Learning by Gradient Descent: Stochastic Gradient Descent

Yulan van Oppen (s2640325)
Jarvin Mutatiina (s3555631)

January 14, 2019

1 Introduction

Assume we have a set of N-dimensional input vectors $\boldsymbol{\xi}$, along with corresponding teacher outputs $\tau(\boldsymbol{\xi})$. Suppose moreover we define a *feed-forward network*, which consists of an input layer, followed by a layer of K hidden units, followed by one output unit (or multiple, but in this report we consider only one). Feeding an input vector $\boldsymbol{\xi}$ to the network is equivalent to setting the N input units in the input layer equal to the components of $\boldsymbol{\xi}$. Information may only flow from the input to the hidden layer, and from the hidden to the output layer. The input units are connected to the hidden units via K input-to-hidden weight vectors $\mathbf{w}_j$ (one for each hidden unit), the latter being of the form

$$g\left(\mathbf{w}^{(j)} \cdot \boldsymbol{\xi}\right)$$

for some function $g : U \to \mathbb{R}$ for some $U \subset \mathbb{R}$. The hidden units are in turn connected to the output units through hidden-to-output weights v_j (also one for each hidden unit). This means the output $\sigma(\boldsymbol{\xi})$ has the form

$$\sigma(\boldsymbol{\xi}) = h\left(\sum_{j=1}^{K} v_j g\left(\mathbf{w}^{(j)} \cdot \boldsymbol{\xi}\right)\right),$$

for some function $h : V \to \mathbb{R}$ for some $V \subset \mathbb{R}$. The general structure of this network is depicted in Figure 1.

We would like to find input-to-hidden and hidden-to-output weights that result in student outputs $\sigma(\boldsymbol{\xi})$ close to the teacher outputs. That is, the weights should minimize the *cost function*, E defined by

$$E := \frac{1}{P}\sum_{\mu=1}^{P} \frac{1}{2}\left(\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)\right)^2. \tag{1}$$

Where P is the number of data points considered, and $\boldsymbol{\xi}^\mu$ represents the μ-th input vector. A single summand is denoted by

$$e^\mu := \frac{1}{2}\left(\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)\right)^2, \tag{2}$$

and represents the (unaveraged) contribution of $\boldsymbol{\xi}^\mu$ to the cost function.

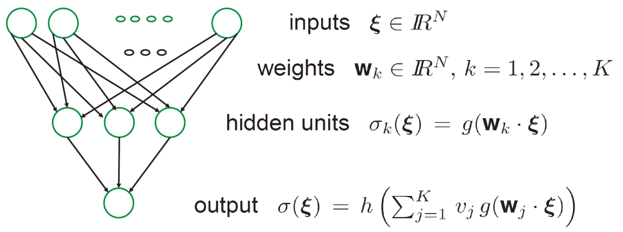
Gradient descent takes initial (usually random) input-to-hidden weight vectors $\mathbf{w}_j$ of unit length and hidden-to-output weights v_j, and updates these weights by the update equations

$$\begin{aligned} \mathbf{w}_j^{(t+1)} &= \mathbf{w}_j^{(t)} - \eta_{\mathbf{w}} \nabla_j E, \\ v_j^{(t+1)} &= v_j^{(t)} - \eta_v \frac{\partial}{\partial v_j} E. \end{aligned}$$

Here ∇_j denotes the gradient with respect to $\mathbf{w}_j$. The *learning rates* $\eta_{\mathbf{w}}$ and η_v need not be equal, and may be time-dependent.

For large data sets, calculating $\nabla_j E$, $j = 1, \ldots, K$, can be computationally expensive. *Stochastic gradient descent* takes an arbitrary input vector $\boldsymbol{\xi}^\mu$ at each learning step, and considers only the gradient of the contribution e^μ with respect to $\mathbf{w}_j$ for each j to the cost function. Each input vector is selected with equal probability. For a time step t, the update equations therefore become

$$\begin{aligned} \mathbf{w}_j^{(t+1)} &= \mathbf{w}_j^{(t)} - \eta_{\mathbf{w}} \nabla_j e^\mu, \\ v_j^{(t+1)} &= v_j^{(t)} - \eta_v \frac{\partial}{\partial v_j} e^\mu. \end{aligned}$$



inputs $\boldsymbol{\xi} \in \mathbb{R}^N$
weights $\mathbf{w}_k \in \mathbb{R}^N,\ k = 1, 2, \ldots, K$
hidden units $\sigma_k(\boldsymbol{\xi}) = g(\mathbf{w}_k \cdot \boldsymbol{\xi})$
output $\sigma(\boldsymbol{\xi}) = h\left(\sum_{j=1}^{K} v_j\, g(\mathbf{w}_j \cdot \boldsymbol{\xi})\right)$

Figure 1: A general feed-forward network with a single output [1].

