Nano Machine learning [MLG_16]

Abhishek Kumar	Manish Bera	Tushar Gupta	Prann Bansal	Madhukant
150035	150381	150771	150510	150368
abhikr	mkbera	tusharg	prann	mkant

Abstract

In this project we take a look at improving on the current techniques of implementation of Machine Learning algorithms in IoT(Internet of Things) devices, particularly we look at the formulation of k-NN based implementations. We study Proto-NN, a novel k-NN based algorithm for making predictions on resource scarce devices, and make improvements on it. Our major contribution is in minimizing(or entirely eliminating) the need for sending training points to a central server(probably a high-end super computer). We successfully investigate into two techniques, namely, (a) Online Proto-NN (b) Ballogorithm. In Online Proto-NN, we completely eliminate the need for sending any data to a server by making the training algorithm completely online. Ballogorithm explores the possibility of sending intelligently picked limited data to the server instead of sending the entire dataset.

1 Introduction

1.1 Motivation

With a boom in the embedded systems industries, IoT (Internet of Things) devices have become a household name. With the growing demand for smart devices, the need for algorithms to make intelligent decisions has increased. Implementation of Machine Learning algorithms on IoT devices has a wide range of applications. Smart watches will be able to detect impending heart attacks or strokes. IoT security will become more robust. Time Table schedulers will become smarter. The possibilities are enormous. But directly implementing Machine Learning algorithms on resource scarce devices is a challenge. With few kilo bytes of memory and even fewer mega hertz processing speed, getting reasonable accuracy like implementations on high end devices is a difficult task. But the need for 'intelligent' machines leads us to look for formulations that are compact (can be stored in low memory, low prediction cost) and also robust (not compromising accuracy).

1.2 k-Nearest Neighbors

One of the simplest and most versatile Machine Learning algorithms is the k-Nearest Neighbor. In the k-NN algorithm the test point is evaluated with its k-Nearest Neighbors and its label is predicted accordingly. The major problem with using k-NN in IoT devices is the space requirement. The k-NN algorithm does not have any training overhead, but instead stores all the points, to be evaluated at test time. This requires a huge amount of space and enormous amount of computation during prediction. All these factors prevent us from using the k-NN algorithm directly in IoT devices.

1.3 Proto-NN

Proto-NN by Manik Verma *et al.* is a novel algorithm, based on k-NN, which is focused towards making accurate predictions on resource scarce devices. Instead of storing all the points of the input data set, the Proto-NN algorithm stores 'prototypes', which act as representatives of the points in

Submitted as project report of CS771 2017-18(Sem I).

the input, and evaluates the test point on these ‘prototypes’ to make a prediction. More over, the algorithm transforms all the points in to a lower dimension space before evaluating them. This means that the prototypes too are in lower dimension. So, apart from storing less number of points, the formulation also stores them in a lower dimension. This leads to a huge amount of space savings.

In Proto-NN, the model is trained using SGD (Stochastic Gradient Descent) on all the input points. So all the training points are sent to a central server (maybe a super computer) so that the model can be trained and sent back to the IoT device, where the prediction can be made.

1.4 Our Contribution

The major draw back of Proto-NN the training takes place on a high end system. This forces the formulation to send all the training points from the IoT device to the central sever. This is, in some cases, undesirable. This implementation has some drawbacks:

1. Huge amount of energy consumption due to enormous data transmission
2. Implementation is impaled if network is jeopardized
3. Risk of data leak if transmission line is insecure
4. The IoT device has no way to update its model itself when it makes some mistake

We address these problems through two different approaches:

Online Proto-NN: In this approach, instead of training on a high end system, the training is done on the IoT device itself in an online fashion. This helps us prevent any data leakage over insecure transmission and helps address the security concerns.

Pros:

1. Will solve the problem of security
2. Can train at micro-controller itself
3. Can keep learning as and when required
4. Zero transmission cost

Cons:

1. Lot of Computation done on micro-controller
2. Will need a good processor

Ballogorithm: Here, instead of sending all the input points to the central server, we send less number of points which are chosen intelligently. This hugely reduces transmission costs.

Pros:

1. Can reduce transmission cost by order of 10
2. Approximately same accuracy as obtained by vanilla Proto-NN
3. Novel extensions possible in future

Cons:

1. Training done on server(we inherit this from Proto-NN)

2 Related Work

Many IoT devices in today’s make predictions based on Machine Learning algorithms. But most of the time the prediction is done on a remote server and relayed to the device. But this is undesirable because such a implementation hugely relies on network strength. The only way to go around this problem is to formulate a way to run prediction algorithms on the device itself.
One such attempt is *Bonsai* tree structure [2] which can be used for making predictions on IoT

devices. Unlike normal decisions trees, the *Bonsai* tree takes into account all the nodes that it had to traverse in order to reach the leaf node to make a decision. The input is sparsely projected into low dimension and then the contribution of each node along the traversal is aggregated to produce the outcome. All the node parameters are learnt jointly along with projection matrices using gradient descent and iterative hard thresholding.
Another work on resource scarce Machine Learning is *Two-Bit Networks* [3]. Two-Bit Network(TBN) is basically compressed CNN(Convoluted Neural Network). Conventional CNN can not be implemented with limited memory and processing power. Moreover large amount of space is required. In a TBN, the weights can take only four values(-2, -1, 1, 2), and thus each weight can be encoded in 2 bits. TBN is trained using SGD(Stochastic Gradient Descent).
The k-NN algorithm,although very beautiful and simple, demands huge amount of space and prediction time. In [4] the authors introduce SNC(Stochastic Neighbor Compression). Apart from speeding up the prediction in a k-NN based setting, SNC makes the model more robust to label noises. The space complexity and prediction time reduce from $O(nd)$ to $O(md)$, where n is the number of training points, m is compressed set of points and d is the number of dimensions of the feature vector. Proto-NN [1] is a novel Machine Learning algorithm based on k-NN. The algorithm learns a few number of prototypes which it uses to make predictions using a k-NN based technique. Apart from being small in number, the prototypes are sparse and in a lower dimension. This helps in space savings. Other model parameters such as projection matrix is also sparse. The model is learnt by SGD(Stochastic Gradient Descent) and IHT(Iterative Hard Thresholding). IHT helps keep the model sparse. Use of kernel machines in real world is limited because of high prediction cost. In [5] the authors introduce an approach to speed up kernel based machines. They provide analysis for modification on the classical Nystrom kernel approximation technique. They derive a divide-and-conquer technique to break the problem into sub-problems and in the process, speed up the prediction.

