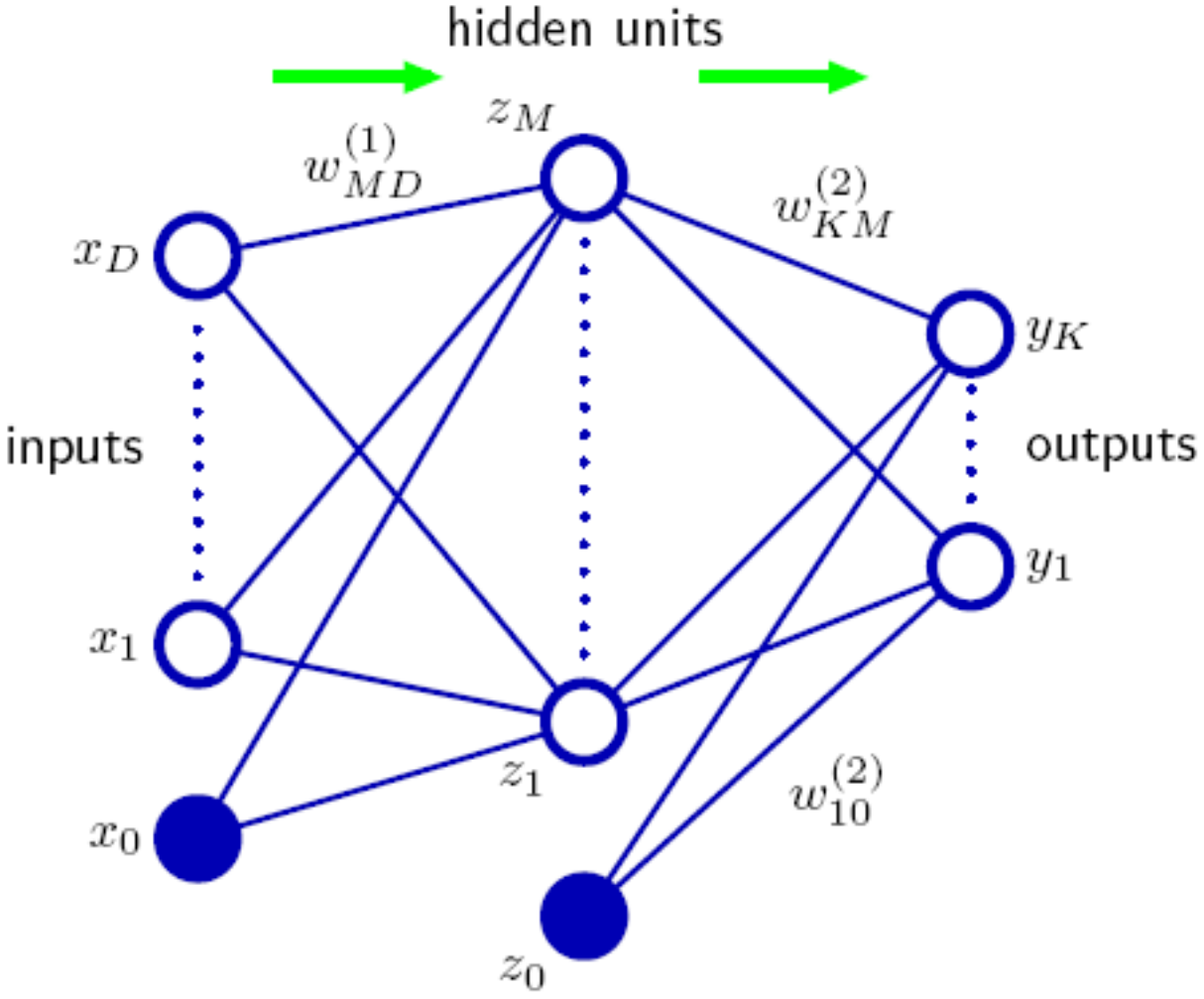
Tutorial Neural Networks and Backpropagation