In particular, we consider the case where $N = 50$, $K = 2$, g is the hyperbolic tangent function, $v_1 = v_2 = 1$ and h is the identity function. This means our student network will have output

$$\sigma(\boldsymbol{\xi}) = \tanh[\mathbf{w}_1 \cdot \boldsymbol{\xi}] + \tanh[\mathbf{w}_2 \cdot \boldsymbol{\xi}]. \tag{3}$$

The remainder of this report will be dedicated to training the network w.r.t. the weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ by means of stochastic gradient descent, and analyzing the effects of varying certain parameters on the (evolution of the) cost function. The Methodology section derives the necessary gradient equations, and moreover includes the complete stochastic gradient descent algorithm. The Results section investigates the consequences of varying the learning rate and the size of the test and training sets. The Discussion section aims to interpret these results. Finally, the Conclusion section summarizes the results in some concluding remarks.

2 Methodology

The data provided consists of $M = 5000$ input vectors $\boldsymbol{\xi}^\mu$ (recall, of dimension $N = 50$), and M corresponding target values $\tau(\boldsymbol{\xi}^\mu)$. We will train on the first P and test on the next Q input vectors. For an input vector $\boldsymbol{\xi}$, the output $\sigma(\boldsymbol{\xi})$ is as defined in (3). Recall $\mathbf{w}_1$ and $\mathbf{w}_2$ are the input-to-hidden weight vectors. These will be optimized by means of stochastic gradient descent.

To this end, we need to determine the gradient of e^μ (c.f. (2)) with respect to $\mathbf{w}_j$ for $j = 1, 2$. This is easily accomplished by means of matrix differentiation, and the chain rule. Letting ∇_j denote the gradient with respect to $\mathbf{w}_j$ for $j = 1, 2$, we find

$$\begin{aligned}
\nabla_j e^\mu &= \nabla_j \left\{ \frac{1}{2}(\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu))^2 \right\} \\
&= (\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)) \nabla_j \sigma(\boldsymbol{\xi}^\mu) \\
&= (\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)) \nabla_j \tanh[\mathbf{w}_j \cdot \boldsymbol{\xi}^\mu] \\
&= (\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)) \left(1 - \tanh^2[\mathbf{w}_j \cdot \boldsymbol{\xi}^\mu]\right) \nabla_j \{\mathbf{w}_j \cdot \boldsymbol{\xi}^\mu\} \\
&= (\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)) \left(1 - \tanh^2[\mathbf{w}_j \cdot \boldsymbol{\xi}^\mu]\right) \boldsymbol{\xi}^\mu.
\end{aligned}$$

As well as the value of the cost function (c.f. (1)), the generalization error

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

is recorded after each time step. This allows for comparison of their graphs as a function of the time step. The generalization error is an important measure of performance of the trained network with respect to new data.

