Chapter: Introduction

2.1 An example of the logistic function is defined by

$$\varphi(x) = \frac{1}{1+\exp(-ax)}$$

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

Answer:

$$\varphi(x) = \frac{1}{1+\exp(-ax)}$$

$$\varphi'(x) = \frac{1}{(1+\exp(-ax))^2} a\exp(-ax) = a\varphi(x)^2\exp(-ax) =$$

$$a\varphi(x)^2\left(1+\exp(-ax)-1\right) = a\varphi(x)^2\left(\frac{1}{\varphi(x)}-1\right) =$$

$$= a\varphi(x)\left(1-\varphi(x)\right)$$

$$\varphi'(0) = a\varphi(0)(1-\varphi(0)) = a\frac{1}{2}\left(1-\frac{1}{2}\right) = \frac{a}{4}$$

2.2 An odd sigmoidal function is defined by

$$\varphi(x) = \tanh(ax)$$

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

Answer:
Start with $\varphi(x) = \tanh(x)$

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

$$\varphi(x)' = \frac{e^x + e^{-x}}{e^x + e^{-x}} - \frac{(e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} =$$

$$= 1 - \tanh(x)^2$$

For $\varphi(x) = \tanh(ax)$ we simply get

$$\varphi(x)' = a\left(1 - \tanh(ax)^2\right)$$
$$\varphi(0)' = a$$

2.3 Yet another odd sigmoidal function is the algebraic sigmoid:

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

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

Answer:
Start with $\varphi(x) = \frac{x}{\sqrt{1+x^2}}$

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

And the derivative at origin

$$\varphi(x)' = \frac{1}{(1+x^2)^{3/2}}$$
$$\varphi(0)' = 1$$

2.4 Suppose we have an activation function as defined in Problem 2.1, where the parameter a governs the slope of the sigmoidal function and x is e.g. a weighted sum of inputs. We would like to absorb the a parameter into x. How can we modify either the weights $(\omega_1, ..., \omega_N)$ or the inputs $(x_1, ..., x_N)$ to accomplish this?

Answer:

$$\varphi(v) = \frac{1}{1+\exp(-av)}$$

$$v = \sum_k x_k \omega_k \Rightarrow$$
$$\varphi(v) = \frac{1}{1 + \exp(-a \sum_k x_k \omega_k)}$$

Now

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

or

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

2.5a Show that the McCulloch-Pitts model of a neuron (i.e. $\theta(x)$) may be approximated by a logistic function with large weights.

Answer:

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$
$$\lim_{a \to \infty} \varphi(v) = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v > 0 \end{cases}$$

Now the argument is

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

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

2.5b Show that a linear neuron may be approximated by a logistic function with small weights.

Answer:
Expand around 0

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

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.

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

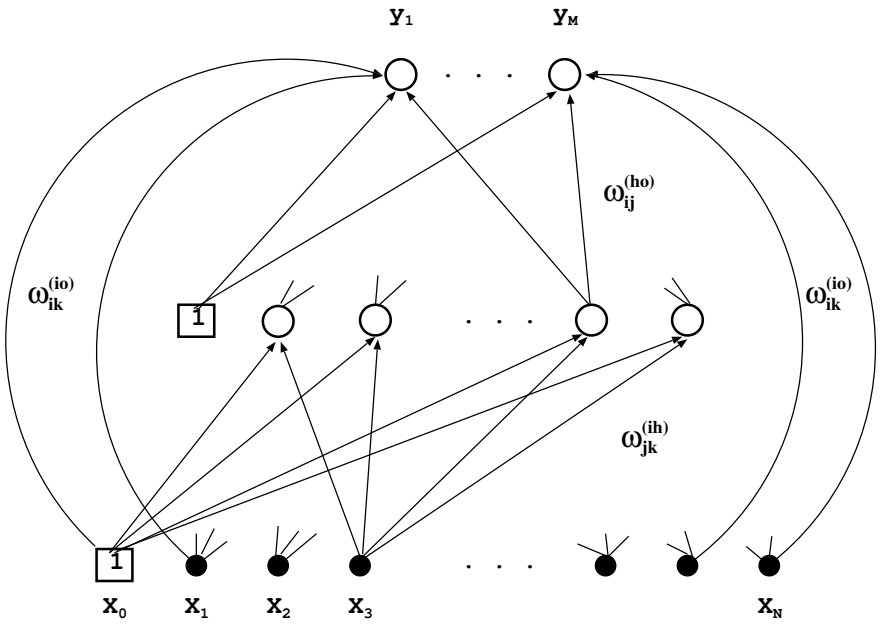

Figure 1: Network with skip-layer weights.

down the input-output mapping defined by this network.

Answer:

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

Chapter: Feed-Forward Networks

Note: Problems comes in a somewhat random order!