3 Online Proto-NN

We will take input data-points $\mathbf{x} \in \mathbb{R}^d$ and its label $\mathbf{y} \in \{0,1\}^L$ one by one. We want to learn a model for multi-classification in online fashion. Since it is a L label multi-classification problem $||\mathbf{y}|| = 1$. kNN requires you to keep all the data points in memory at the time of prediction but clearly we can't afford to have that. We maintain $\mathbf{B} = [\mathbf{b_1}, \mathbf{b_2}, \ldots \mathbf{b_m}] \in \mathbb{R}^{\widehat{d} \times d}$ where $\widehat{d}$ is the projected dimension in which we will do all our calculations. $\widehat{d}$ is significantly smaller than d, allowing us to do calculations on micro-controllers. We have label matrix $\mathbf{Z} = [\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_m}] \in \mathbb{R}^{L \times m}$. Note that we allow label vector to take real values instead binary values because we will do a sort of soft assignment of points to prototypes. Also we have $\mathbf{W} \in \mathbb{R}^{\widehat{d} \times d}$ the dimensionality reducing matrix. γ for Gaussian kernel is a hyper-parameter. Let $\sigma(\mathbf{v})$ be the function that gives index of largest component of vector $\mathbf{v}$. For a point $\mathbf{x}$ its label is calculated by following formula :

$$\widehat{y} = \sigma\left(\sum_{i=1}^{m} \mathbf{z_i} K_\gamma(\mathbf{Wx}, \mathbf{b_i})\right) \tag{1}$$

Note that if we let $\widehat{d} = d$ and $m = n$, then what we will obtain is standard Gaussian Kernel SVM boundary.

Our aim is to minimize the loss associated to $\mathbf{Z}, \mathbf{B}, \mathbf{W}$ using online gradient descent and alternating optimization. We will use squared loss function. Note that using logistic loss for classification problem makes more sense but that requires exponentiation and lot of floating point arithmetic. We would like to avoid such computations on 8 bit micro-controllers.

$$L_{\mathbf{x}}\left(\mathbf{y}, \sum_{i=1}^{m} \mathbf{z_i} K_\gamma(\mathbf{Wx}, \mathbf{b_i})\right) = ||\mathbf{y} - \sum_{i=1}^{m} \mathbf{z_i} K_\gamma(\mathbf{Wx}, \mathbf{b_i})||_2^2$$

Derivative of L with respect to $\mathbf{Z}, \mathbf{B}, \mathbf{W}$ are presented below. In the whole analysis we will denote vectors in bold font. Matrices in capitals. For a matrix $\mathbf{M}$, $\mathbf{M^i}$ denotes i^{th} column and $\mathbf{M_i}$ denotes

i^{th} row, M_{ij} denotes its ij^{th} element.

$$\frac{d}{d\mathbf{Z}}\left(||\mathbf{y}-\sum_{i=1}^{m}\mathbf{z_i}K_\gamma(\mathbf{Wx},\mathbf{b_i})||_2^2\right)=-2\mathbf{y}[K_\gamma(\mathbf{b_1},\mathbf{Wx})\dots K_\gamma(\mathbf{b_m},\mathbf{Wx})]+$$
$$2[\sum_{l=1}^{m}\mathbf{z_l}K_\gamma(\mathbf{b_1},\mathbf{Wx})K_\gamma(\mathbf{b_l},\mathbf{Wx}),\dots,\sum_{l=1}^{m}\mathbf{z_l}K_\gamma(\mathbf{b_m},\mathbf{Wx})K_\gamma(\mathbf{b_l},\mathbf{Wx})]$$

which can be written as

$$\Delta_{\mathbf{Z}}L_{\mathbf{x}}(\mathbf{Z},\mathbf{B},\mathbf{W})=X+Y \tag{2}$$

where

$$X_{ij}=-2y_iK_\gamma(\mathbf{b^j},\mathbf{Wx})$$
$$Y_{ij}=2K_\gamma(\mathbf{b^j},\mathbf{Wx})[\sum_{l=1}^{m}Z_{il}K_\gamma(\mathbf{b^l},\mathbf{Wx})]$$

$$\frac{d}{d\mathbf{B}}\left(||\mathbf{y}-\sum_{i=1}^{m}\mathbf{z_i}K_\gamma(\mathbf{Wx},\mathbf{b_i})||_2^2\right)=4[\mathbf{y}^\top\mathbf{Z}^j(\gamma)^2K_\gamma(\mathbf{b^j},\mathbf{Wx})(\mathbf{b^j}-\mathbf{Wx})]_{j=1\dots m}$$
$$-4\gamma^2[\sum_{l=1}^{m}\mathbf{Z_j}^\top\mathbf{Z_l}K_\gamma(\mathbf{b^j},\mathbf{Wx})(\mathbf{b^j}-\mathbf{Wx})K_\gamma(\mathbf{b^l},\mathbf{Wx})]_{j=1\dots m}$$

which can be written as

$$\Delta_{\mathbf{B}}L_x(\mathbf{Z},\mathbf{B},\mathbf{W})=X'+Y' \tag{3}$$

where

$$X'_{ij}=4\mathbf{y}^\top\mathbf{Z^j}(\gamma)^2K_\gamma(\mathbf{b^j},\mathbf{Wx})(B_{ij}-\mathbf{W_ix})$$
$$Y'_{ij}=-4(\gamma)^2(B_{ij}-\mathbf{W_ix})(\mathbf{Z^j})^\top K_\gamma(\mathbf{b^j},\mathbf{Wx})[\sum_{l=1}^{m}Z^lK_\gamma(\mathbf{b^l},\mathbf{Wx})]$$

