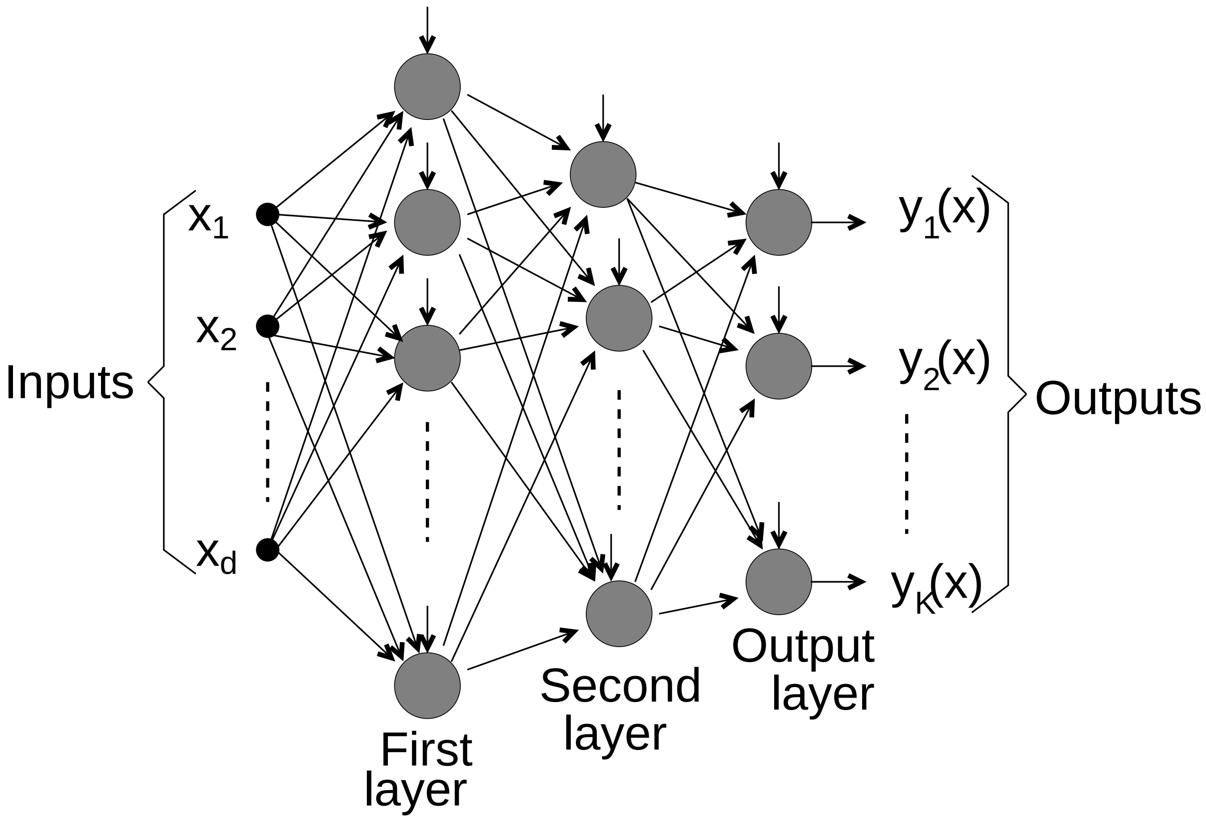
University of Cambridge

MPhil in Computer Speech Text & Internet Technology

Module: Speech Processing II

Lecture 15: Neural Network Introduction



Lent 2003

Introduction

So far in this module we have considered a variety of options for describing the observations generated by an HMM. These include:

- continuous density HMMs with Gaussian mixture models;
- discrete HMMs with multiple codebooks;
- semi-Continuous HMMs.

An alternative approach is to use a neural network. This lecture will introduce the most commonly used form of Neural Network, the *multi-layer perceptron*. Other forms of network are possible. These include:

- **Radial basis functions**;
- **Kohonen networks**.

