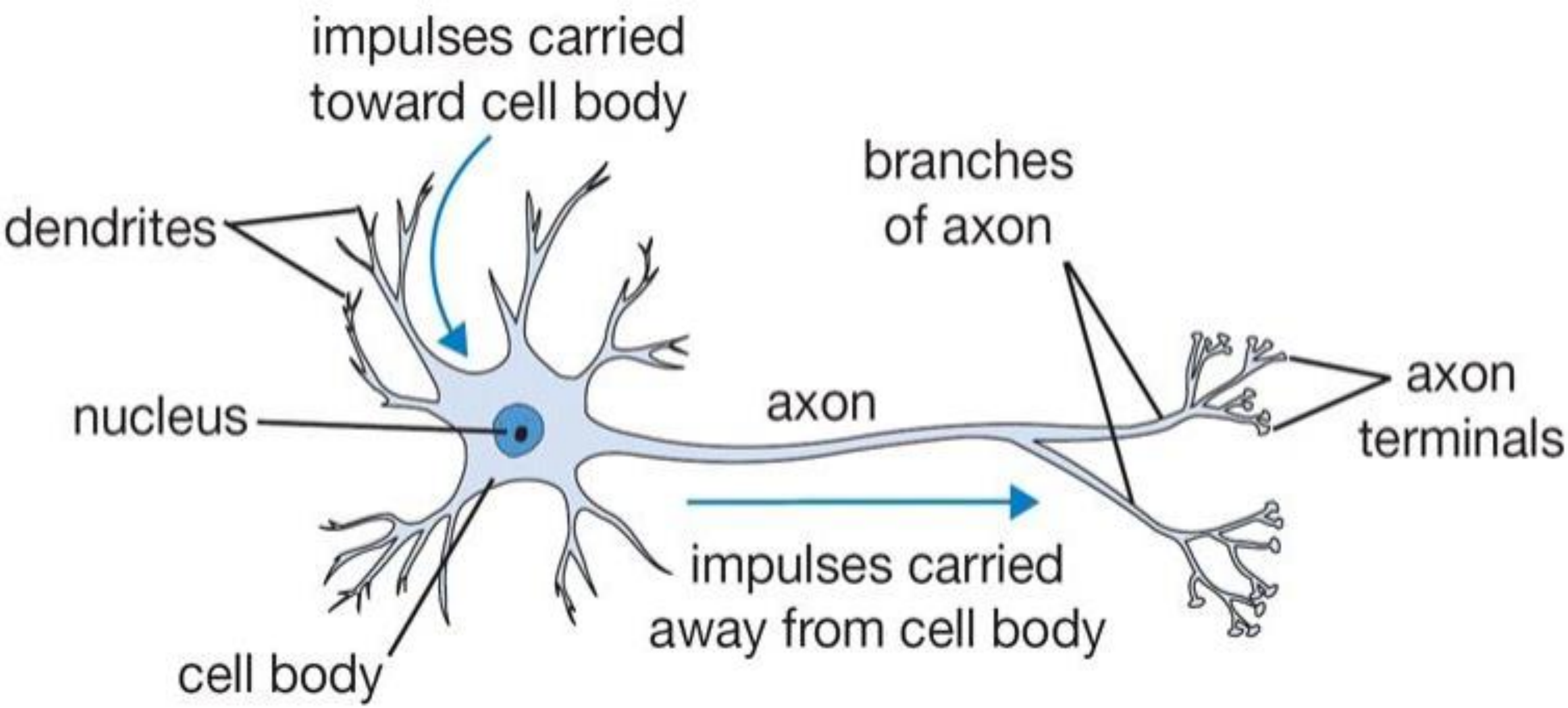
