $S \sim \text{Uniform}(1, N)$   $\triangleright$  Sample a portion of the dataset for further speedup
5:    $p_i \propto M(i, y_i, \mathcal{H}_t) \forall i \in S$ 
6:    $s \sim \text{Multinomial}(P)$ 
7:    $\alpha \leftarrow \frac{1}{N p_s^k}$ 
8:    $\theta_{t+1} \leftarrow \theta_t - \eta \alpha \nabla_{\theta_t} L(\Psi(x_s; \theta_t), y_s)$ 
9:    $\pi_{t+1} \leftarrow \pi_t - \eta \nabla_{\pi_t} M(s, y_s, \mathcal{H}_t; \pi_t)$ 
10:   $\mathcal{H}_{t+1} \leftarrow \mathcal{H}_t \cup \{(s, t, L(\Psi(x_s; \theta_t), y_s))\}$ 
11:   $t \leftarrow t + 1$ 
12: until convergence
```

3.3.1 Smoothing

Modern Deep Learning models often contain stochastic layers, such as Dropout, that given a set of constant parameters and a sample can result into vastly different outputs for different runs. This fact, combined with the inevitable approximation error of our importance model, can result into pathological cases of samples being predicted to have a small importance (thus large weight α_i) but ending up having high loss.

To alleviate this problem we use *additive smoothing* to influence the sampling distribution towards uniform sampling. We observe experimentally, that a good rule of thumb is to add a constant c such that $c \leq \frac{1}{2N} \sum_{i=1}^N L(\Psi(x_i; \theta_t), y_i)$ for all iterations t .

4 Experiments

We analyze experimentally how our importance sampling scheme improves convergence speed and generalization performance, first for image classification with a deep convolution network, on MNIST [LeCun et al., 1998] in § 4.1 and CIFAR10 [Krizhevsky, 2009] in § 4.2, and then for word prediction using deep recurrent network on the Penn Treebank dataset [Marcus et al., 1993] in § 4.3.

We compare three different sampling strategies: **uniform** sampling which is our baseline, **oracle** which uses the model itself to calculate the importance, and **approx** which uses our approximation defined in § 3.3. For the hyperparameter k , which controls the smooth max-loss (analyzed in § 3.2), we choose the values $k = 1$ and $k = 0.5$, and for some additional analysis with MNIST, we also tried with $k = 0.75$.

The approximation is implemented using two stacked LSTMs with a hidden state size of 32. The input to the first LSTM layer is at most the 10 previously observed loss values of a sample (features of one dimension). Regarding the class label, it is initially projected in \mathbb{R}^{32} and subsequently concatenated with the hidden state of the LSTM. The resulting 64 dimensional feature is used to predict the loss with a simple linear layer.

To prevent any over-estimation of performance related to the choice of the step size, we optimized it *on the test set, for the uniform sampling baseline* with a grid-search in $[0.001, 0.01]$ with step 0.001, hence giving the baseline an unfair advantage over our methods.

In all the experiments, we deviate slightly from our theoretical analysis and algorithm by sampling mini-batches instead of single samples in line 6, and using the Adam optimizer [Kingma and Ba, 2014] instead of plain Stochastic Gradient Descent in lines 8 and 9 of Algorithm 1.

Experiments were conducted using Keras Chollet et al. [2015] with TensorFlow Abadi et al. [2016], and the code to reproduce the experiments will be provided under an open source license when the paper will be published.

Table 1: Experimental results across all datasets and methods. The results are averaged over multiple runs (10 for MNIST and 3 for CIFAR10 and Penn Treebank) and we report the mean and the standard deviation. For the Penn Treebank we report the perplexity in the validation set and for the rest the classification error. The learning rate is chosen to maximize the performance of the uniform sampling strategy for all cases. Runs labeled as **oracle** use the trained model to predict the loss while **approx** use the model defined theoretically in § 3.3 and practically in § 4. An epoch is defined as 300 mini-batches for MNIST/CIFAR10, and 2000 for Penn Treebank respectively.

