Adaptive Learning with Dynamic Batch Creation Using Near-Neighbors

Apoorv Walia, Avi Singhal

Abstract

Deep neural networks (DNNs) are the most effective method for many classification tasks. The standard way to train DNNs is through backpropagation using stochastic gradient descent (SGD) based on mini-batches of fixed size. This training step is the computational bottleneck of DNNs, and it can take several days to train a large dataset on high-performance GPUs. In order to speed up the training process, we propose an adaptive batch selection method that adaptively chooses which datapoints to include in a mini-batch. In SGD, it is common to fix the batch size and iteratively apply the algorithm to all datapoints. Instead, our approach involves using near-neighbors of datapoints that led to the most learning (largest gradients). We sample datapoints for the next batch from the non-uniform probability distribution that is based on the loss/gradients previously computed for each datapoint. This allows SGD to focus on the most relevant training datapoints and progress faster.

Keywords: Near-Neighbor, Importance Sampling, Dynamic Batch creation, Convex loss function
Code Links:Github Link

1. Introduction

The abundance of available training data has made the use of deep neural networks practical, leading to significant improvements in many fields, particularly computer vision and natural language processing. However, the complexity of the optimization problem has made computational cost a major challenge in training these large networks. It is apparent that not all samples are equally important during training; many can be handled after just a few epochs and can be ignored without affecting the final model. To address this, we focus on the computation of samples that will have the greatest impact on the model's parameters. This reduces the variance of the gradient estimates and increases the convergence speed of stochastic gradient descent.
Importance sampling using the appropriate distribution is a method that can increase the convergence speed of stochastic gradient descent (SGD) as it will allow reduced variance of the estimator by focusing computation on samples that cause a change in the model's parameters. Different datapoints correspond to different loss function values, and this can result in significant differences in gradient amplitudes and directions. To address this, we propose an adaptive batch selection method that selects the datapoints for an algorithm in order to maximize its progress. Overall, our approach could enable faster and more efficient training of deep neural networks.

2. Related Works

There exist several works that try to improve uniform sampling employed by Stochastic Gradient Descent (SGD) to achieve better convergence rates and decrease the variance of SGD. (Beidi Chen, 2019) uses a Locality Sensitive Hashing (LSH) based approach to approximately sample from the optimal distribution. The optimal distribution is weighted by the L2-norms of the gradient. This approach overcomes the hurdle of maintaining the distribution of weighted norms after each iteration, sampling from which reduces the variance of the estimator. Creating the LSH table is a one time procedure and subsequent retrievals are done in constant time compared to the conventional O(N) time complexity to maintain the distribution. This approach, however, is only applicable to linear models.
Other methods, such as those of (Katharopoulos & Fleuret, 2018) attempt to sample using an upper bound approximation of the gradient norm. This leads to a biased estimator. Other works such as (Loshchilov & Hutter, 2015) employ a importance sampling scheme weighted by losses. They generate additional overhead by creating various structures to keep track of stale losses and their schemes to update them.
Our approach uses a near neighbor approximation technique to create our weighted sampling scheme and extends it to non-linear models.

3. Methodology

How do we choose our ”informative samples”? As alluded to before, we use loss/gradients, the samples which gives higher loss/gradients will help the model learn better. When using the loss as the feature for ”informative samples”, the key assumption is that loss is a good proxy for the gradients. We have performed experiments to verify the correctness of this hypothesis.

Our sampling scheme involves creating the next batch from the neighbours of the samples that generated the largest losses/gradients in the current iteration. Our near neighbor approach for dynamic sampling will be beneficial as long as the data points demonstrate a power law in the losses/gradients. If all the samples have similar losses/gradients, then the notion of data points with highest gradients does not make any sense since all of them are similar. We dynamically create the mini-batch for the next iteration using the nearest neighbours of these “top-K” samples. This is unique to our solution. While other related works maintain a history of losses or other proxies for each data point for sampling, we directly search for their ”Near-Neighbours”. Therein lies another assumption of our model. We believe the optimizing function to be locally smooth, this would allow points in close proximity to share similar loss values.

For this project we used exact near-neighbor search because we want to establish the correctness of the approach. The exact near-neighbor search is very slow for large datasets, but it can be easily improved by using efficient near-neighbor implementations like FAISS or other locality sensitive hashing based implementations.