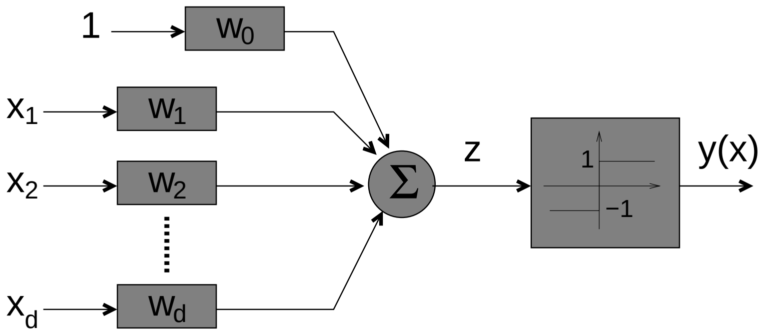
If you are interested in these schemes see the I10 lecture notes at

`http://svr-www.eng.cam.ac.uk/ mjfg/local/i10.html`

This lecture will look at **multi-layer perceptrons** for classification.

Single Layer Perceptron

The simplest form of network is the *single layer perceptron*. The typical form uses a *threshold activation function*, as shown below.



The d-dimensional input vector $\mathbf{x}$ and scalar value z are related by

$$z = \mathbf{w}'\mathbf{x} + w_0$$

z is then fed to the activation function to yield $y(\mathbf{x})$. The parameters of this system are

- **weights**: $\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$, selects the *direction* of the decision boundary
- **bias**: w_0, determines the *position* of the decision boundary.

We have already seen this form when examining linear decision boundaries.

Single Layer Perceptron (cont)

The parameters are often combined into a single composite vector, $\tilde{\mathbf{w}}$, and the input vector extended, $\tilde{\mathbf{x}}$.

$$\tilde{\mathbf{w}} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}; \qquad \tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

We can then write

$$z = \tilde{\mathbf{w}}'\tilde{\mathbf{x}}$$

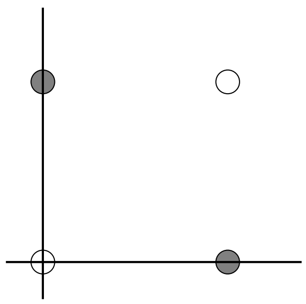
The task is to train the set of model parameters $\tilde{\mathbf{w}}$. For this example a *decision boundary* is placed at $z = 0$. The decision rule is

$$y(\mathbf{x}) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

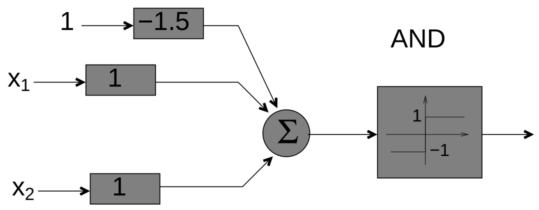
In the Speech Processing I module the percepton training algorithm and least squares estimation were described for training this form of model.

Limitations of SLPs

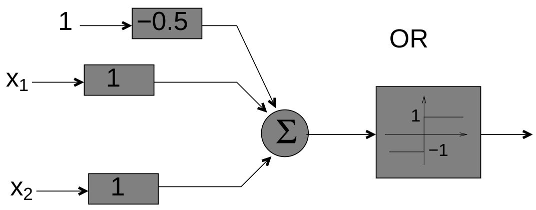
Perceptrons were very popular until the 1960's when it was realised that it couldn't solve the XOR problem.



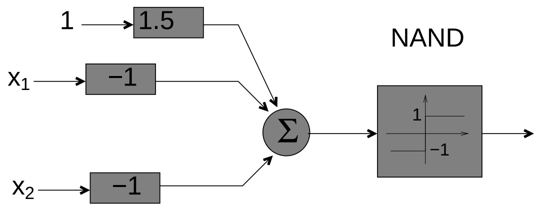
We can use perceptrons to solve the binary logic operators AND, OR, NAND, NOR.



