Propagate E through the Network

Feed E through the network (as in example above)

Record the target and observed values for example E

- i.e., determine weighted sum from hidden units, do sigmoid calc
- Let $t_i(E)$ be the target values for output unit i
- Let $o_i(E)$ be the observed value for output unit i

For categorisation learning tasks,

- Each $t_i(E)$ will be 0, except for a single $t_j(E)$, which will be 1
- But $o_i(E)$ will be a real valued number between 0 and 1

Also record the outputs from the hidden units

- Let $h_i(E)$ be the output from hidden unit i

Multi-layer Feed-forward ANNs

However ... there was no learning algorithm to adjust the weights of a multi-layer network - weights had to be set by hand.

How could the weights below the hidden layer be updated?

The Back-propagation Algorithm

1986: the solution to multi-layer ANN weight update rediscovered

Conceptually simple - the global error is backward propagated to network nodes, weights are modified proportional to their contribution

Most important ANN learning algorithm

Become known as *back-propagation* because the error is send back through the network to correct all weights

The Back-propagation Algorithm

Like the Perceptron - calculation of error is based on difference between target and actual output:

$$E = \frac{1}{2}\sum_j (t_j - o_j)^2$$

However in BP it is the rate of change of the error which is the important feedback through the network

generalized delta rule $\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}}$

Relies on the sigmoid activation function

The Back-propagation Algorithm

The Bp algorithm performs a *gradient descent* in weights space toward a minimum level of error using a fixed *step size* or learning rate η

The gradient is given by :

= rate at which error changes as weights change $\frac{\delta E}{\delta w_{ij}}$

Backpropagation Learning Algorithm

Same task as in perceptrons

- Learn a multi-layer ANN to correctly categorise unseen examples
- We'll concentrate on ANNs with one hidden layer

Overview of the routine

- Fix architecture and sigmoid units within architecture
  - i.e., number of units in hidden layer; the way the input units represent example; the way the output units categorises examples
- Randomly assign weights to the the whole network
  - Use small values (between –0.5 and 0.5)
- Use each example in the set to retrain the weights
- Have multiple epochs (iterations through training set)
  - Until some termination condition is met (not necessarily 100% acc)

Weight Training

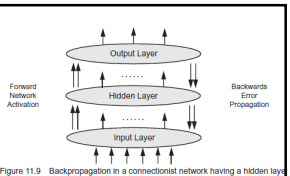


Figure 11.9 Backpropagation in a connectionist network having a hidden layer

The Back propagation algorithm is

1. start at the output layer and
2. propagate error backwards through the hidden layer

- Use notation w_{ij} to specify: Weight between unit i and unit j
- Look at the calculation with respect to example E
- Calculate a value Δ_{ij} for each w_{ij} And add Δ_{ij} on to w_{ij}
- Do this by calculating **error terms** for each unit
- The error term for output units is found And then this information is used to calculate the error terms for the hidden units

So, the error is propagated back through the ANN

Error terms for each unit

The Error Term for output unit k is calculated as:

$$\delta_{O_k} = o_k(E)(1 - o_k(E))(t_k(E) - o_k(E))$$

The Error Term for hidden unit k is:

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in outputs} w_{ki}\delta_{O_i}$$

i.e. For hidden unit h, add together all the errors for the output units, multiplied by the appropriate weight. Then multiply their sum by $h_k(E)(1 - h_k(E))$

Final Calculations

Choose a learning rate, η (= 0.1 say)

For each weight w_{ij}

Between input unit i and hidden unit j

Calculate: $\Delta w_{ij} = \eta \delta_{Hi} x_i$

Where x_i is the input to the system to input unit i for E

For each weight w_{ij} between hidden unit i and output unit j

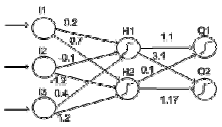
Calculate: $\Delta w_{ij} = \eta \delta_{Oi} h_i(E)$

Where $h_i(E)$ is the output from hidden unit i for E

Finally, add on each Δw_{ij} on to w_{ij}

Worked Backpropagation Example

Start with the previous ANN



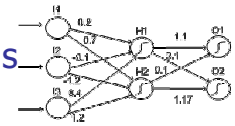
We will retrain the weights

In the of example E = (10,30,20)

Assume that E should have been categorised as O1 (not O2 as the calculated result)

