Data Mining and Machine Learning

Learning MLP Weights using Error Back-Propagation

Peter Jančovič

Objectives

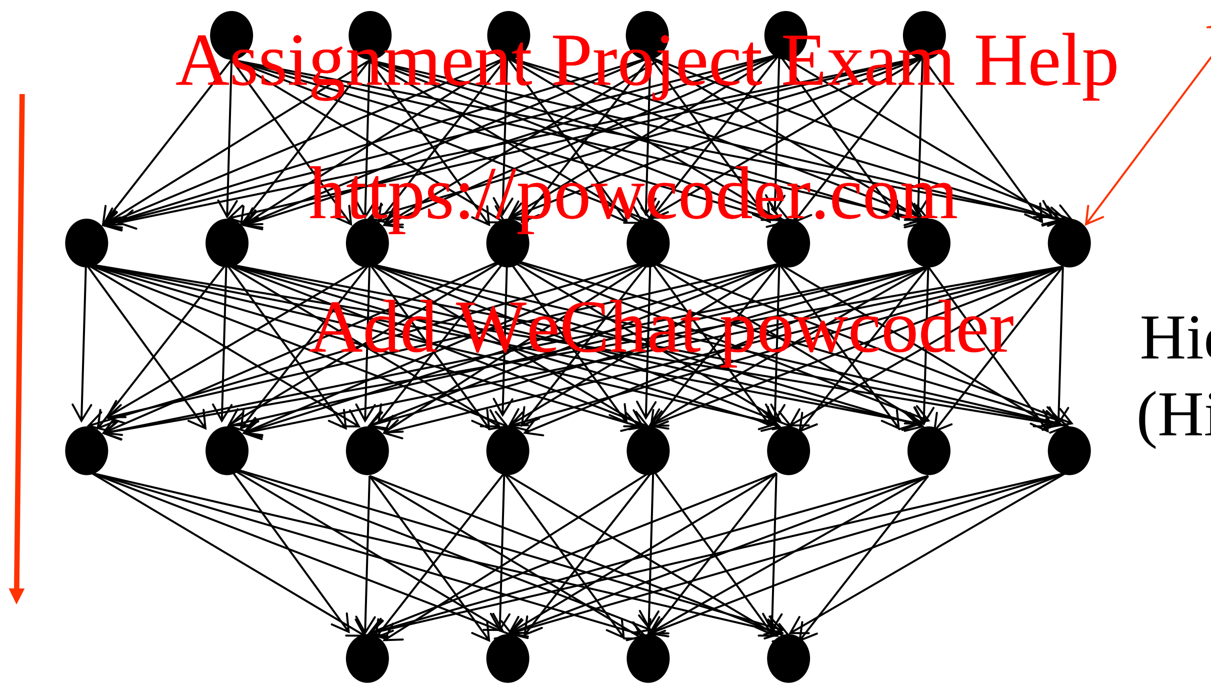
- Outline of the MLP training
  - The error function
  - Optimisation by gradient decent
- The Error Back-Propagation (EBP)
  - Calculating the derivatives
  - Bringing everything together
  - Summary of the EBP algorithm
  - Practical considerations

Feed-forward Neural Networks

<u>Multi-Layer Perceptron</u> - Feed-Forward Neural Network

Input Layer (Input Units)

Artificial neuron



Hidden Layers (Hidden Units)

Output Layer (Output Units)

MLP training

- To define an MLP must decide:
  - Number of layers
  - Number of input units
  - Number of hidden units
  - Number of output units
- Choosing the right numbers of layers and units is a combination of experience and experimentation
- Once these are defined, properties of the MLP are <u>completely</u> defined by the values of the weights
- How do we choose the weight values?