Algorithm 1 Stochastic Gradient Descent algorithm

```
Input: ${\boldsymbol\xi^\mu, \tau(\boldsymbol\xi^\mu)}_{\mu=1}^M$    ▷ Data points $\boldsymbol\xi^\mu \in \mathbb{R}^N$ and corresponding continuous target values $\tau(\boldsymbol\xi^\mu)$

Output: $\mathbf{w}_1$,$\mathbf{w}_2$, $E(t)$, $E_{test}(t)$    ▷ Final weight vectors $\mathbf{w}_1, \mathbf{w}_2$, cost function and generalization error

Parameters: $P$, $Q$, $t_{max}$, $\eta$    ▷ Training set size, test set size, number of learning steps and learning rate $\eta := \eta_{\mathbf{w}}$

Procedure:

  1. Initialize $\mathbf{w}_1, \mathbf{w}_2$;    ▷ Independent random weight vectors $\mathbf{w}_1, \mathbf{w}_2$ with $|\mathbf{w}_1| = |\mathbf{w}_2| = 1$.
  2. For time steps $t = 1, 2, \ldots, t_{max}$ do
       For $i = 1, \ldots, P$ do
         Select $\boldsymbol\xi^{\nu_i}$ from ${\boldsymbol\xi^\mu}_{\mu=1}^P$ randomly    ▷ Each with equal probability of being selected

         $\sigma(\boldsymbol\xi^{\nu_i}) \leftarrow \tanh[\mathbf{w}_1 \cdot \boldsymbol\xi^{\nu_i}] + \tanh[\mathbf{w}_2 \cdot \boldsymbol\xi^{\nu_i}]$    ▷ Recompute student network output for $\boldsymbol\xi^{\nu_i}$
         $\nabla_j e^{\nu_i} \leftarrow (\sigma(\boldsymbol\xi^{\nu_i}) - \tau(\boldsymbol\xi^{\nu_i}))\left(1 - \tanh^2[\mathbf{w}_j \cdot \boldsymbol\xi^{\nu_i}]\right)\boldsymbol\xi^{\nu_i}$  for  $j = 1, 2$

         $\mathbf{w}_1 \leftarrow \mathbf{w}_1 - \eta\nabla_1 e^\nu$    ▷ Update the weights based on $\nabla_j e^\nu$
         $\mathbf{w}_2 \leftarrow \mathbf{w}_2 - \eta\nabla_2 e^\nu$
       end for

       $\sigma(\boldsymbol\xi^\mu) \leftarrow \tanh[\mathbf{w}_1 \cdot \boldsymbol\xi^\mu] + \tanh[\mathbf{w}_2 \cdot \boldsymbol\xi^\mu]$  for  $\mu = 1, \ldots, P$    ▷ Update student network outputs

       $E(t) \leftarrow \frac{1}{P}\sum_{\mu=1}^P \frac{1}{2}\left(\sigma(\boldsymbol\xi^\mu) - \tau(\boldsymbol\xi^\mu)\right)^2$    ▷ Update cost function
       $E_{test}(t) \leftarrow \frac{1}{Q}\sum_{\mu=P+1}^{P+Q} \frac{1}{2}\left(\sigma(\boldsymbol\xi^\mu) - \tau(\boldsymbol\xi^\mu)\right)^2$    ▷ Update generalization error
     end for
```

3 Results

As a starting point, we take $P = Q = 100$, $\eta = 0.05$, $t_{max} = 50$, and we average $E(t)$ and E_{test} as defined in the previous section over 100 runs. The graphs of $E(t)$ and $E_{test}(t)$ with respect to the time step t are depicted in Figure 2. As t increases beyond 10, $E(t)$ decreases whereas E_{test} starts to increase.

The final weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ after t_{max} time steps are displayed as bar graphs in Figure 3. No clear discernible pattern is clearly visible, other that the weights are mostly positive. When increasing P and Q (still letting $P = Q$), the cost function increases (Figure 4) and the generalization error decreases (Figure 5) For $P, Q = 1000$ and $P, Q = 2000$, the difference between the two error measures is the smallest. Since this is desirable, and because the results for $P, Q = 1000$ and for $P, Q = 2000$ are nearly identical, we take the former for computational efficiency.

Therefore setting $P = Q = 1000$, the results of varying η are shown in Figure 6. It shows the optimal value (among the values used) is $\eta = 0.001$. For higher values, the average $E(t)$ is higher for all but very small t. For lower values, the convergence is much slower. Note at this point we only consider constant learning rates. Figure 7 and 8 display the graphs of $E(t)$ and $E_{test}(t)$ with $\eta(t) = 0.001$ for all t versus those with $\eta(t) = \frac{30}{1000t}$. Figure 8 in particular shows the improvement is negligible.

