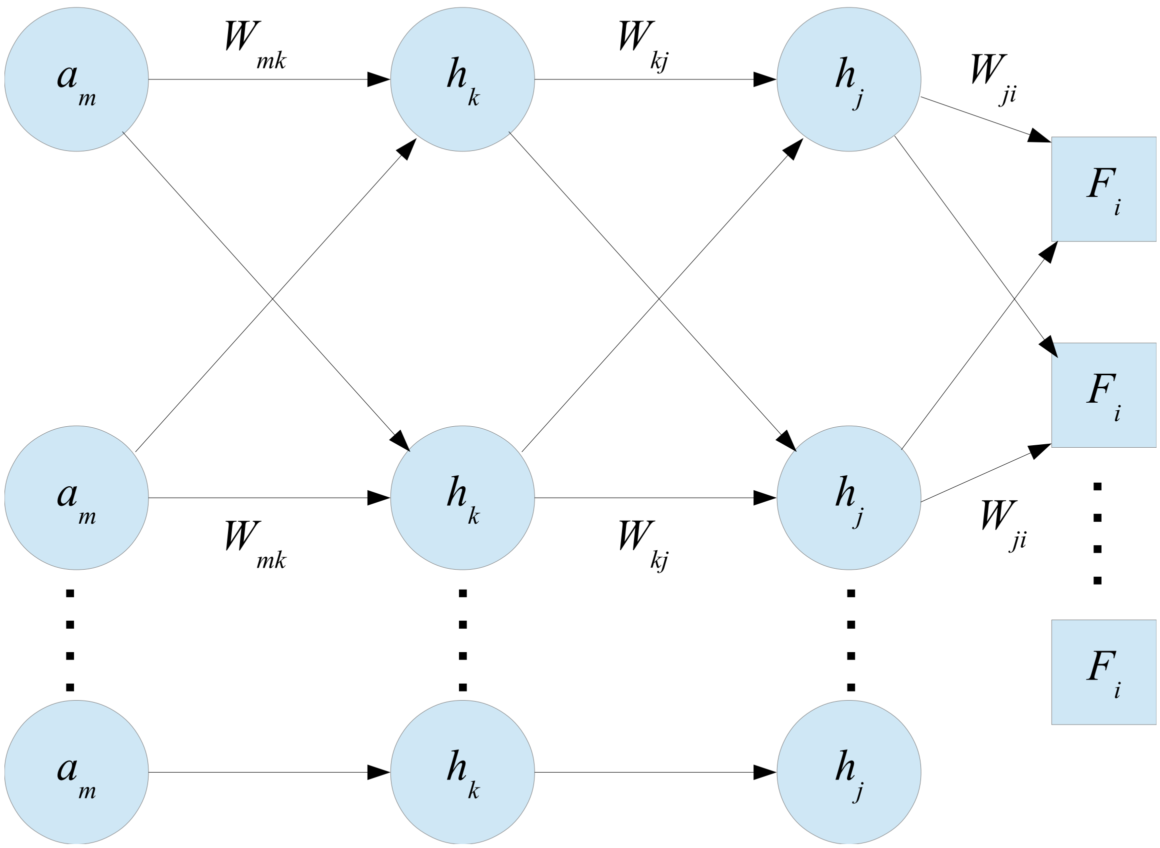
Consider a simply connected network that has an *input activation layer* (a_m), two *hidden activation layers* (h_k, h_j) and an *output activation layer* (F_i). This configuration thus has three *connectivity layers*. Since there are **three connectivity layers**, this configuration is referred to as a **three-layer network**.



In general, you can see that for some set of activations on the right-side there are a collection of weights that connect them to another set of activations on the left-side. For each set of "outputs" (*aka*, right-side) activations we therefore have

$$h_k = f\left(\sum_m a_m w_{mk}\right),\ h_j = f\left(\sum_k h_k w_{kj}\right) \text{ and } F_i = f\left(\sum_j h_j w_{ji}\right)$$

where $f(x)$ is the threshold function (typically a sigmoid $f(x) = \frac{1}{1+e^{-x}}$ or hyperbolic tangent or arctangent). It should be clear that we can generalize the connectivity between any two activation layers since each consists of an input (left-hand side) and an output (right-hand side) connected by a collection of weights. In general we can write

$$Output_i = f\left(\sum_j Input_j Weight_{ji}\right).$$

Where the index order on the weights represent the input index (j) and the output index (i), respectively. Notice that there is really nothing special about the input and output activation layers in terms of the math involved in the connectivity layers. In principle, if the correct values for the weights can be found, then for any input there should be a predictable output.

