# Non Uniform Sampling For Faster Convergence in Neural Networks

**Amol Agrawal**
College of Information and Computer Sciences
University of Massachusetts Amherst
amolagrawal@cs.umass.edu

**Paresh Shukla**
College of Information and Computer Sciences
University of Massachusetts Amherst
pareshshukla@cs.umass.edu

**Rishikesh Jha**
College of Information and Computer Sciences
University of Massachusetts Amherst
rishikeshjha@cs.umass.edu

## Abstract

Training of deep neural networks still remains a difficult and time consuming problem to solve. These networks are trained by uniformly selecting a small random batch of input data and the weights are updated using gradients calculated by these batches. But in general, different input data may have different impact on the network's weights and its training. So we experiment on selecting these batches non-uniformly in order to assign different priority of data point selection while selecting this batch. We selected batches using objective based, gradient based and variance based selection procedure. This is the first instance of using gradient based and variance based sampling for training deep neural networks. We were able to achieve better performance on Fully Connected networks with MNIST dataset but the methods did not prove to be as useful with other models and datasets.

## 1 Introduction

Since the resurgence of neural networks in 2012, they have been used to solve a large variety of machine learning problems. With the growth of computing power, there has been a tendency to build deeper networks which can learn increasingly complex structures within the data. These deeper networks, however are equally difficult to train. Since they work on large amount of data, they take weeks or even months to train. Even with modern high-performance GPUs, these times can be of the order of days. This huge training time becomes a significant bottleneck. Thus any speed up in the convergence of these network can be of significant use and can bring down the training time drastically. Such reduction in training time can open new doors for the applications of these networks in even wider range of areas.

In general, the training of these neural networks is done by following a stochastic non-convex optimization procedure, where the entire training data set is shuffled and divided into a number of batches. Each of these batches are then selected one by one and are used to compute the gradient of loss with respect to the weights, and finally the weights are updated using these gradients. This

means, in each epoch all the data points are selected uniformly with equal probability. However, intuitively this seems to be unwise since different data points contribute differently to the model and its training process. So if we can find the points which are more useful for training, we should be selecting them more often for updating the weights. Here we will be exploring this intuition in a scientific manner.

In our present work we investigate the process of utilizing non-uniform methods of sampling over uniform sampling. We propose three methods of non-uniform sampling: Objective-based, where in each batch the probability of selecting a data point is higher if it leads to a higher loss. Gradient-based sampling, where the data points which results in higher gradient value of the loss with respect to weights are given higher probability of selection. Variance-based sampling, where the data points with higher variance is given a higher probability of selection. Currently papers [1] and[2] show that simple batch selection mechanisms can lead to substantial performance improvements. [4] provides intuition about the role of sample importance in neural networks. The idea of gradient based sampling has not been experimented upon, in deep neural networks. The only application of this approach was done on a least squares problem [3], which means the algorithm needed a lot more refinement for application in deep neural networks. There was no literature found which used variance based sampling for these problems.

All the above mentioned algorithms were designed and tested on three types of networks: a two layer fully connected neural network, a convolutional neural network and an LSTM network. These were compared against the standard uniform batch selection procedure as a baseline. We show in section 6 that non-uniform sampling has some advantages in fully connected networks on MNIST datasets however with other models on other datasets like CIFAR the methods did not give the time gains expected.

In section 2 we discuss some previous works detailing efforts in the field on non-uniform sampling. In section 3 we detail the methodology and the non-uniform methods proposed. Section 4 contains description about datasets used for evalaution while section 5 contains the finer details related to the experiments performed. In section 6 we show the results obtained and discuss how successful the non-uniform methods actuall were. Finally in section 7 we discuss the results and offer concluding words.

## 2 Related Work

[4] discusses stochastic gradient descent where parameters are updated after each sample is seen as given by

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta} \tag{1}$$

where $z_t$ is the example at time $t$ and $\epsilon_t$ is the learning rate. In practise we normally use batch gradient descent[5] where when batch size is 1 it is ordirnary online gradient descent and when batch size is total training size then it is standard gradient descent. As B increase we get speedups due to efficient multiplication of matrices but that also results in increase in the number of updates per computation step. Hence generally B has a sweet spot that can be exploited. For each epoch the training set is broken into batches which are then iterated over. Thus all the samples get a chance and hence have a uniform chance of selection and are therefore uniformly sampled.

Some significant research has been done to investigate methods which take into account the property of the data itself and use it to form batches instead of uniformly sampling. Below we discuss some of them.

[3] uses the approach of gradient based sampling for the least squares problem. It assumes the initial weights $\beta_\mathbf{0}$, are a good estimate for parameters $\beta$, which is then used to calculate the gradient for the $i$th datapoint:

$$\mathbf{g_i} = \frac{\partial \mathbf{l_i}(\beta_\mathbf{0})}{\partial \beta_\mathbf{0}} = \mathbf{x_i}(\mathbf{y_i} - \mathbf{x_i^T}\beta_\mathbf{0}) \tag{2}$$

These gradient values are then used to assign selection probability on each datapoint. Probability of selecting a datapoint $i$ for a sample is given as:

$$\pi_\mathbf{i}^\mathbf{0} = \frac{\|\mathbf{g_i}\|}{\sum_{\mathbf{i=1}}^{\mathbf{n}} \|\mathbf{g_i}\|} \tag{3}$$

These probabilities are then used to sample the $i$th data point for a batch using *Poisson sampling* for selection or rejection of these points. In this process a randomized experiment is performed on each data point, which results either in selection or rejection of the corresponding data point.

This procedure had the limitation of applying only to a least square problems and it made an assumption of starting with a very good approximation of initial weights $\beta_{\mathbf{0}}$. We propose a modification to this idea for deep neural networks by first running a few epochs on uniformly sampled data so that we can get to a good estimate of weights after starting with any type of initialization. We can then calculate the gradient of loss $\mathbf{l_i}(\beta)$ with respect to all the weights for each data point $i$. All these gradient matrices can be used to find the gradient score of each data point $i$ and assign a normalized probability to each point $i$ based on this score. Finally, we can use a Bernoulli sampling to select datapoints based on their probability. For each point a Bernoulli trial is run with a probability $\mathbf{sample\_size} * \pi_{\mathbf{i}}$ and if the outcome is 1, the point is selected for the batch.