	Epoch	Method				
		Uniform	Oracle k=1	Oracle k=0.5	Approx k=1	Approx k=0.5
MNIST	5	0.79% \pm 0.11	0.73% \pm 0.11	0.56% \pm 0.04	0.83% \pm 0.06	0.65% \pm 0.05
	10	0.64% \pm 0.10	0.56% \pm 0.07	0.52% \pm 0.05	0.60% \pm 0.06	0.61% \pm 0.04
	20	0.56% \pm 0.07	0.53% \pm 0.08	0.47% \pm 0.04	0.53% \pm 0.06	0.54% \pm 0.06
CIFAR10	50	12.33% \pm 0.39	10.68% \pm 0.65	9.58% \pm 0.16	11.54% \pm 0.55	10.78% \pm 0.09
	100	10.19% \pm 0.24	9.74% \pm 0.44	9.20% \pm 0.55	9.71% \pm 0.14	9.90% \pm 0.47
	150	7.97% \pm 0.10	7.77% \pm 0.14	7.44% \pm 0.14	7.61% \pm 0.07	7.64% \pm 0.27
PTB	10	184.5 \pm 1.56	178.3 \pm 2.47	187.2 \pm 1.61	179.4 \pm 6.20	180.0 \pm 4.78
	30	138.6 \pm 0.60	137.8 \pm 1.53	139.0 \pm 0.72	136.2 \pm 2.98	134.3 \pm 2.19
	50	130.3 \pm 0.63	129.9 \pm 1.27	130.5 \pm 0.21	128.2 \pm 2.02	127.4 \pm 1.45

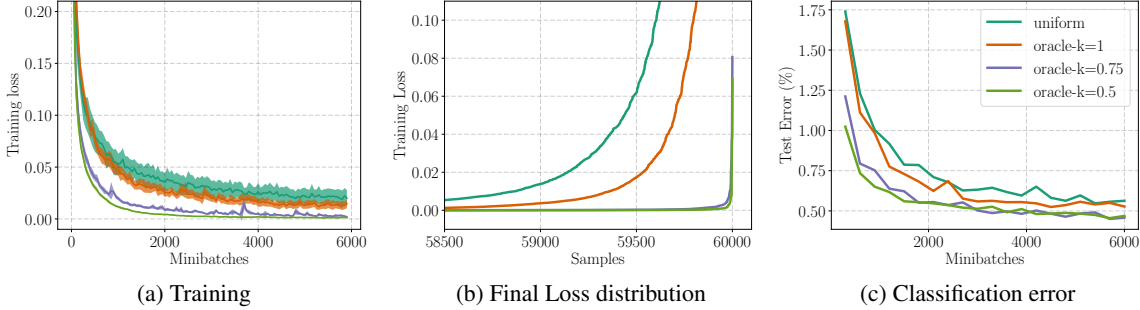


Figure 1: Averaged results of 10 independent runs on MNIST. Figure 1a shows the moving average (solid line) and moving standard deviation (light areas) of the training loss. Figure 1b depicts the per sample loss in the training set after training (the samples are sorted in ascending order according to the loss). Finally, with the last Figure 1c, we show the generalization performance of each run on the test set. The effects of the hyperparameter k (see § 3.2) are obvious, both in terms of minimizing the maximum loss in the training set (1b) and achieving better generalization performance (1c).

4.1 Image classification with MNIST

Our first series of experiments were conducted on MNIST. Given the simplicity of the dataset, we chose for model a down-sized variant of the VGG architecture [Simonyan and Zisserman, 2014], with two convolutional layers with 32 filters of size 3×3 , a max-pooling layer and a dropout layer with a dropout rate of 0.25, without batch normalization, followed by the same combination of layers with twice the number of filters. Finally, we add two fully connected layers with 512 neurons and dropout with rate 0.5, and a final classification layer. All the layers use the ReLU activation function.

Each network is trained for 6,000 iterations with a mini-batch size of 128. We compute the loss and accuracy on the test set every 300 mini-batches. For each set of parameters we run 10 independent runs with different random seeds and report the average. We can see in Figure 1a that importance sampling methods reduce the variance and have significantly smaller oscillations during training than **uniform** sampling, which translates into faster convergence. Figure 1b shows that lower k indeed aggressively reduces the loss of individual samples, which translates in higher accuracy, as shown on Figure 1c. It is noteworthy that training with $k = 0.5$ leads to less than 1% error after a single epoch and that, as shown in Table 1, an error of 0.6% is reached 15 epochs earlier than with **uniform**. Finally, we notice in Table 1, that **approx** also improves the convergence speed and generalization error compared to **uniform** by achieving 0.65% error 5 epochs earlier.

