

31YB: Lecture 7

Mon, 20 Oct

More on MLP, Back Propagation

Reading:

- Beale & Jackson, Chapter 4
- and most books on neural networks ...

Course Summary so far

- Biologically Inspired Computing
- Learning & Generalization
- Perceptron Learning Rule (PLR)
- Delta Rule (DR)
- History
 - Rise of Neural Nets
 - Fall due to Minsky & Papert
 - Rise due to Multi-layer nets

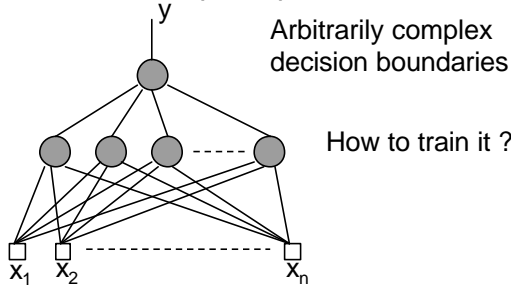
Lecture Summary

Review MLP

Back-propagation (BP) training algorithm

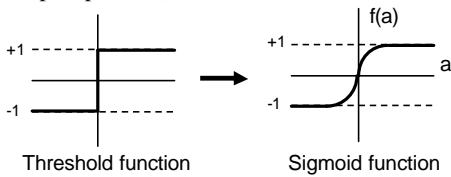
BP Summary

**Recap: Multi-layer Perceptron (MLP)
with threshold perceptron units**



Recap: Simple idea!

- Alter non-linear Perceptron (threshold) activation function so that we can relate the output to the weights in a more informative way (real values of outputs possible)



You'll hear claims for biological plausibility but this mathematical 'fix' is not based on biology!!

**Recap: Sigmoid (S-shaped)
functions -Properties**

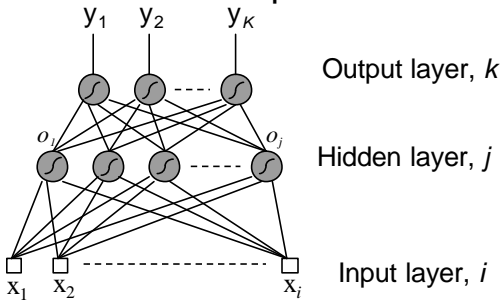
- Approximates a threshold function
- Differentiable (smooth everywhere)
 - hence Delta Rule (DR) applicable
- Positive slope

Popular choice is: $y = f(a) = \frac{1}{1 + e^{-a}}$

Simple derivative (of sigmoidal activation function):

$$f'(a) = f(a)(1 - f(a))$$

Recap: Resulting Multi-layer Perceptron



Above shows a 2-layer MLP, with a single hidden layer. Also, note sigmoid functions on hidden & output units.

How does a weight affect error ?

Network error function (sum-of-squares), summed across all network outputs (from output layer), k :

$$E(\mathbf{w}) = \frac{1}{2} \sum_k (y_{target_k} - y_k)^2$$

The derivative of the error with respect to the weight-connecting (hidden layer) unit j to (output layer) unit k is:

$$\frac{\partial E}{\partial w_{jk}} = -\mathbf{d}_k o_j$$

where O_j is the input (from hidden layer unit j) feeding into the (output layer) unit k ; whereas, the delta or error (also called local gradient) \mathbf{d}_k for the output layer unit, k , now includes the partial derivative terms for each neuron (can be shown to equal the **product** of the corresponding (output) unit's error $(y_{target_k} - y_k)$ AND the derivative of the (output) unit's associated (sigmoidal) activation function, $y_k(1 - y_k)$ - see next !)

Note that this delta (error) term for each output unit also needs to be propagated back (BACK PROPAGATION) to the previous layers's units for calculating their weight updates! - see next!

Weight update Rule

Generally, change weight from any unit j to unit k by **gradient descent** - i.e. change weight by small increment in opposite direction to the gradient (or derivative of error with respect to weight) - resulting weight update rule is now called Generalized Delta Rule (GDR or Back Propagation):

$$\Delta w_{jk} = \eta \mathbf{d}_k o_j$$

where η is learning rate ($\eta > 0$)
 δ_k is the delta (local gradient) for that unit k
 o_j is the input from unit j feeding into unit k

Hence, specifically (for the 2-layer MLP shown earlier), for weight from input layer unit i to hidden layer unit j , the weight changes by:

$$\Delta w_{ij} = \eta \mathbf{d}_j x_i$$

Where x_i is (network's i -th) input feeding into the j -th hidden layer unit. And, for weight from hidden layer unit j to output layer unit k , the weight changes by: $\Delta w_{jk} = \eta \mathbf{d}_k o_j$