Finally, we consider the network with adjustable hidden-to-output weights v_1, v_2. The output for an put vector $\boldsymbol\xi^\mu$ is then

$$\sigma(\boldsymbol\xi^\mu) = v_1 \tanh[\mathbf{w}_1 \cdot \boldsymbol\xi^\mu] + v_2 \tanh[\mathbf{w}_2 \cdot \boldsymbol\xi^\mu].$$

We easily obtain the new gradients of e^μ with respect $v_1, v_2, \mathbf{w}_1$ and $\mathbf{w}_2$, namely

$$\frac{\partial e^\mu}{\partial v_j} = (\sigma(\boldsymbol\xi^\mu) - \tau(\boldsymbol\xi^\mu))\frac{\partial\sigma(\boldsymbol\xi^\mu)}{\partial v_j} = (\sigma(\boldsymbol\xi^\mu) - \tau(\boldsymbol\xi^\mu))\tanh[\mathbf{w}_j \cdot \boldsymbol\xi]$$

and

$$\nabla_j e^\mu = (\sigma(\boldsymbol\xi^\mu) - \tau(\boldsymbol\xi^\mu))\nabla_j \tanh[\mathbf{w}_j \cdot \boldsymbol\xi^\mu] = (\sigma(\boldsymbol\xi^\mu) - \tau(\boldsymbol\xi^\mu))v_j\left(1 - \tanh^2[\mathbf{w}_j \cdot \boldsymbol\xi^\mu]\right)\boldsymbol\xi^\mu$$

for $j = 1, 2$. The rest of the algorithm stays the same, apart from also updating v_1, v_2 for each randomly chosen import vector. Figure 9 displays the averaged (again, over 50 runs) graphs of $E(t)$ and E_{test} with respect to t for $P = Q = 1000$ and $\eta_{\mathbf{w}} = 0.001$ (as among the values tested for, these appeared optimal). For comparison, the results belonging to the network with fixed $v_1 = v_2 = 1$ are plotted as well, displayed in blue. The stochastic gradient descent algorithm with adjustable hidden-to-output algorithm is run for various η_v. The figure shows $\eta_v = 10^{-5}$ to be optimal among the values tested with. However, a significant improvement of holding $v_1 = v_2 = 1$ fixed can be observed.

Having experimentally found optimal parameter values $P = Q = 1000$, $\eta = 0.001$ and v_1, v_2 fixed at 1, we recompute the final weight vectors after $t_{max} = 50$ learning steps. The results are displayed as bar graphs in Figure 10. The graph seems to indicate the even components of the input vectors are of approximately equal relevance, and more important than the odd components. It looks like this holds for both weight vectors.

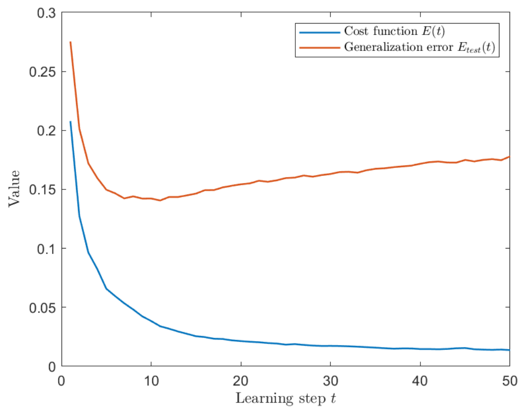


Figure 2: The cost function $E(t)$ and generalization error $E_{test}(t)$ w.r.t t, averaged over 100 runs with $P = Q = 100$, $\eta = 0.05$ and $t_{max} = 50$.

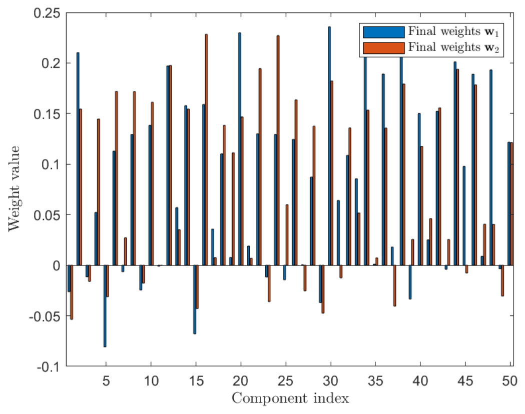


Figure 3: Final weights $\mathbf{w}_1$ and $\mathbf{w}_2$, averaged over 100 runs with $P = Q = 100$, $\eta = 0.05$ and $t_{max} = 50$.