MLP training (continued)

- MLP training needs a set of input vectors ***i*** with corresponding <u>target</u> output vectors ***t*(*i*)**

- Each input vector ***i*** is propagated through the network to produce an output ***o*(*i*)**

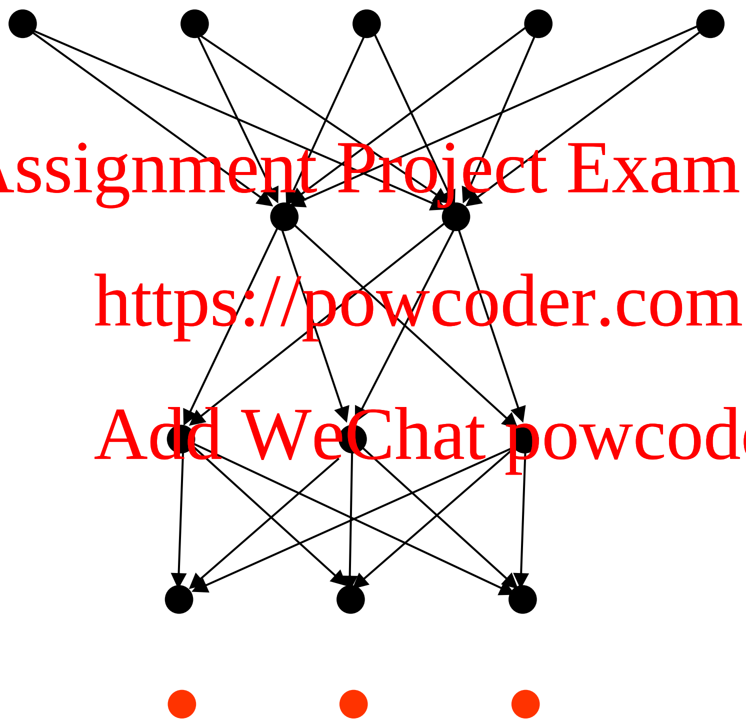
- The error E is the difference between the actual output ***o*(*i*)** and the target output ***t*(*i*)**

$$E = \sum_i \left(o(i) - t(i)\right)^2$$

- Objective of training is to learn the weights which minimise the average error over the training set

