Utilizing the Neural Tangent Kernel to Extend LSH-Sampled SGD to Nonlinear Models

Kelly O. Marshall
Tandon School of Engineering
New York University
New York, NY
km3888@nyu.edu

Will David
Tandon School of Engineering
New York University
New York, NY
wmd9712@nyu.edu

Abstract

Recent work has shown that the number of iterations required by Stochastic Gradient Descent (SGD) can be reduced by using non-uniform sampling to obtain lower variance estimates of the gradient. However, the additional time required to compute the sampling weights for the entire dataset makes these methods slower in practice than regular SGD. This problem can be avoided for linear models by using Locality Sensitive Hashing (LSH) to avoid explicitly computing the sampling weight for each datapoint. With this approach, the overall runtime of SGD can be significantly reduced for the training of linear models. To extend this LSH sampled SGD to arbitrary non-linear models such as neural networks, we propose using the Neural Tangent kernel to approximate the optimal sampling weights. We perform empirical experiments to evaluate the quality of this approximation and the effect of non-uniform sampling on the convergence of SGD for non-linear models.

1 Introduction

Recent work has found that sampling points of high gradient magnitude during stochastic gradient descent(SGD) leads to faster convergence. However, there are a number of challenges that make it difficult to take advantage of this. One reason is that the points that result in high gradient magnitudes vary throughout the training process as the parameters are updated. As a result, trying to take advantage of this for faster training leaves on stuck in what the authors of Chen et al. [2019] dub the "chicken-and-egg loop": If you want to find a point of high gradient magnitude, you have to calculate the gradient at all the points, which means you could have just calculated the full gradient over the whole dataset. Even though a point of high gradient magnitude leads to faster convergence than a randomly selected point, the full gradient will be even better so the advantages of using SGD are nullified. To address this problem, the paper introduces a method to efficiently sample points of high gradient magnitude for linear models: Through expressing the gradient magnitude as an inner product of one vector that changes during training and one vector that does not, the unchanging vector is stored in an LSH scheme and the changing vector can be used to query the LSH scheme to retrieve gradients of high magnitude. This allows for sampling of datapoints with relatively high gradient magnitude in O(1) time per iteration. Their experiments show empirically that this reduces training time for various classes of linear models. In this paper, we propose an algorithm which uses a linear approximation of the model being trained in order to apply the methods outlined in Chen et al. [2019] to neural networks instead of being restricted to linear models. Our work is motivated by study of the Neural Tangent Kernel (NTK), which suggests that for models with more parameters, linear approximations are surprisingly accurate. We test these ideas in several empirical experiments showing the accuracy of linear approximations to gradient magnitude as well as the impact of non-uniform sampling on the training of neural networks.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 Background

2.1 Gradient Descent and Stochastic Gradient Descent

Gradient Descent(GD) is a method for minimizing the average cost over some dataset of size n which is equivalent to finding:

$$w^* = \text{argmin}\mathcal{L}(w) = \text{argmin}_w \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(x_i, w) \tag{1}$$

where $\mathcal{L}$ is the cost function and w is the parameter vector of some model $f(x_i, w)$.
In the simplest form of gradient descent, each iteration consists of computing the gradient of the total cost function $g = \sum_{i=1}^{n} \nabla_w \mathcal{L}(x_i, w)$ and then updating $w \leftarrow w - \eta g$ using some small learning rate η. This is done repeatedly until w converges to some approximation of w^*.
In Stochastic Gradient Descent (SGD) a nearly identical procedure is used to update the parameters, except that instead of explicitly computing g, SGD approximates g with a single sample estimate. In each iteration, a single datapoint x_i is sampled uniformly from the dataset and is then used to compute:

$$g_i = \nabla_w \mathcal{L}(x_i, w) \tag{2}$$

In expectation, we have that $\mathbb{E}[g_i] = g$, so using this single-sample estimate allows us to take steps in a direction which is an unbiased estimator of the true gradient vector. While the variance that comes from using this estimator means that SGD requires a greater number of iterations to reach convergence, this is offset by the time saved by not computing the gradient for the entire dataset. In many cases, this results in a more efficient optimization algorithm than regular gradient descent, making SGD a useful tool for optimization in practice.

