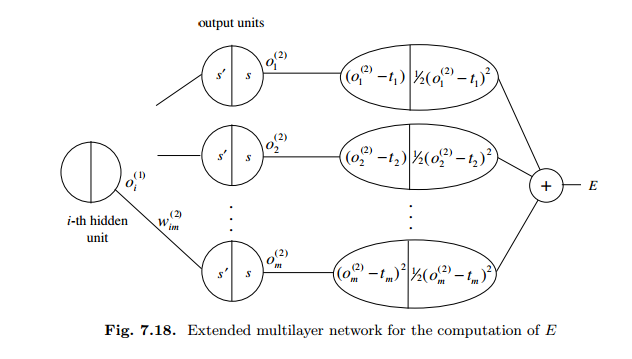
**[Steps of the algorithm](https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf)**

Figure 7.18 shows the extended network for computation of the error function. In order to simplify the discussion we deal with a single input-output pair (o, t) and generalize later to p training examples. The network has been extended with an additional layer of units. The right sides compute the quadratic deviation 1 2 (o (2) i − ti) for the i-th component of the output vector and the left sides store (o (2) i − ti). Each output unit i in the original network computes the sigmoid s and produces the output o (2) i . Addition of the quadratic deviations gives the error E. The error function for p input-output examples can be computed by creating p networks like the one shown, one for each training pair, and adding the outputs of all of them to produce the total error of the training set.



After choosing the weights of the network randomly, the back propagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

1. Feed-forward computation
2. Back propagation to the output layer
3. Back propagation to the hidden layer
4. Weight updates

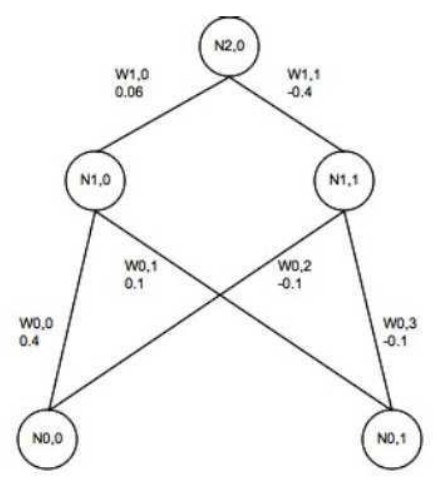
The algorithm is stopped when the value of the error function has become sufficiently small.

1. **Feed-forward computation :**

Feed forward computation or forward pass is two-step process:

* First part is getting the values of the hidden layer nodes
* Second part is using those values from hidden layer to compute value or values of output layer.

|  |  |  |
| --- | --- | --- |
| **N0,0** | **N0,1** | **Output N2,0** |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |
|  |  |  |
| *β = Learning rate = 0.45* | | |
| *α = Momentum term = 0.9* | | |
| *sigmoid function: f(x) = 1.0 / {1.0 + exp(-x)}* | | |



2 Nodes in Input layer

Weights

2 Nodes in Hidden layer

1 Node in Output layer

Weights



Input values of nodes N0, 0 and N0, 1 are pushed up to the network towards nodes in hidden layer (N1, 0 and N1, 1). They are multiplied with weights of connecting nodes and values of hidden layer nodes are calculated. Sigmoid function is used for calculations   
f(x) = 1.0/ (1.0 + exp (−x))

* + N1, 0 = f(x1) = f(w0, 0 ∗ n0, 0 + w0, 1 ∗ n0, 1) = f(0.4 + 0.1) = f(0.5) = 0.622459
  + N1, 1 = f(x2) = f(w0, 2 ∗ n0, 0 + w0, 3 ∗ n0, 1) = f(−0.1 − 0.1) = f(−0.2) = 0.450166

When hidden layer values are calculated, network propagates forward. It propagates values from hidden layer up to an output layer node (N2, 0). This is second step of feed forward computation

* + N2, 0 = f(x3) = f(w1, 0 ∗ n1, 0 + w1, 1 ∗ n1, 1)   
     = f(0.06 ∗ 0.622459 + (−0.4) ∗ 0.450166)   
     = f(−0.1427188)   
     = 0.464381

Having calculated N2, 0, forward pass is completed.

1. **Back propagation to the output layer**

Next step is to calculate error of N2, 0 node. From the table in figure 4, output should be 1. Predicted value (N2, 0) in our example is 0.464381. Error calculation is done the following way:

* + N2, 0Error = n2, 0∗ (1−n2, 0) ∗ (N2, 0Desired−N2, 0)   
     = 0.464381(1−0.464381)∗(1−0.464381)   
     = 0.133225

Once error is known, it will be used for backward propagation and weights adjustment. It is two step processes. Error is propagated from output layer to the hidden layer first. This is where learning rate and momentum are brought to equation. So weights W1, 0 and W1, 1 will be updated first. Before weights can be updated, rate of change needs to be found. This is done by multiplication of the learning rate, error value and node N1, 0 value.

* + ∆W1, 0 = β ∗ N2, 0Error ∗ n1, 0 = 0.45 ∗ 0.133225 ∗ 0.622459 = 0.037317

Now new weight for W1, 0 can be calculated.

