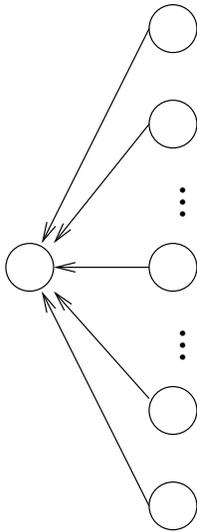


Neural Networks

1. Decision trees are nice good simple learners, but they lack those fuzzy properties that we want to have. Where are those probabilities?
2. Another important category of learning techniques is neural networks. We'll look at are particular example of those called feed-forward back propagation of error. They:
 - (a) Are supervised
 - (b) Use real-valued inputs
 - (c) Use real-valued outputs
3. The I/O is:
 - (a) Input: a vector of numbers (often normalized)
 - (b) Putput: a vector of numbers.
4. Background
 - (a) Neural Network (NN)is an unfortunate name. They are really a distributed statistic function approximator with gradient descent learning.
 - (b) NN gets its name because the people who invented them were using inspiration from the brain in order to design them.
 - (c) It wasn't until the 90s that we realized the full extent of its capabilities and limitations.
 - (d) Because of their inspiration, it is often helpful to take a moment and describe how the brain works
 - i. The brain is made up of neurons
 - ii. Neurons consist of: A cell body, An axon(s), Dendrites
 - iii. Picture should be here.
 - iv. Sum up inputs until it fires.
 - (e) Many kinds o neural networks. We'll look at the Perceptron first.
5. Perceptrons
 - (a) A perceptron is like a single neuron simulator.



- (b) Activation function
- (c) $\sum_i w_{ij} I_i$
- (d) This puts a hyperplane through the input space, classifying some inputs as 1, and the rest as zero.
- (e) Picture here
- (f) Feed some input to the network. Compare that input to what we know the value should have been. change the weights in such a way that the next time the network will output the correct answer.
- (g) $Err = T_j - O_j$
- (h) $w_{ij} = w_{ij} + \alpha \times I_i \times Err_j$
- (i) For any function that is linearly separable, this will learn the correct function. Most things are NOT linear separable uh oh. see xor.
- (j) With and, or, not we could chain them together to build a circuit that could calculate anything. But how would we learn?

6. Backpropagation w/ Gradient Descent

- (a) We need to create a multi-layer network, and come up with a learning rule for it
- (b) 2 layers will be enough (although the proof is complicated)
- (c) Basic Plan
 - i. treat the network as a function
 - ii. Treat the error of the network as a function
 - iii. Make sure the error function is always positive
 - iv. find the slope of the error function
 - v. Change the weights of the network so that the value of the error is reduced

$$O_k = f\left(\sum_j w_{jk} f\left(\sum_i w_{ij} I_i\right)\right)$$

- (d) f is some step-like function. must be continuously differentiable.
- (e) picture here
- (f) $f(x) = \frac{1}{1+e^{-x}}$, $f'(x) = f(x)(1 - f(x))$
- (g) $E = (T - O)^2$ Why squared?

- (h) we want to change the weights to minimize the errors: take the partial derivative w/r/t the weight we want to change. Assume we're trying to change the value of some weight w_{bc} in the outer layer.

Let:

$$\begin{aligned}
 N_k &= \sum_j w_{jk} f\left(\sum_i w_{ij} I_i\right) \\
 M_j &= \sum_i w_{ij} I_i \\
 E &= \frac{1}{2} \sum_k (T_k - O_k)^2 \\
 &= \frac{1}{2} \sum_k (T_k - f(\sum_j w_{jk} f(\sum_i w_{ij} I_i)))^2 \\
 \frac{\partial E}{\partial w_{bc}} &= \frac{\partial}{\partial w_{bc}} \frac{1}{2} \sum_k (T_k - f(N_k))^2
 \end{aligned}$$

Since the partial is w/r/t w_{bc} , we ignore all terms in the sum except those with w_{bc}

$$\begin{aligned}
 &= \frac{\partial}{\partial w_{bc}} \frac{1}{2} (T_c - f(N_c))^2 \\
 &= (T_c - f(N_c)) \frac{\partial}{\partial w_{bc}} (T_c - f(N_c)) \\
 &= -(T_c - f(N_c)) \frac{\partial}{\partial w_{bc}} f(N_c) \\
 &= -(T_c - f(N_c)) f'(N_c) \frac{\partial}{\partial w_{bc}} \sum_j w_{jc} f(M_j)
 \end{aligned}$$

again, we ignore all terms in the sum except those with w_{bc}

$$\begin{aligned}
 &= -(T_c - f(N_c)) f'(N_c) \frac{\partial}{\partial w_{bc}} w_{bc} f(M_b) \\
 &= -(T_c - f(N_c)) f'(N_c) f(M_b)
 \end{aligned}$$

- (i) Since we want to change the weight in the direction *opposite* the gradient, our final update rule for outer weight w_{bc} is:

$$w_{bc} \leftarrow w_{bc} + \alpha (T_c - f(N_c)) f'(N_c) f(M_b)$$

(j) Now, assume we want to change a weight w_{ab} in the hidden layer.

$$\begin{aligned}
\frac{\partial E}{\partial w_{ab}} &= \frac{\partial}{\partial w_{ab}} \frac{1}{2} \sum_k (T_k - f(N_k))^2 \\
&= - \sum_k (T_k - f(N_k)) f'(N_k) \frac{\partial}{\partial w_{ab}} N_k \\
&= - \sum_k (T_k - f(N_k)) f'(N_k) \frac{\partial}{\partial w_{ab}} \sum_j w_{jk} f\left(\sum_i w_{ij} I_i\right) \\
&\text{ignoring terms in sum that don't contain } w_{ab} \\
&= - \sum_k (T_k - f(N_k)) f'(N_k) \frac{\partial}{\partial w_{ab}} w_{bk} f\left(\sum_i w_{ib} I_i\right) \\
&= - \sum_k (T_k - f(N_k)) f'(N_k) w_{bk} f'(M_b) \frac{\partial}{\partial w_{ab}} \sum_i w_{ib} I_i \\
&\text{ignoring terms in sum that don't contain } w_{ab} \\
&= - \sum_k (T_k - f(N_k)) f'(N_k) w_{bk} f'(M_b) \frac{\partial}{\partial w_{ab}} w_{ab} I_i \\
&= - \sum_k (T_k - f(N_k)) f'(N_k) w_{bk} f'(M_b) I_a
\end{aligned}$$

(k) Since we want to change the weight in the direction *opposite* the gradient, our final update rule for hidden weight w_{ab} is:

$$w_{ab} \leftarrow w_{ab} + \alpha \sum_k (T_k - f(N_k)) f'(N_k) w_{bk} f'(M_b) I_a$$