ECE/CS 559 Neural Networks, Fall 2017 - Homeworks #5 and #6
Due: 11/06/2017, the end of class.

Erdem Koyuncu

This homework is worth two homeworks. Half of the total grade will be your Homework #5 grade. The other half will be your Homework #6 grade. All the notes in the beginning of Homework #1 apply. I want everyone to pay special attention to this homework. *It will not be graded as leniently as the previous homeworks.* There are two computer experiments, both of which will use backpropagation. Therefore, I would suggest you to first have a working and well-tested backpropagation implementation. The problems will take some time so do not leave them to the last day! Make sure to include the computer codes in your report.

1. **(50pts)** In this computer project, we will use a neural network for curve fitting.
   (a) Draw $n = 300$ real numbers uniformly at random on $[0, 1]$, call them $x_1, \ldots, x_n$.
   (b) Draw n real numbers uniformly at random on $[-\frac{1}{10}, \frac{1}{10}]$, call them $\nu_1, \ldots, \nu_n$.
   (c) Let $d_i = \sin(20x_i) + 3x_i + \nu_i$, $i = 1, \ldots, n$. Plot the points (x_i, d_i), $i = 1, \ldots, n$.
   We will consider a $1 \times N \times 1$ neural network with one input, $N = 24$ hidden neurons, and 1 output neuron. The network will thus have $3N + 1$ weights including biases. Let $\mathbf{w}$ denote the vector of all these $3N + 1$ weights. The output neuron will use the activation function $\phi(v) = v$; all other neurons will use the activation function $\phi(v) = \tanh v$. Given input x, we use the notation $f(x, \mathbf{w})$ to represent the network output.
   (d) Use the backpropagation algorithm with online learning to find the optimal weights/network that minimize the mean-squared error (MSE) $\frac{1}{n}\sum_{i=1}^{n}(d_i - f(x_i, \mathbf{w}))^2$. Use some η of your choice. Plot the number of epochs vs the MSE in the backpropagation algorithm.
   Hint: As discussed in class, for a given fixed η, the algorithm may not always result in a monotonically decreasing MSE (the descent may overshoot the locally optimal point). You may have to modify the gradient descent algorithm in such a way that you decrease η (e.g. via $\eta \leftarrow 0.9\eta$) whenever you detect that the MSE has increased. Also, beginning with a very large η may result in an immediate divergence of the weights.
   (e) Let us call the weights resulting from the backpropagation algorithm (when it converges) as $\mathbf{w}_0$. The curve $(x, f(x, \mathbf{w}_0))$, $x \in [0, 1]$ will then be a fit to the points (x_i, d_i), $i = 1, \ldots, n$. Plot the curve $f(x, \mathbf{w}_0)$ as x ranges from 0 to 1 on top of the plot of points in (c). The fit should be a "good" fit.
   (f) Your report should include a pseudocode of your training algorithm including all update equations written out explicitly (similar to what I had done in the second question of your second homework). The pseudocode should be written in such a way that anyone would be able to implement your algorithm without knowing anything about neural networks.

As a preparation for the next problem, we review the formulation of a general classification problem. Suppose that we have a set $\mathcal{C}$ of classes. For example, $\mathcal{C} = \{0, 1, \ldots, 9\}$ for the digit classification problem, or we could have $\mathcal{C} = \{\text{cat}, \text{dog}, \text{dragon}\}$ for a problem where we wish to determine whether a given picture contains a cat, dog, or a dragon. In the supervised learning setup, we have a training set $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with the corresponding predetermined classes/labels $c_1, \ldots, c_n \in \mathcal{C}$. We wish to design a neural network that provides the hopefully-correct class of any given input $\mathbf{x}$.

Suppose that you consider a network with m output neurons to solve the classification problem described above. The first thing to do is to assign, for each class $i \in \mathcal{C}$, a representative output vector $\mathbf{d}_i \in \mathbb{R}^m$. Let $\mathbf{D} = \{\mathbf{d}_i : i \in \mathcal{C}\}$ denote the set of all representative output vectors of classes. For example, for the digit classification problem in Homework 2, we considered $m = 10$ output neurons with representative output vectors $\mathbf{d}_0 = [1\,0\,0\,\cdots\,0]^T$, $\mathbf{d}_1 = [0\,1\,0\cdots\,0]^T$, and so on. In particular, for the training sample $\mathbf{x}_i$, the class label is c_i, so that the desired output vector would be $\mathbf{d}_{c_i}$.

The next thing to do is to consider an energy function of the form $\frac{1}{n}\sum_{i=1}^{n} D(\mathbf{d}_{c_i}, f(\mathbf{x}_i, \mathbf{w}))$. Here,

- i is the training sample index,
- n is the number of training samples,
- $\mathbf{d}_{c_i} \in \mathbf{C}$ is the desired output for training sample i,
- $f(\mathbf{x}_i, \mathbf{w})$ is the network output given input (training sample) $\mathbf{x}_i$ and weights $\mathbf{w}$,
- $D(\cdot, \cdot)$ is some arbitrary distance function (metric). In particular, we use the function $D(\mathbf{d}_{c_i}, f(\mathbf{x}_i, \mathbf{w}))$ to measure how far off the network output $f(\mathbf{x}_i, \mathbf{w})$ is from the desired output $\mathbf{d}_{c_i}$. Typically, we choose $D(\mathbf{z}_0, \mathbf{z}_1) = \|\mathbf{z}_0 - \mathbf{z}_1\|^2$.

The next thing to do (which is the harder part) is to use the backpropagation algorithm to find some optimal weights, say $\mathbf{w}_0$, that minimize $\frac{1}{n}\sum_{i=1}^{n} D(\mathbf{d}_{c_i}, f(\mathbf{x}_i, \mathbf{w}))$. Having done this, the question is now how to determine the class of some arbitrary input pattern $\mathbf{x}$? Intuitively, we should choose the class whose representative output vector is closest to the output provided by $\mathbf{x}$ according to distance function $D(\cdot, \cdot)$. In other words, given some arbitrary input pattern $\mathbf{x}$ and weights $\mathbf{w}_0$ (or any weights in general), we first calculate the network output $f(\mathbf{x}, \mathbf{w}_0)$. We can then estimate the class of $\mathbf{x}$ as

$$\arg\min_{i \in \mathcal{C}} D(\mathbf{d}_i, f(\mathbf{x}, \mathbf{w}_0)).$$

In other words, the estimated class i for input pattern $\mathbf{x}$ should minimize $D(\mathbf{d}_i, f(\mathbf{x}, \mathbf{w}_0))$.