The pipeline for our approach i.e. creating new batches using the Top-K losses and retrieving similar elements using FAISS, is defined as below:

Algorithm 1 Neighbor-based batch sampling for training neural networks

```
Input: Initial batch size $B$, maximum number of iterations
       $T$, number of near neighbors $k$, weights $w$
Output: Trained neural network
Initialize the neural network model with weights $\theta$;
 Create a random batch of size $B$: $\mathcal{B} = (x_i, y_i)i = 1^B$;
 while $t < T$ do
    Pass the batch $\mathcal{B}$ through the neural network to compute
      the loss: $L(\mathcal{B}; \theta) = \frac{1}{B} \sum i = 1^B \ell(f_\theta(x_i), y_i)$

      Compute the per-sample gradient: $g_i = \frac{\partial \ell(f_\theta(x_i), y_i)}{\partial \theta}$
      for each $(x_i, y_i) \in \mathcal{B}$;

      Sample $B$ datapoints for the next batch from
      the near neighbors of the current batch with
      weights $g$: $\mathcal{B}' = (x'i, y'i)i = 1^B$, where
      $x'i \sim \text{NN}_{\text{Euclidean}}(x_i, k, g)$;

      Update the neural network parameters using
      the gradient descent rule: $\theta \leftarrow \theta - \eta \nabla \theta L(\mathcal{B}; \theta)$;

      $\mathcal{B} \leftarrow \mathcal{B}'$
      Increment $t$ by 1
end
return *Trained neural network*
```

4. Experiment Settings

We used Pytorch version “1.12.1+cu113”, Google Colab, Wandb, Rice computation resources (servers) for the implementation of the project. The model configuration is 3 hidden layer neural network, each layer configuration is as follows: Fully connected layer $->$batch norm layer $->$activation function. The activation function used is relu for all the experiments. The number of hidden neurons for all the layers is 2048,512,128 for each layer respectively for all the experiments. The input and output layer dimensions depend on the dataset configuration. We used adam optimizer for minimizing the losses and used cross entropy loss as the objective function to minimize.
The datasets used for the experiments are:

- **HIGGS Data Set**:Link
- **SUSY Data Set**:Link
- **SKIN Data Set**:Link
- **CovType Data Set**:Link

Conventional dataloaders cannot be used for creating batches in our approach, this is because the data from which to create the new batches is not fixed in our approach, it changes on every iteration as per the TopK elements. So, we are loading data using a naive approach in which we use “for” loops and some variables to keep track of the total number of inputs over the current epoch.

To establish a fair comparison, we initialized the model once and saved the initial weights of the model. When comparing the results between our approach of batch creation with the random sampling approach, we initialized both models with the same weights. Our metric of comparison between the two approaches is the loss as a function of iterations. We use the same configuration for both the approaches (random sampling for batch creation and our proposed approach) i.e., same number of epochs, same number of steps, model, initial weights, first random batch.For comparing the training and validation losses, we created a subset of the dataset and used these subsets for the comparison between the two approaches at the end of each epoch.

Pytorch does not directly give the per sample gradient, so we use Functorch for getting the per sample gradients. Functorch is an efficient library by Pytorch for this purpose, but even though it is efficient, we still had to perform the forward and backward pass on each batch twice, once using the conventional Pytorch method (this was used to update the model) and the second was due to Functorch to get the per sample gradients which we further used for our methodology.

5. Experiments Conducted and Results

We continued our project from last semester where we were using loss as a metric for creating the next batch, however we observed that it was not giving the desired results even after incorporating importance sampling. The random approach always outperformed our methodology. We have previously tried using FAISS for retrieving near-neighbors efficiently but did not achieve good results. To debug the root cause we shifted to using exact near neighbors to first prove the correctness of our methodology. Once the correctness is proved, we can switch to any efficient near neighbor search implementation. We decided to use gradients instead of losses for next batch creation as using losses did not yield any good results. One of the reasons we feel that losses are not suitable is that the gradient flow does not only depend on the loss, it also depends on the activations of the internal layers, the activations in turn depend upon the input, so even though the loss might be low for a given data point, the gradient flow can be high. These are the datapoints that we need to consider to enable the model to learn better. Using loss as a parameter for creating the batches is leading us to miss out on such datapoints.

We used gradients as the metric for generating the new batches and incorporated importance sampling to ensure unbiased estimator and observed the following results.

In the above experiment, we tried out different batch sizes (256,512,1024). In the experiment, ”NN-K” means that we created a new batch which had batch size random samples + K% samples from near neighbors i.e if batch size is 256, then 256 random samples + k% of 256 near neighbor samples. ”NN–K” means that we used the exactly same batch size samples as in the ”NN-K” experiment + K% random samples. This was done to ensure a fair comparison between the random approach and our methodology. This ensured that the batch size and the elements are the same in both the experiments. The plot of the gradient norms is as expected, the near neighbor approach for each corresponding ”K” has higher gradient than its random version. Also the gradient norm increased as we increase ”K”. The loss plots indicate that random is outperforming our methodology for all values of ”K”. These results are for the HIGGS dataset, we ran the same experiment multiple times to ensure repeatability and observed the same results. We performed the same experiment on SUSY dataset and the results were the same, the random approach was better. But for SUSY, the trend of the gradients was also reversed, i.e the random approach gave higher gradients compared to the near-neighbor approach, this was counter-intuitive, the results are shown in figure 2. This suggested that our approach cannot be used on all datasets, it can only be used on those datasets where the gradient norms of near-neighbors are higher compared to random data points. We conducted experiments to identify which datasets can be used for our near-neighbor approach based batch creation.