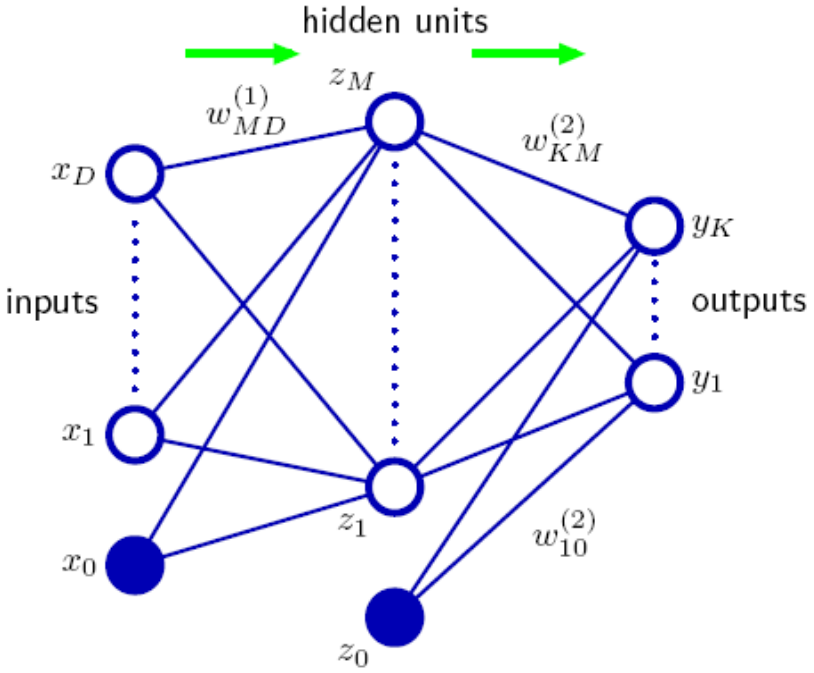
Recommended Reading: Bishop, chapter 5

Two layer FF-NN:



M linear combinations of the input variables $x_1, \ldots, x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$



where $j = 1, \ldots, M$

hidden units :

$$z_j = h(a_j)$$

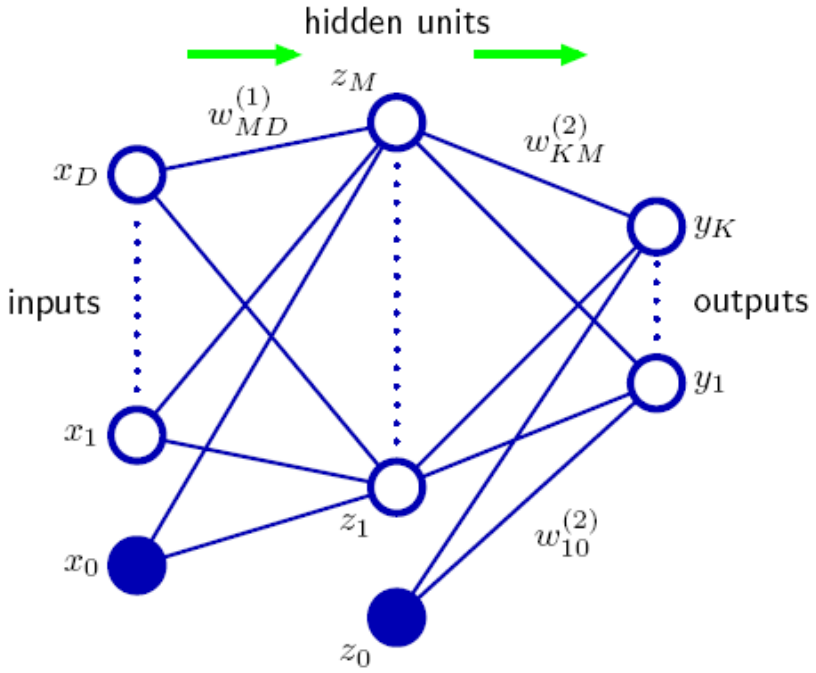
output unit activations

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

where $k = 1, \ldots, K$, and K is the total number of outputs

$$y_k = \sigma(a_k)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}.$$



$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

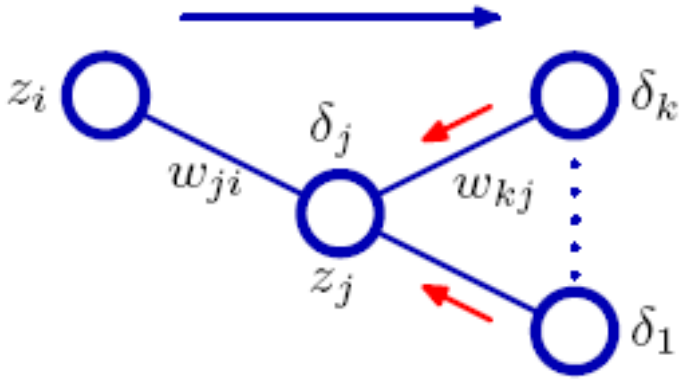
setting x_0 = 1 → absorbing the bias in the notation:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i.$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right).$$