---

2. **(150pts)** In this computer project, we will design a neural network for digit classification using the backpropagation algorithm (see the notes above). You should use the MNIST data set (see Homework 2 for details) that consists of 60000 training images and 10000 test images. **The training set should only be used for training, and the test set should only be used for testing.** Your final report should include the following:

   - The details of your architecture/design, including
     - Your network topology, i.e. how many layers, how many neurons in each layer, etc. (Obviously you will have 784 neurons in the input layer).
     - The way you represented digits $0, \ldots, 9$ in the output layer. For example, one may use the same setup as in Homework 2, 10 output neurons, with $[1\ 0\ \cdots 0]$ representing a 0, $[0\ 1\ 0 \cdots 0]$ representing a 1 and so on. Another person may have just one output neuron with an output of 0 representing a 0, an output of 1 representing a 1, and so on.
     - Neuron activation functions, learning rates for each neuron, and any dynamic update of learning rates (as explained in the question above) if it exists.
     - The energy/distance functions of your choice.
     - Any other tricks such as regularization, dropout, momentum method, etc.
   - The reasons as to why you chose a particular hyperparameter the way it is (e.g. why did you choose 100 hidden neurons, but not 50, why do you have 1 hidden layer but not 2?)
   - Your design process. It is unlikely that the first network you train will actually work. Write about your design process, your failures, together with your comments on why do you think a particular approach failed (e.g. I began with $\eta = 100$ and the algorithm just diverged, η was just too large).
   - A pseudocode of your final algorithm as described in (f) of Question 1 above.
   - A plot that shows epoch number vs the number of classification errors on both training and test images. Another plot that shows the epoch number vs the energy on both training and test images. Your test set should include all 10000 test images.
   - As usual, you should use your own code without any help from any external neural network/machine learning algorithms.

   Your work will be graded mainly based on the depth and the quality of your report. Your final network should also achieve a decent success rate. You should be able to achieve around 95% success rate on the test set (all 10000 images). Very poor network performance will also result in a low grade.

Olivo Iacopo - Homework 5

The main task of tis homework is to implement a neural network in order to minimize the function

$$MSE=\frac{1}{n}\sum_{i=1}^{n}(\ d_i-f(x_i,w))^2$$

Preliminary calculation

The matlab sript has been designed in order to use parametric network topology on a fixed two layer setup.

- *Parameters selection:*

The parameters used from now on are:

- I: Indicates the size of the input
- N: Indicates the number of neurons in the first layer of the network
- O: Indicates the number of neurons present in the output layer of the network.

The network will thus have different elements:

- X is the input vector
- W1 and W2 are the set of weights and biases which are related respectively to the first and second layer
- V1 and V2 are the induced local field.
- $\varphi_1(v)=\tanh(v)$ And $\varphi_2(v)=v$ are the two activation functions.
- Y and Z are the output at the first and second layer. Y will include the bias term as first element

- *Update equation calculation*

In order to train the network we will need to apply newton's method. In order ti calculate $\nabla \mathrm{E}$, we need to use the backpropagation algorithm. In order to calculate the update equations we can use the formula developed during the lesson, for $MSE=\frac{1}{2}\sum_{i=1}^{n}(\ d_i-f(x_i,w))^2$ slightly modified. This will result in the update formulas for the two layers :

- $$\frac{\vartheta \mathrm{E}(w)}{\vartheta w_{j,k}}=-x_k\sum_{i=1}^{O}((z_i-d_i)\varphi_2'(V2_i)W2_{i,j})\varphi_1'(V1_j)$$
  The element-wise equation has been converted in matrix form in order to implement a faster update mechanism in the matlab script:
  $$W1=W1+\frac{2}{n}\eta W2_{tmp}^{T}*\varphi'_2(V2)\circ(D-Z)\circ(\varphi'_1(V1))*x_i^T$$
  We need to note the use of $W2_{tmp}$, which is nothing more than W2 without the bias terms. This is used in order to exclude the bias from the calculation of the backpropagation input coming from the second layer.
- While for W2 we have
  $$\frac{\vartheta \mathrm{E}(w)}{\vartheta w_{i,j}}=-\varphi'_2(V2_i)\varphi_1(V1_j)(z_i-d_i)$$
  Which in matrix form translates to:
  $$W2=W2+\frac{2}{n}\eta\varphi'_2(V2)\circ(D-Z)*Y^T$$

- *Initial weights*

In order to start the training we will need some random weights to start. By looking at the different transfer function present in the different layers, we decide to initialize weights W1 and W2 in different ways:

- W1 is initialized with a uniform random distribution between -1 and 1
- W2 is initialized with a uniform random distribution between 0 and 1

Since we are using random numbers, we do not want to have some zero values. They are hence removed and replaced with some static values.

- *Damping parameter*

Sooner or later, MSE will start to diverge due to the excessive dimension of eta. In order to avoid this we want to implement a mechanism to decreases the learning parameter.

This can easily done by $\eta = 0.9\eta$ in the moment the MSE in the current epoch is equal or bigger than the one in the precedent one.

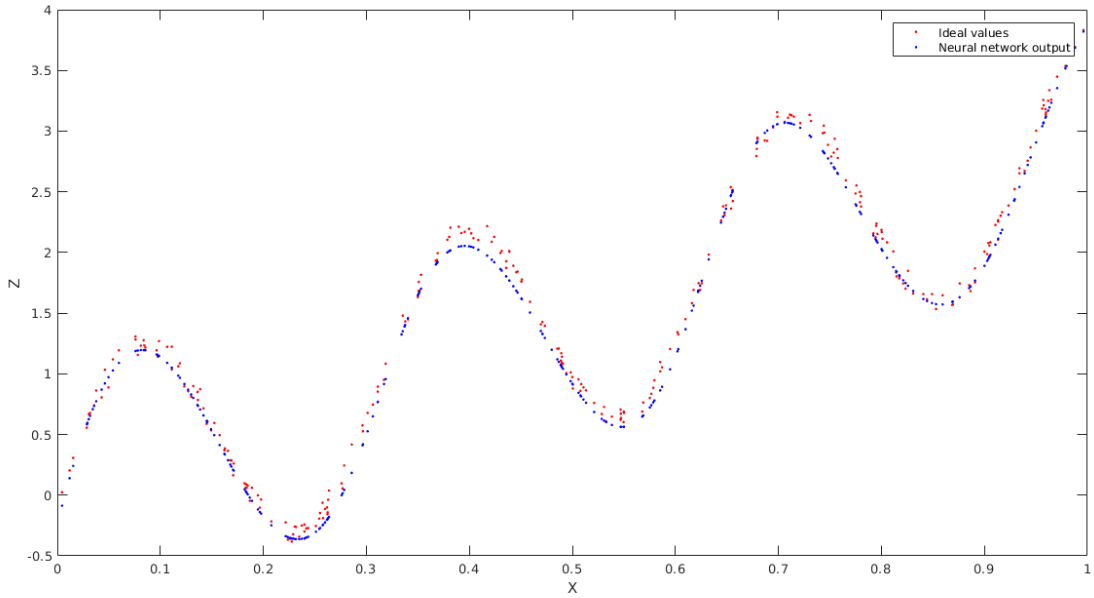
Pseudo-code

After calculating the needed update formulas we are ready to write the script in order to train the network:

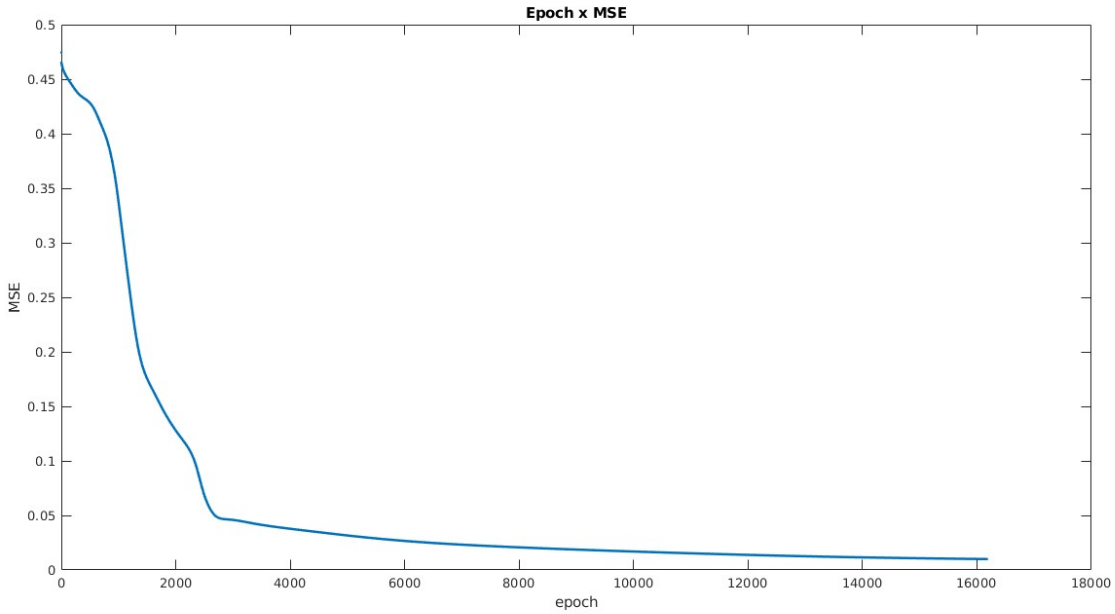
1. Initialize all the needed data
    1. Network topology
    2. Eta
    3. Stopping condition
    4. Train set and test set size
2. Setup the problem
    1. Generate n random elements
    2. Create a set D of desired output, by following the function provided in the homework text
    3. Initialize weights with the values discussed
3. Problem resolution
    1. For each input element:
        1. Calculate V1
           V1 = W1*X(:,in)
        2. Calculate Y
           Y = [1 ; phi1(V1)]
           Note that a trailing one has been added in order to be used as a bias input for the next layer
        3. Calculate V2
           V2 = W2*Y
        4. Calculate Z
           Z = phi2(V2)
        5. Calculate the temporary W2 matrix
           W2tmp = W2(:,2:end)
        6. Update W1 with the formula described before
           W1 = W1 + 2/n*eta * (W2tmp' *(dephi2(V2).*(D(:,in)-Z))).*dephi1(V1) * X(:,in)'
        7. Update W2
           W2 = W2 + 2/n*eta*(dephi2(V2).*(D(:,in)-Z))* Y'
    2. Calculate for both training and test
        1. For each input
            1. Calculate all the coefficients as described in 3.1 (without updating the weights)
            2. MSE = MSE + sum((D-Z).^2)
        2. MSE = MSE/n

3. Create a small log file to keep track of the data
4. Check for the exit condition
    1. If the condition is met, exit the loop
    2. Otherwise repeat from 3.
4. Plot the required graphs

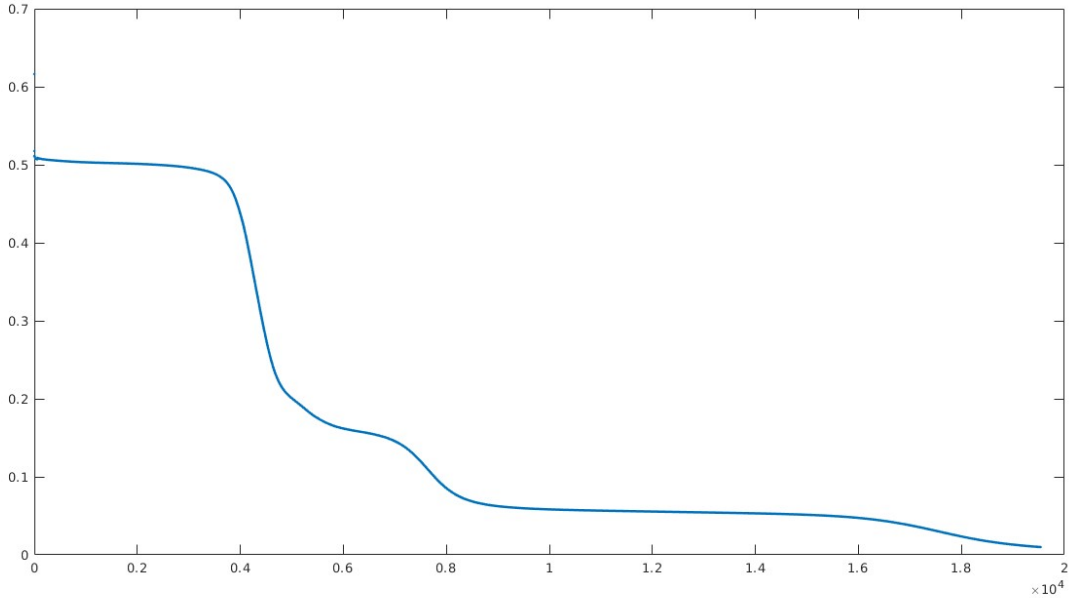
Results



Before being able to use the network we need to make some tries in order to find the correct training parameter for the given problem. After a few tentatives, $\eta = 1.5$ results in a satisfying parameter both in speed and precision. Our exit condition will be when MSE<1%.
The resulting network is able to perform relatively well, in fact the resulting network manages to have MSE=0.1 in a fair amount of time.



The choice of the exit condition is very important. In fact we could think about exiting the function when the learning parameter is not decreasing anymore (hence the derivative is near to 0) or, in order to use more sophisticated methods, when the norm of the gradient is near to 0 (which is exactly the same thing as having 0 derivative). These methods cannot be implemented since it is relatively common for the MSE to assume a stepwise form, making these methods unpractical.



In this case a function using the gradient norm size as stopping condition would stop after few epochs with a very high MSE.

```
%% Preliminary operations to clear the environment
clc
close all
clear all
%% Definitions
n = 300;
% Our network has size IxNxO where O is the number of output neurons (1)
% and I the number of inputs (1)
I = 1;
N = 24;
O = 1;
eta = 1.5
epsilon = 0.01;
%% Problem setup
X = ones(1,n);
for i = 1:1:I
    X = [X; rand(1,n)];
end
V = (rand(n, 1) - 0.5)/5;
D = sin(20 * X(2, :)) + 3*X(2, :) + V(:, 1)';
figure(1)
plot(X(2, :), D, '.r'), hold on
% Weights matrix has shape:
    % Rows = i (neuron of destination)
    % Columns:
        % 1: bias
        % 2: weight (neuron of source)
% Elements of W1
W1 = (rand(N, I + 1)-0.5)*2; %Weights at the first layer (Input-neurons)
W2 = rand(O, N + 1)-0.5; %Weights at second layer (neurons-output)


%Replace 0 values with 0.1
W2(~W2(:))=0.1;
W1(~W1(:))=0.1;
epoch = 0;
history = [];

%% Problem resolution
tic
while 1
    %% Training
    for in = 1:1:n
        % For each input, write first calculate all the needed elements
        V1 = W1*X(:,in);
        Y = [1 ; tanh(V1)];
        V2 = W2*Y;
        Z = V2;
        % Now that we calculated all the factors, we can start applying the
        % backpropagation
        W2tmp = W2(:,2:end);
        % Layer 1
        W1 = W1 + 2/n*eta * (W2tmp' *(D(in)-Z)).*(1 - tanh(V1).^2) * X(:,in)';
        % Layer 2
        W2 = W2 + 2/n*eta*(D(in)-Z)* Y';% Should be (Z(i) - D(in, i)) where i is the
output
    end
    %% MSE Calculation
    MSE = 0;
    for in = 1:1:n
        V1 = W1*X(:,in);
        Y = [1 ; tanh(V1)];
```

```
        V2 = W2*Y;
        Z = V2;

        MSE = MSE + (D(in) - Z)^2;
    end
    MSE = MSE / n;

    epoch = epoch + 1;
    history = [history ; [epoch MSE]];
    if epoch > 1 && history(epoch-1,2) <= MSE
        eta = eta * 0.9
        epoch
        MSE
    end
    if MSE < epsilon
        break;
    end
end

%% Plot the resulting graph
for in = 1:1:n
    % For each input, write first calculate all the needed elements
    V1 = W1*X(:,in);
    Y = [1 ; tanh(V1)];
    V2 = W2*Y;
    Z = V2;
    plot(X(2, in), Z, '.b'), hold on
end
%% Plot the error/epoch history
figure(2)
plot(history(:,1), history(:,2), '.')
toc
```

