Problems for practice in FYTN14/EXTQ40/NTF005F.
Problems marked with asterisk introduce side topics.

Chapter: Introduction

1.1 It is often useful to express the derivative $\varphi'(a)$ of activation functions in terms of $\varphi(a)$ itself. Do so for the following common activation functions.

a) $\varphi(a) = a$

b) $\varphi(a) = \frac{1}{1+\exp(-a)}$

c) $\varphi(a) = \tanh(a)$

d) $\varphi(a) = \max\{0, a\}$ (ignore $a = 0$)

e) $\varphi(a) = \ln[1 + \exp(a)]$

Answer:

a) $\varphi'(a) = 1$

b) $\varphi'(a) = \varphi[1 - \varphi]$

c) $\varphi'(a) = 1 - \varphi^2$

d) $\varphi'(a) = 1$ if $\varphi > 0$, and 0 otherwise. (Ignored $a = 0$. With this formulation it happened to become the left-side derivative. Alternative is $\varphi' = 1$ if $\varphi \geq 0$.)

e) $\varphi'(a) = 1 - \exp(-\varphi)$

1.2 The logistic function $\varphi(a) = \frac{1}{1+\exp(-a)}$ and the hyperbolic tangent $\varphi(a) = \tanh(a)$ are two common sigmoidal activation functions. Consider a similar function $f(a) = \varphi(\beta a)$, where β is a constant. For the two sigmoidal cases, what is the slope $f'(a)$ for $a = 0$?

Answer:

logistic $\varphi(0) = 1/2$, $\varphi'(0) = 1/4$, $f'(0) = \beta/4$.

tanh $\varphi(0) = 0$, $\varphi'(0) = 1$, $f'(0) = \beta$.

1.3 *

Yet another odd sigmoidal function is the algebraic sigmoid:

$$\varphi(a) = \frac{a}{\sqrt{1+a^2}}$$

Compute the derivative of $\varphi(a)$ with respect to a and express it in terms of $\varphi(a)$. What is the value of $\varphi'(a)$ at the origin?

Answer:
Start with $\varphi(a) = \frac{a}{\sqrt{1+a^2}}$. It can be inverted to $a = \frac{\varphi}{\sqrt{1-\varphi^2}}$.

$$\begin{aligned}
\varphi'(a) &= \frac{1}{(1+a^2)^{1/2}} + \frac{a}{(1+a^2)^{3/2}}\left(-\frac{1}{2}\right)2a = \\
&= \frac{1}{(1+a^2)^{1/2}} - \frac{a^2}{(1+a^2)^{3/2}} = \frac{(1+a^2)-a^2}{(1+a^2)^{3/2}} \\
&= \frac{1}{(1+a^2)^{3/2}} = \frac{\varphi^3}{a^3} \\
&= \left(1-\varphi^2\right)^{3/2}
\end{aligned}$$

And $\varphi(0) = 0$ gives $\varphi'(0) = 1$.

1.4 Suppose we have an activation function as the f defined in Problem 2.2, where the parameter β governs the slope of the sigmoidal function and a is e.g. a weighted sum of inputs. We would like to absorb the β parameter into a. How can we modify either the weights $(\omega_1, ..., \omega_N)$ or the inputs $(x_1, ..., x_N)$ to accomplish this?

Answer:

$$f(a) = \frac{1}{1+\exp(-\beta a)}$$

$$a = \sum_k x_k \omega_k \Rightarrow$$

$$f(a) = \frac{1}{1+\exp(-\beta \sum_k x_k \omega_k)}$$

We can absorb β into other variables with

$$\sum_k \beta x_k \omega_k = \sum_k \tilde{x}_k \omega_k \Rightarrow$$

$$\tilde{x}_k = \beta x_k$$

or with

$$\sum_k \beta x_k \omega_k = \sum_k x_k \tilde{\omega}_k \Rightarrow$$
$$\tilde{\omega}_k = \beta \omega_k$$

1.5a Show that the threshold fuction $\theta(a)$ (which is 1 for $a > 0$ and 0 otherwise) may be approximated by a logistic function with large weights.

Answer:

$$\varphi(a) = \frac{1}{1+\exp(-a)} \begin{cases} \to 0, & a \to -\infty \\ \to 1, & a \to +\infty \end{cases}$$

With α the angle between weight vector $\boldsymbol{\omega}$ and input vector $\mathbf{x}$, and $A = \cos(\alpha)$, the argument to φ is

$$a = \sum_k \omega_k x_k = \boldsymbol{\omega}^T \mathbf{x} = |\boldsymbol{\omega}| \cdot |\mathbf{x}| \cdot A$$

So if $|\boldsymbol{\omega}|$ is large we get a large $|a|$ and obtain a threshold function.

1.5b Show that a linear neuron may be approximated by a logistic function with small weights.

Answer:
Expand around 0

$$\varphi(a) = \frac{1}{1+\exp(-a)} \approx \frac{1}{2} + \frac{a}{4}$$

Again

$$\boldsymbol{\omega}^T \mathbf{x} = |\boldsymbol{\omega}| \cdot |\mathbf{x}| \cdot A$$

So if $|\boldsymbol{\omega}|$ is small we have a linear activation function.

Chapter: Feed-Forward Networks

Note: Problems come in a somewhat random order!