Finally

$$\Delta_{\mathbf{W}}L_x(\mathbf{Z},\mathbf{B},\mathbf{W})=X''+Y'' \tag{4}$$

where

$$X''_{ij}=-4\mathbf{y}^\top(\gamma)^2x_j(\sum_{l=1}^{m}\mathbf{Z^l}K_\gamma(\mathbf{b^l},\mathbf{Wx})(B_{il}-\mathbf{W_ix})$$
$$Y''_{ij}=-2(\gamma)^2\sum_{k=1}^{m}\sum_{l=1}^{m}[\mathbf{Z^k}^\top\mathbf{Z^l}x_jK_\gamma(\mathbf{b^k},\mathbf{Wx})K_\gamma(\mathbf{b^l},\mathbf{Wx})(2\mathbf{W_ix}-B_{ik}-B_{il})]$$

3.1 Algorithm

We now present our algorithm. We will carry out alternating optimization over $\mathbf{Z},\mathbf{B},\mathbf{W}$

Algorithm 1 Online Proto-NN

```
1:  procedure ONLINE PROTO-NN
2:      Initialize $\mathbf{W},\mathbf{B},\mathbf{Z}$
3:
4:      while new datapoint $(\mathbf{x},\mathbf{y})$ comes do
5:          Predict its label using formula (1)
6:          for $e = 1$ to $epochs$ do
7:              $W = W - \eta_{\mathbf{W}}\Delta_{\mathbf{W}}L_x(\mathbf{Z},\mathbf{B},\mathbf{W})$ using formula (4)
8:              $B = B - \eta_{\mathbf{B}}\Delta_{\mathbf{B}}L_x(\mathbf{Z},\mathbf{B},\mathbf{W})$ using formula (3)
9:              $Z = Z - \eta_{\mathbf{Z}}\Delta_{\mathbf{Z}}L_x(\mathbf{Z},\mathbf{B},\mathbf{W})$ using formula (2)
10:         end for
11:     end while
12:     return $(\mathbf{Z},\mathbf{B},\mathbf{W})$
13: end procedure
```

Step Length : For step length we will adopt a simple strategy because we can't do too many calculations at micro-controller. Hence, we chose $\frac{C}{\sqrt{t}}$ and tuned C. We also experimented with constant step length.

Initialization We initialize $\mathbf{W}$ from multivariate Gaussian of $\mu = 0, \sigma = I_{\widehat{d} \times d}$. For $\mathbf{B}$ we take first m points and make them prototypes. We use their labels to initialize $\mathbf{Z}$.

Since the problem is non-convex, initialization and step length matter a lot. Trying out other strategies for them have been left for future.

Convergence Although the problem is non-convex still [1],[7] have shown that it converges to a local minimum. Moreover, if the loss function satisfies strong convexity in small neighborhood of global optimum, then appropriate initialization may give us convergence to global optimum even if we apply iterative hard thresholding on the gradients obtained [1], [8].

Time Complexity
The time taken to update the model for a single data point depends on original dimensions(D), projected dimensions(d), number of prototypes(n), number of labels(L) and epochs(e). The exact formula is as follows:

$$O(e(Dd + dn + Ln + nd + Lnd + n^2 d + n^2 L))$$

4 Hyper-parameters and Experiments for Online Proto-NN

We performed our experiments on MNIST dataset with 60K training points. Since our algorithm is online, testing and training had to be done simultaneously. We kept a separate batch of 20K points and kept testing our model after short intervals on these points and tuned our hyper-parameters to get the best possible accuracy. For the MNIST dataset, the original dimensions(d) is 100 and the the number of labels for multi-classification(L) is 3. The hyper-parameters involved in our experiments are gammaγ, $epochs$, step-length(η), projected dimensions($\widehat{d}$) and number of prototypes(m). Note that for simplicity we keep ($n_{\mathbf{W}} = n_{\mathbf{Z}} = n_{\mathbf{B}} = \eta$). The detailed observations are presented below.

4.1 Projected Dimensions

The following table shows how accuracy varies with change in projected dimensions as we receive more testing points.

Projected Dimension	500	10000	20000	30000	40000	50000	60000	Model Size(KB)
15	70.24	70.008	70.265	70.285	70.41	70.345	70.385	26.0
50	72.605	73.705	72.725	72.255	72.27	72.495	72.875	40.0
100	74.57	74.675	74.76	74.75	74.76	74.805	74.81	75.2

Clearly we can see that the accuracy is higher when the value of projected dimensions is high which was very much expected as when we project our data points to lower dimensions to train our model, we also loose some crucial information. The lower the value of projected dimensions, the higher the loss will be. But at the same time we can't have a high value of projected dimensions because it will lead to an increase in model size and total computation So we decide to take the value of projected dimensions to be 15.

4.2 Number of Prototypes

The following table shows how accuracy varies with change in projected dimensions as we receive more testing points.

