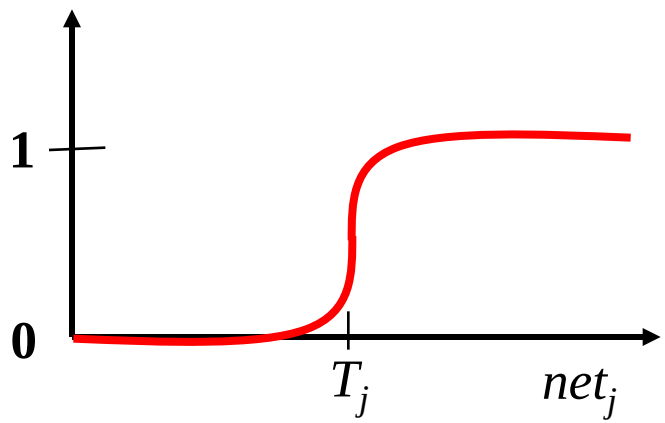
Differentiable Output Function

- Need non-linear output function to move beyond linear functions.
  - A multi-layer linear network is still linear.
- Standard solution is to use the non-linear, differentiable sigmoidal "logistic" function:

$$o_j = \frac{1}{1+e^{-(net_j - T_j)}}$$



Can also use tanh or Gaussian output function

Gradient Descent

- Define objective to minimize error:

$$E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

where D is the set of training examples, K is the set of output units, t_{kd} and o_{kd} are, respectively, the teacher and current output for unit k for example d.

- The derivative of a sigmoid unit with respect to net input is:

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

- Learning rule to change weights to minimize error is:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

Backpropagation Learning Rule

- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

where η is a constant called the learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j

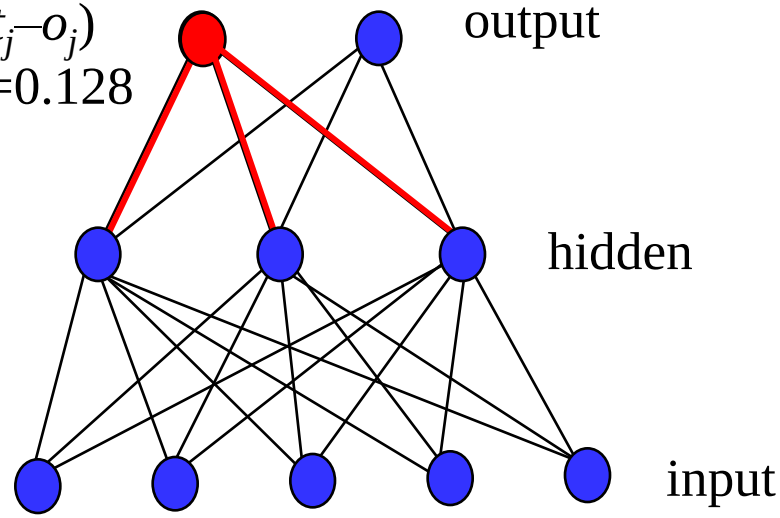
Error Backpropagation

- First calculate error of output units and use this to change the top layer of weights.

Current output: o_j=0.2
Correct output: t_j=1.0
Error $\delta_j = o_j(1-o_j)(t_j-o_j)$
0.2(1–0.2)(1–0.2)=0.128

Update weights into j

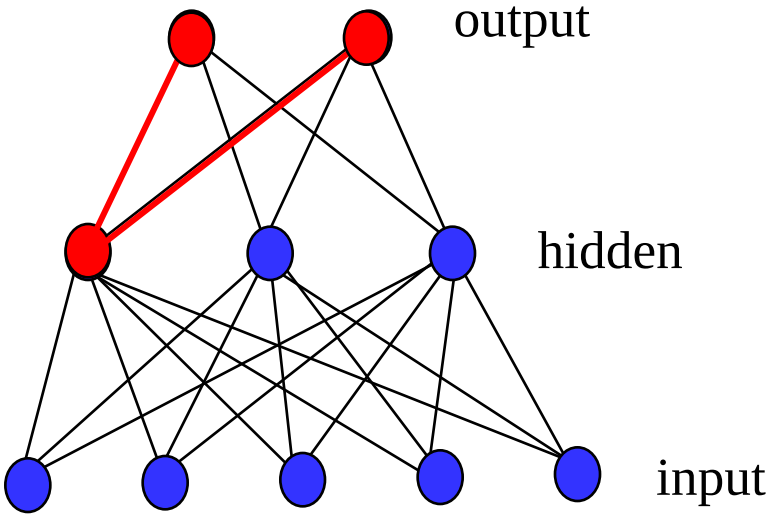
$$\Delta w_{ji} = \eta \delta_j o_i$$



Error Backpropagation

- Next calculate error for hidden units based on errors on the output units it feeds into.