The figure 3 shows the average gradient and the near neighbor gradient plots for all the datasets under consideration. In this experiment, we used the complete dataset and performed a gradient descent, we obtained the per sample gradient and calculated the average gradient. We also obtained the top32 datapoints with the highest gradients, then we obtained 32 near neighbors for each of the top32 datapoints. Then we calculated the average gradient of the neighbors of the top32 datapoints. The idea behind this experiment is that the average gradient should be lower than the gradient of near-neighbors. This is because this is the main idea behind our dynamic batch creation methodology. So if the average gradient and the near-neighbor gradients are similar, then our approach will not be useful for that particular dataset. It can be seen that the gradients for SUSY,HIGGS,COVTYPE converge after a few iterations, this highlights that they are not suitable for our approach and it also explains the reason that for the random approach to outperform our methodology. For the skin dataset, we observed that the near-neighbor gradients are much higher than the average gradients, so this could be a dataset where our approach can work. We used our approach on skin dataset and obtained results.

It can be seen in figure 4 that our method is better than random upto 80 iterations, after that the training converges and all the experiments reach similar value. Also the gradient plot shows higher gradients as we increase the percentage of near neighbor samples in the batch. In these experiments, the batch size is fixed and ”NN-K” implies that K % of samples in the batch are from near-neighbors and the rest are random. ”RANDOM” implies that all samples in the batch are randomly chosen. Another interesting thing in this dataset was that it was a low dimensional (3-d) dataset, which indicated that its loss function might be smooth compared to a higher dimensional dataset. We hypothesized that our approach will only work for smooth (convex) functions and will fail for non-convex cases. The reason for this hypothesis is that if the loss function vs data is non-convex, then the gradient of points which are close will not be similar, in this case our approach will not be valid. To confirm our hypothesis we conducted the below experiments. We also provide mathetmatical proof of this in the appendix.

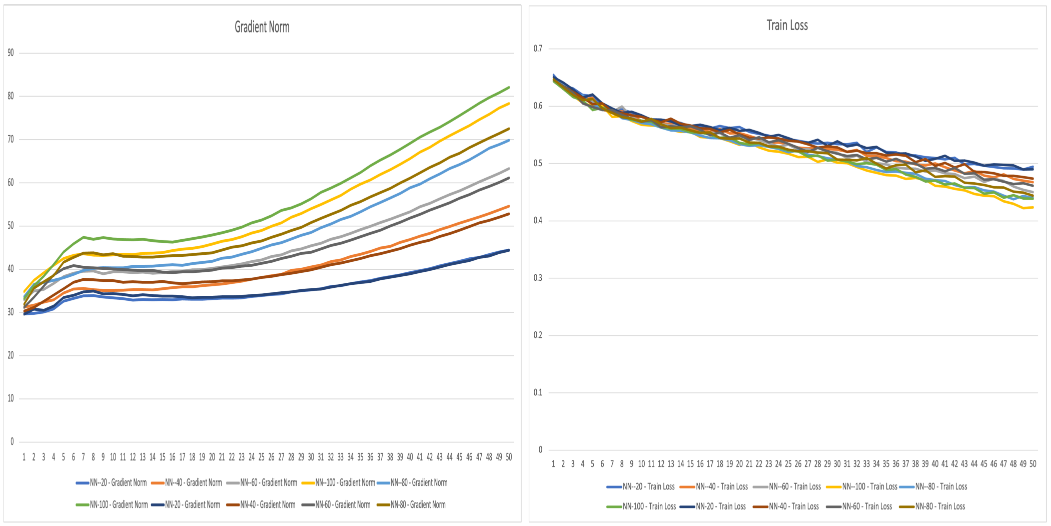


Figure 1. Using gradients for dynamic batch creation with importance sampling on HIGGS dataset.

