Neural Networks

Assignment - 1

March 14, 2018

Abstract

Many different approaches have been implemented in the filed of Machine-Learning. The following sheet is an endeavor of addressing some ideas about the implementation of different machine learning and classification algorithms. All the above attempt was applied on MNIST dataset (simple-version) consists of 1707 images in the train-set and 1000 in test-set, each one represented by a 256-length array. Conclusions and remarkable points are discussed and analyzed.

1 Analyze Distances Between Matrices

The purpose of the first task is to gain a general idea about clouds of points in highly dimensional spaces. More precisely, in this part of the assignment we will introduce some measures about the cloud points of each digit, in order to start building a simple algorithm for classifying hand-written digits in the following tasks. For each digit d, d = 0, 1, . . . , 9, we will consider a cloud of points in 256 dimensional space, C_d which consists of all training images (vectors) that represent d.

In particular for each cloud point of the 10 digits we will calculate:

- Center C_i : Vector of means for each cloud.
- Radius r_i : The highest distance from the center between all points for the corresponding cloud.
- Number of points n_i : How many images(vectors) are contained in each class.
- Distances between centers $dist_{ij} = dist(c_i, c_j)$: Distances between all classes' centers .

All the above measurements are required for this classification problem. To get started, the center for each cloud is represented by the vector of means of all instances corresponding to each class. So, we have 10 centers(K=10), one for each digit, and each one of them is the averaged 16x16 image(256-dimensional vector) resulted from the computation mentioned above.

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
Radius	15.89	9.48	14.16	14.74	14.53	14.45	14.03	14.90	13.70	16.13
Points	319	252	202	131	122	88	151	166	144	132

Table 1: Radius and number of points for each cloud

As it shown in the above table, the highest number of images belongs to the cloud of digit 0 and the lower number of images belongs to the cloud for digit 5. According to the results of the Distance Matrix between the ten cloud centers, it is obvious that the most dissimilar digits are: '0' and '1', which is reasonable because of their shape, and the most similar are: '9' and '7' with distances 14.449 and 5.426 respectively. This leads us to conclude that digits whose centers are far apart, would be easily discriminated while the opposite would occur, for digits whose centers have small distance values. Furthermore, the average distance value is 8.91 and the median is 8.87. There are 21 different pairs of digits with distance higher than the average and 24 pairs with lower. So, taking into account that all the distances fluctuate near the mean distance, the pairs with distances above the average are more easily separable than the ones located below the average value. Moreover, it has to be mentioned that the accuracy of the classifier depends also on the radius of each digit(class), thus the latter influences the size of the cloud.

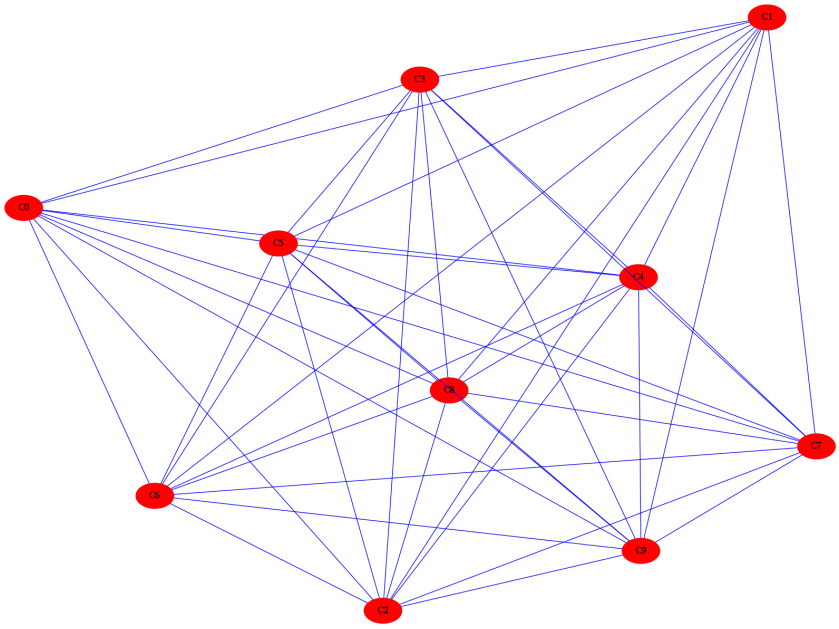

Figure 1: Distance matrix represented as network

The above plot is a network representation of the Distance Matrix. The nodes(red) of this network represent the different digit classes and the edges(blue) are the actual distances between the class-centers. It can be easily seen that nodes 9 and 7 are very close due to their small distance and classes 0 and 1 are far apart. The following figure visualizes the graphical/image representation of digits by the vector of centers.