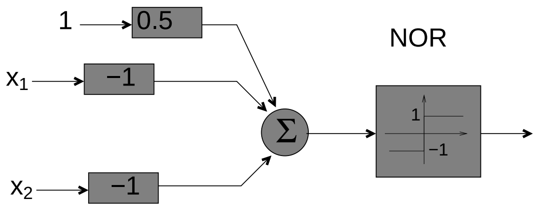
(a) AND operator



(b) OR operator



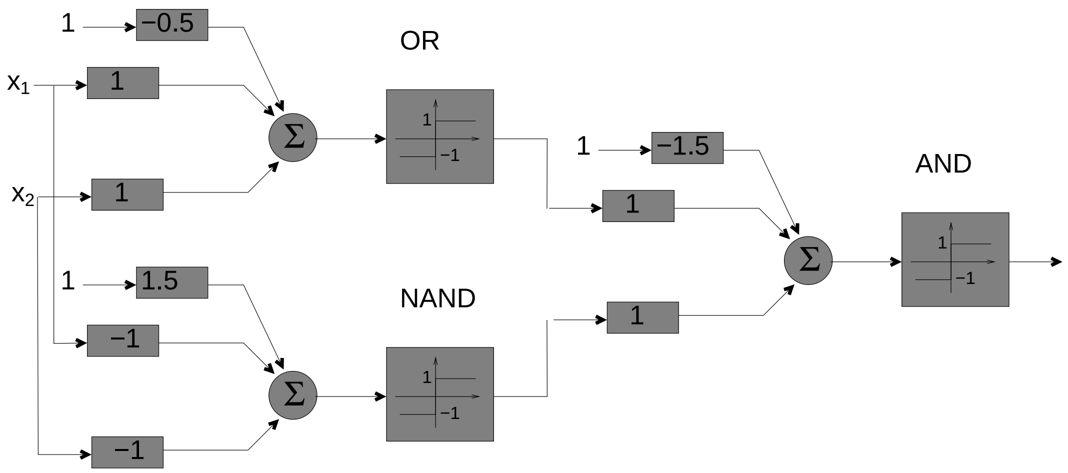
(c) NAND operator



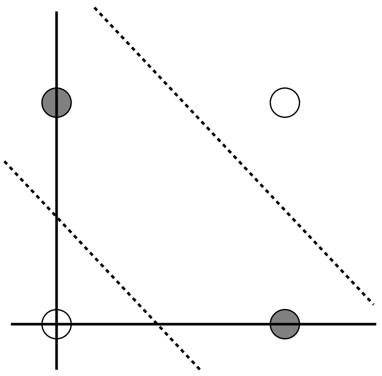
(d) NOR operator

XOR (cont)

But XOR may be written in terms of AND, NAND and OR gates



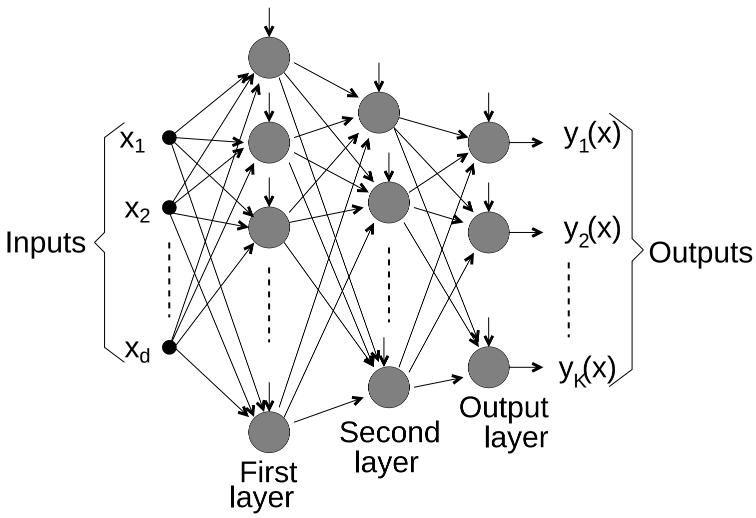
This yields the decision boundaries



So XOR can be solved using a two-layer network. The problem is how to train multi-layer perceptrons. In the 1980's an algorithm for training such networks was proposed, *error back propagation*.

Multi-Layer Perceptron

From the previous slide we need a multi-layer perceptron to handle the XOR problem. More generally multi-layer perceptrons allow a neural network to perform arbitrary mappings.

