Machine Intelligence 1 **Exercise Sheet 3**
WS 2016/17, Obermayer/Böhmer due: 10.11.2016

Multi-layer Perceptrons and backpropagation

Exercise T3.1: Error measures (tutorial)

(a) What effect will the choice of error measure have on the predictor?

(b) Give examples of problems in which each of the measures discussed in the lecture, i.e. *linear, quadratic, maximum penalty* and *tolerate small errors*, will have an advantage over the others?

(c) If the output of a neural network is interpreted as the probability that the input blongs to one class, which error function could be suitable?

Exercise T3.2: Validation (tutorial)

(a) What is validation and why is it needed?

(b) What is the differences between *over-fitting* and *under-fitting*?

(c) Name and discuss the techniques presented in the lecture to perform validation.

Exercise H3.1: Binary Classification (homework, 3 points)

For binary targets $y_T^{(\alpha)} \in \{0, 1\}$ and network architecture $\mathcal{A}$, one can interpret the network output $y(\mathbf{x}; \mathbf{w}, \mathcal{A}) \in (0, 1)$ as a probability $P(t = 1|\mathbf{x}; \mathbf{w}, \mathcal{A})$. A suitable error function for this problem is:

$$E^T = \frac{1}{p} \sum_{\alpha=1}^{p} e^{(\alpha)}$$

with

$$e^{(\alpha)} = -\left[y_T^{(\alpha)} \ln y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A}) + (1 - y_T^{(\alpha)}) \ln \left(1 - y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A})\right)\right].$$

(a) (1 point) Show that

$$\frac{\partial e^{(\alpha)}}{\partial y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A})} = \frac{y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A}) - y_T^{(\alpha)}}{y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A}) \left(1 - y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A})\right)}$$

(b) (1 point) Consider that the architecture $\mathcal{A}$ consists of an MLP with one hidden layer. The non-linear transfer function for the output neuron $(i = 1, v' = 2)$ is assumed to be

$$f(h_1^2) = \frac{1}{1 + e^{-h_1^2}},$$

where h_1^2 is the total input of the output neuron. Show that its derivative can be expressed as

$$f'(h_1^2) = f(h_1^2)(1 - f(h_1^2)).$$

(c) (1 point) Using the results from a) and b), show that the gradient of the error function $e^{(\alpha)}$ with respect to the weight w_{1j}^{21} between the the single output neuron $(i = 1, \nu' = 2)$ and neuron j of the hidden layer $(v = 1)$ is

$$\frac{\partial e^{(\alpha)}}{\partial w_{1j}^{21}} = \left(y(\mathbf{x}^{(\alpha)}; \mathbf{w}, \mathcal{A}) - y_T^{(\alpha)}\right) S_j^1$$

where S_j^1 is the activity of neuron j of the hidden layer $(v = 1)$.

Solution

(a) Let $e := e^{(\alpha)}$, $y := y(\underline{\mathbf{x}}; \underline{\mathbf{w}}, \mathcal{A})$ and $y_T := y_T^{(\alpha)}$:

$$\frac{\partial e}{\partial y} = -\Big[\frac{y_T}{y} - \frac{1 - y_T}{1 - y}\Big] = \frac{y(1 - y_T) - y_T(1 - y)}{y(1 - y)} = \frac{y - y_T}{y(1 - y)}\,.$$

(b) Let $h := h_1^2$:

$$\frac{\partial f(h)}{\partial h} = \frac{e^{-h}}{(1 + e^{-h})^2} = f(h)\frac{1 + e^{-h} - 1}{1 + e^{-h}} = f(h)\left(1 - f(h)\right).$$

(c) Let $e := e^{(\alpha)}$, $y_T := y_T^{(\alpha)}$, $w := w_{1j}^{21}$, $h := h_1^2$ and $y := y(\underline{\mathbf{x}}; \underline{\mathbf{w}}, \mathcal{A})$.
Note that $h = \sum_{j=0}^{n_{\text{hid}}} w_{1j}^{21} S_j^1$ and $y = f_1^2(h)$:

$$\frac{\partial e}{\partial w} \;=\; \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w} \;=\; \frac{y - y_T}{y(1 - y)} \cdot y\big(1 - y\big) \cdot S_j^1 \;=\; (y - y_T)\, S_j^1\,.$$

Exercise H3.2: MLP Regression (homework, 7 points)

The task is to implement a simple MLP with one hidden layer and apply the backpropagation algorithm to learn its parameters for a simple regression task.