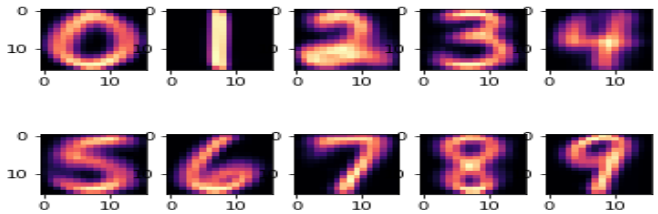

Figure 2: MNIST-digits visualized by the centers

2 Simple classifier implementation

2.1 Algorithm

In this section a simple distance-based classifier will be discussed. This algorithm is one of the most naive approaches used for classification due to the usage of distances as measurement. The advantage is that it is very easy to understand it as basic method for classification in the filed of machine learning and it requires no prior knowledge of statistics because it is a simple distance method. The disadvantages are basically that it is computationally expensive especially for big datasets and that it can not handle complex classification cases. The idea behind it is simplistic and reasonable: search for the closest class-center between all instances in the dataset and classify each one according to the smallest distance. More precisely, first calculate all distances between the class-centers and the images(points) from the training-set and consequently classify each image into the class with the smaller distance. After the classification stage, evaluation is achieved using confusion matrices where the classifier results are represented in a symmetric table with columns as the truth labels and rows as the predicted. Confusion matrix describes the performance of a classification model when the true values of classes are known.

2.2 Implementation on MNIST

The implementation of this algorithm on MNIST dataset is made by calculating all distances between the 1707 different images of the train set and the centers for the 10 classes. Centers were calculated using only the train-set and they used as the estimation for the classification on the test-set. The distances are measured with the Euclidean distance using the following formula.

$$Euclidean(x, y) = \sqrt{\sum (x_i - y_j)^2}$$

Consequently, each image was classified into the class with the minimum distance. Afterwards, for the evaluation of this algorithm, the following confusion matrices were produced.

	0	1	2	3	4	5	6	7	8	9
0	271	0	3	0	0	3	10	0	1	0
1	0	252	0	0	8	0	4	4	2	3
2	0	0	167	2	1	2	5	0	1	0
3	0	0	9	120	0	3	0	0	10	1
4	2	0	9	1	95	4	2	2	2	10
5	4	0	1	3	0	67	0	2	3	0
6	36	0	3	0	3	3	129	0	1	0
7	0	0	4	1	0	1	0	140	0	6
8	6	0	6	3	0	2	1	1	121	0
9	0	0	0	1	15	3	0	17	3	112

Table 2: Confusion Matrix for Train-set

The table above provides information about misclassified instances of the training set by representing the actual labels(classes) in the rows and the predicted ones in columns. The diagonal of this matrix represents the number of True-positive cases due to symmetry. It can be easily observed that many instances of digit six were classified in zero and vise versa, which is reasonable, probably because of the their similar shape(curvy) but also according to the Distance Matrix, digits 0 & 6 are not so far apart with distance counted by 8.15. Moreover, digits 9 & 7 seem to be difficult to be classified correctly, probably because of their small distance counted by 5.42 but also due to their similar shape. These conclusions could be verified by casting an eye over to the graphical 2-D representation in Figure:1 which visualizes the actual distances between all different centers(classes). It can be observed that node C_0 is close to C_6 as node C_9 is very closed to C_7, so the assumptions made in Task 1 seem to be valid.

	0	1	2	3	4	5	6	7	8	9
0	178	0	2	3	1	3	7	0	3	0
1	0	120	0	0	3	0	0	2	2	5
2	3	0	69	3	3	0	2	1	0	0
3	2	0	6	61	0	6	0	0	6	0
4	4	0	8	1	69	3	2	5	3	8
5	2	0	1	8	0	38	1	0	3	0
6	23	1	0	0	1	1	78	0	0	0
7	1	0	2	0	1	0	0	50	0	5
8	10	0	13	1	0	0	0	0	73	2
9	1	0	0	2	8	4	0	6	2	68

Table 3: Confusion Matrix for Test-set

The above matrix is the confusion-matrix for the test-set where the final performance of the classifier can be evaluated. As observed in the table above, it is obvious that many cases of digit six from the test-set, were classified in label zero while many cases of digit nine were classified not-correctly in label four or seven. This is quite a reasonable fact due to the conclusions that were made from the results of the Table:2 and the initial belief of which digits are more difficult to be classified correctly based on the Distance Matrix.