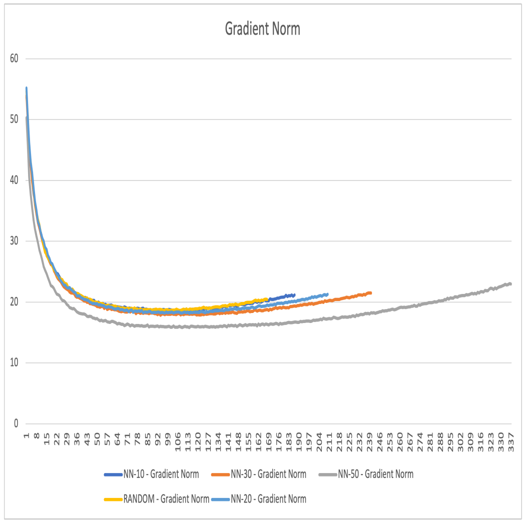


Figure 2. Gradients for SUSY dataset

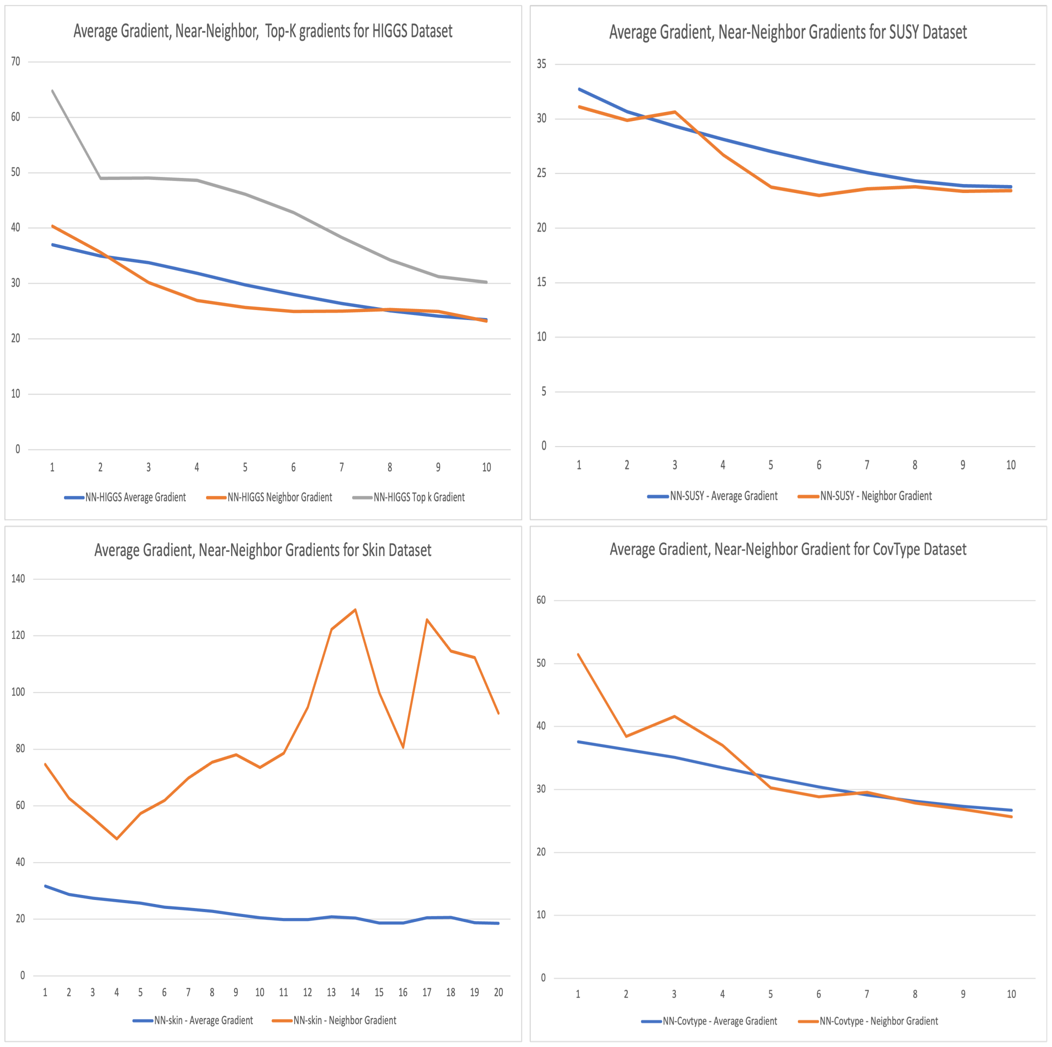


Figure 3. Average,Near-Neighbor Gradients for different datasets

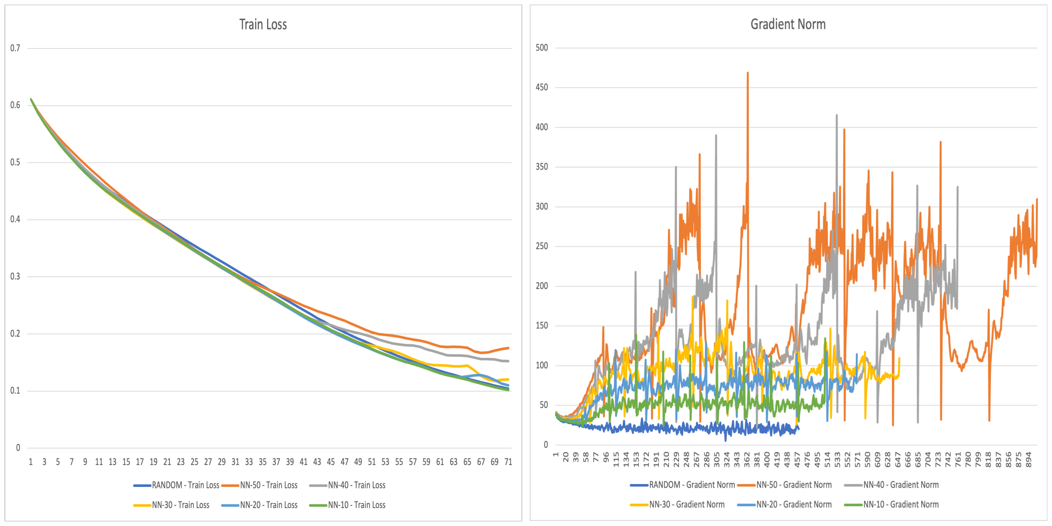


Figure 4. Dynamic Batch creation using SKIN dataset

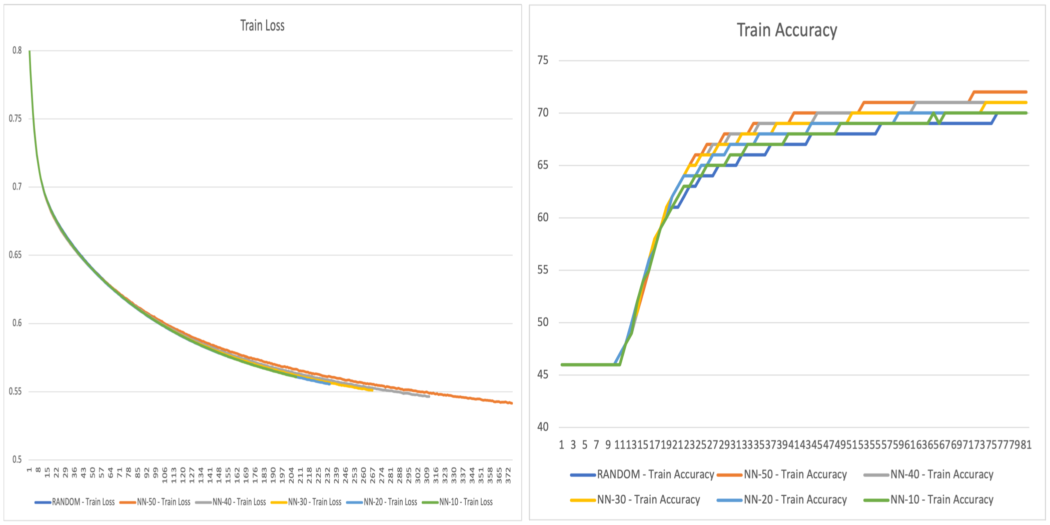


Figure 5. Dynamic Batch creation using Logistic Regression on SUSY dataset

We used logistic regression with binary cross entropy loss as it converts the loss function to convex space. We hypothesized that if we use this model and loss function on HIGSS and SUSY datasets, our approach should outperform the random, previously random was better. If our approach on SUSY and HIGGS dataset with logistic regression and binary cross entropy loss function outperforms the random method, then our hypothesis is correct.

Figure 5 presents the train loss and accuracy for SUSY, it can been that our approach outperforms random. Figure 6 presents the results for HIGGS. It can be seen that the number of iterations are different for the experiments in the figures, it is because we kept the batch size as a constant for all the experiments, so if batch size if 512, then for ”NN-50”, there will be 256 random samples and 256 near-neighbor samples. So the data loader was initialised with batch size 256 and then 256 samples from near-neighbor were appended. Since the size of the dataloader will be different for each ”NN-K”, hence the number of iterations is different. But we kept the epochs as constant, and also our comparison is based on the number of iterations to ensure the comparison is fair.

These results prove our hypothesis is correct and that our approach outperforms random for convex cases.