You should notice that you can fully specify the network connectivity model with only a few parameters

- Number of Input Nodes
- Number of Hidden Layers
- Number of Nodes in Each Hidden Layer
- Number of Output Nodes.

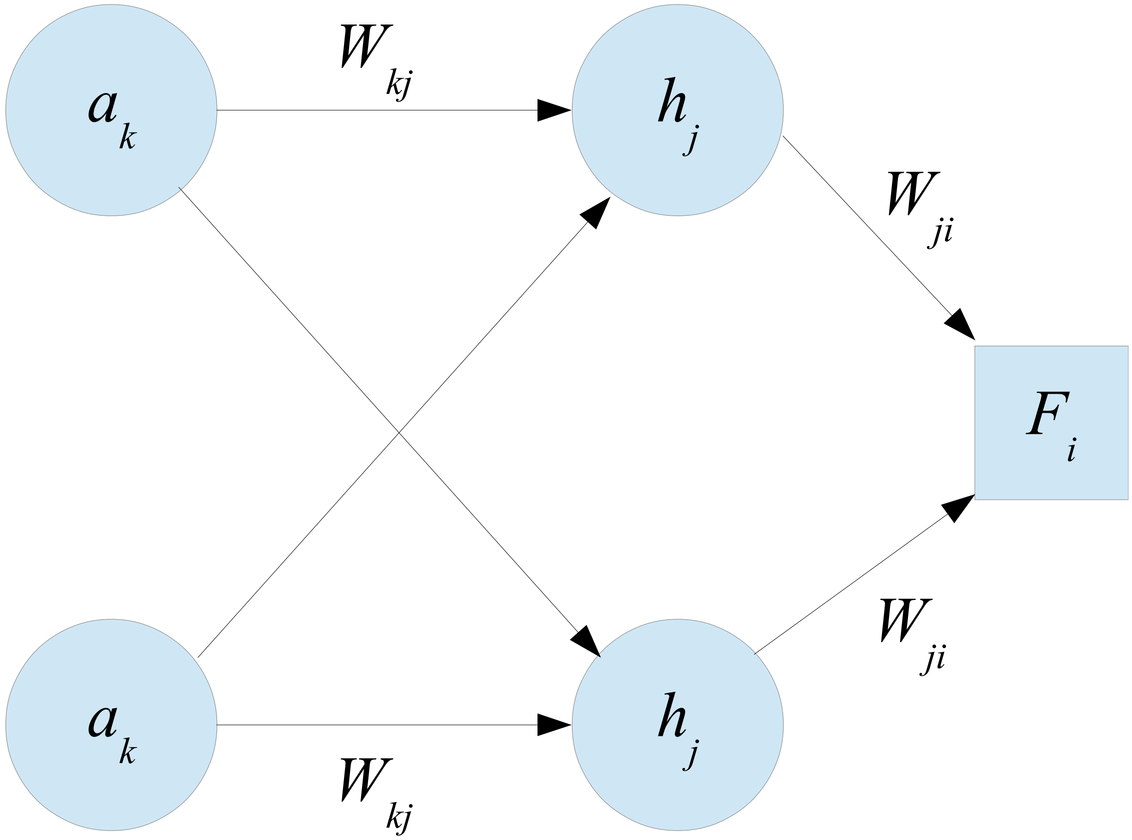
In Java your constructor for the network would therefore need only three parameters:

```
int inputNodes;
int[] hiddenLayerNodes;
int outputNodes;
```

where the number of hidden layers can be extracted from `hiddenLayerNodes.length`. From these parameters you can determine the dimensions of the array of weights for a fully connected feed-forward network.