Error Back-Propagation

$$i = i_1,\ldots,i_5$$



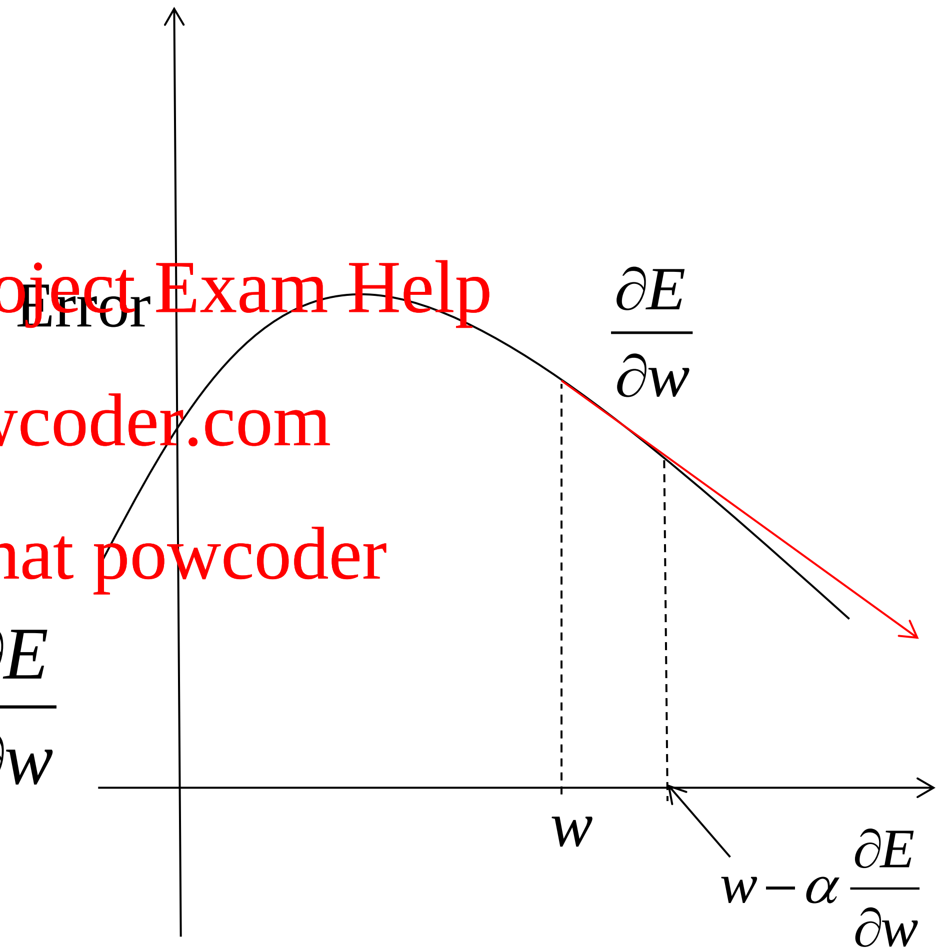
$$t(i) = t(i)_1,\ldots,\ t(i)_3$$

MLP training (continued)

- MLP training uses gradient descent
- For each weight w

  calculate $\dfrac{\partial E}{\partial w}$
- Subtract a proportion of $\dfrac{\partial E}{\partial w}$

  from w



MLP training (continued)

- MLP weights learnt <u>automatically</u> from training data
- Training uses an iterative computational technique called <u>Error-Back-Propagation (EBP)</u>
- There are many variants of EBP

Error-back propagation (EBP)

1. Choose initial values for the weights
2. Propagate each training sample ***i*** through the network to obtain ***o***(***i***). Set ***E*** = |***t***(***i***) – ***o***(***i***)|
3. EBP calculates $\frac{\partial E}{\partial w}$ for each weight w by propagating the error back up through the network
4. When all training patterns have been seen, w is changed by an amount proportional to the average value of $\frac{\partial E}{\partial w}$
5. Repeat until the change in error falls below a threshold

MLP training - the error function

- A **training set** consists of
  - A set of **input** vectors $i^1, \dots, i^N$, where the dimension of i^n is equal to the number of MLP input units
  - For each n, a **target** vector t^n, where the dimension of t^n is equal to the number of output units
- Let o^n be the output vector corresponding to i^n. Define the **error** E by

$$E_n = \frac{1}{2}\|o^n - t^n\|^2 = \frac{1}{2}\sum_{j=1}^{J}(o_j^n - t_j^n)^2 \quad (1)$$