2.2 Non-Uniform Sampling For SGD

While SGD samples each datapoint with equal probability $\frac{1}{n}$, a non-uniform sampling approach can yield a lower-variance estimator of the true gradient. To do this, we sample each datapoint x_i with its own probability $x_i \in (0, 1)$ such that $\sum_{i=1}^{n} = 1$. Assigning greater probability to low-variance datapoints improves gradient estimation and allows SGD to converge in fewer iterations, as was first shown in Alain et al. [2016]. In particular, they established that the expected variance of a single datapoint is inversely correlated with the magnitude of the gradient step taken when using the point for SGD, $\|g_i\|_2$. This gives us the useful statement that the optimal SGD sampling weight for a datapoint x_i is equal to:

$$p_i^* = \frac{\|\nabla_{w_t} \mathcal{L}(x_i, w_t)\|_2}{\sum_{i=1}^{n} \|\nabla_{w_t} \mathcal{L}(x_i, w_t)\|_2} \tag{3}$$

The main caveat to using this weighting to improve SGD is the time required to compute the value of p_i for each point in the dataset. Since the main factor considered in most applications isn't the number of iterations but rather the total runtime and computational cost, this approach offers little benefit on it own. In fact, since arriving at all the values of p_i requires computing each value of g_i, one could easily just compute the true value of g, defeating the purpose of using SGD.

2.3 LSH-Sampled SGD

The adaptive sampling method for SGD mentioned in section 2.2 may seem useless, since a naive approach takes $O(N)$ time to sample a data point where N is the number of data points, in which case you may as well use that time to run multiple training iterations rather than carefully selecting the data point that would most benefit the model if used for one iteration. However, recent work by Chen et al. [2019] has shown that through utilizing locally sensitive hashing (LSH), the adaptive sampling method for SGD mentioned in section 2.2 can be done *for linear models* in $O(1)$ time after some pre-processing.

This LSH method works for linear models because the gradient magnitude for a single data point can be written as the inner product of two vectors - one that is the same for all data points that changes

during training, and one that is different for each data point and does not change throughout training:

$$\|\nabla_{w_t} f(x_i, w_t)\|_2 = |2(w_t \cdot x_i - y_i)\|x_i\|_2| = 2|[w_t, -1] \cdot [x_i\|x_i\|_2, y_i\|x_i\|_2]| \tag{4}$$

And since the data points are normalized, and we only need a quantity proportional to the gradient magnitude, not the true gradient magnitude,

$$2|[w_t, -1] \cdot [x_i\|x_i\|_2, y_i\|x_i\|_2]| = 2|[w_t, -1] \cdot [x_i, y_i]| \propto |[w_t, -1] \cdot [x_i, y_i]| \tag{5}$$

Thus, the vectors on the right hand side of the inner product can be stored in a locality sensitive hashing structure, and queried by the vector on the left hand side in $O(1)$ to achieve fast adaptive sampling. With each iteration, the query to the LSH structure changes as the weights of the model change, but the hash table stays the same. Only a few hashes are necessary to sample x_i that corresponds with the adaptive sampling distribution described in section 2.2. The locally sensitive hashing structure gives points with high gradient magnitude with probability proportional to their magnitude. We pay a one time pre-processing cost of developing the hashing structure, and achieve fast adaptive sampling. Details of the LSH sampling algorithm will are discussed in section 3.

2.4 The Neural Tangent Kernel

Recent work on the subject of the Neural Tangent Kernel(NTK) can provide insight on the linearization of neural network models Jacot et al. [2018]. Firstly, the core idea is to approximate a neural network up to quadratic factors using its Taylor expansion. This gives us the approximation:

$$f(x, w) \approx f(x, w_0) + \nabla_w f(x, w_0)^T (w - w_0) \tag{6}$$

This yields a linear model which induces a kernel on the input. Theoretical analysis on this kernel suggests that for larger networks with wide layers, this approximation will be relatively accurate because the weights in such networks don't tend to deviate very far from their initializations.