Number of Prototypes	500	10000	20000	30000	40000	50000	60000
100	67.54	69.12	69.845	69.915	70.005	70.115	70.19
200	68.02	71.285	71.905	72.415	72.975	73.09	73.49

The hyper-parameter number of prototypes behaves similar to the hyper-parameter projected dimensions as increasing the value of number of prototypes will mean that we get more more number of representations for our data points which is bound to give better results but at the same time it will increase the size of model. So we choose the number of protoypes to be 100 which gives a model size of 26KB.

4.3 Step Length

The following table shows how accuracy varies with change in projected dimensions as we receive more testing points.

Step length	500	10000	20000	30000	40000	50000	60000
0.1	0.50005	0.50005	0.50005	0.50005	0.50005	0.50005	0.50005
0.01	68.05	74.98	76.98	77.19	76.09	77.81	78.67
0.005	71.31	74.12	76.06	76.78	76.66	76.905	78.34
0.001	70.09	71.845	72.87	73.08	73.86	74.175	74.82
0.0005	69.52	70.84	71.55	71.96	72.575	73.06	73.24
0.0001	70.44	70.51	70.625	70.84	71.305	71.255	71.22

Here we are using a constant step length. For various values of step length we find that the value 0.005 works best for MNIST dataset. The reason that we are not using step length computed using Adagrad etc. is that it will increase computations by a high margin and we want to avoid doing large computations as much as possible because of the constraint on memory and speed in an IOT device. So we stick we a constant step length.

4.4 Gamma

Gamma is the kernel hyper-parameter that keeps the value of kernel in check. Its value is constant and lies between 0 and 1. The following table shows how accuracy varies with change in value of gamma for the complete dataset.

Epochs	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
Accuracy	64.22	78.365	78.415	78.915	79.405	79.16	77.135	77.825	77.12	75.94

Clearly we get the highest accuracy at gamma equal to 0.25. So we will use this value of gamma for our computations.

4.5 Epochs

Epochs means number of times we do alternating optimization on a data point, i.e. it corresponds to number of times "for" loop runs in line 6 Algorithm 1 runs. The following table shows how accuracy varies with change in epochs for the complete dataset.

Epochs	1	2	3	4	5	6	7	8	9	10
Accuracy	79.295	78.895	79.505	79.555	78.895	79.535	78.485	76.135	77.99	75.105

We can observe that we get a good accuracy by using the value of epochs to be somewhere near 5. The Proto-NN code uses the value of epochs between 20 and 30 but we don't have that kind of luxury because as mentioned above in the time complexity analysis of Online Proto-NN, the time complexity directly depends on the value of epochs. So if the value of epochs becomes 10 times, the time taken will also become 10 times. Moreover since our devices have very less RAM and computation power, the time taken in a real scenario will be enormous. The devices will also consume much power. So we will use the value of epochs somewhere between 0 and 5.

4.6 Graphical representations for above hyper parameters

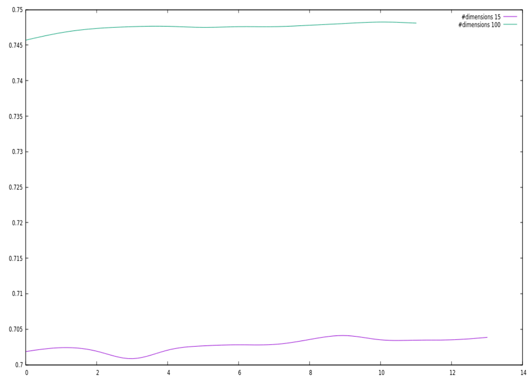

Figure 1: Projected Dimensions

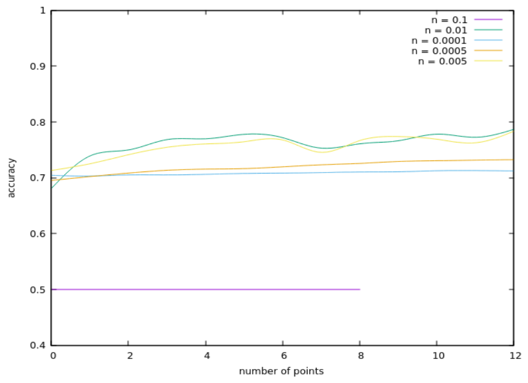

Figure 2: Step Length

The x and y axis represent number of points and accuracy respectively. Every 2 units on the x-axis stands for 6000 data points.

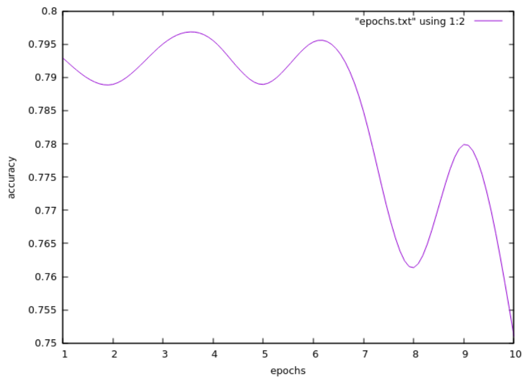

Figure 3: Epochs

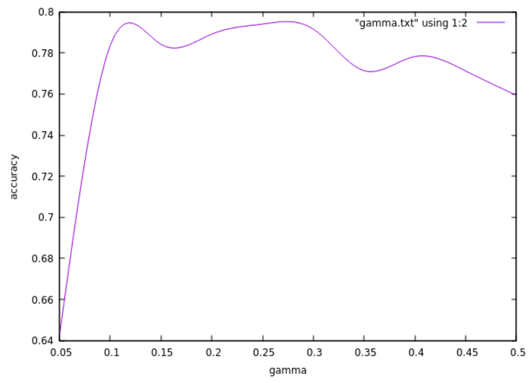

Figure 4: Gamma

5 Comparison between Proto-NN and ONLINE Proto-NN

	Online Proto-NN	Proto-NN	Proto-NN
Projected Dimensions	15	15	30
Number of Prototypes	100	100	200
Step Length	0.005	variable	variable
Gamma	0.25	0.3038	0.2161
Epochs	5	5	150
Accuracy	79.905	86.05	87.1

The sizes of final models computed by Proto-NN and Online algorithm are 26.0KB and 28.5KB respectively which are comparable. However the accuracy of Original Proto-NN exceeds that of Online algorithm. This is expected as we made the algorithm online at the cost of accuracy.

