

31YB Lecture 8:

Thurs, 23 Oct

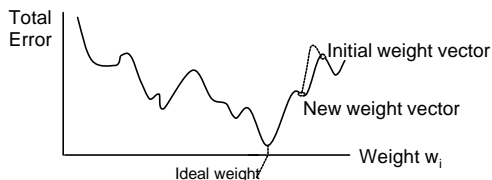
More on MLP/BP

- Review: BP algorithm
- MLP/BP: A worked example
- MLP/BP training Issues

Review: 'Back-prop' algorithm summary

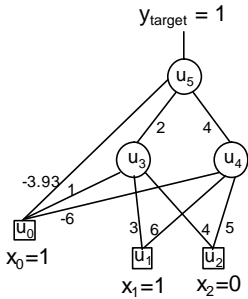
- ◆ Initialise weights at random, choose a learning rate η
- ◆ Until network is trained:
 - ◆ For each training example (inputs and target outputs):
 - Do **forward pass** through net (with fixed initial/current weights) to produce **network outputs** (e.g. for J hidden layer nodes & N inputs in 2-layer MLP):
$$y_k = f\left(\sum_{j=0}^J w_{jk} o_j\right)$$
 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**: $d_k = (y_{target} - y_k) y_k (1 - y_k)$
 - For **hidden units j** (from last to first hidden layer, for the case of more than 1 hidden layer) **compute deltas by back propagation**:
$$d_j = o_j (1 - o_j) \sum_k w_{jk} d_k$$
 - For **all weights**, **change** each weight by **gradient descent** $\Delta w_{ij} = \eta d_j y_i$
Specifically, for a 2-layer MLP, for each 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 each weight from hidden layer unit j to output layer unit k , the weight changes by: $\Delta w_{jk} = \eta d_k o_j$

Recap: Generalised Delta Rule or BP



- 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

MLP/BP: A worked example



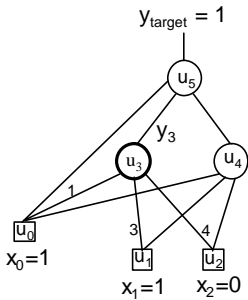
Current state:

- Weights on arrows e.g. $w_{13} = 3$, $w_{35} = 2$, $w_{24} = 5$
- Bias weights e.g. bias for u_4 is $w_{04} = -6$

Training example (e.g. for logical OR problem):

- Input pattern is $x_1=1$, $x_2=0$
- Target output is $y_{\text{target}}=1$

Worked example: Forward Pass



Output for any neuron/unit j can be calculated from:

$$a_j = \sum_i w_{ij} x_i$$

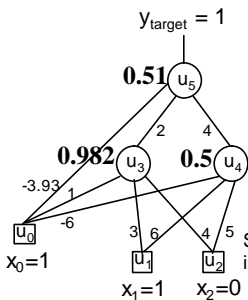
$$y_j = f(a_j) = \frac{1}{1 + e^{-a_j}}$$

e.g Calculating output for Neuron/unit 3 in hidden layer:

$$a_3 = 1*1 + 3*1 + 4*0 = 4$$

$$y_3 = f(4) = \frac{1}{1 + e^{-4}} = 0.982$$

Worked example: Forward Pass

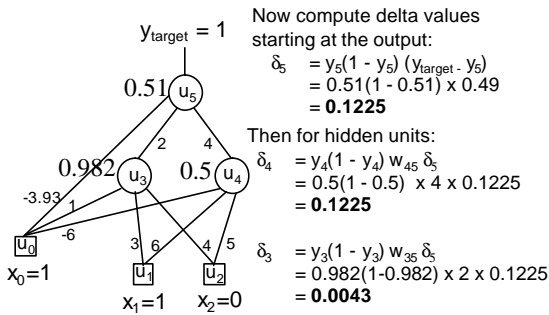


Unit activation output

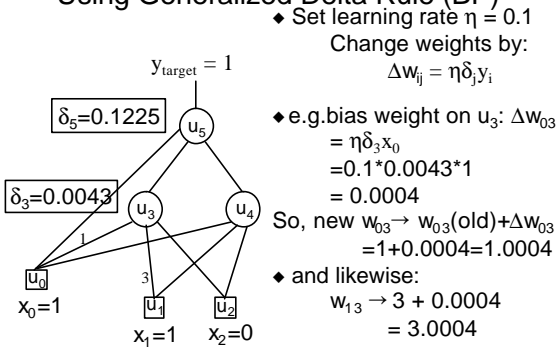
	a_j	y_j
u_3	4.00	0.982
u_4	0.00	0.500
u_5	0.04	0.510

So the error for this training example is: $(y_{\text{target}} - y_5) = (1 - 0.510) = 0.490$

Worked example: Backward Pass



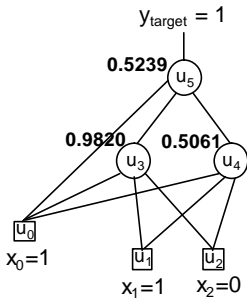
Worked example: Update Weights Using Generalized Delta Rule (BP)



