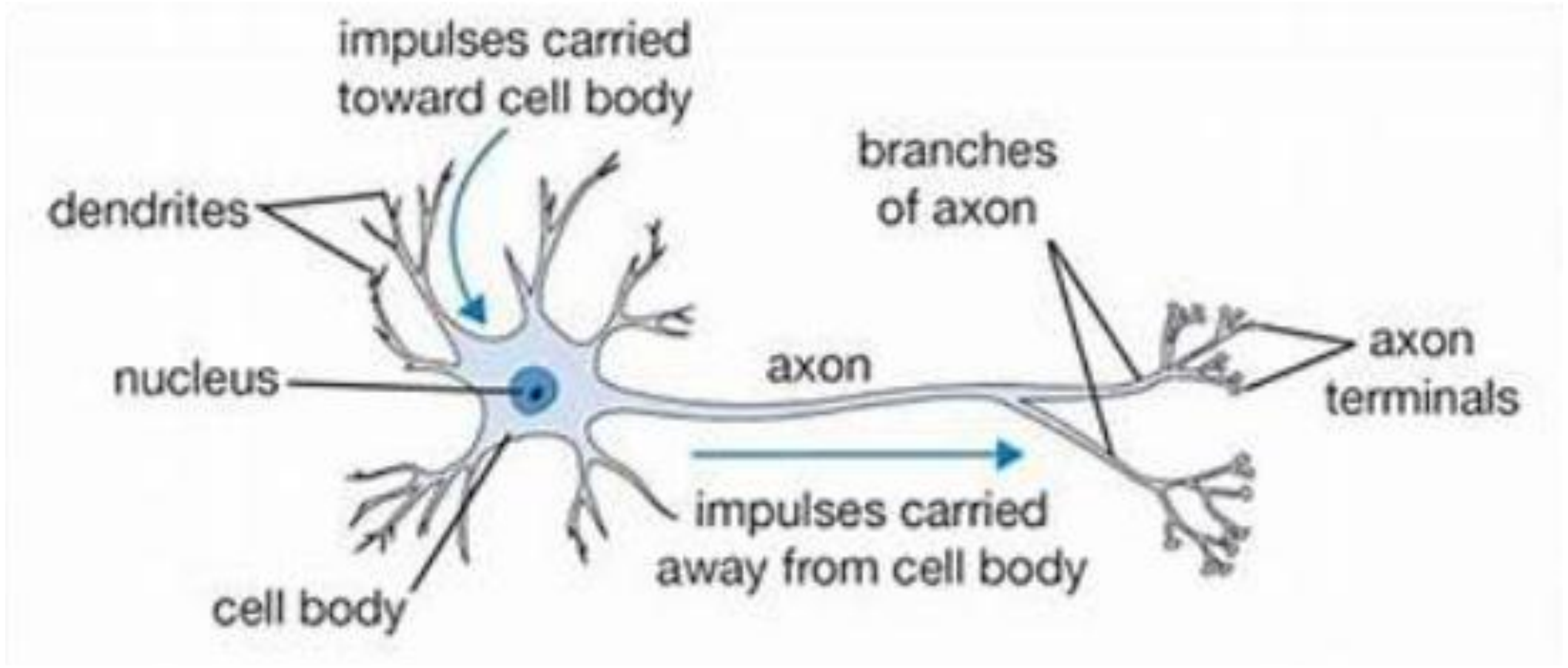
Artificial Neural Network

THE BRAIN AND THE NEURON

THE NEURON


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

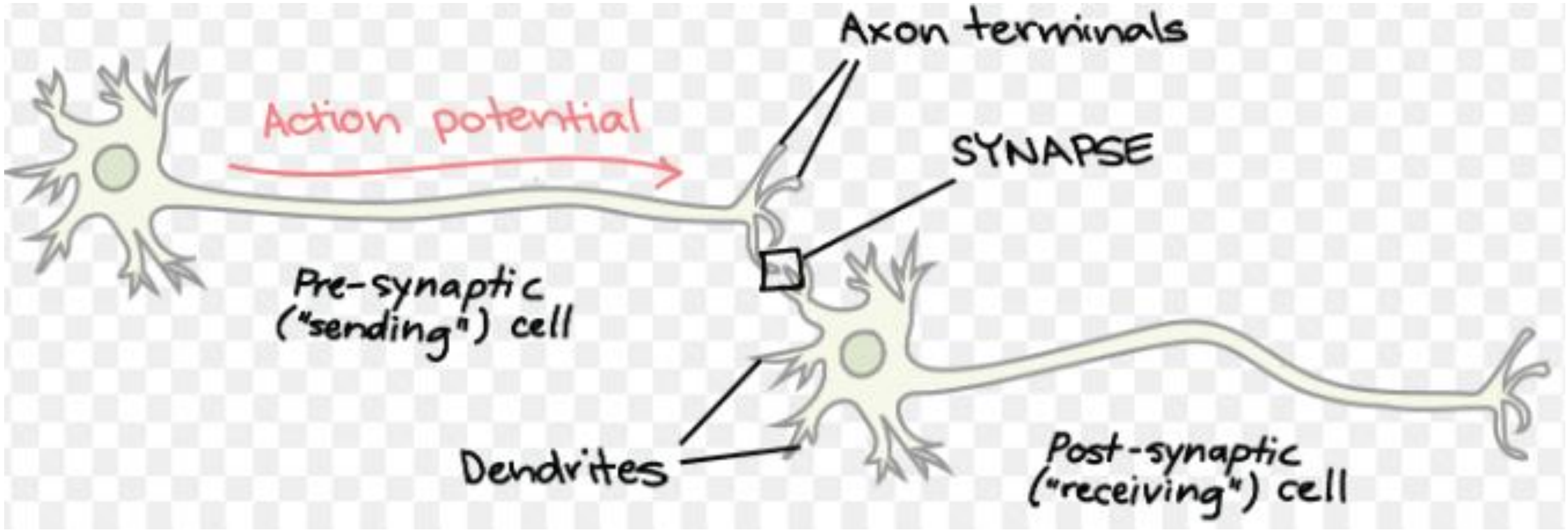
Learning in animals

- In animals, learning occurs within the brain. If we can understand how the brain works, then there might be things in there for us to copy and use for our machine learning systems.
- The processing units of the brain are nerve cells called neurons. (more than 100 billion)
- They come in lots of different types, depending upon their particular task.