Olivo Iacopo - Homework 6

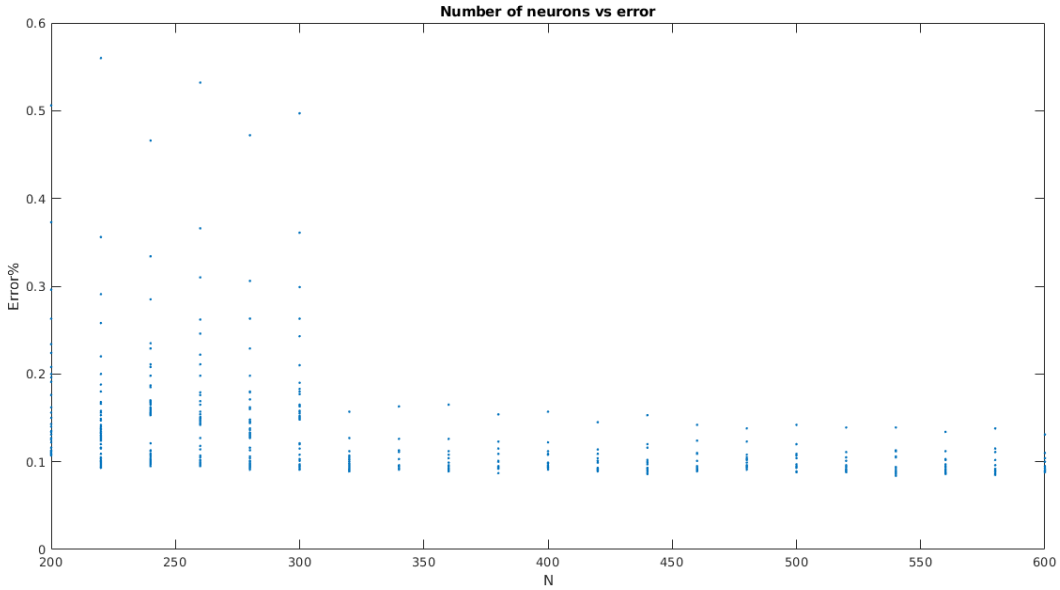
The main objective of this homework is to apply what we learned during the class about network tuning. The task is to create a Neural network capable to classify correctly more than 95% of the MNIST dataset. Particular attention must be paid to network performance and error rate.
In order to create the network there are some choices to be taken:

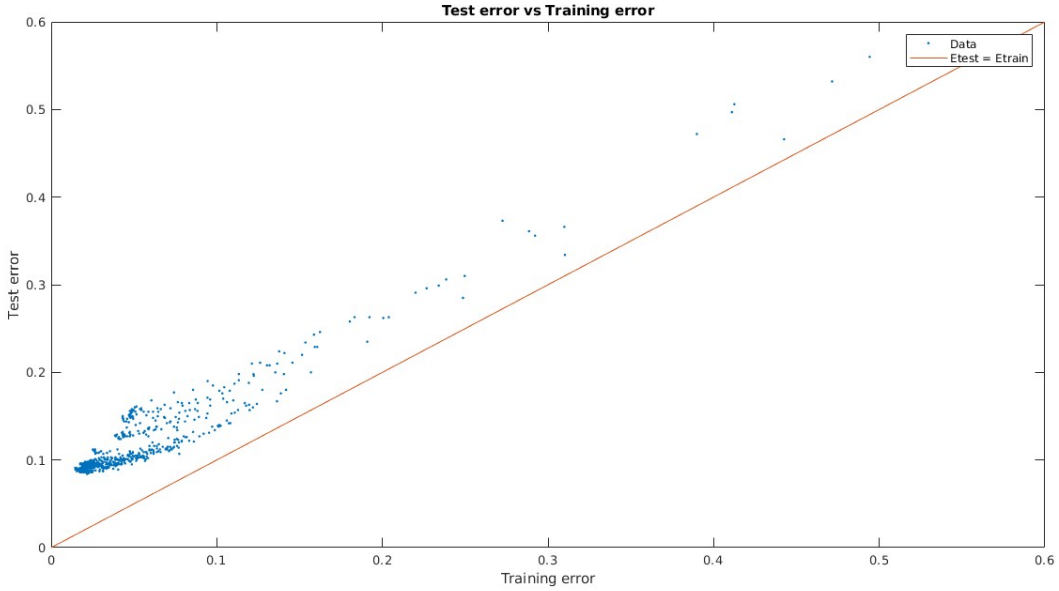
Design choices
They are related on the topology, the activation functions and the initialization of the network

Topology
There are three main choices regarding the topology of the network:

- Number of layers
  It has been decided to use a two layer network (in fact, two layer networks are suitable for most of the uses with decent performance). Increasing the number of layers means increasing the difficulty to train the network. In fact, the more layers we use, the more the network will be prone to saturate (due to a gradient that multiplies itself many times) in both the directions.
- Number of neurons per layer
  We could decide nearly any number N between I and O. As a rule of thumb we could take
  $$N = \frac{I+O}{2} = \frac{784+10}{2} \approx 400$$
  In order to experiment with matlab and the possible values, a very simple search method has been implemented:
  - A smaller dataset has been taken (6000 training elements and 1000 test elements) in order to speed up computation time
  - The network has been trained for a whole night by changing the number of hidden neurons and eta and storing the resulting data in a history array.

  After on night of testing, some data could be taken and analyzed





We note how usually the training set has considerably lower errors than the test set. We can use as decision criteria the difference between the misclassifications on the test and on the training data. This is just a heuristics and it does not pretend to be the reflection of the result presented with the full data set. However we may infer that the network will behave similarly. By analyzing the plot, I could make a more practical proof on the impact of eta and the number of neurons on the training time and relative results. It has been decided to use N = 320 due to the closeness to the training misclassification and the test one and due to the fact that it is near to the value proposed by the rule of thumb. There is another value with similar properties, but having bigger N implies more time to train the network. Moreover, we can note in the first plot how N=320 is the smallest amount of neurons resulting in a overall better training result. Moreover, by analyzing the data, it is possible to notice the overfitting behavior. Lower misclassification errors in the test set happen when the error in the training set is between 2% and 3%. this explains the choice of the stopping condition in the algorithm.
This data has not been considered to the choice of eta, since it is strongly dependent on the size of the training set, hence a smaller one couldn't provide significant data.

- Output Layer and representation
  It has been decided to use ten neurons as output layer. The index corresponding to the classified number will have value 1, while the others will have value 0. This will deeply influence the distance function.