We further wanted to investigate the maximum distance up to which the gradients of the nearby points are similar to original data point. This would give us some idea about the smoothness of the loss function. For the skin-dataset, we observed that the maximum euclidean distance between the datapoints is 2. So we varied the euclidean distance from 0-2 in steps of 0.1. For HIGGS, the maximum euclidean distance between the datapoints is 4, so we varied the euclidean distance from 0-4 in steps of 0.1. We generated synthetic data points with the above euclidean distance from the original data point and observed the gradients. We generated 100 datapoints at each distance, for plotting we averaged the gradients of each of the 100 points. Figure 7 (in the appendix section) shows the results for the different experiments. We trained the model and saved the model a few epochs before convergence so that it shows the significance of using the near-neighbor approach for batch creation.

For Skin, the data point that gave the highest gradient has nearest neighbor at 0.01, the rest of the neighbors are farther (top 32 points lie within 0.03 distance). It can be seen in figure 7(b) that the gradients are very close to the original data point as expected, the same holds true for the case when we used logistic regression and hence our near-neighbor approach was working in both the cases for this dataset.

For HIGGS, the data point that gave the highest gradient has nearest neighbor at 0.7, the rest of the neighbors are farther (top 32 points lie within 0.9 distance). It can be seen in Figure 7(d), that starting at distance 0.7, we see non-decreasing and unpredictable gradient behavior which violates smoothness and explains why our near-neighbor approach did not work in this case. When we use Logistic regression on the same dataset, it can be seen that the curve is smooth upto 1 distance, and even after that the variation is much lower compared to the neural network experiment, hence our approach was working in this case.