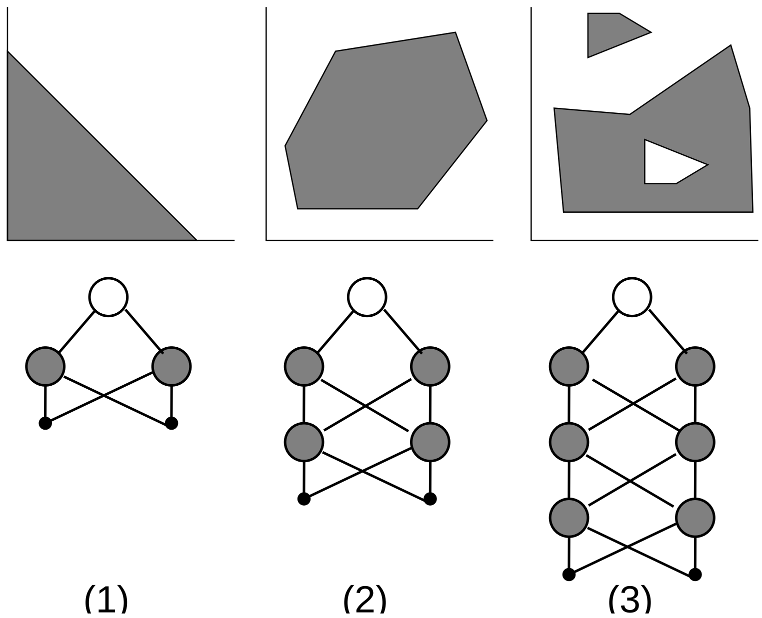


A 2-hidden layer neural network is shown above. The aim is to map an input vector $\mathbf{x}$ into an output $\mathbf{y}(\mathbf{x})$. The layers may be described as:

- **Input** layer: accepts the data vector or pattern;
- **Hidden** layers: one or more layers. They accept the output from the previous layer, weight them, and pass through a, normally non-linear, activation function.
- **Output** layer: takes the output from the final hidden layer weights them, and possibly pass through an output non-linearity, to produce the target values.

Possible Decision Boundaries

The nature of the decision boundaries that may be produced varies with the network topology. Here only threshold (see the single layer perceptron) activation functions are used.



There are three situations to consider

1. **Single layer**: this is able to position a hyperplane in the input space.

2. **Two layers** (one hidden layer): this is able to describe a decision boundary which surrounds a single convex region of the input space.

3. **Three layers** (two hidden layers): this is able to to generate arbitrary decision boundaries

Note: any decision boundary can be approximated arbitrarily closely by a two layer network having sigmoidal activation functions.

Number of Hidden Units

From the previous slide we can see that the number of hidden layers determines the decision boundaries that can be generated. In choosing the number of layers the following considerations are made.

- Multi-layer networks are harder to train than single layer networks.
- A two layer network (one hidden) with sigmoidal activation functions can model any decision boundary.

Two layer networks are most commonly used in pattern recognition (the hidden layer having sigmoidal activation functions).

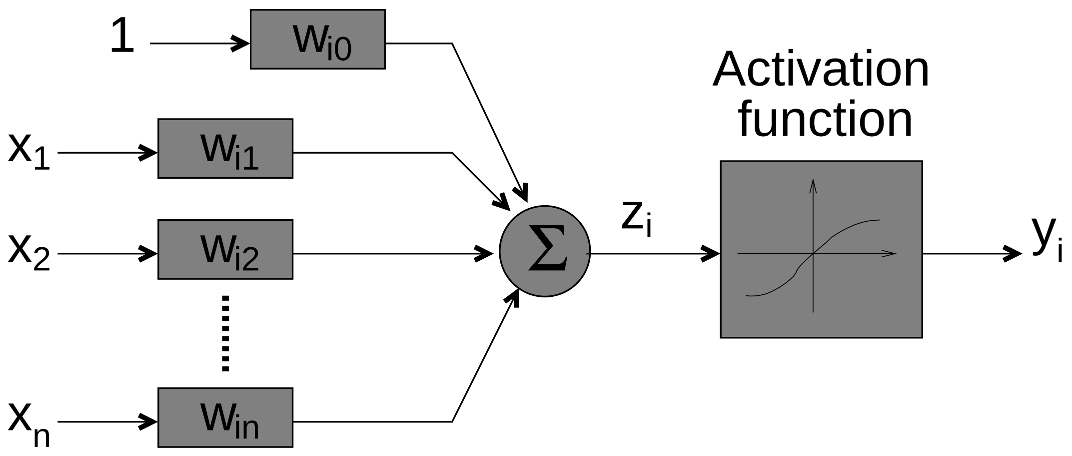
How many units to have in each layer?

- The number of output units is determined by the number of output classes.
- The number of inputs is determined by the number of input dimensions
- The number of hidden units is a design issue. The problems are:
  - too few, the network will not model complex decision boundaries;
  - too many, the network will have poor *generalisation.*

Hidden Layer Perceptron

The form of the hidden, and the output, layer perceptron is a generalisation of the single layer perceptron. Now the weighted input is passed to a general *activation function*, rather than a threshold function.

Consider a single perceptron. Assume that there are n units at the previous level.