6 Ballogorithm

We have to note that the Ballogorithm is to be used only in the training phase. The job of this algorithm is to select a subset of the training points for sending to the server for getting a model using the original Proto-NN algorithm. We assume that points arrive in an online fashion and we have to either add them to a buffer (which we will send for training eventually) or to reject them straight away. Since we already have an embedded device on which we want to use our algorithm, we assume that the upper limit of the size of the buffer is predetermined, and thus, not a hyper parameter. The main points to keep in mind while building a core-set of points are :

- The points being sent should be representative of the entire data. In other words, the points which were not sent should be reasonably represented (both in quantity and values) in the final dataset.
- We must not exceed the memory limit at any point of time.

6.1 Algorithm

Building upon the above points, we will present the main idea for the ballogorithm.

We maintain a set of balls (initially empty) listed according to their class. Each ball has a center, radius, class and capacity associated with it. Hence, we have L lists, each with its own set of balls. We receive a new data point, say $(\mathbf{x}, y)$. We go to the list y and check if the point lies inside any preexisting ball. A point lies inside a ball if the distance between the center and the point is less than the radius, and the number of points inside is still less than the capacity. If the point is inside, we just delete the point and update the capacity of the ball. If not, we generate a new ball with $\mathbf{x}$ as its center, the preset radius and full capacity. If the number of balls exceeds a limit (which is decided by the memory limit), we just send all the balls to the server and reinitialize our lists.

The radius and capacity are hyper parameters that we will tune separately. We denote the set of lists as $l = \{l_1, l_2, ...l_L\}$. We denote each list as a set of balls, which were representative of the points belonging to the same class only. Let $l_i = \{B_1, B_2, ...B_{n_i}\}$ be the set of balls belonging to the ith class. Let each ball be denoted by $B_i = (\mathbf{c_i}, r, cap_i)$, where $\mathbf{c_i}$ is the center, r the radius and cap_i the remaining capacity of the ith ball. We now present our algorithm. We will carry out this code as long as we are in the training phase.

Algorithm 2 Ballogorithm

```
 1: procedure UPDATE-BALLS
 2:     Initialize $l = \{l_1, l_2, ...l_L\}, init\_capacity, r, buffer\_limit, n\_balls = 0$
 3:     while new datapoint $(\mathbf{x}, \mathbf{y})$ comes do
 4:         $list \leftarrow l_y$
 5:         for $i \in [1, n_y]$ do
 6:             if $dist(\mathbf{c_i}, \mathbf{x}) < r$ and $cap_i > 0$ then
 7:                 $cap_i \leftarrow cap_i - 1$
 8:                 continue
 9:             else
10:                 $newBall \leftarrow \{\mathbf{x}, r, init\_capacity\}$
11:                 $l_y \leftarrow l_y \bigcup newBall$
12:                 $n\_balls \leftarrow n\_balls + 1$
13:             end if
14:         end for
15:         if $n\_balls = buffer\_limit$ then
16:             $SendToServer(l)$
17:             $delete(l)$
18:             Reinitialize $l = \{l_1, l_2, ...l_L\}, n\_balls = 0$
19:         end if
20:     end while
21: end procedure
```

7 Hyper-parameters and Experiments for Ballogorithm

The only hyper parameters are the radius and the capacity of each ball. We will explore the intuition behind both of these, and their effects on the data transmitted to the server and the accuracy (both training and testing) of the models hence generated. Since we are using the MNIST for our comparisons, the default data-points transmitted are 60,000. For all the given data, the buffer size was set to 1,000 points.

Radius defines the maximum distance within which a point can be assigned to a given ball. Since all points in the ball will be represented by the ball center, it is important that we restrict the distance within which the ball can gather points. It may be intuitive that increasing the radius would just reduce the number of balls (at least when the capacities are large). However, in the subsection below, we can see that independent of capacity, smaller radius may lead to fewer, and hence densely packed balls.

Non triviality of Hyper parameter: radius

Consider the points (0,0), (3.9,0), (3.9,2.5) and (3.9,-2.5) in a 2D plane as shown below:

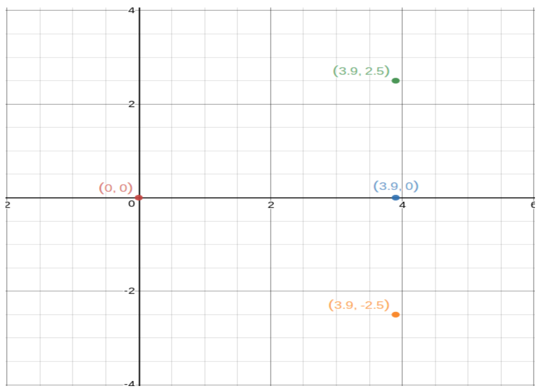

Figure 5: Projected Dimensions

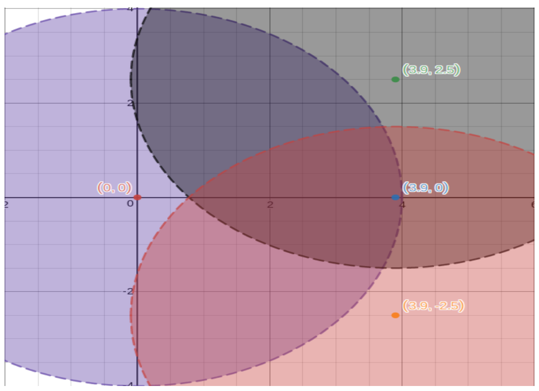

Figure 6: Step Length

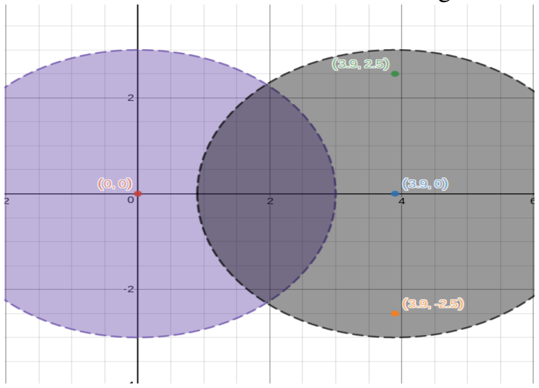

Figure 7: Step Length

Now, using ball radius 4, we will run our algorithm on these points in the same order as above. We reach the status below. As we can see, we need 3 balls to cover the set. Now, we will try the same problem with radius 3. As we can see, we needed only 2 balls in this case.

Fortunately, this phenomena can be neglected when there are a very large number of points and limited capacities are assigned to the balls. However, it is possible that while trying to tune radius at a finer level, we may reach a point where the number of balls sent are at a local minimum. The global minimum, obviously, will occur at infinite radius.

Capacity The capacity defines the number of points that are allowed to fit inside a ball. Capacity was implemented so that points occurring in a dense region in the input get represented fairly in the final compressed dataset. If the capacities were not defined (and hence, infinite), the test accuracies fall down to about 50%. Again, we might think that increasing capacities will lead to fewer balls being generated. However, similar to previous subsection, it can be shown to affect the points being sent non trivially. Again, in the general trend, we see that increasing the capacities does indeed lead to fewer points.

7.1 Experimental Results

Radius	Capacity	Data Sent	Training Accuracy	Testing Accuracy
1	5	25184	84.7	86.4
1	10	20182	84.0	86.1
1	15	17771	83.0	84.9
1	20	16584	81.7	85
1	25	15765	81.6	83.8
1	30	15278	81.0	84.8
2	5	19730	87.8	85.7
2	10	13305	87	84
2	15	10339	86.8	83.9
2	20	8516	87.9	81.8
2	25	7454	85.6	82.1
2	30	6505	87.6	82.3
3	5	19602	88	85.6
3	10	13275	88	83.7
3	15	10207	86.8	83.3
3	20	8506	87	82.9
3	25	7222	87.9	82.7
3	30	6305	88.2	81.8
4	5	19456	88.3	84.8
4	10	13073	87.1	84
4	15	10202	87	83.6
4	20	8568	87.9	82.8
4	25	7390	89.1	81.7
4	30	6393	88.3	81.2
5	5	19456	88.3	84.8
5	10	13073	87.1	84
5	15	10202	87	83.6
5	20	8568	87.9	82.8
5	25	7390	89.1	81.7
5	30	6393	88.3	81.2