2.3 Other Distances

All previous results were obtained from the classification by using the Euclidean-Distance. There are many other distances which could be used in such problems. For instance Mahalanobis distance is a very useful measurement for classification and clustering problems. The big advantage is that it handles better the data structure hence it takes into account the covariance structure of the data.

$$D_m(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}$$

where $\vec{x} = (x_1, x_2, ..., x_n)^T$, $\vec{y} = (y_1, y_2, ..., y_n)^T$ and S is the covariance matrix of $\vec{x}\&\vec{y}$. We can define Mahalanobis as a generalization of Euclidean distance for covariance matrix S, when S $\neq$ Identical. Furthermore, it can be observed that Mahalanobis distance is the exponential part of the Multivariate-Gaussian-Distribution. The following table illustrates the accuracies in the two data-sets using different distances.

	TrainSet	TestSet
Euclidean	0.86	0.80
Cosine	0.86	0.79
Mahalanobis	0.96	0.72
Sqeuclidean	0.86	0.80
Cityblock	0.76	0.72

Table 4: Accuracy table

As it can be easily seen, the best accuracy in train-set is produced by Mahalanobis, counted by 0.96, which is in fact reasonable because the centers were calculated from the train set. The covariance matrix S describes dependence relations between $\vec{x}$ and $\vec{centers}$, so this is why it produces better results in the train-set classification. However, test-set accuracy with the same distance seems to be low due to the same reason. This could drive us to conclude that Mahalanobis distance is a very powerful tool but probably for clustering purposes and not for supervised prediction/classification tasks. The best result in test-set is given by the Euclidean-distance counted by 0.80. The rest distances seem to be inappropriate for this image-classification task.

3 Naive-Bayes classifier

3.1 Bayesian classifier approach

During the past years, many different approaches for classification have been developed in order to deal with various discrimination problems for instance text, image and even sound classification tasks. Naive-Bayes classifier is a discrimination approach based on probabilistic classifiers which uses the Bayes-Theorem in order to classify input values given some features and some prior probabilities. This approach is based on the strong assumption of independence of input data. The idea behind this algorithm is to define the prior probabilities of how the input data is distributed, combine them with the likelihood probability-function and consequently, end up with the posterior probability which indicates the classification label. Naive-Bayes classifier is easy to implement and could be proved really practical for discrimination problems like MNIST dataset, thus it is based on the maximum likelihood concept. Furthermore, it does not require big computational power due to the small amount of probabilities need to be calculated.

3.2 Bayes rule classifier on MNIST

Before proceeding to the implementation on MNIST data, it is essential to initialize the basic probabilistic formulas that will be used, and declare the chosen feature. Certainly, there are many different approaches for the procedure of feature extraction, thus discrimination criteria could be very abstract. The one which is chosen for MNIST data, given that we are only concerned about digits 5 and 7, is based on how many black pixels each image has. In other words, it is expected that images which represent digit 5 will have more dark pixels than the ones representing digit 7. Before continuing on the classification algorithm, it would be convenient to make some remarks on how the feature discriminates the training data.

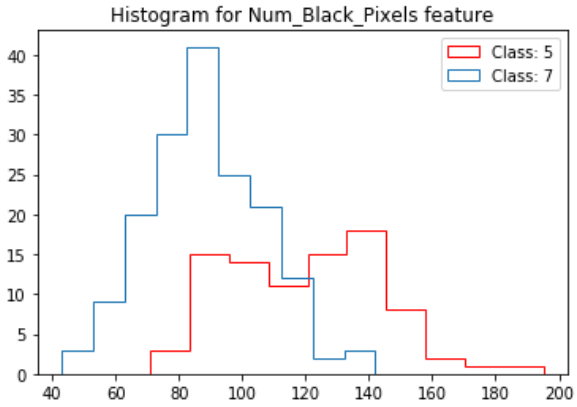

Figure 3: Histogram

The above plot illustrates the quality of the chosen feature in terms of discrimination. Thus, the feature is considered to be a good classifier if the two histograms would be far apart while an infective one, if there is overlapping. In this case, the quality of the classifier is moderate hence there is overlapping between the two plots but there is also some area where the discrimination will be easier. Therefore, this feature will be defined as X. The prior probabilities that reflect the probabilities of classes 5 and 7 given by:

$$p(C_5) = \frac{n_5}{N} = 0.35$$
$$p(C_7) = \frac{n_7}{N} = 0.65$$

where $n_5 = 88$, $n_7 = 166$ and N=256. Given the above calculations, and using Bayes Theorem the posterior probability for each class given the value of feature X, is easily

obtained using the formula below:

$$p(C_k \mid X) = \frac{p(C_k)p(X \mid C_k)}{p(X)} \propto p(C_k)p(X \mid C_k) \qquad (1)$$