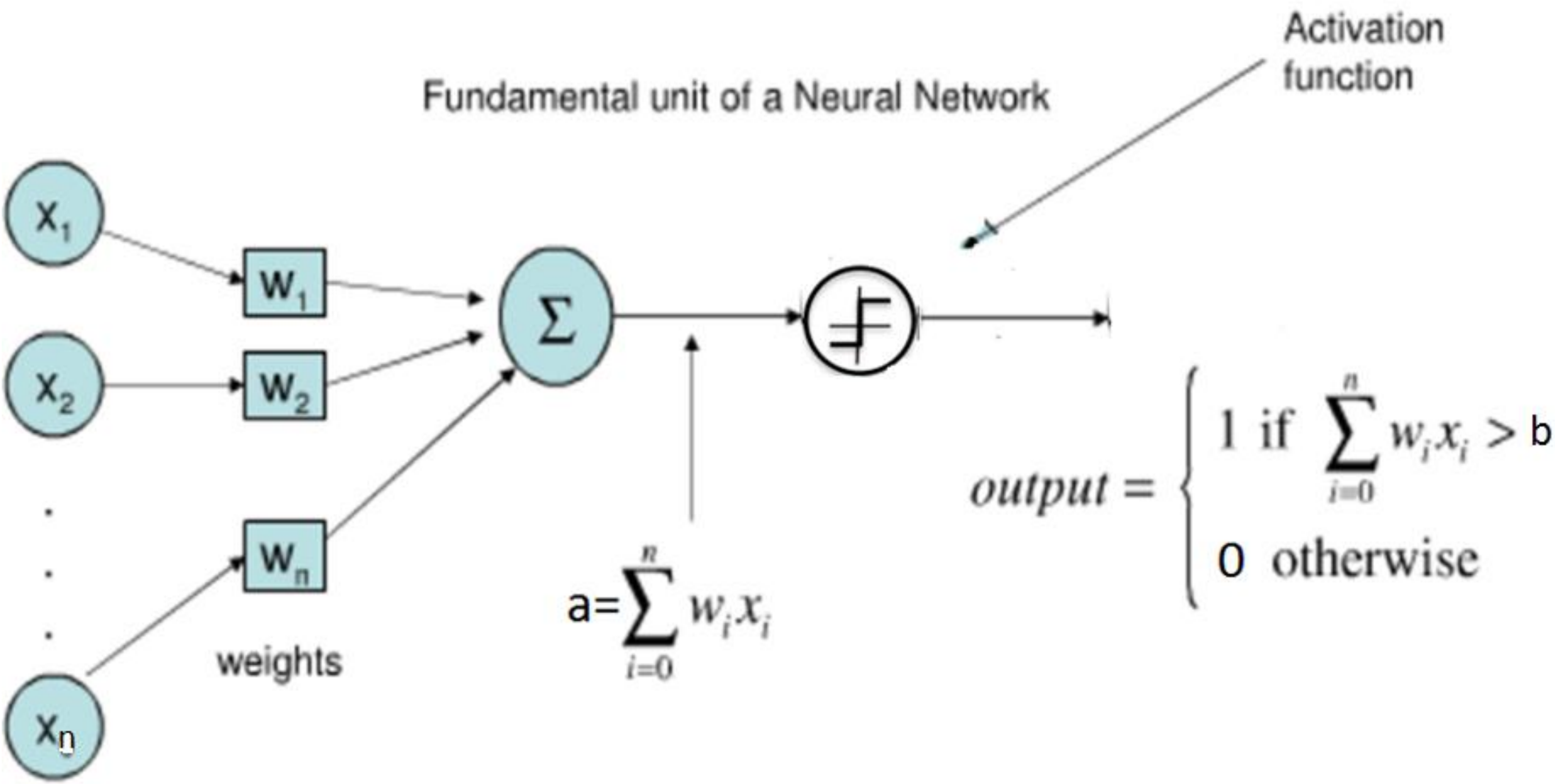
Neuron