2.1 Figure 1 shows a multi-layer perceptron with direct input to output weights.

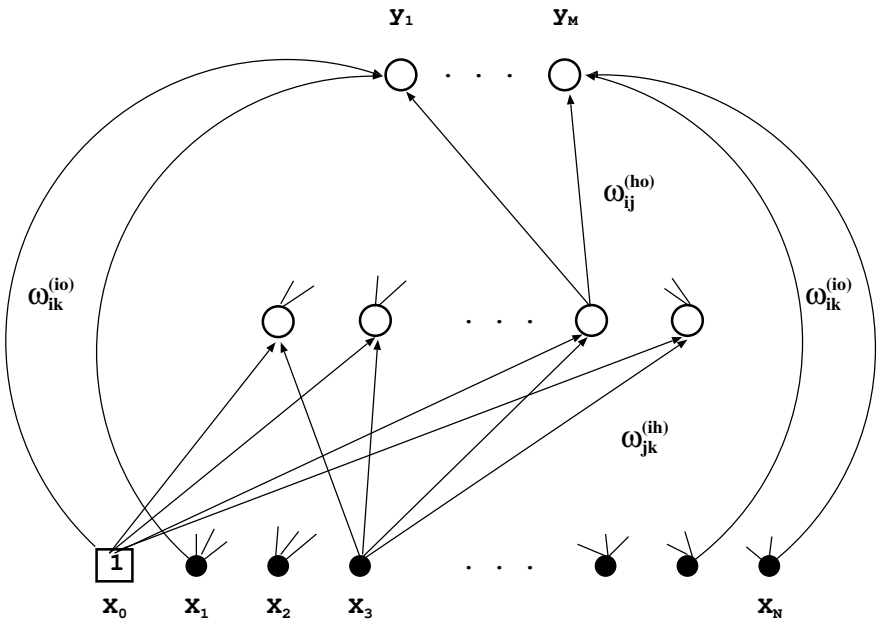

Figure 1: Network with skip-layer weights.

a) Write down the input-output mapping defined by this network.
b) Bias is shown as a special node with constant value 1, but there is no bias node in the hidden layer. Explain why it is not needed!

Answer:
a)

$$y_i = \varphi_o \left(\sum_k x_k \omega_{ik}^{(io)} + \sum_j \omega_{ij}^{(ho)} \varphi_h \left(\sum_k \omega_{jk}^{(ih)} x_k \right) \right)$$

We know that $x_0 = 1$.

b) The output nodes only need one bias, which they get from $\omega_{i0}^{(io)}$. A second bias from the hidden layer would create a sum of two constants, which is just a redundant way to write a single constant. In more conventional MLP:s, where there are no direct input-to-output links, it is common to have a bias node in the hidden layer, so that all links to the output nodes come from the hidden layer.

2.2 Figure 2 shows a simple perceptron. This output y is given by

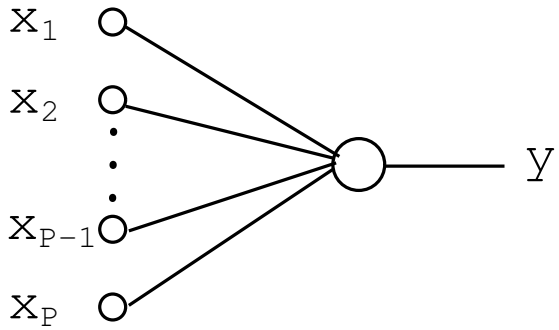

Figure 2: The perceptron

$$y = \varphi\left(\sum_{k=1}^{P} x_k \omega_k + \omega_o\right)$$

where $\varphi(a)$ can be either sigmoidal or a threshold function. Show that the *decision boundary* implemented by this network is always a hyperplane in P dimensions.

Answer:

$$y = \varphi\left(\sum_{k=1}^{P} x_k \omega_k + \omega_o\right)$$

If we assume that $\varphi(a)$ is the sigmoidal function, then a decision boundary is given by the condition

$$y = \alpha$$

where α is some number, $\in [0, 1]$.

$$\varphi\left(\boldsymbol{\omega}^T \mathbf{x} + \omega_o\right) = \alpha$$

is equivalent to

$$\boldsymbol{\omega}^T \mathbf{x} + \omega_o = \tilde{\alpha}$$

since we can invert $\varphi(a)$.

$$\boldsymbol{\omega}^T \mathbf{x} + (\omega_o - \tilde{\alpha}) = 0$$

is a hyperplane in P dimensions.

If $\varphi(a)$ is the threshold function then we directly have the condition for the decision boundary as

$$\boldsymbol{\omega}^T\mathbf{x} + \omega_o = 0$$

which again is a hyperplane in P diemensions.

2.3 **a)** Derive the gradient descent updating rule for the simple perceptron (figure 2) that uses a logistic activation function. A summed-square error function $E = \sum_n (y_n - d_n)^2$ is used as loss.

b) Consider a pattern μ that has become misclassified, so that y_μ is very close to $(1 - d_\mu)$. Discuss how this pattern will contribute to the wieght update, and explain why the combination (logistic output) with (summed error loss) can be dangerous.