Activation functions:

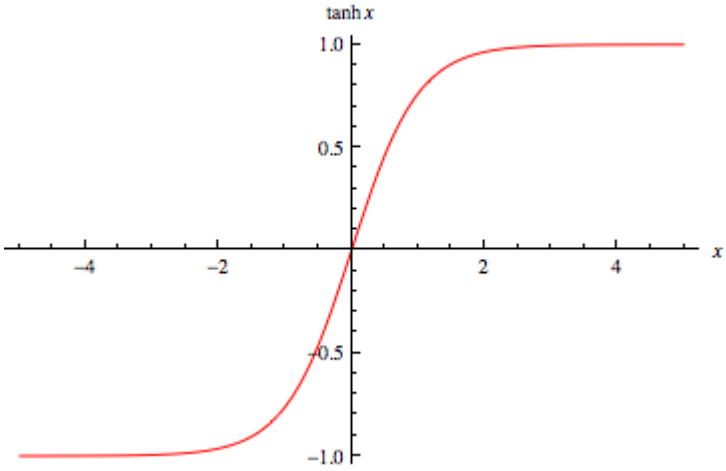
Since the network is two layers deep, two activation functions are needed. As a rule of thumb the activation function should be odd. Hence *tanh* is a perfect candidate for the first layer. In order to experiment it has been decided to use a non odd activation function in the second layer: *sigmoid*. It is the function that mostly represent the heaviside function in continuous domain. The perceptron with the highest output will be considered as a 1, while all the others will be 0.

Distance function

The distance function is deeply influenced by the size of the output layer and the activation function. The principle is the same as the one described in Homerowk2. We check all the outputs and we select the one with the highest value. The index is then compared with the expected output. If they differ, it is counted as a misclassification.

Weight initialization

The initial weights must be chosen depending on the function which will use the local field as an input. W1 (hence V1) is the input of the *tanh* function.



We note how the function saturates near +2 and -2. Hence we want that W1 will have a distribution which vanishes around that value. We have two possibilities:

- Uniform distribution
  It allows us to distribute weights equally among the whole interval. The drawback is that we want more weights near 0, since the gradient is higher in that point and the update will be thus faster.
- Normal distribution
  Is our choice. Due to its shape most of the values will be around 0, and it will vanish at 3σ. We will select 0 mean and σ = 2/3. Moreover we need to consider that V1 will contain I+1 elements, hence we decrease the standard deviation to σ = 2/3I.

The choice for the *sigmoid* function follows the same principle, in fact *sigmoid* will have range [-6, 6], and V2 will contain N+1 elements. Hence σ is set to 2/N and the mean to 0.

Cross function

We needed to use the provided cross function

$$MSE = \frac{1}{2}\sum_{i=1}^{n}(d_i - f(x_i, w))^2$$

Learning rate

It has been decided empirically. I needed to run many tests in order to find the most suitable learning parameter. It is possible to use eta>1 since, due to the cross function used for the back propagation calculation, there is a multiplying factor of $\frac{2}{n}$. A rule of thumb should be that $\eta = \frac{1}{\sqrt{n}}$ but after various checks, 12 is the value who performs the best.

Stopping condition

Similarly to Homework5, it isn't possible to use a stopping condition based on the gradient norm itself. Hence the data produced during the night training session had been analyzed. By checking the progress of all the training sessions I noted that the tests scored best when the training error is around 2%-3%, this leads to the decision of ε = 0.03. Once more, this is a value who has been decided by controlling data built on a different problem set, but the expectation is it to represent in a decent way the effective data distribution on the whole problem.

Tweaks
Methods used in order to increase the performance and lower the error

Input normalization
It has been noted that inputs do not usually have 0 mean. This could affect the network quality. In order to normalize inputs it is sufficient to perform an easy subtraction. Normalization must occur both in the training and in the test sets.

Anonymous functions
They are a special class of matlab functions. They are used in order to increment the modularity of the code and ease the process of changing the activation functions. They are defined at the top of the script and they are automatically substituted to their representation when the script is run. This allows very high performance allowing the code to remain clean and more easy to understand.

Design process
Most of the code is the same as the one present in Homework5 and Homework2. This is due to multiple facts:

- Homework2 already provide a clear, easy and customizable way to read the MNIST dataset
- Homework5 has been used as a test bench in order to develop the backpropagation formulas. They have been written in a general way from the beginning, allowing me to use them for both the homeworks without needing to change anything.

Parameters selection:
Similarly for Homework5, parameters are

- I: Indicates the size of the input
- N: Indicates the number of neurons in the first layer of the network
- O: Indicates the number of neurons present in the output layer of the network.

The network will thus have different elements:

- X is the input vector
- W1 and W2 are the set of weights and biases which are related respectively to the first and second layer
- V1 and V2 are the induced local field.
- $\varphi_1(v)=\tanh(v)$ And $\varphi_2(v)=v$ are the two activation functions.
- Y and Z are the output at the first and second layer. Y will include the bias term as first element

Update equation calculation
In order to train the network we will need to apply newton's method. In order ti calculate $\nabla \mathrm{E}$, we need to use the backpropagation algorithm. In order to calculate the update equations we can use the formula developed during the lesson, for $MSE=\frac{1}{2}\sum_{i=1}^{n}(d_i-f(x_i,w))^2$ slightly modified. This will result in the update formulas for the two layers :

- $\frac{\vartheta \mathrm{E}(w)}{\vartheta w_{j,k}}=-x_k\sum_{i=1}^{O}((z_i-d_i)\varphi_2'(V2_i)W2_{i,j})\varphi_1'(V1_j)$
  The element-wise equation has been converted in matrix form in order to implement a faster update mechanism in the matlab script:

$$W1 = W1 + \frac{2}{n}\eta W2^{T}_{tmp} * \varphi'_2(V2) \circ (D - Z) \circ (\varphi'_1(V1)) * x_i^T$$

We need to note the use of $W2_{tmp}$, which is nothing more than W2 without the bias terms. This is used in order to exclude the bias from the calculation of the backpropagation input coming from the second layer.

- While for W2 we have

$$\frac{\vartheta E(w)}{\vartheta w_{i,j}} = -\varphi'_2(V2_i)\varphi_1(V1_j)(z_i - d_i)$$

Which in matrix form translates to:

$$W2 = W2 + \frac{2}{n}\eta\varphi'_2(V2) \circ (D - Z) * Y^T$$

Development process

In order to develop the script, the Homework5 and Homework2 have been mixed. The weight equation is the same as Homework5. It is needed to add φ functions in order to make them match the designed ones.

On the first version, activation functions were contained inside some script files and called at each execution. This slows down the execution noticeably. Hence the use of Anonymous function was introduced, this increasing the performance significantly.

Moreover, in order to increase the convergence speed and precision, input weights have been normalized.

In order to decide the size of the hidden layer, some tentatives have been done, but trying all the options is very time expensive. Hence the decision to create a simple heuristic to run on the problem subset in order to automate the process. However, some hands on tries needed to be done after the data analysis. I tried to run the network also with 150 hidden neurons, but the accuracy is low and we need a very low eta. Moreover I tried to use a bigger network, with N=500, but the training process was a bit slow.

