Machine Intelligence 1 — **Exercise Sheet 3**

WS 2021/22, Obermayer/Kashef — due: 15.11.2021 at 23:55

Multilayer Perceptrons and Backpropagation Algorithm

Exercise T3.1: Cost functions (tutorial)

(a) What effect will the choice of error measure (particularly *quadratic or linear*) produce?

(b) Outline the relation between the quadratic error function and the Gaussian conditional distribution for the labels.

(c) Derive a suitable error function (*cross entropy*) for the following case: the output of a neural network is interpreted as the probability that the input belongs to the first of two classes.

(d) Summarize the error measures and output layers for regression and classification.

Exercise T3.2: Parameter optimization (tutorial)

(a) Recap MLP architecture, outline gradient descent, and derive the back propagation algorithm (backprop) for a MLP with L layers.

(b) Discuss the consequence of parameter space symmetries: (i) permutation of neuron indices within a layer, (ii) reversal of signs across consecutive layers.

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

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

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

with

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

(a) (1 point) Show that

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

(b) (1 point) Consider an MLP with one hidden layer. The nonlinear transfer function for the output neuron $(i=1, v=2)$ is assumed to be

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

where h_1^2 is the total input[1] of the output neuron. Show that its derivative can be expressed as

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

[1]The total input of a neuron is sometimes referred to as a *logit*

(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 $(i = 1, v = 2)$ and neuron j of the hidden layer $(j > 0, v = 1)$ is

$$\frac{\partial e^{(\alpha)}}{\partial w_{1j}^{21}} = \left(y(\underline{\mathbf{x}}^{(\alpha)}; \underline{\mathbf{w}}) - y_T^{(\alpha)}\right) f(h_j^1).$$

Solution

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

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

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

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

(c) Let $e := e^{(\alpha)}$, $y_T := y_T^{(\alpha)}$, $w := w_{1j}^{21}$, $h := h_1^2$ and $y := y(\underline{\mathbf{x}}; \underline{\mathbf{w}})$
and note that for N_{hid} hidden neurons
$h = w_{10}^{21} + \sum_{j=1}^{N_{\text{hid}}} w_{1j}^{21} f(h_j^1)$ and
$y = f_1^2(h)$:

$$\frac{\partial e}{\partial w} = \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial w} = \frac{y - y_T}{y(1-y)} \cdot y\left(1-y\right) \cdot f(h_j^1) = \left(y - y_T\right) f(h_j^1).$$

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

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

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

(A) **Initialization:**

1. Construct the MLP using a single hidden layer with 3 hidden nodes ($N_1 = 3$) and an output layer with a single output neuron ($N_L = N_2 = 1$).
2. Use the $\tanh$ transfer function for the hidden neurons and the linear transfer function (i.e. the identity) for the output neuron.

[2] Normally, you would not have access to such information and your training algorithm will not make direct use of it how the data was generated.

3. Set the weights and biases to random values from the interval $[-0.5, 0.5]$.

(B) **Iterative learning:**

1. For each input value $x^{(\alpha)}$ of the training set, do:
   (a) **Forward Propagation:** Calculate the total input and activity of the hidden neurons and the output neuron.
   (b) Compute the **output error** $e^{(\alpha)}$ using the quadratic error cost function.
   (c) **Backpropagation:** Calculate the "local errors" δ_i^v for the output and the hidden layer for each training point.
   (d) Calculate the gradient of the error function w.r.t. the first and second layer weights $w_{ij}^{1\,0}$ and $w_{ij}^{2\,1}$ respectively [3].
2. Calculate the batch gradient in order to obtain the direction of the weight updates:

$$\Delta w_{ij}^{v'v} = -\frac{\partial E^T_{[\mathbf{w}]}}{\partial w_{ij}^{v'v}} = -\frac{1}{p}\sum_{\alpha=1}^{p} \frac{\partial e^{(\alpha)}_{[\mathbf{w}]}}{\partial w_{ij}^{v'v}}$$

   where $j = 0, \dots, N_v$ and $i = 1, \dots, N_{v'}$
3. **Weight update:** Use gradient descent with a fixed learning rate $\eta = 0.5$ to update the weights in each iteration according to

$$\underline{\mathbf{w}}^{(t+1)} = \underline{\mathbf{w}}^{(t)} + \eta\Delta\underline{\mathbf{w}}^{(t)}$$

(C) **Stopping criterion**:
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.

