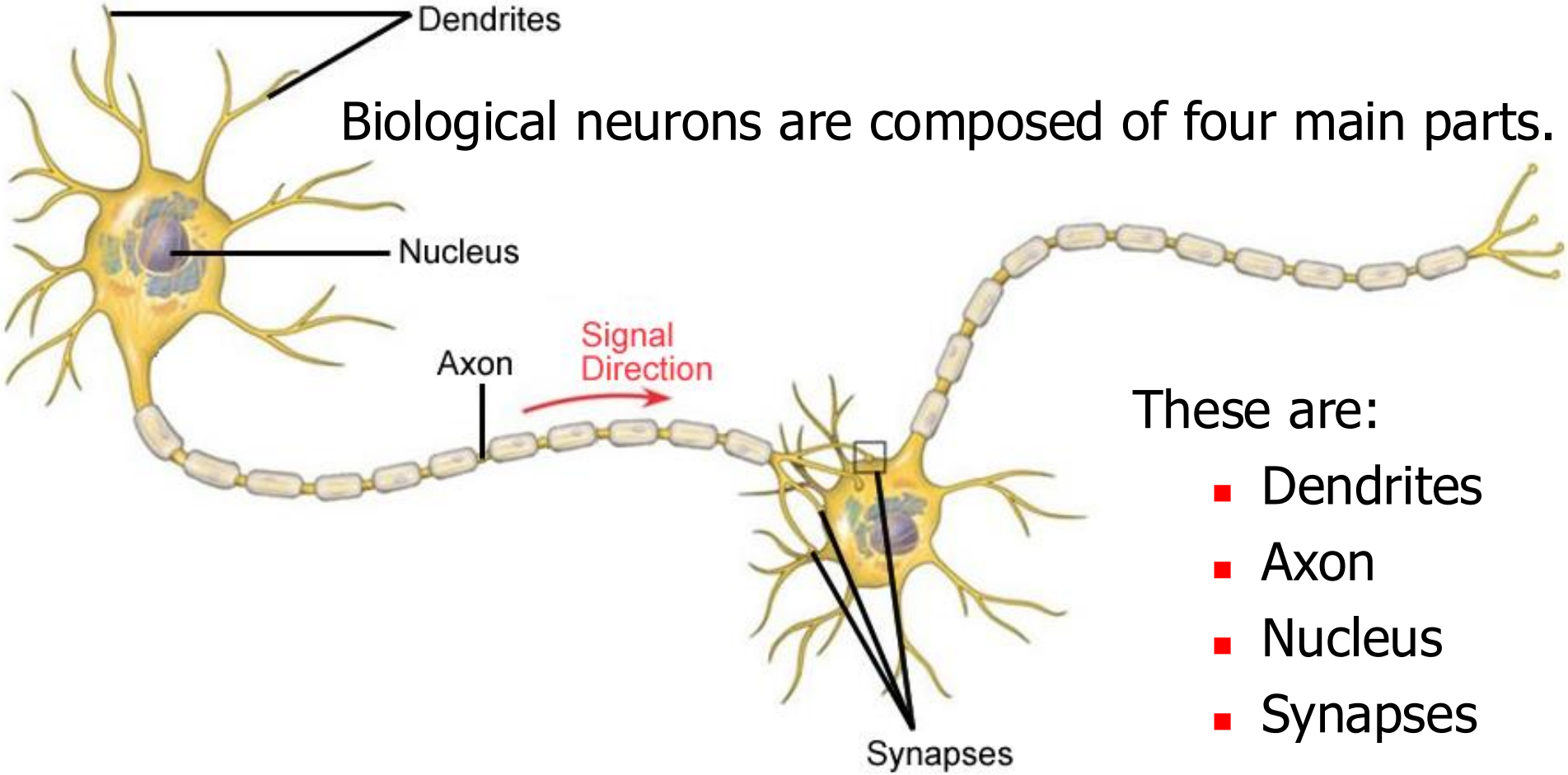
Machine Learning

11. week

- Introduction to Artificial Neural Networks
- Single Layer Artificial Neuron
  - Perceptron
  - AdaLinE (Adaptive Linear Element)