Will use a learning rate of η = 0.1

Previous Calculations



Need the calculations from when we propagated E through the ANN:

Input units		Hidden units			Output units		
Unit	Output	Unit	Weighted Sum Input	Output	Unit	Weighted Sum Input	Output
I1	10	H1	7	0.999	O1	1.0996	0.750
I2	30	H2	-5	0.0067	O2	3.1047	0.957
I3	20						

$o_1(E) = 0.750$ and $o_2(E) = 0.957$

$t_1(E) = 1$ and $t_2(E) = 0$ [Assumption says it should be O1]

Error Values for Output Units

$t_1(E) = 1$ and $t_2(E) = 0$ $o_1(E) = 0.750$ and $o_2(E) = 0.957$

So: $\delta_{O_k} = o_k(E)(1 - o_k(E))(t_k(E) - o_k(E))$

$\delta_{O1} = o_1(E)(1 - o_1(E))(t_1(E) - o_1(E)) = 0.750(1-0.750)(1-0.750) = 0.0469$

$\delta_{O2} = o_2(E)(1 - o_2(E))(t_2(E) - o_2(E)) = 0.957(1-0.957)(0-0.957) = -0.0394$

Error Values for Hidden Units

Now: $\delta_{O1} = 0.0469$ and $\delta_{O2} = -0.0394$

$h_1(E) = 0.999$ and $h_2(E) = 0.0067$ (output of hidden from the table)

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in outputs} w_{ki}\delta_{O_i}$$

So, for H1, we add together:

$(w_{11}*\delta_{01}) + (w_{12}*\delta_{O2}) = (1.1*0.0469)+(3.1*-0.0394) = -0.0706$

And multiply by: $h_1(E)(1-h_1(E))$ to give us:

$\delta_{H1} = -0.0706 * (0.999 * (1-0.999)) = 0.0000705$

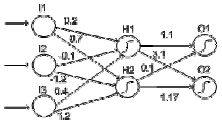
For H2, we add together:

$(w_{21}*\delta_{01}) + (w_{22}*\delta_{O2}) = (0.1*0.0469)+(1.17*-0.0394) = -0.0414$

And multiply by: $h_2(E)(1-h_2(E))$ to give us:

$\delta_{H2} = -0.0414 * (0.067 * (1-0.067)) = -0.00259$

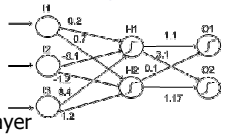
Calculation of Weight Changes



For weights between the input and hidden layer each w_{ij} And add Δ_{ij} on to w_{ij}

Input unit	Hidden unit	η	δ_H	x_i	$\Delta = \eta*\delta_H*x_i$	Old weight	New weight
I1	H1	0.1	-0.0000705	10	-0.0000705	0.2	0.1999295
I1	H2	0.1	-0.00259	10	-0.00259	0.7	0.69741
I2	H1	0.1	-0.0000705	30	-0.0002115	-0.1	-0.1002115
I2	H2	0.1	-0.00259	30	-0.00777	-1.2	-1.20777
I3	H1	0.1	-0.0000705	20	-0.000141	0.4	0.39999
I3	H2	0.1	-0.00259	20	-0.00518	1.2	1.1948

Calculation of Weight Changes



For weights between hidden and output layer

Hidden unit	Output unit	η	δ_O	$h_i(E)$	$\Delta = \eta*\delta_O*h_i(E)$	Old weight	New weight
H1	O1	0.1	0.0469	0.999	0.000469	1.1	1.100469
H1	O2	0.1	-0.0394	0.999	-0.00394	3.1	3.0961
H2	O1	0.1	0.0469	0.0067	0.00314	0.1	0.10314
H2	O2	0.1	-0.0394	0.0067	-0.0000264	1.17	1.16998

Weight changes are not very large

- Small differences in weights can make big differences in calculations
- But it might be a good idea to increase η

Calculation of Network Error

Could calculate Network error as

- Proportion of mis-categorised examples

But there are multiple output units, with numerical output

- So we use a more sophisticated measure:

$$\frac{1}{2} \sum_{E \in examples} \left(\sum_{k \in outputs} (t_k(E) - o_k(E))^2 \right)$$

Not as complicated as it looks

- Square the difference between target and observed
  - Squaring ensures we get a positive number
- Add up all the squared differences
  - For every output unit and every example in training set

Backpropagation Training Algorithm

The algorithm is composed of two parts that get repeated over and over a number of ***epochs***:

I. The ***feedforward*** : the activation values of the hidden and then output units are computed.

II. The ***backpropagation*** : the weights of the network are updated--starting with the hidden to output weights and followed by the input to hidden weights--with respect to the sum of squares error, the ***Delta Rule***.