The output from the perceptron, y_i may be written as

$$y_i = \phi(z_i) = \phi(w_{i0} + \sum_{j=1}^{n} w_{ij}x_j)$$

where $\phi()$ is the activation function.

We have already seen one example of an activation function the threshold function. Other forms are also used in multi-layer perceptrons.

Activation Functions

There are a variety of non-linear activation functions that may be used. Consider the general form

$$y_j = \phi(z_j)$$

and there are n units, perceptrons, for the *current* level.

- **Heaviside** (or step) function:
$$y_j = \begin{cases} 0, & z_j < 0 \\ 1, & z_j \geq 0 \end{cases}$$
These are sometimes used in *threshold* units, the output is binary.
- **Sigmoid** (or logistic regression) function:
$$y_j = \frac{1}{1 + \exp(-z_j)}$$
The output is continuous, $0 \leq y_j \leq 1$.
- **Softmax** (or normalised exponential or generalised logistic) function:
$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{n} \exp(z_i)}$$
The output is positive and the sum of all the outputs at the current level is 1, $0 \leq y_j \leq 1$.
- **Hyperbolic tan** (or tanh) function:
$$y_j = \frac{\exp(z_j) - \exp(-z_j)}{\exp(z_j) + \exp(-z_j)}$$
The output is continuous, $-1 \leq y_j \leq 1$.

Training Criteria

A variety of training criteria may be used. Assuming we have supervised training examples

$$\{\{\mathbf{x}_1, \mathbf{t}(\mathbf{x}_1)\} \ldots, \{\mathbf{x}_n, \mathbf{t}(\mathbf{x}_n)\}\}$$

The target values, $\mathbf{t}(\mathbf{x})$, are K-dimensional.

One standard example is:

- **Least squares error**: one of the most common training criteria.

$$\begin{aligned} E &= \frac{1}{2} \sum_{p=1}^{n} ||\mathbf{y}(\mathbf{x}_p) - \mathbf{t}(\mathbf{x}_p)||^2 \\ &= \frac{1}{2} \sum_{p=1}^{n} \sum_{i=1}^{K} (y_i(\mathbf{x}_p) - t_i(\mathbf{x}_p))^2 \end{aligned}$$

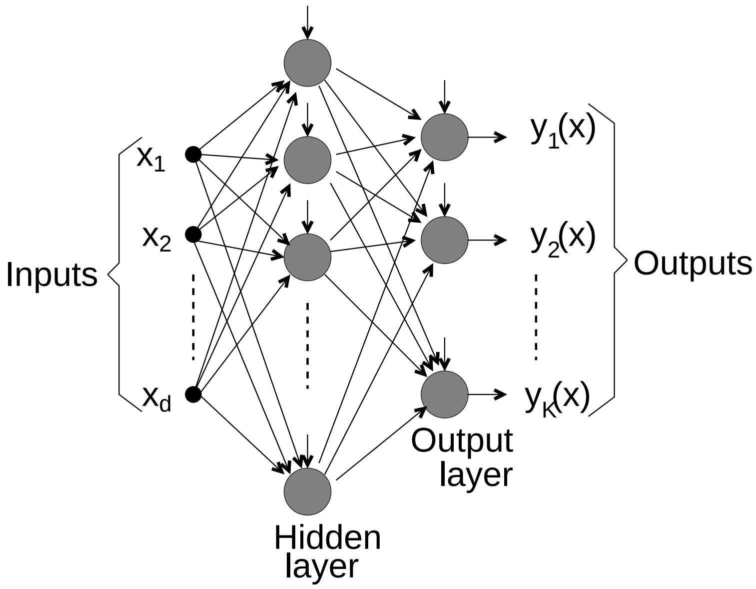
  This may be derived from considering the targets as being corrupted by zero-mean Gaussian distributed noise.

Alternative forms are (for reference):

- **cross-Entropy for two classes**: consider the case when $t(\mathbf{x})$ is binary (and softmax output), or more generally
- **cross-Entropy for multiple classes**: is related to Kullback-Leibler distance between the distribution of the output and the target.

Error Back Propagation

Interest in multi-layer perceptrons (MLPs) resurfaced with the development of the *error back propagation* algorithm. This allows multi-layer perceptons to be simply trained.



