Submodular Importance Sampling for Neural Network Training

Krishna Kant Singh

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

Department of Computer Science

June 2018

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Krishna Kant Singh)

_____

(Roll No.)

Approval Sheet

This Thesis entitled Submodular Importance Sampling for Neural Network Training by Krishna Kant Singh is approved for the degree of Master of Technology from IIT Hyderabad

_____

(————-) Examiner
Dept. of Computer Science
IITH

_____

(————-) Examiner
Dept. of Computer Science
IITH

_____

(Dr Vineeth N Balasubramanian) Adviser
Dept. of Computer Science
IITH

_____

(————-) Co-Adviser
Dept. of Computer Science
IITH

_____

(———) Chairman
Dept. of Computer Science
IITH

Acknowledgements

Firstly, I would like to thank my thesis advisor, Dr. Vineeth N Balasubramanian for introducing me to the field of Machine Learning. I thank him for his helpful discussion and patience which helped me a lot. Next, I want to thank Dr. Naveen Sivadasan whom I worked with on the problem of Sub-Graph classification. It was a joy to work with him. I also want to extend a thank you to Dr. Kotaro Kataoka how actually helped a lot during my initial research period. I would like to express my constant gratitude to Dept of Computer Science IIT Hyderabad for providing the motivation and resources to help me in successfully completing my work. I am also thankful to my seniors and friends for their friendly advice and words of encouragement in the due course of my research. I would like to especially thank Krutika Verma for helpful conversation and clearing a lot my doubts. Finally, I want to thank my family without whom none of this was possible.

Abstract

Stochastic Gradient Descent(SGD) algorithms are the workhorse on which Deep Learning systems have been built upon. The standard approach of uniform sampling in SGD algorithm leads to high variance between the calculated gradient and the true gradient, consequently resulting in longer training times.
Importance sampling methods are used for sampling mini-batches that reduce this variance. There exist provable importance sampling techniques for variance reduction but, they generally do not fare well in the case of Deep Learning models.

Our work proposes sampling strategies that create diverse mini-batches which consequently leads to the reduction in the variance of the SGD algorithm. We pose the task of creation of such mini-batches as, maximization of a submodular objective function. The proposed submodular objective function samples minibatches that such that more uncertain and diverse set of samples are selected with high probability.

Submodular functions can be optimized easily using the GREEDY[1] algorithm but, even the newer variants suffer from performance issues when the size of the dataset is large. We propose a new faster submodular optimization method which is inspired by [2].

We prove theoretically that our sampling scheme reduces variance in the case of SGD algorithm. We also show that Determinantal point process(DPP) sampling can also be seen as a special case of our algorithm.

We showcase the generalization of our method by testing it on several deep learning data sets like MNIST, FMNIST, CIFAR-10 datasets. We study the effect of learning rate, network architecture etc on our proposed method. We study how different features affect the performance of our algorithm. We also study the case of transfer learning with our algorithm used for selection of the dataset. In all the experiments, we compare our algorithm with Loss based sampling and Random sampling for comparison.

Contents

Part I

Introduction

Chapter 1

Introduction and Motivation

Stochastic Gradient Descent(SGD) methods are one of the main reasons for the success of Deep Learning. With the availability of "Big-Data" stochastic methods have proven to be the most effective methods for training Deep Neural Networks.

In machine learning, we are interested in minimizing the loss function generally of the form,

$$F(\theta) = \textbf{argmin} \frac{1}{N} \sum_{i=1}^{N} f(x_i; \theta)$$

Typically, f is convex and denotes the misfit between the i-th training instance and its corresponding target. The training data is represented as x_i, y_i and N denotes the total number of data points. A typical way of solving these optimization problems is through gradient descent algorithms,

$$\begin{aligned} \theta_{t+1} &= \theta_t - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla(; \theta_t) \\ \eta &: \text{learning rate} \\ \nabla(; \theta_t) &= \sum_{i=1}^{N} \nabla(f(x_i; \theta_t)) \end{aligned} \tag{1.1}$$

The above formulation requires gradients for each of the N data points, which is prohibitively expensive even for a small N. SGD provides a solution to above problem where the parameter update is conditioned upon only a single data point selected uniformly at random

$$\theta_{t+1} = \theta_t - \eta \nabla(x_i, \theta_t) \tag{1.2}$$

It can be shown that $\nabla(x_i, \theta_t)$ is an unbiased estimate of the full batch gradient $\nabla(:, \theta_t)$, $\mathbb{E}(\nabla(x_i, \theta_t)) = \frac{1}{N} \sum_{i=1}^{N} \nabla f(x_i; \theta_t)$ but, SGD introduces high variance between the estimated gradient $\nabla(x_i, \theta_t)$ and the true gradient $\nabla(; \theta_t)$, due to this reason stochastic gradients methods have a much slower convergence. A simple way

of reducing the variance is to use mini-batch SGD.

$$\theta_{t+1} = \theta_t - \frac{1}{k}\eta \sum_{i=1}^{k} \nabla(x_i, \theta_t) \tag{1.3}$$

k : batch size

1.1 Importance Sampling

Creation of a minibatch which is representative of the full data set can help in reducing the variance due to the stochastic sampling. Higher importance should be given to data points that provide the most information to the network, in literature this called as hard example mining[4]. This way of choosing the data points is also popularly called Importance Sampling.

In standard SGD full gradient is approximated using a Monte Carlo estimate given by

$$\tilde{\nabla}(f(;\theta_t)) = \frac{1}{B}\sum_{i=1}^{B} p(x_i)\nabla(f(x_i;\theta_t)) \tag{1.4}$$

$$= E_{p(x)}\nabla(f(x_i;\theta_t) \tag{1.5}$$

Here $p(x)$ denotes the sampling distribution that is used for selecting B data points ie , $P(x = x_i) = p(x = x_i) = \frac{1}{B}$

This approximation can be improved using Importance Sampling as follows

$$\begin{aligned}
\tilde{\nabla}(f(;\theta_t)) &= \frac{1}{B}\sum_{i=1}^{B} p(x_i)\nabla(f(x_i;\theta_t)) \\
\tilde{\nabla}(f(;\theta_t)) &= \frac{1}{B}\sum_{i=1}^{B} \frac{p(x_i)}{q(x_i)} q(x_i)\nabla(f(x_i;\theta_t)) \\
\tilde{\nabla}(f(;\theta_t)) &= E_{q(x)}\left(\frac{p(x_i)}{q(x_i)}\nabla(f(x_i;\theta_t))\right)
\end{aligned} \tag{1.6}$$

Here q(x) denotes the surrogate sampling distribution, that is used for selecting B data points ie , $P(x = x_i) = q(x = x_i)$

In chapter 3 we show that if q(x) is chosen appropriately, is an unbiased estimator. Also, we prove that using importance sampling leads to variance reduction.

1.2 Importance Sampled SGD

For importance sampling based SGD we can write the update rule as follows,

$$\begin{aligned}
\theta_{t+1} &= \theta_t - \eta \sum_{i=1}^{N} w_{I_t}\nabla(f(x_{I_t};\theta_t)) \\
&= \theta_t - E_{p(x)}(w_{I_t}\nabla(f(x_{I_t};\theta_t))
\end{aligned} \tag{1.7}$$

The probability of sampling a data point x_i is given by $P(x = x_{I_t}) = p^t(x_i)$. We can see that p(x) is adaptive and changes at each iteration t, w_{I_t} denotes the rescaling coefficients for x_{I_t}. The above reduces to standard SGD if $w_{I_t} = 1$ and $p^t(x_i) = \frac{1}{N}$

$$\theta_{t+1} = \theta_t - \frac{\eta}{N} \sum_{i=1}^{N} \nabla(f(x_{I_t}; \theta_t)) \tag{1.8}$$

$$\tag{1.9}$$

Some known strategies for importance sampling in SGD are to sample data points proportional to the loss, or to gradients of the parameters. Sampling with respect to gradients is essentially a case of hard example mining.
The drawback with gradient-based sampling is that it requires the full gradient and is sensitive to outliers. Moreover, mini-batches should not only consist of the most informative data points but also points that are diverse enough to represent the full data set for eg it is better to include data points from all classes rather than just one class. This becomes much more important in the case of imbalanced datasets.

1.3 Diversity

The diversity of mini-batch is an important quality for circumventing bias in mini-batches. A popular method for modeling diversity in literature is to use Submodular Functions. A submodular function is defined as,
A function $f : 2^V \longrightarrow R$ is submodular if for any A$\subseteq$ B $\subseteq$ V, and v $\in$ V/B,we have that:

$$f(A \cup v) - f(A) \geq f(B \cup v) - f(B)$$

using this definition we can see that submodular functions are good candidates for modeling diversity. The above property is popularly known as the diminishing return property. A submodular function is ideally suited for selection of a diverse set of data points. Also, a submodular function is easy to maximize as they just require the greedy solution at each step which makes their optimization fast.

Another popular approach for modeling diversity is Deterministic Point Process(DPP) which use a similarity kernel $\mathbf{L}^{N \times N}$, $\mathbf{L}_{ij}$ denoting the similarity between data points i and j for sampling diverse data points.
The DPP assigns a probability of sub sampling any subset Y $\subseteq \{1, ..., N\}$, proportional to the determinant of the sub-matrix $\mathbf{L_Y}$ of $\mathbf{L}$,

$$P(Y) = \frac{\det(\mathbf{L}_Y)}{\det(\mathbf{L} + I)} \propto \det(\mathbf{L}_Y) \tag{1.10}$$

For instance, if Y consists of only two elements i and j, then P(Y) $\propto \mathbf{L}_{ii}\mathbf{L}_{jj} - \mathbf{L}_{ij}\mathbf{L}_{ji}$ where $\mathbf{L}_{ij}$ and $\mathbf{L}_{ji}$ measure the similarity between elements i and j, being more similar lowers the probability of co-occurrence. On the other hand, a diverse subset has a higher value of the determinant. Thus, DPP naturally favours a diversified selection of subsets.

DPP and other similar models come under the class of functions called Probabilistic Submodular Functions(PSF). A major problem in using DPP for minibatch selection is that sampling from a DPP is an expensive operation it scales polynomially with the number of data points size $O(N^3)$.
Thus, a need arises for methods that model diversity as well as DPP but, are much easier to sample from. Another, issue with DPP is that it is hard to always define a meaningful similarity kernel for datasets.

The main contributions of our work are as following

- Designing a sampling scheme that encodes both the importance of sample and diversity for the creation of representative batches for the whole data set.
- Constructing a method for scalable optimization of submodular objective function, thus enabling their use in the era of Big Data.
- We prove that our method is an unbiased estimator for the full gradient and that it reduces variance in SGD and SGD-Type algorithms.
- Our designed approach leads to faster convergence and higher classification accuracy in the image classification task. Out methods is also more robust to the hyperparameters such as learning rate and minibatch size.

As far as we know our work is one of the first to use submodular functions for importance sampling in the context of variance reduction in SGD. Also, we are one of the first papers that perform large-scale importance sampling based image classification task on datasets such as CIFAR-10 and MIT-67 datasets.

Part II

Prerequisites

Chapter 2

Deep Neural Networks and Stochastic Gradient Descent

In this chapter, we first give a brief overview of Neural Networks and Deep Learning. We then show how the Backpropagation algorithm along with SGD(Stochastic Gradient Descent) is used to train a neural network.

Machine learning is commonly divided into two main types:

- Supervised Learning: These methods try to learn a hypothesis $\mathcal{H} : X \rightarrow Y$ which maps the input data domain to the output data domain as well as possible. If the output set Y is finite, we say that the task is a classification task and if it is continuous we say that it is a regression task.
- Unsupervised Learning: If all we have is the input data X and no output data to guide our training, the task is called unsupervised learning. The focus in this type of learning is to discover the hidden structure in the data. Common problems in unsupervised learning are dimensionality reduction and clustering of the input data.

Recently deep Learning has revolutionized the field of both supervised and Unsupervised Learning. Tasks which once seemed impossible to be done by a machine are now being excelled at. ImageNet Classification challenge which sparked the boom for the Deep Learning Research is now a solved problem with the machines achieving a super-human level of accuracies on the task.

Though deep learning methods still suffer from some huge problems like

- Large Dataset: Deep Learning Networks require huge amounts of data to be trained. It is not always possible to have access to such data. There is a need for algorithms that train faster and better using only a small subset of data.
- Performance on Imbalanced datasets: Deep learning system generally perform poorly on imbalanced datasets, this is due to the fact that the network is unable to learn classes/events which occur very sparsely in the dataset.

Our work tries to mitigate both of the above points through the use of importance sampling for the training of Deep Neural Networks.

In the next section we derive the now popular back propagation algorithm used to train neural networks.

2.1 Neural Network

[1] We take a Neural Network (NN) as shown in fig 2.1 with the following parameters,

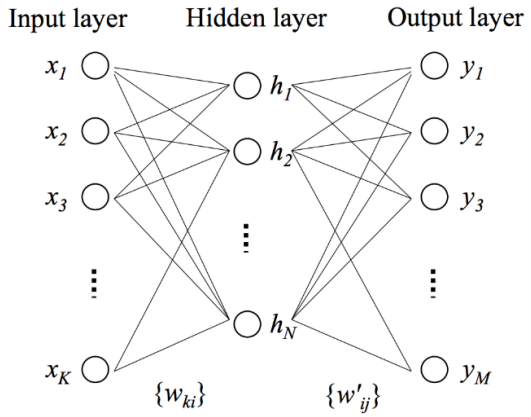


Figure 2.1: A one layer NN network

- Input vector of length of K, $x = (x_1, x_2, x_3, ..., x_K)$.
- Hidden layer consists of N neurons $h = (h_1, h_2, h_3, ..., h_N)$.
- Ouput layer is of size M, $\mathbf{y} = (y_1, y_2, y_3, ..., y_M)$.

Every neuron in the input layer is connected to all the neurons in the hidden layer, the connection weight between neuron i and k is given by w_{ki}. Similarly, all the neurons in the hidden layer are connected to the output layer, the connection weight between the ith hidden neuron and j th output neuron is given by w'_{ij}.
We can think of w_{ki} as the k,i entry in the matrix $\mathbf{W}^{K\times N}$, similarly w'_{ij} can be considered as the i,j entry in the $\mathbf{W}^{K\times N\prime}$,. The activation function f is the sigmoid activation function.