So, after computing the two posterior probabilities for each class respectively, the oncoming observations will be classified in the class with the highest posterior probability. The class-conditional probability, $p(X \mid C_k)$, is obtained according to the following procedure:

- Calculate X for all 5's and 7's
- Construct j classes(bins) according to feature's values
- Calculate the frequencies of each label(5's, 7's) in each bin, n_j, j=1,...,J
- Calculate the sum of all frequencies for C_5 and C_7, N_i, i = 5,7

The above procedure, leads to the desired discretization of the chosen feature X which is computed as $p(X \mid C_k) = \frac{n_j}{N_i}$ and will be plugged in equation (1) ,ending up to the posterior probabilities. After applying the Naive-Bayes algorithm on both train and test datasets we came up with the following results:

Accuracy	
Train set	0.77
Test set	0.75

Table 5: Accuracy Table

The results shown above, indicate that the feature which used for this classification task, was not efficient in terms of accuracy. Counting the amount of black pixels may be misleading and has a strong impact on the amount of misclassified digits. Thus, it is a quite naive and imprecise discrimination feature for the specific image analysis. Without a doubt, there is a variety of features which probably lead to better results such as the ratio between the number of black pixels in the upper and down half of the image. It should be mentioned that the number of bins could probably affect the accuracy results thus, splitting the feature in more bins may facilitate the discrimination.

4 Signle-Layer multi-class perceptron

In this section the algorithm of a simple perceptron will be introduced. This approach is based on linear functions which discriminate objects following a network representation. In this specific task, a single layer multiclass perceptron needs to be implemented, which means that the network includes only one layer with hidden nodes and classifies the objects with linear functions. As mentioned before, MNIST train-data contains 1707 instances of digits each one represented by a 256-length array, so all the equations are in the R^{256} dimensional space. The following schema illustrates the network for the MNIST classification.

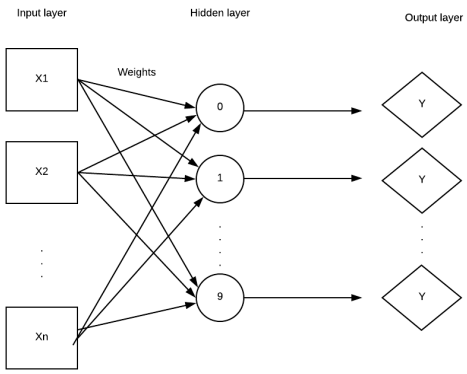

Figure 4: Perceptron Schema

According to the schema it can be observed that all input values are connected to all nodes. The input vector is connected to the hidden layer through a matrix of randomly initialized numbers(weights) all input values, communicate to each hidden-node with a random weight. Nodes in this network represent the ten different classes(labels) and each one of them produces the classification decision. The network is able to train(tune) the weights using the following perceptron learning rule.

Figure 5: Perceptron learning rule

```
W = initialize weights randomly
WHILE (misclassified = True) do:
    select the misclassified example
    W_new = W_old + (learning_rate)*d*x
END-WHILE
```

Learning-rate (η) defines the speed (step size) of the update procedure and usually takes small values (e.g 0.01). It has to be mentioned, that as the learning-rate value gets higher, the algorithm converges faster, however small learning rate values produce more robust and accurate predictions. Parameter "d" defines the misclassification occurrence during the update loop and takes values (-1 & 1). If x is misclassified and d=1 then the value of WX has to be bigger while d=-1 the corresponding WX has to be smaller.

In this specific implementation of single layer multiclass perceptron on MNIST data we followed the below procedure:

1. Represent the actual labels(ground-truth) of data by a hot-vector of length 10 filled with 10-1 zeros and a "1" in the position of the corresponding label. e.g label 7=[0,0,0,0,0,0,0,1,0,0]

2. Add a single column of ones in the input dataset X for the bias. X is now shaped (1707 , 256+1).

3. Initialize randomly the matrix W, representing the weights, with shape (256+1 , 10).

4. Multiply the two matrices X*W and result in matrix Z. Now Z contains 1707 rows (one for each image) and 10 columns indicating the 10 different labels.

