331 – Intro to Intelligent Systems
Week 10
Neural Networks

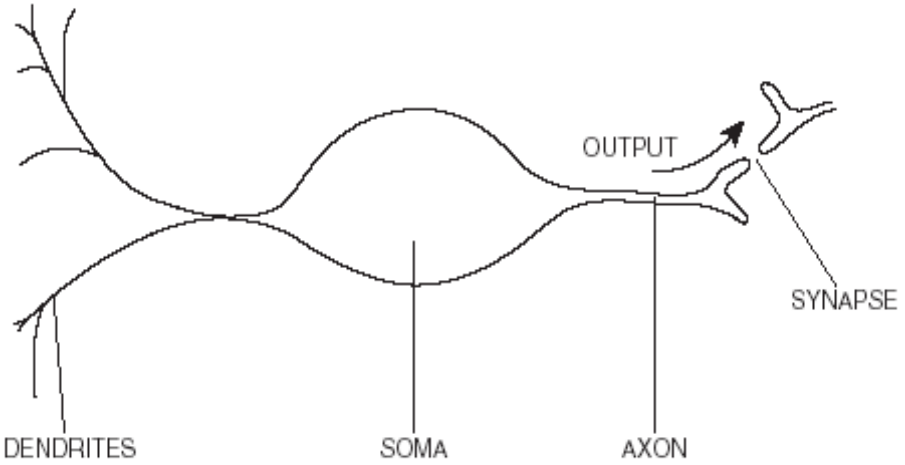
T.J. Borrelli

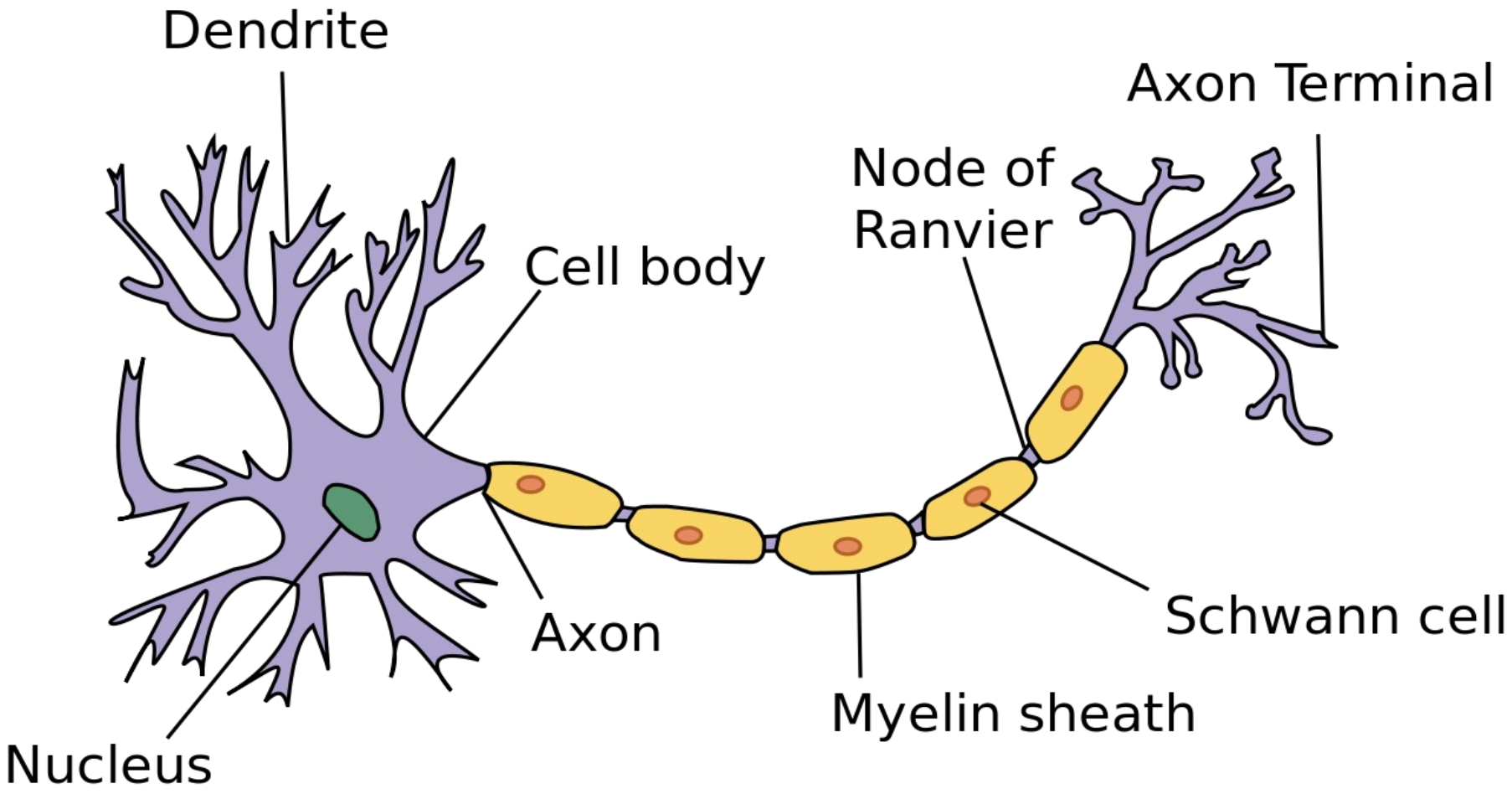
Neural Networks

- A neural network is a network of artificial neurons, which is based on the operation of the human brain
- Neural networks usually have their nodes arranged in layers
- One layer is the input layer and another is the output layer
- There are one or more hidden layers between these two

Biological Neurons

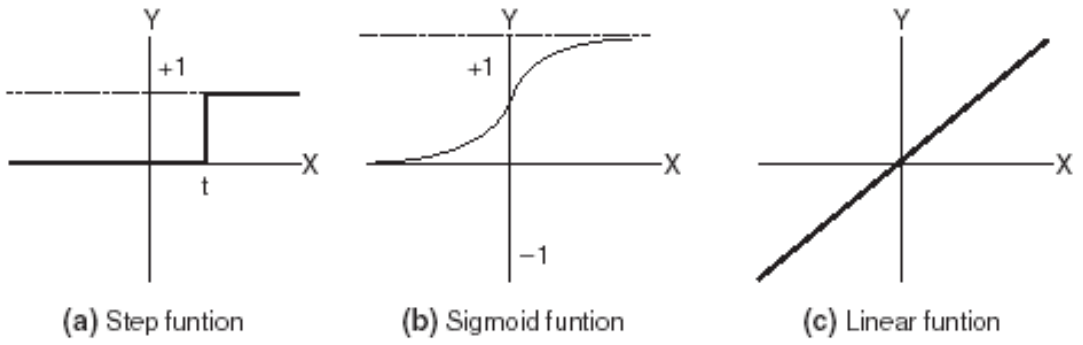
- The human brain is made up of billions of simple processing units called neurons
- Inputs are received on dendrites, and if the input levels are over a threshold, the neuron fires, passing a signal through the axon to the synapse which then connects to another neuron




Dendrite
Axon Terminal
Node of
Ranvier
Cell body
Axon
Schwann cell
Myelin sheath
Nucleus

Artificial Neurons

- Each neuron in the network receives one or more inputs
- An activation function is applied to the inputs which determines the output of the neuron (the activation level)
  - Three typical activation functions:



(a) Step funtion (b) Sigmoid funtion (c) Linear funtion

Artificial Neurons

- A step activation function works as follows:

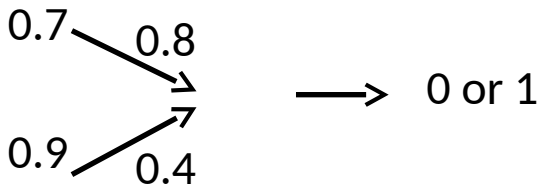
$$X = \sum_{i=1}^{n} w_i x_i \qquad Y = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \le t \end{cases}$$

- Each node *i* has a weight, w_i associated with it
  - The input to node *i* is x_i
  - t is the threshold
- If the weighted sum of the inputs to the neuron is above the threshold, the neuron will fire