- Their general operation is similar in all cases: The chemicals within the fluid of the brain raise or lower the electrical potential inside the body of the neuron. If this membrane potential reaches some threshold, the neuron spikes or fires, and a pulse of fixed strength and duration is sent down the axon

Learning in animals

- The axons divide into connections to many other neurons, connecting to each of these neurons in a synapse
- After firing, the neuron must wait for some time to recover its energy (the refractory period) before it can fire again

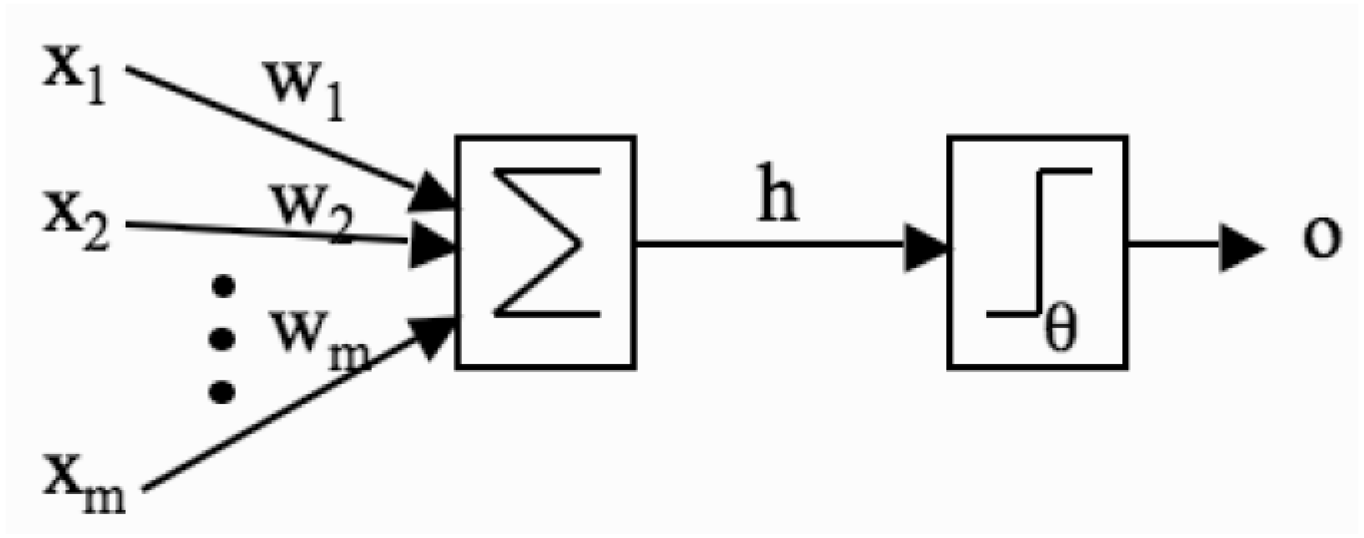
How neurons communicate with each other at synapses