Biological Neuron Cell

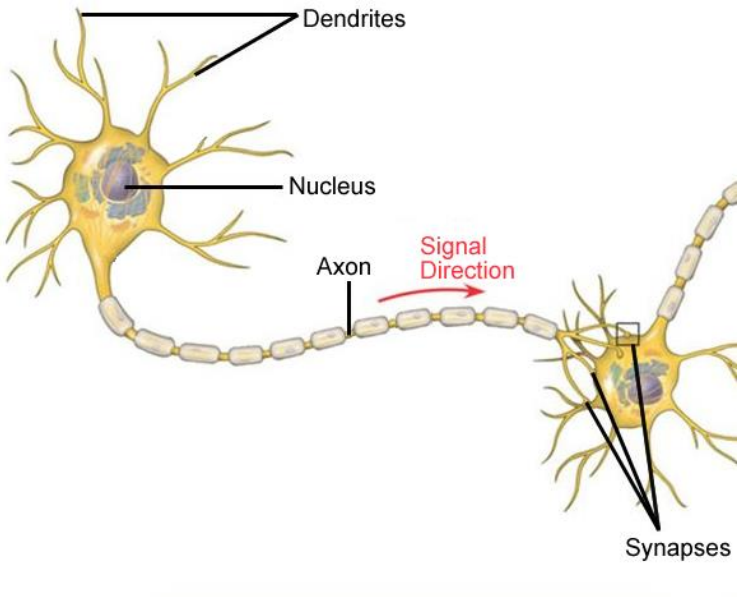


Biological neurons are composed of four main parts.

These are:

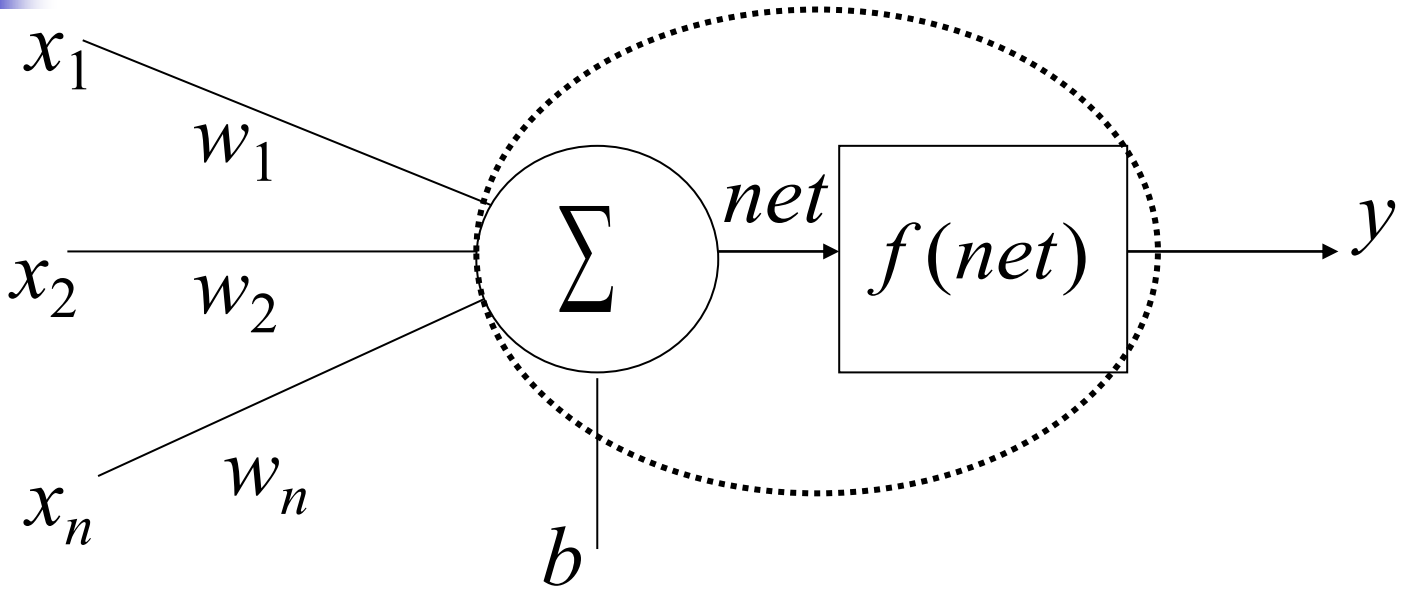
- Dendrites
- Axon
- Nucleus
- Synapses

Biological Neuron Cell



- Dendrites: Sense (as sensors) and then send these signals to the Nucleus.
- Nucleus: According to all coming signals (from Dendrites and Synapses), makes a decision.
- Axon: Carries the produced decision to neighbor neuron.
- Synapses: Transmit signals at the Axon to Nucleus of next neuron.

