Artificial Neural Networks (Gerstner). Solutions for week 5

Deep Learning 1 - BackProp

Exercise 1. Derive BackProp

A network with 2 hidden layers followed by an output layer has the output in the final layer (=3rd layer)

$$\begin{aligned}
\hat{y}_n^\mu &= x_n^{(3)} && (1)\\
&= g^{(3)}[a_n^{(3)}] && (2)\\
&= g^{(3)}[\sum_i w_{ni}^{(3)} g^{(2)}[a_i^{(2)}]] && (3)\\
&= g^{(3)}[\sum_i w_{ni}^{(3)}[g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}]] && (4)\\
&= g^{(3)}[\sum_i w_{ni}^{(3)}[g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})]] && (5)\\
&= g^{(3)}[\sum_i w_{ni}^{(3)}[g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^\mu)]] && (6)
\end{aligned}$$

We use an error function

$$E(\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \boldsymbol{w}^{(3)}) = \frac{1}{2}\sum_\mu \sum_n [t_n^\mu - \hat{y}_n^\mu]^2 \quad (7)$$

Calculate the derivative $\partial E / \partial w_{52}^{(1)}$, hence the derivative with respect to the weight connecting the second input to the fifth hidden neuron in the first layer.

Solution:

For convenience of calculations, we define the online error function of a single output unit as

$$\tilde{E}(\mu, n) = \tilde{E} = \frac{1}{2}\left[t_n^\mu - \hat{y}_n^\mu\right]^2, \quad (8)$$

and a useful quantity, the delta error for unit k in layer l as

$$\delta_k^{(l)} = \frac{\partial \tilde{E}}{\partial a_k^{(l)}}. \quad (9)$$

Starting with the output layer, we can compute

$$\frac{\partial \tilde{E}}{\partial w_{ni}^{(3)}} = \frac{\partial \tilde{E}}{\partial a_n^{(3)}} \frac{\partial a_n^{(3)}}{\partial w_{ni}^{(3)}} = \delta_n^{(3)} x_i^{(2)}. \quad (10)$$

Note that at the output layer $\delta_n^{(3)}$ is defined by

$$\frac{\partial \tilde{E}}{\partial a_n^{(3)}} = \frac{\partial}{\partial a_n^{(3)}} \frac{1}{2}\left[t_n^\mu - g^{(3)}[a_n^{(3)}]\right]^2 = -\left[t_n^\mu - g^{(3)}[a_n^{(3)}]\right] g'^{(3)}[a_n^{(3)}] \quad (11)$$

This error can be back propagated with the chain rule to the downstream layers as

$$\frac{\partial \tilde{E}}{\partial w_{ij}^{(l)}} = \frac{\partial \tilde{E}}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} x_j^{(l-1)}, \quad (12)$$

where

$$\delta_i^{(l)} = \frac{\partial \tilde{E}}{\partial a_i^{(l)}} = \sum_j \delta_j^{(l+1)} \frac{\partial a_j^{(l+1)}}{\partial a_i^{(l)}} = \sum_j \delta_j^{(l+1)} w_{ji}^{(l+1)} g'^{(l)}[a_i^{(l)}] \quad (13)$$

For the last equality we used the definition of $a_n^{(l+1)} = \sum_k w_{nk}^{(l+1)} g^{(l)}[a_k^{(l)}]$ and the chain rule to see that $\frac{\partial a_j^{(l+1)}}{\partial a_i^{(l)}} = w_{ji}^{(l+1)} g'^{(l)}[a_i^{(l)}]$. Therefore

$$\frac{\partial E}{\partial w_{52}^{(1)}} = \delta_5^{(1)} x_2^\mu, \tag{14}$$

where

$$\begin{aligned}
\delta_5^{(1)} &= \sum_j \delta_j^{(2)} w_{j5}^{(2)} g'^{(1)}[a_5^{(1)}] && (15)\\
\delta_j^{(2)} &= \sum_n \delta_n^{(3)} w_{nj}^{(3)} g'^{(2)}[a_j^{(2)}] && (16)\\
\delta_n^{(3)} &= \frac{\partial \tilde{E}}{\partial a_n^{(3)}} = -\left[t_n^\mu - g^{(3)}[a_n^{(3)}]\right] g'^{(3)}[a_n^{(3)}] && (17)
\end{aligned}$$