Backpropagation

somewhere in the network:



$$a_j = \sum_i w_{ji} z_i$$

$$z_j = h(a_j).$$

Error function:

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}.$$

useful notation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

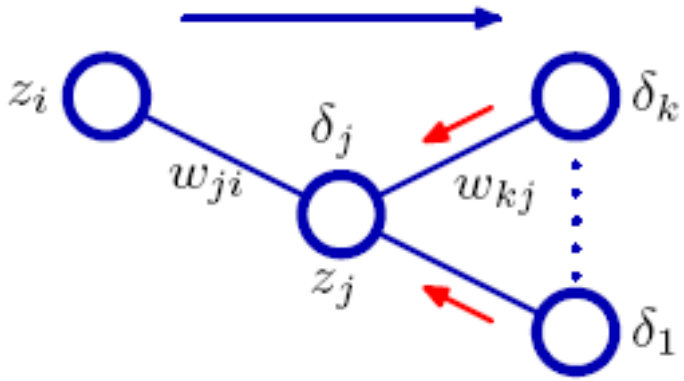
Problem -1(a): Derive the backprogagation formulas!

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}.$$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

$$a_j = \sum_i w_{ji} z_i$$

$$z_j = h(a_j).$$



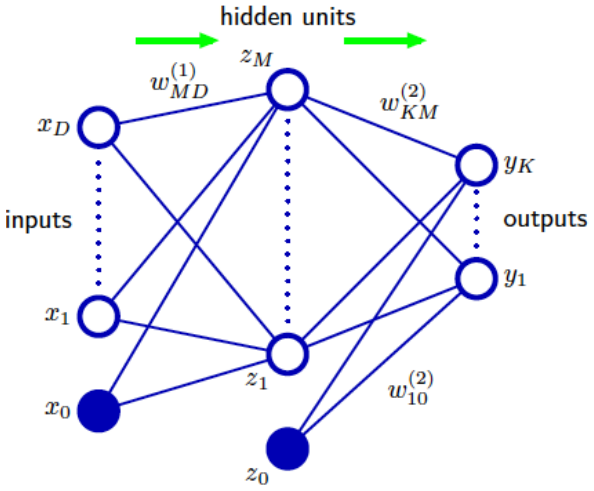
$$\frac{\partial a_j}{\partial w_{ji}} = z_i.$$

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

for the output units, we have $\delta_k = y_k - t_k$

for hidden units $$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = h'(a_j)\sum_k w_{kj}\delta_k$$

Problem -1(b): Apply the backpropagation formula to the simple network



where we have id() as activation on the output neurons and tanh() as activation for the hidden neurons!

the output units have linear activation functions, so that $y_k = a_k$, while the hidden units have logistic sigmoid activation functions given by

$$h(a) \equiv \tanh(a)$$

where

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}.$$

A useful feature of this function is that its derivative can be expressed in a particularly simple form:

$$h'(a) = 1 - h(a)^2.$$

We also consider a standard sum-of-squares error function, so that for pattern n the error is given by

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

where y_k is the activation of output unit k, and t_k is the corresponding target, for a particular input pattern $\mathbf{x}_n$.

For each pattern in the training set in turn, we first perform a forward propagation using

$$\begin{aligned} a_j &= \sum_{i=0}^{D} w_{ji}^{(1)} x_i \\ z_j &= \tanh(a_j) \\ y_k &= \sum_{j=0}^{M} w_{kj}^{(2)} z_j. \end{aligned}$$

source: Bishop

Next we compute the δ's for each output unit using

$$\delta_k = y_k - t_k.$$

Then we backpropagate these to obtain δs for the hidden units using

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k.$$

Finally, the derivatives with respect to the first-layer and second-layer weights are given by

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j.$$

Problem 0: Analyze the **symmetries in the weight space**!