Answer:

a) Derivatives $\frac{\partial E}{\partial \omega_k} = \sum_n x_{nk}\varphi'(a_n)2(y_n - d_n) = 2\sum_n x_{nk}y_n(1 - y_n)(y_n - d_n)$ give updates $\omega_k \to \omega_k - \eta\frac{\partial E}{\partial \omega_k}$, where η is a positive constant.

b) When $y_\mu \approx 1 - d_\mu$ the factor $y_\mu(1 - y_\mu)$ is small (either y_μ or $(1 - y_\mu)$ is near zero). The pattern μ hardly contributes to the gradient and becomes ignored during training. The property could be considered a feature and not a bug: you may want your MLP to "give up" on strange outliers. However, that can be dangerous since it is the MLP itself that determines which patterns to give up, and it may do so prematurely.

2.4 * Consider two one-dimensional, Gaussian distributed classes C_1 and C_2 that have a common variance equal to 1. Their mean values are:

$$\begin{aligned} \mu_1 &= -10 \\ \mu_2 &= +10 \end{aligned}$$

a) Write down an expression for the class distributions $p(x|C_1)$ and $p(x|C_2)$.

We would now like to construct a Bayes' classifier for C_1 and C_2. The condition

$$P(C_1|x) = P(C_2|x)$$

defines the Bayes' boundary.

b) Derive this boundary (i.e. rule for classification).

Answer:

From the text it follows

$$\begin{aligned} p(x|C_1) &= \frac{1}{\sqrt{(2\pi)}}e^{-(x-\mu_1)^2/2}, \qquad \mu_1 = -10 \\ p(x|C_2) &= \frac{1}{\sqrt{(2\pi)}}e^{-(x-\mu_2)^2/2}, \qquad \mu_2 = 10 \end{aligned}$$

Bayes' classifier says that all x that fulfills the following defines the class 1 region,

$$P(C_1|x) > P(C_2|x)$$

or

$$\frac{p(x|C_1)P(C_1)}{p(x)} > \frac{p(x|C_2)P(C_2)}{p(x)}$$

or

$$p(x|C_1)P(C_1) > p(x|C_2)P(C_2)$$

Now define the "boundary" x_c as

$$\log \frac{p(x_c|C_1)}{p(x_c|C_2)} = \log \frac{P(C_2)}{P(C_1)} \ \ (\equiv \alpha)$$

or

$$-(x_c - \mu_1)^2/2 + (x_c - \mu_2)^2/2 = \alpha$$

or

$$x_c = \frac{\alpha}{\mu_1 - \mu_2} + \frac{(\mu_1 + \mu_2)}{2}$$

2.5a Figure 3 shows a neural network solving this XOR problem, defined as $(0,0) \rightarrow 0$,

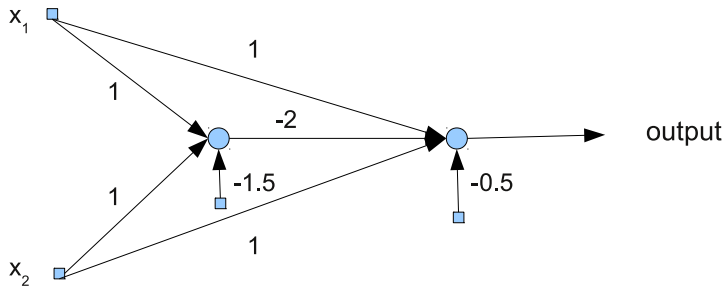

Figure 3: Network solving the XOR problem

$(0,1) \rightarrow 1$, $(1,0) \rightarrow 1$ and $(1,1) \rightarrow 0$. The activation function for both nodes is the threshold function. Show that it solves the XOR problem by constructing (a) decision regions, and (b) a truth table for the network.

Answer:
The output from this net is given by

$$y = \varphi\Big(x_1 + x_2 - 0.5 - 2 * \varphi\big(x_1 + x_2 - 1.5\big)\Big)$$

We assume that $\varphi(\cdot)$ is the usual threshold function. This means that the hidden node defined the following decision line (see Fig. 4, left),

$$x_2 = -x_1 + 1.5$$

Denote the output from the hidden node by h, then the output is given by

$$y = \varphi\big(x_1 + x_2 - 0.5 - 2h\big)$$

From which we get the final decision plane(s),

$$x_2 = -x_1 + 0.5 + 2h$$

Depending on the value of h we get two decision lines:

$$\begin{aligned} x_2 &= -x_1 + 0.5 \qquad \text{if } h = 0 \\ x_2 &= -x_1 + 2.5 \qquad \text{if } h = 1 \end{aligned}$$

See figure 4, right. This ones solves the XOR problem. Note that it does not generalize well.

2.5b One variant of the XOR problem is the following mapping: $(-1,-1) \to 0$, $(+1,-1) \to 1$, $(-1,+1) \to 1$ and $(+1,+1) \to 0$. Figure 5 shows a perceptron that can handle this XOR problem. The trick here is to define an input that is the product of x_1 and x_2. So we can see this as transformation of input space to easier handle the XOR problem. But we can also analyze this simple network an see what new boundaries this network is defining in the original 2D input space.

This network is given by

$$y = \theta\left(x_1 * x_2 * \omega_{12} + \omega_0\right)$$

(θ is the threshold function). What boundary does this network implement? Should it be able to handle the XOR problem?

