Backpropagation

Sayeda Suaiba Anwar
Lecturer, Department of CSE
East Delta University

Definition

Backpropagation is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows to reduce error rates and make the model reliable by increasing its generalization.

Iteratively process a set of training tuples & compare the network's prediction with the actual known target value.

For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value

Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer.

Steps of Backpropagation

- Initialize weights to small random numbers, associated with biases
- Propagate the inputs forward (by applying activation function)
- Backpropagate the error (by updating weights and biases)
- Terminating condition (when error is very small)

1. Initialize the weights

- **The weights in the network are initialized to small random numbers** (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5).
- Each unit has a *bias associated with* it

2. Propagate the inputs forward

- **First, the training tuple is fed to the network's input** layer.The inputs pass through the input units, unchanged.
- That is, for an input unit, j, its output, O_j , is equal to its input value, I_j .
- Next, the net input and output of each unit in the hidden and output layers are computed.The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs.
- Each connection has a weight.

2. Propagate the inputs forward

To compute the net input to the unit, each input connected to the unit is multiplied by its corresponding weight, and this is summed. Given a unit, j in a hidden or output layer, the net input, I_j , to unit j is

✓
$$\boxed{I_j = \sum_i w_{ij} O_i + \theta_j,}$$

where w_{ij} is the weight of the connection from unit i in the previous layer to unit j; O_i is the output of unit i from the previous layer; and j is the **bias of the unit.**

2. Propagate the inputs forward

- Each unit in the hidden and output layers takes its net input and then applies an **activation** function to it.
- The **logistic or sigmoid function is used here for easy understanding.**
- Given the net input I_j to unit j, then O_j , the output of unit j, is computed as

$$\boxed{O_j = \frac{1}{1+e^{-I_j}}.}$$

2. Propagate the inputs forward

- We compute the output values, O_j , for each hidden layer, up to and including the output layer, which gives the network's prediction.
- In practice, it is a good idea to cache (i.e., save) the intermediate output values at each unit as they are required again later when backpropagating the error.

3. Backpropagate the error

The error is propagated backward by updating the weights and biases to reflect the error of the network's prediction. For a unit *j in the output layer*, the error E_{rrj} *is computed by*

$$\boxed{Err_j = O_j(1 - O_j)(T_j - O_j),}$$

where O_j is the actual output of unit j, and T_j is the known target value of the given training tuple.

3. Backpropagate the error

To compute the error of a hidden layer unit j, the weighted sum of the errors of the units connected to unit j in the next layer are considered.

The error of a hidden layer unit j is

$$\boxed{Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk},}$$

where w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and E_{rrk} is the error of unit k.

3. Backpropagate the error

•The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where $\triangle w_{ij}$ is the change in weight w_{ij} :

$$\Delta w_{ij} = (l) Err_j O_i.$$

$$w_{ij} = w_{ij} + \Delta w_{ij}.$$

The variable ***l*** is the **learning rate, a constant typically having** a value between 0.0 and 1.0.

3. Backpropagate the error

Biases are updated by the following equations, where 1j is the change in bias j :

$$\Delta\theta_j = (l)\,Err_j.$$
$$\theta_j = \theta_j + \Delta\theta_j.$$

Note that here we are updating the weights and biases after the presentation of each tuple.

This is referred to as **case updating**.

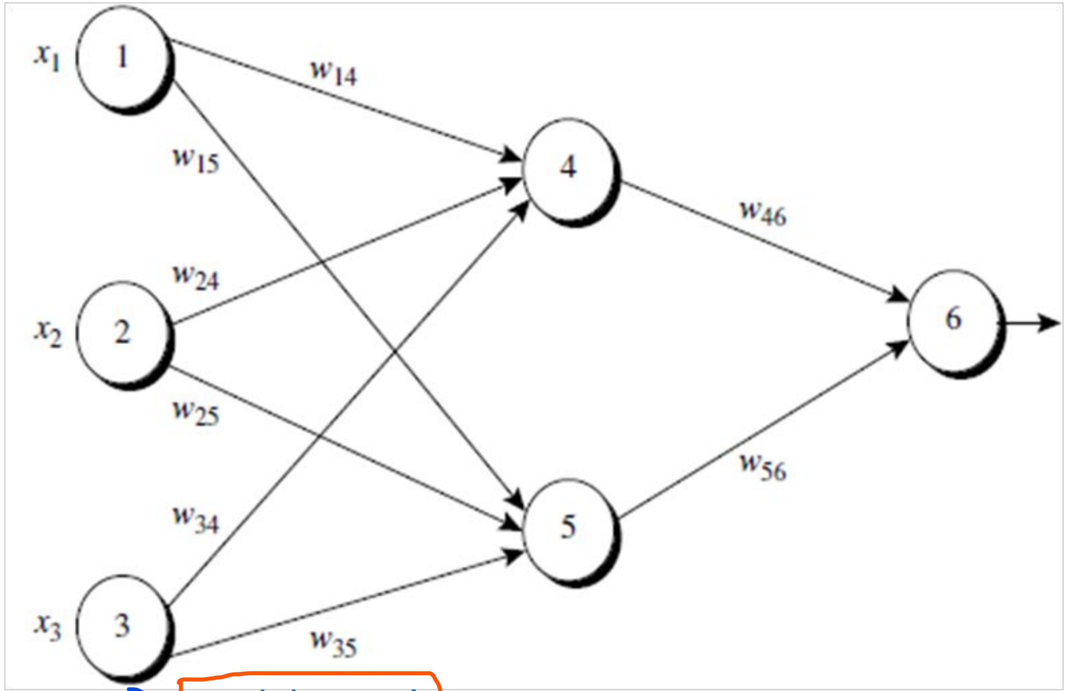
Alternatively, the weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all the tuples in the training set have been presented.