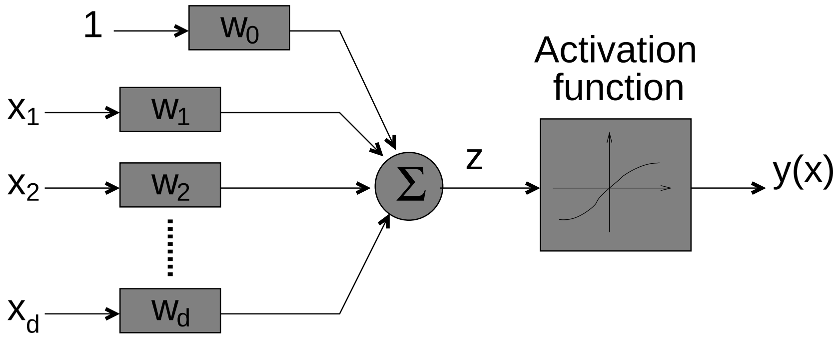
A single hidden layer network is shown above. As previously mentioned with sigmoidal activation functions arbitrary decision boundaries may be obtained with this network topology.

The error back propagation algorithm is based on *gradient descent*. Hence the activation function must be differentiable. Thus *threshold* and *step* units will not be considered. We need to be able to compute the derivative of the error function with respect to the weights of *all* layers.

All gradients in the next few slides are evaluated at the *current* model parameters.

Single Layer Perceptron

Rather than examine the multi-layer case instantly, consider the following single layer perceptron.



We would like to minimise (for example) the square error between the target of the output, $t(\mathbf{x}_p)$, and the current output value $y(\mathbf{x}_p)$. Assume that the activation function is known to be a sigmoid function. The cost function may be written as

$$E = \frac{1}{2}\sum_{p=1}^{n}(y(\mathbf{x}_p) - t(\mathbf{x}_p))^2 = \sum_{p=1}^{n} E^{(p)}$$

To simplify notation, we will only consider a single observation $\mathbf{x}$ with associated target values $t(\mathbf{x})$ and current output from the network $y(\mathbf{x})$. The error with this single observation is denoted E.

The first question is how does the error change as we alter $y(\mathbf{x})$.

$$\frac{\partial E}{\partial y(\mathbf{x})} = y(\mathbf{x}) - t(\mathbf{x})$$

But we are not interested in $y(\mathbf{x})$ - how do we find the effect of varying the weights?

SLP Training (cont)

We can calculate the effect that a change in z has on the error using the chain rule

$$\frac{\partial E}{\partial z} = \left(\frac{\partial E}{\partial y(\mathbf{x})}\right)\left(\frac{\partial y(\mathbf{x})}{\partial z}\right)$$

However what we really want is the change of the error rate with the weights (the parameters that we want to learn).

$$\frac{\partial E}{\partial w_i} = \left(\frac{\partial E}{\partial z}\right)\left(\frac{\partial z}{\partial w_i}\right)$$

The error function therefore depends on the weight as

$$\frac{\partial E}{\partial w_i} = \left(\frac{\partial E}{\partial y(\mathbf{x})}\right)\left(\frac{\partial y(\mathbf{x})}{\partial z}\right)\left(\frac{\partial z}{\partial w_i}\right)$$

All these expressions are known so we can write

$$\frac{\partial E}{\partial w_i} = (y(\mathbf{x}) - t(\mathbf{x}))y(\mathbf{x})(1 - y(\mathbf{x}))x_i$$

This has been computed for a single observation. We are interested in terms of the complete training set. We know that the total errors is the sum of the individual errors, so

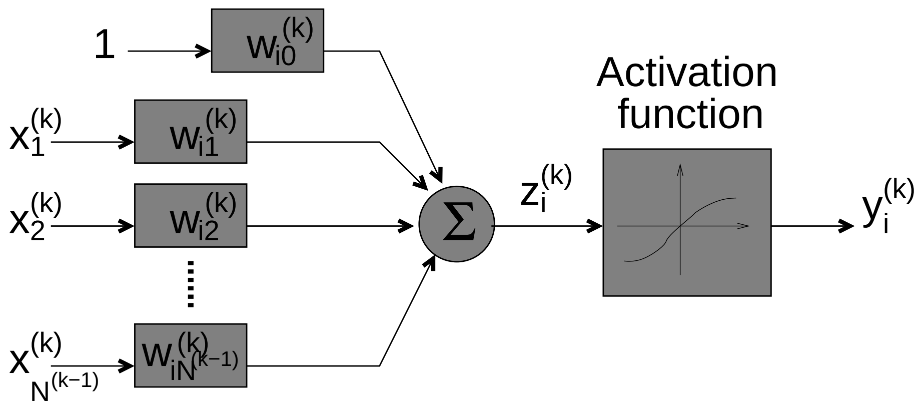
$$\boldsymbol{\nabla} E = \sum_{p=1}^{n}(y(\mathbf{x}_p) - t(\mathbf{x}_p))y(\mathbf{x}_p)(1 - y(\mathbf{x}_p))\tilde{\mathbf{x}}_p$$

So for a single layer we can use gradient descent schemes to find the "best" weight values.