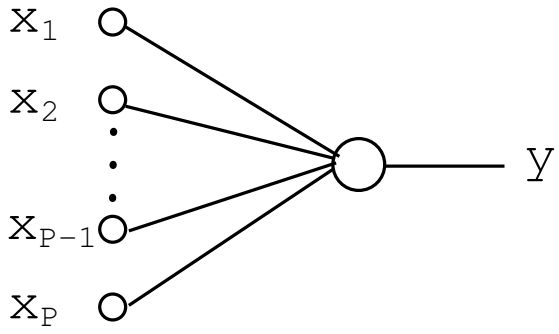

Figure 2: The perceptron

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

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

where $\varphi(x)$ 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 $\varphi(x)$ 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 equivallent to

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

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

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

is a hyperplane in P dimensions.

If $\varphi(x)$ 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.

3.2 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.

3.3 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

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

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

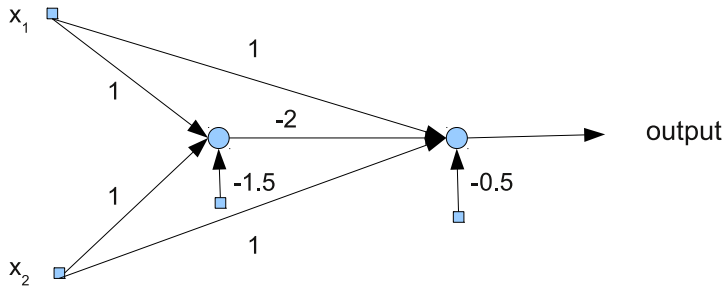

Figure 3: Network solving the XOR problem

$(0, 1) \to 1$, $(1, 0) \to 1$ and $(1, 1) \to 0$. 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.

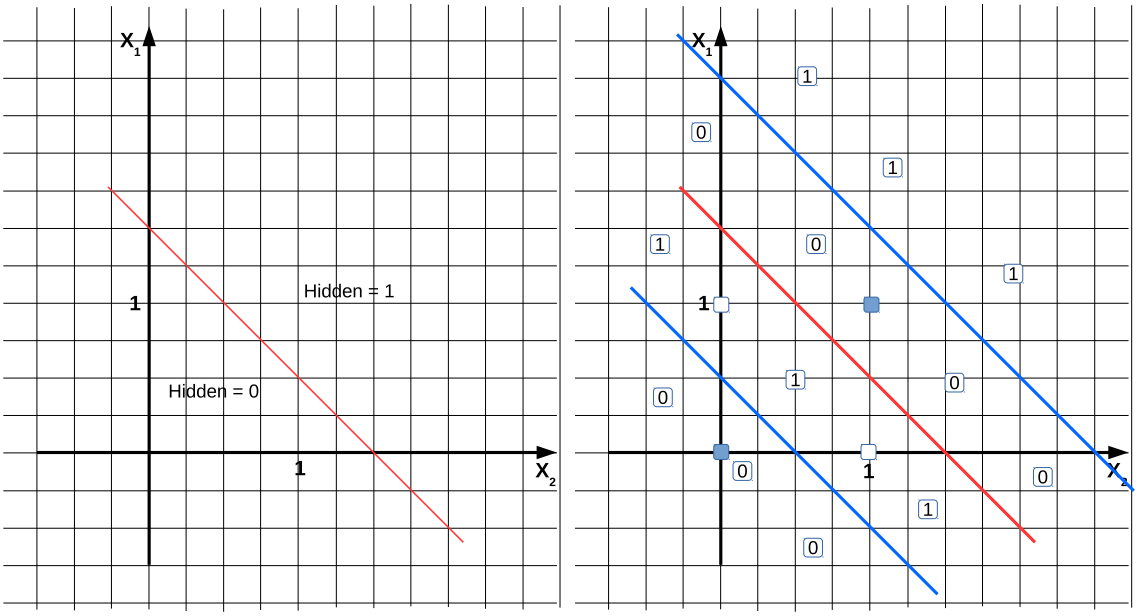

Figure 4: Decision planes implemented by this network. Both for the hidden node (red) and the output (blue).

3.4b One variant of the XOR problem is the following mapping: $(-1,-1) \rightarrow 0$, $(+1,-1) \rightarrow 1$, $(-1,+1) \rightarrow 1$ and $(+1,+1) \rightarrow 0$. 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.

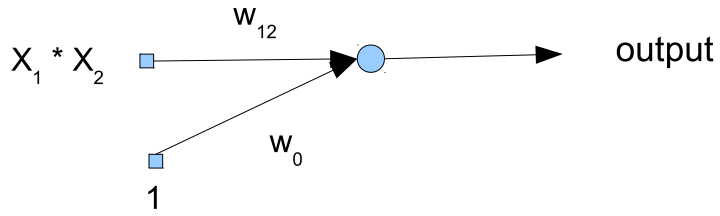

Figure 5: Network solving the XOR problem

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.