5. The predicted label of each image in Z arises from the maximum value of each row thus it indicates the location of the closest label prediction.

6. Construct a hot-vector for the outputs, as in step_1, witch is the same shape as matrix Z and indicates the predicted labels with "1" in the position of the maximum value in each row.

7. Calculate the error by subtracting the two hot-vectors, Error = Truth - Prediction. Now Error is a (1707×10) matrix with 1 at the position(column) of actual label and -1 at the misclassified. If there are no misclassified cases then the Error tensor includes only zeros.

8. Use the update while loop from Figure:5 where d=$Error^T X$ in order to train the weights and extract their optimal values.

The mathematical representation of the linear functions used in perceptron is the following:

$$Z = W_0 + \sum W_i X_i = W_0 + W_1 X_1 + W_2 X_2 + ... + W_n X_n$$

The algorithm converges when it reaches accuracy=1 which means that there are no misclassified objects. This also means that MNIST images are linearly separable in the R^{265} dimensional space and there are hyperplanes which separates them. The graph below provides information about the accuracy and error fluctuations over the iterations until convergence.

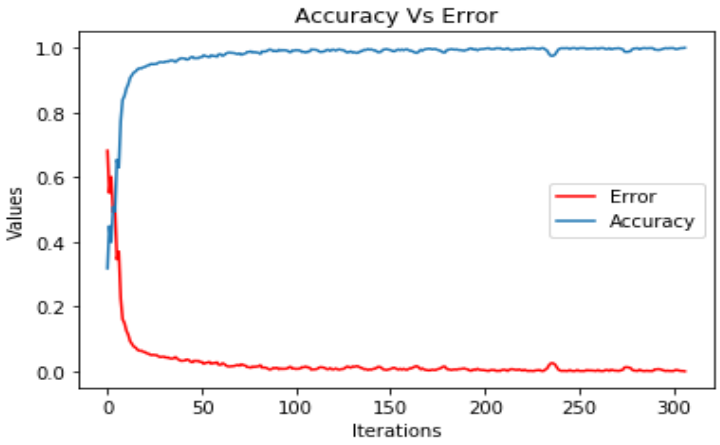

Figure 6: Accuracy-Error over epochs

After the training procedure, the trained parameters(weights) have to be tested. The evaluation is made by multiplying the trained matrix of weights with the test data and calculating the accuracy. The evaluation produced the below results.

Accuracy	
Train-set	1.00
Test-set	0.87

Table 6: Perceptron-Accuracy

5 Gradient Descent Algorithm

The purpose of this task is to build a simple three-layer network for XOR data, train it using different weights and successfully predict the logical port XOR output. However, the procedure of training can be proven fruitless if the weights keep stable or randomly change their values. We will introduce the Gradient Descent algorithm applied on Mean Square Error function for the XOR data.

- *Gradient Descent* : Before describing the gradient descent method it is important to briefly mention the concept of the mean square error which will be used in the aftermath. When it comes to evaluation and training process of a neural network it is crucial to initialize a cost function which will quantify the error of the network in terms of prediction. One of the most common cost functions is

the Mean Square Error which measures the distance between the predicted and the desired output takes the average of the squares. Using this cost function, gradient descent generates a rule -delta rule- for updating the initialized weights into new "corrected" weights taking into account the previously computed components of the network. The latter is achieved using partial derivatives over the weights reflecting the impact of changes in each one of them over the network.

- *XOR data* : XOR gate is a digital logic function consists of two or more inputs, and outputs 1("True") when the two inputs are different and 0("False") when the two inputs are the same. In the general case, if the number of "Trues" in the input is odd the XOR gate outputs 1("True"). It should be mentioned that the XOR problem cannot be solved by a single perceptron, hence the input data are not linearly separable and they require more complicated rules than hyperplanes. In this task we will limit in the two inputs case. The desired XOR output is expected to have the structure of the below table.

Input		Output
x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Table 7: XOR gate structure

The perceptron consists of two input nodes, one hidden layer with two nodes and one output. Both nodes of the hidden layer have three inputs,taking into account the bias, and the same applies for the output node.

The mathematical representations for each node which reflect the combinations of inputs weight and biases are given in the formulas below:

$$\begin{aligned} node_1 &= x_1 w_1 + x_2 w_2 + bias \cdot w_3 \\ node_2 &= x_1 w_4 + x_2 w_5 + bias \cdot w_6 \\ output &= node_1 \cdot w_7 + node_2 \cdot w_8 + bias \cdot w_9 \end{aligned}$$

where, $bias$=1 and w_i the weights. Results of the network for each non-input node are converted through the sigmoid function driving us to the final output. The aforementioned process is implemented via the `xor_net()` function.