Next parameter to tune is eta. This has been found empirically. By starting from eta=1 the networks converges too slowly. Hence in order to check the ideal size of the parameter I started with a very big learning parameter, approximately 400. This, as expected diverged soon. Hence, by increasing the parameters size to 50, the network converged. By checking when the parameter was resized, we could have an idea of the ideal dimension of it. In fact, if eta is too big, the MSE calculated each epoch is not ‘non increasing’ and eta keeps resizing thus slowing the weight conversion. In fact, with a smaller eta, the network trains faster.

Preudo-code

After calculating the needed update formulas we are ready to write the script in order to train the network:

1. Initialize all the needed data
    1. Network topology
    2. Eta
    3. Stopping condition
    4. Train set and test set size
    5. For each needed dataset
        1. Open the file descriptor
        2. Read first row values in order to understand the data shape
        3. Calculate the data type

4. Read all the elements in the data set in one row
5. Reshape the data in order to provide a matrix
6. Close the file descriptor

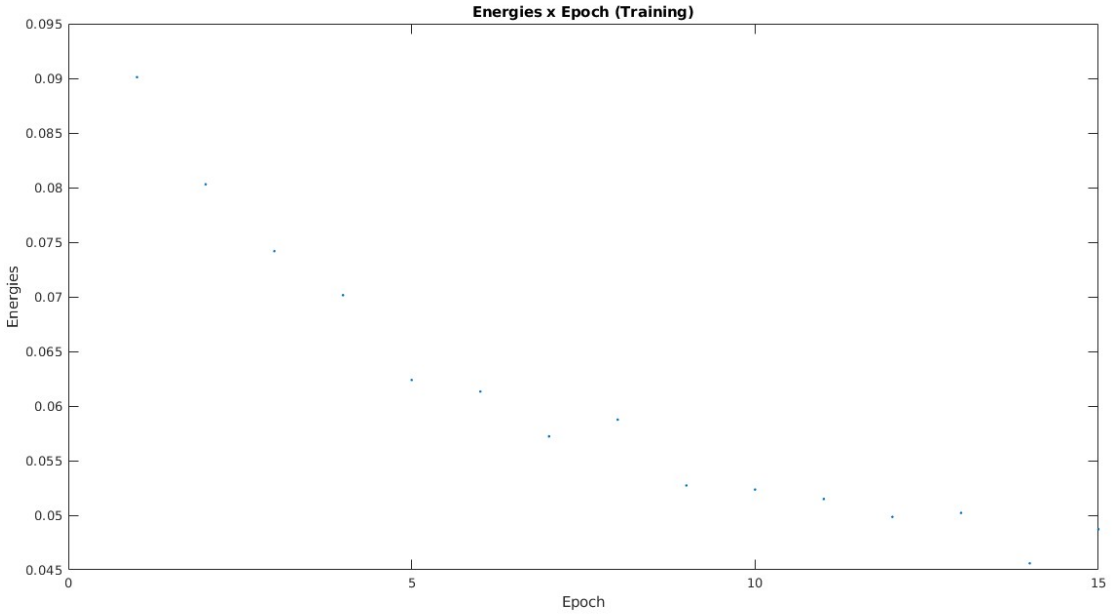
6. Normalize the input training and testing data sets
    1. Calcolate the average of each row
    2. Subtract the average to the current row
7. Create the desired output vector based on the desired output dataset
    1. Create a all 0 array
    2. Put 1 in the position given by the dataset

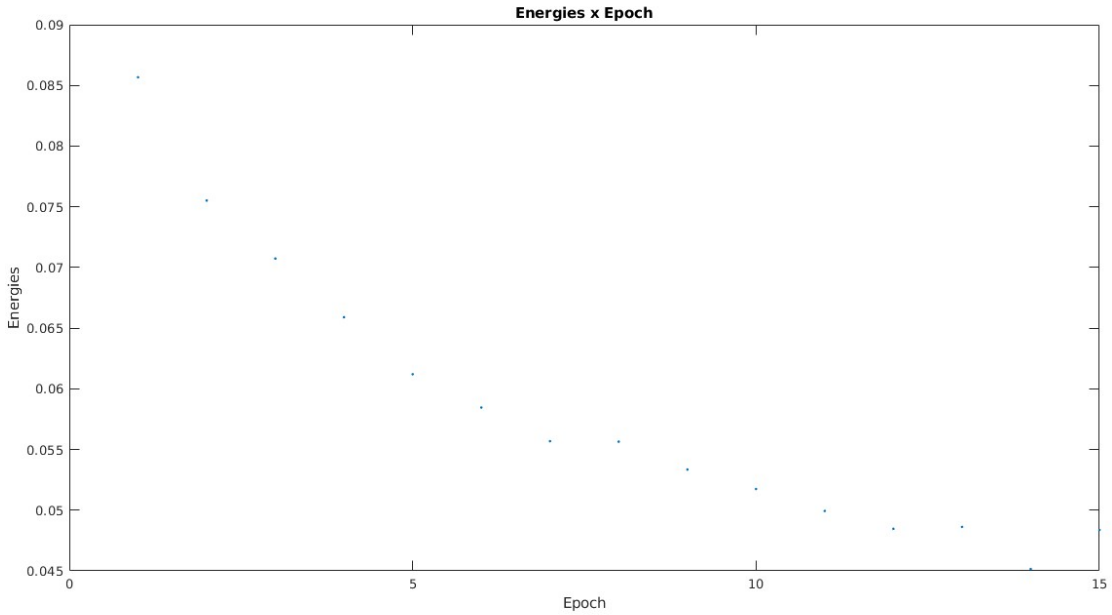
2. Setup the problem
    1. Initialize weights with the values discussed
3. Problem resolution
    1. For each input element:
        1. Calculate V1
           V1 = W1*X(:,in)
        2. Calculate Y
           Y = [1 ; phi1(V1)]
           Note that a trailing one has been added in order to be used as a bias input for the next layer
        3. Calculate V2
           V2 = W2*Y
        4. Calculate Z
           Z = phi2(V2)
        5. Calculate the temporary W2 matrix
           W2tmp = W2(:,2:end)
        6. Update W1 with the formula described before
           W1 = W1 + 2/n*eta * (W2tmp' *(dephi2(V2).*(D(:,in)-Z))).*dephi1(V1) * X(:,in)'
        7. Update W2
           W2 = W2 + 2/n*eta*(dephi2(V2).*(D(:,in)-Z))* Y'
    2. Calculate for both training and test
        1. For each input
            1. Calculate all the coefficients as described in 3.1 (without updating the weights)
            2. If there is misclassification
                1. MSE = MSE + sum((D-Z).^2)
                2. Misc = Misc + 1
        2. Misc = Misc/n
        3. MSE = MSE/n
    3. Create a small log file to keep track of the data
    4. Check for the exit condition
        1. If the condition is met, exit the loop
        2. Otherwise repeat from 3.
4. Plot the required graphs

Note that the heuristics part is not included in the pseudocode since it is not to be considered as a part of the solution algorithm itself. In order to implement it, we need to wrap the points 2 and 3 in a for loop which will initialize the size of N to a different value for each iteration.

