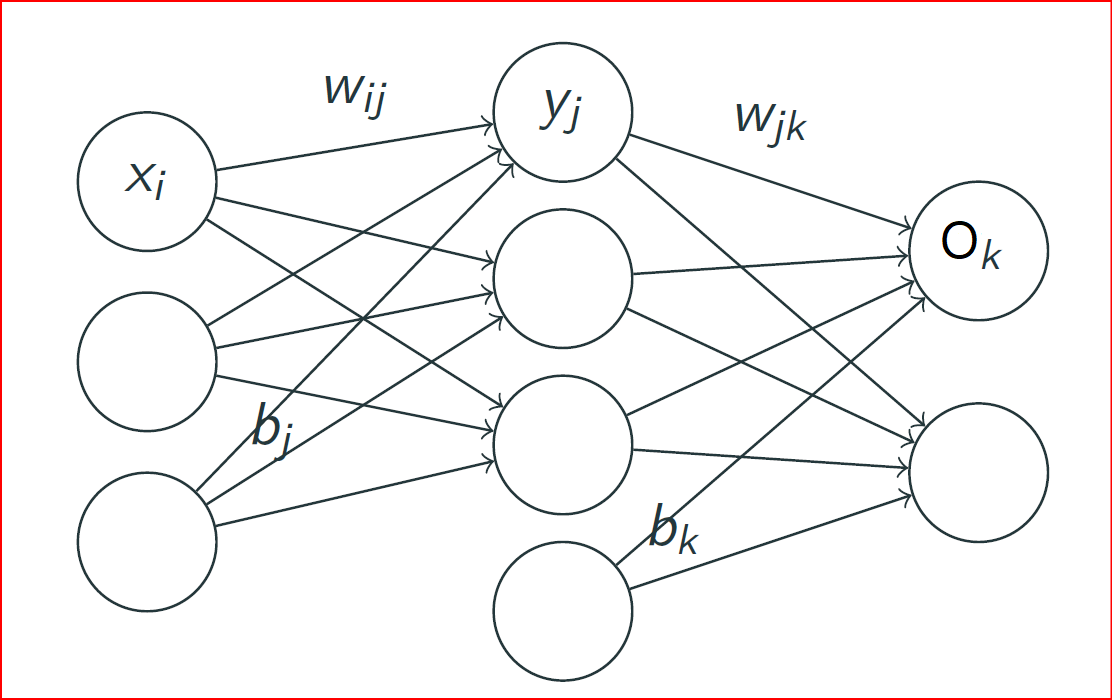
Determine and of loss function

*E(w, b) =*

for a network with one input layer (with NI neurons), output layer (with NO neurons) and hidden layer (with NH neurons). Note that every neuron is assumed to be connected to every neuron of the next layer, i.e., a Multi-Layer Perceptron is considered. Further the sigmoid function is the considered action function for every neuron in the hidden and output layer.

**Solution:**



Here,

*wij*: weights connecting node i in layer (*l – 1*) to node *j* in layer *l*.

*bj, bk*: bias for nodes *j* and *k*.

*uj, uk*: inputs to nodes *j* and *k* (where *uj = bj + ∑xiwij*).

*gj, gk*: activation function for node *j* (applied to *uj*) and node *k*.

*yj = gj(uj), Ok = gk(uk)*: output/activation of nodes *j* and *k*.

*tk*: target value for node *k* in the output layer.

Nodes in the hidden layer

Forward-propagate for each output *Ok*

*Ok* = *gk(uk) = gk(bk + ∑yjwjk)* = *gk(bk + ∑gj(bj + ∑xiwij) wjk)*

Error function,

*E(w, b) =*

Let's start at the output layer with weight *Wjk*, and *uk* = *bk* + *∑Wjkyj*

Now,

Now,

Using the value of (2), (3), and (4) we can write (1) as follows:

Here,

Now we know,

uj = bi + *∑wjkxi*

uk = bk + *∑wjkgj(ui)*

Ok = gk(uk)

Now,

Now,

Using the value of (6), (7), (8), and (9) we can write (5) as follows

Here,

Now since we know the *Ok, yj, xi, uk* and *uj* for a given set of parameter values *w, b*, we can use these expressions to calculate the gradients at each iteration and update them.

Update the weights and biases with learning rate *Ƞ*. For example