Answer:
The output from this net is given by

$$y = \Theta\left(x_1 * x_2 * \omega_{12} + \omega_0\right)$$

Setting the argument to 0, means finding all the (x_1, x_2) that lies on the boundary.

$$\begin{aligned} x_1 * x_2 * \omega_{12} + \omega_0 &= 0 \\ x_2 &= -\frac{\omega_0}{\omega_{12}} \frac{1}{x_2} \end{aligned}$$

So we have a "1/x" curve as the boundary, see figure 6.

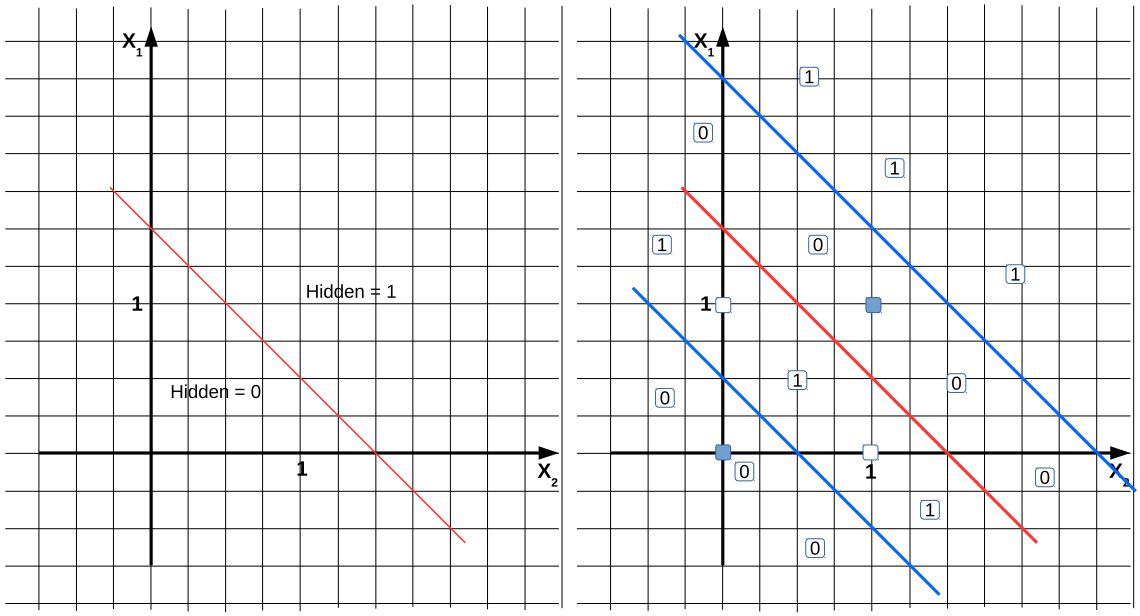

Figure 4: Decision planes implemented by this network. Both for the hidden node (red) and the output (blue).

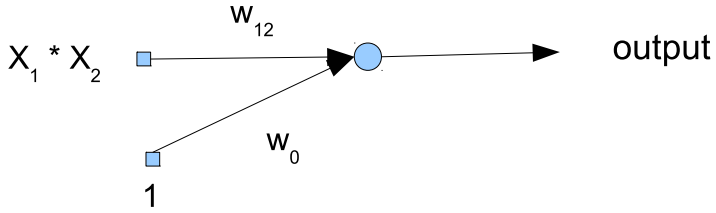

Figure 5: Network solving the XOR problem

2.6 Show that a multi-layer perceptron with linear activation functions is equivalent to a single layer network.

Answer:
We start with a single hidden layer. With notation $\tilde{\omega}_{jk}$ for weights from input k to hidden node j and ω_{ij} for weights from hidden node j to output i, the argument a_{ni} to output i for pattern n is $a_{ni} = \sum_{j\geq 1} \omega_{ij}\varphi_h(\sum_k \tilde{\omega}_{jk}x_{nk}) + \omega_{i0})$, For a linear activation

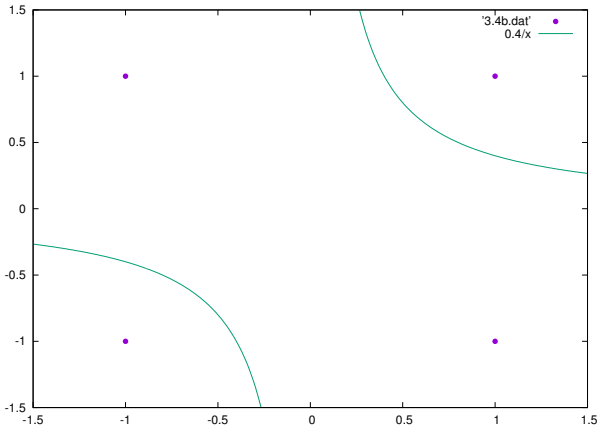

Figure 6: Decision boundary implemented by simple perceptron with a $x_1 * x_2$ input.

function $\varphi_h(a) = a$, this is $a_{ni} = \sum_{j\geq 1} \omega_{ij} \sum_k \tilde{\omega}_{jk} x_{nk} + \omega_{i0} = \sum_k \left[\sum_{j\geq 1} \omega_{ij}\tilde{\omega}_{jk}\right] x_{nk} + \omega_{i0}$. With weights $W_{ik} = \sum_{j\geq 1} \omega_{ij}\tilde{\omega}_{jk}$ and perceptron bias $B_i = W_{i0} + \omega_{i0}$, we see that $a_{ni} = \sum_{k\geq 1} W_{ik}x_{nk} + B_i$, is equivalent to a direct contact from inputs to node i, without any hidden layer.