7.1.1 Data Sent

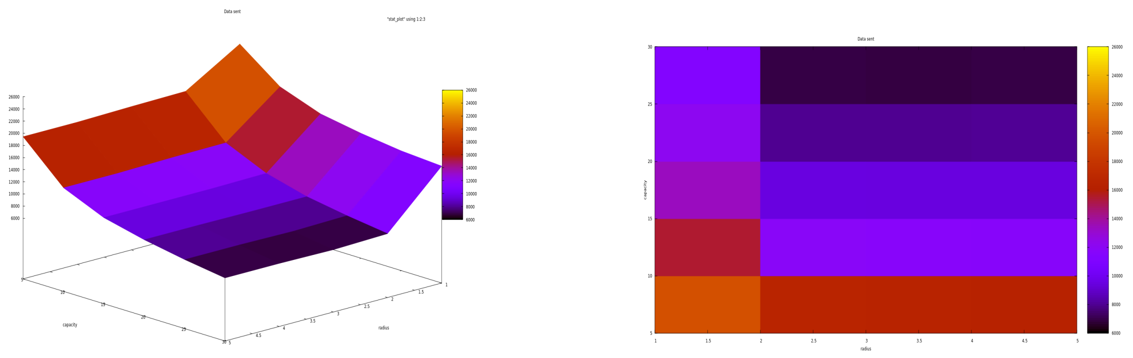

Figure 8: Amount of data send to server vs radius and capacity

7.1.2 Training Accuracy

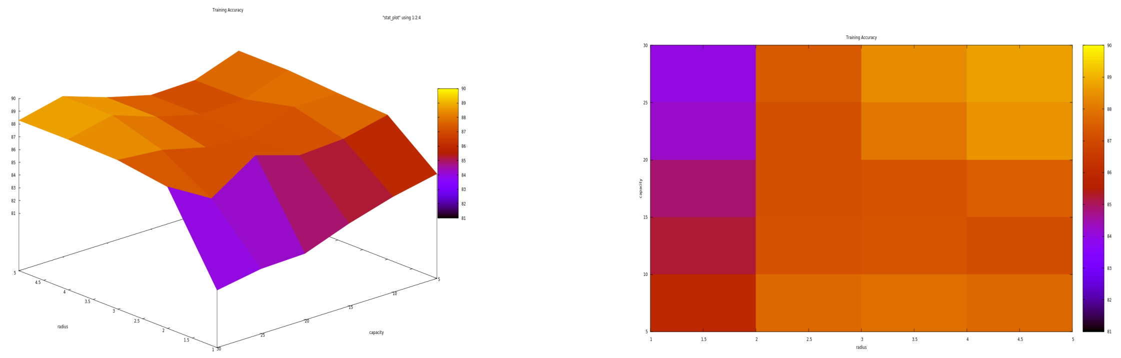

Figure 9: Training accuracy vs radius and capacity

7.1.3 Testing Accuracy

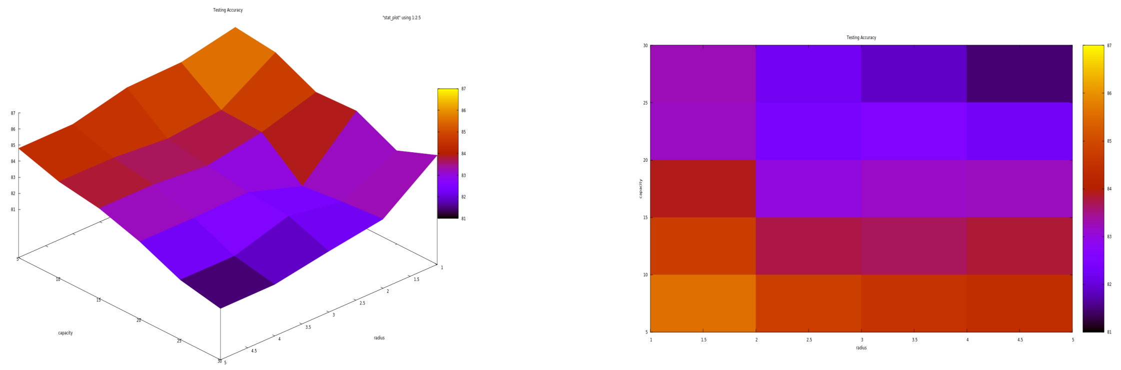

Figure 10: Testing accuracy vs radius and capacity

As we can see from the above data, we are getting test accuracies of 81.2% to 86.4% depending on the number of points sent. The original Proto-NN, using the same hyper parameters, gives 87.1% accuracy. However, even if we are willing to reduce our accuracy margin by as little as 0.7%, we can afford to send less than 42% of the data-points. Also, the lowest accuracy of 81.2% comes at the privilege of sending only 10.6% of the data. The trade-off, clearly, is between the number of points sent and the accuracy we get. Depending on the application and hardware used, our preferences may vary, and we may want to sacrifice more accuracy to get extended battery life.

8 Other Failed Attempts

8.1 Percepto-NN

This was our first failed attempt. We tried to tweak the Proto-NN procedure by reducing the number of points to main server. Sending all the data to main server through any network is not considered to be good in terms of power consumption and security issues.

In this we used the perceptron method and tried to send only the misclassified points. More specifically we initially send all incoming data points to main server until a particular fraction of data set is not reached , as we used MNIST 60k data set so we send $1/6^{th}$ i.e. 10k points to main server and ran Proto-NN Algorithm on it to get an initial model. This gave us an initial model which consist dimensionality reduction matrix W, Prototypes B and Score vectors Z. Using this model we start executing our Percepto-NN method -

Algorithm 3 Percepto-NN

```
 1: procedure PERCEPTO-NN
 2:     Receive $\mathbf{W}, \mathbf{B}, \mathbf{Z}$ from server
 3:     while new datapoint $(\mathbf{x}, \mathbf{y})$ comes do
 4:         Predict its label using formula (1)
 5:         if predicted label is NOT correct then
 6:             send $(\mathbf{x}, \mathbf{y})$ to main server
 7:             $accuracy \leftarrow$ get_validation accuracy
 8:             if $accuracy < min\_accuracy$ then
 9:                 Receive new $\mathbf{W}, \mathbf{B}, \mathbf{Z}$ from server
10:             end if
11:         end if
12:     end while
13:     return $(\mathbf{Z}, \mathbf{B}, \mathbf{W})$
14: end procedure
```

While getting new models from server, Proto-NN is ran only on the initially received points that consist the initial $1/6^{th}$ points and all the mis classified points till now.

Problem :

This method clearly doesn't work for the data set that are not entirely separable. As one can see that the outliers in data set will always be misclassified and if a data set contain only 1% outliers then Model will be trained to classify these outliers but these outliers can't be separated thus our model can't become a good model.

One can see in this perspective also that Model is not getting rewarded for the correct predictions. We are always penalizing the model if it is predicting incorrectly, and forcing it to predict them correctly.Surely a large part of previous correctly classified data points will get wrongly predicted in the updated models. and hence the training accuracy will be badly affected which is shown in the table also.

Proto-NN	1	2	3	4	5	6
Training Accuracy	84.83	79.72	71.31	66.56	61.52	56.75
Testing Accuracy	82.38	83.94	84.88	84.72	83.16	82.87

For experimentation purpose we invoke the server to give us recomputed models after every 10k points, For MNIST 60k we receive recomputed model 6 times as MNIST 60k contain 60k points.

8.2 Second Approach

In our previous approach we were not rewarding the model for its correct prediction in any form hence model was performing badly on the previously correctly classified points and giving bad training accuracy.

So, this time we tried to reward the model also for its correct prediction by assigning each prototype a weight in soft manner.