Devliverables:

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

(b) (1 point) For the final network, plot the output 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* (i.e., for different initial conditions) 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?

Hints:

(1) Modularizing your code can help narrow down potetnial mistakes in your implementation. One way to go about it is to have separate functions for the forward propagation, calculating the error, calculating the different terms that make up the gradients (error funciton, non-linearity). This can help verifying the different parts of your code.

[3] The weights in the second layer are those connecting the hidden neurons with the output node. Since we only have one output neuron, $w_{ij}^{2\,1}$ is effectively $w_{1j}^{2\,1}$

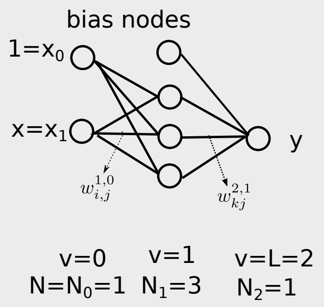
(2) Making use of broadcasting in numpy and turning everything into a matrix multiplication will reduce the number of lines in your code. Matrix multuplication can also speed up your training process. But some of these code optimizations could make your code less readable and harder to debug.

(3) Don't forget to update the bias weights in your network. Every hidden neuron and the output neuron has a bias weight.

(4) Keep in mind that the transfer function applied to the bias value is the identity function. Don't apply the non-linear transfer function to the bias nodes.

(5) A sudden "spike" in your error plot is ok.

Solution



1. **Forward Propagation:**

```
Given $\underline{\mathbf{x}}^{(\alpha)}, y_T^{(\alpha)}, \underline{\mathbf{w}} = \{w_{ij}^{v'v}\} = \{w_{ij}^{v'\,v'-1}\}$
 where $\alpha = 1,\ldots,p,\quad v' = 1,2$ and $j = 0,\ldots,N_v,\quad i = 1,\ldots,N_{v'}$
$h_1^0$ := $x_1^{(\alpha)}$    // input
$f_i^{v'}(h_i^{v'})$ := $f^{v'}(h_i^{v'})$    // simplify notation, $f$ is the same for all neurons in the same layer
$f^0(h_i^0)$ := $h_i^0$    $\forall i = 1,\ldots,N_0$    // identity function for input nodes
$h_0^v$ := 1    $\forall v = 0,\ldots,L-1$    // bias nodes
for $v' = 1,2$ do
    $\mu$ := $v' - 1$    // index for preceeding layer
    $h_i^{v'}$ := $w_{i0}^{v'\mu} + \sum_{j=1}^{N_\mu} w_{ij}^{v'\mu} f^\mu(h_j^\mu)$    $\forall i = 1,\ldots,N_{v'}$
end
```

2. **Output error:**

$$E^T = \frac{1}{p}\sum_{\alpha=1}^{p} e^{(\alpha)} = \frac{1}{2p}\sum_{\alpha=1}^{p}\left(y(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}) - y_T^{(\alpha)}\right)^2$$