Forward Pass The forward pass can be expressed by the following equations,

$$h_i = f(u_i) = f(\sum_{k=1}^{K} w_{ki}x_k) \text{[This is the ouput of the i th neuron in the hidden layer]} \tag{2.1}$$

$$y_j = f(u_j^{'}) = f(\sum_{i=1}^{K} w_{ij}^{'}h_i) \text{[This is the ouput of the j th neuron in the ouput layer]} \tag{2.2}$$

$$\text{if we assume a Mean Squared Error Loss} \tag{2.3}$$

$$E(x) = \frac{1}{2}\sum_{j=1}^{M}(y_j - t_j)^2 \tag{2.4}$$

[1] In this section subscript$_i$ denotes denotes i th dimension. But everywhere else subscript$_i$ denotes i th data point

Backward Pass

In the backward pass we compute the loss and gradient of loss wrt(with respect to) to all the parameters($\mathbf{W}$) in the NN. We the pass these gradient back and update the parameters of the network.

$$\frac{\partial E(x)}{w_{ij}^{'}} = \frac{\partial E(x)}{\partial y_j}\frac{\partial y_j}{\partial u_j'}\frac{\partial u_j'}{w_{ij}'} \tag{2.5}$$

$$\frac{\partial E(x)}{\partial y_j} = \sum_{i=1}^{M}(y_j - t_j) \tag{2.6}$$

$$\frac{\partial y_j}{\partial u_j'} = y_j(1 - y_j) \tag{2.7}$$

$$\frac{\partial u_j'}{w_{ij}'} = h_i \tag{2.8}$$

Hence,

$$\frac{\partial E(x)}{w_{ij}^{'}} = \sum_{i=1}^{M}(y_j - t_j)y_j(1 - y_j)h_i \tag{2.9}$$

Similarly we can get,

$$\frac{\partial E(x)}{w_{ki}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial u_j'}\frac{\partial u_j'}{\partial h_i}\frac{\partial h_i}{\partial u_i}\frac{\partial u_i}{\partial w_{ki}} \tag{2.10}$$

This is the famous Backpropogation algorithm it is so called, because we are back propagating the gradients from output to the input.

The update rule for parameters is given by

$$w_{ij}'^{+} = w_{ij}' - \eta * \frac{\partial E(x)}{w_{ij}'} \tag{2.11}$$

where η denotes the learning rate.

2.1.1 Gradient Descent

Back propagation only computes the gradient for only one sample, in practice back propagation is generally combined with learning algorithms such as gradient descent which computes the gradient for all the samples in the training data. Hence, Gradient Descent update rule is given by

$$w_{ij}'^{+} = w_{ij}' - \frac{\eta}{N}\sum_{x}\frac{\partial E(x)}{w_{ij}'} \tag{2.12}$$

For ease of discussion we now employ the following convection for parameter updates

$$\theta_{t+1} = \theta_t - \frac{\eta}{N}\sum_{i=1}^{N}\nabla(f(x_i;\theta_t)) \tag{2.13}$$

Here f denotes the Loss function and x_i denotes the i th point in the dataset $\mathcal{D}$ of size N . We can see that computing the gradient for all of the N data points is computationally expensive. In practice only a part of dataset is used for computing the gradient and updating of the parameters in one iteration.

2.1.2 Stochastic Gradient Descent (SGD)

The stochastic gradient estimates the true gradient on the whole dataset by using only a single data point. The update rule for the SGD algorithm is given by

$$\theta_{t+1} = \theta_t - \eta \nabla(f(x_i; \theta_t)) \tag{2.14}$$

We can show that the stochastic gradient $\nabla(f(x_i; \theta_t)$ is an unbiased estimate of the true gradient $\nabla(f(; \theta_t)$, if we assume that the samples are iid (independent identically distributed) and we choose any point uniformly at random then,

$$E(\nabla(f(x_i; \theta_t))) = \frac{1}{N} \sum_{i=1}^{N} \nabla(f(x_i; \theta_t) \tag{2.15}$$

This shows that the stochastic gradient estimate is unbiased, but the variance of stochastic gradient algorithm is non-zero as shown below ,
Using Taylor Series and assuming that f is an L-smooth function we get

$$\mathbb{E}\, f(x; \theta_{t+1}) \leq \mathbb{E}\, f(x; \theta_t) - \eta\, \mathbb{E}\left(\nabla f(x, \theta)^T (\theta_{t+1} - \theta_t)\right) - \frac{L\eta^2}{2}\, \mathbb{E}\, ||(\theta_{t+1} - \theta_t)||^2 \tag{2.16}$$

$$\leq \mathbb{E}\, f(x; \theta_t) - \eta ||\, \mathbb{E}\, \nabla f(x; \theta_t)||^2 - \frac{L\eta^2}{2}\, \mathbb{E}\, ||t \nabla f(x; \theta_t)||^2 \text{ [Using Eq 2.14]} \tag{2.17}$$

$$\leq \mathbb{E}\, f(x; \theta_t) - \eta ||\, \mathbb{E}\, \nabla f(; \theta_t)||^2 - \frac{L\eta^2}{2}\, \mathbb{E}\left[\nabla f(x; \theta_t)^2\right] \tag{2.18}$$

$$\leq \mathbb{E}\, f(x; \theta_t) - \eta(1 - \frac{L\eta}{2}) ||\, \mathbb{E}\, \nabla f(; \theta_t)||^2 + \frac{L\eta^2}{2} \mathbb{V}(\nabla f(x; \theta)) \tag{2.19}$$

Where , $\mathbb{V}(\nabla f(x; \theta)) = \mathbb{E}\, ||\nabla f(x; \theta) - \mathbb{E}\, \nabla f(x; \theta)||^2$.

From the above Eq 2.19 it can be easily seen that a smaller variance helps in getting to a better optimal solution faster.
In the next chapter we see some popular techniques in literature that have been used for variance reduction.

Chapter 3

Variance Reduction Techniques

Variance Reduction[1] is an important part of many approximation algorithms like SGD@. These techniques help in finding a more precise approximation quantity of interest resulting in better performance models that are being trained using these approximation algorithms. Variance reductions techniques have found usage in a variety of machine learning applications like MCMC algorithms, Policy Gradients, and SGD all of which use variance reduction techniques for a better generalization performance of the model being trained.

This chapter discusses the existing methods of variance reduction in literature. We study all the techniques from the point of view of Monte Carlo Estimation of $E(f(x))$ but all this also readily apply to Stochastic Gradient Algorithms. We give references to those papers that use these techniques for SGD and describe in detail in section**??**.

In this chapter, we, firstly give a brief introduction to Monte Carlo Estimation and then introduce variance reduction techniques such as Stratified Sampling, Control Variates, and Importance Sampling

3.1 Basics

To motivate our discussion for importance sampling we take the problem of finding the mean of a function of a Random Variable. Consider a random variable X having probability mass function or probability density function denoted by $p_X(x)$ which is greater than zero on a set of values $\mathcal{X}$. Then the expected value of a function f(X)is given by,

$$\mathbb{E}(f(X)) = \sum_{x \in \mathcal{X}} f(x) p_X(x) \tag{3.1}$$

if X is discrete and,

$$\mathbb{E}(f(X)) = \int_{x \in \mathcal{X}} f(x) p_X(x) dx \tag{3.2}$$

if X is continuous
If we were to take n samples of $(x_1, x_2, ..., x_n)$ and compute the mean of f(x), we get [2] of $\mathbb{E}(f(X))$

$$\widetilde{f}_n(X) = \frac{1}{n} \int_{x \in \mathcal{X}} f(x) dx \tag{3.3}$$

$\widetilde{f}(X)$ is called the monte carlo estimate of $\mathbb{E}(f(X))$. If $\mathbb{E}(f(X))$ exist then Weak Law of Large Numbers tells us

[1] The major Source of this chapter is the Handbook of Monte Carlo Methods
[2] Here we assume the samples are iid.

that for any arbitarily small ϵ

$$\lim_{n \to \inf} P(|\widetilde{f}(X) - \mathbb{E}(f(X))| \geq \epsilon) = 0. \tag{3.4}$$

This simply states that as n increases our approximation get closer to the exact value.
Every stochastic approximation algorithm contains two important concepts, bias and variance.

- Bias can be defined as the mean of difference between the estimated and the exact value of a Random Variable. More formally for the above example,

$$Bias(\widetilde{f}(X)) = \mathbb{E}(\widetilde{f}(X) - \mathbb{E}(f(X))) \tag{3.5}$$

  An estimator is called unbiased if its bias is equal to zero.

- Variance can be defined as the measure of dispersion between estimates from the exact quantity. For the above example we have

$$Var(\widetilde{f}(X)) = \mathbb{E}[\{\widetilde{f}(X) - \mathbb{E}(f(X)\}^2] \tag{3.6}$$

If we assume the samples are IID. We can show that Monte Carlo Estimates are unbiased.

$$\mathbb{E}(\tilde{f}(X)) = \mathbb{E}\left(\frac{1}{n}\int_{x \in \mathcal{X}} f(x)dx\right) = \frac{1}{n}\int_{x \in \S} \mathbb{E}(f(x))dx = \mathbb{E}(f(X)) \tag{3.7}$$

The variance of Monte Carlo Estimates is given by the following

$$Var(\tilde{f}(X)) = Var\left(\frac{1}{n}\int_{x \in \mathcal{X}} f(x)dx\right) = \frac{Var(f(X))}{n} = \frac{1}{n}\int_{x \in \mathcal{X}} [f(x) - \mathbb{E}(f(X)]^2 p_X(x)dx \tag{3.8}$$

the variance of the estimator is non zero this could lead to catastrophic results for the subsequent models that uses this quantity. To have a better approximation we need to reduce is the variance this is the goal of variance reduction methods. Below we list and explain some popular methods to achieve variance reduction.

- Stratified Sampling
- Control Variates
- Importance Sampling

3.2 Stratified Sampling

Consider estimation of $\theta = E[f(X)] = \int_{x\in\mathcal{X}} f(x)p(x)dx$. For stratified sampling, we first partition the domain of integration $\mathcal{X}$ into m disjoint subsets $\mathcal{X}_i$, i = 1, 2, . . . , m ie $\mathcal{X}_i \cap \mathcal{X}_j = 0\ \forall i \neq j$ and $\cup_{i=1}^{m}\mathcal{X}_i = \mathcal{X}$. Defining

$$\mathbb{E}(\theta_i) = \mathbb{E}(f(X)|X \in \mathcal{X}_i) \tag{3.9}$$

$$= \int_{\mathcal{X}_i} f(x)p(x)dx \tag{3.10}$$

for i=1,2 ..... m. We can write

$$\theta = \int_{\mathcal{X}} f(x)p(x)dx = \sum_{i=1}^{m}\int_{\mathcal{X}_i} f(x)p(x)dx = \sum_{i=1}^{m}\theta_i \tag{3.11}$$

The motivation behind this method is to sample more from regions that have more variability or diverseness, rather than sampling evenly across the whole region S.
We now derive the mean and variance for stratified sampling Define

$$g_i = \int_{\mathcal{X}_i} p(x)dx \tag{3.12}$$

and (3.13)

$$p_i(x) = \frac{1}{g_i}p(x) \tag{3.14}$$

Then $\sum_{i=1}^{m} g_i = 1$ and

$$\int_{\mathcal{X}_i} p_i(x)dx = 1 \tag{3.15}$$

By expressing

$$f_i(x) = \begin{cases} f(x), \text{if} x \in \mathcal{X}_i \\ 0, \text{otherwise} \end{cases} \tag{3.16}$$

we can see,

$$\theta_i = \int_{\mathcal{X}_i} g_i f(x)\frac{p(x)}{g_i}dx \tag{3.17}$$

Using Monte Carlo Estimate for $\mathbb{E}(f(x_i))$ we get

$$\tilde{\theta_i} = \frac{g_i\sum_{j=1}^{N_i} f_i(x_j)}{N_i} \tag{3.18}$$

where $x_1, x_2, ...x_{N_i}$ are random samples from the distribution with density $g_i(x)$ with the variance given by

$$Var(\tilde{\theta_i}) = \frac{g_i^2\sigma_i^2}{N_i} \tag{3.19}$$

where $\sigma_i^2 = (Var)(f(x)|X \in \mathcal{X}_i)$. Thus for the full the estimator

$$\mathbb{E}(\tilde{f(X)}) = \sum_{i=1}^{m}\mathbb{E}(\tilde{f(x_i)}) = \sum_{i=1}^{m}\frac{g_i}{N_i}\sum_{j=1}^{N_i} f_i(x_j) \tag{3.20}$$

with the variance given by

$$Var(\theta) = \sum_{i=1}^{m} \frac{g_i^2 \sigma_i^2}{N_i} \tag{3.21}$$

Variance reduction can now be cast as the following optimization problem

$$\text{argmin}_{N_i} Var(\mathbb{E}(\tilde{f(X)})) \tag{3.22}$$

$$\text{st} \sum_{i=1}^{m} N_i = N \tag{3.23}$$

using the Lagrangian Dual we get

$$\sum_{i=1}^{m} \frac{g_i^2 \sigma_i^2}{N_i} + \lambda(N - \sum_{i=1}^{m} N_i) \tag{3.24}$$

The optimum value comes out to be $N_i = \frac{g_i \sigma_i N}{\sum_{i=1}^{m} g_i \sigma_i}$

The method is simple to understand and implement but it is not always possible to dive the data into mutually exclusive strata moreover estimating a value for σ_i(Variance of strata i) is hard and computationally expensive.

3.3 Control Variate

We again use the problem of estimating θ. Control Variate method find another function h(x) which is similar to f(x) such that $\mathbb{E}(h(x)) = \int h(x)p(x)dx = \tau$ is known, this function $h(x)$ is called the control variate. The equation $\theta = E[f(X)]$ can equivalently be written as

$$\theta = \int [f(x) - h(x)]p(x)dx + \tau = \mathbb{E}[f(x) - h(x)] + \tau \tag{3.25}$$

The Monte Carlo estimate for above equation is

$$\widetilde{\theta} = \frac{1}{N} \sum_{i=1}^{N} [f(x_i) - h(x_i)] + \tau, \tag{3.26}$$

and (3.27)

$$var(\tilde{\theta}) = \frac{var(f_x) + var(h_x) - 2cov(f(x), h(x))}{N} \tag{3.28}$$

The variance of the estimator can be reduced if we choose the above the $h(x)$ such that

$$corr(f(x), h(x)) > \frac{1}{2} \tag{3.29}$$

$corr$ denotes the correlation between $f(x)$ and $h(x)$

Finding such a $h(x)$ is not trivial and sometimes not possible for many problems. We now look at the most popular variance reduction technique in literature Importance Sampling.

3.4 Importance Sampling

Importance sampling attempts to reduce the variance of the Monte Carlo estimate by changing the distribution from which the actual sampling is carried out.

Monte Carlo estimates uses samples generated from the underlying distribution of x (with density p(x)). Suppose that it is possible to find another distribution H(x) with density h(x) with the property that $\frac{f(x)p(x)}{h(x)}$ has a small variance. Then θ can be expressed as

$$\theta = \int f(x)g(x)dx = \int f(x)\frac{p(x)}{h(x)}h(x)dx = \mathop{\mathbb{E}}_{h(x)}[\phi(x)] \tag{3.30}$$

where $\phi(x) = \frac{f(x)p(x)}{h(x)}$ and the expectation is with respect to H . Now the monte carlo estimate for θ becomes

$$\widetilde{\theta} = \frac{1}{N}\sum_{i=1}^{N}\phi(x_i) \tag{3.31}$$

where $x_i = x_1, ..., x_n$ is sampled from H. The variance of this estimator is given by

$$var(\widetilde{\theta}) = \frac{1}{N}\int(\phi(x_i) - \theta)^2 h(x)dx = \frac{1}{N}\int\Big(\frac{f(x)p(x)}{h(x)}\Big)^2 h(x)dx - \frac{\theta^2}{N} \tag{3.32}$$

For variance reduction we need,

$$\int\Big(\frac{f(x)p(x)}{h(x)}\Big)^2 h(x)dx \leq \int f(x)^2 p(x)dx \tag{3.33}$$

Below we show that the optimal distribution for variance reduction is of the form $h(x) \propto f(x)p(x)$.