Exercise 2. Algorithmic complexity of Backprop

a. Estimate the number of steps for one loop of gradient calculations with Backprop for a network with one hidden layer (applying each pattern once) with $P = 2000$ patterns, where N_{in} is the dimensionality of the input space, N_{h} is the number of hidden neurons, and $N_{\text{out}} = 5$ is the dimensionality of the output space.

   (To do so, imagine that you have to write a program for Backprop. Sketch the steps that must be taken. Count each multiplication and each summation as one step and each evaluation of the gain function g or its derivative g' as one step. Ignore biases for simplicity.)

   Show that for large networks, the algorithm scales linearly with the number of weights.

b. Let us assume your laptop makes 10^{10} floating point operations per second. What is the minimal time needed for one cycle if $N_{\text{in}} = N_{\text{h}} = 100$?

   Moreover, BackProp often needs more than 10 000 cycles (presentation of each pattern once) to find the minimum - do you want to wait in front of the machine or is there time to have a coffee break?

c. Finally, let us assume that the input consists of pixel patterns with 256x256 pixels. You have two hidden layers of 1000 units each. Calculate the number of time steps for one loop. Also calculate the number of weights.

   Consider now each weight as a free parameter which has to be estimated from the data. How many data samples do you need to estimate (= optimize) all weights of your network?

d. In a realistic application you would have to optimize the number of hidden neurons by testing the performance for different network sizes. For the test you apply previously unknown patterns (which were not used during training) and you take the network which finally performs best. Let us assume that in addition to the 2000 training patterns you have also 2000 test patterns (both of dimension 256x256). How many networks (of what size of hidden neurons) can you train and test in 48 hours?

Solution:

a. The multiplication of a $N \times M$-matrix with a M-vector requires NM multiplications and $N(M-1)$ summations.

   The forward direction thus requires $N_{\text{h}}N_{\text{in}} + N_{\text{out}}N_{\text{h}}$ multiplications, $N_{\text{h}}(N_{\text{in}}-1)+N_{\text{out}}(N_{\text{h}}-1)$ summations and $N_{\text{h}} + N_{\text{out}}$ evaluations of the gain function; in total, $2(N_{\text{h}}N_{\text{in}} + N_{\text{out}}N_{\text{h}})$ steps per pattern.

In the backward direction, the matrix multiplication of the δ requires $N_\text{h}N_\text{out}$ multiplications and $N_\text{h}(N_\text{out}-1)$ summations. $N_\text{out}+N_\text{h}$ evaluations of g' and $N_\text{out}+N_\text{h}$ multiplications of g' with δs are needed. Finally, $N_\text{h}N_\text{out}+N_\text{in}N_\text{h}$ multiplications are needed for the outer-products of δs and inputs, and $N_\text{h}N_\text{out}+N_\text{in}N_\text{h}$ multiplications for the product with the learning rate and $N_\text{h}N_\text{out}+N_\text{in}N_\text{h}$ summations for finally adding these values to the old weights. Thus, the backward direction requires in total $2N_\text{h}N_\text{out}+2N_\text{out}+N_\text{h}+3(N_\text{h}N_\text{out}+N_\text{in}N_\text{h})$ steps per pattern.

Summing the number of steps of the forward and backward direction we arrive at a total of $7N_\text{out}N_\text{h}+5N_\text{h}N_\text{in}+2N_\text{out}+N_\text{h}$ steps per pattern, or $P\big(7N_\text{out}N_\text{h}+5N_\text{h}N_\text{in}+2N_\text{out}+N_\text{h}\big)$ in total.

The total number of weights is $N=N_\text{out}N_\text{h}+N_\text{h}N_\text{in}$. Thus Backprop is $\mathcal{O}(N)$.

b. $2000(7\cdot5\cdot100+5\cdot100\cdot100+10+100)/(10^{10}$ flop/s$)\approx 0.011$ s. For 10 000 cycles one has time for a quick coffee of 110 s.