Axon terminals
Action potential
SYNAPSE
Pre-synaptic ("sending") cell
Dendrites
Post-synaptic ("receiving") cell

Hebb’s Rule

- Hebb’s rule says that the changes in the strength of synaptic connections are proportional to the correlation in the firing of the two connecting neurons.
- So if two neurons consistently fire simultaneously, then any connection between them will change in strength, making it stronger.
- However, if the two neurons never fire simultaneously, the connection between them will die away.
- The idea is that if two neurons both respond to something, then they should be connected
- Egs:

McCulloch and Pitts Neurons & Perceptron – Mathematical models



$$h = \sum_{i=1}^{m} w_i x_i,$$

The inputs *xi* are multiplied by the weights *wi*, and the neurons sum their values. If this sum is greater than the threshold then the neuron fires; otherwise it does not.

Perceptron-Components

•**A set of weighted inputs** *wi* that correspond to the synapses.
•**An adder** that sums the input signals (equivalent to the membrane of the cell that collects electrical charge).

$$h = \sum_{i=1}^{m} w_i x_i,$$

•**An activation function** (a threshold function)
whether the neuron fires ('spikes') for the current inputs

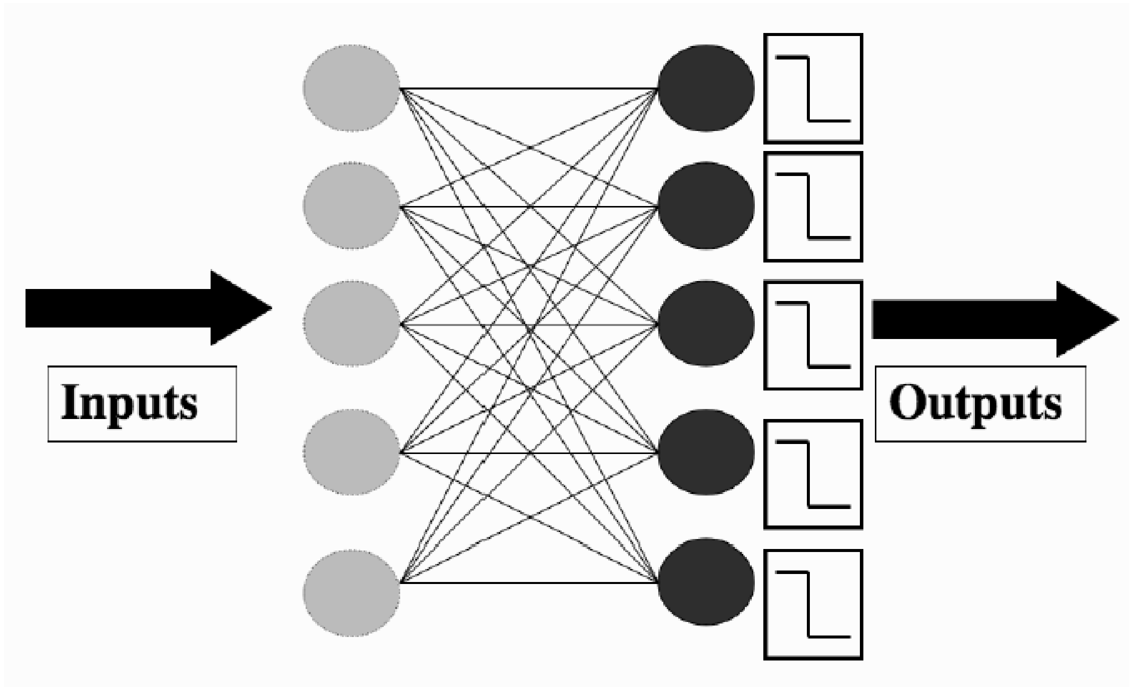
$$o = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta. \end{cases}$$