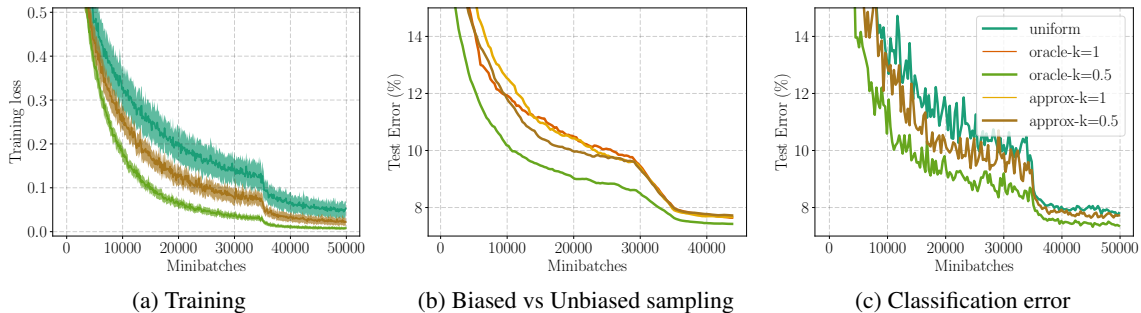


Figure 2: Averaged results of 3 independent runs on CIFAR10. Figure 2a shows the moving average (solid line) and the moving standard deviation (shaded areas) of the training loss. Figure 2b depicts the moving average of the test classification error of the four importance sampling schemes. Finally, Figure 2c compares the generalization performance of uniform, oracle and approximate importance sampling.

4.2 Image classification with CIFAR10

Our second series of experiments were conducted on CIFAR10, which is a more challenging dataset, commonly used to evaluate new Deep Learning methods. We developed a VGG-inspired network (with batch normalization), which consists of three convolution-pooling blocks with 64, 128 and 256 filters, two fully connected layers of sizes 1024 and 512, and a classification layer. Dropout is used in a similar manner as for the MNIST network with rates 0.25 after each pooling layer and 0.5 between the fully connected layers. The activation function in all layers is ReLU.

Each network is trained for 50,000 iterations using a batch size of 128 samples. After 35,000 iterations we decrease the learning rate, by multiplying it by 10^{-1} . We perform minor data augmentation by creating 500,000 images generated by random flipping, horizontal and vertical shifting of the original images. For the importance sampling strategies, we use smoothing as described in § 3.3.1. In particular, the importance of each sample is incremented by $\frac{1}{2}\bar{L}$, where \bar{L} is the mean of the training loss computed by the exponential moving average of the mini-batch losses. Furthermore, we run each method with 3 different random seeds and report the mean.

Figure 2 depicts the results of our experiment on CIFAR10. In Figure 2a, we reproduce the results observed in the MNIST experiment in terms of variance reduction. In addition, we note that the proposed approximation is also able to reduce the gradient variance and achieve faster convergence in the training set. Furthermore, in Figure 2b we observe that biased sampling has better classification performance both using the oracle and the approximation, thus validating our initial assumption that this kind of bias can result in improved generalization ability. Finally, in the last Figure 2c, we can see that importance sampling affects the training primarily during the first epochs. This phenomenon occurs due to the large initial learning rate (necessary for fast convergence) which results in large gradient variance. Specifically, we see that uniform sampling achieves less than 10% error only after the learning rate decrease at epoch 116 instead of 78 for our approximation and 48 for the oracle. The results in Table 1 also show clear improvement compared to **uniform** for every method, both **oracle** and **approx**. Specifically, our approximation with $k = 0.5$ reaches the performance of uniform sampling at 100 epochs almost 50 epochs faster.

4.3 Word prediction with Penn Tree Bank

To assess the generality of our method, we conducted experiments on a language modeling task. We used the Penn Treebank language dataset, as preprocessed by Mikolov et al. [2011]¹, and a recurrent neural network as the language model. Initially, we split the dataset into sentences and add an “End of sentence” token (resulting in a vocabulary of 10,001 words). For each word we use the previous 20 words, if available, as context for the neural network. Our language model is similar to the small LSTM used by Zaremba et al. [2014] with 256 units, a word embedding in \mathbb{R}^{64} and dropout with rate 0.5.