   Note that the time needed may be considerably higher as it does not solely depend on the number of floating point operations per second.

c. $N_\text{in}=65\ 536$, $N_\text{h}=N_{\text{h}_1}=N_{\text{h}_2}=1000$, $N_\text{out}=5$. The number of steps per pattern is $7N_\text{out}N_\text{h}+7N_\text{h}^2+5N_\text{h}N_\text{in}+2N_\text{out}+2N_\text{h}=334\ 717\ 010$.

   The number of parameters is $N_\text{in}N_\text{h}+N_\text{h}^2+N_\text{h}N_\text{out}=66\ 541\ 000$.

   One training sample corresponds to one (equality or inequality) constraint. To constrain N parameters to unique values one needs N equality constraints. With inequality constraints the solution space is usually larger than a single point. Thus, if we want to have at least as many training samples as parameters, we should have at least 66 541 000 samples. However, given appropriate regularization, the model may perform well on the test set, even if much fewer samples are used for training.

d. Assuming again 10^{10} flop/s and 1000 epochs over the 2000 training samples, the network with 2 layers of each 1000 hidden of the previous exercise takes $334\ 717\ 010\cdot 2\ 000\cdot 1\ 000/(10^{10}$ flop/s$)\approx$ 67 000 s $\approx$ 18.6 h for training but it takes basically no time ($\approx$ 26 sec) for testing on 2000 test patterns.

   A network with only one hidden layer of 1000 neurons is not considerably faster ($\approx$ 18 h of training), since most time is spent on the first layer with the huge input dimensionality. But if the first hidden layer has only 100 neurons, training time reduces to a few hours, even if multiple layers are used.

Exercise 3. Error functions for Backprop

Several researchers have tried to use error functions other than the quadratic error.

a. Convince yourself that a change of the error function only affects the δs of the output layer. More precisely, the formula for the output δs in the algorithm has to be changed, but the formula for the back-propagation step remains the same.

b. To be specific, consider error functions of the type

$$E_p=\sum_\mu\sum_i|t_i^\mu-\hat{y}_i^\mu|^p$$

   where the index i runs over all output neurons and $\hat{y}_i=g^{(n)}(a_i)$. What are the output δs for quadratic error ($p=2$)? What are the δs for linear error ($p=1$)? Note that you have to keep track of the absolute value signs.

c. Calculate the output δ for the 'cross-entropy' error function which will be introduced next week in the context of maximum likelihood of statistical learning

$$E_{log} = \sum_\mu \sum_i \left[t_i^\mu \text{log}\frac{t_i^\mu}{\hat{y}_i^\mu} + (1 - t_i^\mu)\text{log}\frac{1 - t_i^\mu}{1 - \hat{y}_i^\mu} \right] .$$

Here, $t_i^\mu = 0$ means 'target output feature i not present for input μ' and $t_i^\mu = 1$ means 'feature present'. The outputs $\hat{y}_i^\mu = 1$ can be interpreted as the probability of 'output feature i is present' for input μ. $\hat{y}_i^\mu = 1$ means that the hypothesis 'feature i is present' is definitely true.

d. Assume the cross-entropy on the transfer function

$$g^{(n)}(a) = \frac{1}{2}[1 + \tanh(a)] \tag{18}$$

and show that the δs of the output layer are simply $\delta_i^\mu = -2[t_i^\mu - \hat{y}_i^\mu]$.

e. Relate the gain function g in (d) to the sigmoidal function in class.