[1] used the approach of online batch selection for training deep neural networks while using Adam and AdaDelta. It used a simple selection strategy of assigning ranks to each data point and then selecting them for a batch using this rank. These ranks of the datapoint $i$ were sorted in descending order with respect to their latest computed loss. Each rank was selected with a probability based on their contribution to the loss.

$$\mathbf{p_i} = \frac{\mathbf{1}/\exp(\log \mathbf{s_e}/\mathbf{N})^{\mathbf{i}}}{\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{1}/\exp(\log \mathbf{s_e}/\mathbf{N})^{\mathbf{i}}} \tag{4}$$

where $\mathbf{s_e}$ is the ratio between greatest and smallest probabilities of selection at each epoch and is called the selection pressure parameter. It affects the effective size of the training data. This selection pressure was exponentially decayed as

$$\mathbf{s_e} = \mathbf{s_{e_0}} \exp(\frac{\log(\mathbf{s_{e_{end}}}/\mathbf{s_{e_0}})}{\mathbf{e_{end}} - \mathbf{e_0}})^{(\mathbf{e_{end}} - \mathbf{e_0})} \tag{5}$$

where $\mathbf{e_0}$ and $\mathbf{e_{end}}$ represents first and last epochs. Intuitively, this was done so that different points get selected in different batches instead of selecting the same points repeatedly.

The problems with this approach was the computation time for calculating and sorting these losses for each batch across epochs, which took $\mathbf{O}(\mathbf{N} \log \mathbf{N})$. We propose a simpler method for selecting a batch where for each epoch we first compute the loss for each data point $i$ and and sort these losses in descending order. This sorting is done just once for each epoch. Next, we select a batch of data points $i$ which contributes maximum to the loss. After each iteration, the loss value of only the selected batch is updated and hence it takes only $\mathbf{O}(\mathbf{b} \log \mathbf{N})$ time to update the sorted list, where $b$ is the batch size, which implies $\mathbf{b} << \mathbf{n}$.

Another method of sampling is leverage based. Leverage is essentially a measure of how far away the independent variable values of an observation are from those of the other observations.[1]. In a linear regression model, the leverage score for the i-th data is given by

$$h_{ii} = [H]_{ii} \tag{6}$$

where $h_{ii}$ is the i-th diagonal element of the projection matrix

$$H = X(X^T X)^{-1} X^T \tag{7}$$

[6, 7, 8] developed leverage-based sampling in matrix decomposition. [6] applied the sampling method to approximate the LS solution. [2] derived the bias and variance formulas for the leverage-based sampling algorithm in linear regression using the Taylor series expansion. [9] further provided upper bounds for the mean-squared error and the worst-case error of randomized sketching for the LS problem. [10] proposed a sampling-dependent error bound then implied a better sampling distribution by this bound. However in case of neural networks, where datasets are relatively larger, computations required to calculate leverage is very computationally intensive which completely beats the aim of faster convergence.

One of the papers under review at the time of writing this report[2] discusses the sample importance in training deep neural networks. The importance of a sample is described as the change it induces

[1] https://en.wikipedia.org/wiki/Leverage_(statistics)

[2] https://openreview.net/forum?id=r1IRctqxg

on the parameters. The entire process is carried out by assigning a weight to each sample by which the loss corresponding to that point is scaled during the training. Entire training objective being:

$$\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{L}(\mathbf{y_i}, \mathbf{f}(\mathbf{x_i}, \theta)) + \mathbf{R}(\theta) \tag{8}$$

After scaling loss value of each data point $i$ with its corresponding weight, $v_i$, the objective becomes:

$$\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{v_i} \mathbf{L}(\mathbf{y_i}, \mathbf{f}(\mathbf{x_i}, \theta)) + \mathbf{R}(\theta) \tag{9}$$

So with different sample weights in different iterations, parameter update step can be written as:

$$\theta^{\mathbf{t+1}} = \theta^{\mathbf{t}} - \eta \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{v_i^t} \mathbf{g_i^t} - \eta \mathbf{r^t} \tag{10}$$

where $\mathbf{g_i^t} = \frac{\partial}{\partial \theta^{\mathbf{t}}} \mathbf{L_i}(\mathbf{y_i}, \mathbf{f}(\mathbf{x_i}, \theta^{\mathbf{t}}))$ and $\mathbf{r^t} = \frac{\partial}{\partial \theta^{\mathbf{t}}} \mathbf{R}(\theta^{\mathbf{t}})$

Now the sample importance was defined as

$$\phi_i^t = \frac{\partial}{\partial v_i^t} \Delta \theta^t \tag{11}$$

$$\beta_{i,d}^t = \sum_{j \epsilon Q_d} (\phi_{i,j}^t)^2 \tag{12}$$

$$\alpha_i^t = \sum_j (\phi_{i,j}^t)^2 \tag{13}$$

$$\tau_i = \sum_t \alpha_i^t \tag{14}$$

where $\phi_i^t$ defines parameter affectibility of sample $i$ for $t$th iteration. $\beta_{i,d}^t$ defines the importance of $i$th sample for parameters of $d$th layer. So, $i$th sample's importance for all parameters in model is $\alpha_i^t$ for $t$th iteration.

Their results show that different samples have different importance and hence contribute differently in the weights update as training progresses. The paper also finds out an interesting relation between negative log likelihood and sample importance, wherein although generally, the two quantities show a positive correlation but not every data point with a high value of negative log likelihood had a high sample importance. But the positive correlation between the two kept increasing as the training proceeded.

Although, such a method is good for analysis of effect of sample points in different iterations and different layers, it is not particularly applicable to the training of deep neural network directly. So, we propose various methods to incorporate the idea of non-uniform sampling directly into the training of deep neural networks.

## 3 Methodology

We investigate three non-uniform sampling methodologies. Each methodology was tested on 3 types of neural networks. Below we describe the process in detail.

### 3.1 Non-Uniform Sampling Methods

#### 3.1.1 Objective Based Sampling

Objective based sampling refers to the formation of mini-batches based on the influence each data point has on total loss. The motivation for this lies in the fact that data points capable of producing a significant contribution in loss carry information which can be preferentially learnt by the model to reduce training times. In a sense these loss inducing data points are a form of "important samples"

in the sense that experiencing them more during training can help the network to learn faster. We perform a full batch pass of and keep track of the loss incurred by each data point. This is then used to rank all the data points and then mini-batches are formed for further training based on this ranking.

To ensure that data points causing large loss are always treated preferentially we update the loss of each mini-batch and update the ranks of these data points by their new losses. A pseudo code of the procedure can be seen in algorithm 1.