¹<http://www.fit.vutbr.cz/~imikolov/rnnlm/>

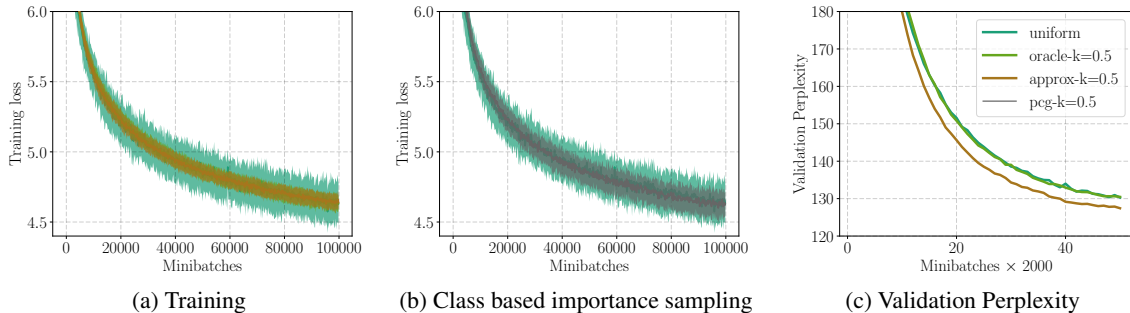


Figure 3: Averaged results of 3 independent runs on Penn Treebank. Figure 3a shows the moving average (solid line) and the moving standard deviation (shaded area) of the training loss. Figure 3b depicts the variance reduction achieved with a simpler univariate Gaussian model per class. Finally, Figure 3c compares the perplexity on the validation set between training with importance sampling and uniform sampling.

In terms of training the language models, we use a batch size of 128 words and train each network for 100,000 iterations. For the importance sampling strategies, we use constant smoothing instead of adaptive as in the CIFAR10 experiment. To choose the smoothing constant, we experiment with the values $\{0.5, 1, 2.5\}$ and choose 0.5 because it performs better in terms of variance minimization during training. Finally, we run each method with 3 different random seeds and report the mean.

The results of the language modeling experiment are presented in Figure 3. In Figure 3a, it is clear that the variance is reduced using both the oracle and our approximation of the loss, and that the gap in performance between the two is insignificant. We theorize that the rich class information can be utilized to model the loss more effectively. To validate this theory we also conduct an experiment modeling the loss by a univariate Gaussian for each class (denoted as “pcg”) and use the mean as the importance. The corresponding results are depicted in Figure 3b. We observe a reduction in variance but it is less significant than with our approximation.

Finally, both in Figure 3c and Table 1, we observe that importance sampling improves the convergence speed as well as the generalization error in the language modeling task. Surprisingly, the oracle performs worse than our approximation in this task, which can be explained by the filtering across mini-batches that the approximation provides. Nevertheless, it still improves on uniform with significant variance reduction (see Figure 3a) that could be used, for instance to increase the learning rate.

5 Conclusion

We have proposed an importance sampling scheme suitable for use with Deep Learning models and an optional biased gradient estimator that can focus on hard examples in the training set and improve the generalization performance of our models. Furthermore, we have shown that the loss can be approximated with a model with significantly lower complexity paving the way for efficient and practical training of Deep models with importance sampling.

Subjects for further research include the experimentation with other models for approximate importance sampling as well as the combination of our proposed importance sampling method with optimized implementations that can significantly improve wall clock training time.

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

- Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.
- A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6(Sep):1579–1619, 2005.
- O. Canévet, C. Jose, and F. Fleuret. Importance sampling tree for large-scale empirical expectation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1454–1462, 2016.
- F. Chollet et al. keras. <https://github.com/fchollet/keras>, 2015.
- D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- A. Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, Department of Computer Science, University of Toronto, 2009.
- M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.
- Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.
- M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.
- T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Černocký. Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.
- D. Needell, R. Ward, and N. Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2014.
- P. Richtárik and M. Takáč. On optimal probabilities in stochastic coordinate descent methods. *arXiv preprint arXiv:1310.3438*, 2013.