Solution:

a. From the lecture slides and the chain rule, we see that $\delta_i^{(\mu,n_{\max})} = E'(\hat{y}_i^\mu) \cdot g'(a_i^{(\mu,n_{\max})})$. The error function does not appear explicitly in the formula for δ before the output layer; only implicitly through the change in $\delta_i^{(\mu,n_{\max})}$. Once we have recomputed $\delta_i^{(\mu,n_{\max})}$ and stored the result, it can be substituted into the formula for all of the other δs.

b. In general the error function gives

$$\begin{aligned} E_p(\hat{y}_i^\mu) &= |t_i^\mu - \hat{y}_i^\mu|^p \\ E_p'(\hat{y}_i^\mu) &= -p \cdot (t_i^\mu - \hat{y}_i^\mu) \cdot |t_i^\mu - \hat{y}_i^\mu|^{p-2} \end{aligned}$$

For quadratic error, this reduces to

$$\begin{aligned} E_2'(\hat{y}_i^\mu) &= -2(t_i^\mu - \hat{y}_i^\mu) \\ \delta_i^\mu &= -2(t_i^\mu - \hat{y}_i^\mu) \cdot g'(a_i^{(n_{\max})}), \end{aligned}$$

and for linear error

$$\begin{aligned} E_1'(\hat{y}_i^\mu) &= -1(t_i^\mu - \hat{y}_i^\mu) \cdot |t_i^\mu - \hat{y}_i^\mu|^{-1} \\ \delta_i^\mu &= -(t_i^\mu - \hat{y}_i^\mu) \cdot |t_i^\mu - \hat{y}_i^\mu|^{-1} \cdot g'(a_i^{(n_{\max})}) \end{aligned}$$

c. For the cross–entropy function we have

$$\begin{aligned} E_{log} &= \sum_\mu \sum_i \left[t_i^\mu \text{log}\frac{t_i^\mu}{\hat{y}_i^\mu} + (1 - t_i^\mu)\text{log}\frac{1 - t_i^\mu}{1 - \hat{y}_i^\mu} \right] \\ E_{log} &= \sum_\mu \sum_i [t_i^\mu \text{log} t_i^\mu - t_i^\mu \text{log}\hat{y}_i^\mu + (1 - t_i^\mu)\text{log}(1 - t_i^\mu) - (1 - t_i^\mu)\text{log}(1 - \hat{y}_i^\mu)] \\ E_{log}'(\hat{y}_i^\mu) &= -t_i^\mu/\hat{y}_i^\mu + (1 - t_i^\mu)/(1 - \hat{y}_i^\mu) \\ \delta_i^{(n_{\max})} &= -\left[t_i^\mu/\hat{y}_i^\mu - (1 - t_i^\mu)/(1 - \hat{y}_i^\mu)\right] \cdot g'(a_i^{(n_{\max})}) \end{aligned}$$

d. For this transfer function we find that

$$\begin{aligned}
g(a_i^\mu) &= \hat{y}_i^\mu \\
&= \frac{1}{2}[1+\tanh(a_i^\mu)] \\
g'(a_i^\mu) &= \frac{1}{2}[1-\tanh^2(a_i^\mu)] \\
&= \hat{y}_i^\mu \cdot (1-\tanh(a_i)) \\
\delta_i^{(\mu,n_{\max})} &= -\left[t_i^\mu/\hat{y}_i^\mu - (1-t_i^\mu)/(1-\hat{y}_i^\mu)\right] \cdot \hat{y}_i^\mu \cdot (1-\tanh(a_i)) \\
\delta_i^{(\mu,n_{\max})} &= -\left[t_i^\mu - (1-t_i^\mu)\cdot\frac{\hat{y}_i^\mu}{1-\hat{y}_i^\mu}\right]\cdot(1-\tanh(a_i)) \\
\delta_i^{(\mu,n_{\max})} &= -\left[t_i^\mu - (1-t_i^\mu)\cdot\frac{1+\tanh(a_i)}{1-\tanh(a_i)}\right]\cdot(1-\tanh(a_i)) \\
\delta_i^{(\mu,n_{\max})} &= -\left[t_i^\mu(1-\tanh(a_i)) - (1-t_i^\mu)(1+\tanh(a_i))\right] \\
\delta_i^{(\mu,n_{\max})} &= -\left[t_i^\mu - t_i^\mu\tanh(a_i) - 1 - \tanh(a_i) + t_i^\mu + t_i^\mu\tanh(a_i)\right] \\
\delta_i^{(\mu,n_{\max})} &= -2\left[t_i^\mu - \hat{y}_i^\mu\right]
\end{aligned}$$

