University of Groningen

Faculty of Mathematics and Natural Sciences
Department of Computing Science

Neural Networks and Computational Intelligence

Assignment III

Authors

Name	Student Number	E-mail
Sebastian Wehkamp	S2589907	s.wehkamp@student.rug.nl
Dimitris Laskaratos	S3463702	d.laskaratos@student.rug.nl

February 2, 2018

university of groningen

faculty of science and engineering

Contents

1 Problem description

In this exercise we must implement stochastic gradient descent and use it solve a regression problem. To use gradient descent we have to have a error landscape on which we want to find a (local) minimum. This is done by calculating the gradient at every step and moving in the direction of the gradient. This error function is already provided by the exercise and can be seen below. The result is an effective way of finding an optimal set of parameters for the neural network. This implementation of gradient descent will be used to solve a regression problem

$$E = \frac{1}{P}\frac{1}{2}\sum_{\mu=1}^{P}(\sigma(\xi^{\mu}) - \tau(\xi^{\mu}))^2$$

2 Problem solution

To implement gradient descent we considered a simple feed forward neural network with real-valued output:

$$\sigma(\xi) = (tanh[w_1 \cdot \xi] + tanh[w_2 \cdot \xi])$$

where $\xi \in \mathbb{R}^N$ represents an input vector w_1 and w_2 are the N-dim. We initialize both weight vectors with random numbers and unit length.
Every learning step we randomly pick one sample, ξ^v. Since this is a stochastic gradient descent only the contribution of this single sample ξ^v contributes to the weight change. The weights are updated following the rule:

$$w_i \leftarrow w_i - \eta\nabla_i e^v$$

Where η is the learning rate.
So to adapt the weights we must know the gradient ∇_i with respect to weight vector w_i. We can derive this as listed below:

$$\begin{aligned}\nabla_i = \frac{\partial e^v}{\partial w_j} &= [\frac{1}{2}(\sigma(\xi^v) - \tau(\xi^v))^2]' \\ &= (\sigma(\xi^v) - \tau(\xi^v))\sigma'(\xi^v) \\ &= (\sigma(\xi^v) - \tau(\xi^v))[tanh(w_1 \cdot \xi^v) + tanh(w2 \cdot \xi^v)]' \\ &= (\sigma(\xi^v) - \tau(\xi^v))(1 - tanh^2(w_j \cdot \xi^v))\xi^v\end{aligned} \tag{1}$$

The last thing which must be done is calculating the cost function E and the generalization error E_test.

$$E = \frac{1}{P}\frac{1}{2}\sum_{\mu=1}^{P}(\sigma(\xi^{\mu}) - \tau(\xi^{\mu}))^2$$

$$E_{test} = \frac{1}{Q}\frac{1}{2}\sum_{p=P+1}^{P+Q}(\sigma(\xi^{p}) - \tau(\xi^{p}))^2$$

3 Implementation

When we implement everything listed in the problem solution we end up with the code shown in Listing 1. Note that this is only the part of the code which implements the main learning step loop.

```
for i = 1:tmax
    for j = 1:p
        % Select random vector and label from training data set
        idx = randi(size(vectors(1:p), 2));
        vector = vectors(:,idx);
        label = labels(idx);

        % Calculate gradient
```

```
9           gradient = gradientCalc(weights,vector,label);
10
11          % Update weights
12          weights = weights - (learning_rate * gradient);
13      end
14
15      % Calc cost function
16      cost(i) = mean(1/2 * (sum(tanh(weights'*vectors(:, 1:p)))' - labels(1:p
            )).^2);
17      % Calc generalization error
18      error(i) = mean(1/2 * (sum(tanh(weights'*vectors(:, p+1:p+Q)))' -
            labels(p+1:p+Q)).^2);
19  end
```

Listing 1: Code belonging to learning step in gradient descent.

```
1   % Function which calculates the gradient
2   function grad = gradientCalc(weights,vector, label)
3       grad = ((sum(tanh(weights'*vector)) - label) * (1 - tanh(weights'*
            vector).^2) * vector')';
4   end
```

Listing 2: Function which calculates the gradient.

It works as follows. We pick a random vector out of the first P vectors and check the label belonging to the vector. The gradient is calculated using the code shown in Listing 2 which is an implementation of ∇_i we calculated in the previous section. The last step is to calculate the cost function and the generalization error and store the results so that they can be plotted easily.