$$E = \sum_{n=1}^{N} E_n \quad (2)$$

- The aim is to adjust the MLP parameters to minimize E

Optimization by gradient descent

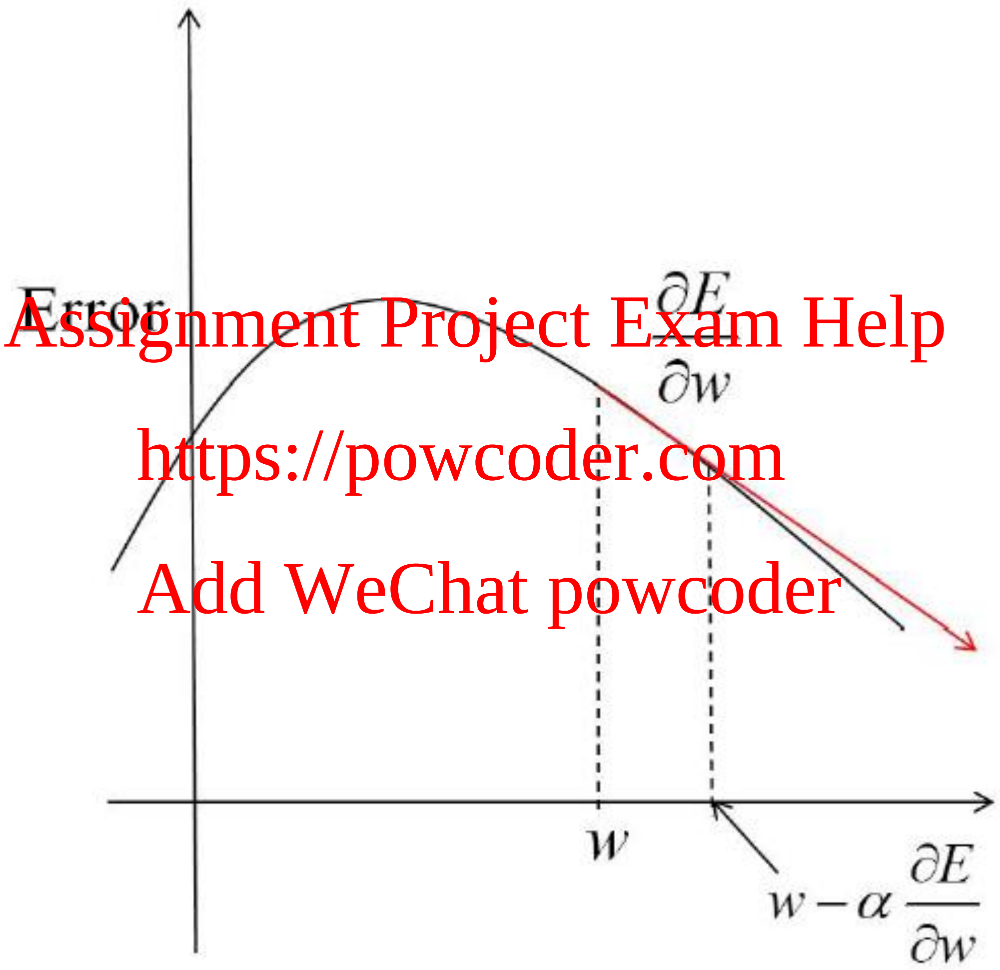
- The parameters of the MLP are the **weights** $w_{i,j}$
- In Error Back-Propagation, the weights are optimized using gradient descent