- S. Shalev-Shwartz and Y. Wexler. Minimizing the maximal loss: How and why. In *Proceedings of the 32nd International Conference on Machine Learning*, 2016.
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- L. Wang, Y. Yang, M. R. Min, and S. Chakradhar. Accelerating deep neural network training with inconsistent stochastic gradient descent. *arXiv preprint arXiv:1603.05544*, 2016.
- W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.
- P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1–9, 2015.

Appendix

A Justification for sampling with the loss

The goal of this analysis is to justify the use of the loss as the importance metric instead of the gradient norm and provide additional evidence (besides the experiments in the paper) that it is an improvement over uniform sampling.

Initially, in § A.1 we will show that sampling with the loss is better at minimizing an upper bound to the variance of the gradients than uniform sampling.

Subsequently, in § A.2 we provide additional empirical evidence that the loss is a surrogate for the gradient norm by computing the exact gradient norm for a limited number of samples and comparing it to the loss.

A.1 Theoretical justification

Initially, we will show in lemma 2 that the most common loss functions for classification and regression define the same ordering as their gradient norm. Subsequently, we will use this derivation, in § A.1.1, together with an upper bound to the variance of the gradients and show that sampling according to the loss reduces this upper bound compared to uniform sampling.

Firstly, we prove two lemmas that will be later used in the analysis.

Lemma 1. *Let $f(x) : \mathbb{R} \rightarrow \mathbb{R}$ be a strictly convex monotonically decreasing function then*

$$f(x_1) > f(x_2) \iff \left| \frac{\partial f}{\partial x_1} \right| > \left| \frac{\partial f}{\partial x_2} \right| \quad \forall x_1, x_2 \in \mathbb{R}. \quad (18)$$

Proof. By the definition of $f(x)$ we have

$$\frac{\partial^2 f}{\partial x^2} > 0 \quad \forall x \quad (19)$$

$$\frac{\partial f}{\partial x} \leq 0 \quad \forall x, \quad (20)$$

which means that $\frac{\partial f}{\partial x}$ is monotonically increasing and non-positive. In that case we have

$$x_1 < x_2 \iff f(x_1) > f(x_2) \quad (21)$$

$$x_1 < x_2 \iff \frac{\partial f}{\partial x_1} < \frac{\partial f}{\partial x_2} \iff \left| \frac{\partial f}{\partial x_1} \right| > \left| \frac{\partial f}{\partial x_2} \right| \quad (22)$$

therefore proving the lemma. \square

Lemma 2. *Let $L(\psi) : D \rightarrow \mathbb{R}$ be either the negative log likelihood or the squared error loss function defined respectively as*

$$L_1(\psi) = -y^T \log(\psi) \quad y \in \{0, 1\}^d \text{ s.t. } y^T y = 1 \quad D = [0, 1]^d \text{ s.t. } \|\psi\|_1 = 1 \quad (23)$$

$$L_2(\psi) = \|y - \psi\|_2^2 \quad y \in \mathbb{R}^d \quad D = \mathbb{R}^d \quad (24)$$

where y is the target vector. Then

$$L(\psi_1) > L(\psi_2) \iff \|\nabla_\psi L(\psi_1)\| > \|\nabla_\psi L(\psi_2)\| \quad (25)$$

Proof. In the case of the squared error loss we have

$$\|\nabla_{\psi} L(\psi)\|_2^2 = \|-2(y - \psi)\|_2^2 = 4L(\psi), \quad (26)$$

thus proving the lemma.