5.1.1 Weight-space symmetries

One property of feed-forward networks, which will play a role when we consider Bayesian model comparison, is that multiple distinct choices for the weight vector $\mathbf{w}$ can all give rise to the same mapping function from inputs to outputs (Chen *et al.*, 1993). Consider a two-layer network of the form shown in Figure 5.1 with M hidden units having 'tanh' activation functions and full connectivity in both layers. If we change the sign of all of the weights and the bias feeding into a particular hidden unit, then, for a given input pattern, the sign of the activation of the hidden unit will be reversed, because 'tanh' is an odd function, so that $\tanh(-a) = -\tanh(a)$. This transformation can be exactly compensated by changing the sign of all of the weights leading out of that hidden unit. Thus, by changing the signs of a particular group of weights (and a bias), the input–output mapping function represented by the network is unchanged, and so we have found two different weight vectors that give rise to the same mapping function. For M hidden units, there will be M such 'sign-flip' symmetries, and thus any given weight vector will be one of a set 2^M equivalent weight vectors .

Similarly, imagine that we interchange the values of all of the weights (and the bias) leading both into and out of a particular hidden unit with the corresponding values of the weights (and bias) associated with a different hidden unit. Again, this clearly leaves the network input–output mapping function unchanged, but it corresponds to a different choice of weight vector. For M hidden units, any given weight vector will belong to a set of $M!$ equivalent weight vectors associated with this interchange symmetry, corresponding to the $M!$ different orderings of the hidden units. The network will therefore have an overall weight-space symmetry factor of $M!2^M$. For networks with more than two layers of weights, the total level of symmetry will be given by the product of such factors, one for each layer of hidden units.

It turns out that these factors account for all of the symmetries in weight space (except for possible accidental symmetries due to specific choices for the weight values). Furthermore, the existence of these symmetries is not a particular property of the 'tanh' function but applies to a wide range of activation functions (Kůrková and Kainen, 1994). In many cases, these symmetries in weight space are of little practical consequence, although in Section 5.7 we shall encounter a situation in which we need to take them into account.

source: Bishop

1 Multiple targets

Problem 1: Consider a regression problem involving multiple target variables in which it is assumed that the distribution of the targets, conditioned on the input vector $\boldsymbol{x}$, is a Gaussian of the form

$$p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}), \boldsymbol{\Sigma})$$

where $\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$ is the output of a neural network with input vector $\boldsymbol{x}$ and a weight vector $\boldsymbol{w}$, and $\boldsymbol{\Sigma}$ is the covariance of the assumed Gaussian noise on the targets. Given a set of independent observations of $\boldsymbol{x}$ and $\boldsymbol{z}$, write down the error function that must be minimised in order to find the maximum likelihood solution for $\boldsymbol{w}$, if we assume that $\boldsymbol{\Sigma}$ is fixed and known. Now assume that $\boldsymbol{\Sigma}$ is also to be determined from the data and write down an expression for the maximum likelihood solution for $\boldsymbol{\Sigma}$. Note that the optimisations of $\boldsymbol{w}$ and $\boldsymbol{\Sigma}$ are now coupled

remark: having a non-diagonal Σ just just means that the components of the output vector may be correlated among themselves.
The training examples $\{(x^{(n)},z^{(n)})\}$ are still iid

remark2: since we assume that $z = y + \epsilon$ (where $\epsilon \sim N(0, \Sigma)$, resulting in $p(z|x,w) = N(z|y(x,w),\Sigma)$) ("the training output z is a Gaussian noised version of the actual output y(x,w)") this practically means the same as "the actual output y(x,w) is a Gaussian noised version of the training output z":
$y = z - \epsilon$

, we get for the likelihood

$$\prod_{i=1}^{N} p(\boldsymbol{z}_i \mid \boldsymbol{x}_i, \boldsymbol{w}) = \prod_i \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)\right)$$

and therefore, one needs to minimise

$$\sum_i \frac{1}{2}(\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)$$

This only works, if $\boldsymbol{\Sigma}$ is known. If not, one also needs to use maximum likelihood to find a good estimate of the covariance matrix. This must be done iteratively, because $\boldsymbol{\Sigma}$ depends on the weights which change according to the backpropagation algorithm. Using equations (51) and (55) from the matrix cookbook, one gets the well known maximum likelihood estimate for $\boldsymbol{\Sigma}$

$$\Sigma = 1/N \sum_i (\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)(\boldsymbol{y}(\boldsymbol{x}_i, \boldsymbol{w}) - \boldsymbol{z}_i)^T$$

$$\frac{\partial \ln|\det(\mathbf{X})|}{\partial \mathbf{X}} = (\mathbf{X}^{-1})^T = (\mathbf{X}^T)^{-1} \quad (51)$$

$$\frac{\partial \mathbf{a}^T \mathbf{X}^{-1} \mathbf{b}}{\partial \mathbf{X}} = -\mathbf{X}^{-T} \mathbf{a}\mathbf{b}^T \mathbf{X}^{-T} \quad (55)$$

