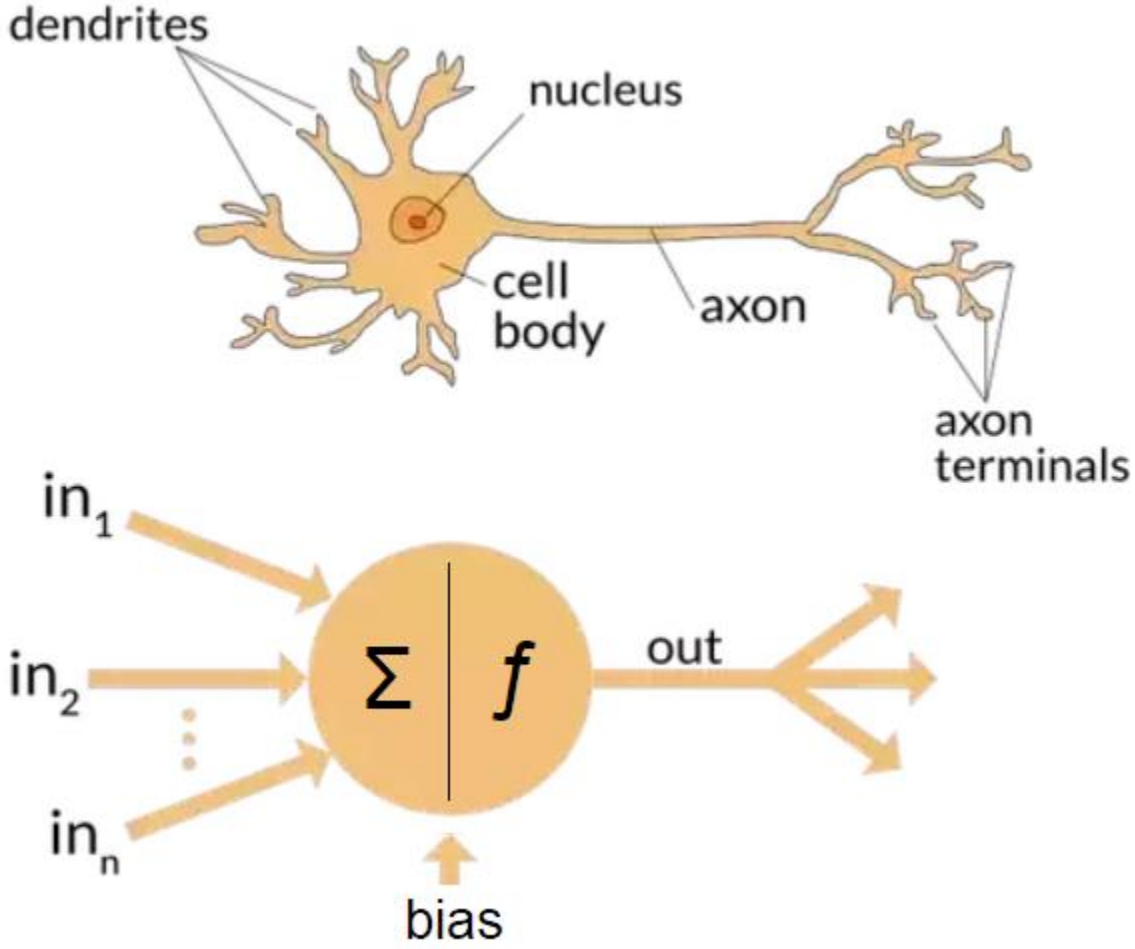
Neural Networks

Muhammad Atif Tahir

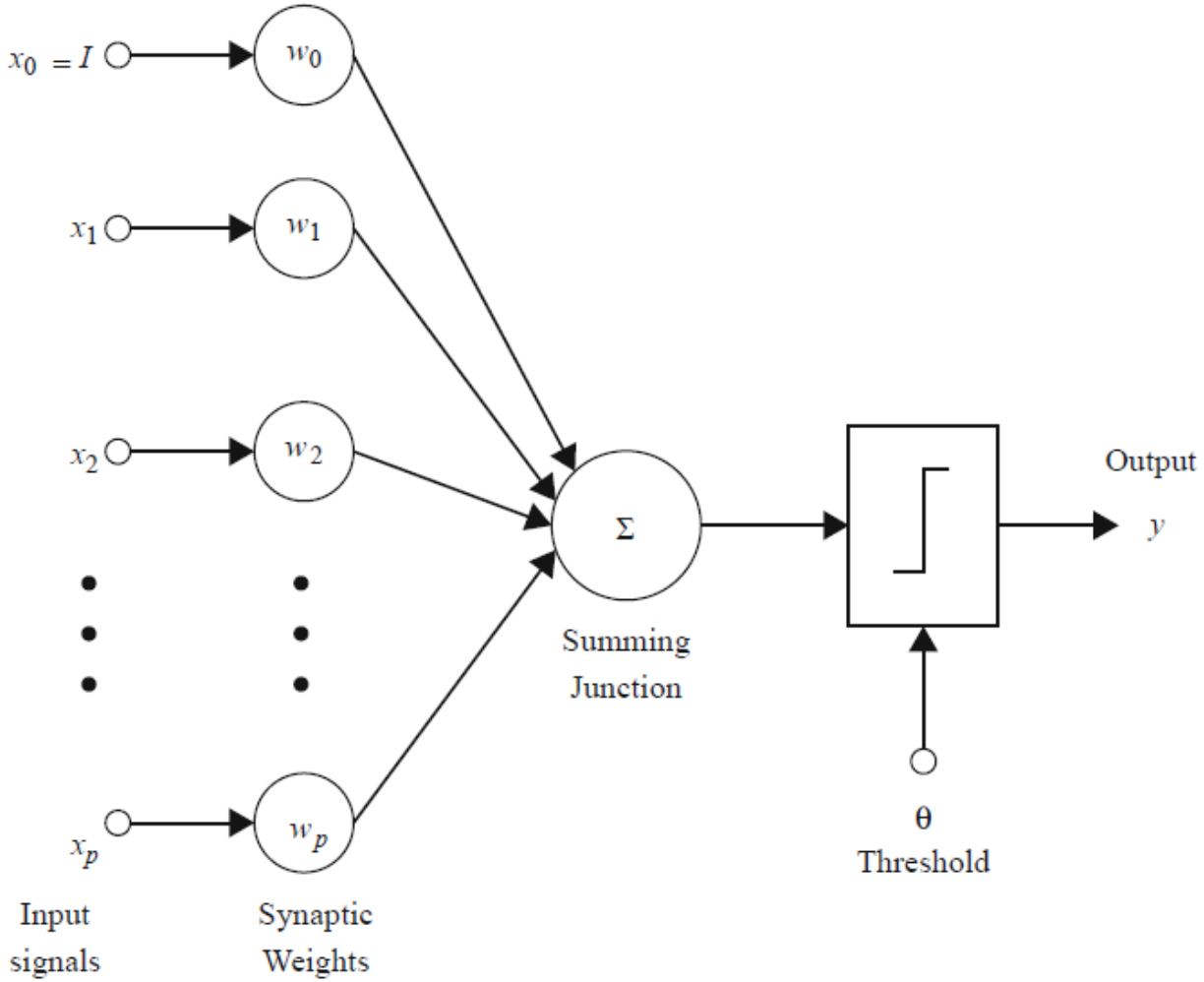
Introduction to NN (Nice Video)

- https://www.youtube.com/watch?v=bfmFfD2RIcg

Structure of a neuron



Structure of ANN



ANN

- A mathematical model of the neuron, is called the perceptron
- Try and mimic our understanding of the functioning of the brain,
  - in particular its parallel processing characteristics,
  - in order to emulate some of its pattern recognition capabilities
- An artificial neural network is a parallel system, which is capable of resolving paradigms that linear computing cannot resolve
- Like its biological predecessor, an ANN is an adaptive system, i.e., parameters can be changed during operation (training) to suit the problem

Multilayer backpropogation

Algorithm: Backpropagation. Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

Input:

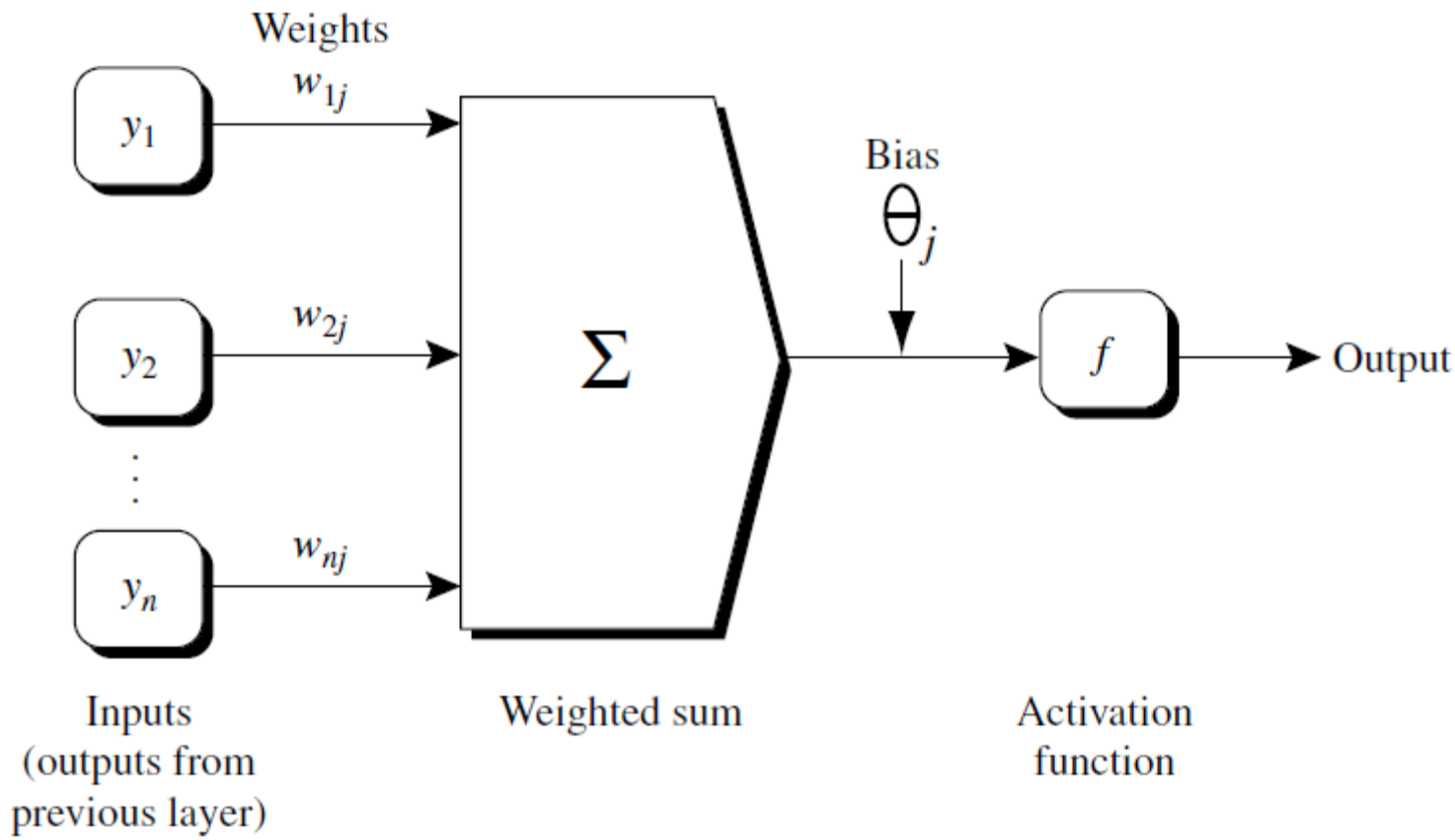
- *D*, a data set consisting of the training tuples and their associated target values;
- *l*, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

(1) Initialize all weights and biases in *network*;
(2) **while** terminating condition is not satisfied {
(3) **for** each training tuple $\boldsymbol{X}$ in D {
(4) // Propagate the inputs forward:
(5) **for** each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7) **for** each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
(10) // Backpropagate the errors:
(11) **for** each unit j in the output layer
(12) $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
(13) **for** each unit j in the hidden layers, from the last to the first hidden layer
(14) $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
(15) **for** each weight w_{ij} in *network* {
(16) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18) **for** each bias θ_j in *network* {
(19) $\Delta \theta_j = (l) Err_j$; // bias increment
(20) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21) } }