Claim 3.4.1. It can be easily shown that if $h(x) = \frac{f(x)p(x)}{c}$ then for any other distribution $h\hat{(}x)$. Var(h(x)) $\leq$ Var $(h\hat{(}x))$.

$$\begin{aligned}
Var(\widetilde{\theta})_{h(x)} - \frac{\theta^2}{N} &= \int\Big(\frac{f(x)p(x)}{h(x)}\Big)^2 h(x)dx && (3.34)\\
&= \Big(\int\Big(f(x)p(x)dx\Big)\Big)^2 && (3.35)\\
&= \Big(\int\Big(\frac{f(x)p(x)}{q(x)}q(x)dx\Big)\Big)^2 && (3.36)\\
&\leq \int\Big(\frac{f(x)^2p(x)^2}{q(x)^2}q(x)dx)\Big) && (3.37)\\
&\leq \int\Big(\frac{f(x)^2p(x)^2dx}{q(x)}\Big) && (3.38)\\
&\leq Var(\widetilde{\theta})_{h\hat{(}x)} - \frac{\theta^2}{N} && (3.39)
\end{aligned}$$

We get 3.35 setting h(x) = $\frac{f(x)p(x)}{c}$ and using c = $\int(f(x)p(x)dx)$
3.37 using Cauchy-Schwartz Inequality.

From the above discussion, we have shown that the importance sampling methods are easiest to optimize for variance reduction and suffer from no faults like how to distribute the data into strata or how to define control variate. These are the major reason we prefer to use importance sampling in our work instead of the other two methods described above.

Chapter 4

Submodularity and Diversity

Submodular functions are a well know and widely studied class of functions defined on sets. Diversity is a very important topic in the context of Machine Learning. Finding a set of diverse items is a hard and important task and finds application in the field of Document/ Video Summarization, Facility location problems etc. Submodular functions have a natural diminishing return property which makes them suitable for modeling diversity. Submodular function have used in a wide variety of problems like Graph Cut, Facility location, Document Summarization etc.
In this chapter, we first define submodular functions. Then discuss how submodular functions have been used in variance reductions tasks in existing literature. Optimization of submodular functions is discussed later for the case of large datasets. Finally, the relationship between submodular functions and diversity is explained through an example.

4.1 Submodularity

Submodularity is a property of a set function that assigns a value to each of the the subsets of the set. ie $f : 2^V \rightarrow \mathbb{R}$. This can be thought of as assigning a utility value to each subset.
Definition 4.1.1. Submodular: For a set function $f : 2^V \rightarrow \mathbb{R}$ is submodular if for every $A \subseteq B \subseteq V$ and e $\in$ V \ B it holds that

$$f(A \cup e) - f(A) \geq f(B \cup e) - f(B) \tag{4.1}$$

Equivalently , a function $f : 2^V \rightarrow \mathbb{R}$ is submodular if for every A, B $\subseteq$ V,

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B) \tag{4.2}$$

The first definition can intuitively be explained as , the increase in uitility of subsets A and B after adding an element e(not present in B) is greater for the set with smaller size. This is called the diminishing return property of submodular functions.

Definition 4.1.2. Monotonicity : A set function $f : 2^V \rightarrow \mathbb{R}$ is monotone if for every $A \subseteq B \subseteq V$, f(A) $\leq$ f(B)

Definition 4.1.3. Discrete Derivative For a set function $f : 2^V \rightarrow \mathbb{R}$, $S \subseteq V$ and $e \in V$, let $\Delta_f(e|S) := f(S \cup \{e\}) - f(S)$ be the discrete derivative of f at S with respect to e.

Discrete derivative is also commonly know as marginal gain of f on S with e. Next we go through some important examples of submodular functions used in practice.

4.1.1 Examples

- Facility Location: Suppose we want to select some locations out of a set V = 1, . . . , n, to open up facilities in order to serve a collection of m customers. If we open up a facility at location j, then it provides service of value $M_{i,j}$ to customer i, where $M \in R_{m \times n}$. If each customer chooses the facility with highest value, the total value provided to all customers is modeled by the set function

$$f(S) = \sum_{i=1}^{m} max_{j \in S} M_{i,j} \tag{4.3}$$

Claim 4.1.1. $f(S)$ is a submodular function.

Proof. We need to show that $\triangle_f(e|A) \leq \triangle_f(e|b)$ where $A \subseteq B \subseteq V$ for the funtion $f(S)$ to be submodular. Consideer the following cases
Case I If $\triangle_f(e|B) = 0 then \triangle f(e|A) = 0$. Hence we have the $\triangle_f(e|B) = f(e|A)$
Case II If $\triangle_f(e|A) \leq 0$ and $\triangle_f(e|B) = 0$ then we have $\triangle_f(e|A) \leq \triangle_f(e|B)$
Case III If $\triangle_f(e|B) \leq 0$ then $\triangle_f(e|A) = \triangle_f(e|B)$
In all the above cases we have $\triangle_f(e|A) \leq \triangle_f(e|B)$. Hence $f(S)$ is a submodular function. □

- Entropy: Given a joint Probability Distribution P(X) over a discrete-valued random vector $X = X_1, X_2, ...X_n$, the function f(S) = H(X_s) is monotone submodular, where

$$H(X_s) = -\sum_{x_S} P(x_S) log_2 P(x_S) \tag{4.4}$$

where we use the notational convention that X_S is the random vector consisting of the coordinates of X indexed by S, and likewise x_S is the vector consisting of the coordinates of an assignment x indexed by S.

Claim 4.1.2. To show that $H : 2^S \rightarrow [0, \inf)$ is submodular consider:

Proof.

$$H(X, s) - H(X) \geq H(Y, z) - H(Y) = H(z|X) \geq H(z|Y) \tag{4.5}$$

where $H(z|Y) = H(z|X \cup (Y \setminus X)) \geq H(z|X)$, since conditioning cannot increase entropy. □

4.1.2 Optimization of Submodular Functions

The application of submodular functions in so many areas makes it naturally to study submodular optimization. In our work, we are interested in the maximization of submodular functions. Formally,

$$\max_{S \subseteq V} f(S) \text{ subject : to constraints on S} \tag{4.6}$$

the most basic constraint is called the cardinality constraint which is $|S| \leq$ k for some k. Even in the simple example of trying to find the best k places to eat for maximum fun, this problem is NP-Hard. Though this problem can be solved very efficiently using the Greedy algorithm from [1] if $f(S)$ is submodular.

4.1.3 Greedy Algorithm

Algorithm 1 (Greedy) Algorithm by [1]

Input: Set V, Set function $f : 2^V \rightarrow \mathbb{R}$, Size of subset k.
Output: Subset $S_k \subseteq V$ of size k

1: $S_0 = \phi$
2: **for** $i = 0$ to k **do**
3: $\quad v_* \leftarrow argmax_{v\ inV \setminus S_{i1}} \Delta(v|S_{i-1})$
4: $\quad S_i \leftarrow S_{i-1} \cup v_*$
5: **end for**

Theorem 1. *Fix a nonnegative monotone submodular function $f : 2^V \rightarrow \mathbb{R}^+$ and let $S_{i:i\geq 0}$ be the greedily selected sets defined in 1 Then for all positive integers k and l,*

$$f(S_l) \leq (1 - e^{-l/k}) max_{S:|S|\leq k} f(s) \tag{4.7}$$

In particular, for l=k, $f(S_k) \geq (1 - 1/e) max_{|S|\leq k} f(S)$

Proof. Let $S^* \in$ argmax$\{f(S) : |S| \leq k\}$ be an optimal set of size k (due to monotonicity of f we can assume w.l.o.g. it is of size exactly k), and order the elements of S arbitrarily as $v_1^*, ..., v_k$. Then we have the following sequence of inequalities for all i ¡ l,

$$\begin{aligned} f(S^*) &\leq f(S^* \cup S_i) && (4.8)\\ &= f(S_i) + \sum_{i=1}^{k} \Delta(v_j^* | S_i \cup \{v_1^*, ..., v_k\}) && (4.9)\\ &\leq f(S_i) + \sum_{v \in S^*} \Delta(v|S_i) && (4.10)\\ &\leq f(S_i) + \sum_{v \in S^*} (f(S_{i+1} - f(S_i)) && (4.11)\\ &\leq f(S_i) + k(f(S_{i+1} - f(S_i)) && (4.12) \end{aligned}$$

The first equation 4.8 follows from Monotonicity of f. The next inequality 4.9 is from the telescoping sum. 4.10 uses Sub modularity . 4.11 holds because S_{i+1} is built greedily from S_i in order to maximize the marginal benefit $\Delta(v|S_i)$, and 4.12 merely reflects the fact that $|S| \leq k$. Now define $\delta_i := f(S^*) - f(S_i)$ we get

$$\delta_{i+1} \leq (1 - \frac{1}{k})\delta_i \tag{4.13}$$

Hence, $\delta_l \leq (1 - \frac{1}{k})^l \delta_0$. Next, $\delta_0 = f(S^*) - f(\phi) \leq f(S^*)$ since f is nonnegative by assumption, using $1 - x \leq e^{-x}$ for all $x \in \mathbb{R}$ we have

$$\delta_l \leq \left(1 - \frac{1}{k}\right)^l \delta_0 \leq e^{-l/k} f(S^*) \tag{4.14}$$

Rearranging we get $f(S_l) \geq (1 - e^{-l/k}) f(S^*)$ □

Though the Greedy Algorithm1 is pretty simple to implement it suffers from a major flaw that it does not scale well for large datasets. In particular, the run time for the Greedy Algorithm is $0(n * k)$ function evaluations which are prohibitively expensive even for moderate size n(number of data points).
To decrease the number of function evaluation the LAZY-GREEDY algorithm by [5] is proposed. The LAZY-GREEDY algorithm cleverly makes use of submodularity to reduce the running time by keeping the elements

in a sorted decreasing order based upon the marginal gains, at each iteration i, it evaluates the element on top of the list, say e, and updates its upper bound, $\rho(e) \leftarrow \Delta(e|S_{i-1})$. If after the update ρ(e) $\geq$ ρ(e') for all e' $\neq$ e, submodularity guarantees that e is the element with the largest marginal gain. The worst case bounds for LAZY-GREEDY algorithm are similar to the Greedy algorithm but in practice, it boosts the speed up by order of magnitudes. Still for a large k(cardinality constraint) and repeated selection of points the LAZY-GREEDY algorithm is still slow and impractical to be applied.

4.1.4 STOCHASTIC-GREEDY

STOCHASTIC-GREEDY [2] algorithm tries to improve upon the LAZY-GREEDY algorithm using a very simple insight of sub sampling the dataset before using the LAZY GREEDY or GREEDY algorithm. It is in a sense equivalent to stochastic gradient descent which improves the running time of gradient descent for convex optimization using a sub sampling step.

Algorithm 2 STOCHASTIC GREEDY Algorithm by [2]

Input: set V, set function $f : 2^V \rightarrow \mathbb{R}$, Size of subset k.
Output: subset $S_k \subseteq V$ of size k
1: $S_0 = \phi$
2: **for** $i = 0$ to k **do**
3: R $\leftarrow$ a random subset of size $\frac{n}{k \log(\frac{1}{\epsilon})}$, obtained by sampling from $V \setminus S_{i-1}$.
4: $v_* \leftarrow$ argmax$\Delta_{v \in R}(v|S_{i-1})$
5: $S_i \leftarrow S_{i-1} \cup v_*$
6: **end for**

STOCHASTIC-GREEDY algorithm is the first linear-time algorithm for maximizing a non-negative monotone submodular function subject to a cardinality constraint k. STOCHASTIC-GREEDY achieves a (1 - 1/e - ϵ) approximation guarantee to the optimum solution with running time O(n log(1/ϵ)). Achieving a approximation ratio of (1 - 1/e -ϵ) requires $R \geq n/klog(1/\epsilon)$ for further details the readers is implored to read [2]
There have been other recent works [6] that show that rather than taking the max element we could also select the elements according to the probability conditioned on the marginal gain of the elements.

4.2 Variance Reduction and Submodularity

Variance reduction and Submodularity look pretty unrelated at the start but as shown below for certain problems we can cast the variance reduction problem as a submodular maximization problem.
Let us consider the case where we have a finite number of locations V and we can place a sensor at each location to observe an associated quantity with these locations given by X_v, the joint probability of the observations is given by $P(X_v)$. We want to reduce the cost of placing the sensors choosing only a subset of locations A $\subseteq$ V to observe, the locations A have to selected such that the average predictive variance,

$$V(A) = \frac{1}{n} \sum_i \sigma^2_{i|A} \tag{4.15}$$

is minimized. $\sigma^2_{i|A}$ denotes the predictive variance at the location i.

$$\sigma^2_{i|A} = \int P(x_A) \, \mathbb{E} \left[(X_i - \mathbb{E}[X_i|x_A])^2 | x_A \right] dx_A \tag{4.16}$$

We need to solve the following optimization problem for getting the minimum average predictive variance.

$$A^* = argmin_{|A| \leq k} V(A) \tag{4.17}$$

As can be seen clearly that 4.17 problem has a combinatorial search space which makes the problem hard to solve. It can be show that the problem is NP-hard in general [7]. Fortunately [7] show that the variance reduction

$$F_s(A) = \sigma_s^2 - \sigma_s|A^2 \tag{4.18}$$

at any particular location s, satisfies the following diminishing returns behavior: Adding a new observation reduces the variance at s more, if we have made few observations so far, and less, if we have already made many observations. Since now we can say that each of $F_S(A)$ is submodular, the average variance reduction is also submodular

$$F(A) = V(\phi) - V(A) = \frac{1}{n} \sum_s F_s(A) \tag{4.19}$$

It can also be shown that the average variance reduction is monotonic ie for all A $\subseteq$ B $\subseteq$ V it holds that F(A) $\leq$ F(B), and normalized F(ϕ) = 0.
Hence the problem of minimizing the average variance is an instance of the problem

$$max_{A \subset V} F(A), \text{subject to} |A| \leq K, \tag{4.20}$$

since F is monotonic, normalized, submodular we can use the above mentioned optimization methods to solve the problem easily.
A particular case of the above example is seen in the case of GeoStatistics in Kriging variance reduction minimisation problem and also in the Gaussian Process predictive variance reduction problem.