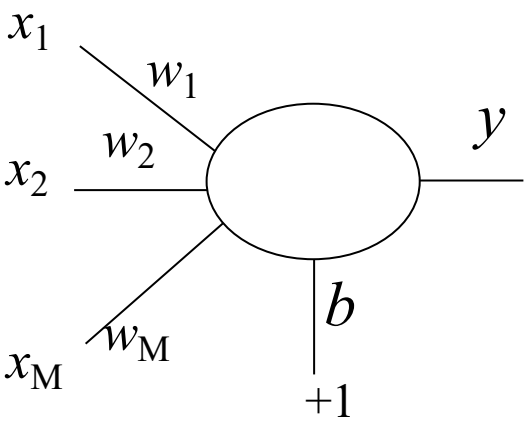
Artificial Neuron Model



$$net = b + \sum_{i=1}^{M} x_i w_i \qquad y = f(net)$$

Artificial Neuron Model

- A neuron should have at least two inputs and an output.
- Each input (x_i) has a weight. The neuron at first computes weighted summations of inputs. A bias (b) is added this summation.
- The output (y) can be computed by giving summation above into an activation function (f).



$$y = f\left(b + \sum_i w_i x_i\right)$$

Activation Function

In biological neurons, if summation (net) of values from dendrites exceeds a threshold, a pulse signal is sent to output (axon). Otherwise, output stays passive. Here, activation is mathematically represented by 1 value.

$$net = b + \sum_i w_i x_i \qquad f(net) = \begin{cases} 1 & net > 0 \\ 0 & - \end{cases}$$

Activation Function

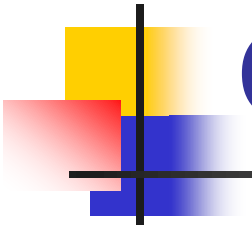
In single layer artificial neural models such as perceptron and adaline, even a nonlinear equation is chosen as activation function, these kind of models cannot solve complex nonlinear problems. Then we will see multilayered neural models which are used perceptron and adaline in a complex architecture.

$$net = b + \sum_i w_i x_i \qquad f(net) = \begin{cases} 1 & net > 0 \\ 0 & - \end{cases}$$

Learning Rule

But here, we should learn learning rules of both perceptron and adaline at first. In 1958, Perceptron rule is proposed by Rosenblatt, and then in 1960, Adaline (Delta or LMS) rule is suggested by Widrow and Hoff. Both methods are similar to each other. Each weight of input is updated slowly and step by step.

$$w_i(t+1) = w_i(t) + \Delta w_i(t),$$



General Algorithm

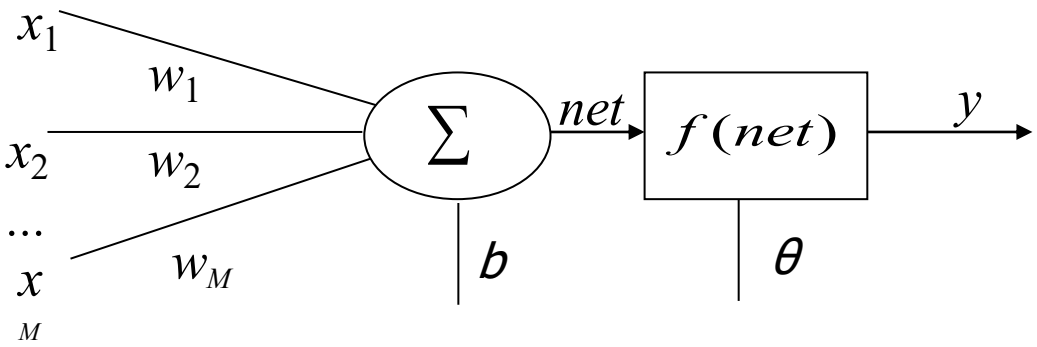
We can summarize both methods as a common algorithm.

1. Assign a random value to weights.
2. For each sample in train set, compute output.
3. If the output is different from desired value, update weights according to learning rule.
4. Go to step 2.

Perceptron

In a traditional activation function, a special threshold value is used (θ).

$$f(net) = \begin{cases} 1 & net > \theta \\ 0 & - \end{cases}$$



Perceptron

If error at the output is 1, $\Delta w_i(t)$ values are computed as

$$e(t) = d(t) - y(t) = \{-1, 0, 1\}$$

$$\Delta w_i(t) = \eta\, e(t) x_i(t)$$

$$\Delta b(t) = \eta\, e(t)$$

where η is learning rate which is chosen between [0 1].