**Algorithm 1:** Objective Based Sampling

**Result:** Trains on a mini batch using objective based sampling

```
X = Training batch of size N;
Objective = Forward(X);
SortedObjective = Sort(Objective, descending);
for number of Epochs do
    for batch in N/batch size do
        ObjectiveBatch = index of sortedObjective[0:batchsize];
        updatedObjective = train(X[ObjectiveBatch]);
        ;          // Get the updated value of objective for the batch
        for obj in updatedObjective do
            insert obj in sortedObjective ;          // O(log N) using binary search
        end
    end
end
```

### 3.1.2 Gradient Based Sampling

Gradient based sampling refers to the formation of mini-batches based on the influence each data point has on the gradients of loss for this data point calculated with respect to the weights. Intuitively, a datapoint causing higher gradient value for the weights will contribute more to the learning of the network. So we would want to select these datapoints with higher probability. We calculate loss for each data point individually and then compute the gradients of these individual losses with respect to all the weights. After this we calculate the norm of entire gradients matrices corresponding to all the layers and using these norms we assign a score for each data point. A data point with higher score should be more likely to be selected. We normalize these scores into a probability distributions.

We finally use Bernoulli's trials to select data points for the mini-batch. We run a bernoulli trial for each datapoint with probability equal to batch size multiplied by probability of that datapoint. This means the expected values of sum of all these Bernoulli trials is equal to batch size. We select all the datapoints which give 1 as output for the Bernoulli trial. Thus, selecting a different number of input points for each iteration. But their expected value being equal to the batch size. Algorithm 2 shows the pseudo code for the Gradient Based Sampling while Algorithm 3 shows the pseudo code for Bernoulli's trials.

### 3.1.3 Variance Based Sampling

Variance based sampling refers to the formation of mini-batches based on the variance among features in a data point. We calculate variance of each data sample and rank them in decreasing order. We normalize these scores into a probability distribution and use Bernoulli's trials to select data points for the mini-batch.

The intuition behind this comes from the fact data points having high variance contain are harder for the network to learn and therefore learning them preferentially might help the network to converge faster. Algorithm 4 shows the pseudo code for variance based sampling and 3 shows Bernoulli's trials

**Algorithm 2:** Gradient Based Sampling

**Result:** Train on a mini batch using gradient based sampling

```
X = Training batch of size N;
for (num in preProcessingEpochs) do
    train(X) ;                          // find good approximation of weights
end
for i in numData do
    Loss = Forward(X);
    Backward on Loss;
    norm = 0;
    for each Param in ModelParameters do
        grad = param.gradient;
        squareSum = squareSumOfMatrix(grad);
        norm = norm + squareSum
    end
    norm[i] = norm;
end
prob = norm / sum(norm) ;           // normalized probability of selection
for number of Epochs do
    for batch in N/batch size do
        selection = bern(batchSize, prob) ;                          // refer 3
        batch = x[selection];
    end
end
```

**Algorithm 3:** Bernoulli's Trials

**Data:** prob, batchSize

**Result:** Returns a selection array of expected size batchSize

```
for i in numData do
    selection[i] = bernoulliTrial(batchSize*prob)
end
return index where selection[i] == 1
```

**Algorithm 4:** Variance Based sampling

**Result:** Trains on a mini batch using variance based sampling

```
x = trainingData var = zeros of size numData for i in numData do
    var[i] = variance(x[i])
end
prob = var / sum(var) ;             // normalized probability of selection
for number of Epochs do
    for batch in N/batch size do
        selection = bern(batchSize, prob) ;                          // refer 3
        batch = x[selection];
    end
end
```

### 3.2 Neural Network Models

#### 3.2.1 Fully Connected Neural Network

In a fully connected layer, each neuron is connected to every neuron in the previous layer and each connection has an assigned weight. A fully connected neural network model is also sometimes called Deep Neural Network(DNN). The exact structure of the network used by us is as follows

- Three fully connected layers
- Two hidden layers of size 500 and 50 respectively
- Output layer of size 10 corresponding to number of classes

We apply log softmax to the output from the final layer and use negative log likelihood loss to calculate loss.
Figure 1 provides a simplified version of the model.

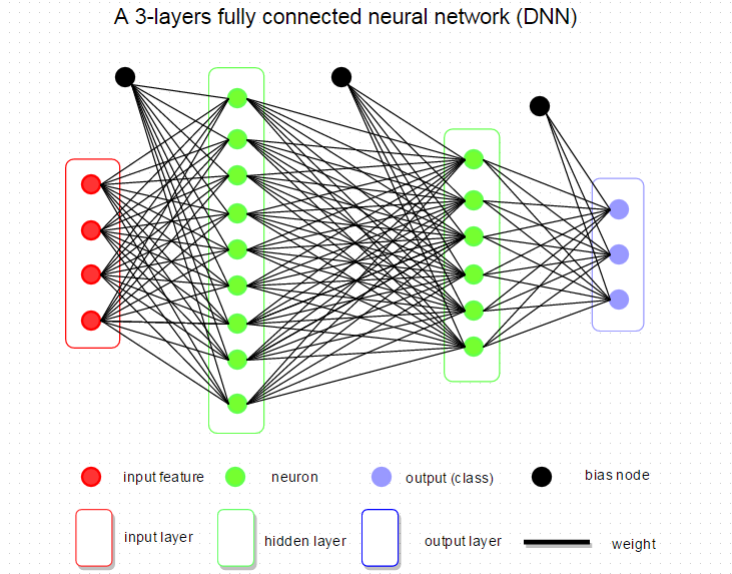



Figure 1: FC architecture used

#### 3.2.2 Convolutional Neural Network

In addition to fully connected layers, convolutional neural networks(or CNNs) also have Convolutional and Pooling layers. CNNs have achieved huge popularity in image recognition tasks. The network used by us can be described as follows

- We use two convolutional layers, each followed by a max pooling layer.
- The output from the second convolutional layer has a dropout layer before the max pooling layer.
- Each max pooling layer has non linearity applied on it using ReLU
- We then pass the outputs through two fully connected layers of size 50 and 10.
- Output from the first fully connected has non linearity and dropout applied to it before being passed through the second fully connected layer
- The second fully connected layer is the output layer
- The output is finally passed through a log softmax layer

Figure 2 provides a simplified representation of the model.

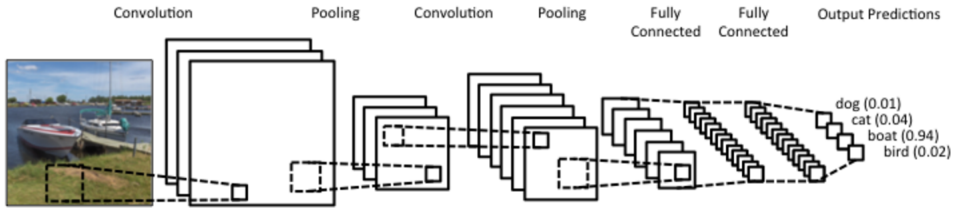
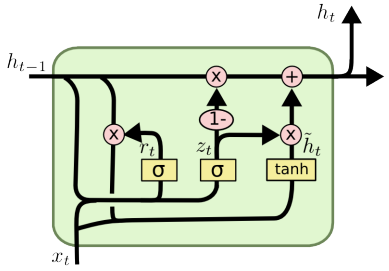



Figure 2: CNN architecture used

### 3.2.3 Long Short Term Memory

In order to investigate effect of our proposed non-uniform sampling techniques on RNNs and LSTMs, we have chosen a network with two layers of LSTM memory blocks stacked on top of each other followed by a fully connected layer. Both the LSTM layers consists of 64 memory blocks. The output of the LSTM blocks is passed through a fully connected layer consisting of 64 neurons.