If synaptic weights are $w1 = 1, w2 = -0.5, w3 = -1$ and $x1 = 1, x2 = 0, x3 = 0.5$
h = 1 × 1 + 0 × −0.5 + 0.5 × −1 = 1 + 0 + −0.5 = 0.5
h = 0.5 in the example, and 0.5 > 0, so the neuron does fire,

Perceptron-Components

- Real neurons are much more complicated
- How should we change the weights and thresholds of the neurons so that the network gets the right answer more often?

The Perceptron network



The Perceptron network, consisting of a set of input nodes (left) connected to McCulloch and Pitts neurons using weighted connections.

The Perceptron network

- To work out whether or not a neuron should fire: we set the values of input nodes to match the elements of an input vector and then use below Equations for each neuron

$$h = \sum_{i=1}^{m} w_i x_i ,$$ and $$o = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \le \theta . \end{cases}$$

- For a neuron that is correct, we are happy, but any neuron that fired when it shouldn’t have done, or failed to fire when it should, needs to have its weights changed

The Perceptron network

<u>If the neurons gets the wrong answer for an input vector,(its output does not match the target)</u>

- There are *m* weights that are connected to that neuron, one for each of the input nodes.
- If we label the neuron that is wrong as *k*, then the weights that we are interested in are *wik*, where *i* runs from 1 to *m*.
- Change the values of these weights by
- Apply a learning rate also, usually labelled