4. Terminating Condition

Training stops when -

- All **Dw**$_{ij}$ in the previous epoch are so small as to be below some specified threshold
- The percentage of tuples misclassified in the previous epoch is below some threshold
- A pre-specified number of epochs has expired.

In practice, several hundreds of thousands of epochs may be required before the weights will converge.



Example

Figure shows a multilayer feed-forward neural network. Let the learning rate be **0.9**. The initial weight & bias values of the network are given in Table, along with the first training tuple, ***X* = (1, 0, 1)**, ***with a class label of 1*****.**

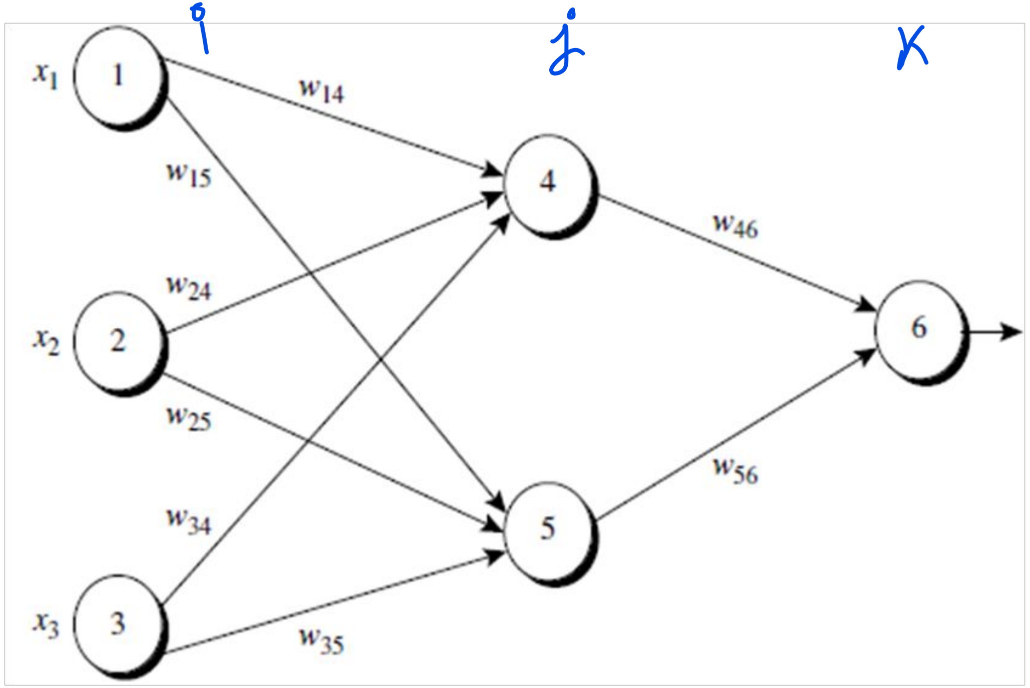
T

Bias

Training tuple

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

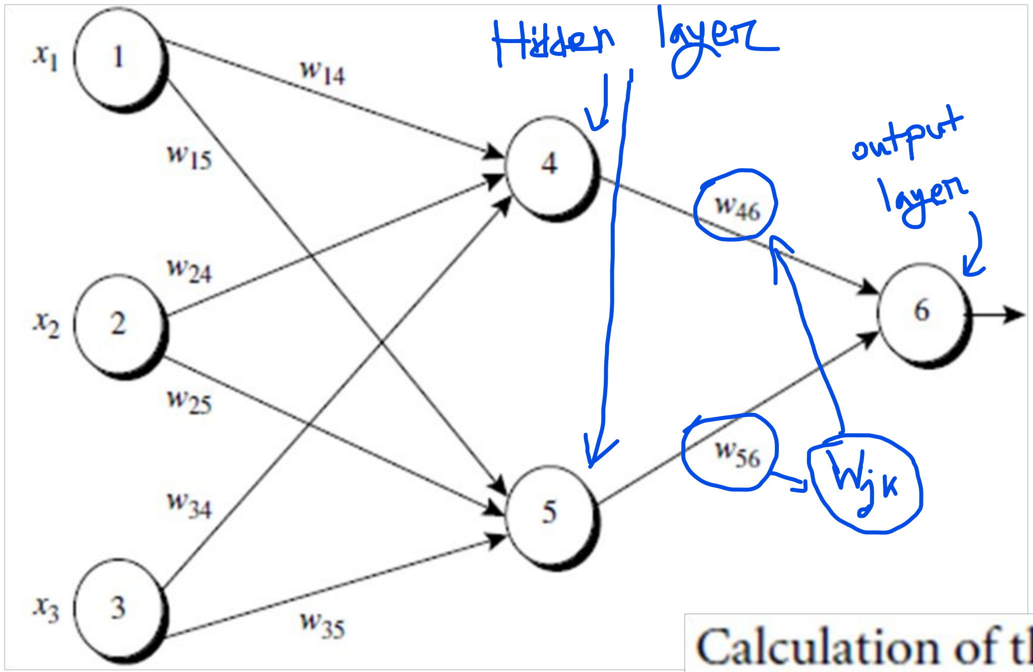


$$I_j = \sum_i w_{ij} O_i + \theta_j,$$

$$O_j = \frac{1}{1+e^{-I_j}}.$$

Net Input and Output Calculations

Unit, j	*Net Input, I_j*	*Output, O_j*
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$



Output Layers

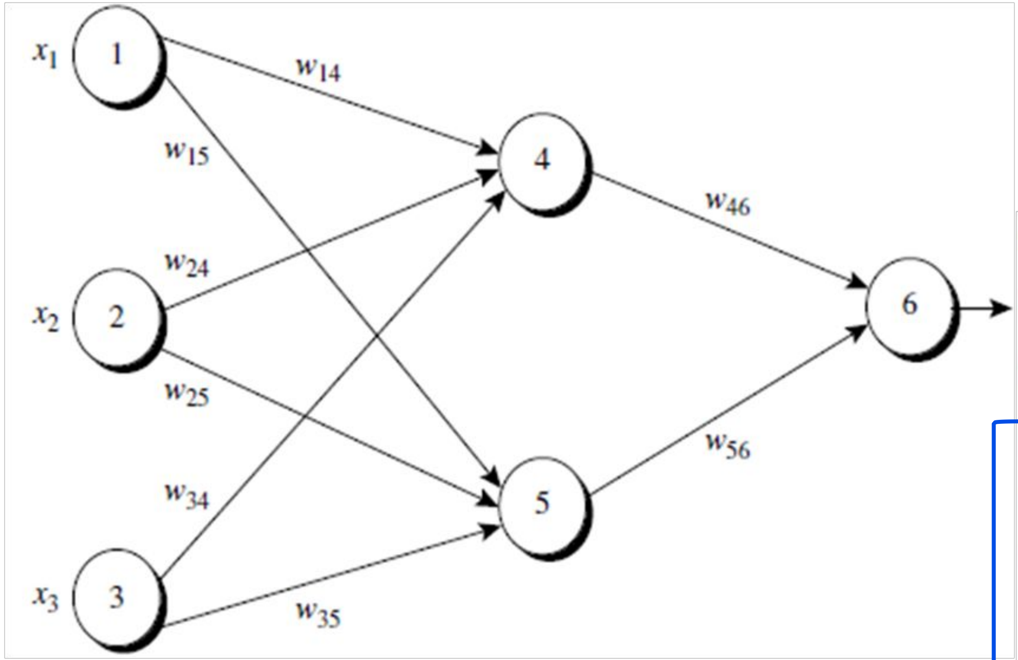
$$\boxed{Err_j = O_j(1 - O_j)(\underline{T_j} - O_j),}$$

Hidden Layers

$$\boxed{Err_j = O_j(1 - O_j)\sum_k \underline{Err_k}\,\underline{w_{jk}},}$$

Calculation of the Error at Each Node

	Unit, j	Err_j
OL	6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
HL	5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
HL	4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$



$$\Delta\theta_j = (l)\,Err_j.$$
$$\theta_j = \theta_j + \Delta\theta_j.$$

Calculations for Weight and Bias Updating

w l E O

Weight or Bias	*New Value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

$$\Delta w_{ij} = (l)\,Err_j O_i.$$
$$w_{ij} = w_{ij} + \Delta w_{ij}.$$

"How can we classify an unknown tuple using a trained network?"

- To classify an unknown tuple, **X, the tuple is input to the trained network, and the net input and** output of each unit are computed.
- If there is one output node per class, then the output node with the highest value determines the predicted class label for **X.**
- **If there is only one output** node, then output values greater than or equal to 0.5 may be considered as belonging to the positive class, while values less than 0.5 may be considered negative.
- Several variations and alternatives to the backpropagation algorithm have been proposed for classification in neural networks.