However we want to train multi-layer perceptrons!

Error Back Propagation Algorithm

Now consider a particular node, i, of hidden layer k. Using the previously defined notation, the input to the node is $\tilde{\mathbf{x}}^{(k)}$ and the output $y_i^{(k)}$.



From the previous section we can simply derive the rate of change of the error function with the weights of the output layer. We need to now examine the rate of change with the k^{th} hidden layer weights.

A general error criterion, E, will be used. Furthermore we will not assume that $y_j^{(k)}$ only depends on the $z_j^{(k)}$. For example a softmax function may be used. In terms of the derivations given the output layer will be considered as the $L + 1^{th}$ layer.

The training observations are assumed independent and so

$$E = \sum_{p=1}^{n} E^{(p)}$$

where $E^{(p)}$ is the error cost for the p observation and the observations are $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

EBP Algorithm (cont)

How does the error function vary as we alter the parameters of level k where there are $N^{(k)}$ units.

The following recursion can be used

$$
\begin{aligned}
\boldsymbol{\delta}^{(k)} &= \left(\frac{\partial E}{\partial \mathbf{z}^{(k)}}\right) \\
&= \left(\frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{z}^{(k)}}\right)\left(\frac{\partial E}{\partial \mathbf{z}^{(k+1)}}\right) \\
&= \left(\frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}}\right)\left(\frac{\partial \mathbf{z}^{(k+1)}}{\partial \mathbf{y}^{(k)}}\right)\boldsymbol{\delta}^{(k+1)}
\end{aligned}
$$

where

$$
\frac{\partial E}{\partial \mathbf{z}^{(k)}} = \begin{bmatrix} \frac{\partial E}{\partial z_1^{(k)}} \\ \vdots \\ \frac{\partial E}{\partial z_{N^{(k)}}^{(k)}} \end{bmatrix}, \quad
\frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}} = \begin{bmatrix}
\frac{\partial y_1^{(k)}}{\partial z_1^{(k)}} & \frac{\partial y_2^{(k)}}{\partial z_1^{(k)}} & \cdots & \frac{\partial y_{N^{(k)}}^{(k)}}{\partial z_1^{(k)}} \\
\frac{\partial y_1^{(k)}}{\partial z_2^{(k)}} & \frac{\partial y_2^{(k)}}{\partial z_2^{(k)}} & \cdots & \frac{\partial y_{N^{(k)}}^{(k)}}{\partial z_2^{(k)}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial y_1^{(k)}}{\partial z_{N^{(k)}}^{(k)}} & \frac{\partial y_2^{(k)}}{\partial z_{N^{(k)}}^{(k)}} & \cdots & \frac{\partial y_{N^{(k)}}^{(k)}}{\partial z_{N^{(k)}}^{(k)}}
\end{bmatrix}
$$

The gradient at level k can be written in terms of level $k+1$. We can therefore *back propagate* calculating the gradients.

Gradient descent can then be used to optimise the model parameters.

References

[1] C M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[2] G D Cook and A J Robinson. Boosting the Performance of Connectionist Large-Vocabulary Speech Recognition. In *Proceedings ICSLP*, 1996.

[3] H Hermansky. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87:1738–1752, 1990.

[4] D J Kershaw, M M Hochberg and A J Robinson. Context-Dependent Classes in a Hybrid Recurrent Network-HMM Speech Recognition System. In *Proceedings NIPS*, pages 750–756 1996.

[5] N Morgan and H A Bourlard. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Acedemic Publishers, 1994.

[6] J Neto, L Almeida, M M Hochberg, C Martins, L Nunes, S J Renals and A J Robinson. Unsupervised Speaker-Adaptation For Hybrid HMM-MLP Continuous Speech Recognition System. In *Proceedings Eurospeech*, pages 187-190 1995.

[7] A J Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University, 1989.