2 Error functions

Problem 2: Show that maximising likelihood for a multi-class neural network model in which the network outputs have the interpretation $y_k(\boldsymbol{x}, \boldsymbol{w}) = p(z_k = 1 \mid \boldsymbol{x})$ is equivalent to the minimisation of the cross-entropy error function.

We assume a 1-of-K coding scheme for the multi-class model, thus having K output neurons. Given N i.i.d. input–target pairs, we can write the likelihood function as

$$\prod_{n=1}^{N} \prod_{k} y_k(\boldsymbol{x}_n, \boldsymbol{w})^{z_{nk}}$$

The negative log likelihood thereof gives us

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \ln y_k(\boldsymbol{x}_n, \boldsymbol{w})$$

Theory to all the problems: see Bishop 5.2

Regression problem with FF neural networks:

- $p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$
- N iid training examples: → $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta).$
- - log: → $\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$
- → $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$
- > 1 - dimensional output: → $p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(\mathbf{t}|\mathbf{y}(\mathbf{x}, \mathbf{w}), \beta^{-1}\mathbf{I}\right)$
- $\frac{\partial E}{\partial a_k} = y_k - t_k$
- Activation-function on output neurons: Identity

Binary classification problem with FF neural networks:

- Activation-function on output neurons: sigmoid: $y = \sigma(a) \equiv \frac{1}{1+\exp(-a)}$
- $y(\mathbf{x}, \mathbf{w})$: conditional probability $p(\mathcal{C}_1|\mathbf{x})$
- $p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$
- N iid training examples, neg. log → cross entropy error:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

- $\frac{\partial E}{\partial a_k} = y_k - t_k$

remark: for standard binary classification we have only one output neuron, thus the index k is not required.

Multiple independent binary classification problems with FF neural networks:

- Activation-function on K output neurons each: sigmoid:

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$

- $$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} \left[1 - y_k(\mathbf{x}, \mathbf{w})\right]^{1-t_k} .$$

- N iid training examples, neg. log → cross entropy error:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\}$$

- $$\frac{\partial E}{\partial a_k} = y_k - t_k$$

Modeling advantage compared to multiclass linear classification models
(e.g. multiclass logistic regression (Bishop 4.3.4):
„shared“ weights in first layer of neural network → first layer: „nonlinear feature extraction“

Multi-class classification problems with FF neural networks:

- Activation-function on K output neurons : „K-class sigmoid-generalization“ == softmax function:

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

- 1 of K coding schema, N iid training examples:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} \prod_k y_k(\boldsymbol{x}_n, \boldsymbol{w})^{t_{nk}}$$

- ne. log → multiclass cross entropy error:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\boldsymbol{x}_n, \boldsymbol{w})$$

- $$\frac{\partial E}{\partial a_k} = \sum_{n=1}^{N} y_k(\boldsymbol{x}_n, \boldsymbol{w}) - t_{nk}$$

Problem 3: Show that the derivative of the error function

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N} \{z_n \ln y_n + (1 - z_n) \ln(1 - y_n)\}$$

(y_n denotes $y(\boldsymbol{x}_n, \boldsymbol{w})$) with respect to the activation a_k for the output unit having a logistic sigmoid activation function satisfies

$$\frac{\partial E}{\partial a_k} = y_k - z_k$$

$$\frac{\partial E}{\partial a_k} = -[\sum_{n=1}^{N} z_n \frac{1}{y_n}\frac{\partial y_n}{\partial a_k} - (1-z_n)\frac{1}{1-y_n}\frac{\partial y_n}{\partial a_k}]$$

where $y_n = \sigma(a_n))$. Using the result from the first part of exercise 2 $\partial/\partial x \sigma(x) = \sigma(x)(1-\sigma(x))$ we see that

$$\frac{\partial y_n}{\partial a_k} = y_n(1-y_n)\delta_{nk}$$

and thus

$$\frac{\partial E}{\partial a_k} = -z_k(1-y_k) + (1-z_k)y_k = y_k - z_k$$

Problem 4: Show that the derivative of the standard multi-class error function

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk} \ln y_k(\boldsymbol{x}_n, \boldsymbol{w})$$

with respect to the activation a_k for output units having a softmax activation function satisfies

$$\frac{\partial E}{\partial a_k} = \sum_{n=1}^{N} y_k(\boldsymbol{x}_n, \boldsymbol{w}) - z_{nk}$$