To construct the network you should start by allocating the space needed to hold the weights based on the configuration parameters. You will need to be able randomize the weights for training purposes. Note that this action is NOT part of the construction of the network since once the optimal weights are known you would set them to those values. You will need to be able to perform the general activation-weight multiplication of the form $Output_i = f\left(\sum_j Input_j Weight_{ji}\right)$ and have a high level routine that manages what constitutes inputs and outputs for this action. You will need the ability to dump the weights to the terminal (for debugging purposes) as well as a way of stating the structure of the network configuration (again for debugging). Lastly you will need a way of minimizing the error function based on the training set data, but that is the topic of the "Minimizing the Error Function" document.

The problem is now in figuring out what the value of all the weights is to be to make the network do something useful. To find a set of acceptable weights (and there can be more than one set that will work well enough for a given set of inputs), we are going to have to train our network. In order to construct a functioning example, let us build a simple XOR network. This simple network has a two-node input activation layer, a two-node hidden activation layer and a single-node output layer.



We have the following relationships within the network

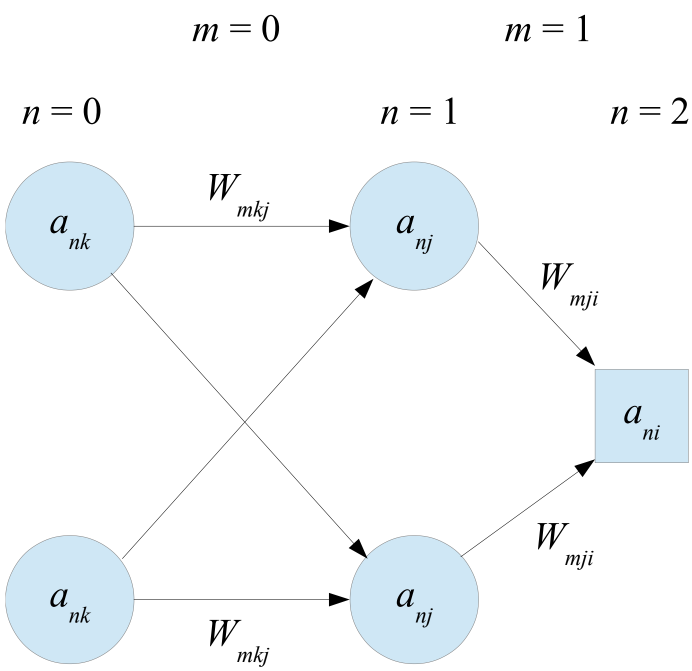
$$h_j = f\left(\sum_k a_k w_{kj}\right),$$

$$F_i = f\left(\sum_j h_j w_{ji}\right),$$

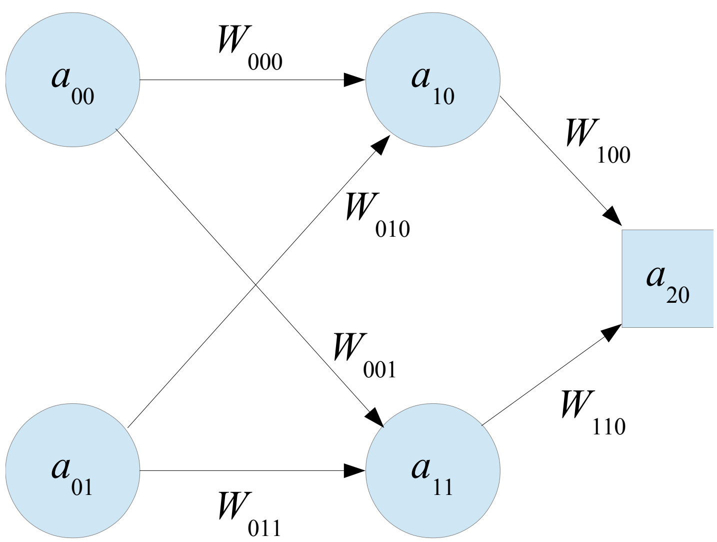
$$E_t = \frac{1}{2}\sum_i \left(T_{it} - F_{it}\right)^2$$ (more on how to interpret this expression later)

where E_t is the *error function* for training activation set t which is a function of all the weights in the network and the T_i values are the target outputs expected from a known set of input values (these are our training data).