4.3 Diversity and Submodularity

The diversity of a set can be defined as the measure of the difference between the elements of the dataset. Diversity is a very important concept in the area of Machine learning particularly in the area of Result Diversification. A recommender's output should be diverse enough to capture the preferences of the user but not too diverse so that user preferences are orthogonal to the recommended items. Diversity is also an important requirement in the task of summarization of videos and text data.
The problem of adding a subset V (of the size of k) to an already sampled set S^* (of size m), such that we get the maximum diversity can be cast as the following optimization problem

$$\text{argmax}_{V \in S \setminus S*, |V| \leq k} f(V \cup S^*) - f(S^*) \tag{4.21}$$

Where f is a set function from $f : 2^S \rightarrow \mathbb{R}$ that denotes the value of diversity of the set. It can be shown that the problem is NP-hard in general and candidate solution space is very large. But if f is submodular the problem can be solved as a Submodular Optimization problem which as seen earlier is a much easier problem to solve. From this, we see that Submodular function are good candidates for modeling diversity in any problem this can be attributed to their diminishing return property.

We now show an example[8] of how a submodular function can model diversity. Let f be a set function, $f : 2^S \rightarrow$

$\mathbb{R}$ of the following form

$$f(S) = \sum_{i \in S} u_i + \sum_{d=1}^{L} (\max_{i \in S} w_{i,d} - \sum_{i \in S} w_{i,d}) \tag{4.22}$$

Here u_i indicates a modular function which denotes the quality of the element i. this term only captures frequencies of the individual items, ignoring any interdependencies between the items. For countering this, the paper assigns to each item i an L-dimensional vector $w_i \in R^L_{\geq 0}$. The intuition behind this is that each of these L dimensions will capture some concept and can interpret $w_{i,d}$ as a quantification of how relevant that item i is for that specific concept d. To quantify the diversity of some items S $\subseteq$ V with respect to dimension d, the paper propose the term max$w_{i,d} - \sum_{i \in S} w_{i,d}$. This term is a non positive penalty, evaluating to 0 iff S contains at most one item i with positive value $w_{i,d} \geq 0$. Up until now, we have shown a function that can model diversity, we now proved that the obtained function is submodular.

Claim 4.3.1. f(s) is a submodular function

Proof. We take $A \subseteq B \subseteq S$, take an element v $\in S \setminus B$, then we assume

$$\Delta_f(v|A) \geq \Delta_f(v|B) \tag{4.23}$$

$$f(v \cup A) - f(A) \geq f(v \cup B) - f(B) \tag{4.24}$$

Expanding and Simplying we get (4.25)

$$\sum_{d=1}^{L} \big[(\max_{i \in A \cup v} w_{i,d}) - (\max_{i \in A} w_{i,d}) \geq \sum_{d=1}^{L} \big[(\max_{i \in B \cup v} w_{i,d} - \max_{i \in B} w_{i,d}) \tag{4.26}$$

We now show that this is always true when A $\subseteq$ B since,

- Case 1: Assume that for some d $w_{v,d} \leq w_{i \in A,d}$ then LHS=RHS.
- Case2: Assume that for some d $w_{v,d} \geq w_{i \in A,d}$ and $w_{v,d} \leq w_{i \in B,d}$. Using the fact that $w_{i,d} \geq 0 \forall i$. LHS $\geq$ RHS
- Case3: Assume that for some d $w_{v,d} \geq w_{i \in B,d}$. Using the fact that $A \subseteq B$ means $w_{v,d} \geq w_{i \in A,d}$ But for any d $w_{i \in A,d} \leq w_{i \in B,d}$. We see that LHS $\geq$ RHS Hence the above function is indeed submodular.

□

4.3.1 Probabilistic Submodular Functions

Definition 4.3.1. Probabilistic Submodular Functions: are probability distribution over $A \subseteq V$ of the form P(A) = $\frac{\pm exp(F(A)}{\mathcal{Z}}$. Where F(A) is a submodular function on V.

The normalizing quantity $\mathcal{Z} = P_{S \subseteq V}(exp \pm F(S))$ is called the partition function, and $\log \mathcal{Z}$ is also known as free energy in the statistical physics literature. A probabilistic way of modeling diversity is through use of Ising Models and Detrimental Point Processes as show in 1.10

We can show that both Ising models and DPP are a special case of Probabilistic Submodular functions. In particular, DPP is a distribution over a set A of the form P(A) = $\frac{exp(F(A))}{\mathcal{Z}}$, where F(A) = $\log |L_Y|$. Here, L $\in \mathbb{R}^{N \times N}$ is a positive semi-definite matrix, L'_Y is the square submatrix indexed by Y' and || denotes the determinant. Because L'_y is positive semi-definite, F(A) is submodular, and hence DPPs are log-submodular. A major problem is the complexity of sampling and learning these PSF for the case of large datasets.

From the above, we have shown how existing methods make use of submodular functions to model diversity and how some existing methods can be categorized as special cases of submodular functions. We also show that diversity is an important metric to consider in many problems and that submodular functions model diversity well. A major problem with using submodular functions is the computation cost, in our work we come up with a novel submodular function which takes a minimum amount to compute and also introduce methods to scale submodular optimization to large datasets.

Part III

Related Work

Chapter 5

Optimization Based Methods

5.1 Optimization Based Methods

In this chapter, we give a brief survey of the works that use optimization techniques to solve the variance reduction problem in SGD.

As stated earlier one of the major disadvantages of using mini batch SGD is a random sampling of data points can lead to a large variance between the true gradient and the mini-batch gradient leading to slower convergence of SGD. Many techniques have been introduced to tackle this issue.
Variance Reduction techniques can broadly be classified into two categories namely non-sampling based and sampling based methods. Remember in mini-batch SGD we have

$$\theta_{t+1} = \theta_t - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla g(x_i; \theta_t) \tag{5.1}$$

Where B denotes the size of the minibatch. If we choose B=1, the variance of the gradient can be shown to be

$$\begin{aligned} Var(\nabla f(x_i; \theta_t)) &= \mathbb{E}[(\nabla f(x_i; \theta_t) - \nabla f(; \theta_t)^2] \\ \nabla f(; \theta_t) &: \text{Full batch Gradient} \\ \nabla f(x_i; \theta_t) &: \text{mini batch Gradient} \end{aligned} \tag{5.2}$$

For reduction in variance we can either change how we create mini-batches(importance sampling), change learning rate(η), change batch size(b) or change the way we updates the parameters. In this chapter we elaborate the non-sampling based/ optimization based techniques.

5.2 Non-Sampling Based

These techniques sample the data points using a uniform distribution. These methods can further be specialized into the following categories
Averaging Techniques: These techniques reduce variance using averaging of parameter [9] or by averaging of the gradients [10]. Averaging based methods get smooth approximation of gradients or parameters, leading to a reduction in variance.

The simplest example of this method is the minibatch gradient decent algorithm, it can be shown that it reduces variance as follows, Using Eq 2.14

$$\theta_{t+1} = \theta_t - \eta\nabla(f(x_i;\theta_t)) \tag{5.3}$$

minibatch SGD changes the update rule to compute the gradient over a minibatch of data points $B = \{x_j \in \mathcal{D}| = 1, 2, \ldots, b\}$ as follows

$$\theta_{t+1} = \theta_t - \frac{\eta}{b}\sum_{i=1}^{B}\nabla(f(x_i;\theta_t)) \tag{5.4}$$

Where b denotes the size of a uniformly random sampled batch from full dataset $\mathcal{D}$.
Now using Eq 2.19 for minibatch SGD we get ,

$$\mathbb{E}\, f(x;\theta_{t+1}) \leq f(x;\theta_t) - \eta(1 - \frac{L\eta}{2})||\,\mathbb{E}\,\nabla f(;\theta_t)||^2 + \frac{L\eta^2}{2b}\mathbb{V}(f(x;\theta) \tag{5.5}$$

Here we see that the variance reduces by factor b. Although the computation cost has increased by a similar factor of b, this cost can be reduced using a parallel version of SGD but in general, this cost is innocuous. In general minibatch gradient descent leads to a better generalization error in lesser time.

Momentum based techniques use momentum from past gradients to choose the descent direction[11] [12]. The descent direction is based on the averaged descent direction, which is updated periodically. The simplest momentum based technique uses the following update rule

$$\theta_{t+1} = \gamma\theta_t - \eta\nabla(f(x);\theta_t)) \tag{5.6}$$

where $\gamma \leq 1$ parameter. Momentum-based methods reduce the oscillation in the update of the parameters. It does so by adding a damping factor γ, the factor is useful when the direction of the gradient changes drastically the γ factor dampens the update thereby reducing the update in the wrong direction. The gamma factor favors a smooth change of direction in gradients. As a result, we gain faster convergence and reduced oscillation. When momentum method is combined with minibatch SGD Eq 2.19 changes to

$$\mathbb{E}\, f(\mathbf{x};\theta_{t+1}) \leq f(\mathbf{x};\theta_t) - \eta(1 - \frac{L\eta}{2})||\,\mathbb{E}\,\nabla f(;\theta_t)||^2 + \frac{\gamma^2\eta^2}{2b}\mathbb{V}(f(\mathbf{x};\theta) \tag{5.7}$$

Here we can see that addding the momentum term has reduced the variance further by a factor of γ^2.

Adaptive Learning Rates and Batch Size: Setting adaptive learning rates based on local curvature[13] or based on gradient information [14] has been proved to reduce variance in SGD. We show a particular case of AdaGrad Method, the update rule changes to

$$\theta_{t+1} = \gamma\theta_t - \frac{\eta}{\epsilon + G_t}\nabla(f(\mathbf{x};\theta_t)) \tag{5.8}$$

where $G_t^{D\times D}$, is a diagonal matrix, where each diagonal element i, is the sum of the squares of the gradients w.r.t. θ_t^i (i th dimension of θ) up to time step t , while ϵ is a small value added to avoid division by zero. Adagrad adapts the learning rate based on the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data. The interested reader is pointed to [13] for a more theoretical justification of variance reduction.
Other papers have explored optimizing the batching size for variance reduction. These methods try to find an adaptive schedule for the batch size that would reduce the variance. [15] adapt batch size based on the variance

of the gradient estimates, [16] adaptively increases the batch size during the training process starting with a small batch size and doubling the batch sizes at specified intervals.

Chapter 6

Importance Sampling Based Methods

Importance sampling based methods reduce the variance by finding an optimal distribution to sample the data points. As discussed earlier in case of importance sampling3.30 instead to sampling from the distribution p(x) we try to find another surrogate distribution h(x) to sample the data points from. The tricky part is finding a distribution h(x) which is easy to sample from as well as it reduces the variance of the approximated function. Importance sampling methods have widely used in literature, below we discuss some of these which are our most relevant to our work.

6.1 Online Learning To Sample [3]

This is one of the first works to successfully show the usage of importance sampling methods for the case of Deep Learning models. The main ideas of the paper can be described as follows, The variance of importance sampling is given by Eq 3.32

$$var(\widetilde{\theta}) = \frac{1}{N}\int \Big(\frac{f(x)p(x)}{h(x)}\Big)^2 h(x)dx - \frac{\theta^2}{N} \tag{6.1}$$

Suppose that distribution comes from a family of distributions H(x; τ), we can then optimize for the value of τ to find the best distribution that leads to maximum reductions in variance ,

$$var_\tau(\widetilde{\theta}) = \frac{1}{N}\int \Big(\frac{f(x)p(x)}{h(x;\tau)}\Big)^2 h(x;\tau)dx - \frac{\theta^2}{N} \tag{6.2}$$

if $f(x) \in H(x;\tau)$ then, there exist a τ^* such that $h(x;\tau^*) \propto f(x)$. In general this is not true and also finding a closed form solution for the min $var_\tau(\tilde{\theta})$ is not possible. The paper proposes for the use of an approximate learning method to solve the above problem as follows. Taking the gradient wrt τ we get,

$$\nabla var_\tau(\tilde{\theta}) = E_{h(x;\tau)}\Big(\frac{f(x)p(x)}{h(x;\tau)}\Big)^2 \tag{6.3}$$

$$= E_{h(x;\tau)}\Big(\frac{f(x)^2\nabla_\tau h(x;\tau)}{h(x;\tau)^3}\Big) \tag{6.4}$$

$$= E_{h(x;\tau)}\Big(\Big(\frac{f(x)}{h(x;\tau)}\Big)^2 \nabla_\tau \log h(x;\tau)\Big) \tag{6.5}$$

$$\tag{6.6}$$

Now using SGD we can update a value for τ as,

$$\tau_{t+1} = \tau_t - \eta\left(\left(\frac{f(x_t)}{h(x_t;\tau)}\right)^2 \nabla_\tau \log h(x_t;\tau)\right) \tag{6.7}$$

here $x_t \sim h(x_t;\tau)$.

The paper is very innovative using SGD for solving the variance reduction problem in SGD. The paper is able to show decent results on many deep learning tasks like Image in painting, matrix factorization etc.
The paper suffers from a major drawback of solving another optimization problem which requires both time and suffers from high variance, making the usability of the method limited.

6.2 Importance Sampling using Extra Information

Another important recent work which has gained popularity is Adaptive Sampling for SGD by Exploiting Side Information[17]. This work finds the distribution q(x) using additional information from the problem which it calls as side information. This side information can be anything like labels etc, is used to create bins and assign each data point to a its respective bin based on upon the side information of the data points. The paper proposes the following update rule

$$\theta_{t+1} = \theta_t - \frac{1}{N}\frac{1}{P(i)}\nabla f(i;\theta_t) \tag{6.8}$$

Where $P(i) = \frac{p_k}{C_k}$, this sampling scheme can be thought of as a two step process of the uniform sampling scheme, first sample any bin uniformly C_k at random and then sample from the bin according to the distribution p_k which is the probability of selecting a data point inside bin k. If p_k is equal to C_k the above reduces to uniform sampling. Instead of keeping p_k static the paper proposes a way of finding the optimal distibribution. The optimal distribution p_k is found using the solution of the variance reduction problem for SGD@. The optimal distribution p_k is expressed as follows,

$$p_k \propto \frac{|C_k|}{N}\sqrt{\frac{1}{|C|_k}\sum_{i\in C_k}||\nabla f(i;\theta_t)||^2} \tag{6.9}$$

Intuitively, we select data points from bins that have a larger gradient.
This is a simple idea achieves great results on tasks of Logistic Regression but the method does not scale well to the larger datasets because of the need for calculating the gradient for finding p_k. Also it is not always possible to define side information for each and every dataset and problem.

6.3 Diversity based sampling

Up until now, we have only seen examples of sampling methods that consider the importance of samples individually. As stated earlier it is also important to consider the interactions of the samples in the minibatches with each other. The minibatches so constructed should be representative of the full dataset. In this section, we elaborate on how existing methods have tried to model this representativeness for usage in sampling methods for variance reduction.