If there are multiple hidden layers, the argument above can be used repeatedly to remove all hidden layers.

2.7 Show, for networks with tanh() as hidden node activation function, that the network mapping is invariant if all of the weights and the threshold feeding into and out of a node have their signs changed. Demonstrate the corresponding symmetry for hidden nodes with logistic activation functions.

Answer:
A node j gets its value with an activation function $\varphi()$ of an argument a_j.

$$h_j = \varphi(\sum_k \omega_{jk}\tilde{h}_k) = \varphi(a_j)$$

where $\tilde{h}$ denotes node values in an earlier layer (possibly the input layer, in which case $\tilde{h} = x$). If we change sign on all weights ω_{jk} the argument a_j changes sign. For any odd activation function, such as tanh, we have $\varphi(-a) = -\varphi(a)$. So, the node value h_j simply changes sign. This can of course be compensated by changing the sign of all outgoing weights from node j.

If instead

$$\varphi(a) = \frac{1}{1+e^{-a}}$$

we have

$$\varphi(-a) = \frac{1}{1+e^{a}} = 1 - \varphi(a)$$

This means that we get the following relation for the outgoing weights from node j if we want to keep the same function,

$$\omega^{old} h_j = \omega^{new}(1 - h_j)$$

leading to

$$\omega^{new} = \omega^{old} \frac{h_j}{1 - h_j}$$

So changing the sign of all weights feeding into a hidden node with the logistic activation function we need to change all outgoing weights weights according to ω^{new}.

We could also use the simpler $\omega^{new} = -\omega^{old}$ and then absorb the constant shift into the biases b for all receiving nodes: $\omega_{ij}^{old} h_j + b_i^{old} = (-\omega_{ij}^{old})(1-h_j) + b_i^{new} \Rightarrow b_i^{new} = b_i^{old} + \omega_{ij}^{old}$

2.8 Consider a 1-hidden layer MLP, with logistic activation function in the hidden layer, i.e. $\varphi(a) = 1/(1 + \exp(-a))$. Show that there exists an equivalent network, which computes exactly the same function, but with hidden activation functions given by $\varphi(a) = \tanh(a)$.

Hint: First find a relation between the logistic and the tanh functions and then find the transformation of the weights needed.

Answer:
The logistic function looks like a rescaled logistic function. So let's try

$$2\varphi(a) - 1 = 2\frac{1}{1+e^{-a}} - 1 =$$
$$\frac{2 - (1+e^{-a})}{1+e^{-a}} = \frac{1-e^{-a}}{1+e^{-a}} =$$
$$\frac{e^{a/2} - e^{-a/2}}{e^{a/2} + e^{-a/2}} = \tanh(a/2)$$

so

$$2\varphi(a) - 1 = \tanh(a/2)$$

From this we can see what kind of transformations needed. See figure 7

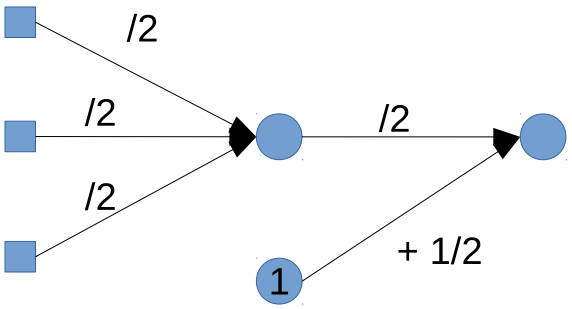

Figure 7: The transformations needed to change from logistic to tanh activation function.

2.9 Show, for a feed-forward network with tanh() as hidden activation function, and a summed-square error function as loss, that the origin in weight space is a stationary point of the loss function.

Answer:

The derivative of the loss E w.r.t. a hidden-to-output weight ω_{ij} is $\frac{\partial E}{\partial \omega_{ij}} = \sum_n \frac{\partial E}{\partial a_{ni}} \frac{\partial a_{ni}}{\partial \omega_{ij}} = \sum_n \frac{\partial E}{\partial a_{ni}} h_{nj}$, since argument a_{ni} to node i for pattern n is $a_{ni} = \sum_j \omega_{ij} h_{nj}$. With all weights and biases 0, every hidden node argument $\tilde{a}_{nj}$ is 0, and $\tanh(0) = 0$ so all hidden node values $h_{nj} = 0$. That gives $\frac{\partial E}{\partial \omega_{ij}} = 0$.

The derivative w.r.t. a input-to-hidden weight $\tilde{\omega}_{jk}$ is $\sum_n \frac{\partial E}{\partial h_{nj}} \frac{\partial h_{nj}}{\partial \tilde{\omega}_{jk}}$. But $\frac{\partial E}{\partial h_{nj}} = \sum_i \frac{\partial E}{\partial a_{ni}} \frac{\partial a_{ni}}{\partial h_{nj}} = \sum_i \frac{\partial E}{\partial a_{ni}} \omega_{ij} = 0$ since all $\omega_{ij} = 0$.

With all derivatives zero, all weight updates are zero, and we are in a stationary point.