Similarly for the all weights w_{ij} :

i	j	w_{ij}	δ_j	y_i	Updated w_{ij}
0	3	1	0.0043	1.0	1.0004
1	3	3	0.0043	1.0	3.0004
2	3	4	0.0043	0.0	4.0000
0	4	-6	0.1225	1.0	-5.9878
1	4	6	0.1225	1.0	6.0123
2	4	5	0.1225	0.0	5.0000
0	5	-3.92	0.1225	1.0	-3.9078
3	5	2	0.1225	0.9820	2.0120
4	5	4	0.1225	0.5	4.0061

Verification that it works



On next forward pass:

The new activations are:

$$y_3 = f(4.0008) = 0.9820$$

$$y_4 = f(0.0245) = 0.5061$$

$$y_5 = f(0.0955) = \mathbf{0.5239}$$

Thus the new error

$$(y_{\text{target}} - y_5) = (1 - 0.5239) = 0.476$$

has been reduced by **0.014**

(from **0.490** to **0.476**)

Ref: "Neural Network Learning & Expert Systems" by Stephen Gallant

Training

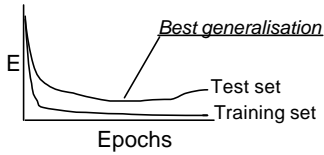
- This was a single iteration of back-prop
- Training requires many iterations with many training examples or *epochs* (one epoch is entire presentation of complete training set)
- It can be slow !
- Note that computation in MLP is local (with respect to each neuron)
- Parallel computation implementation is also possible

Training and testing data

- How many examples ?
 - The more the merrier !
- Disjoint training and testing data sets
 - learn from training data but evaluate performance (generalization ability) on unseen test data
- Aim: minimise error on *test* data

Training and testing errors

- Concept of 'Overtraining' / 'Overfitting' - point at which training error continues to fall but test set error starts to increase



- Best generalisation corresponds to minimum test error (not training error) - More on this later!

Refinements to the BP training rule

- Many refinements suggested to
 - decrease training time
 - improve performance on test data (generalisation ability)
- A 'cottage industry' in late 80's/early 90's

A popular refinement: Momentum

- Momentum: add in a 'bit' of the previous weight change

$$\Delta w_{ij} = \mathbf{hd}_j y_i + \mathbf{a} \Delta w_{ij_{previous}}$$

$$\alpha > 0.0$$

- ♦ Often decreases training time

Other refinements

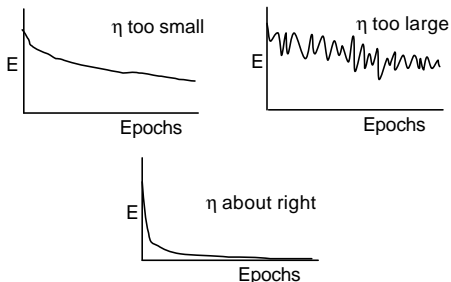
- More complex methods for choosing direction to move weights
- Line search to determine how large a step
- Start with large η , then diminish as progress
- Constructive algorithms
 - Add hidden units during training
- Pruning algorithms
 - Remove weights or hidden units during training

Some MLP/BP Issues: Choice of Experimental Parameters

- What values should be used for η and α ?
- How many layers should be used?
- How many units in each hidden layer?
 - Generally a value for η between 0.03 & 0.1, and for α between 0.7 and 0.95, found useful but..? (*answer*: choice dependent on problem, and trial & error approach usually easier in practice!)
 - Use one hidden layer, at least until you are convinced you need more than one (how does one know?) (*answer*: problem dependent, trial & error)
 - it is usually best to use a smaller number of hidden units than the number of inputs, in order to allow a compact representation to form. But there can be other issues as well, and there is no good answer to this problem, yet!

Some MLP/BP Issues: Choice of Experimental Parameters - contd.

- What value for η ?



Some MLP/BP Issues: contd..

- And, importantly, when to stop training?
 - How does one decide, either network has converged or, it will not converge (question also in DR)
 - One can stop training, when: (I) number of epochs exceeds some pre-specified value, or (II) the error is less than some pre-specified value
- ... but, which error should we try to minimize?
 - If one chooses the usual mean/sum squared error, one runs the danger of having almost correct output for almost all of the output units, but having a relatively large error for one output unit, or one pattern.
 - To avoid this, one can use different error function: however, one can also use usual error measure for training, but alter the criterion for training cessation
 - One can continue until: $E_{\maxerror} = \max_p \max_k | Y_{\text{target } k}^p - Y_k^p |$
(where p indexes the patterns, and k the output units) is less than some prespecified amount. E_{\maxerror} gives the largest error on a single output unit for any pattern
- ... and what value of error do we stop at?
 - For classification networks with discrete desired outputs, $0 < E_{\maxerror} < 0.5$, otherwise ? (*answer: problem dependent, trial & error*)

Next Lecture (9): Mon 27 Oct:

- More MLP/BP issues (finish off) and discuss MLP/ANN applications and generalization issues
- The Assignment (50% of final mark) has been handed out today (& is on course website) and is due back on 20th Nov
- First Lab is today, Thursday, 2-3PM, 4X7