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: LSTM structure

## 4 Datasets

We utilize two widely available datasets and one synthetic one.

### 4.1 MNIST

The MNIST[11] database consists of 60,000 training samples and 10,000 testing samples. It is a database of handwritten digits. The database is widely used for training and testing in the field of machine learning. The dataset is available at [3] and the digits in it have been size-normalized and centered in a fixed-size image. We do not use any preprocessing and data augmentation methods. Each black and white image is of size 28*28 where each pixel has a value in range(0, 255) and these values act as input features.

MNIST is widely used in neural network community for testing and training of neural nets and since two of our models are type of neural networks therefore using MNIST makes sense.

### 4.2 CIFAR10

CIFAR10[12] consists of 50000 training and 10000 test images. Each of the 60000 images is a 32*32 color image and is spread across 10 classes. Each class has 6000 images. It offers additional

[3] http://yann.lecun.com/exdb/mnist/

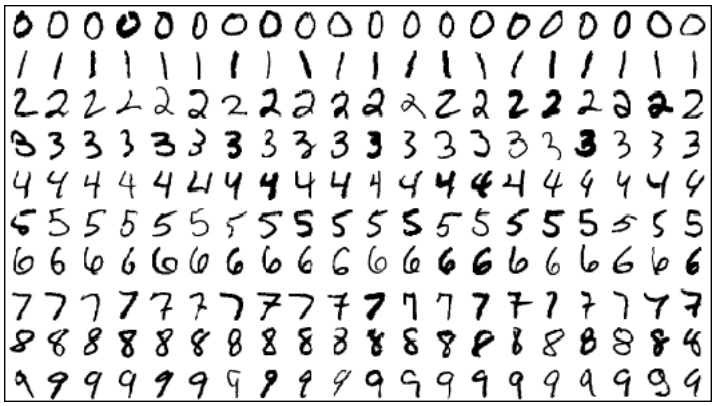


Figure 4: MNIST samples

information than MNIST by inclusion of color and objects from different walks of life. Like MNIST it is widely used for testing and training of machine learning models. The dataset is available at [4].

Each color image has three channels, each of size 32*32. Each of these pixels as a feature. CIFAR10 is widely used in neural network community for testing and training of neural nets and since two of our models are type of neural networks therefore using CIFAR10 makes sense.

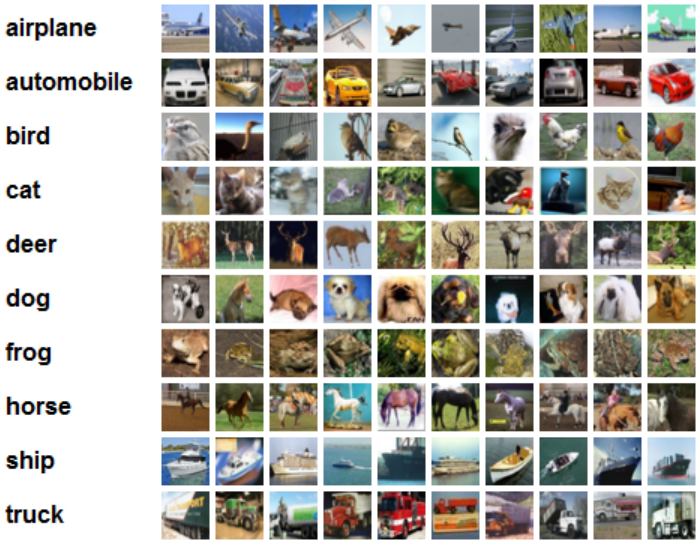



Figure 5: CIFAR10 samples

### 4.3 Sine Wave Data

We generate synthetic data in form of sine wave. Since we can't use MNIST or CIFAR-10 datasets with LSTM, since it requires a time series data. Additionally, we are using synthetic data to control the amount of input fed to our LSTM model which takes a very long time to train. Thus by creating our own data we can reduce the training times for the sake of the project.

We generate 1000 samples where each sample has a feature vector length of 1000.

[4] https://www.cs.toronto.edu/ kriz/cifar.html

# 5 Experiments

## 5.1 Rationale

Our goal is to show how non uniform sampling can result in faster convergence during training period of neural networks. Therefore we perform multiple rns of CNN, FN and LSTM on their respective datasets. Each run differs in the type of sampling methodology used. We claim that non-uniform sampling methods can perform better than uniform sampling techniques, therefore runs having uniform sampling as its sampling methodology serve as baseline.

To show this we keep track of duration of each epoch for all the runs of different sampling methodologies as well as the training loss after each epoch. We can then use these two metrics to definitily say whether or not a sampling method is better than the other.

We also measure performance of these runs after every epoch on validation and test sets. Validation sets are used for tuning of the hyperparameters as discussed in subsection 5.3. Test sets are used to see the actual generalization results of our models after training them.

## 5.2 Datasets Used

We utilized MNIST and CIFAR-10 for evaluating sampling methods on CNN and FC and utilize synthetic data in form of Sine wave for use on LSTM. MNIST and CIFAR1-10 were chosen as they are industry standard datasets for evaluating neural network models specially convolutional and fully connected ones.

LSTMs are notorious for time consumption during training therefore we utilized synthetic data in form of sine wave data to train the LSTM.

### 5.2.1 Preprocessing

- **MNIST** : The MNIST dataset available is already size normalized and image centered and hence no preprocessing was done.
- **CIFAR-10** : The CIFAR-10 was preprocessed to be transformed to 0 mean and 1 standard distribution.
- **Sine Wave** : No preprocessing was done.

## 5.3 Hyperparameters

Hyperparameter tuning was done by evaluating performance on validation set. A hold out validation set was made from 20% of the training data and it was used to evaluate hyperparameters. Same split of training and validation was guarenteed across runs by setting the seed for the random number generator.

### 5.3.1 Optimizer

We utilized the prevelant Adam[13] optimizer. Adam combines the goodness of AdaGrad[14] and RMSProp and decays the learning rate appropriately. It is being used frequently in the deep learning community and provides stable results.