2.10 Lets return to the multi-layer perceptron defined in Figure 1. Assume a summed-square-error function $E = \sum_n \sum_i (y_i(\mathbf{x}_n) - d_{ni})^2$, where d_{ni} are the targets. The activation functions are $g^h()$ and $g^o()$ for the hidden and output layer, respectively. Derive an expression for

$$\frac{\partial E}{\partial \omega_{jk}^{ih}}, \qquad \frac{\partial E}{\partial \omega_{ij}^{ho}}, \qquad \text{and} \qquad \frac{\partial E}{\partial \omega_{ik}^{io}}$$

Answer:
With

$$E = \sum_n \sum_i (y_i(\mathbf{x}_n) - d_{ni})^2 = \sum_n \sum_i (y_{ni} - d_{ni})^2 = \sum_n \sum_i E_{ni}$$

and

$$y_{ni} = g^o\left(\sum_k x_{nk}\omega_{ik}^{(io)} + \sum_j \omega_{ij}^{(ho)} g^h\left(\sum_k \omega_{jk}^{(ih)} x_{nk}\right)\right)$$

$$\begin{aligned}\frac{\partial E}{\partial \omega_{ik}^{(io)}} &= \sum_n \frac{\partial E_{ni}}{\partial y_{ni}} \frac{\partial y_{ni}}{\partial \omega_{ik}^{(io)}} \\ &= 2\sum_n (y_{ni} - d_{ni}) g'^o(\cdot) x_{nk}\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial \omega_{ij}^{(ho)}} &= \sum_n \frac{\partial E_{ni}}{\partial y_{ni}} \frac{\partial y_{ni}}{\partial \omega_{ij}^{(ho)}} \\ &= 2\sum_n (y_{ni} - d_{ni}) g'^o(\cdot) g^h(\cdot)\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial \omega_{jk}^{(ih)}} &= \sum_n \sum_i \frac{\partial E_{ni}}{\partial y_{ni}} \frac{\partial y_{ni}}{\partial \omega_{jk}^{(ih)}} \\ &= 2\sum_n \sum_i (y_{ni} - d_{ni}) g'^o(\cdot) \omega_{ij}^{(ho)} g'^h(\cdot) x_{nk}\end{aligned}$$

2.11 *

Consider a binary classification problem in which the target values are $d \in \{0, 1\}$, with a network output $y(\mathbf{x}, \mathbf{w})$ that represents $p(d = 1|\mathbf{x})$, and suppose that there is a probability ϵ that the class label on a training data point has been incorrectly set. Assuming independent and identically distributed data, write down the loss function corresponding to the negative log likelihood. Verify that the usual cross-entropy error function is obtained when $\epsilon = 0$. Note that this loss function makes the model robust to incorrectly labelled data, in contrast to the usual loss function.

Hint 1: Introduce $k \in \{0, 1\}$ as the true class label.
Hint 2: Express $p(d = 1|\mathbf{x})$ in terms of $p(k|\mathbf{x})$ and ϵ, where d is the dataset label (observed).

Answer:

Let $d \in \{0, 1\}$ denote the data set label and let $k \in \{0, 1\}$ denote the true class label. We want the network output to have the interpretation $y(\mathbf{x}, \mathbf{w}) = p(k = 1|\mathbf{x})$. From the rules of probability we have

$$\begin{aligned} p(d = 1|\mathbf{x}) &= \sum_{k=0}^{1} p(d = 1|k)p(k|\mathbf{x}) \\ &= \epsilon p(k = 0|\mathbf{x}) + (1 - \epsilon)p(k = 1|\mathbf{x}) \\ &= \epsilon(1 - y(\mathbf{x}, \boldsymbol{\omega})) + (1 - \epsilon)y(\mathbf{x}, \boldsymbol{\omega}). \end{aligned}$$

The conditional probability of the data label is then

$$p(d|\mathbf{x}) = p(d = 1|\mathbf{x})^d (1 - p(d = 1|\mathbf{x}))^{1-d}.$$

Forming the likelihood and taking the negative logarithm we then obtain the loss function in the form

$$\begin{aligned} E(\boldsymbol{\omega}) = -\sum_{n=1}^{N} \Big[& d_n \log \Big((1 - \epsilon)y(\mathbf{x}_n, \boldsymbol{\omega}) + \epsilon(1 - y(\mathbf{x}_n, \boldsymbol{\omega})) \Big) \\ & + (1 - d_n) \log \Big(1 - (1 - \epsilon)y(\mathbf{x}_n, \boldsymbol{\omega}) - \epsilon(1 - y(\mathbf{x}_n, \boldsymbol{\omega})) \Big) \Big]. \end{aligned}$$

Chapter: Generalization

Network Ensembles

3.1 Suppose we have L trained models $y_i(\mathbf{x})$ $(i = 1, ..., L)$. All models try to approximate $h(\mathbf{x})$. We can write each mapping $y_i(\mathbf{x})$ as,

$$y_i(\mathbf{x}) = h(\mathbf{x}) + \epsilon_i(\mathbf{x})$$

and the corresponding error D_i for each model as,

$$D_i = \mathrm{E}\left[(y_i(\mathbf{x}) - h(\mathbf{x}))^2 \right] = \mathrm{E}\left[\epsilon_i(\mathbf{x})^2 \right]$$

where (here) $\mathrm{E}[\cdot]$ is defined as,

$$\mathrm{E}\left[\epsilon(\mathbf{x})^2 \right] = \int \epsilon(\mathbf{x})^2 p(\mathbf{x}) d\mathbf{x}$$

and where $p(\mathbf{x})$ is the distribution of the input data $\mathbf{x}$. Let now D_{av} be the mean error of the L networks

$$D_{\text{av}} = \frac{1}{L}\sum_i D_i$$

Now introduce an ensemble in the form

$$y_{\text{ens}}(\mathbf{x}) = \frac{1}{L}\sum_i y_i(\mathbf{x})$$

The ensemble error is

$$D_{\text{ens}} = \text{E}\left[(y_{\text{ens}}(\mathbf{x}) - h(\mathbf{x}))^2\right]$$