6.3.1 Mini-Batch Diversification

One way of modeling representativeness and interaction of the samples is by using the diversity of the minibatch. In [18] argue that creation minibatches that are diverse enough to be representative of the full dataset result in

a variance reduction. The diversity of minibatch can be defined as the dissimilarity between data points of the minibatch in some metric space. Diversity serves an important purpose when sampling from the highly imbalanced datasets since it samples from less represented classes.

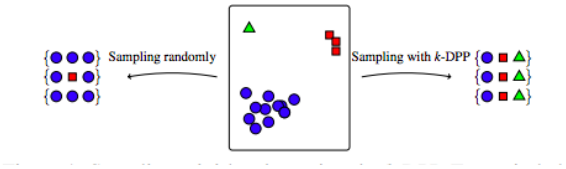


Figure 6.1: Example of DPP sampling on Imbalanced Dataset

As stated earlier Eq 4.21 diversity maximization is a NP-Hard Problem. The authors suggest using Determinantal Point Process(DPP) for solving the diversity maximization problem.
DPP sampling firstly creates a similarity kernel matrix $\mathbf{L}$, where $\mathbf{L}_{ij}$ denotes the similarity between data points i and j. The probability of sub sampling any subset Y of the set {1,...,N}, is proportional to the determinant of the sub-matrix $\mathbf{L_Y}$,

$$P(Y) = \frac{\det(\mathbf{L}_Y)}{\det(\mathbf{L}+I)} \propto \det(\mathbf{L}_Y) \tag{6.10}$$

a DPP based sampling approch promotes diversity as explained in section**??**. The paper used a different variant of DPP sampling called the k-DPP sampling which is DPP sampling conditioned on given size k .

$$P_L^K(Y) = \frac{\det(\mathbf{L}_Y)}{\sum_{|Y'|=k} \det(\mathbf{L}'_Y)} \tag{6.11}$$

DPP sampling can be considered particular case of Probabilistic Submodular Function(PSF). Potentially other PSF can also be used to create diverse minibatches eg Repulsive Point Processes(RPP) eg [19]. A major issue with PSF based sampling methods is the computational time taken for sampling is approximated O(N^3) ie cubic in the number of data points, though faster sampling methods have been proposed they are still pretty slow, this makes these methods infeasible for use in deep learning applications.

6.3.2 Active Bias Sampling

In active bias sampling techniques, we bias our probability distribution based on a particular criterion. The criteria can be anything from magnitude of loss [20], gradient [21], Lipschitz constant [22] and variance of the samples [23].

6.3.3 Variance Reduction in SGD using Importance Sampling

Until now we have not formally shown how a importance sampling methods can reduce variance. We now give a proof sketch to show importance sampling methods help in reducing variance SGD. We can rewrite the SGD update rule defined in Eq 2.14 for importance sampling as follows ,

$$\theta_{t+1} = \theta_t - \eta w_{I_t} f(x_{I_t}; \theta_t) \tag{6.12}$$

Where x_{I_t} is sampled from a probability distribution P at time step t ie $P(I_t = i) = p_i^t$, w_{I_t} denotes the reweighing of the sample x_{I_t}, if $P = \frac{1}{N}$ and $w_{I_t} = 1$ the equation reduces to the standard SGD formulation.

If we define the convergence speed S of SGD as the reduction of distance between the parameter θ and the optimal parameters θ^* in two consecutive iteration t and t+1.

$$S = -\mathbb{E}_p[||\theta_{t+1} - \theta^*||^2 - ||\theta_t - \theta^*||^2] \tag{6.13}$$

Also if we take $w_{I_t} = \frac{1}{N p_{I_t}}$, taking the expectation of the gradient we get,

$$E(w_{I_t} \nabla f(x_{I_t}; \theta_t)) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_t} f(x_{i_t}; \theta_t) \tag{6.14}$$

Define $G_{I_t} = w_{I+t} \nabla f(x_{I_t}; \theta_t)$ using Eq 6.13

$$\begin{aligned}
S &= -\mathbb{E}_P[||\theta_{t+1} - \theta^*||^2 - ||\theta_t - \theta^*||^2] && (6.15)\\
&= -\mathbb{E}_P[\theta_{t+1}^T \theta_{t+1} - 2\theta_{t+1}\theta^* - \theta_t^T \theta_t + 2\theta_t^T \theta^* && (6.16)\\
&\text{Using the update rule and } G_{I_t}, && (6.17)\\
&= -\mathbb{E}_P[-2\eta(\theta_t - \theta^*) G_{I_t} + \eta^2 G_{I_t}^T G_{I_t} && (6.18)\\
&= 2\eta\theta_t - \theta^* \mathbb{E}_P G_{I_t} - \eta^2 E(G_{I_t})^T E(G_{I_t}) - \eta^2 Tr(V_{p_t}[G_{I_t}]) && (6.19)
\end{aligned}$$

The proof is taken from [21]. We can see that we can increase the convergence speed / reduce the variance of SGD if we find a optimal distribution that minimizes $Tr(V_{p_t}[G_{I_t}])$. [22] in their seminal work show that the optimal distribution for the above solution is

$$p_t = \frac{G_{I_t}}{\sum_{I=1}^{N} G_{I_t}} \tag{6.20}$$

It can be seen that the above distribution is computation very inefficient as it requires full gradient over the dataset for each time step t. A work around is approximate the G_{I_t} by another quantity which is easier to compute as follows,

$$\begin{aligned}
\text{min} Tr(V_{p_t}[G_{I_t}]) \iff \text{min} Tr(V_{p_t}[H_I]) && (6.21)\\
H_I \geq G_{I_t} \forall t && (6.22)
\end{aligned}$$

Then the optimal distribution is given by

$$p_t = \frac{H_I}{\sum_{I=1}^{N} H_I} \tag{6.23}$$

One proposed H is to use the Lipschitz constant of the Loss Functions f, since $G_{I_t} \leq L_I \forall t$. In practice even using loss based sampling [24] and [20] performs really well. They test their implementation on many deep learning tasks for image classification and show good results.

All sampling-based methods suffer from slow speeds as they require the calculation of the probability distribution to sample the data points from. Also always choosing the data points using the most optimal distribution can lead to a decrease in performance in practice [21]. This tradeoff between variance reduction and finding a probability distribution can be overcome if we use the epsilon-greedy strategy. Choosing data points using a random distribution in ϵ number of epochs and using importance sampling to select data points in $1 - \epsilon$ number of epochs. They suggest using the following update rule for epsilon.

$$\epsilon = \epsilon * a + (1 - a) * \left(1 - \frac{1}{\sum_{i=1}^{N} g_i^2} \left|\left| g - \frac{1}{|U|} \right|\right|^2\right)^{\frac{-1}{2}} \tag{6.24}$$

where $a < 1$ denotes the annealing term, g denotes the score vector for each data point assigned by the importance

sampling distribution and U denotes the uniform sampling probability for the data point. They show that this really improves the performance of importance sampling schemes on large datasets that prevalent in Deep Learning. We also use this strategy in our experiments.

Part IV

Methodology

Chapter 7

Submodular Importance Sampling

We now present our methodology for submodular importance sampling of mini-batches. Firstly a close relationship between diversity and submodularity is established, we then present our methodology for submodular importance sampling, a novel submodular scoring function is introduced that scales well for the needs of modern deep learning systems, and finally it is shown how our submodular importance sampling scheme leads to variance reduction in SGD.

As mentioned earlier, SGD samples data points randomly which can lead to mini-batches which are biased and present an inexact representation of the dataset. This consequently, leads to slow convergence. Diversity-based sampling methods attempt to sample data points such that the resulting mini-batch is a better representative of the full dataset. A simple example of diversity-based sampling is Stratified Sampling ([25]) where data points are selected from strata, proportional to size/population of the strata, thereby leading to much more balanced minibatches. Submodular functions have been used earlier to model diversity in subset selection problem ([8]), recommendation systems [26] etc. We exploit this submodular nature of diversity measures for the creation of diversified minibatches.

Submodular functions are characterized by the diminishing return property, i.e.,

$$F(A \cup i) - F(A) \geq F(B \cup i) - F(B) \tag{7.1}$$

for all $A \subseteq B \subseteq V \setminus i$ and $i \in V$. Intuitively, submodularity suggests that the marginal gain of adding an item i in the smaller set A is larger than the marginal gain of adding that element in the larger set B. Submodular Functions are natural candidates for capturing representativeness(covering) and diversity in a dataset. Formally, choosing a diverse subset from a given set can be modeled as an optimization problem as follows. Let $V = \{1, \cdots, k\}$ be a set of k items. Let $f : 2^V \rightarrow \mathbb{R}^+$ be a diversity function, which maps any subset of E to a non-negative real number. Then the problem of selection of maximum diverse set[26] turns into

$$A_k = \arg\max_{A \in \Theta} \sum_{t=1}^{K} \Delta_{A_{t-1}}(a_t) \tag{7.2}$$

where $A_k = \{a_1, \cdots, a_k\}$ is a subset of items from E, Θ is the power set of E, $A_t = \{a_1, \cdots, a_t\}$ is a set of the first t items in A, and $\Delta_{A_{t-1}}(a_t) = F(A_t) - F(A_{t-1})$ is the gain in diversity after a_t is added to A_{t-1}also known as the marginal gain of a_t. For any general F, this problem is NP-hard but if F is submodular, the maximization problem (diverse subset selection) can be solved using a greedy approach due to [1].

Motivated by this, we cast the problem of selection of k data point for minibatch diversification as a submod-

ular maximization with cardinality constrained. We now introduce our algorithm called Greedy Submodular-SGD(GreedySSGD).

Algorithm 3 GreedySSGD Algorithm

Input: Training set X, Model M, minibatch size k, $|X|$ = n
Output: Selected batch R of size $\frac{n}{k}\log(\frac{1}{\epsilon})$
$A = \phi$ ▷ *Denotes set of sampled points*
F $\leftarrow$ Submodular Utility function
for j= 0 to k **do**
    $\mathbf{T} \leftarrow$ a random subset obtained by sampling $\frac{n}{k}\log(\frac{1}{\epsilon})$ random elements from V $\setminus A_{t-1}$
    argmax $a_j \leftarrow argmax_{a \in \mathbf{T}} \Delta_{A_{t-1}}(a)$
    $A_t \leftarrow A_{t-1} \cup a_j$
end for

The above algorithm is due to [2] called the Lazier than Lazy greedy algorithm. This algorithm selects a set of random points $\mathbf{T}$ and performs the greedy optimization for the utility function(F) based on the set of points from $\mathbf{T}$. The size of $\mathbf{T}$ needs to greater than $\frac{n}{k}\log(\frac{1}{\epsilon})$ for an approximation guarantee of $(1 - e^{-1}) - \epsilon$ to the optimal solution. The above algorithm is run for each epoch independently giving a minibatch of size k at each epoch.

A probabilistic version of the the general greedy algorithm was recently presented in [6]. The extension of the probabilistic version to the case of lazier than lazy greedy is trivial since the probabilistic version is equivalent to the greedy algorithm for t=0 see [6]. A analysis for approximation guarantee of using probabilistic greedy with lazier than lazy greedy is left as a future work.

Algorithm 4 Prob SSGD Algorithm

Input: Training set X, Model M, Minibatch size k, n = $|X|$
Output: Selected batch R of size $\frac{n}{k}\log(\frac{1}{\epsilon})$
$A = \phi$ ▷ *Denotes set of sampled points*
F $\leftarrow$ Submodular Utility function
for t= 0 to k **do**
    $\mathbf{T} \leftarrow$ a random subset obtained by sampling $\frac{n}{k}\log(\frac{1}{\epsilon})$ random elements from V $\setminus A_{t-1}$
    $\forall$ a $\in \mathbf{T}$:
    $p_{(a|A_{t-1})} \leftarrow \frac{exp(\frac{1}{\alpha}\Delta_{A_{t-1}}(a)}{\sum_{a' \in T} exp(\frac{1}{\alpha}\Delta_{A_{t-1}}(a))}$
    $a^* \leftarrow$ Sample 'a' with probability $p_{a|A_{t-1}}$
    $A_t \leftarrow A_{t-1} \cup a^*$
end for

The above algorithm selects k data points based on the above-defined probability distribution 6. The above algorithm is run for each of the n epochs independently. The prob-ssgd algorithm can be thought of as circumventing the exploration-exploitation dilemma while choosing a minibatch. The minibatch created would still be diverse but, now the minibatch also includes data points which would never have been chosen by the Greedy-SSGD algorithm in any of the epochs. Till now we have only shown that creating a diverse minibatch problem is equivalent to maximizing the 7.2 and that the maximization problem is easily solved by Greedy-SSGD or ProbGreedy-SSGD if the function F is submodular.

Now we show how we can create such a function F. From the above we see that good choice for a Submodular Utility function is critical for the creation of diverse set and scaling of the sampling scheme. The function F besides trying to maximize coverage and diversity should also assign a high score to data points that the model is uncertain about. Since these points are the ones that would make the model learn faster. Also, the marginal gain for function F should be easy to evaluate. This is especially important in the case of Deep Learning Models as

these functions need to be evaluated for each minibatch creation.

Based on the above we define the following Utility function.

$$\begin{aligned} F(a_i|A_{i-1}, X) &= E(a_i) + \rho(a_i) + D(a_i, A_{i-1}) \\ &a_i : \text{i'th data point in set X;} \\ &A_{i-1} : \text{Previously selected items;} \\ &X : \text{Data Set} \end{aligned} \tag{7.3}$$

$$E(a_i) = -P(y|a_i, w_t) * logP(y|a_i, w_t) \tag{7.4}$$

$$w_t : \text{weight at iteration t} \tag{7.5}$$

$$p(a_i) = \frac{1}{|A|} \sum_{j=1}^{|A|} \phi(a_i, a_j) \tag{7.6}$$

$$D(a_i) = min_{a_j \in A: j \neq i} ||(a_i, a_j)||_p \tag{7.7}$$

Here A denotes a set of already sampled data points. Initially $A = \phi$ and it grows as more and more data points are sampled from the dataset.

Equation 7.5 denotes the Entropy of data point a_i based on the current model parameters at time t (w^t), it quantifies the uncertainty of the model on data point a_i. A high value of E(a_i) indicates the a_i should be sampled more frequently as it would make the model learn more quickly. In literature, this is also called as hard-example mining. Thought we have to cautious because a_i could also be an outlier making it of less importance. Hence we have to balance the exploration-exploitation dilemma.

Equation 7.6 denotes mean kernelized distance of a_i from the batch of data points that presently constitute A. Mean kernelized distance makes sure we select data points which is less similar to already selected data points A. Hence mean kernelized distance ensures that we select a diverse set of samples at each iteration.

Equation 7.7 avoids selection of duplicate samples in the batch.

7.1 Submodularity of Utility function

We now prove that the above score function is normalized monotonic and submodular.

Lemma 2. The utility function $\mathbf{F}(a_i|A_{i-1})$ is sub-modular and monotonic.