3 Methods: Approximate Gradient Sampling

By using the NTK approximation at the end of section 2.4, we can approximate the gradient magnitude of any non-linear model with the inner product of two vectors: One that is the same for each point and changes throughout training and one that is different for each point but doesn't change throughout training. Thus, we extend LSH-sampled SGD to non-linear models such as neural networks, and ultimately any arbitrary differentiable model.

Starting with the NTK approximation:

$$f(x, w) \approx f(x, w_0) + \nabla_w f(x, w_0)^T (w - w_0) \tag{7}$$

Define:

$$b_i = f(x_i, w_0) - \nabla_w f(x_i, w_0) \cdot w_0 \tag{8}$$

$$a_i = \nabla_w f(x_i, w_0) \tag{9}$$

Then, we can rewrite the NTK approximation as

$$f(x_i, w) = b_i + a_i \cdot w \tag{10}$$

With MSE loss L,

$$L = \frac{1}{2}(y_i - a_i \cdot w - b_i)^2 \tag{11}$$

$$\frac{\partial L}{\partial w} = [w, -1] \cdot [a_i|a_i|, (y_i - b_i)|a_i|] \tag{12}$$

$$|\frac{\partial L}{\partial w}| = |[w, -1] \cdot [a_i|a_i|, (y_i - b_i)|a_i|]| \tag{13}$$

$$\hat{p} = |[w, -1] \cdot [a_i|a_i|, (y_i - b_i)|a_i|]| \tag{14}$$

Thus, by the same method as described in section 2.3 for linear models, fast adaptive sampling can be achieved in approximation for non-linear models such as neural networks. Using this we are able to compute an estimate of the gradient magnitude for a single datapoint m_i and then use LSH to sample points in proportion to our estimate of the optimal sampling coefficients.

$$\hat{p_i} = \frac{m_i}{\sum_{j=1}^{n} m_j} \tag{15}$$

The following pseudocode demonstrates the end-to-end method for training a non-linear model with LSH sampling and the NTK approximation. Note that for the LSH sampling method used with linear models, $x^i_{lsh} = x_i$ and $y^i_{lsh} = y_i$, but for the LSH sampling used with nonlinear models via the NTK approximation, $x^i_{lsh} = a_i|a_i|$ and $y^i_{lsh} = (y_i - b_i)|a_i|$ where a_i and b_i are defined above. This pseudocode has been adapted from the pseudocode for LSH sampling with linear models presented in recent work by Chen et al. [2019].

Algorithm 1 LSH Sampling Algorithm (Sample)

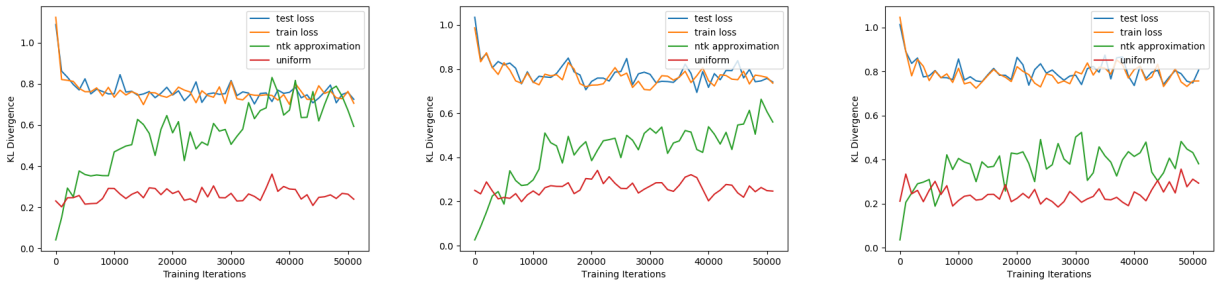
Input: Hash functions H, $HT[][]$ (L has tables), K, Query, cp(x,Q) representing Pr(h(x)=h(Q))
Output: sampled data x, sampling probability p
l,S = 0
while true **do** ti = random(1,L) bucket = H(Query,ti)
    if HT[ti][bucket]=empty **then**
        l++
    S=|HT[ti][bucket]|
    x= randomly pick one element from HT[ti][bucket]
    break;
p = $cp(x, Query)^K(1 - cp(x, Query)^K)^{l-1}\frac{1}{S}$
return x, p