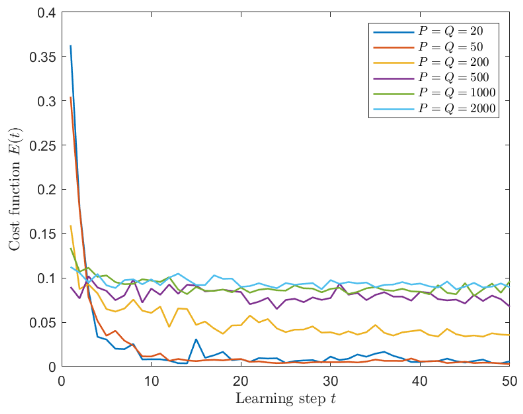


Figure 4: The cost function $E(t)$ w.r.t t with $\eta = 0.05$, $t_{max} = 50$ and various P, Q.

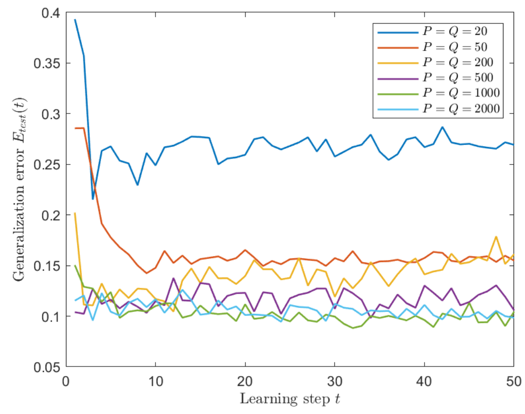


Figure 5: The generalization error $E_{test}(t)$ w.r.t t with $\eta = 0.05$, $t_{max} = 50$ and various P, Q.

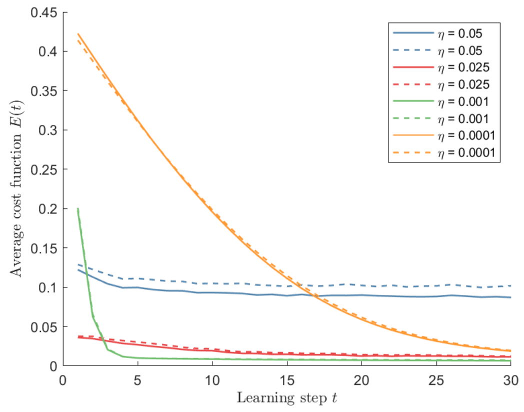


Figure 6: The cost function $E(t)$ (solid) and generalization error $E_{test}(t)$ (dashed) w.r.t t, averaged over 50 runs with $P = Q = 1000$, $t_{max} = 50$ and various η.

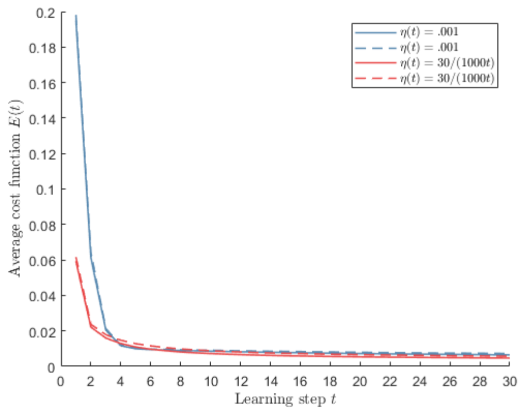


Figure 7: The cost function $E(t)$ (solid) and generalization error $E_{test}(t)$ (dashed) w.r.t t, averaged over 50 runs with $P = Q = 1000$, $t_{max} = 50$ and various time-dependent $\eta(t)$.

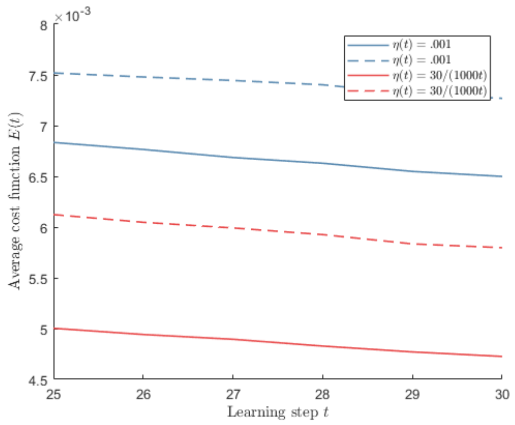


Figure 8: The same as Figure 7, focused on the range $25 \leqslant t \leqslant 30$.