However, in both the cases it can be seen that the gradients of the synthetic data is much larger than the true neighbor gradients for the same euclidean distance. We believe this is because the synthetic data has been generated from different distribution compared to the original data. Since the distribution is different, these points will give higher gradients as the model will want to learn this data. We verified this

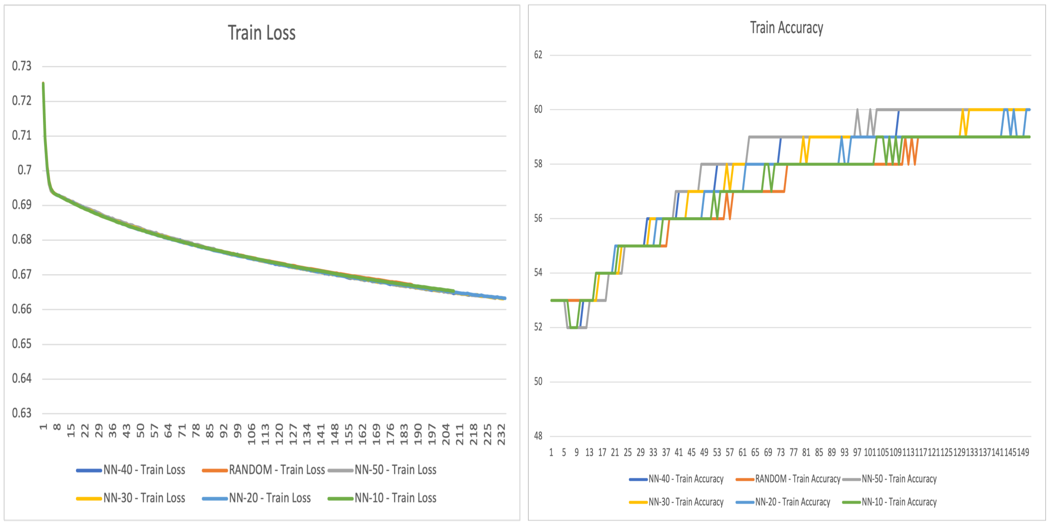


Figure 6. Dynamic Batch creation using Logistic Regression on HIGGS dataset

by checking the loss of the original data and the loss of the synthetic data, the loss of the synthetic data was much higher which confirms the claim that the distribution is different. This is akin to generating ”adversarial” examples for the model. Adding random noise to the original data-point creates something which the model has never seen before, thus generating large loss and gradients on these synthetic examples.

6. Conclusion and Future Work

We generated dynamic batches using nearest neighbors of samples with the largest gradients, thereby giving them more importance i,e we performed weighted sampling to create the next batch. We showed that using nearest neighbors to generate batches will enable more gradient flow through the model. We observed that not all datasets are suitable for our approach, the datasets where the near-neighbor gradients are higher than the average gradient are the most suitable datasets for our approach. We showed that our methodology outperforms the random for convex cases by converting the non-convex loss function to convex by using logistic regression. We also tried generating synthetic datapoints near the high gradient points to identify the distance upto which the gradients are similar to the original data point.

We noticed that the nearby points have similar gradients when we use logistic regression on HIGGS, SKIN dataset and hence our near-neighbor approach for dynamic batch creation is working. In the case of using a neural network, we observe that the gradients are different from the original data and are irregular for HIGGS dataset and hence our approach does not work there. We exaplined that the difference between the gradients of the true neighbors and synthetic neighbors is different due to generation from a different distribution.

Our work, however, shows promise and merits for further exploration. Our future work includes finding a solution for the non-convex case to optimize training of neural networks. We also plan to replace the exact near-neighbor search with an efficient implementation like FAISS or a locality sensitive hashing based approach.

Acknowledgements

We would like to thank Aditya Desai and Gaurav Gupta for their support and guidance throughout the project.

References

Beidi Chen, Yingchen Xu, A. S. Fast and accurate stochastic gradient estimation. In *Advances in Neural Information Processing Systems*, 2019.

Katharopoulos, A. and Fleuret, F. Not all samples are created equal: Deep learning with importance sampling, 2018. URL https://arxiv.org/abs/1803.00942.

Loshchilov, I. and Hutter, F. Online batch selection for faster training of neural networks. *CoRR*, abs/1511.06343, 2015. URL http://arxiv.org/abs/1511.06343.

Appendix

Here, we would like to present some mathematical proofs. Let's consider a machine learning model that maps an input data point x to an output y, i.e., $y = f(x)$. We assume that f is a differentiable function, and its gradient $\Delta f(x)$ is well-defined.

Now, let's define the gradient space S as the set of all gradients of f evaluated at all possible data points x in the input space X, i.e., $S = \{\Delta f(x) | x \in X\}$.