* + W1, 0New = w1, 0Old + ∆W1, 0 + (α ∗ ∆(t − 1)) = 0.06 + 0.037317 + 0.9 ∗ 0 = 0.097137
  + ∆W1, 1 = β ∗ N2, 0Error ∗ n1, 1 = 0.45 ∗ 0.133225 ∗ 0.450166 = 0.026988
  + W1, 1New = w1, 1Old + ∆W1, 1 + (α ∗ ∆(t − 1)) = −0.4 + 0.026988 = −0.373012

The value of ∆ (t − 1) is previous delta change of the weight. In our example, there is no previous delta change so it is always 0. If next iteration were to be calculated, this would have some value.

1. **Back propagation to the hidden layer**

Now error has to be propagated from hidden layer down to the input layer. This is bit more complicated than propagating error from output to hidden layer. In previous case, output from node N2, 0 was known beforehand. Output of nodes N1, 0 and N1, 1 was unknown. Let’s start with finding N1, 0 error first. This will be calculated multiplying new weight W1, 0 value with error for the node N2, 0 value. Same way error for N1, 1 node will be found.

* + N1, 0Error = N2, 0Error\* W1, 0New = 0.133225 ∗ 0.097317 = 0.012965
  + N1, 1Error = N2, 0Error \* W1, 1New = 0.133225\* (−0.373012) = −0.049706

Once error for hidden layer nodes is known, weights between input and hidden layer can be updated. Rate of change first needs to be calculated for every weight:

* + ∆W0, 0 = β \* N1, 0Error \* N0,0 = 0.45 \* 0.012965 = 0.005834
  + ∆W0, 1 = β \* N1, 0Error \* N0, 1 = 0.45 \* 0.012965 \* 1 = 0.005834
  + ∆W0, 2 = β \* N1, 1Error \* N0, 0 = 0.45 \* −0.049706 \* 1 = −0.022368
  + ∆W0, 3 = β ∗ N1, 1Error ∗ N0, 1 = 0.45 ∗ −0.049706 ∗ 1 = −0.022368

Than we calculate new weights between input and hidden layer.

* + W0, 0New = W0, 0Old + ∆W0, 0 + (α \* ∆(t − 1)) = 0.4 + 0.005834 + 0.9 \* 0 = 0.405834
  + W0, 1New = w0, 1Old + ∆W0, 1 + (α ∗ ∆(t − 1)) = 0.1 + 0.005834 + 0 = 0.105384
  + W0, 2New = w0, 2Old + ∆W0, 2 + (α ∗ ∆(t − 1)) = −0.1 + −0.022368 + 0 = −0.122368
  + W0, 3New = w0, 3Old + ∆W0, 3 + (α ∗ ∆(t − 1)) = −0.1 + −0.022368 + 0 = −0.122368

1. **Weight updates**

Important thing is not to update any weights until all errors have been calculated. It is easy to forget this and if new weights were used while calculating errors, results would not be valid. Here is quick second pass using new weights to see if error has decreased.

* + N1, 0 = f(x1) = f(w0, 0 ∗ n0, 0 + w0, 1 ∗ n0, 1) = f(0.406 + 0.1) = f(0.506) = 0.623868314
  + N1, 1 = f(x2) = f(w0, 2 ∗ n0, 0 + w0, 3 ∗ n0, 1) = f(−0.122 − 0.122) = f(−0.244) = 0.43930085
  + N2, 0 = f(x3) = f(w1, 0 ∗ n1, 0 + w1, 1 ∗ n1, 1)   
     = f(0.097 ∗ 0.623868314 + (−0.373) ∗ 0.43930085) = f(−0.103343991) = 0.474186972

Having calculated N2, 0, forward pass is completed.

Next step is to calculate error of N2, 0 node. From the table in figure 4, output should be 1. Predicted value (N2, 0) in our example is 0.464381. Error calculation is done in following way.

* + N2, 0Error = n2, 0∗(1−n2, 0)∗(N2, 0Desired−N2, 0)   
     = 0.474186972∗(1−0.474186972)∗(1−0.474186972) = 0.131102901

So after initial iteration, calculated error was 0.133225 and new calculated error is 0.131102. Our algorithm has improved, not by much but this should give good idea on how BP algorithm works. Although this was very simple example, it should help to understand basic operation of BP algorithm. It can be said that algorithm learned through iterations. According to Dspguide.com [2010] number of iterations in typical NN would be any number from ten to ten thousands. This is only one example set pass that could be repeated many times until error is small enough.

**Reference:**

<http://www.dataminingmasters.com/uploads/studentProjects/NeuralNetworks.pdf>

<https://theclevermachine.wordpress.com/2014/09/08/derivation-derivatives-for-common-neural-network-activation-functions/>

<https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/>

<https://theclevermachine.wordpress.com/2014/09/11/a-gentle-introduction-to-artificial-neural-networks/>

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

<http://neuralnetworksanddeeplearning.com/chap4.html>

<http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/>

<http://karpathy.github.io/neuralnets/>

<http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html>