e. From the definition in (d) we have

$$\begin{aligned}
g(x) &= \frac{1}{2}[1+\tanh(x)] \\
g(x) &= \frac{1}{2}\left[1+\frac{e^x-e^{-x}}{e^x+e^{-x}}\right] \\
g(x) &= \frac{1}{2}\left[\frac{e^x+e^{-x}}{e^x+e^{-x}}+\frac{e^x-e^{-x}}{e^x+e^{-x}}\right] \\
g(x) &= \frac{e^x}{e^x+e^{-x}} \\
g(x) &= \frac{1}{1+e^{-2x}} \\
g(x) &= \sigma(2x)
\end{aligned}$$

where σ is the sigmoid function.

Exercise 4. BackProp with Gaussian units in the first layer (from exam 2018)

Our data base $(\boldsymbol{x}^\mu, \boldsymbol{t}^\mu)$ has N-dimensional input and N-dimensional target output. We have a network with two hidden layers.

Use a quadratic error function for each pattern μ

$$E(\mu) = \frac{1}{2}\sum_{m=1}^{N}[t_m^\mu - \hat{y}_m^\mu]^2,$$

where $\hat{y}_m^\mu = \sum_{k=1}^{K} w_{mk}^{(3)} x_k^{(2)}$.

The second hidden layer has normal sigmoidal units with gain function $g(a)$, while the first hidden layer contains Gaussian basis functions. Thresholds are implicit (by adding an extra unit) and will not be treated explicitly.

Thus

$$x_k^{(2)} = g\big(\sum_{j=1}^{J} w_{kj}^{(2)} \exp[-0.5||\boldsymbol{x}^\mu - \boldsymbol{c}_j||^2]\big),$$

where $\boldsymbol{c}_j$ is the center of the Gaussian of unit j.

Our aim is to calculate the derivative of the error function with respect to the parameters c_{ji}, that is, component i of Gaussian unit j.

A direct calculation with the chain rule yields that the derivative with respect to c_{45} is

$$\frac{dE(\mu)}{dc_{45}} = -\sum_{m=1}^{N} [t_m^\mu - \hat{y}_m^\mu] \sum_{k=1}^{K} w_{mk}^{(3)} g'(a_k) w_{k4}^{(2)} (x_5^\mu - c_{45}) \exp[-0.5||\boldsymbol{x}^\mu - \boldsymbol{c}_j||^2]$$

Reorder the terms of the gradient calculation for arbitrary c_{ij} so as to arrive at an efficient backpropagation algorithm with a forward pass and a backward pass and an update step for the parameters c_{ij}. Summarize your results in pseudo-code.

Solution:

a. Initialization

   Parameters $w_{ij}^{(n)}$ and c_{ij} are initialized at small values $[-\epsilon, \epsilon]$.

b. Forward pass

   Pattern $\boldsymbol{x}^\mu$ is applied at the input

$$\begin{aligned} x_j^{(1)} &= \exp[-0.5||\boldsymbol{x}^\mu - \boldsymbol{c}_j||^2] \\ a_k^{(2)} &= \sum_{j=1}^{J} w_{kj}^{(2)} x_j^{(1)} \\ x_k^{(2)} &= g(a_k^{(2)}) \\ \hat{y}_m^\mu &= \sum_{k=1}^{K} w_{mk}^{(3)} x_k^{(2)} \end{aligned}$$

c. Backward pass

$$\begin{aligned} \delta_m^{(3)} &= -(t_m^\mu - \hat{y}_m^\mu) \\ \delta_k^{(2)} &= g'(a_k^{(2)}) \sum_{m=1}^{N} w_{mk}^{(3)} \delta_m^{(3)} \\ \delta_j^{(1)} &= x_j^{(1)} \sum_{k=1}^{K} w_{kj}^{(2)} \delta_k^{(2)} \end{aligned}$$

d. Update of parameters c_{ij}

$$\Delta c_{ji} = -\eta \delta_j^{(1)} (x_i^\mu - c_{ji})$$