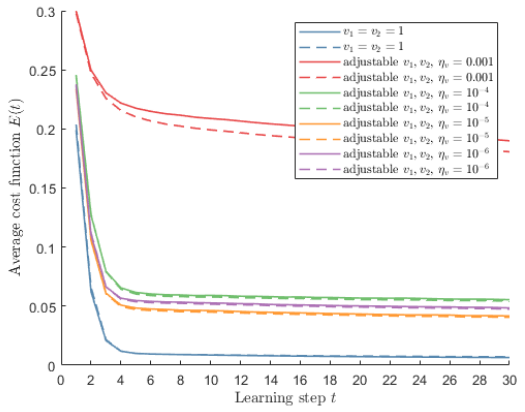


Figure 9: The cost function $E(t)$ (solid) and generalization error $E_{test}(t)$ (dashed) w.r.t t, averaged over 50 runs with $P = Q = 1000$, $t_{max} = 50$ and $\eta_{\mathbf{w}} = 0.01$. Here v_1, v_2 are adjustable (except for the graphs in blue) and η_v is varied upon.

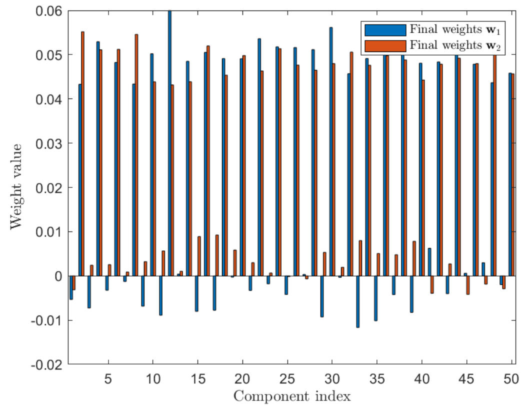


Figure 10: Final weights $\mathbf{w}_1$ and $\mathbf{w}_2$, averaged over 100 runs with $P = Q = 1000$, $\eta = 0.001$ and $t_{max} = 50$.

4 Discussion and Conclusion

Figure 2 shows a well known problem; upon running the algorithm for additional learnings steps beyond $t = 5$, the generalization error increases as the cost function decreases. This indicates an overly complex model, meaning the weight vectors are perfectly adjusted to the training set. This, however, does not generalize well.

Figure 4 and 5 show this undesirable result can be counteracted by increasing the size of the training set (and consequently that of the test set). The fact that the difference between $E(t)$ and $E_{test}(t)$ is reduced for larger P, Q outweighs the fact that $E(t)$ is significantly higher, since the model with higher P, Q generalizes better.

In the previous section, we remarked that increasing η above 0.001, rendered $E(t)$ and $E_{test}(t)$ higher for all but very small t (c.f. Figure 6). This can be explained by the fact that for too large η, the weight vector will oscillate around a (local) optimum. A finer step size is required for convergence. Moreover, decreasing η below 0.001 lead to another undesirable effect, namely the significant loss in convergence rate. The graphs (orange) in Figure 6 support this claim.

The improvement due to using a time-dependent $\eta(t)$, appeared negligible, c.f. Figure 7. This could be due to the constant $\eta = 0.001$ already achieving approximately minimal $E(t)$ and $E_{test}(t)$, and that further decrease is not possible due to random noise in the data.

The addition of adjustable hidden-to-output weights did not improve the cost function and the generalization error for any η_v. This is surprising, as we would expect that additional optimization would be possible. Perhaps different parameter settings for P, Q and $\eta_{\mathbf{w}}$ could achieve this. In any case, further investigation is appropriate.

5 Conclusion

We observed that for a relatively small training set, advancing in learning steps beyond $t = 5$ leads to an overly complex model that does not generalize well. This is a well known problem which is frequently observed in practice. Increasing the size of the test set counteracts this, as experiments have confirmed. This is in line with theory [1].

With a sufficiently large test set, further improvement was obtained only by decreasing the learning rate $\eta := \eta_{\mathbf{w}}$ to 0.001. Higher values of η would lead to oscillation around local optima, and lower values to a slower convergence rate. This constant learning rate possibly achieves minimal cost and generalization error, as considering a time dependent $\eta(t)$ did not lead to a significant improvement.

The fact that making the hidden-to-output weights adjustable did not lead to improvements is questionable, since we would expect it to enable further optimization. As mentioned in the previous section, this unintuitive result should be researched in further work.

References

[1] M. Biehl. *Lecture notes in Neural Networks and Computational Intelligence*. Faculty of Science and Engineering, University of Groningen, January 2019.