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

3.6 Show, for networks with tanh() hidden nodes 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:
For any hidden node j we have,

$$h_j = \varphi(v_j)$$

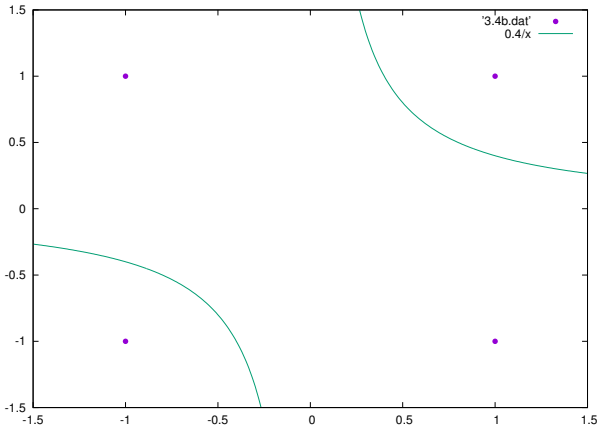

Figure 6: Decision boundary implemented by simple perceptron with a $x_1 * x_2$ input.

where $\varphi(v_j) = \tanh(x)$. Now $\varphi(x)$ has the property

$$\varphi(-x) = -\varphi(x)$$

Since $-v_j = -\boldsymbol{\omega}_j^T \mathbf{x} = (-\boldsymbol{\omega}_j^T)\mathbf{x}$ this means that if we change the sign of all weights feeding into node j, we changed sign on the output of this node. This can of course be fixed by changing the sign of all outgoing weights from node j.

If,

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

Here we have,

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

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}.

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

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

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

so

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

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

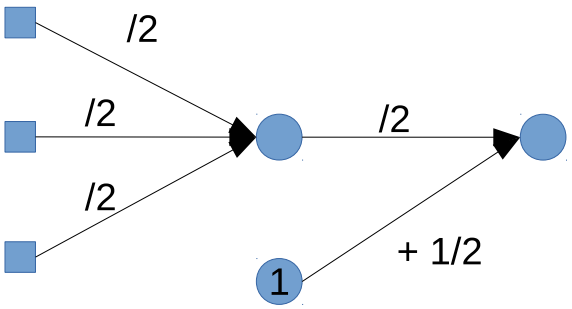

Figure 7: The transformations needed to change from logistic to tanh activation function.

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

3.9 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

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

3.10 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 error function corresponding to the negative log likelihood. Verify that the ususal cross-entropy error function is obtained when $\epsilon = 0$. Note that this error function makes the model robust to incorrectly labelled data, in contrast to the usual error 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 $t \in \{0, 1\}$ 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

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

The conditional probability of the data label is then

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

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

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

3.11 Recall the early stopping technique for improving generalization and limiting overtraining. (a) Compare weight decay with early stopping. What do these techniques have in common? How do their effects differ? (b) When implementing a minimization procedure using early stopping, why would you prefer simple gradient descent to a second order method like Quasi Newton?

Answer:
(a) Both weight decay and early stopping act to improve generalization by reducing the effective complexity of the network. Both techniques have the effect of limiting the size of the weights in the network. Where they differ is in how the sizes of the weights are limited. Weight decay tends to decrease the value of "unimportant" weights to zero, allowing a smaller number of high valued weights. Early stopping tends to do the opposite, weights are allowed to increase from small initial values to moderate values; learning is stopped before they can grow very large. So weight decay results in effectively fewer large weights and early stopping results in a larger number of moderate weights.

(b) As the weights are updated, their values tend to increase from small random values. The increase in the weight values has the effect of gradually increasing the overall nonlinearity of the network by moving the input of each logistic (or tanh) function further from the linear region centered about zero. The increase in the nonlinearity of the network increases the expressive power of the network. The idea behind early stopping is to stop learning when the weights are large enough to allow the network to express the true target function, but not so large that it can express the peculiarities of the training set. Compared to gradient descent, Newton's method and conjugate gradient method tend to take larger steps in weight space, allowing the size of the weights to increase significantly in a single step. When implemented with one of these optimization techniques, early stopping becomes less effective at limiting the complexity of the network because it has less control over the final size of the weights.

Network Ensembles

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

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

where ω_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

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

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

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

where C is the matrix with the matrix elements

$$C_{ij} = \frac{1}{N}\sum_n \epsilon_i(n)\epsilon_j(n)$$

with $\epsilon_i(n) = y_i(\mathbf{x}(n)) - d(n)$.

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

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

$$y_{\text{com}}(\mathbf{x}(n)) = \sum_i^L \omega_i \epsilon_i(\mathbf{x}(n)) + d(n)$$

This means that we can write the error as

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

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

$$E = \sum_i \sum_j \omega_i \omega_j C_{ij} \qquad \text{with} \qquad \sum_i \omega_i = 1$$

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

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

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

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

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

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