Algorithm 2 Approximate LSH Sampling with NTK Training Algorithm (Pre-Process and Train)

Require: $D = \{(x_i, y_i)\}_{i=1}^N, w_0, \eta$
Require: LSH Family H, parameters K, L
Output: w^*
$[x^i_{lsh}, y^i_{lsh}]$ = $[a_i|a_i|, (y_i - b_i)|a_i|]$ (a_i and b_i defined above)
HT = Put preprocessed training data vectors $[x^i_{lsh}, y^i_{lsh}]$ into LSH data structure
Get x'_{train}, y'_{train} from preprocessed data
$t = 0$
while NotConverged **do**
    $x^i_{lsh}, p = Sample(H, HT, K[w_t, -1])$ (Algorithm 1)
    Get $x^{i'}_{train}, y^{i'}_{train}$ from preproprecessed data
    $w_{t+1} := w_t - \eta_t(\frac{\nabla_{w_t} f(x^{i'}_{train}, w_t)}{pN})$
Return w^*

4 Experiments

We perform experiments to investigate the feasability of the propose methods above. Our experiments use the Million Song Dataset, in which each datapoint is a 90 dimensional feature vector representing a song and the corresponding label is the song's year of releaseDua and Graff [2017]. This dataset is also used in Chen et al. [2019]
In our experiments, we did not use the LSH component needed to achieve computational efficiency as we were not able to find a suitable implementation. Instead, our experiments are aimed at answering how well the linearized model approximates the optimal sampling distribution and how much effect there is on SGD's performance on training neural networks with this non-uniform sampling. These are the two major factors which determine the feasibility of the proposed algorithm. Code to reproduce these experiments can be found here



(a) 8-neuron hidden layer

(b) 32-neuron hidden layer

(c) 90-neuron hidden layer

4.1 Accuracy of Gradient Magnitude Approximation

To measure the effectiveness of our approximation of the gradient magnitude, we can measure the KL divergence between the true optimal sampling distribution for reducing variance p_i (3) and the one given by our estimate $\hat{p}$ (15). We compare this against the KL divergence between optimal distribution and the uniform distribution used by regular SGD. This baseline allows us to see when we would be better of using the distribution given by our approximation as opposed to regular sampling. To empirically verify the idea that the NTK approximation will be more effective with wider layers, we make this comparison for neural networks each with a single hidden layer of a particular size.

The results of these experiments are shown in 1a,1b,1c respectively. As expected, the deviation between the optimal distribution and the approximation distribution quickly increases with more training iterations. However, for much of the model's training process -almost until convergence- the approximation distribution outperforms the baseline of uniform sampling. This suggests that there is value in using this approximation and that doing so would reduce the number of iterations needed for SGD. We also see that there does indeed seem to be a benefit in using wider networks.

4.2 Effect of Approximate Sampling Method on Convergence

In this experiment, we attempt to test the benefits of the approximate sampling in terms of iterations required for convergence. We compare four sampling methods while fixing the model and all training parameters:

1. Standard Stochastic Gradient Descent ($O(1)$ time for each sample).
2. True Adaptive Sampling ($O(N)$ time for each sample, but "good" samples).
3. Approximate Adaptive Sampling, resetting LSH structure after each epoch ($O(1)$ time for each sample after prepossessing, and "approximately good" samples).
4. Approximate Adaptive Sampling, resetting LSH structure after each 50 iterations ($O(1)$ time for each sample after prepossessing, and "approximately good" samples).