Afterwards, the predicted results from the `xor_net()` function are going to be plugged in the formula of the mean square error in order to calculate the distance between the calculated output and the desired output shown in Table:7. This leads us to construct the `mse()` function which takes as input the initial weights, so the error is calculated by:

$$MSE = \frac{(predicted - desired)^2}{2}$$

which returns an error of 40% for the initialized weights on the four input vectors. The purpose of the training process is to minimize the error of the prediction and reach the point where the network will correctly predict the XOR output. In order to accomplish that, the use of gradient descent algorithm is straight-forward and by constructing the `grdmse()` function we can produce the partial derivatives. As a result, `grdmse()`, using the mean square error function, produces nine partial derivatives for each weight.

XOR network updating rule:

$$weigths_{new} = weidhts_{old} - \eta \cdot grdmse(weights)$$

As mentioned before, the weights initialization is a random draw from the standard normal distribution multiplied by 0.01 and the learning rate η is equal to 0.01. The training of the network contains experimentation with different values of η and different initialization for the weights. The results below illustrate different values of MSE for all of these experiments and lead us to the following conclusions.

- **Standard Normal weights :** The algorithm begins the train process fluctuates between 1 and 2 misclassified outputs and ends up with MSE value equal to 0.13.

η	0.01	0.1	1
Epoch	978930	97898	9765

Table 8: Normal Weights

- **Random Uniform weights:** The algorithm begins training the data having 2 misclassified outputs and stops in MSE equal to 0.13.

η	0.01	0.1	1
Epoch	69684	6971	699T

Table 9: Uniform Weights

The following graphs represent the error fluctuations during the training for *sigmoid* on Figure:7 and *tanh* on Figure:8, activation functions and learning rate equals to 1.

(a) Uniform initialized weights (b) Gaussian initialized weights

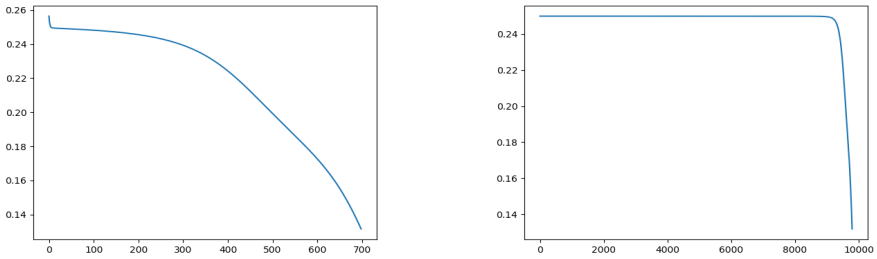

Figure 7: Error vs Epoch with Sigmoid function , η=1

(a) Uniform initialized weights (b) Gaussian initialized weights

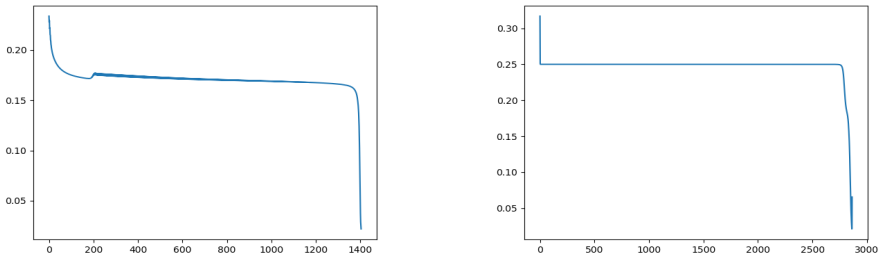

Figure 8: Error vs Epoch with Tan function , η=1

Final Remarks: Taking into account the outcomes of the training procedure, we can conclude that as the learning rate increases, the number of epochs required for the training process decreases, thus the algorithm converges faster. This fact, applies on both activation functions, *sigmoid* and *tanh*, however the former produces better results in terms of time-convergence. In particular, in Figure:8 it is observed, that there is a long stability in the error function for both kinds of initialized weights. However, in the case of normal weights the error drops more sharply while in the uniform case the curve drops smoothly. Furthermore, initialization of weights from the uniform distribution leads to more rapid convergence than the Gaussian initialization in both cases of *sigmoid* and *tanh* activation functions.