3. **Backpropagation:**
Local error of the output neuron (linear neuron),

$$\delta_1^L = \delta_1^2 = [f_1^2]'(h_1^2) = 1$$

Local error of hidden neurons (v'=1),

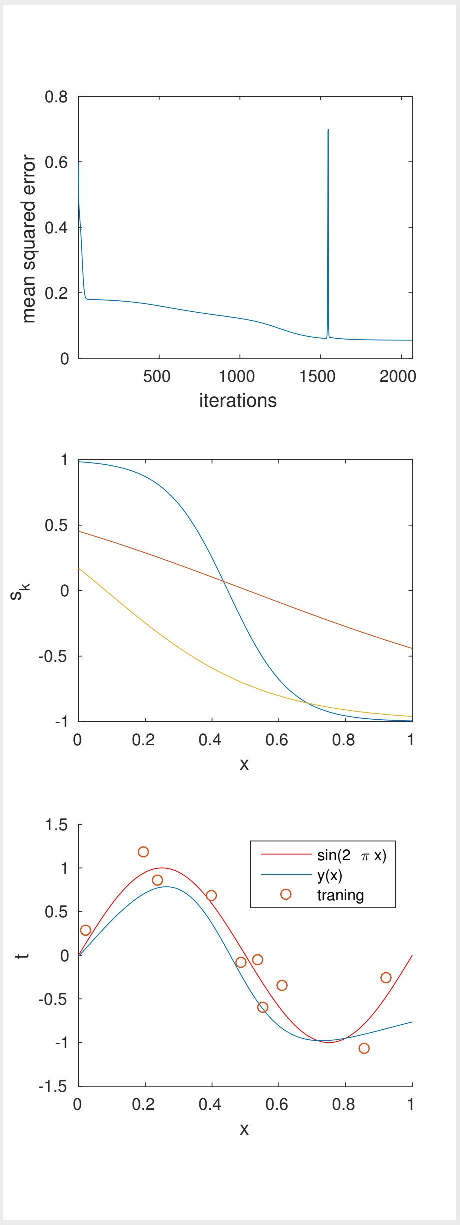
$$\delta_i^1 = [f^1]'(h_i^1)\sum_{k=1}^{N_2}\delta_k^2 w_{k\,i}^{2\,1} \quad \text{where} \quad i = 1,2,3$$

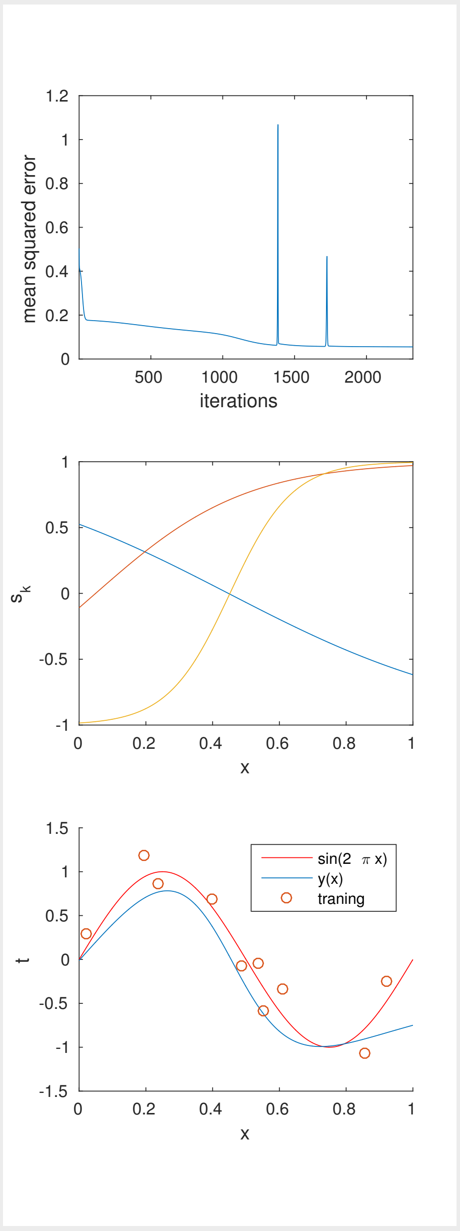
(The index i is omitted from f_i^1, since it is identical for all neurons **within** the same layer v' - except for bias nodes)
substituting activation function $f^1 = \tanh(\cdot)$ in the above equation,

$$\delta_i^1 = \left(1 - \tanh^2(h_i^1)\right)\sum_{k=1}^{N_2}\delta_k^2 w_{k,i}^{2,1} \quad \text{for } i = 1,2,3$$

$$\frac{\partial y}{\partial w_{ij}^{v'v}} = \begin{cases}\delta_i^{v'} & \text{for } j = 0\\ \delta_i^{v'} f^v(h_j^v) & \text{for } v' = 1,2 \quad j = 1,\ldots,N_v, \quad i = 1,\ldots,N_{v'}\end{cases}$$

$$\frac{\partial e^{(\alpha)}}{\partial w_{i\,j}^{v'v}} = \left(y(\underline{\mathbf{x}}^{(\alpha)};\underline{\mathbf{w}}) - y_T^{(\alpha)}\right)\frac{\partial y}{\partial w_{i\,j}^{v'v}}$$