4 Evaluation

We started our evaluation using the settings required by the exercise. This meant we use a η of 0.05, P of 100, Q of 100 and 2000 epochs. The training and generalization error can be seen in Figure 1 while the accompanying weights can be found in Figure 2. In the image you can see that both the training error and generalization error start at the same level as expected. However while the training error decreases very quickly and stabilizes after about 200 epochs the generalization error goes up very fast and keeps increasing slightly. This means that we over fitted to our training data and we cannot generalizes our neural network to novel data.

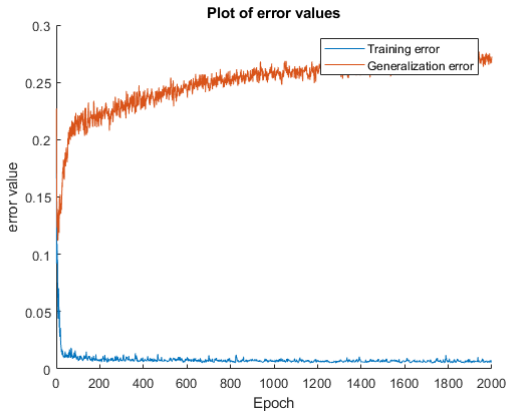


Figure 1: Plot of the training and generalization error created using $P = 100$, $Q = 100$, $\eta = 0.05$

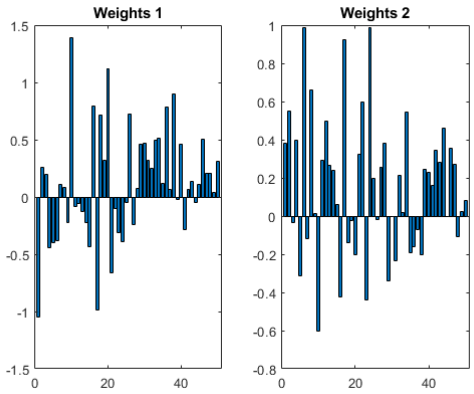
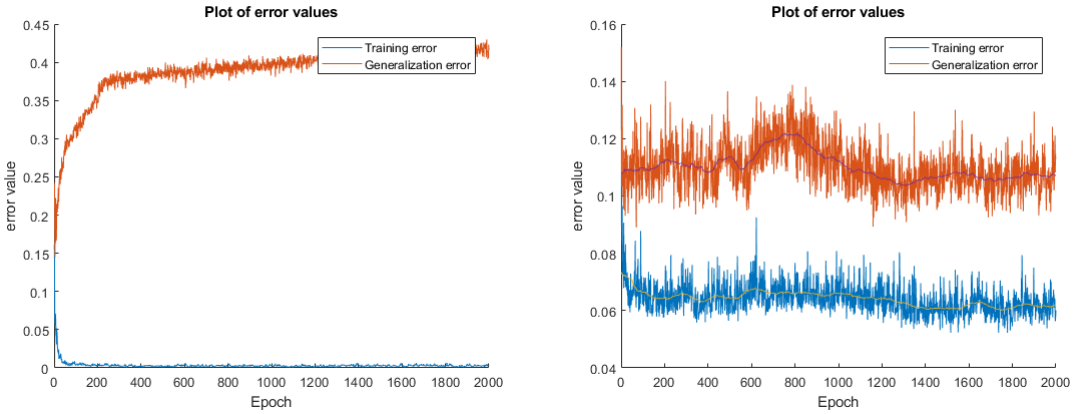


