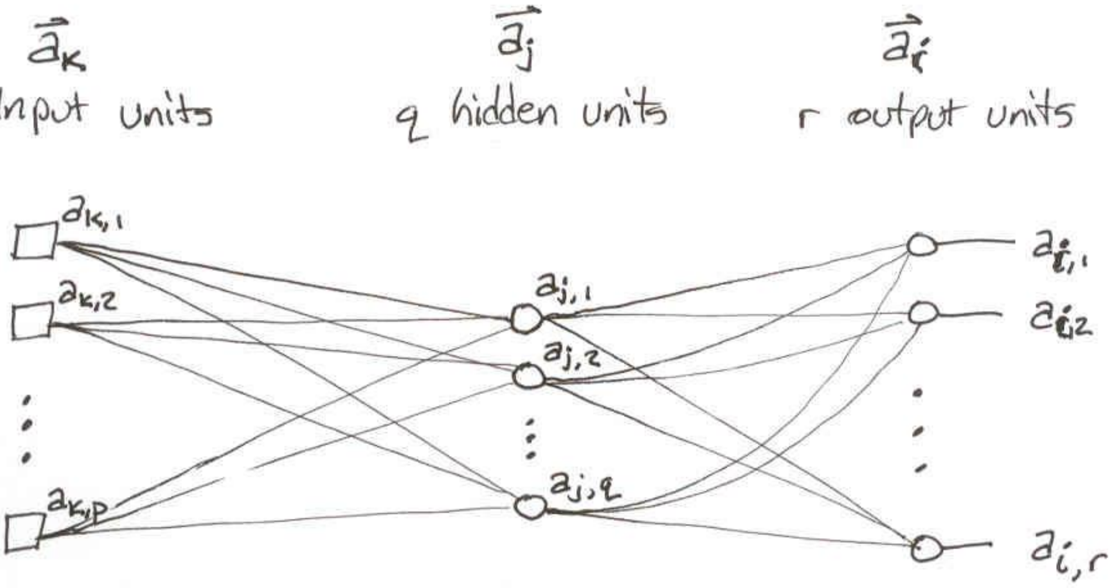
Neural Network notation and backpropagation

The notation in our text kind of stinks, so here's something better but still based on their notation.



the "a" values refer to the output of the unit. (Except, of course, the input units simply pass the value, so the $\vec{a}_k$ are input and output.)

The text uses $W_{k,j}$ to collectively refer to all (or any) weight from the input layer to the hidden layer. We will give each weight a unique notation:

$W_{k,m}^{j,n} \equiv$ the weight from unit m in layer k to unit n in layer j. Here $m \in \{1,2,\dots p\}$ and $n \in \{1,2,\dots q\}$

Similarly, we have $W_{j,m}^{i,n}$ to represent the weight from unit $m \in \{1,2,\dots q\}$ in layer j to unit $n \in \{1,2,\dots r\}$ in layer i.

We can refer to all the weights for the inputs of a particular unit in vector form. For example,

$$\overrightarrow{W_k^{j,n}} = \left(W_{k,1}^{j,n} , W_{k,2}^{j,n} , \dots W_{k,p}^{j,n} \right)$$

We will also use "in" variables to refer to the activation level of a unit. For example, for hidden unit n (in layer j) we have:

$$in_{j,n} = \sum_{m=1}^{p} W_{k,m}^{j,n} a_{k,m} = \overrightarrow{W_k^{j,n}} \cdot \vec{a}_k$$

The output of that unit is then

$$a_{j,n} = g(in_{j,n}) \qquad \text{where} \qquad g(x) = \frac{1}{1+e^{-x}}$$

We are now ready to do backpropagation. A single training example is of the form $(\vec{x}, \vec{y})$ where $\vec{x}$ is a p-dimensional input vector and $\vec{y}$ is a r-dimensional output vector.

First we compute the actual output of the neural net for the given input:

- Let $\vec{a}_k = \vec{x}$
- Compute the activation level of each hidden unit and then its output. For $n = 1 \dots q$

  $$in_{j,n} = \overrightarrow{W_k^{j,n}} \cdot \vec{a}_k$$

  $$a_{j,n} = g(in_{j,n})$$

- Compute the activation level of each output unit and then its output: For $n = 1 \ldots r$