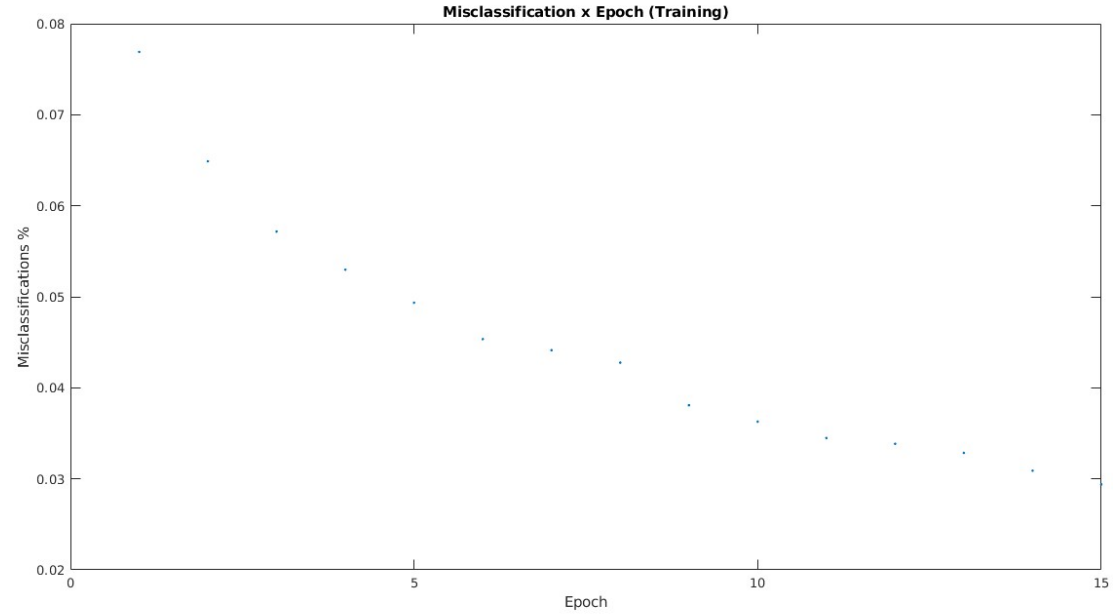
Results

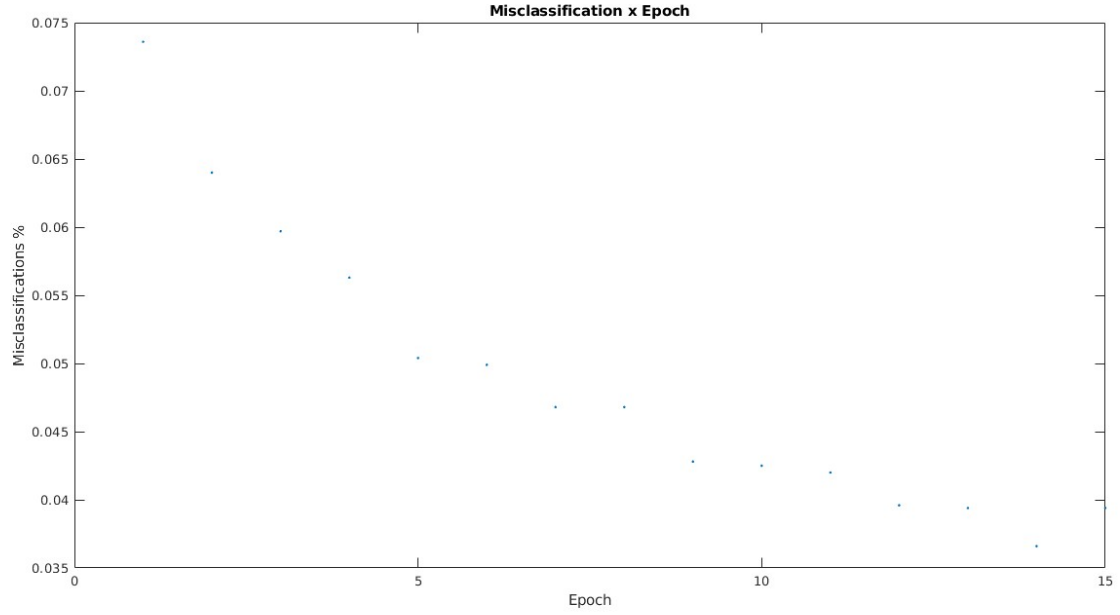
The resulting network is able to reach results well over 95%. In fact it has been tried to reach 97% on the test set with success (The training process is very long tough). In order to reach the desired 97% on the training set (in order to guarantee the test fit to be above 95%) ten minutes are sufficient.
The obtained Energies are:





While the percentage misclassifications are

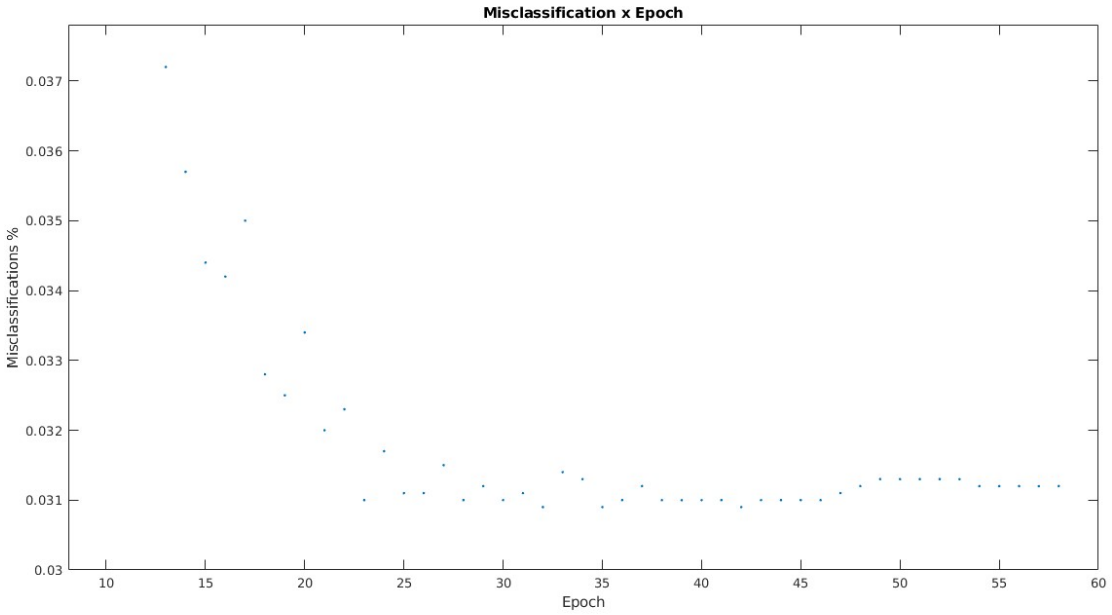




We can clearly note how the plots for the misclassification and for the energies are very similar. This is due to the way energy is calculated: The update formula $MSE = \frac{1}{n}\sum_{i=1}^{n}(d_i - f(x_i, w))^2$ when it is applied to the output of the network will result in the error be either 0 in case of correct classification or 2 in case of misclassification. Due to the chosen exit parameter, no overfitting is present in this training session. We can note hoe the error in the test set is clearly higher thant the one in the training set as was expected from the data obtained in the previous night and wih training precision of 97% the network performs a correct classification in 96.4% of the cases present in the test set.

The training with the official parameters was longer than the one shown in the graph, in order to check if effectively the test error is lower when the training error is in range [2% - 3%]. In fact, the best precision of 3% is reached when the test error is 1.9%.

However, if we run the algorithm in order to achieve a very high test precision, overfitting is present and resulting tests will have similar results (97%). In order to verify the presence of overfitting, the network has been trained in order to reach a precision of 99.5% in the test set.