$$\Delta w_{ik} = -(y_k - t_k) \times x_i$$

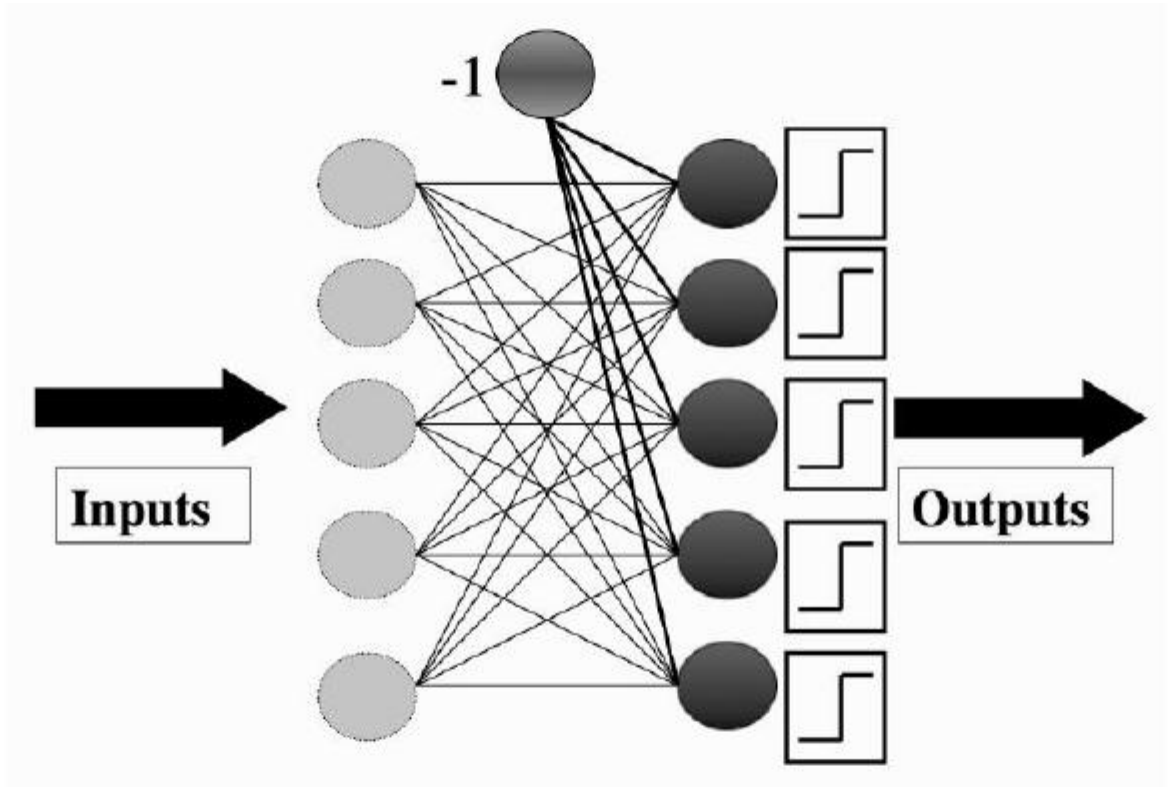
η $0.1 < \eta < 0.4$

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i.$$

The Bias Input

- When all the inputs are zero, irrespective of the the weights , the result will be zero (since zero times anything equals zero). The only way that we can control whether the neuron fires or not is through the threshold.
- If it wasn’t adjustable and we wanted one neuron to fire when all the inputs to the network were zero, and another not to fire, then we would have a problem.
- Now, we add an extra input weight to the neuron, with the value of the input to that weight always being fixed

Bias input



The Perceptron Learning Algorithm

- **Initialisation**
  - set all of the weights w_{ij} to small (positive and negative) random numbers
- **Training**
  - for T iterations or until all the outputs are correct:
    - for each input vector:
      - compute the activation of each neuron j using activation function g:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij}x_i \leq 0 \end{cases}$$

      - update each of the weights individually using:

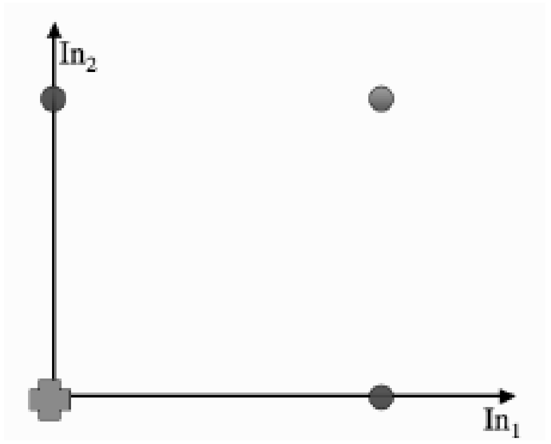
$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i$$

- **Recall**
  - compute the activation of each neuron j using:

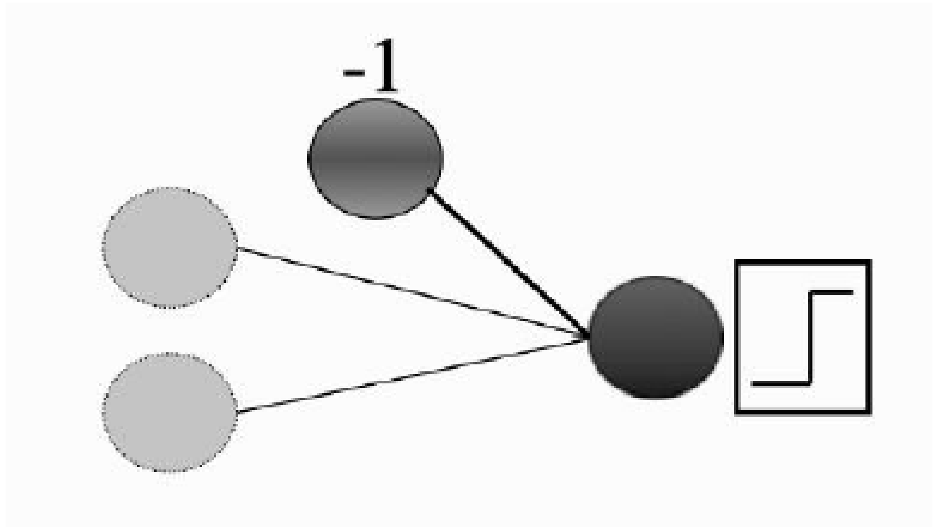
$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases}$$