### 5.3.2 Learning Rate

Learning rate was optimized by doing a grid search over 10 values selected in logspace between $10^{-5}$ and $10^{-2}$. Figure 6 and 7 show validation set loss per epoch for different learning rates tried.

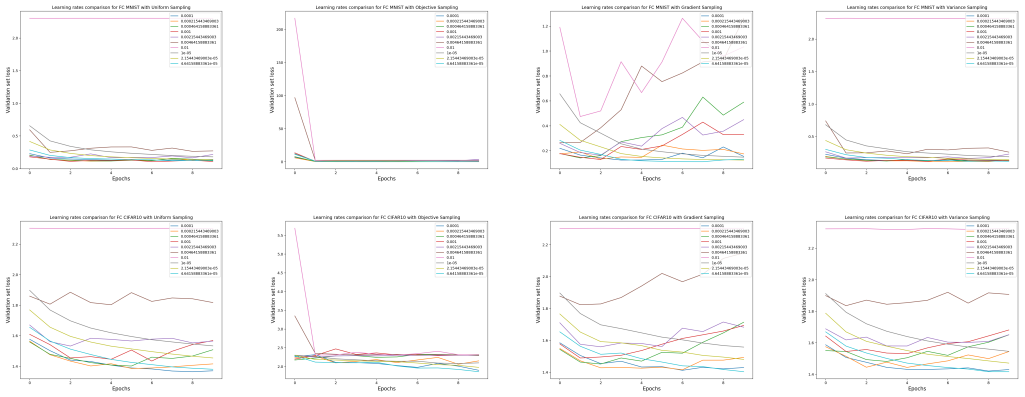


Figure 6: FC network learning rate tuning on MNIST and CIFAR10 with different sampling methods

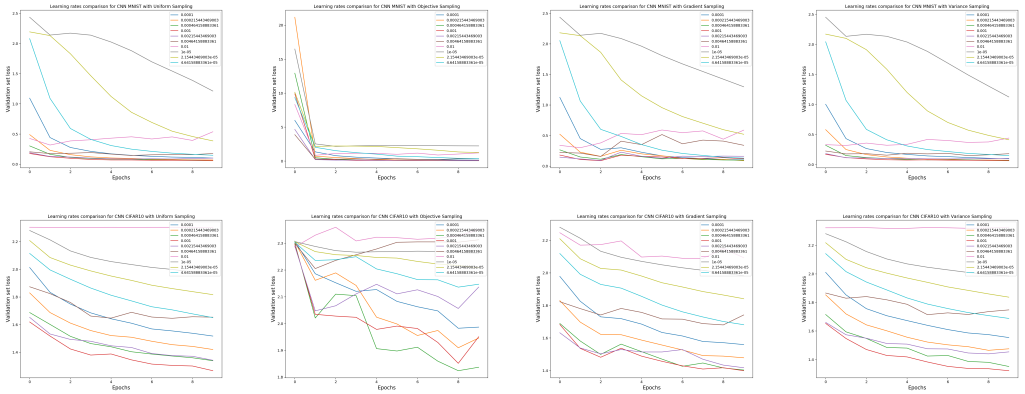


Figure 7: CNN learning rate tuning on MNIST and CIFAR10 with different sampling methods

### 5.3.3 Batch size

Batch size directly effects the duration of an epoch[5], therefore we fixed the batch size at 64 for fair comparison across all runs.

## 5.4 Performance Metrics

Since we are contesting the claim that non-uniform methods of sampling can resulting in faster convergence of neural networks during training phase, we use duration of each epoch and the training loss after each epoch as performance metric. Each final combination of hyperparams was run three times and the results were averaged.

## 5.5 Computational Resources

All the experiments were run on Ryzen 6 1600 processor(6 core, 12 threads) with Nvidia GeForce 1070 GPU.

# 6 Results

## 6.1 Two layer Fully Connected Network

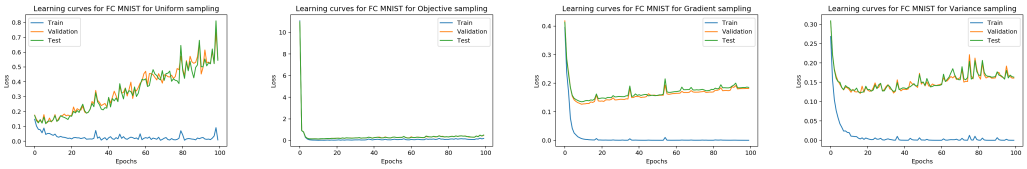
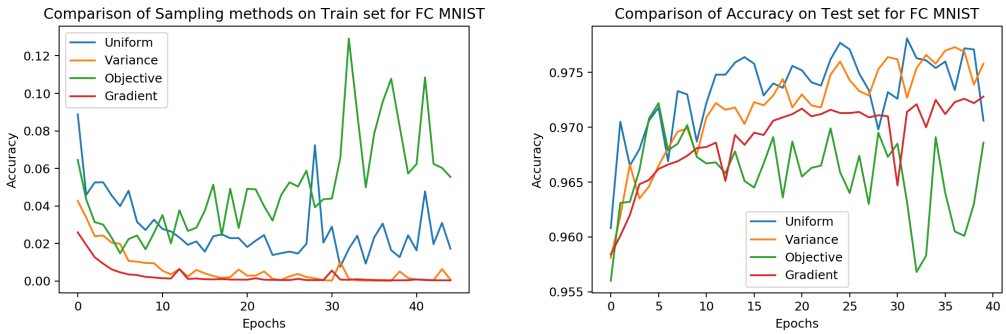



Figure 8: Fully Connected train, val and test loss on MNIST with different sampling methods



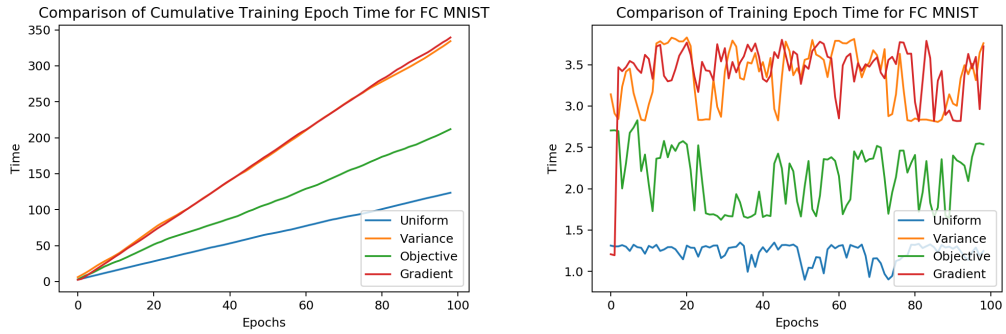

(a) Loss on training set

(b) Test set performance

Figure 9: Fully Connected performance comparison on MNIST with different sampling methods

As can be seen in 9, training loss of gradient based and variance based sampling methods quickly converges to their minimum values. They take almost half as many epochs as needed by the uniform sampling. However, objective based sampling gives higher loss compared to uniform sampling. On test set accuracy we can see that uniform sampling gives slightly better accuracies.