Backpropagation Training Algorithm

Until all training examples produce the correct value (within ε), or mean squared error stops to decrease, or other termination criteria:

Begin epoch
For each training example, E, do:
Calculate network output for E's input values
Compute error between current output and correct output or E
Update weights by backpropagating error and using learning rule
End epoch

Backpropagation: The Momentum

Backpropagation has the disadvantage of being too slow if η, the learning rate, is small and it can oscillate too widely if η is large.

To solve this problem, we can add a ***momentum*** to give each connection some inertia, forcing it to change in the direction of the downhill "force".

Old Delta Rule: $\Delta w_{ij} = \eta \delta_{Hi} x_i$, $\Delta w_{ij} = \eta \delta_{Oi} h_i (E)$

New Delta Rule: $\Delta w_{ij}(t+1) = \eta \delta_{Hi} x_i + \alpha \Delta w_{ij}(t)$

And $\Delta w_{ij}(t+1) = \eta \delta_{Oi} h_i (E) + \alpha \Delta w_{ij}(t)$

where i,j are any input and hidden, or, hidden and output units;
t is a time step or epoch;
and α is the momentum parameter which regulates the amount of inertia of the weights.

Backpropagation Training Algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- In practice, it does converge to low error for many large networks on real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.
- To avoid local-minima problems, run several trials starting with different random weights (***random restarts***).

Pros of Backpropagation

- Proven training method for multi-layer nets
- Able to learn any arbitrary function (XOR)
- Most useful for non-linear mappings
- Works well with noisy data
- Generalizes well given sufficient examples
- Rapid recognition speed
- Has inspired many new learning algorithms

Cons of Backpropagation

- May fall in Local minimum - but not generally a concern
- Seems complicated biologically implausible
- Space and time complexity high:
  lengthy training times $O(W^3)$
- May still be considered as a black box: *how it's making decisions?*
- Best suited for supervised learning only
- Works poorly on dense data with few input variables

Hidden Unit Representations

Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.

On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..

However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.

ANN Representing function Thms

Boolean functions: Any Boolean function can be represented by a two-layer network with sufficient hidden units.

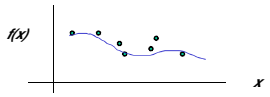
Continuous functions: Any bounded continuous function can be approximated with arbitrarily small error by a two-layer network.

Arbitrary function: Any function can be approximated to arbitrary accuracy by a three-layer network.

Generalization

The objective of learning is to achieve good *generalization* to new cases, otherwise just use a look-up table.

Generalization can be defined as a mathematical *interpolation* or *regression* over a set of training points:



Generalization

Weight Decay: an automated method of effective weight control

Adjust the bp error function to penalize the growth of unnecessary weights:

$$E = \frac{1}{2}\sum_j (t_j - o_j)^2 + \frac{\lambda}{2}\sum_i w_{ij}^2 \quad \Rightarrow \quad \Delta w_{ij} = \Delta w_{ij} - \lambda w_{ij}$$

where: λ = weight -cost parameter

w_{ij} is decayed by an amount proportional to its magnitude;

Generalization

A Probabilistic Guarantee

N = # hidden nodes m = # training cases

W = # weights ε = error tolerance (< 1/8)

Network will generalize with 95% confidence if:

1. Error on training set < $\varepsilon/2$
2. $m > O(\frac{W}{\varepsilon}\log_2\frac{N}{\varepsilon}) \approx m > \frac{W}{\varepsilon}$

Over-Training

Is the equivalent of over-fitting a set of data points to a curve which is too complex

Occam's Razor (1300s) : *"plurality should not be assumed without necessity"*

The simplest model which explains the majority of the data is usually the best

Preventing Over-training

- Use a separate *test* or *tuning set* of examples
- Monitor error on the test set as network trains
- Stop network training just prior to over-fit error occurring - *early stopping* or *tuning*
- Number of effective weights is reduced
- Most new systems have automated early stopping methods