The Perceptron network for OR logic function

In_1	In_2	t
0	0	0
0	1	1
1	0	1
1	1	1



Data for the OR logic function and a plot of the four datapoints.



Perceptron Network Example

•The algorithm tells us to initialize the weights to small random numbers, so we'll pick $w0 = -0.05, w1 = -0.02, w2 = 0.02$.
•Now we feed in the first input, where both inputs are 0: (0, 0). Remember that the input to the bias weight is always −1, so the value that reaches the neuron is $-0.05 \times -1 + -0.02 \times 0 + 0.02 \times 0 = 0.05$.
•This value is above 0, so the neuron fires and the output is 1,which is incorrect according to the target
•The update rule tells us that we need to apply $w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i$.
to each of the weights separately (we'll pick a value of $\eta = 0.25$ for the example):

Perceptron Network Example

$$\begin{aligned} w_0 &: -0.05 - 0.25 \times (1-0) \times -1 = 0.2, \\ w_1 &: -0.02 - 0.25 \times (1-0) \times 0 = -0.02, \\ w_2 &: 0.02 - 0.25 \times (1-0) \times 0 = 0.02. \end{aligned}$$

Now we feed in the next input (0, 1) and compute the output 0.2*-1+-0.02*0+0.02*1=-.2+0.02=-0.1.8 < 0 which will not fire and it is wrong. So update the weights

$$\begin{aligned} w_0 &: 0.2 - 0.25 \times (0-1) \times -1 = -0.05, \\ w_1 &: -0.02 - 0.25 \times (0-1) \times 0 = -0.02, \\ w_2 &: 0.02 - 0.25 \times (0-1) \times 1 = 0.27. \end{aligned}$$

For the (1, 0) input
-0.05*-1+-0.02*1+0.27*0= 0.05+-0.02=0.03, which will fire, the answer is correct.

Perceptron Network Example (cntd.)

- For the (1, 1) input the answer is already correct
- -0.05*-1+-0.02*1+0.27*1= 0.05+-0.02 +.27=.30, which will fire.

- For the (0, 0) input
- -0.05*-1+-0.02*0+0.27*0= 0.05, which will fire.
- the answer is incorrect.

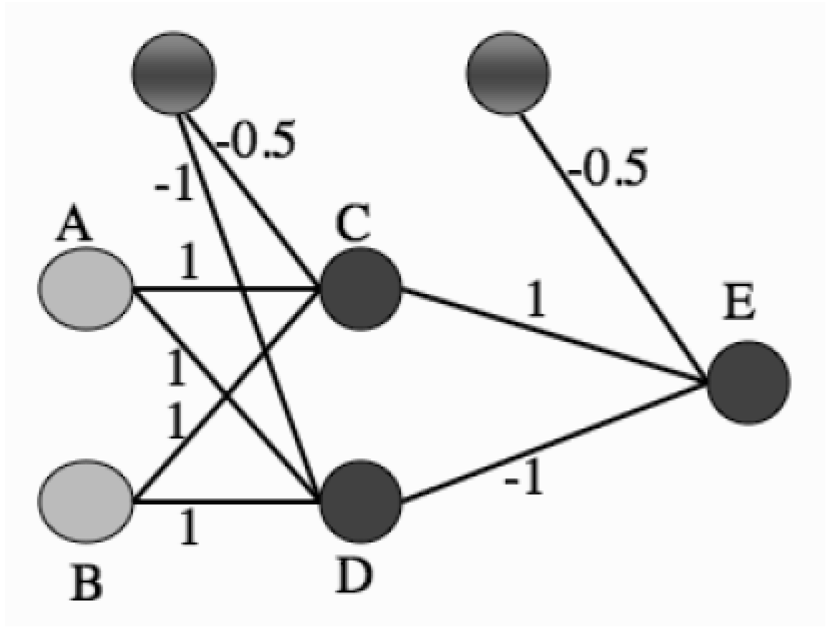
Update the weights and continue the steps