But key point to note here is: during training, using non-uniform sampling methods, esp. gradient-based and variance-based the network reaches its smallest loss in a lot few epochs than the uniform sampling method. This shows, in this case we got a faster convergence towards the optima using the non-uniform methods - gradient based and variance based methods.




(a) Total training time (in ms)

(b) Training time for each epoch (in ms)

Figure 10: Fully Connected training time for different sampling methods on MNIST

As expected, we can see in 10, the time taken for each epoch by non-uniform sampling methods is higher than uniform sampling methods, since uniform sampling need not do any extra work for samples selection. The same reason results in increasing the total time of running same number of epochs with non-uniform sampling methods. But as we saw above, we can get to better convergence with lesser epochs using non-uniform methods so this should not be a problem.

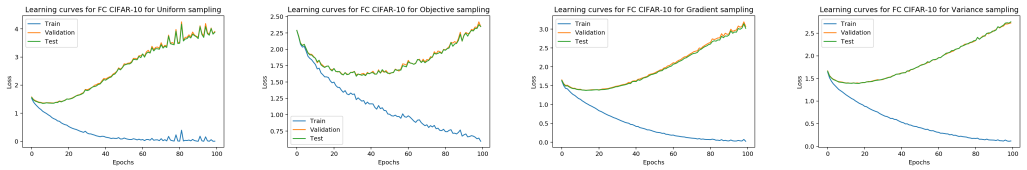
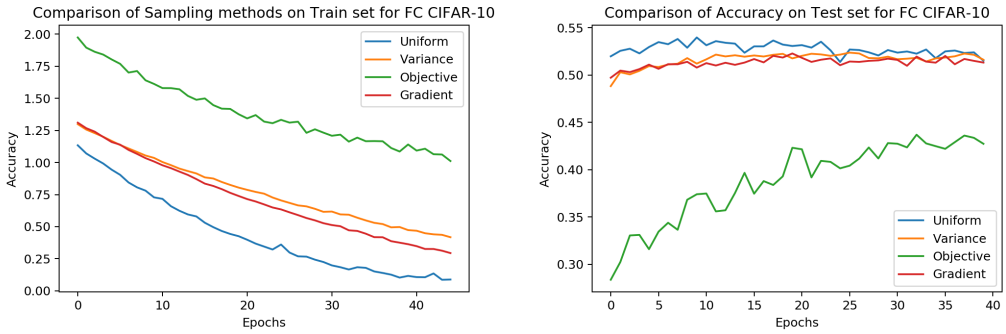



Figure 11: Fully Connected train, val and test loss on CIFAR10 with different sampling methods

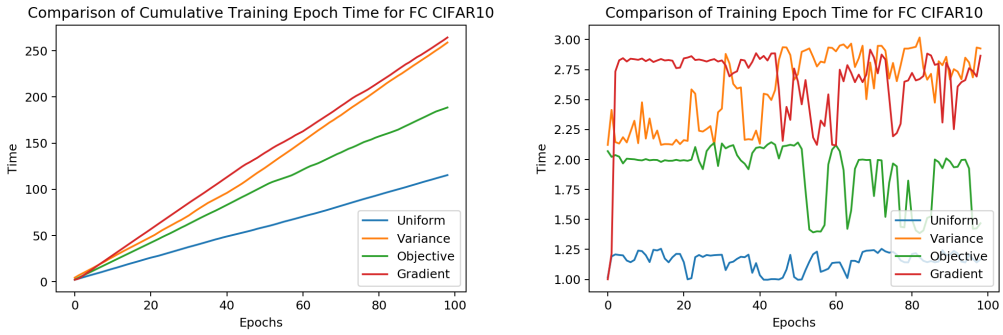



(a) Loss on training set

(b) Test set performance

Figure 12: Fully Connected performance comparison on CIFAR10 with different sampling methods

As can be seen in 12, training loss of uniform sampling method is better than all the non-uniform sampling methods across epochs. Although gradient-based and variance-based methods come close to uniform sampling. This shows that uniform sampling performs better on CIFAR-10 dataset. The same thing can be shown on the test set performance where gradient-based and variance-based methods give similar performance as that achieved by uniform sampling method. Objective based sampling consistently gives poor results.




(a) Total training time (in ms)

(b) Training time for each epoch (in ms)

Figure 13: Fully Connected training time for different sampling methods on CIFAR10

As expected, we can see in 13, non-uniform sampling methods take longer to run in each epoch. As expected, gradient-based sampling takes the most time since computing gradient, as well as sampling

the data set using these gradients is an expensive process. Since we get similar performance in terms of loss and accuracy by using non-uniform and uniform sampling methods but non-uniform methods take longer so this wasn't beneficial in case of CIFAR10 dataset.

### 6.2 Convolutional Neural Network

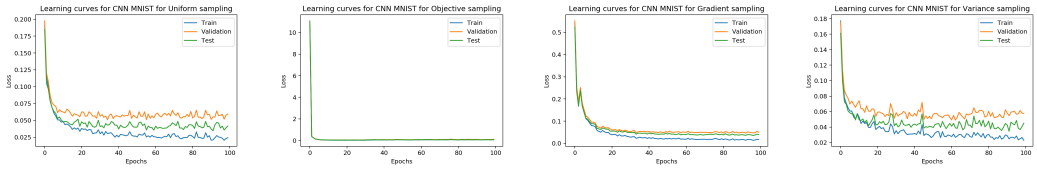
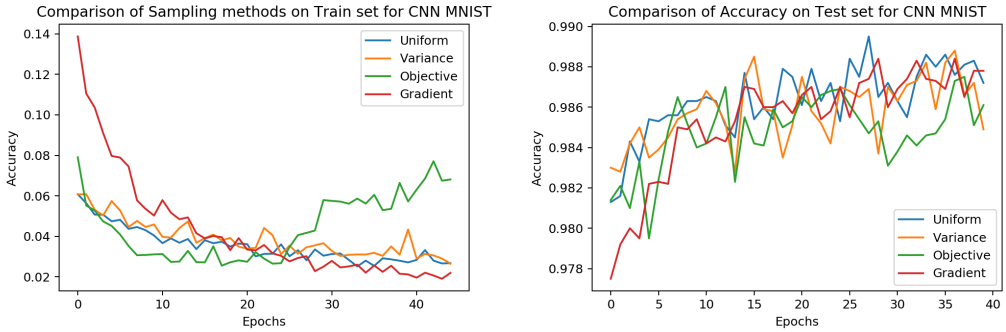