Where O_j is the (j -th hidden unit output) which is feeding INPUT to the k -th output unit

Computation of all Delta (error) values for MLP training

- For each output unit k , the delta (error) can be shown to equal: $d_k = (y_{target_k} - y_k) y_k (1 - y_k)$

The above equation states that the delta (or local gradient) for each output node k equals the product of:

- the corresponding error for that node, and
 - the derivative of its associated (sigmoidal) activation function
- Then, for each hidden unit j , the delta (error) can be shown to equal:

$$d_j = o_j (1 - o_j) \sum_k w_{jk} d_k$$

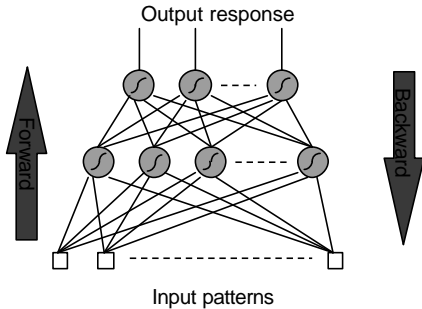
where o_j is the output of the (hidden layer) unit j , and k ranges over all units in the output layer which have connections from the hidden layer unit j

The above equation states that the delta (or local gradient) for each hidden layer node j equals the product of:

- the derivative of its associated (sigmoidal) activation function, and
- the weighted sum of the deltas d s computed for all the neurons in the next (output) layer (or next hidden layer for the case of more than 1 hidden layered MLP) that are connected to hidden neuron j

N.B. $y_k(1-y_k)$ and $o_j(1-o_j)$ are derivatives of output unit k and hidden unit j 's neuronal (sigmoidal) activation fns. respectively

Conceptually: Forward Activity - Backward Error



Forward Propagation of Activity

- Forward Direction layer by layer:
 - Inputs applied
 - Multiplied by weights
 - Summed
 - 'Squashed' by sigmoid activation function
 - Output passed to each neuron in next layer
- Repeat above until network output produced

Back-propagation of error

- Compute error (delta or local gradient) for each output unit d_k
- Layer-by-layer, compute error (delta or local gradient) for each hidden unit d_j by back-propagating errors (as shown previously)

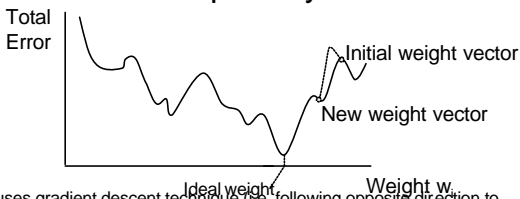
Can then update the weights Δw_{ij} using the Generalised Delta Rule (GDR), also known as the Back Propagation (BP) algorithm

'Back-prop' algorithm summary

- ◆ Initialise weights at random, choose a learning rate η
 - ◆ Until network is trained:
 - ◆ For each training example (input pattern and target outputs):
 - Do forward pass through net (with fixed weights) to produce outputs - assuming J hidden layer nodes and N inputs for a 2-layer MLP:

$$y_k = f\left(\sum_{j=0}^N w_{kj} o_j\right)$$
 where o_j is output from each hidden node j : $o_j = f\left(\sum_{i=0}^N w_{ij} x_i\right)$
 - For each output unit k , compute deltas: $d_k = (y_{target_k} - y_k) y_k (1 - y_k)$
 - For hidden units j (from last to first hidden layer, for the case of more than 1 hidden layer) compute deltas: $d_j = o_j (1 - o_j) \sum_k w_{jk} d_k$
 - For all weights, change weight by gradient descent $\Delta w_{ij} = \eta d_j y_i$
Specifically, for the 2-layer MLP, for weight from input layer unit i to hidden layer unit j , the weight changes by: $\Delta w_{ij} = \eta d_j x_i$
And, for weight from hidden layer unit j to output layer unit k , weight changes by: $\Delta w_{jk} = \eta d_k o_j$
- (N.B. Above mathematical equations NOT examinable!)

Generalised Delta Rule or BP: Graphically



- BP uses gradient descent technique (i.e. following opposite direction to that of the gradient of error with respect to weights) for changing weights
- Total error minimisation using BP is an example of a non-linear optimisation problem: many local minima exist in addition to the single desired global (error) minimum (represented by the ideal weight)
- There are many weights in general: resulting in a complex high-dimensional weight space **Next Lecture (Thurs, 23 Oct): More on BP/MLP (worked example)**