Multi Layer Perceptron

Just as it did for the Perceptron, training the MLP consists of two parts:

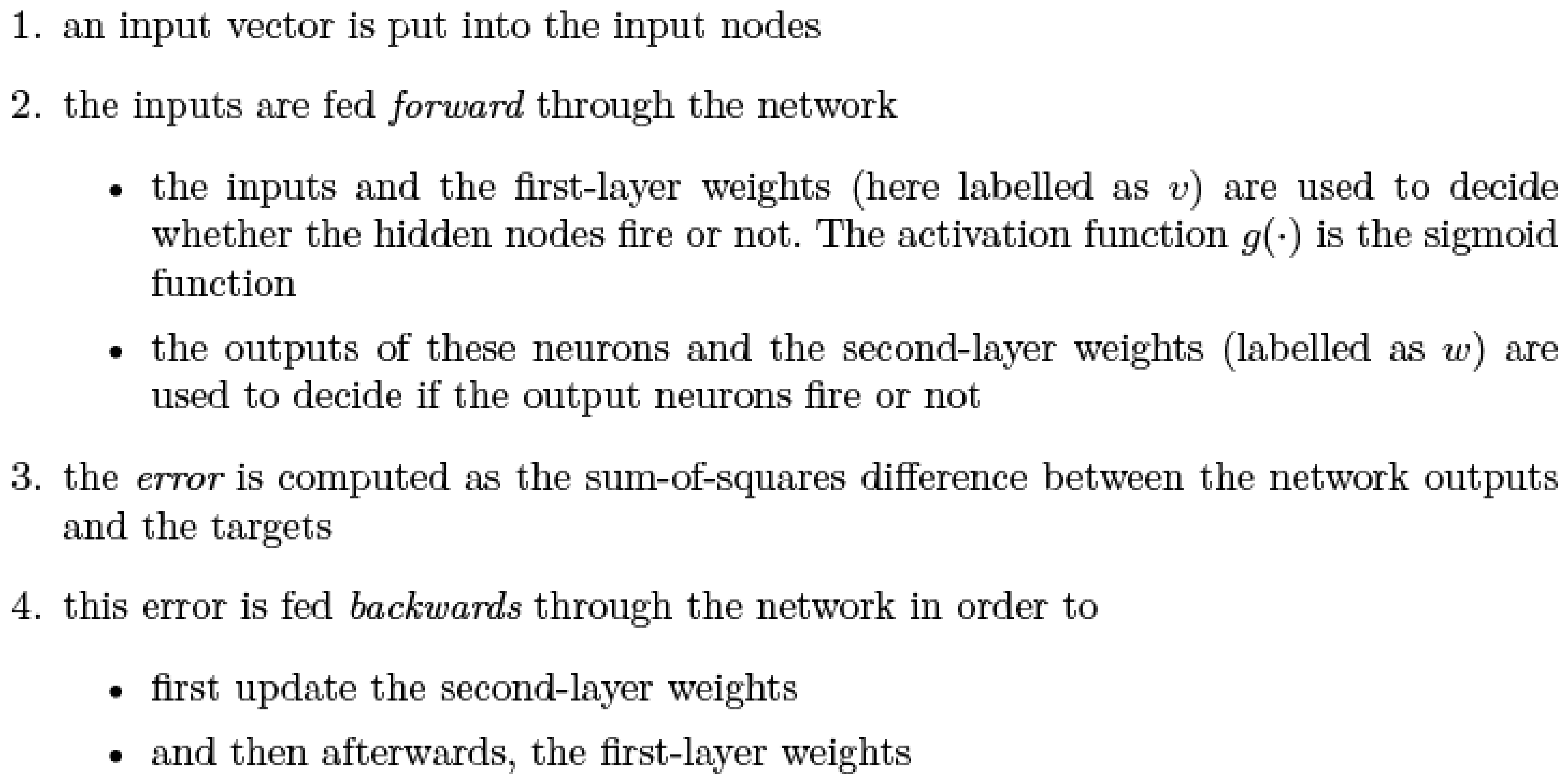
-working out what the outputs are for the given inputs and the current weights