Due to computational limitations, we trained with each method on a portion of the MSD Year Prediction dataset - 1000 points for 2 epochs (2000 iterations total). Also due to computational limitations, we used a network with one hidden layer of size 32, which is still in the under-parameterized regime, meanwhile the NTK approximation holds far more strongly in an over-parameterized regime. Thus, this experiment would be far stronger if trained on more data points, a wider network, and even done on multiple datasets.

As expected from the theory, Method 2 resulted in lower training loss than Method 1 after training. For Method 3, resetting once every epoch seems to have resulted in too weak of an approximation for useful training to occur. For Method 4, we see more effective training than Method 3 by resetting the approximation more frequently, but still less effective training than SGD in Method 1. However, the comparison of Method 3 and Method 4 suggests that if the approximation could be made stronger via other means than resetting more frequently, such as using wider networks, the Approximate Adaptive Sampling could result in a performance improvement as compared to standard SGD.

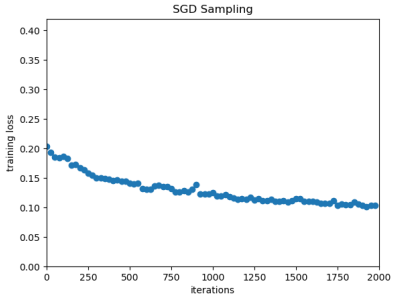


Figure 2: Sampling Method 1

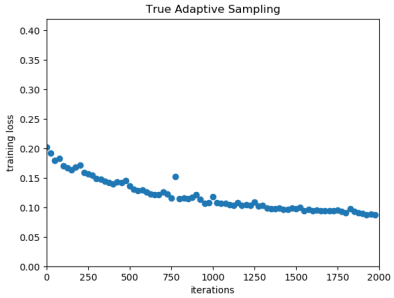


Figure 3: Sampling Method 2

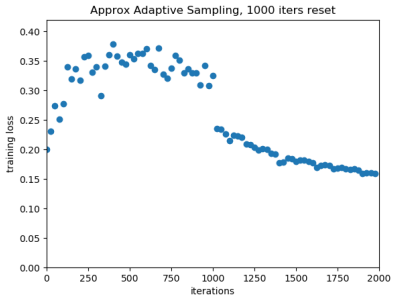


Figure 4: Sampling Method 3

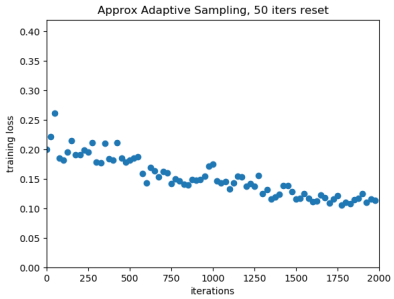


Figure 5: Sampling Method 4

The change in behavior after the approximation reset via re-hashing in Method 3 is worth noting. The loss over the training set goes up dramatically at first, likely because it is sampling the same points over and over again but ultimately doing worse across the whole training set. After this slight training and a reset, we see the training resume with the loss steadily going down, which hints that frequent resets may be more important early on in training.

5 Conclusion

To our knowledge, our sampling method is the first to enable adaptive sampling of points in proportion with their gradient magnitude for differentiable non-linear models such as neural networks, albeit in approximation. Our experiments suggest that this approximation is strong enough to enable an improvement in the number of iterations needed for Stochastic Gradient Descent to converge. Computational limitations prohibited us from (1) training on a sufficient amount of data points for conclusive results, (2) training on enough data sets for conclusive results that are not specific to a single data set, and (3) using large enough networks that may lead to a more accurate approximation as well as more accurately represent real world modeling scenarios. However, our experiments suggest that using larger networks would also prove to be beneficial. As a result, future work includes extending our experiments to larger data samples, more data sets, and larger networks.

References

Guillaume Alain, Alex Lamb, Chinnadhurai Sankar, Aaron Courville, and Yoshua Bengio. Variance reduction in sgd by distributed importance sampling, 2016.

Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. Lsh-sampling breaks the computation chicken-and-egg loop in adaptive stochastic gradient estimation, 2019.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 8571–8580. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.