Let us fully expand out our diagram with all the proper index values. The problem is that we are using k, j and i to help keep track of the activation layer and when we put in the numbers we won't know where we are. We need another index to keep track of the activation and connectivity layers, so we are going to redraw our diagram to use a more general notation where all activations are simply labeled as a_{ni} with an activation layer index n and a node index i, j, or k, and all the weights are W_{mkj} and W_{mji}, where m is the connectivity layer, k is an input node index, j is a hidden node index and i is an output node index.



Now we can put in the index values for this simple example.



For each of the right-side activations we can now expand our example.

$h_j = a_{1j} = f\left(\sum_{k=0}^{1} a_{0k} w_{0kj}\right)$, so

$$h_0 = a_{10} = f\left(\sum_{k=0}^{1} a_{0k}\, w_{0k0}\right) = f\left(a_{00}\, w_{000} + a_{01}\, w_{010}\right)$$

$$h_1 = a_{11} = f\left(\sum_{k=0}^{1} a_{0k}\, w_{0k1}\right) = f\left(a_{00}\, w_{001} + a_{01}\, w_{011}\right)$$

and

$$F_i = a_{2i} = f\left(\sum_{j=0}^{1} a_{1j}\, w_{1ji}\right) \text{, so}$$

$$F_0 = a_{20} = f\left(\sum_{j=0}^{1} a_{1j}\, w_{1j0}\right) = f\left(a_{10}\, w_{100} + a_{11}\, w_{111}\right)$$

But what about the error function? There are two ways to view the error function. First we can consider the full error for all the training data. This minimization is mathematically rigorous in that you want to minimize the error with respect to the *N*-dimensional weight space for all training sets simultaneously. For the XOR problem, the input and output activation values are clearly defined.

Input Activations		**Output Activations T_i**
0	0	$T_0 = 0$
1	0	$T_1 = 1$
0	1	$T_2 = 1$
1	1	$T_3 = 0$

These values represent our full training set. Note that you must have a network constructed before you can train it. Training is NOT part of the network; it is simply a way to determine the values of the weights that should be used by the network. For this complete training set our simple example becomes

$$E = \sum_{t=0}^{3} E_t = \frac{1}{2} \sum_{t=0}^{3} \sum_{i=0}^{0} \left(T_{it} - F_{it}\right)^2$$

$$E = \frac{1}{2} \sum_{t=0}^{3} \left(T_{0t} - F_{0t}\right)^2 = \frac{1}{2}\left[\left(T_{00} - F_{00}\right)^2 + \left(T_{01} - F_{01}\right)^2 + \left(T_{02} - F_{02}\right)^2 + \left(T_{03} - F_{03}\right)^2\right]$$

$$E = \frac{1}{2} \sum_{t=0}^{3} \left(T_{0t} - F_{0t}\right)^2 = \frac{1}{2}\left[\left(0.0 - F_{00}\right)^2 + \left(1.0 - F_{01}\right)^2 + \left(1.0 - F_{02}\right)^2 + \left(0.0 - F_{03}\right)^2\right]$$

Alternatively, you could also minimize the error for each training activation set. It turns out that the latter converges in about half the time as the former in most cases, so it is a useful exercise to explore both options. Our error function then takes on four values:

$$E_t = \frac{1}{2}\sum_{i=0}^{0}\left(T_{it} - F_{it}\right)^2$$

$$E_0 = \frac{1}{2}\left(T_{00} - F_{00}\right)^2,\ E_1 = \frac{1}{2}\left(T_{01} - F_{01}\right)^2,\ E_2 = \frac{1}{2}\left(T_{02} - F_{02}\right)^2,\ E_3 = \frac{1}{2}\left(T_{03} - F_{03}\right)^2$$

or

$$E_0 = \frac{1}{2}\left(0.0 - F_{00}\right)^2,\ E_1 = \frac{1}{2}\left(1.0 - F_{01}\right)^2,\ E_2 = \frac{1}{2}\left(1.0 - F_{02}\right)^2,\ E_3 = \frac{1}{2}\left(0.0 - F_{03}\right)^2$$

You now have a complete example of a simple XOR network that can be put into Excel and coded in your favorite language. The two approaches should agree at each step along the way to the output value and the error function for a known set of weights.