Weights initialization

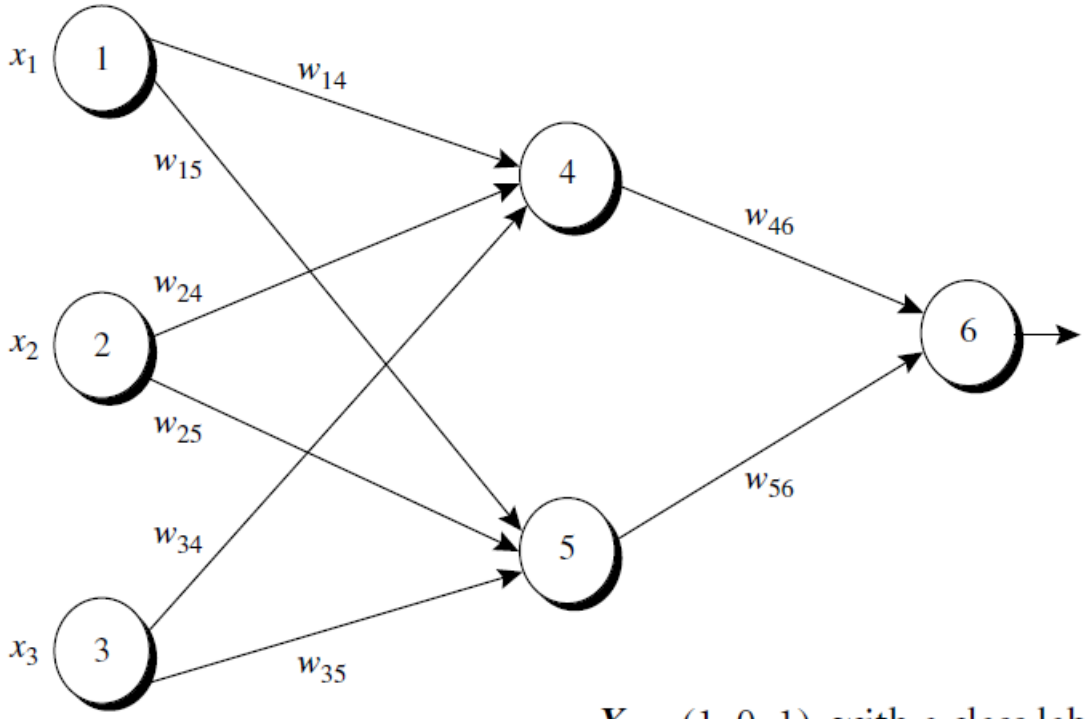
- **Initialize the weights:** The weights in the network are initialized to small random numbers (e.g., ranging from -1.0 to 1.0, or -0.5 to 0.5). Each unit has a *bias associated with*. The biases are similarly initialized to small random numbers


Weights
w_{1j}
y_1
w_{2j}
y_2
w_{nj}
y_n
Bias
θ_j
Σ
f
Output
Inputs
(outputs from
previous layer)
Weighted sum
Activation
function

Convergence condition

- **Terminating condition:** Training stops when
- All Δwij *in the previous epoch are so small as to be below some specified* threshold, or
- The percentage of tuples misclassified in the previous epoch is below some threshold, or
- A prespecified number of epochs has expired.

Worked example



$\boldsymbol{X} = (1, 0, 1)$, with a class label of 1.

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

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	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Calculation of the Error at Each Node

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

Calculations for Weight and Bias Updating

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	$-0.4 + (0.9)(-0.0087) = -0.408$

Classification of unknown datapoint

- 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. (There is no need for computation and/or backpropagation of the error)
- 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

Critique of ANN

- Neural networks involve long training times and are therefore more suitable for applications where this is feasible.
- They require a number of parameters that are typically best determined empirically such as the network topology or “structure.” Neural networks have been criticized for their poor interpretability
- For example, it is difficult for humans to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network. These features initially made neural networks less desirable for data mining

Advantage

- Advantages of neural networks, however, include their high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained. They can be used when you may have little knowledge of the relationships between attributes and classes.
- They are well suited for continuous-valued inputs *and outputs, unlike most* decision tree algorithms. They have been successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text

- Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process

- It can perform tasks which a linear classifier cannot
- Multilayer feed-forward networks, given enough hidden units and enough training samples, can closely approximate any function