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$



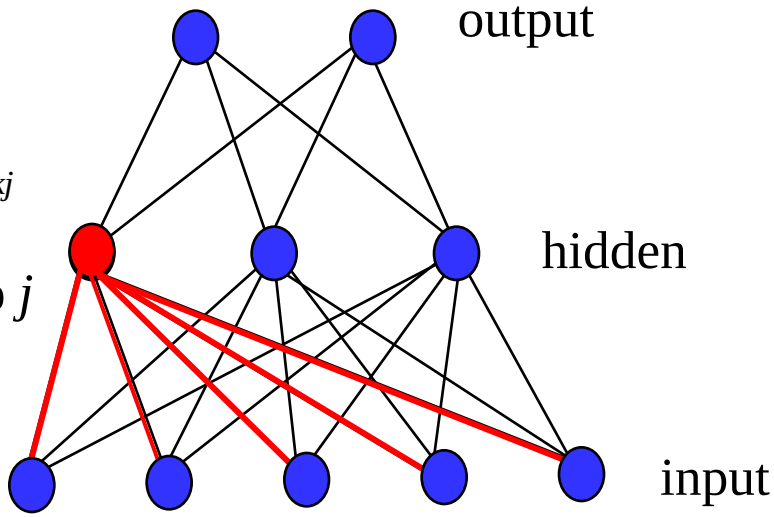
Error Backpropagation

- Finally update bottom layer of weights based on errors calculated for hidden units.

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$

Update weights into j

$$\Delta w_{ji} = \eta \delta_j o_i$$



Backpropagation Training Algorithm

Create the 3-layer network with *H* hidden units with full connectivity between layers. Set weights to small random real values.

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

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

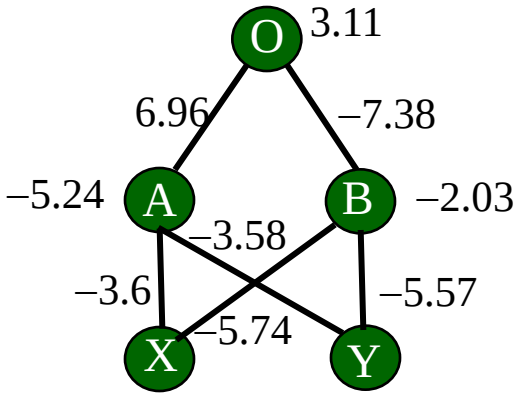
Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, 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*).
  - Take results of trial with lowest training set error.
  - Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).

Representational Power

- **Boolean functions**: Any boolean function can be represented by a two-layer network with sufficient hidden units.
- **Arbitrary function**: Any function can be approximated to arbitrary accuracy by a three-layer network.

Sample Learned XOR Network



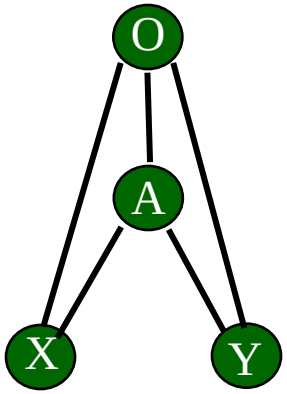
Hidden Unit A represents: $\neg(X \wedge Y)$

Hidden Unit B represents: $\neg(X \vee Y)$

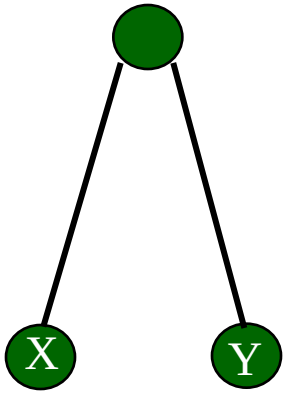
Output O represents: $A \wedge \neg B = \neg(X \wedge Y) \wedge (X \vee Y)$

$= X \oplus Y$

What about now?



And now?



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. (KBANN: Towell & Shavlik)

Parameter Tuning

1. Initial weights?
2. Learning rate?
3. Momentum?
4. Generalization?
5. # Hidden nodes?
6. Recurrent links?