Figure 2: Plot of the weights created using $P = 100$, $Q = 100$, $\eta = 0.05$

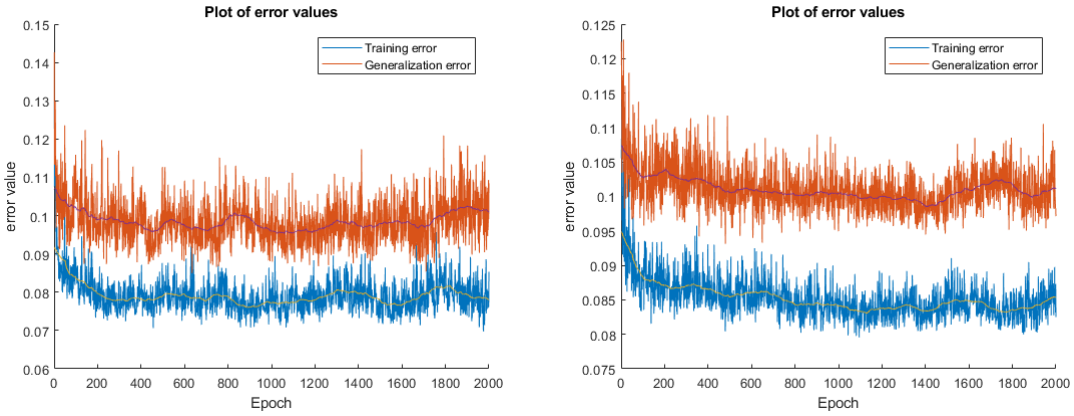
Our next experiment was to see how both of the error rate depend on the values of P and Q. To do this we used the values 50, 100, 500, 1000, 2000 for both P and Q. The results for the error rates can be seen in Figure 3 and Figure 4. Note that to make the images better readable we plotted a moving average for both error values. The first thing which stands out is that when we use a value of $P = 50$ the generalization increases overtime just as with $P = 100$ in Figure 1. When we use a value of $P > 100$ like $P = 500$ the training error increases somewhat with respect to $P = 100$ but the generalization error is much lower and seems to stabilize over time thus there is no overfitting. When we start using even larger values for P as seen in Figure 4b they get even closer together.



(a) Plot of the training and generalization error created using $P = 50$, $Q = 50$, $\eta = 0.05$

(b) Plot of the training and generalization error created using $P = 500$, $Q = 500$, $\eta = 0.05$

Figure 3: Comparison of error rates using varying values of P and Q.



(a) Plot of the training and generalization error created using $P = 1000$, $Q = 1000$, $\eta = 0.05$

(b) Plot of the training and generalization error created using $P = 2000$, $Q = 2000$, $\eta = 0.05$

Figure 4: Comparison of error rates using varying values of P and Q.

We also tried modifying the learning rate and set it to a very low value of $\eta = 0.001$. The results can be seen in Figure 5 and Figure 6. Due to the very slow learning rate it takes a while before the training error is at its minimum however when it reaches its minimum the training error is almost 0. The generalization error is also very small and both the training and generalization error stabilize after 600 epochs. The final weights values as seen in Figure 6 are also far from random values. One of the weight plots always contains positive values while the other contains both positive and negative values and alternating between positive and negative values. The weights not being random means we actually learned something.

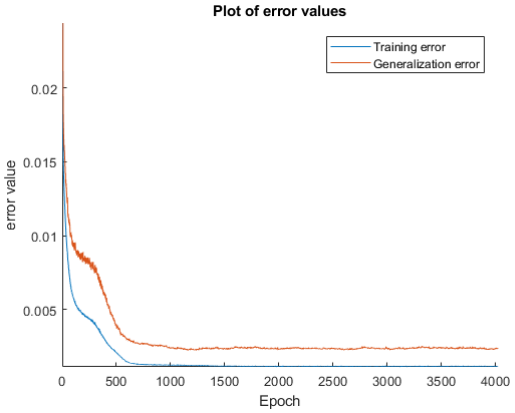


Figure 5: Plot of the training and generalization error created using $P = 500$, $Q = 500$, $\eta = 0.001$

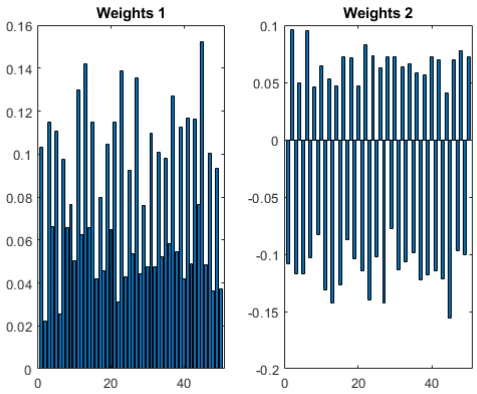


Figure 6: Plot of the weights created using $P = 500$, $Q = 500$, $\eta = 0.001$