Artificial Neurons

Example: Consider a simple neuron that has just two inputs. Each input has a weight associated with it:

$w_1 = 0.8$ and $w_2 = 0.4$



The inputs to the neurons are:

$x_1 = 0.7$ and $x_2 = 0.9$

The summed weight of these inputs is:

$(0.8 \times 0.7) + (0.4 \times 0.9) = 0.92$

Inhibited and strengthened connections

X_1	X_2	$X_1 * X_2$ (Output)
+	+	+
+	-	-
-	+	-
-	-	+

Artificial Neurons

The activation level Y, is defined for this neuron as:

$$Y = +1 \text{ for } X > t$$
$$0 \text{ for } X \leq t$$

Hence, if t < 0.92, then this neuron will fire (output = 1) with this particular set of inputs. Otherwise it will not fire (output = 0).

For a neural network to learn, the weights associated with each connection can be **changed** in response to particular sets of inputs and events.

Perceptrons

- A perceptron (Rosenblatt, 1958) is a single neuron that classifies a set of inputs into one of two categories (usually {1,-1}, or {1,0})
- The perceptron usually uses a step function, which returns 1 if the weighted sum of inputs exceeds a threshold, and –1 (or 0) otherwise
- A perceptron can learn to represent a Boolean operator, such as AND or OR, where the result is classified as *true* if the output is 1 and *false* if the output is 0

Perceptrons

- The perceptron is trained as follows:
  - First, inputs are given random weights (usually between –0.5 and 0.5)
  - A sample training data is presented. If the perceptron misclassifies it, the weights are modified according to the following formula:

    $$W_i \leftarrow W_i + (\alpha \times x_i \times e)$$

  - where α is the learning rate (between 0 and 1) and e is the size of the error

Perceptrons

- In other words, if the perceptron incorrectly classifies a positive piece of training data as negative, then the weights need to be modified to increase the output for that set of inputs
  - Error = expected - actual
- This can be done by adding a *positive* value to the weight of an input that had a *negative* output value, and vice versa
  - $W_i \leftarrow W_i + (\alpha \times x_i \times e)$
  - e is 0 if the output is correct, otherwise it is positive if the output is too low and negative if the output is too high
  - This is known as the ***perceptron training rule***

Perceptrons

- Once this modification to the weights has taken place, the next piece of training data is used in the same way
- Once all of the training data have been applied, the process starts again, until all of the weights are correct and all errors are 0
- Each iteration of this process is known as an ***epoch***

Perceptrons

Example: Use the perceptron training rule to show how a perceptron can learn the logical OR function. Use a threshold of 0 and a learning rate of $\alpha = 0.2$.

First, the weight associated with each of the two inputs is initialized to a random value between -1 and 1:

$w_1 = -0.2$ and $w_2 = 0.4$

Next, the first epoch is run. The training data will consist of the four combinations of 1's and 0's possible with the 2 inputs. Hence, the first piece of training data is:

$x_1 = 0$ and $x_2 = 0$

The expected output is $x_1 \vee x_2 = 0$

Perceptrons

The output is:

$Y = \text{Step}((-0.2 \times 0) + (0.4 \times 0)) = 0$

Hence the output Y is as expected and the error is 0. So the weights do not change.

Now for $x_1 = 0$ and $x_2 = 1$ the expected output is 1.

$Y = \text{Step}((-0.2 \times 0) + (0.4 \times 1)) = \text{Step}(0.4) = 1$

Again, this is correct, and so the weights do not change.

For $x_1 = 1$ and $x_2 = 0$ the expected output is also 1.

$Y = \text{Step}((-0.2 \times 1) + (0.4 \times 0)) = \text{Step}(-0.2) = 0$

This is incorrect, so the weights must be adjusted.

Perceptrons

Use the perceptron training rule $W_i \leftarrow W_i + (\alpha \times x_i \times e)$ to assign new values to the weights.

The error is 1 and the learning rate is 0.2, so assign the following value to w_1:

$w_1 = -0.2 + (0.2 \times 1 \times 1) = -0.2 + 0.2 = 0$

Use the same formula to assign a new value to w_2:

$w_2 = 0.4 + (0.2 \times 0 \times 1) = 0.4$

Because w_2 did not contribute to this error, it is not adjusted.