Now assume that the errors $\epsilon_i(\mathbf{x})$ are uncorrelated and have zero mean, i.e. $\text{E}\left[\epsilon_i\right] = 0$ and $\text{E}\left[\epsilon_i\epsilon_j\right] = 0, \;\; i \neq j$. Under this assumption show that,

$$D_{\text{ens}} = \frac{1}{L}D_{\text{av}}$$

Answer:
First

$$D_{\text{av}} = \frac{1}{L}\sum_i D_i = \frac{1}{L}\text{E}\left[\epsilon_i(\mathbf{x})^2\right]$$

second

$$D_{\text{ens}} = \text{E}\left[(y_{\text{ens}}(\mathbf{x}) - h(\mathbf{x}))^2\right] = \text{E}\left[(\frac{1}{L}\sum_i y_i(\mathbf{x}) - h(\mathbf{x}))^2\right] = \text{E}\left[(\frac{1}{L}\sum_i \epsilon_i(\mathbf{x}))^2\right]$$

If $\text{E}\left[\epsilon_i\right] = 0$ and $\text{E}\left[\epsilon_i\epsilon_j\right] = 0, \;\; i \neq j$, then

$$D_{\text{ens}} = \frac{1}{L^2}\sum_i \text{E}\left[\epsilon_i(\mathbf{x})^2\right] = \frac{1}{L}D_{\text{av}}$$

3.2 Consider a committee machine consisting of L MLP:s, with the following averaging,

$$y_{\text{com}}(\mathbf{x}) = \sum_{i=1}^{L}\alpha_i y_i(\mathbf{x})$$

where α_i is the weighting factor and $y_i(\mathbf{x})$ is the i:th committee member. Given a data set $\{\mathbf{x}_n, d_n\}_{n=1}^N$ the task is now to minimize the following error function

$$\begin{aligned} E &= \frac{1}{N}\sum_n \left(y_{com}(\mathbf{x}_n) - d_n\right)^2 \\ &= \frac{1}{N}\sum_n \left(\sum_i \alpha_i y_i(\mathbf{x}_n) - d_n\right)^2 \end{aligned}$$

with respect to α_i. One can solve this minimization problem using Lagrangian multipliers. Show that for the constraint $\sum_i \alpha_i = 1$, the solution is

$$\alpha_i = \frac{\sum_l (C^{-1})_{il}}{\sum_{jl}(C^{-1})_{jl}}$$

where C is the matrix with the matrix elements

$$C_{ij} = \frac{1}{N}\sum_n \epsilon_i(\mathbf{x}_n)\epsilon_j(\mathbf{x}_n)$$

with $\epsilon_i(\mathbf{x}_n) = y_i(\mathbf{x}_n) - d_n$.

Hint: The Lagrangian multiplier technique adds the constraint as an additional term to the function we want to minimize. E.g. $\lambda\left(\sum_i \alpha_i - 1\right)$

Answer:

Define the error $\epsilon_i(\mathbf{x})$ as the error of the i:th network in the ensemble

$$\epsilon_i(\mathbf{x}_n) = y_i(\mathbf{x}_n) - d_n$$

This leads to

$$y_{\text{com}}(\mathbf{x}_n) = \sum_i^L \alpha_i \left[\epsilon_i(\mathbf{x}_n) + d_n\right] = d_n + \sum_i^L \alpha_i \epsilon_i(\mathbf{x}_n)$$

This means that we can write the error as

$$\begin{aligned} E &= \frac{1}{N}\sum_n \left(\sum_i \alpha_i \epsilon_i(\mathbf{x}_n)\right)^2 = \\ &= \frac{1}{N}\sum_n \left(\sum_i \alpha_i \epsilon_i(\mathbf{x}_n)\right)\left(\sum_j \alpha_j \epsilon_j(\mathbf{x}_n)\right) = \\ &= \sum_i \sum_j \alpha_i \alpha_j \underbrace{\frac{1}{N}\sum_n \epsilon_i(\mathbf{x}_n)\epsilon_j(\mathbf{x}_n)}_{C_{ij}} \end{aligned}$$

where C_{ij} is the error correlation matrix. So far we have

$$E = \sum_i \sum_j \alpha_i \alpha_j C_{ij} \qquad \text{with} \qquad \sum_i \alpha_i = 1$$

Use the technique of Lagrangian multipliers, which means that we incorporate the constraint into the error function.

$$\tilde{E} = \sum_i \sum_j \alpha_i \alpha_j C_{ij} + \lambda \left(\sum_i \alpha_i - 1 \right)$$