- For this, need to calculate the partial derivative

$$\frac{\partial E}{\partial w_{i,j}} \tag{3}$$

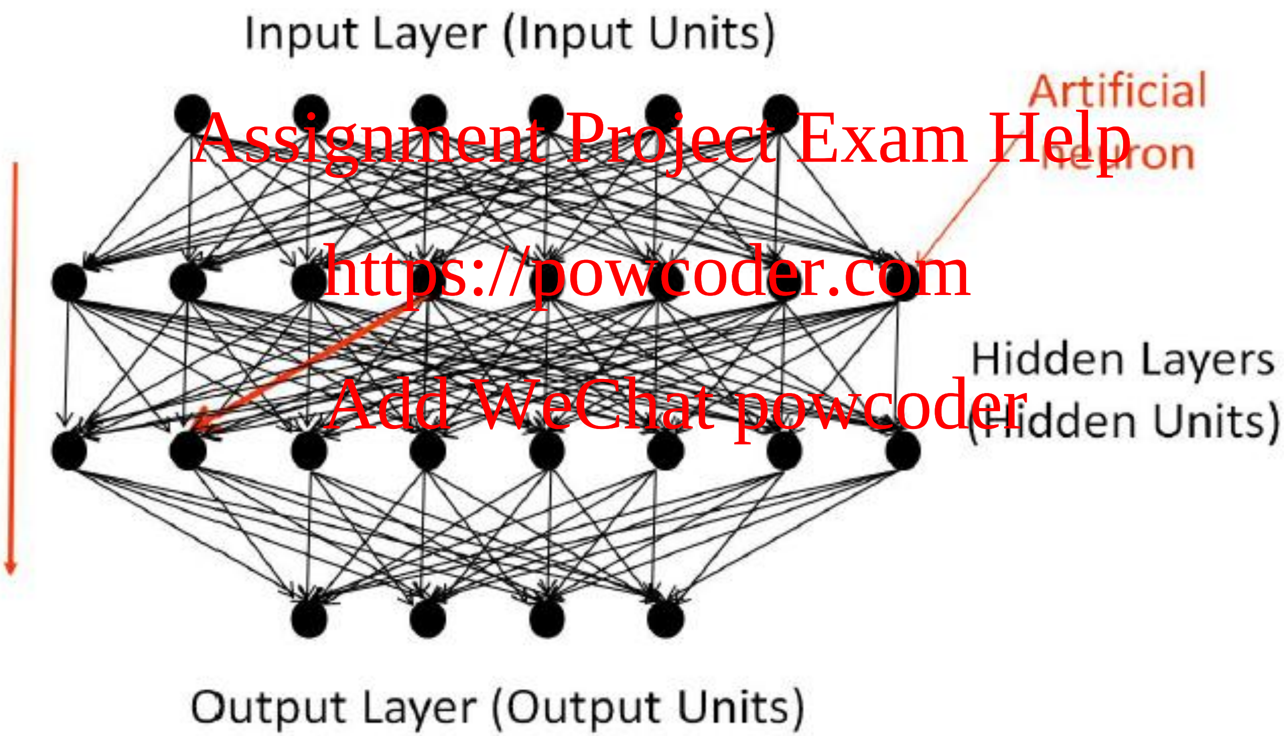
for each weight $w_{i,j}$

Optimization by gradient descent



Calculating the derivatives

- How do we do this for connections deep inside the MLP?



Calculating the derivatives - the chain rule

- Notation:
  - Assume a single training vector, so drop the superscript n
  - The input to the j^{th} unit is net_j
  - The output from the j^{th} unit is o_j
- Assume

$$o_j = \phi(net_j) = \phi(\sum_{k=1}^{K} w_{k,j} o_k) \quad (4)$$

$$\phi(x) = \frac{1}{1+e^{-x}} \quad (5)$$

- Applying the **chain rule** twice

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \quad (6)$$

Calculating the 2nd and 3rd derivatives

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

- The final derivative is easy:

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} (\sum_{k=1}^{K} w_{kj} o_k) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i \tag{7}$$

- ... as is the second derivative:

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \phi(net_j)}{\partial net_j} = \phi(net_j)(1 - \phi(net_j)) \tag{8}$$

- The problem is the first derivative $\frac{\partial E}{\partial o_j}$

Evaluating the first derivative

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

- If the neuron is in the **output** layer then from equation (1)

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial o_j}\frac{1}{2}\sum_{i=1}^{J}(o_i - t_i)^2 = o_j - t_j \qquad (9)$$

- If the neuron is not in the output layer, then it is possible to derive an expression for $\frac{\partial E}{\partial o_j}$ in terms of $\frac{\partial E}{\partial o_k}$, where the k^{th} neuron is in the layer **below** the j^{th} neuron:

$$\frac{\partial E}{\partial o_j} = \sum_k \left(\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k} w_{jk}\right) \qquad (10)$$

The Error Back-Propagation algorithm

Putting everything together:

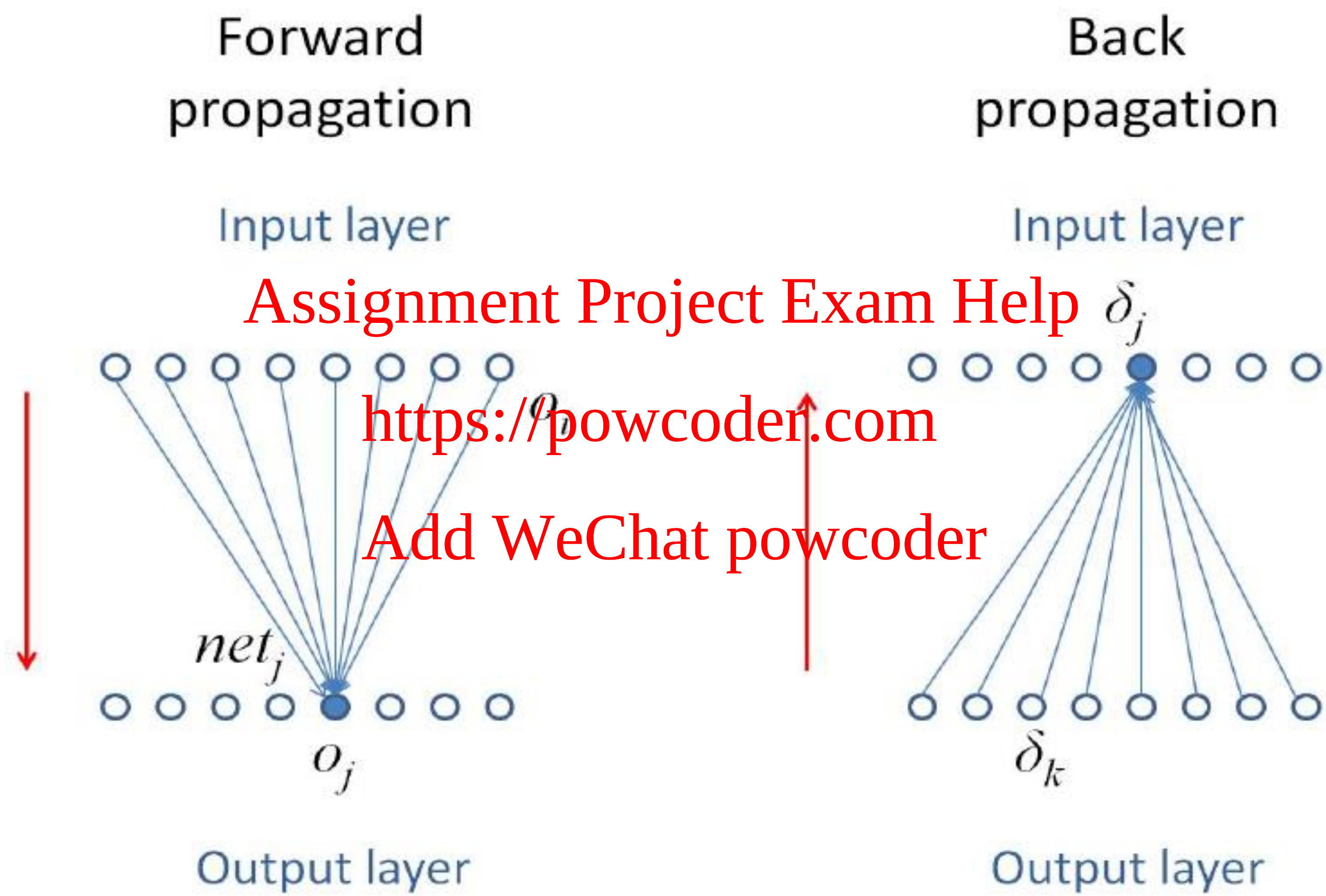
$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i \tag{11}$$

where

$$\delta_j = \begin{cases} (o_j - t_j) o_j (1 - o_j) \text{ if } j \text{ is an output neuron} \\ (\sum_k w_{jk} \delta_k) o_j (1 - o_j) \text{ if } j \text{ is not an output neuron} \end{cases} \tag{12}$$

Sum is over all units k in layer below unit j (closer to output layer)

Error Back-Propagation



The Error Back-Propagation algorithm

Error Back-Propagation (EBP) works as follows:

- Apply a training pattern x to the input layer
- Propagate input **forward** through MLP to obtain output o
- Calculate δ_i for each output neuron using equation (12) for an output neuron

- Propagate the δs **backwards** through the MLP using equation (12) for an "inner" neuron

- For each weight w_{ij} calculate $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j o_i$
- Replace each weight w_{ij} with $w_{ij} + \Delta w_{ij}$

η is called the **learning rate**

Do this for each training pattern

Repeat whole process until reduction in error sufficiently small

Practical considerations

- It is normal to accumulate the Δw_{ij}s over multiple training patterns before updating the weights
- The EBP algorithm is a **gradient descent** algorithm. If is designed to reduce the error E after each iteration. The utility of the final MLP will depend on the initial choice of the w_{ij}s and the quality and quantity of the training data

Summary

- MLP training
- Error Back Propagation (EBP)