Perceptrons

The final piece of training data is now used, $x_1 = 1$ and $x_2 = 1$ and the new weights:

$Y = \text{Step}((0 \times 1) + (0.4 \times 1)) = \text{Step}(0.4) = 1$

This is correct, so the weights are not adjusted.

This is the end of the first epoch, and at this point the method runs again and continues to repeat until all four pieces of training data are classified correctly.

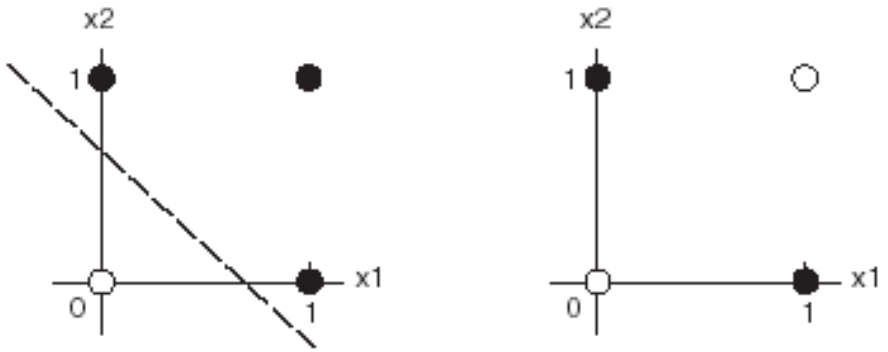
Perceptrons

This table shows the complete sequence – it takes just 3 epochs for the perceptron to correctly learn to classify input values. Lines in which an error was made are highlighted.

Epoch	X1	X2	Expected Y	Actual Y	Error	w1	w2
1	0	0	0	0	0	-0.2	0.4
1	0	1	1	1	0	-0.2	0.4
1	**1**	**0**	**1**	**0**	**1**	**0**	**0.4**
1	1	1	1	1	0	0	0.4
2	0	0	0	0	0	0	0.4
2	0	1	1	1	0	0	0.4
2	**1**	**0**	**1**	**0**	**1**	**0.2**	**0.4**
2	1	1	1	1	0	0.2	0.4
3	0	0	0	0	0	0.2	0.4
3	0	1	1	1	0	0.2	0.4
3	1	0	1	1	0	0.2	0.4
3	1	1	1	1	0	0.2	0.4

Perceptrons

- Perceptrons can only classify linearly separable functions
- The left graph shows a linearly separable function (OR)
- The right graph shows a non-linearly separable function (X-OR)



Perceptrons

- The reason that a single perceptron can only model functions that are linearly separable can be seen by examining the following functions:

$$X = \sum_{i=1}^{n} w_i x_i \qquad Y = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \leq t \end{cases}$$

- Using these functions, we are effectively dividing the search space using a *line for which X = t*. Hence in a perceptron with two inputs, the line that divides one class from the other is defined as:

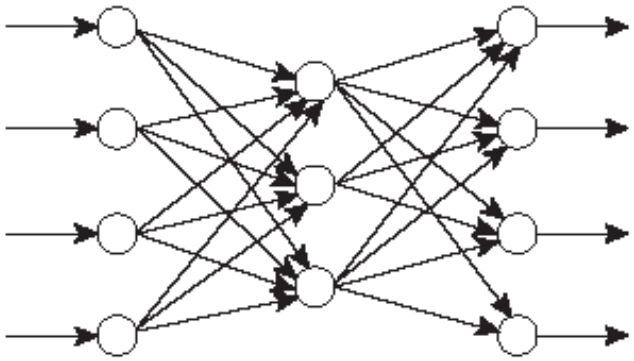
$$w_1x_1 + w_2x_2 = t$$

- The perceptron works by identifying a set of values for w_i which generates a suitable linear function. In cases where a linear function does not exist, the perceptron cannot succeed.