The first thing to notice is that the test error never drops below 3%, independently on the precision of the test set. We can clearly note how the error increases after the 45^{th} epoch. This is the classical case of overfitting. This helps us to understand that it is not needed to train the network for a huge amount of time in order to reach a very low training error, since it will have little or negative influence on the test set classification.

```
%% Preliminary operations to clear the environment
clc
close all
clear all

phi1 = @(x) tanh(x);
phi2 = @(x) sigmf(x, [1 0]);
dephi1 = @(x) (1 - tanh(x).^2);
dephi2 = @(x) sigmf(x, [1 0]) .*(1 - sigmf(x, [1 0])) ;
%% Constants
n = 60000;
n_test = 10000;
% Our network has size IxNxO where O is the number of output neurons (1)
% and I the number of inputs (1)
I = 784;
N = 320;
O = 10;
eta = 12;
epsilon = 0.005;

%% Data reading
training_set_images = 'train-images.idx3-ubyte';
training_set_labels = 'train-labels.idx1-ubyte';
test_set_images = 't10k-images.idx3-ubyte';
test_set_labels = 't10k-labels.idx1-ubyte';
dataType = ["", "", "", "", "", "", "", "uint8", "int8", "", "", "int16", "int32",
"float", "double"];
Identity = eye(10);

% Import Training data
fp = fopen(training_set_images,'r');
data = fread(fp, 4, 'uint8', 'ieee-be');
precision = dataType(data(3));
data = fread(fp, data(4), 'uint32', 'ieee-be');
nElem = data(1);
data = fread(fp, prod(data), precision, 'ieee-be');
tr_Images = reshape(data, [], nElem);
fclose(fp);

% Read Training labels
fp = fopen(training_set_labels,'r');
data = fread(fp, 4, 'uint8', 'ieee-be');
precision = dataType(data(3));
data = fread(fp, data(4), 'uint32', 'ieee-be');
nElem = data(1);
data = fread(fp, prod(data), precision, 'ieee-be');
tr_Labels = reshape(data, [], nElem);
fclose(fp);

% Read Test data
fp = fopen(test_set_images,'r');
data = fread(fp, 4, 'uint8', 'ieee-be');
precision = dataType(data(3));
data = fread(fp, data(4), 'uint32', 'ieee-be');
nElem = data(1);
data = fread(fp, prod(data), precision, 'ieee-be');
tt_Images = reshape(data, [], nElem);
fclose(fp);

% Read Test Labels
fp = fopen(test_set_labels,'r');
data = fread(fp, 4, 'uint8', 'ieee-be');
```

```
precision = dataType(data(3));
data = fread(fp, data(4), 'uint32', 'ieee-be');
nElem = data(1);
data = fread(fp, prod(data), precision, 'ieee-be');
tt_Labels = reshape(data, [], nElem);
fclose(fp);

%% Problem setup
    % Normalize images
    tr_Images(:,1:n) = tr_Images(:, 1:n) - mean(tr_Images(:, 1:n));
    tt_Images(:,1:n_test) = tt_Images(:, 1:n_test) - mean(tt_Images(:, 1:n_test));

X = [ones(1,n); tr_Images(:,1:n)];
T = [ones(1,n_test) ; tt_Images(:,1:n_test)];
D = zeros(O,n);
for i= 1:1:n
    D(tr_Labels(1,i)+1, i) = 1;
end

totalHistory = [];
for Steta = 1:2:200
    for StN = 200:20:800
        %eta = Steta;
        %N = StN;
        fprintf('N: %d - eta: %d \n', N, eta);
        % Weights matrix has shape:
            % Rows = i (neuron of destination)
            % Columns:
                % 1: bias
                % 2: weight (neuron of source)
        % Elements of W1
       % W1 = ((rand(N, I+1)-0.5)*2)*6/I;
       % W2 = ((rand(O, N+1)-0.5)*2)*6/N;
        W1 = normrnd(0, 0.75/I, N, I+1); % Weights at the first layer (Input-neurons)
[-3, 3]
        W2 = normrnd(0, 2/N, O, N+1); % Weights at second layer (neurons-output) [-6,
6]

        %Replace 0 random values
        W1(~W1(:))=(1/I)*(2*round(rand()) -1);
        W2(~W2(:))=(1/N)*(2*round(rand()) -1);
        epoch = 0;
        history = [];
        % save backupHistory
        %% Problem resolution
        tic
        while 1
            %% Training
            for in = 1:1:n
                % For each input, write first calculate all the needed elements
                V1 = W1*X(:,in);
                Y = [1 ; phi1(V1)];
                V2 = W2*Y;
                Z = phi2(V2);
                % Now that we calculated all the factors, we can start applying the
                % backpropagation
                W2tmp = W2(:,2:end);
                % Layer 1
                W1 = W1 + 2/n*eta * (W2tmp' *(dephi2(V2).*(D(:,in)-Z))).*dephi1(V1) *
X(:,in)';
                % Layer 2
                W2 = W2 + 2/n*eta*(dephi2(V2).*(D(:,in)-Z))* Y';% Should be (Z(i) - D
```

```
(in, i)) where i is the output
            end
            %% Misclassification Calculation
            Misc = 0;
            MSE = 0;
            for in = 1:1:n_test
                V1 = W1*T(:,in);
                Y = [1 ; phi1(V1)];
                V2 = W2*Y;
                Z = phi2(V2);
                [num, index] = max(Z);
                index = index-1;
                if index ~= tt_Labels(in)
                    Misc = Misc + 1;
                    MSE = MSE + sum((D-Z).^2); % We just need to do this when there
are errors
                end
            end
            Misc = Misc / n_test;
            MSE = MSE/n_test;

            MSETrain = 0;
            MiscTrain = 0;
            for in = 1:1:n
                V1 = W1*X(:,in);
                Y = [1 ; phi1(V1)];
                V2 = W2*Y;
                Z = phi2(V2);
                [num, index] = max(Z);
                index = index-1;
                if index ~= tr_Labels(in)
                    MiscTrain = MiscTrain + 1;
                    MSETrain = MSETrain + sum((D-Z).^2); % We just need to do this
when there are errors
                end
            end
            MiscTrain = MiscTrain / n;
            MSETrain = MSETrain / n;
            epoch = epoch + 1;
            fprintf('%d %f \t | eta: %f \t trainError: %f \t testError: %f\n', epoch,
toc, eta, MiscTrain, Misc);
            history = [history ; [N Steta toc epoch eta MiscTrain Misc MSE MSETrain]];
            if epoch > 1 && history(epoch-1,7) <= Misc
                eta = eta * 0.8;
            end
            if toc > 3000
                break;
            end
            if eta < 0.01
                break;
            end
            if MiscTrain < epsilon
                break;
            end
        end
        totalHistory = [totalHistory ; history];
        break;
    end
    break;
end
%% Plot the error/epoch history
figure(1)
```

```
plot(history(:,4), history(:,6), '.')
title('Misclassification x Epoch (Training)')
xlabel('Epoch')
ylabel('Misclassifications %')
figure(2)
plot(history(:,4), history(:,7), '.')
title('Misclassification x Epoch')
xlabel('Epoch')
ylabel('Misclassifications %')
figure(3)
plot(history(:,4), history(:,9), '.')
title('Energies x Epoch (Training)')
xlabel('Epoch')
ylabel('Energies')
figure(4)
plot(history(:,4), history(:,8), '.')
title('Energies x Epoch')
xlabel('Epoch')
ylabel('Energies')
toc
```