Because k is also used for indexing the K components of one result vector, we will look at $\frac{\partial E}{\partial a_l}$. Also we first want to derive $\frac{\partial \sum_k z_{nk} \ln y_k(x_n,w)}{\partial a_l}$. So

$$\begin{aligned}
\frac{\partial \sum_k z_{nk} \ln y_k(\boldsymbol{x}_n, \boldsymbol{w})}{\partial a_l} &= \frac{z_{nl}}{y_l(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\exp(a_l)(\sum_k \exp(a_k)) - \exp(a_k)^2}{(\sum_k \exp(a_l))^2} - \sum_{k \neq l} \frac{z_{nk}}{y_k(\boldsymbol{x}_n, \boldsymbol{w})} \frac{\exp(a_k)\exp(a_l)}{(\sum_k \exp(a_k))^2} \\
&= \frac{z_{nl}}{y_l(\boldsymbol{x}_n, \boldsymbol{w})} y_l(\boldsymbol{x}_n, \boldsymbol{w}) - \sum_k \frac{z_{nk}}{y_k(\boldsymbol{x}_n, \boldsymbol{w})} y_k(\boldsymbol{x}_n, \boldsymbol{w}) y_l(\boldsymbol{x}_n, \boldsymbol{w}) \\
&= z_{nl} - y_l(\boldsymbol{x}_n, \boldsymbol{w})
\end{aligned}$$

We have used the fact $\sum_k t_{nk} = 1$. Summing over all training cases, one gets

$$\frac{\partial E}{\partial a_l} = \sum_{n=1}^{N} y_l(\boldsymbol{x}_n, \boldsymbol{w}) - z_{nl}$$

3 Robust classification

Problem 5: Consider a binary classification problem in which the target values are $z \in \{0, 1\}$, with a network output $y(\boldsymbol{x}, \boldsymbol{w})$ that represents $p(z = 1 \mid \boldsymbol{x})$, 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 well known error function for binary classification is obtained when $\varepsilon = 0$. Note that this error function makes the model robust to incorrectly labelled data, in contrast to the usual error function.

To solve this problem, we introduce a new random variable l_i, denoting whether the class label for input i is correct (c) or not (w), thus $p(l_i = w) = \varepsilon$. So for a given input $\boldsymbol{x}_i$ we get

$$p(z_i \mid \boldsymbol{x}_i) = p(z_i, l_i = c \mid \boldsymbol{x}_i) + p(z_i, l_i = w \mid \boldsymbol{x}_i)$$

Using $p(a, b) = p(a \mid b)p(b)$ we get

$$\begin{aligned} p(z_i \mid \boldsymbol{x}_i) &= p(z_i \mid l_i = c, \boldsymbol{x}_i)p(l_i = c) + p(z_i \mid l_i = w, \boldsymbol{x}_i)p(l_i = w) \\ &= \left(y(\boldsymbol{x}, \boldsymbol{w})^{z_i}(1 - y(\boldsymbol{x}, \boldsymbol{w})^{(1-z_i)})\right)(1 - \varepsilon) + \left(y(\boldsymbol{x}, \boldsymbol{w})^{(1-z_i)}(1 - y(\boldsymbol{x}, \boldsymbol{w})^{z_i})\right)\varepsilon \end{aligned}$$

The likelihood now follows straightforward

$$\prod_i \left(y(\boldsymbol{x}_n, \boldsymbol{w})^{z_i}(1 - y(\boldsymbol{x}_n, \boldsymbol{w})^{(1-z_i)})\right)(1 - \varepsilon) + \left(y(\boldsymbol{x}_n, \boldsymbol{w})^{(1-z_i)}(1 - y(\boldsymbol{x}_n, \boldsymbol{w})^{z_i})\right)\varepsilon$$

Taking the negative log thereof gives the error function to be minimised. Obviously this resembles the standard binary cross entropy error function if $\varepsilon = 0$.