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?

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?

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?

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?

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.

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

2.6 Figure 1 shows a multi-layer perceptron with direct input to output weights. Write down the input-output mapping defined by this network.

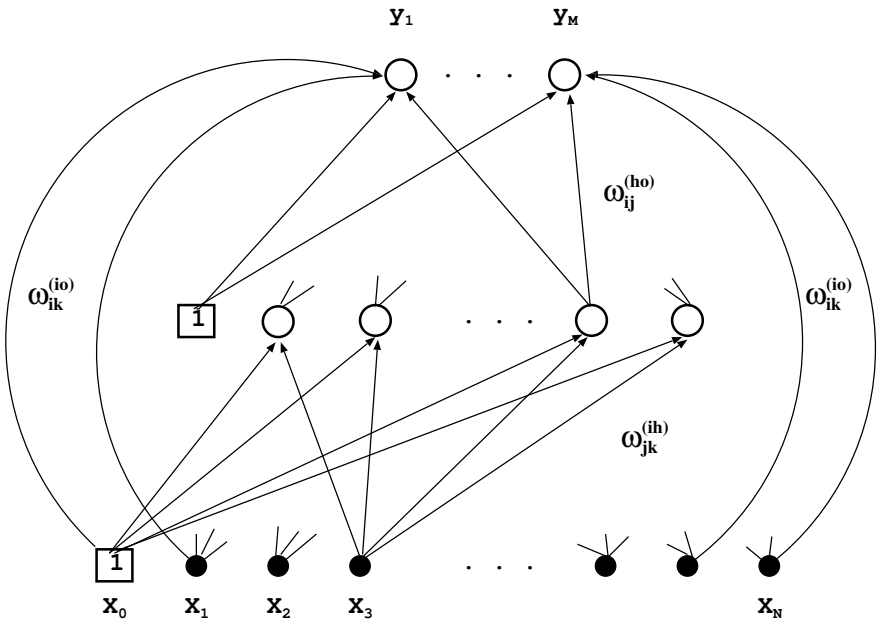

Figure 1: Network with skip-layer weights.

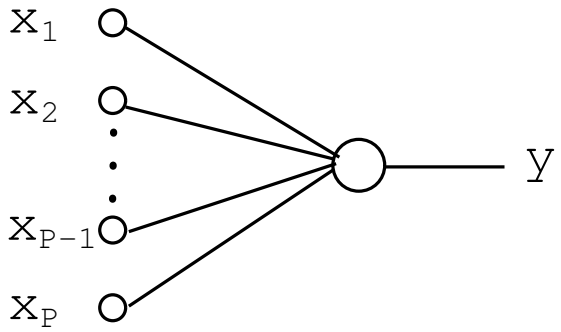

Figure 2: The perceptron

Chapter: Feed-Forward Networks

Note: Problems comes in a somewhat random order!

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.

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).

3.4 Figure 3 shows a neural network solving this XOR problem, defined as $(0,0) \rightarrow 0$, $(0,1) \rightarrow 1$, $(1,0) \rightarrow 1$ 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.

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 4 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?

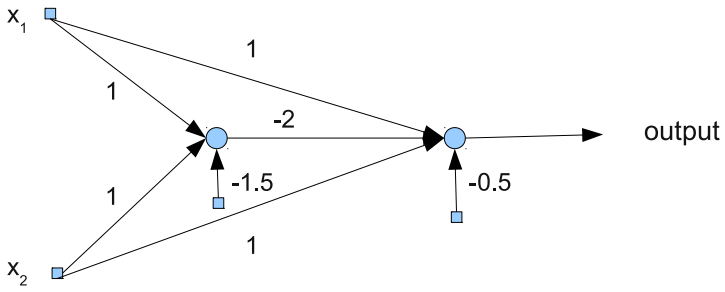

Figure 3: Network solving the XOR problem

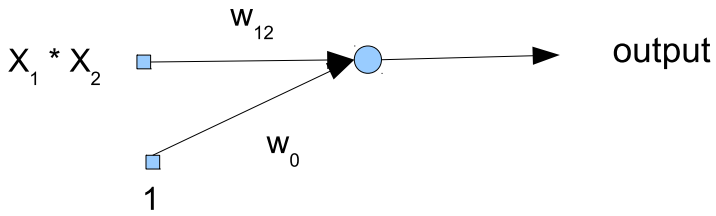

Figure 4: Network solving the XOR problem

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.

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.

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).

3.11 Recall the early stopping technique for improving generalization and limiting over-training. (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?

Network Ensembles

4.1 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

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

Now introduce an ensemble in the form

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

The ensemble error is

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

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

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

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

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

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. $\lambda \left(\sum_i \omega_i - 1\right)$