Figure 14: CNN train, val and test loss on MNIST with different sampling methods



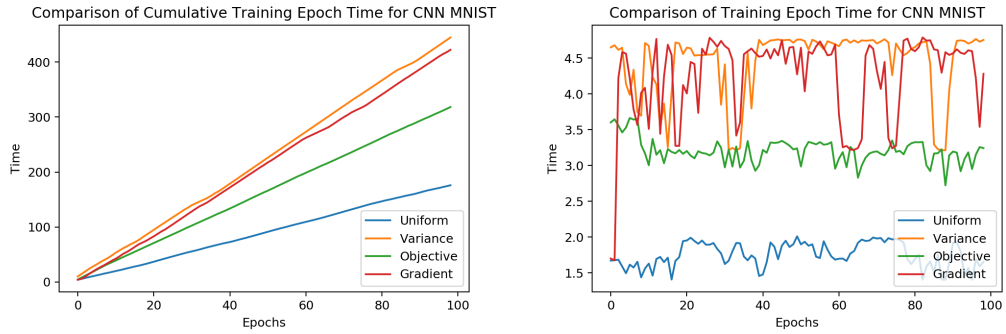

(a) Loss on training set

(b) Test set performance

Figure 15: CNN performance comparison on MNIST with different sampling methods

As can be seen in 15, training loss of different sampling techniques is comparable but gradient based and variance based sampling methods give smaller loss compared to other methods after few epochs. Objective based sampling gives slightly higher loss compared to uniform sampling. After a few epochs we can see the test accuracies giving similar results as well for all the sampling methods. Gradient based sampling starts poorly but quickly catches up with other sampling methods and eventually gives almost same results as uniform sampling method.

The number of epochs required for convergence is also similar for all the sampling methods. Objective based method continues to perform poorly.

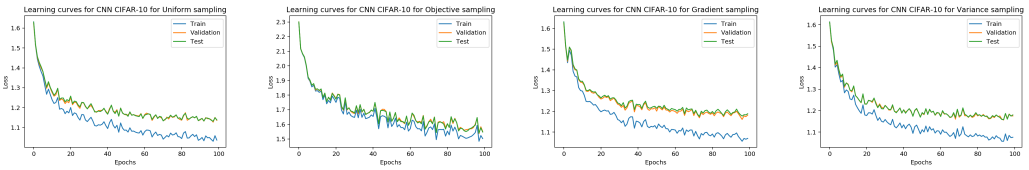


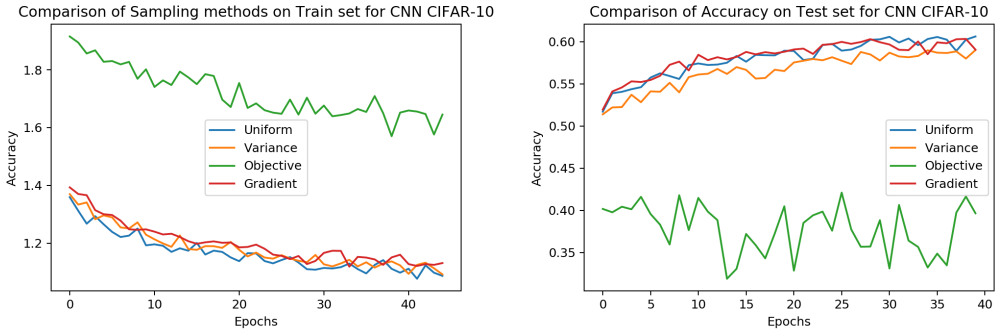

(a) Total training time (in ms)

(b) Training time for each epoch (in ms)

Figure 16: CNN training time for different sampling methods on MNIST

As expected, we can see in 16, the time taken for each epoch by non-uniform sampling methods is higher than uniform sampling methods, since uniform sampling need not do any extra work for samples selection. The same reason results in increasing the total time of running same number of epochs with non-uniform sampling methods. Since we are not getting convergence on lesser epochs using non-uniform sampling methods, it doesn't seem to be useful for using on CNN with MNIST dataset.



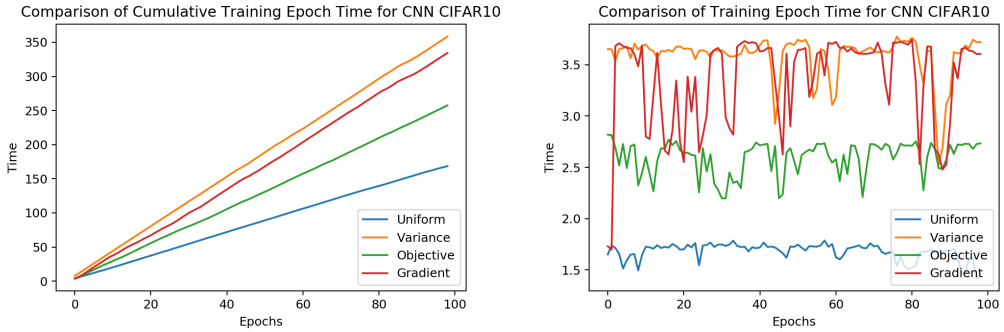




(a) Loss on training set

(b) Test set performance

Figure 17: CNN performance comparison on CIFAR10 with different sampling methods

As can be seen in 17, objective based methods give horrible results. But training loss of all the other sampling techniques is comparable and they take similar number of epochs to reach their optima. The same is reflected in test set performance as well where all the other methods except from object based give similar accuracies across epochs.