Sample

According to X and D, update weights for one iteration:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Let initial values be as

$$w = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad b = 0 \qquad \theta = -1 \qquad \eta = 0.5$$

Sample

At first, we should compute output for the first sample.

$$net_1 = b + x_{1,1}w_1 + x_{1,2}w_2$$
$$= 0 + 1*1 + 0*2 = 1$$
$$y_1 = f(1) = 1 \quad e = 0$$

Because the output is correct, weights does not be updated.

$$x_1 = [1 \quad 0]$$
$$x_2 = [0 \quad 1]$$
$$d_1 = 1$$
$$d_2 = 0$$
$$w_1 = 1$$
$$w_2 = 2$$
$$b = 0$$
$$\theta = -1$$
$$\eta = 0.5$$

Sample

Then, for the second sample,

$$net_2 = b + x_{2,1}w_1 + x_{2,2}w_2$$
$$= 0 + 0*1 + 1*2 = 2$$
$$y_2 = f(2) = 1 \quad e = -1$$

Because of error, weights should be updated.

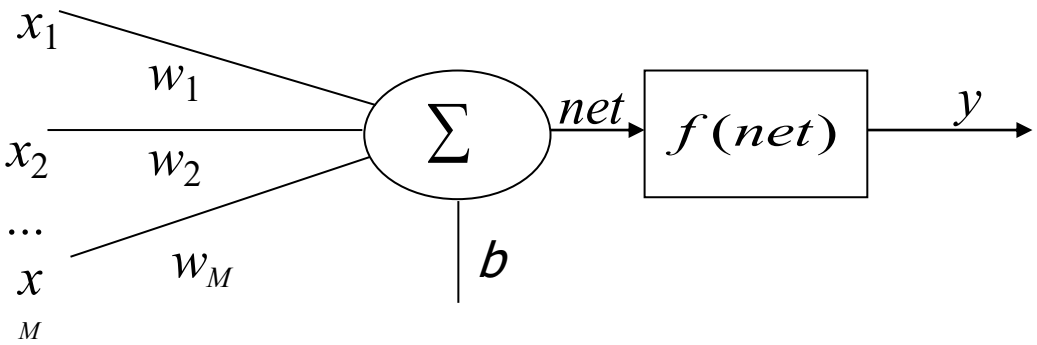
$$b = 0 - 0.5 = -0.5$$
$$w_1 = 1 - 0.5*0 = 1$$
$$w_2 = 2 - 0.5*1 = 1.5$$

$$x_1 = [1 \quad 0]$$
$$x_2 = [0 \quad 1]$$
$$d_1 = 1$$
$$d_2 = 0$$
$$w_1 = 1$$
$$w_2 = 1.5$$
$$b = -0.5$$
$$\theta = -1$$
$$\eta = 0.5$$

AdaLinE (Adaptive Linear Element)

Unlike Perceptron, AdaLinE uses zero (0) value for threshold.

$$f(net) = \begin{cases} 1 & net > 0 \\ 0 & - \end{cases}$$



AdaLinE (Adaptive Linear Element)

Like linear regression, we will use least squares method again. To find update amount in weights, we should find place with zero gradient between mean squared error and weight.

$$w_i(t+1) = w_i(t) - \frac{1}{2}\eta\left[\nabla\left(E\left\{e^2(t)\right\}\right)\right]$$

where $\nabla\left(E\left\{e^2(t)\right\}\right)$ term is gradient vector.

AdaLinE (Adaptive Linear Element)

Gradient vector can be computed as

$$\nabla\left(E\left\{e^2(t)\right\}\right) = -2x_i(t)e(t)$$

And, if we write its place in the previous equation

$$w_i(t+1) = w_i(t) + \eta x_i(t)e(t)$$

AdaLinE (Adaptive Linear Element)

Update equation for the weights is the same with that in Perceptron. Only difference is in computing the error.

$$e_{perceptron}(t) = d(t) - f\left(\sum x_i w_i + b\right)$$

$$e_{adaline}(t) = d(t) - \sum x_i w_i + b$$

Sample

We can repeat the second iteration of the previous sample.

$$net_2 = 0 + 0*1 + 1*2 = 2$$

$$y_2 = 1 \quad e = 0 - 2 = -2$$

Because of error, weights should be updated.

$$b = 0 - 2*0.5 = -1$$

$$w_1 = 1 - 2*0.5*0 = 1$$

$$w_2 = 2 - 2*0.5*1 = 1$$

$$x_1 = [1 \quad 0]$$

$$x_2 = [0 \quad 1]$$

$$d_1 = 1$$

$$d_2 = 0$$

$$w_1 = 1$$

$$w_2 = 1$$

$$b = -1$$

$$\eta = 0.5$$