Training Data: The file `RegressionData.txt` from the ISIS platform contains a small training dataset $\{x_n, t_n\}$, $n = 1, \ldots, N$ with $N = 10$. The input values $\{x_n\}$ in the first column are random numbers drawn from a uniform distribution over the interval $[0, 1]$. The target values $\{t_n\}$ were generated using the function $\sin(2\pi x_n)$ and Gaussian noise with standard deviation $\sigma = 0.25$ was added.

Initialization: Set the weights and biases of a MLP with one hidden layer (three neurons) and a single output neuron to random values from the interval $[-0.5, 0.5]$.

Backpropagation: To implement the iterative learning procedure, your program should realize the following steps for each weight update:

1. **Forward Propagation:** Calculate the activations of the hidden neurons using the $\tanh$ transfer function and the activation of the output neuron using the linear transfer function (i.e. the identity) for each input value of the training set.

2. Compute the **output error** using the quadratic error cost function.
3. **Backpropagation:** Calculate the "local errors" δ_i^ν for the output and the hidden layer for each training point. Use these "local errors" to calculate the batch gradient of the error function w.r.t. the first and second layer weights, from which you obtain the direction of the weight updates:

$$\Delta w_{ji}^{10} = -\frac{\partial E^T(\boldsymbol{w})}{\partial w_{ji}^{10}} = -\frac{1}{N}\sum_{n=1}^{N}\frac{\partial e_n^T(\boldsymbol{w})}{\partial w_{ji}^{10}}$$

$$\Delta w_{kj}^{21} = -\frac{\partial E^T(\boldsymbol{w})}{\partial w_{kj}^{21}} = -\frac{1}{N}\sum_{n=1}^{N}\frac{\partial e_n^T(\boldsymbol{w})}{\partial w_{kj}^{21}}$$

4. **Weight update:** Use gradient descent with a fixed learning rate $\eta_t = 0.5$ to update the weights in each iteration according to

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \Delta \boldsymbol{w}^{(t)}$$

Stop the iterative weight updates if the error E^T has converged, i.e. $|\Delta E^T|/E^T$ has fallen below some small value (e.g. 10^{-5}) or a maximum number of iterations $t_{max} = 3000$ has been reached.

(a) (2 point) Plot the error E^T over the iterations.

(b) (1 point) For the final network, plot the activations of hidden units for all inputs.

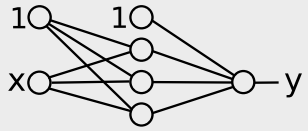
(c) (1 point) Plot the output values over the input space (i.e. the input-output function of the network) together with the training dataset.

(d) (2 point) Plot (a)–(c) *twice* next to each other and discuss: is there a difference, and if so, why?

(e) (1 point) What might have been the motivation for using a quadratic error function here?

Solution

For simplicity, we will call the weights for the hidden layer $\underline{\mathbf{W}}$ and for the output neuron $\underline{\mathbf{v}}$.



$\underline{\mathrm{W}} \in \mathbb{R}^{3\times 2}$ $\underline{\mathrm{v}} \in \mathbb{R}^{1\times 4}$

The output is therefore:

$$y(x) = \underline{\mathbf{v}}\begin{bmatrix}1\\ f\big(\underline{\mathbf{h}}(x)\big)\end{bmatrix}, \qquad \text{with} \qquad \underline{\mathbf{h}}(x) = \underline{\mathbf{W}}\begin{bmatrix}1\\ x\end{bmatrix}$$

The derivatives follow the chain rule:

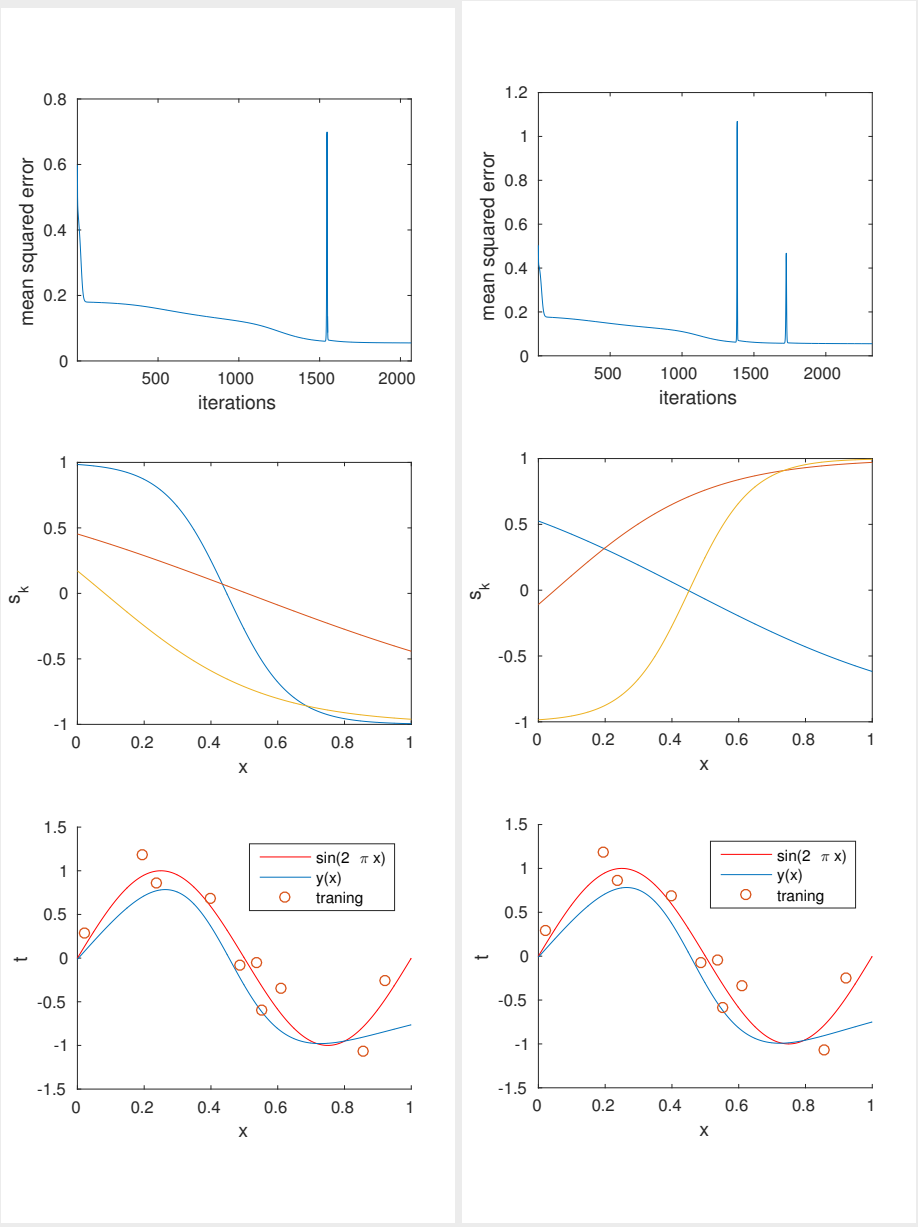
$$\frac{\partial e_n^T}{\partial v_i} = \frac{\partial e_n^T}{\partial y(x_n)}\frac{\partial y(x_n)}{\partial v_i} = \Big(y(x_n) - t_n\Big)\begin{bmatrix}1\\ f\big(\underline{\mathbf{h}}(x_n)\big)\end{bmatrix}_i,$$

$$\frac{\partial e_n^T}{\partial W_{ij}} = \frac{\partial e_n^T}{\partial y(x_n)}\frac{\partial y(x_n)}{\partial h_i(x_n)}\frac{\partial h_i(x_n)}{\partial W_{ij}} = \Big(y(x_n) - t_n\Big)\, v_i \Big(\underbrace{1 - f^2\big(h_i(x_n)\big)}_{f'\left(h_i(x_n)\right)}\Big)\begin{bmatrix}1\\ x_n\end{bmatrix}_j.$$

Let in the following $\underline{\mathbf{x}} = [x_1, \ldots, x_N]$, $\underline{\mathbf{t}} = [x_1, \ldots, t_N]$, $\underline{\mathbf{H}} = \underline{\mathbf{W}} \begin{bmatrix} 1 \\ \underline{\mathbf{x}} \end{bmatrix}$ and $\underline{\mathbf{y}} = \underline{\mathbf{v}} \begin{bmatrix} 1 \\ f(\underline{\mathbf{H}}) \end{bmatrix}$.
The average over above derivatives is in matrix notation:

$$\Delta \underline{\mathbf{v}} = -\frac{1}{N} \begin{bmatrix} 1 \\ f(\underline{\mathbf{H}}) \end{bmatrix} (\underline{\mathbf{y}} - \underline{\mathbf{t}})^\top, \qquad \text{and} \qquad \Delta \underline{\mathbf{W}} = -\frac{1}{N} \underline{\mathbf{D}}_{(\underline{\mathbf{v}}_{2:4})} \left(1 - f^2(\underline{\mathbf{H}})\right) \underline{\mathbf{D}}_{(\underline{\mathbf{y}} - \underline{\mathbf{t}})} \begin{bmatrix} 1 \\ \underline{\mathbf{x}} \end{bmatrix}^\top.$$