For the log likelihood loss we can use the fact that only one dimension of y can be non-zero and prove it using lemma 1 because $f(x) = -\log(x)$ is a strictly convex monotonically decreasing function. \square

A.1.1 Main analysis

The goal for importance sampling is to minimize

$$\text{Tr}(\mathbb{V}[\nabla_{\theta} L(\Psi(x_i; \theta), y_i)]) = \mathbb{E}[\|\nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|_2^2]. \quad (27)$$

To perform importance sampling, we sample according to the distribution P with probabilities p_i and use per sample weights $\alpha_i = \frac{1}{Np_i}$ in order to have an unbiased estimator of the gradients. Consequently, the variance of the gradients is

$$\mathbb{E}_P[\|\alpha_i \nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|_2^2] = \sum_{i=1}^N p_i \alpha_i^2 \|\nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|_2^2 \quad (28)$$

$$= \sum_{i=1}^N \frac{1}{Np_i} \frac{1}{N} \|\nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|_2^2 \quad (29)$$

$$= \sum_{i=1}^N \alpha_i \frac{1}{N} \|\nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|_2^2. \quad (30)$$

Assuming that the neural network is Lipschitz continuous (assumption that holds when the weights are not infinite) with constant K , we derive the following upper bound to the variance

$$\mathbb{E}_P[\|\alpha_i \nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|_2^2] \leq \sum_{i=1}^N \alpha_i \frac{1}{N} \|\nabla_{\theta} \Psi(x_i; \theta)\|_2^2 \|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\|_2^2 \quad (31)$$

$$\leq K^2 \sum_{i=1}^N \alpha_i \frac{1}{N} \|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\|_2^2. \quad (32)$$

Since we have a finite set of samples, there exists a constant c such that

$$L(\Psi(x_i; \theta), y_i) + c \geq \|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\| \quad \forall i \in \{1, 2, \dots, N\}. \quad (33)$$

However, using lemma 2 we know that this upper bound is better than uniform because $L(\Psi(x_i; \theta), y_i)$ and $\|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\|$ grow and shrink in tandem. In particular the following equation holds,

$$\begin{aligned} & \sum_{i=1}^N (L(\Psi(x_i; \theta), y_i) + c - \|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\|) < \\ & \sum_{i=1}^N (\max \|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\| - \|\nabla_{\Psi(x_i; \theta)} L(\Psi(x_i; \theta), y_i)\|). \end{aligned} \quad (34)$$

A.2 Empirical justification

In this section, we provide empirical evidence regarding the use of the loss instead of the gradient norm as the importance metric. Specifically, we conduct experiments computing the exact gradient norm and the loss value for the first 20,000 samples during training. The gradient norm is normalized in each mini-batch to account for the changes in the norm of the weights.

Subsequently, we plot the loss sorted by the gradient norm. If there exists C such that $L(\Psi(x_i; \theta), y_i) = C \|\nabla_{\theta} L(\Psi(x_i; \theta), y_i)\|$ we should see approximately a line. In case of order preservation we should see a monotonically increasing function.

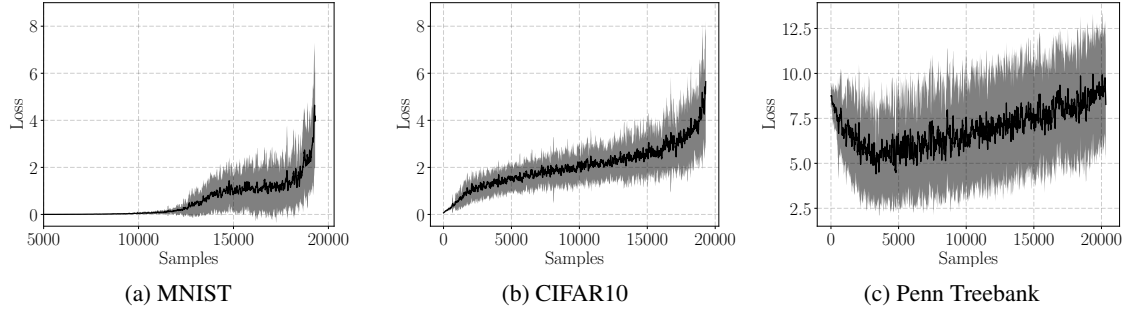


Figure 4: Loss values sorted by gradient norm. The solid line is a moving average (50 samples window) and the shaded area denotes one standard deviation.

In figure 4, we observe a correlation between the gradient norm and the loss. In all cases, samples with high gradient norm also have high loss. In the Penn Treebank dataset, (Figure 4c), there exist some samples with high loss but very low gradient norm. This can be explained because LSTMs use the $\tanh(\cdot)$ activation function which can have very low gradient but incorrect output. We note that this cannot hurt performance, it just means that we waste some CPU/GPU cycles on samples that are incorrectly classified but will not affect the parameters heavily.