(a) Total training time (in ms)

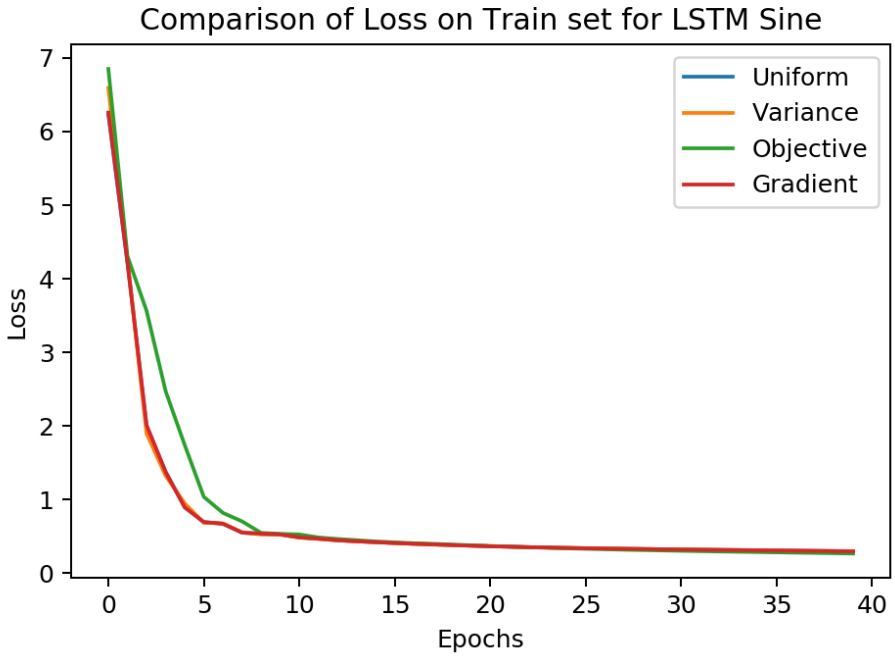
(b) Training time for each epoch (in ms)

Figure 18: CNN training time for different sampling methods on CIFAR10

As expected, we can see in 18, the time taken for each epoch by non-uniform sampling methods is higher than uniform sampling methods, since uniform sampling need not do any extra work for samples selection. The same reason results in increasing the total time of running same number of epochs with non-uniform sampling methods. Since both uniform and non-uniform sampling

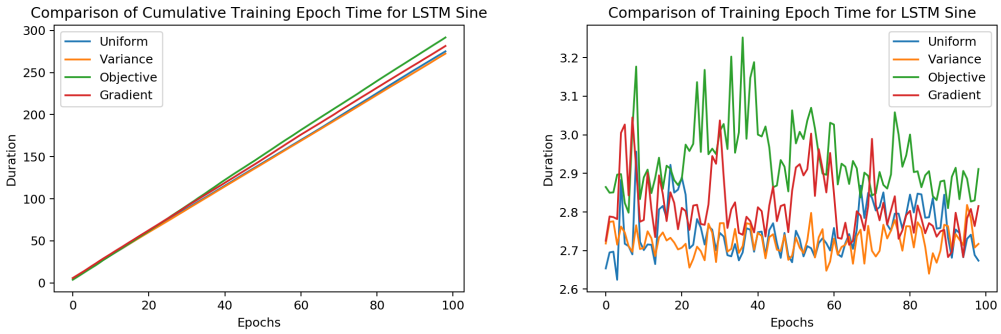
methods are giving similar results but non-uniform sampling takes longer to run. Hence, it is not useful to use these methods eith CNN on CIFAR10 dataset.

### 6.3 LSTM Network




(a) Loss on training set

Figure 19: LSTM performance comparison on Sine data with different sampling methods




(a) Total training time (in ms)

(b) Training time for each epoch (in ms)

Figure 20: LSTM training time for different sampling methods on Sine data

Gradient based sampling initially performs better than objective-based and uniform sampling. However after some epochs uniform sampling takes the lead and performs better than all the sampling methods. We were not able to get any conclusive or meaningful results from the LSTM networks using any of the sampling methods.

## 7 Discussion and Conclusion

In our project we investigated various methods of non-uniform sampling and whether they can provide any advantage over uniform sampling methods by giving faster convergence during training. Out of the three methods we investigates, viz, Objective based, Gradient based and Variance based, objective based method gave poor performance in terms of speed and accuracy across all datasets and all types of neural networks used. Variance based and gradient based methods were able to give good results in terms of accuracy and loss in CIFAR dataset but not as good as uniform sampling method. But they gave better result than uniform sampling on Fully Connected Networks using MNIST dataset.

The biggest problem with non-uniform sampling methods was the time taken sample selection. As future work if we can come up with better approaches to drive down the sample selection time they might be able to generalize better across models and datasets. As an example, the biggest time taking step of gradient based sampling was the calculation of gradients for each data point. However this process can be parallelized since gradient calculation for one data point is completely independent of the other data points. Thus sampling time can be reduced by a factor of number of parallel threads available in the machine. Similarly for objective based sampling, we can improve our sampling algorithm by randomly sampling from the dataset using Poisson sampling with selection probability proportional to the objective score of the sample. Also we can update loss for entire dataset after every epoch so that our objective approximation does not get too far from the actual objective.

Previous literature [3] and [1] showed improvements in convergence times by using gradient based approach on least squares problems. Our results seem to agree with them on MNIST with Fully Connected Network but not otherwise.

We were unable to get any conclusive results on LSTM networks. This can be due to two reasons, first, because of the dataset used which was very limiting thus it was unable to show any differences across different methods. Secondly, LSTM models are harder to train and require significant amount of time and resources to train properly. In future we can try to train LSTM networks with better datasets and more training time.

Although intuitively the idea of non uniform sampling seemed like it can produce better results compared to uniform sampling at the cost of time spent on sampling selection. The idea was that the improvement in convergence rate would make up for the time spent in sample selection. The results however proved otherwise. We were not able to improve much on the accuracy and loss but as expected sampling time was increased. However this topic of non-uniform sampling still needs more exploration which others can look into.

## References


[1] Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. *CoRR*, abs/1511.06343, 2015.

[2] Ping Ma, Michael Mahoney, and Bin Yu. A statistical perspective on algorithmic leveraging. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 91–99, Bejing, China, 22–24 Jun 2014. PMLR.

[3] Rong Zhu. Gradient-based sampling: An adaptive importance sampling for least-squares. In *NIPS*, 2016.

[4] Léon Bottou and Yann LeCun. Large scale online learning. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[5] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[6] M.W. Mahoney P. Drineas and S. Muthukrishnan. Sampling algorithms for l2 regression and applications. In *In Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms,pages 11271136*, 2006.

[7] M.W. Mahoney P. Drineas and S. Muthukrishnan. Relative-error cur matrix decomposition. In *SIAM Journal on Matrix Analysis and Applications, 30:844881*, 2008.

[8] M.W. Mahoney and P. Drineas. Cur matrix decompositions for improved data analysis. In *Proceedings of the National Academy of Sciences, 106:697702*, 2009.

[9] G. Raskutti and M.W. Mahoney. A statistical perspective on randomized sketching for ordinary least-squares. In *In Proc. of the 32nd ICML Conference*, 2015.

[10] R. Jin T. Yang, L. Zhang and S. Zhu. An explicit sampling dependent spectral error bound for column subset selection. In *In Proc. of the 32nd ICML Conference*, 2015.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[12] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.