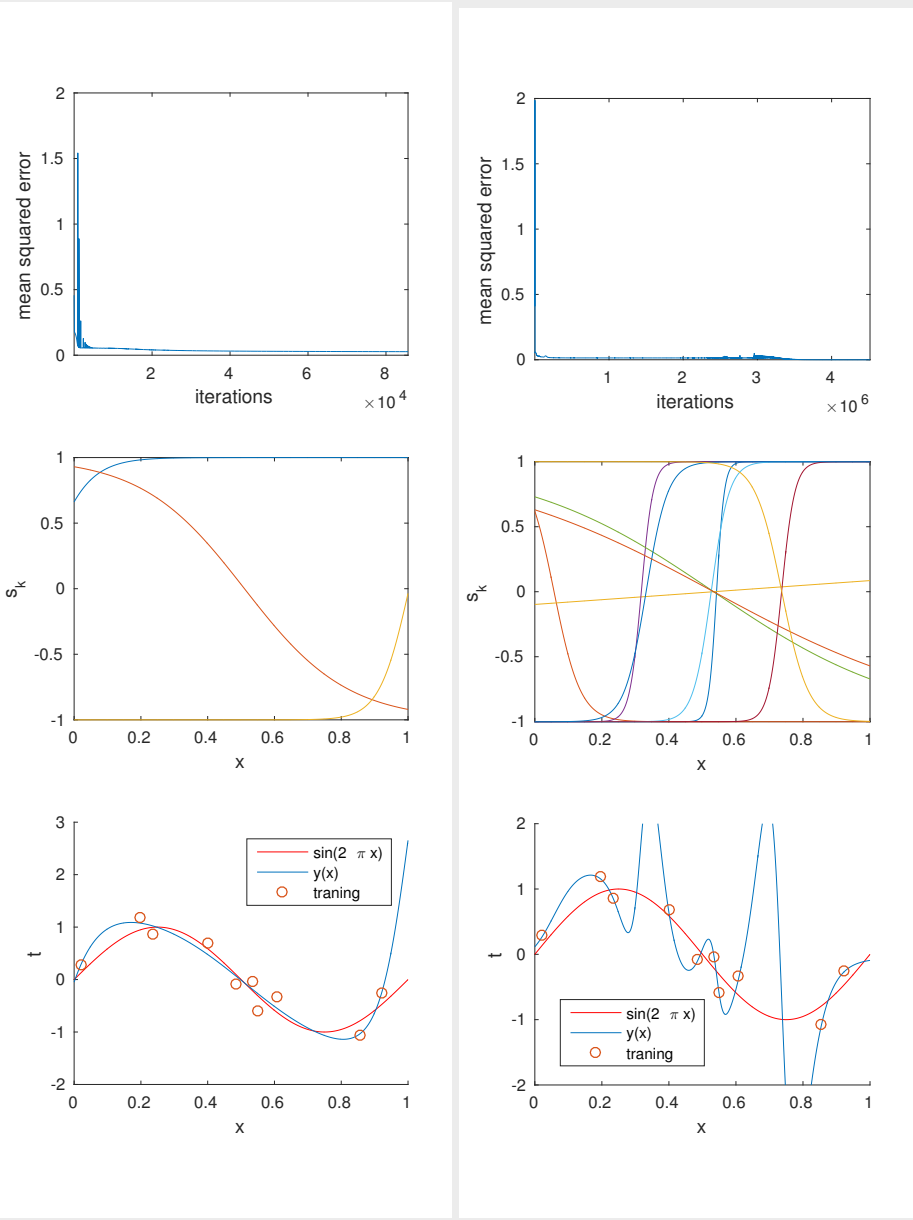
Here $\underline{\mathbf{D}}_{(\underline{\mathbf{x}})}$ denotes the diagonal matrix with diagonal elements from vector $\underline{\mathbf{x}}$ and $\underline{\mathbf{v}}_{a:b}$ refers to the enties a to b of vector $\underline{\mathbf{v}}$.



(d) (i) Different initializations require different time to converge (upper plots). There may also be some spikes in the cost due to a large learning rate η_t. (ii) Neurons can take on differet roles, due to initialization (middle plots). (iii) Between training samples the learned function can vary (lower plots).

(e) The mean squared error corresponds to the additive Gaussian noise to the labels.

If you decide to play around some more with the code: the approximation quality increases significantly with the number of iterations (left plot). However, increasing the number of hidden neurons (to 10 in the right plot) leads to massive over-fitting!



Total 10 points.