The condition $\partial \tilde{E} / \partial \alpha_i = 0$ gives

$$0 = \sum_j \alpha_i C_{ij} + \sum_k \alpha_i C_{ki} + \lambda$$

Since $C_{ij} = C_{ji}$ by its definition, and summation indices can be given arbitrary name, we get

$$0 = 2 \sum_j \alpha_i C_{ij} + \lambda = 0 \quad \Rightarrow$$
$$\alpha_i = -\frac{\lambda}{2} \sum_j C_{ij}^{-1}$$

Now use this expression in the constraint $\sum_i \alpha_i = 1$

$$-\frac{\lambda}{2} \sum_i \sum_j C_{ij}^{-1} = 1 \quad \Rightarrow$$
$$-\frac{\lambda}{2} = \frac{1}{\sum_i \sum_j C_{ij}^{-1}} \qquad \text{and finally}$$
$$\alpha_i = \frac{\sum_j C_{ij}^{-1}}{\sum_{i'} \sum_j C_{i'j}^{-1}}$$

3.3 When we create a bootstrap sample from a dataset of size N, we randomly select samples N times, with replacement. Some samples are picked many times, others are never picked.

a) What is the probability that pattern i isn't selected in the first pick for the bootstrap sample?

b) What is the probability that pattern i isn't selected to the bootstrap sample in any of the N picks?

c) Show that the large N limit of the result in (b) is $1/e$.

Answers:

a) $(N-1)/N = 1 - 1/N$, the fraction of other patterns.

b) $(1-1/N)^N$. Replacement means that every pick is independent of others, so it is just the product of probabilities from (a).

c) $\lim_{N\to\infty}(1+x/N)^N = \exp(x)$. We have $x = -1$.

Chapter: CNN, Autoencoder and GAN

4.1 How many trainable parameters are there in a convolution filter that has a $K \times K$ kernel acting on C channels? Each channel is a 2D picture of size $W \times H$.

Answer:
$K^2C + 1$. The "+1" represents the bias in the activation function. Importantly, the number of parameters is independent of the input image size $W \times H$. This illustrates that the CNN can be seen as a sparse MLP with *shared weights*.

4.2 A CNN has three input channels with 6x6 images. A convolutional layer consists of 2 filters with 3x3 kernels and stride 1, without padding. This is followed by 2x2 max pooling with stride 2. The result is flattened and sent as input to an MLP.

a How many input nodes does the MLP get?

Answer:
8. The 3x3 kernel with stride 1 will reduce the image size from 6x6 to 4x4. Since there are two filters, there will be 2 channels with 4x4 images. After pooling, they are reduced to 2x2 images. There are 2 (channels) $\times$ 2x2 = 8 input nodes to the MLP.

b How many trainable parameters does the CNN have, before the MLP?

Answer:
56. Each 3x3 filter acting on 3 input channels has 3x3x3= 27 links and 1 bias. Two filters therefore contain 2x28= 56 trainable parameters. The max pooling step does not contain any trainable paramters.

c The same input images are convoluted with the same 2 filters, but now with zero-padding. This is followed by the same max pooling. What are the new answers to (a) and (b)?

Answer:
18 and 56. Without any other specification, "padding" can be assumed to preserve image sizes, so the convolution creates 2 channels of 6x6 images, that are reduced to 3x3 in the pooling step. The MLP is given 2 $\times$ 3x3= 18 input nodes. The number of trainable parameters is unaffected by padding.

4.3 *

A filter in layer l in a CNN has kernel size K_l and stride S_l. If the input and hidden images are multi-dimensional, consider symmetric kernels and strides, so that the same values apply to all dimensions.

a) Show that size of the receptive field R_l for a node created by the filter in l and the displacement D_l between receptive fields of neighbouring nodes can be determined recursively by

$$\begin{aligned} D_1 &= S_1 \\ R_1 &= K_1 \\ D_{l+1} &= S_{l+1} D_l \\ R_{l+1} &= R_l + (K_{l+1} - 1) D_l \end{aligned}$$

provided $S_l \leq K_l$.

b) A CNN begins with a 3x3 convolution with stride 1, followed by a 2x2 max pooling with stride 2, and then a 3x3 convolution with stride 2. What is the size of the receptive field for a node after the first convolution, the max pooling, and the last convolution, respectively?

Answer:

a) In general $R_{l+1}(i)$ for node i in layer l is the union of all $R_l(j)$ for the nodes j in layer l influencing i via the kernel K_{l+1}. This union can be seen as $R_l(j^*)$ for one node in l, plus the extension of the field for every extra node in l covered by K_{l+1}. As long as $S_l \leq K_l$, there will not appear any "gaps" between receptive fields of neighbouring nodes, and R_{l+1} grows with D_l for every extra node covered by K_{l+1}, giving a total of

$$R_{l+1} = \underbrace{R_l}_{\text{one node}} + \overbrace{(K_{l+1} - 1)}^{\text{number of other nodes}} \underbrace{D_l}_{\text{addition per node}}$$

b) After first convolution: $R_1 = K_1 = 3$, $D_1 = S_1 = 1$. After max pooling: $R_2 = 3 + (2-1) \cdot 1 = 4$, $D_2 = 2 \cdot 1 = 2$ After second convolution: $R_3 = 4 + (3-1) \cdot 2 = 8$. The results are obtained with $K_1 = 3, S_1 = 1, K_2 = 2, S_2 = 2, K_3 = 3$. Stride $S_3 = 2$ will influence D_3, and the receptive field for forward layers. Note that convolution and max pooling need to be treated separately in calculations of receptive fields.