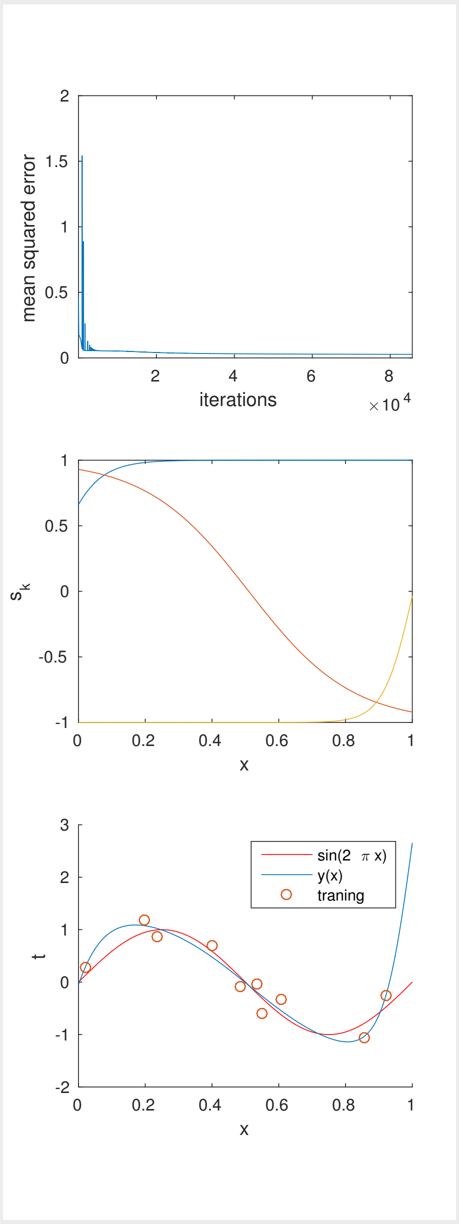


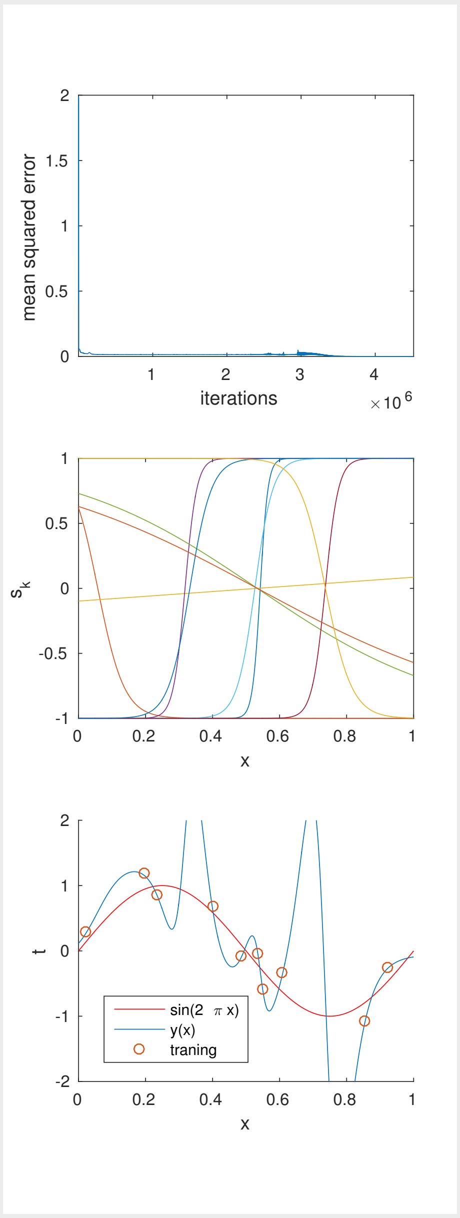


(d) (i) Different initializations require different time to converge (upper plots). There may also be some spikes in the cost due to a large learning rate η_t. (ii) Neurons can take on differet roles, due to initialization (middle plots). (iii) Between training samples the learned function can vary (lower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