Algorithm 4 Second Approach

```
1: procedure SECOND APPROACH
2:     Receive initial $\mathbf{W}, \mathbf{B}, \mathbf{Z}$ from server
3:     $\mathbf{C} \leftarrow$ Receive initial weights of each prototype
4:     while new datapoint $(\mathbf{x}, \mathbf{y})$ comes do
5:         Predict its label using formula (1)
6:         if predicted label is NOT correct then
7:             send $(\mathbf{x}, \mathbf{y})$ to main server
8:         else
9:             update $\mathbf{C}$
10:        end if
11:        accuracy $\leftarrow$ GET_VALIDATION ACCURACY(
12:        if ) then$accuracy < min\_accuracy$
13:            Send updated $\mathbf{C}$ to main server
14:            Receive new $\mathbf{W}, \mathbf{B}, \mathbf{Z}$ from server
15:            $\mathbf{C} \leftarrow$ Receive weights of each prototype
16:        end if
17:    end while
18:    return $(\mathbf{Z}, \mathbf{B}, \mathbf{W})$
19: end procedure
```

In assigning weights to each prototype We follow the following algorithm -

Algorithm 5 Weight Assignment

```
1: procedure WEIGHT ASSIGNMENT($\mathbf{C}, \mathbf{B}, (\mathbf{x}, \mathbf{y})$)    ▷ Assigning weight to B for point $(\mathbf{x}, \mathbf{y})$
2:     Calculate similarity function or Kernel for each Prototype $K_i((\mathbf{x}, \mathbf{y}), B^i)$
3:     sum $\leftarrow \sum_{i=1}^{b} K_i((\mathbf{x}, \mathbf{y}), B^i)$
4:     for all prototypes $B^i$ do
5:         $\mathbf{C_i} \leftarrow \mathbf{C_i} + K_i((\mathbf{x}, \mathbf{y}), B^i)$
6:     end for
7:     return $\mathbf{C}$
8: end procedure
```

After sending the weights of prototype to server we restructure the data set so that Proto-NN can run easily. As currently Proto-NN is not designed for weighted loss functions. Thus, For restructuring -

- Round off the weights to nearest integer
- Repeat the i^{th} prototype $\left\lceil C_i \right\rceil$ times
- Add Gaussian error in all the repeated data points
- Make the final data points by combining these repeated points and misclassified points and ran Proto-NN on this

Currently our main aim was to reduce the amount of data to be send to server. The reason which led us to this algorithm was that if number of points can't be reduced to a significant level then we can try to send the data in reduced form. So, we send the prototypes as representatives of correctly classified points along with the mis classified points.

[Dimension conflict] Prototypes which are received by the micro-controller are in lower dimension so after restructuring prototypes which are still in lower dimension and the misclassified points should have to be of same dimension for combining both data. So, while sending the misclassified point we reduce its dimension using our dimensionality reduction matrix $\mathbf{W}$. Hence at main server we have the complete representation of data set in form of prototypes repetition and misclassified points, both of them in same dimensions.

Problem :

We were repeating the prototypes and adding Gaussian error so that we don't have any repeated point but still the random Gaussian error was making several points too close that they were same w.r.t. the floating point approximation of computer. Hence calculating the gradient over these points was diminishing to zero which throws the divide by zero exception. Hence, this method was completely discarded due to implementation reasons.

9 Future Work

There are still some scope of extension to our above algorithms, Online Proto-NN and Ballogorithm.

Online Proto-NN

- The hyper-parameter γ for the Gaussian kernel and other hyper-parameters can be learned also while learning matrices $\mathbf{Z}, \mathbf{B}, \mathbf{W}$ by extending our Alternating Optimization.
- Hard Thresholding can be done on $\mathbf{Z}, \mathbf{B}, \mathbf{W}$ to make them sparse, which will result in simpler calculations of gradient and reduced model size.

Ballogorithm

- Instead of hard assignment of the point to a ball we can shift to a soft version of assignment, such that for incoming points we will assign fraction to the nearby circles instead of only one circle.
- After choosing ball as a coreset option we can try choosing different types of coreset and implement the same type of algorithm like ballogorithm on them.
- Instead of sending the point itself, we can send it after projecting it to a lower dimension.

References

[1] Chirag Gupta , Arun Sai Suggala , Ankit Goyal , Harsha Vardhan Simhadri , Bhargavi Paranjape , Ashish Kumar , Saurabh Goyal , Raghavendra Udupa , Manik Varma , Prateek Jain
ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices *Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1331-1340, 2017.*
`http://proceedings.mlr.press/v70/gupta17a.html`

[2] Ashish Kumar, Saurabh Goyal, Manik Varma
Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things *Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1935-1944, 2017.*
`http://proceedings.mlr.press/v70/kumar17a.html`

[3] Wenjia Meng, Zonghua Gu, Ming Zhang, Zhaohui Wu
Two-Bit Networks for Deep Learning on Resource-Constrained Embedded Devices *arXiv:1701.00485 [cs.LG]*
`https://arxiv.org/abs/1701.00485`

[4] Matt J. Kusner, Stephen Tyree, Kilian Weinberger , Kunal Agrawal
Stochastic Neighbor Compression*In ICML, 2014b.*
`http://mkusner.github.io/#code`

[5] Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon
Fast Prediction for Large-Scale Kernel Machines *Advances in Neural Information Processing Systems 27 (NIPS 2014)*
`https://papers.nips.cc/paper/5481-fast-prediction-for-large-scale-kernel-machines`

[6] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of Geometric Approximation via Coresets
`http://www.math.tau.ac.il/ michas/core-survey.pdf`

[7] Blumensath, Thomas and Davies, Mike E.
Iterative thresholding for sparse approximations.*Journal of Fourier Analysis and Applications, 14:629–654, 2008.*
`https://arxiv.org/pdf/0805.0510.pdf`

[8] Jain, Prateek, Tewari, Ambuj, and Kar, Purushottam. On iterative hard thresholding methods for high-dimensional m-estimation. *In NIPS, pp. 685–693, 2014*
`https://arxiv.org/abs/1410.5137`