We want to show that if the gradient space S is convex, then the following statement is true:

"For any data point x_0 with a large gradient $||\Delta f(x_0)||$, its near neighbors x_1 and x_2 will also have large gradients, i.e., $||\Delta f(x_1)||$ and $||\Delta f(x_2)||$ are large."

Proof:
Let $f_\theta(x)$ be a neural network with parameters θ and input x, and let $L(\mathcal{B};\theta)$ be the loss function for a batch $\mathcal{B}$ with respect to the parameters θ. Let $g_i = \frac{\partial L}{\partial \theta}(f_\theta(x_i), y_i)$ be the gradient of the loss with respect to the parameters for a given input-output pair (x_i, y_i). Let S be the set of all input-output pairs (x, y).

Suppose that the gradient of the loss with respect to the input x is given by $\nabla_x L = \left(\frac{\partial L}{\partial x_1}, \ldots, \frac{\partial L}{\partial x_n}\right)$. Let $x_1, x_2 \in S$ be two input-output pairs with gradients g_1 and g_2, respectively. Suppose that x_2 is a nearest neighbor of x_1 in the input space, i.e., $|x_2 - x_1|_2 \leq |x_2 - x'|_2$ for all $x' \in S$ such that $x' \neq x_1$.

Assume that the data-gradient space is convex, i.e., for any two points $x_1, x_2 \in S$, the line segment connecting x_1 and x_2 lies entirely within the set of points that have gradients g such that $g = \frac{\partial L}{\partial x}(f_\theta(x), y)$ for some x and y.

Then, we claim that if g_1 is a high gradient, i.e., $|g_1|_2$ is large, then g_2 must also be a high gradient.

To see why this is true, suppose that $|g_1|_2$ is large but $|g_2|_2$ is small. Then, by the convexity assumption, there must exist a point x' on the line segment connecting x_1 and x_2 such that the gradient of the loss with respect to x' is equal to some g' such that $|g'|_2$ is small. But this contradicts the assumption that x_2 is the nearest neighbor of x_1 in the input space, since $|x_2 - x'|_2 < |x_2 - x_1|_2$ implies that x' is a closer neighbor of x_1 than x_2.

Therefore, we have shown that if the data-gradient space is convex and g_1 is a high gradient, then g_2 must also be a high gradient.

Now, let's prove that if the parameter-gradient space is convex, then so is the data-gradient space.

Proof:

Suppose we have a machine learning model with input space X, output space Y, and a parameter space W. Let $f : X \times W \to Y$ be the function that maps inputs and parameters to outputs.

Assume that the parameter-gradient space is convex. That is, for any two parameters w_1 and w_2 in W and any two outputs y_1 and y_2 that can be obtained by setting the parameters to w_1 and w_2, respectively, the convex combination of the gradients of y_1 and y_2 with respect to w_1 and w_2 lies in the parameter-gradient space. That is, for any $0 \leq \lambda \leq 1$,

$$\begin{aligned}&\nabla_w f(x, \lambda w_1 + (1-\lambda) w_2)\\ &= \lambda \nabla_w f(x, w_1) + (1-\lambda)\nabla_w f(x, w_2)\end{aligned}$$

for all $x \in X$.

We want to show that the data-gradient space is also convex. That is, for any two inputs x_1 and x_2 in X and any two outputs y_1 and y_2 that can be obtained by feeding x_1 and x_2 to the model, respectively, the convex combination of the gradients of y_1 and y_2 with respect to x_1 and x_2 lies in the data-gradient space. That is, for any $0 \leq \lambda \leq 1$,

$$\begin{aligned}&\nabla_x f(\lambda x_1 + (1-\lambda) x_2, w)\\ &= \lambda \nabla_x f(x_1, w) + (1-\lambda)\nabla_x f(x_2, w)\end{aligned}$$

for all $w \in W$.

To prove this, let x_1 and x_2 be two inputs in X, and let $y_1 = f(x_1, w_1)$ and $y_2 = f(x_2, w_2)$ be the corresponding outputs. Without loss of generality, we can assume that $w_1 = w_2 = w$. Then we have

$$\begin{aligned}&\nabla_x f(\lambda x_1 + (1-\lambda)x_2, w)\\ &= \nabla_x f(\lambda x_1 + (1-\lambda)x_2, \lambda w + (1-\lambda)w)\\ &= \lambda \nabla_x f(x_1, \lambda w + (1-\lambda)w) + (1-\lambda)\nabla_x f(x_2, \lambda w + (1-\lambda)w) \quad \text{(by convexity of parameter-gradient space)}\\ &= \lambda \nabla_x f(x_1, w) + (1-\lambda)\nabla_x f(x_2, w) \quad \text{(by the chain rule)}\end{aligned}$$

Therefore, the data-gradient space is also convex, as desired.