Multilayer Neural Networks

- Multilayer neural networks can classify a range of functions, including non-linearly separable ones
- Each input layer neuron connects to all neurons in the hidden layer
- The neurons in the hidden layer connect to all neurons in the output layer

A feed-forward network



Backpropagation

- Multilayer neural networks learn in the same way as perceptrons
- However, there are many more weights, and it is important to assign credit (or blame) correctly when changing weights
- Backpropagation networks use the *sigmoid* activation function, as it is easy to differentiate

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \frac{d\sigma(x)}{dx} = \sigma(x)\cdot(1-\sigma(x))$$

Backpropagation

- After values are fed forward through the network, errors are fed back to modify the weights in order to train the network
- For each node, we calculate an error gradient
- For a node *k* in the output layer, the error e_k is the difference between the desired output and the actual output

Backpropagation

For node j, X_j is the input,

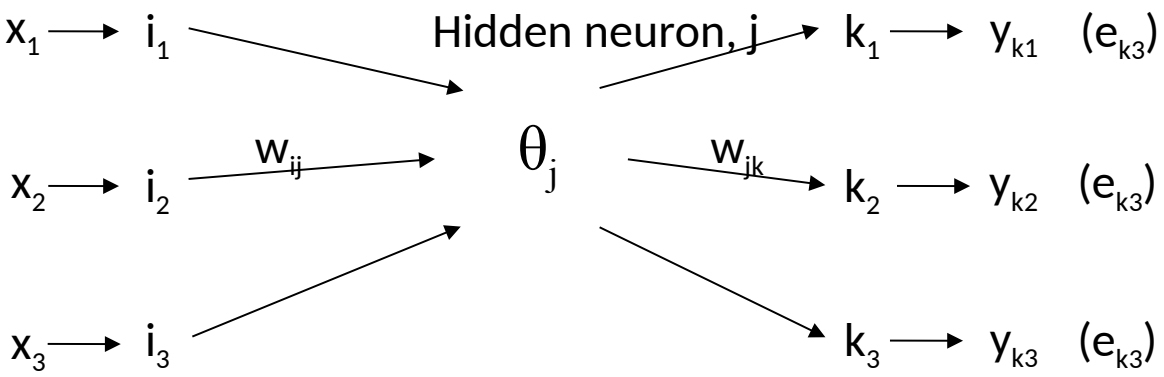
$$X_j = \sum_{i=1}^{n} w_{ij} - \theta_j$$

N is the number of inputs to j

θ_j is the threshold for node j

$$Y_j = \frac{1}{1 + e^{-X_j}}$$

Y_j is the output



Backpropagation

The error gradient for k is: $\delta_k = y_k \,.\, (1 - y_k) \,.\, e_k$

Similarly, for a node j in the hidden layer: $\delta_j = y_j \,.\, (1 - y_j) \sum_{k=1 \text{ to } n} w_{jk}\, \delta_k$

Now the weights are updated as follows:

$$W_{ij} \leftarrow W_{ij} + (\alpha \times x_i \times \delta_j)$$

$$W_{jk} \leftarrow W_{jk} + (\alpha \times y_j \times \delta_k)$$

α is the learning rate, (a positive number below 1)

Recurrent Networks

- Feed forward networks do not have memory
- Recurrent networks can have connections between nodes in any layer, which enables them to store data, i.e., have a memory
- Recurrent networks can be used to solve problems where the solution depends on previous inputs as well as current inputs (e.g. predicting stock market movements)

Evolving Neural Networks

- Neural networks can be susceptible to local maxima
- Evolutionary methods (genetic algorithms) can be used to determine the starting weights for a neural network, thus avoiding these kinds of problems
- Genetic algorithms can also be used to determine the connections between nodes, and how many input nodes should be used

Hebbian Learning

- Hebb's theory of learning is based on the observation that in biological systems when one neuron contributes to the firing of another neuron, the connection between the two is strengthened
- We can show how a network can use Hebbian learning to transfer its response from a primary or unconditioned stimulus to a conditioned stimulus
- This allows us to model the type of learning studied in Pavlov's experiments

Inhibited and strengthened connections

X_1	X_2	$X_1 * X_2$ (Output)
+	+	+
+	-	-
-	+	-
-	-	+