Proof. Let A_1 and A_2 be two subsets of data points such that $A_1 \subseteq A_2 \subseteq X$. Take a point a such that $a \in X \setminus A_2$. The marginal gain in utility by addition of a to A_1 is given by

$$\mathbf{F}(A_1 \cup a) - \mathbf{F}(A_1) = \rho(a) + E(a) + min_{a_j \in A_1} \phi(a, a_j) \tag{7.8}$$

The gain in utility by addition of a to A_2 is given by

$$\mathbf{F}(A_2 \cup a) - \mathbf{F}(A_2) = \rho(a) + E(a) + min_{a_j \in A_2} \phi(a, a_j). \tag{7.9}$$

Since $A_1 \subseteq A_2$ the minimum distance for point $'a'$ from any other points will always be greater for the set A_1 as

there may exist some point a_j in the superset A_2, which is closer to $'a'$ than any element in its subset A_1.

$$min_{a_j \in A_1} \phi(a, a_j) \geq min_{a_j \in A_2} \phi(a, a_j). \tag{7.10}$$

Thus we have

$$F(A_1 \cup a) - F(A_1) \geq F(A_2 \cup a) - F(A_2) \tag{7.11}$$

□

The score function is also monotonically non decreasing function. As by addition of $a \in X \setminus A_1$. Would change the utility function by $\rho(a) + E(a) + min_{a_j \in A1} \phi(a, a_j)$. Since distance and entropy functions are both positive, we have

$$F(A_1 \cup a) \geq F(A_1) \tag{7.12}$$

It also normalized as $F(\phi) = 0$ Hence we have proven that our utility function is a monotonic submodular function.

7.2 Unbiased and Variance Reduction

Variance reduction using submodular functions has been studied in the context of sensor placement problems [27]. In their seminal work [7] show that the problem of variance reduction can be solved as a submodular maximization problem for many functions in the context of forward linear regression.
As stated earlier SGD algorithms can introduce a large variance in the gradients calculations which can lead to slow convergence of these algorithms. We discuss here why the convergence of Greedy-SSGD and ProbGreedy-SGD is faster than random SGD. We note some key insights into how Greedy-SSGD and ProbGreedy-SSGD are faster due to a decrease of the variance in the calculation of the minibatch gradient. The proof is inspired from [18] which proves the above results for DPP sampling.

Notation Let $p_i \in \{0, 1\}$ be an indicator random variable indicating if the i_{th} data point was selected according to the ProbGreedy or Greedy SSGD algorithm. $q_i = \mathbb{E}(p_i)$ denotes the expected value of selection under the sampling scheme.
Proposition 1. The following the SGD scheme leads to an unbiased stochastic gradient

$$\theta_{t+1} = \theta_t - \eta \frac{1}{q_i} \sum_{i=1}^{N} p_i \nabla(\theta_t; x_i) \tag{7.13}$$

This is due to the fact that $\mathbb{E}(\frac{1}{q_i} \sum_{i=1}^{N} p_i \nabla(\theta_t; x_i)$ =$\frac{1}{q_i} \sum_{i=1}^{N} \mathbb{E}(p_i) \nabla(\theta_t, x_i)$ = $\sum_{i=1}^{N} \nabla(\theta_t; x_i)$. Which is equal to the true gradient.

Then the correlation between points i and j can be written as

$$C_{ij} = \frac{\mathbb{E}[(p_i - q_i)(p_j - q_j)]}{\mathbb{E}(p_i)\,\mathbb{E}(p_j)} = \frac{\mathbb{E}(p_i, p_j)}{q_i, q_j} - 1 \tag{7.14}$$

Since the samples are not iid under the importance sampling distribution $\mathbb{E}(p_i, p_j) \neq \mathbb{E}(p_i)\,\mathbb{E}(p_j)$.

Theorem 3. *For all data points x_i, x_j for all parameters θ in the region of interest whenever the*

$$\forall_{i \neq j} \nabla(x_i, \theta)^T \nabla(x_j) C_{ij} < 0 \tag{7.15}$$

then SSGD has a lower variance than SGD algorithm.

The batch gradient and be represented as

$$\nabla_B = \sum_{i=1}^{N} p_i \nabla(\theta, x_i) \tag{7.16}$$

The full gradient and be represented as E(∇_B)

$$\nabla_F = \sum_{i=1}^{N} q_i \nabla(; x_i) \tag{7.17}$$

The difference between stochastic gradient and full gradient is given by

$$\nabla_b - \nabla_F = \sum_{i=1}^{N} (p_i - q_i) \nabla(\theta, x_i) \tag{7.18}$$

$$\text{Var}(\nabla_b) = Tr(Cov(\nabla_b)) = \mathbb{E}[(\nabla_b - \nabla_F)(\nabla_b - \nabla_F)^T] \tag{7.19}$$

$$\text{Var}(\nabla_b) = \sum_{i,j=1}^{N} \mathbb{E}\left[(p_i - q_i)(p_j - q_j)\right] \nabla(\theta, x_i)^T \nabla(\theta, x_j) \tag{7.20}$$

$$\text{Var}(\nabla_b) = \sum_{i,j=1}^{N} \mathbb{E}[(p_i p_j) - (q_i q_j)] \nabla(\theta, x_i)^T \nabla(\theta, x_j) \tag{7.21}$$

For computing $\mathbb{E}(p_i, p_j)$ we do the following

$$\mathbf{E}(p_i, p_j) = \mathbf{E}(p_i^2)\delta_{ij} + \mathbf{E}(p_i, p_j)(1 - \delta_{ij}) \tag{7.22}$$

$$= \mathbf{E}(p_i)\delta_{ij} + (C_{ij} + 1) q_i q_j (1 - \delta_{ij}) \tag{7.23}$$

Simplifying we get

$$\text{Var}(\nabla_b) = (q_i - q_i^2)||\nabla(x_i, \theta)|| + \sum_{i \neq j}^{N} C_{ij} q_i q_j \nabla(x_i, \theta)^T \nabla(x_j, \theta) \tag{7.24}$$

As we can see for the variance to be reduced the product between C_{ij} and $\nabla(x_i, \theta)^T \nabla(x_j, \theta)$ has to negative. We now justify why Greedy-SSGD and ProbGreedy-SSGD reduce variance. For any sampling function that is

diversity inducing we have the following cases: If we assume that the loss function is smooth then
Case 1: $\nabla(x_i, \theta)^T \nabla(x_j, \theta)$ are aligned we can assume that these points are spatially close to each other since the loss function is assumed to be smooth. If we use a kernel function in ssgd that is euclidean distance, close points would have negative correlation under SSGD sampling scheme making $C_{ij}\ b_i b_j \nabla(x_i, \theta)^T \nabla(x_j, \theta)$ negative.
Case2: $\nabla(x_i, \theta)^T \nabla(x_j, \theta)$ are not aligned we can assume that these points are far apart since the loss function is assumed to be smooth. If we use a kernel function in SSGD that is euclidean distance, far apart points would have negative correlation under SSGD sampling scheme making $C_{ij} b_i b_j \nabla(x_i, \theta)^T \nabla(x_j, \theta)$.
Both the cases are satisfied in GreedySSGD and ProbGreedy-SSGD proving that the indeed reduce the variance for minibatch SGD.

Part V

Experiments and Results

Chapter 8

Results

8.1 Datasets

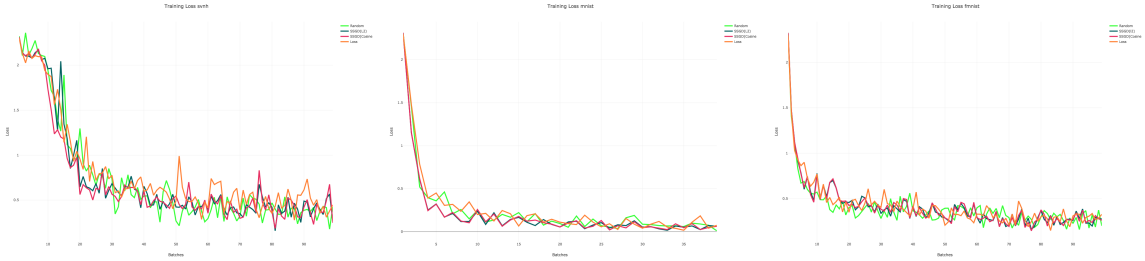
We use the following datasets for performing our experiments

Dataset	Train Set Size	Test Set	Classes	Remarks
MNIST	60,000	10,000	10	B/w Digits Images
F-MNIST	60,000	10,000	10	B/w Fashion Clothing Images
EMNIST	731,668	82,587	67	Handwritten Character Digits(Imbalance)
Oxford102	1030	6129	102	Color Flower Images/ Test set highly imabalanced
MIT-67	5360	1340	67	Indoor Images
Cifar-10	60,000	10,000	10	Real life objects

Table 8.1: Dataset Description

8.2 Vanilla SSGD and Vanilla ProbSSGD

We use pre-computed $L2$ and cosine distance features for MNIST, FMNIST, and SVNH dataset. For MNIST and FMNIST a 2 layer CNN was used and for SVNH a 3 layer CNN was used. For SGD we used a constant learning rate of 0.1 with a batch size of 64. We trained our network for around 10 epochs for each of the sampling methods.



(a) SVNH (b) MNIST (c) FMNIST

Figure 8.1: Loss Vs Batches when using the SSGD3 algorithm. There is no substantial increase the speed of convergence of sgd algorithm using our method. For distance metric with ssgd we used both L2 measure and Cosine Measure.

We found that our algorithm discussed in 3 and 4 are comparable to both Uniform Random Sampling and Loss Based sampling as can been seen from our results 8.1 Conclusion from the above experiments:

- Time: For a Batch Size=64 on MNIST dataset. We need to randomly sample 940 data points 64 times for selection of 64 data points. Even with pre-computed features this causes large overhead in the selection time for data points.
- Balanced Datasets: We also found that diversity based sampling methods does not have any significant advantage over other sampling schemes when the dataset is balanced.

For the algorithm to become more computationally efficient we propose the following changes,

8.2.1 Epsilon Greedy SSGD

Algorithm 5 Submodular-SGD (eps-SSGD) Algorithm

Input: Training set X, Model M, minibatch size k, $|X|$ = n, Fwd Batch Size L, ϵ
Output: Selected a random batch R of size L

$A = \phi$ ▷ *Denotes set of sampled points*
F $\leftarrow$ Submodular Utility function
for j= 0 to t **do**
  if $\epsilon > \mathcal{U}(0,1)$ **then**
    argmax $a_j \leftarrow \text{argmax}_{a \in \mathbf{T}} \Delta_{A_{t-1}}(a)$
  else
    $a_j \leftarrow \mathcal{U}(1,2,....n)$
  end if
  $A_t \leftarrow A_{t-1} \cup a_j$
end for

In the epsilon greedy alogrithm we first select L data points uniformly at random. We then select k data points from these L points. Note here that we do not sample R again and again for sampling each data point. We also introduce a concept of exploration and exploitation by choosing the best data point only ϵ times and randomly selecting data points at other times.

8.2.2 Adaptive Epsilon Greedy SSGD

We can see in the previous algorithm that value of ϵ does not change but, [21] suggest a adaptive value of ϵ according to Eq 6.24. We call this method the AdaptiveSSGD method. The value of ϵ keeps on updating according

Algorithm 6 Submodular-SGD (EpsSSGDMod) Algorithm

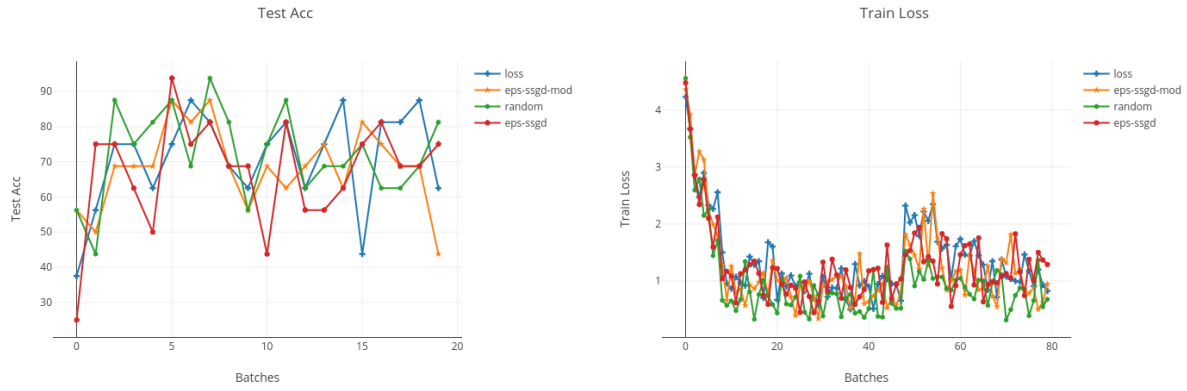
Input: Training set X, Model M, minibatch size k, $|X|$ = n, Fwd Batch Size L, ϵ, α
Output: Selected a random batch R of size L

$A = \phi$ ▷ *Denotes set of sampled points*
F $\leftarrow$ Submodular Utility function
for j= 0 to t **do**
  if $\epsilon > \mathcal{U}(0,1)$ **then**
    argmax $a_j \leftarrow \text{argmax}_{a \in \mathbf{T}} \Delta_{A_{t-1}}(a)$
  else
    $a_j \leftarrow \mathcal{U}(1, 2, ...n)$
  end if
  Update ϵ accroding to 6.24
  $A_t \leftarrow A_{t-1} \cup a_j$
end for

to 6.24.

8.2.3 Transfer Learning on MIT67

We test the above stated Algorithms6 and 5 on the task of transfer learning. Transfer learning is a very important task in the case of deep learning models. It is used when we want to use the existing features of a pre-trained network to make predictions on a new dataset.
We fine tune a Resnet-18 conv net by freezing all but the last layer. A batch size of 16 with the forward batch size of 64 was used. The learning rate is annealed by 0.1 every 7 epochs. The total number of training epochs is 24. For Distance calculation cosine distance between feature vectors of data points were used.



(a) Test Acc

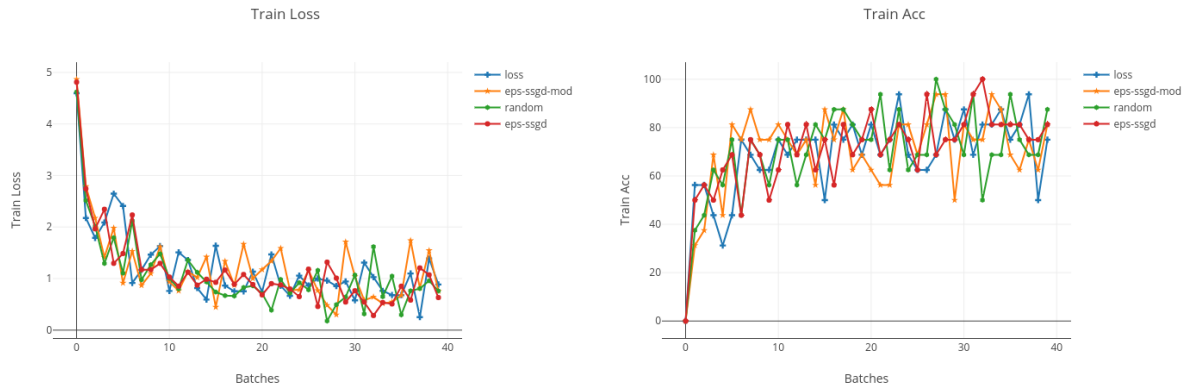
(b) Train Loss

Figure 8.2: (a) Test acc for random sampling is better than the our proposed method (b) Train Loss for random sampling is better than our proposed method .