-updating the weights according to the error, which is a function of the difference between the outputs and the targets



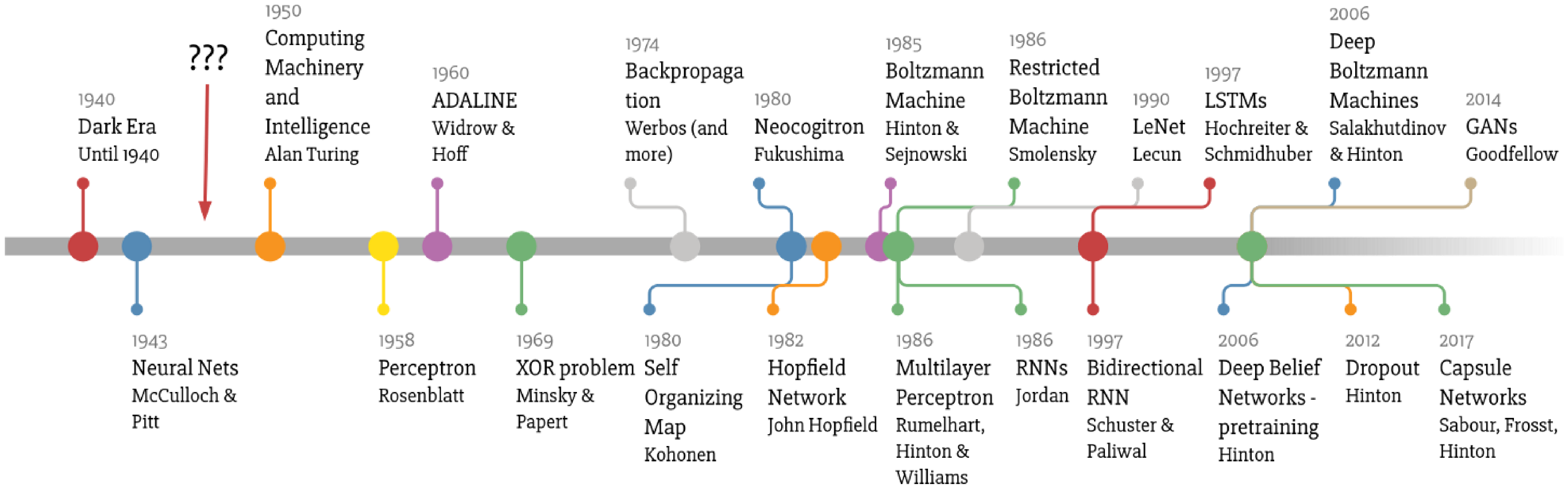
A Multi-layer Perceptron network showing a set of weights that solve the XOR problem.

The Multi-layer Perceptron Algorithm



1. an input vector is put into the input nodes
2. the inputs are fed *forward* through the network
   - the inputs and the first-layer weights (here labelled as v) are used to decide whether the hidden nodes fire or not. The activation function $g(\cdot)$ is the sigmoid function
   - the outputs of these neurons and the second-layer weights (labelled as w) are used to decide if the output neurons fire or not
3. the *error* is computed as the sum-of-squares difference between the network outputs and the targets
4. this error is fed *backwards* through the network in order to
   - first update the second-layer weights
   - and then afterwards, the first-layer weights

Deep Learning Timeline



Made by Favio Vázquez