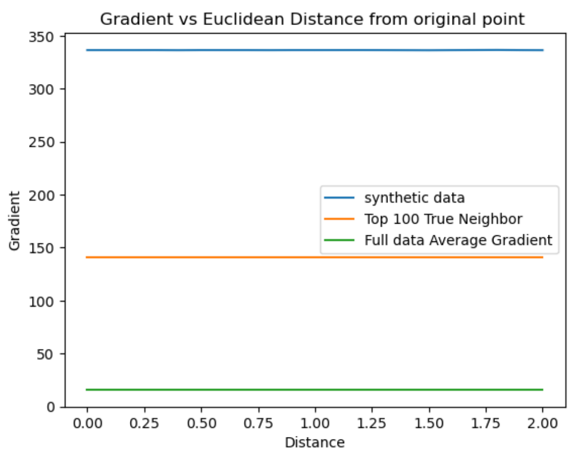
Finally we try to explain why our method may not work in a non-convex setting. We can show that if the gradient vs data space is not convex, then nearby points to x_0 with high gradients may not have high gradients.

Consider a neural network with non-convex loss function $\mathcal{L}(\theta)$, where θ denotes the model parameters. Let x be an input data point and let $f_\theta(x)$ be the model's output for that input. Suppose x has a high gradient, i.e., $\nabla_\theta \mathcal{L}(\theta)|_x$ is large.

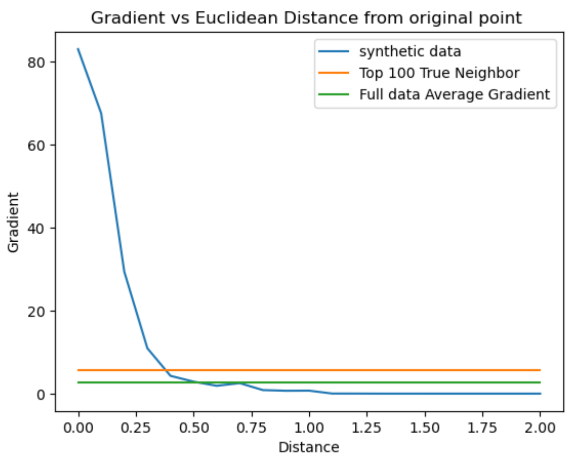
Now consider a neighboring input point x', but in a region where the loss function is relatively flat or has a lower gradient than at x. Then, the gradient of the loss function with respect to the model parameters at x', denoted by $\nabla_\theta \mathcal{L}(\theta)|x'$, could be much smaller than $\nabla \theta \mathcal{L}(\theta)|_x$.

This can happen, for example, if the loss function has a plateau or a saddle point in the neighborhood of x'. In such cases, the gradient of the loss function with respect to the model parameters can be small, even though the input point is a neighbor of a high gradient point.

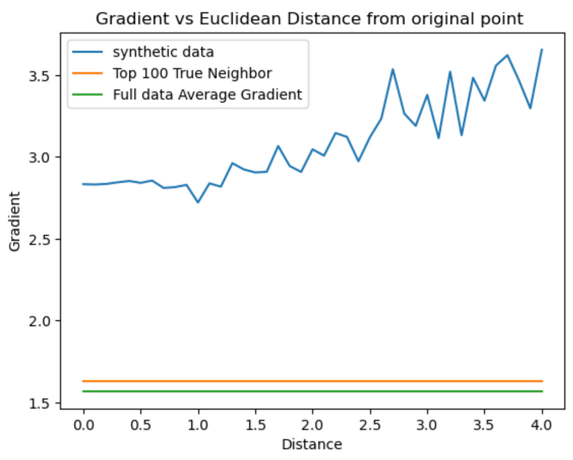
Therefore, in non-convex spaces, neighboring points of high gradient points do not necessarily have high gradients themselves.



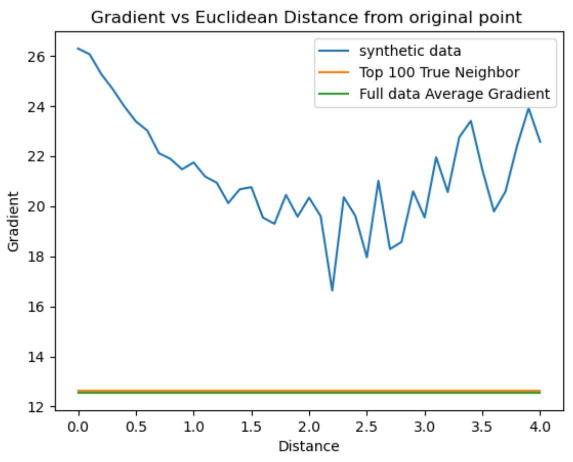
((a)) SKIN DATASET (Logistic Regression)



((b)) SKIN DATASET (3 Layer Neural Network +Cross entropy loss)



((c)) HIGGS DATASET(Logistic Regression))



((d)) HIGGS DATASET(3 Layer Neural Network +Cross entropy loss)

Figure 7. Varying euclidean distance of synthetic data point from original data point with different configurations