As we can see that on the task of transfer learning for the MIT-67 dataset random sampling performs much better than SSGD and its variants. This can primarily be attributed to the balanced nature of the dataset.

8.3 Logistic Regression on Oxford Dataset

Our next experiment explores the need for adaptive feature vectors. The oxford dataset is a highly imbalanced dataset and for making our task harder we used the test set for training and the training set for test making the imbalance more severe. This imbalance is pretty commonplace in the real world.
We used the cosine distance metric on features from a pre-trained VGG-16 on the ImageNet dataset. With SGD of batch size 16 and forward batch size of 64. We annealed the learning rate after every 5 epochs by a factor of 0.1 The starting learning rate is set to 0.1. We run our experiment for 15 epochs.



(a) Train Loss

(b) Train Acc

Figure 8.3: Even with highly imbalanced dataset we see no major improvements with using SSGD and variants.

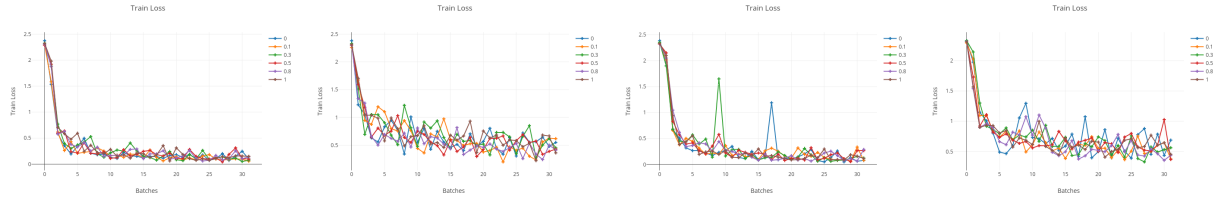
If we look closely at the above graphs, we can see that SSGD and variants show much better performance on the initial few batches. But after a while, we can see that random sampling becomes equivalent to the SSGD algorithm. This phenomenon occurs because of the annealing of the learning rates, as we keep on annealing the learning rates it counters the effect of using a more diverse batch. Since a diverse batch selects data points that have the minibatch gradient approximate the true gradient much better. But a smaller learning rate actually counters this effect of the better gradient by making only small increments in the parameters updates because of a small learning rate. We explore the effect of learning rate further in 8.3.2

8.3.1 Effect of ϵ value on Loss

We experimented by tweaking the value of ϵ in 5. We vary the epsilon value from 0 to 1 on both the MNIST and F-MNIST dataset. A batch size of 32 with the forward batch size of 128 was used. The learning rate was held constant at 0.01.

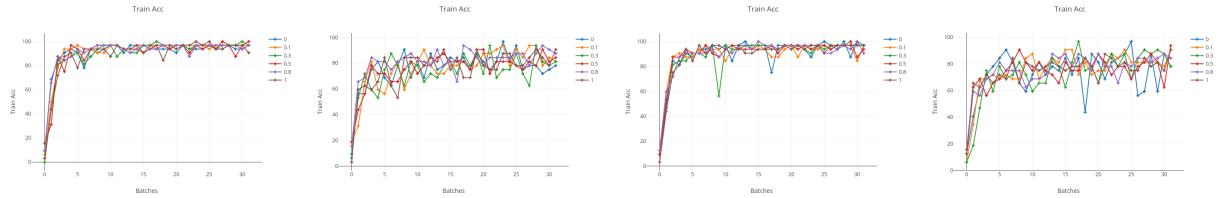
We also tested our submodular function F 7.5 with another standard submodular function from literature [8] called FLID function, which is also used to model diversity. We test both these function on Eps-SSGD[5] algorithm. The FlID function is given by

$$f(S) = \sum_{i \in S} u_i + \sum_{d=1}^{L} (\max_{i \in S} w_{i,d} - \sum_{i \in S} w_{i,d}) \tag{8.1}$$



(a) (b) (c) (d)

Figure 8.4: (a) & (b) denoted the training loss using ours submodular function f 7.5 on MNIST and FMNIST dataset respectively. (c) & (d) denote the training loss using FLID function on MNIST and FMNIST dataset respectively



(a) (b) (c) (d)

Figure 8.5: (a) & (b) denoted the training accuracy using ours submodular function f 7.5 on MNIST and FMNIST dataset respectively. (c) & (d) denote the training accuracy using FLID function on MNIST and FMNIST dataset respectively

The FLID functions is more sensitive to the epsilon value than our function F (Eq. 7.5).
The best values are

- FMNIST
  - Eps FLID: ϵ=0.1(Green)
  - Eps SSGD: ϵ=0.8(purple)
- MNIST
  - Eps FLID: ϵ=1(Green)
  - Eps SSGD: ϵ=0.1(yellow)

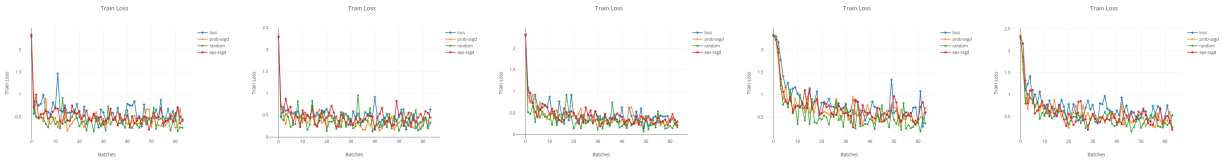
Though the best epsilon values dosen't show any important pattern but if we look more closely we see that if eps=0.1 both our function F(Eq. 7.5) and FLID function perform quite well. This shows that only small degree randomness needs to introduced for the submodular sampling schemes to perform better. This can be attributed to the exploration exploitation dilemma.

8.3.2 Effect of Learning Rate

In this experiment, we test the dependence of SSGD and its variants on the learning rate. Batch size was set to 32 and the forward batch size was set to 128 in this experiment. We trained our network for 10 epochs.
This experiment confirms that our method does actually create minibatches that are diverse since diverse mini-batches also entail large gradients. We see that in Fig 10.8(a) and Fig 10.6(a) there is a problem of explosion of the gradient in the SSGD and variants. This means that gradients very very large. These large gradients can helpful for faster convergence on difficult to train dataset.

(a) lr=0.2 (b) lr=0.1 (c) lr=0.01 (d) lr=0.001 (e) lr=0.002

Figure 8.6: Training Loss with different learning rates for MNIST dataset.

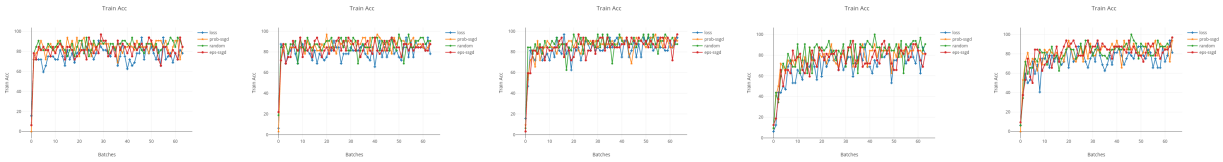


(a) lr=0.2 (b) lr=0.1 (c) lr=0.01 (d) lr=0.001 (e) lr=0.002

Figure 8.7: Training Loss with different learning rates for FMNIST dataset.

(a) lr=0.2 (b) lr=0.1 (c) lr=0.01 (d) lr=0.001 (e) lr=0.002

Figure 8.8: Training Accuracy with different learning rates for MNIST dataset.



(a) lr=0.2 (b) lr=0.1 (c) lr=0.01 (d) lr=0.001 (e) lr=0.002

Figure 8.9: Training Accuracy with different learning rates for FMNIST dataset.

8.3.3 Imbalanced Dataset

In this experiment, we use the EMNIST dataset which consists of handwritten character digits. It contains 814,255 images with 67 imbalanced classes. We show the applicability of our method on such imbalanced datasets.

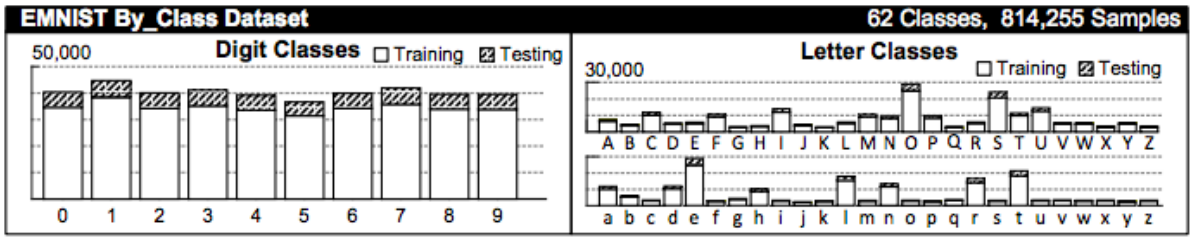
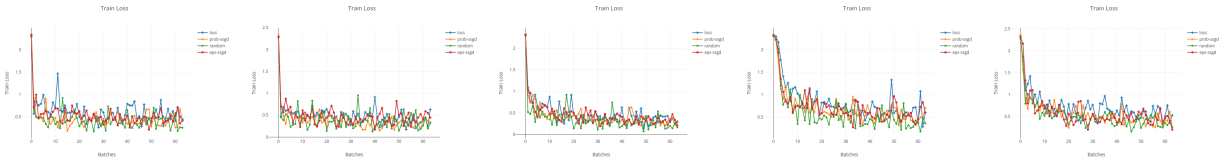
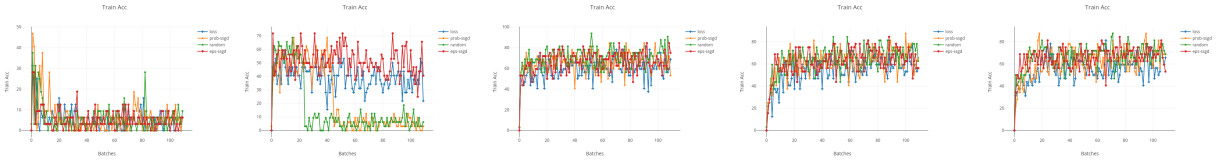


Figure 8.10: Description of EMNIST Image



(a) lr=0.2 (b) lr=0.1 (c) lr=0.01 (d) lr=0.001 (e) lr=0.002

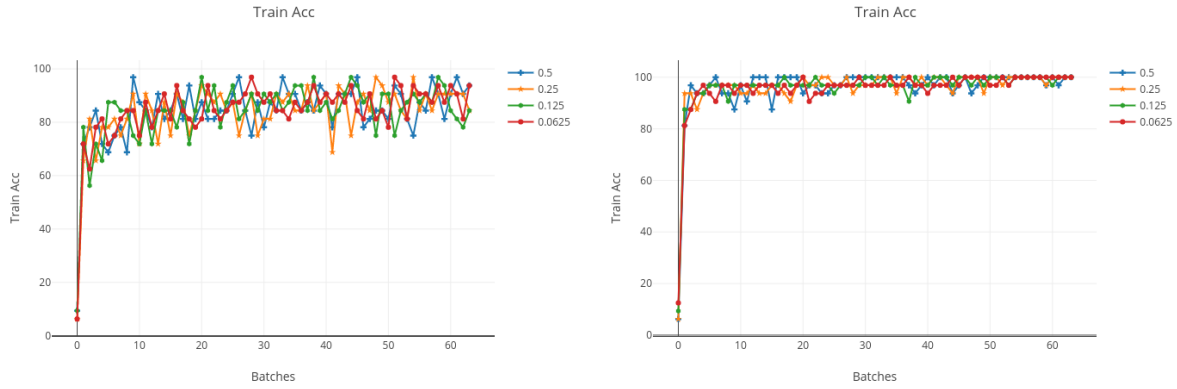
Figure 8.11: Training Loss with different learning rates for EMNIST dataset.



(a) lr=0.2 (b) lr=0.1 (c) lr=0.01 (d) lr=0.001 (e) lr=0.002

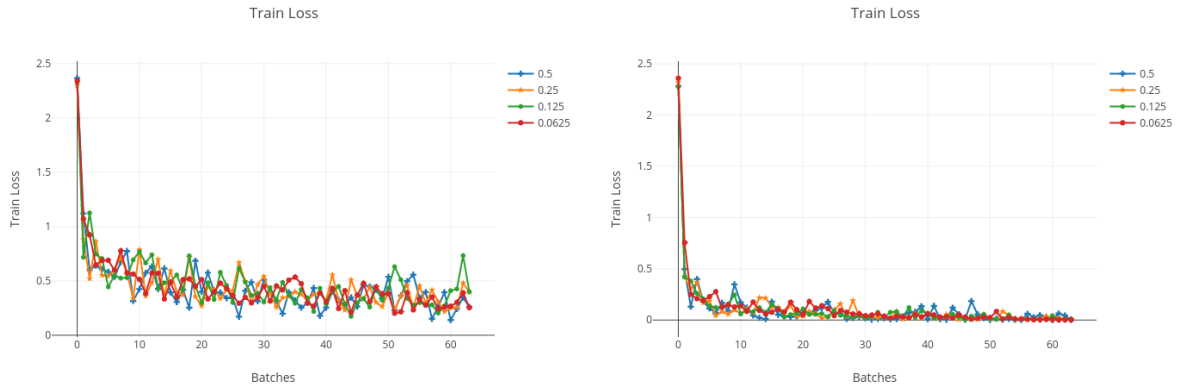
Figure 8.12: Training Accuracy with different learning rates for EMNIST dataset.

Imbalanced datasets are hard to train on and particular care should be taken about the learning rate parameter for these datasets. We see in Fig 10.11(a) and Fig 10.12(a) that for a learning rate of 0.2 all the algorithms failed. But the interesting case was for the learning rate of 0.1. As we can see that Eps-SSGD performs really well when a learning rate of 0.1 was used but random sampling fails to learn with that learning rate. This shows us that our algorithm can outperfom random sampling with the correctly tuned learning rate.