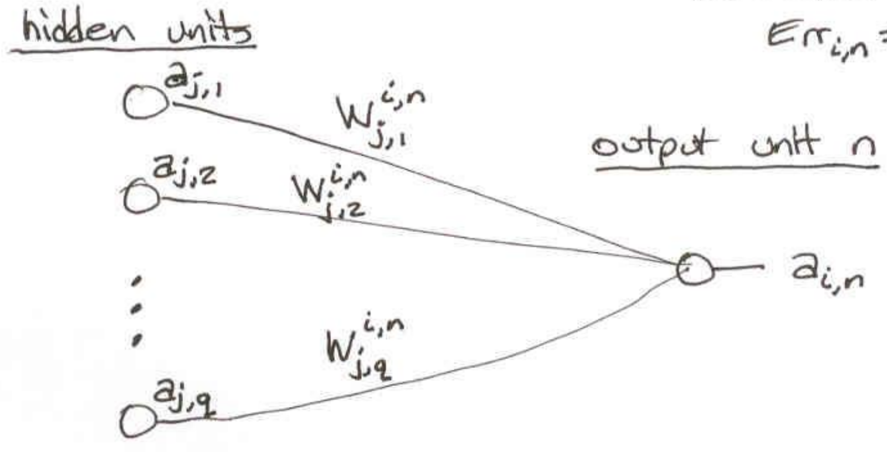
$$in_{i,n} = \overrightarrow{W_j^{i,n}} \cdot \overrightarrow{a_j}$$

$$a_{i,n} = g(in_{i,n})$$

The actual output of an output unit $a_{i,n}$ will in general be different than the desired output y_n; so we will do backpropagation to change the weights.

The difference is the error:
$Err_{i,n} = y_n - a_{i,n}$

First, consider an output unit



From $_\wedge$the idea of gradient descent, we can compute

$$\Delta_{i,n} = Err_{i,n} \times g'(in_{i,n})$$

And then update each weight as follows:

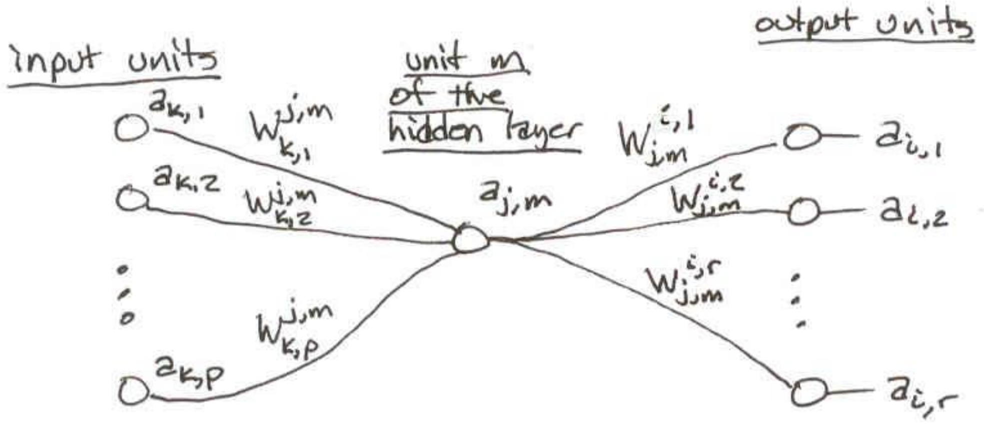
$$W_{j,m}^{i,n} \leftarrow W_{j,m}^{i,n} + \alpha \times a_{j,m} \times \Delta_{i,n}$$

Or, in vector form

$$\overrightarrow{W_j^{i,n}} \leftarrow \overrightarrow{W_j^{i,n}} + \alpha \times \overrightarrow{a_j} \times \Delta_{i,n}$$

This adjusts the weights from each hidden unit (layer j) to unit n of the output layer (layer i) — these are the $\overrightarrow{W_j^{i,n}}$ — in order to reduce the error of that output unit.

Next, consider a hidden unit:



The problem is that we don't know what the ~~error~~ output of the hidden unit should be, so we don't know what the error is.

The idea of backpropagation is that each hidden unit is responsible for some part of the error for all of the output units, in proportion to the weight from the hidden unit to the output unit $W_{j,m}^{i,n}$

So, we compute

$$\Delta_{j,m} = g'(in_{j,m}) \underbrace{\sum_n W_{j,m}^{i,n} \Delta_{i,n}}$$

This is adding up ~~this~~ the hidden unit's part of the error for all output units

Then we can update the weights in the same way:

$$W_{k,l}^{j,m} \leftarrow W_{k,l}^{j,m} + \alpha * a_{k,l} * \Delta_{j,m}$$

(this is the weight from unit l of the input layer k~~[illegible]~~ to unit m of the hidden layer j.) In vector form:

$$\overrightarrow{W}_k^{j,m} \leftarrow \overrightarrow{W}_k^{j,m} + \alpha * \overrightarrow{a}_k * \Delta_{j,m}$$

The context of all the above is in learning a neural network:

- Initialize all weights to random values (typically in $[-0.5, 0.5]$
- Repeat:

This is an epoch $\Big[$ – For each training example $(\vec{x}, \vec{y})$, update weights as above

Until: Error on an independent "testing data set" reaches a minimum.

(or other stopping criterion is met)

Usually, you will start α at a small value such as 0.1 ~~[illegible]~~ or 0.05 and decrease it slightly perhaps every 100, 500, or 1000 epochs.