(a) (b)

Figure 8.13: Training loss (a) FMNIST (b) MNIST with different batch sizes on Eps-SSGD



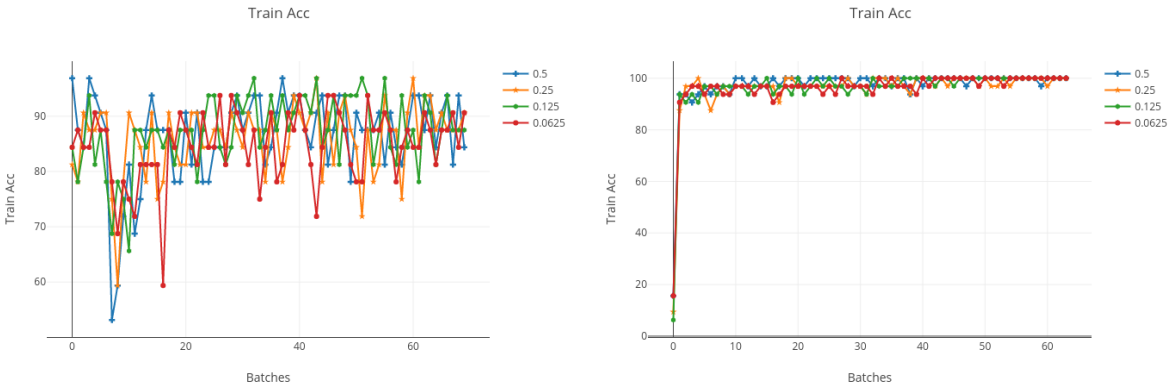
(a) (b)

Figure 8.14: Training Acc (a) FMNIST (b) MNIST with different batch sizes with Eps-SSGD

8.3.4 Effect of Batch Ratio

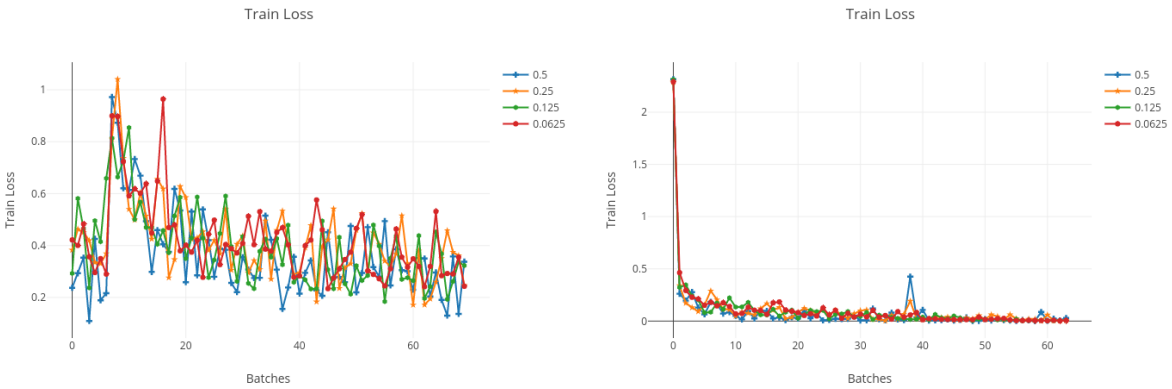
Since we modified the Lazier than Lazy greedy algorithm by selecting k samples from set of randomly sampled set of points of size B rather than selecting only one data point, we investigate the effect of batch ratio on the algorithm. We define Batch Ratio(BR)=$\frac{k}{B}$. We test both 3 and 4 algorithms on the FMNIST and MNIST dataset. We now investigate batch ratios effect on 6 algorithm

The results of these experiments are really interesting as it shows that there is not much correlation between batch ratio and the convergence of SSGD this is rather strange as it is orthogonal to the theoretical results in [2]. We leave the exploration of the theoretical consequence of this experiment as future work.



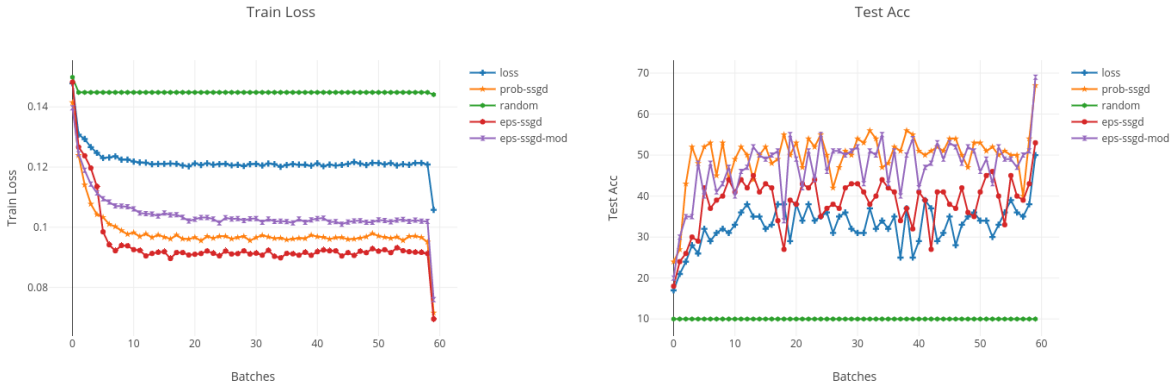
(a) (b)

Figure 8.15: Training Loss (a) FMNIST (b) MNIST with different batch sizes with Eps-SSGD-Mod



(a) (b)

Figure 8.16: Training Acc (a) FMNIST (b) MNIST with different batch sizes with Eps-SSGD-Mod



(a) (b)

Figure 8.17: (a) Training Loss of cifar-10 (b) Test Accuracy of cifar-10 dataset

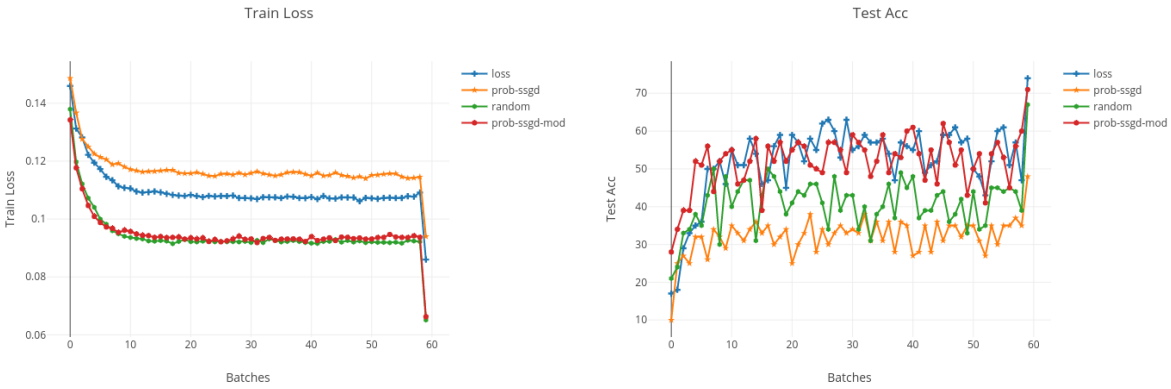
8.3.5 Results on Cifar10

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. We train a VGG-11 network using a batch size of 16 with a forward batch size of 64. We used a constant learning rate of 0.1 and trained the network for 60 epochs.

We see that on a hard to train dataset like Cifar-10, our algorithm performs much better than both random and loss based sampling. In this example, random sampling fails completely learning nothing this can be attributed to not tuning the learning rate for SGD.

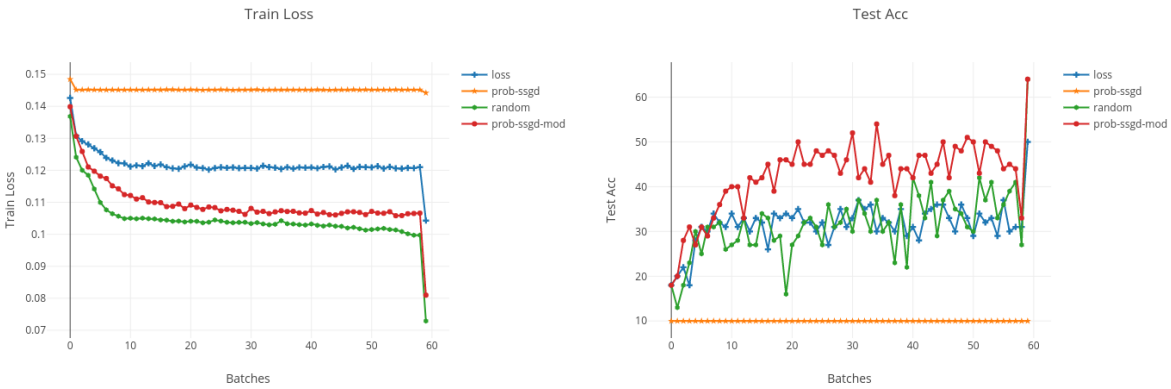
8.3.6 Effect of Depth

As the depth on the network increases surprisingly random sampling starts to learn again this can be attributed to smoother loss landspace with deeper networks but even with deeper depths SSGD and its variants perform much better both on the train and test dataset.



(a) (b)

Figure 8.18: (a) Training Loss of cifar-10 (b) Test Accuracy of cifar-10 dataset for VGG-13



(a) (b)

Figure 8.19: (a) Training Loss of cifar-10 (b) Test Accuracy of cifar-10 dataset for VGG-16

8.4 Time Analysis

We now show the avg time per epochs for all tasks stated above. The standard SSGD

Task	Random	Loss	SSGD	ProbSSGD	ProbSSGD-Mod
Transfer Learning	≈ 10	≈ 13	-	≈ 22	≈ 20
Logistic Regression	≈ 2	≈ 9.6	-	≈ 4.6	≈ 2.4
Cifar-11	≈ 2.44	≈ 5.18	≈ 6.53	≈ 6.59	≈ 7.108
Cifar-13	≈ 3.84	≈ 6.81	-	≈ 9.05	≈ 7.82
Cifar-16	≈ 4.26	≈ 5.91	-	≈ 6.42	≈ 8.61

Table 8.2: Avg time per epoch in each of the tasks in min.

As we can see that SSGD and its variants are really slow as compared to random sampling. This heavily outweighs the benefits of using SSGD and variants. We need to come up with a better engineering solution for making submodular sampling more time efficient. This analysis does not show the full picture though because SSGD and its variant train the neural network in much lesser epochs than random sampling for datasets like Cifar-10.

Chapter 9

Conclusion and Future Work

In this work, we presented a novel framework to sample a diverse minibatch at each iteration for faster convergence of SGD algorithm. We argue that modeling diversity is an NP-hard problem but can be solved easily if we use submodular functions. We introduced a novel submodular function and several new submodular optimization techniques for scaling to a large dataset. Our submodular optimization methods reduce variance in SGD in a provable manner. We applied our framework on different datasets and different tasks like transfer learning, Logistic Regression, Image Classification etc and extensively studied the effects of parameters such as learning rate, batch ratio, epsilon on the performance of our algorithm. The main contribution of our work is showing both empirically and theoretically that diversity-based sampling for minibatch creation can lead to variance reduction in SGD. Which in turn leads to faster convergence of SGD algorithms.
Although our algorithm works comparable to random sampling on many datasets such as FMNIST and MNIST we show that in the case of CIFAR-10 dataset our algorithms outperform random and loss based sampling by huge margins. We also show that for imbalanced dataset if we tune the learning rate correctly we can see faster convergence of SGD using our algorithm. Another interesting result of our work was disproving the bounds in [2], we showed that the dependency on the batch ratio is not that critical. Our algorithm is much slower in terms of wall clock time as compared to the random sampling algorithm and we need to find better engineering solutions to this problem. Below we list a few possible directions to make our algorithms faster,

- Instead of assigning scores to each data points assigns probabilities to each subset using Probabilistic Submodular Functions.
- Use of distributed submodular maximization algorithms. These algorithms are really fast and have provable bounds for the accuracy of the results.

One area which is still left unexplored is the usage of our algorithm in the setting Natural Language Processing and Multitask training. This is a really interesting setting since submodular functions have been extensively used in both these areas. A diverse batch would really be useful in the case of multi-task learning as the batch would now contain information from all tasks leading to better models being trained. We leave the exploration of these task as future work.

Bibliography

[1] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsI. *Mathematical Programming* 14, (1978) 265–294.

[2] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier Than Lazy Greedy. In AAAI. 2015 1812–1818.

[3] G. Bouchard, T. Trouillon, J. Perez, and A. Gaidon. Accelerating stochastic gradient descent via online learning to sample. *arXiv preprint arXiv:1506.09016* .

[4] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016 761–769.

[5] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In Optimization techniques, 234–243. Springer, 1978.

[6] S. Tschiatschek, A. Sahin, and A. Krause. Differentiable Submodular Maximization. *arXiv preprint arXiv:1803.01785* .

[7] A. Das and D. Kempe. Algorithms for subset selection in linear regression. In Proceedings of the fortieth annual ACM symposium on Theory of computing. ACM, 2008 45–54.

[8] S. Tschiatschek, J. Djolonga, and A. Krause. Learning Probabilistic Submodular Diversity Models Via Noise Contrastive Estimation. In AISTATS. 2016 770–779.

[9] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization* 30, (1992) 838–855.

[10] Y. Nesterov. Introductory lectures on convex optimization: A basic course, volume 87. Springer Science & Business Media, 2013.

[11] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In Advances in neural information processing systems. 2013 315–323.

[12] M. Schmidt, N. Le Roux, and F. Bach. Minimizing Finite Sums with the Stochastic Average Gradient. arXiv. org .

[13] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In International Conference on Machine Learning. 2013 343–351.

[14] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* .

[15] L. Balles, J. Romero, and P. Hennig. Coupling adaptive batch sizes with learning rates. *arXiv preprint arXiv:1612.05086* .

[16] A. Devarakonda, M. Naumov, and M. Garland. AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv preprint arXiv:1712.02029* .

[17] S. Gopal. Adaptive sampling for SGD by exploiting side information. In International Conference on Machine Learning. 2016 364–372.

[18] C. Zhang, C. Öztireli, and S. Mandt. Diversified Mini-Batch Sampling using Repulsive Point Processes .

[19] C. Zhang, C. Öztireli, S. Mandt, and G. Salvi. Active Mini-Batch Sampling using Repulsive Point Processes. *arXiv preprint arXiv:1804.02772* .

[20] A. Katharopoulos and F. Fleuret. Biased Importance Sampling for Deep Neural Network Training. *arXiv preprint arXiv:1706.00043* .

[21] A. Katharopoulos and F. Fleuret. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. *arXiv preprint arXiv:1803.00942* .

[22] P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In international conference on machine learning. 2015 1–9.

[23] H.-S. Chang, E. Learned-Miller, and A. McCallum. Active Bias: Training More Accurate Neural Networks by Emphasizing High Variance Samples. In Advances in Neural Information Processing Systems. 2017 1003–1013.

[24] I. Loshchilov and F. Hutter. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343* .

[25] P. Zhao and T. Zhang. Accelerating minibatch stochastic gradient descent using stratified sampling. *arXiv preprint arXiv:1405.3080* .

[26] A. Ashkan, B. Kveton, S. Berkovsky, and Z. Wen. Optimal Greedy Diversity for Recommendation. In IJCAI. 2015 1742–1748.

[27] A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research* 9, (2008) 235–284.