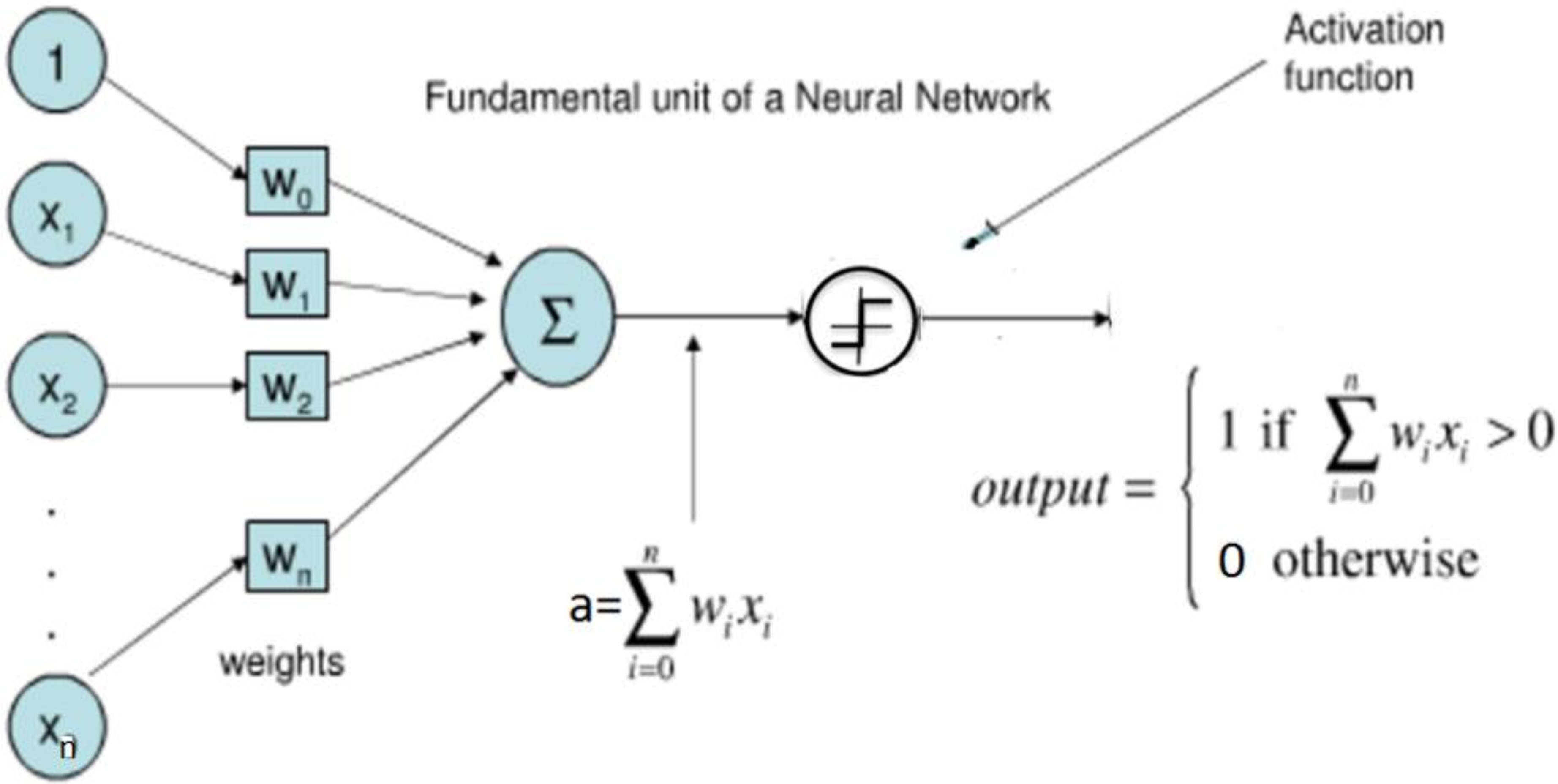
http://neuroscience.uth.tmc.edu/s1/chapter01.html



Perceptron: model of real neuron



Perceptron: simplified



Meaning of firing

- Each incoming connection has a weight and the neuron simply sums up all the weighted inputs
- Based on this sum, it decides whether to “fire” or not. If the weighted sum(evidence) is positive, it “fires” and otherwise it doesn’t fire
- Impact of weights
  - Features with positive weights will cause the evidence to increase
  - Features with negative weights will cause the evidence to decrease
  - Features with zero weights are ignored, because the evidence is the same regardless of the value of this feature

It’s only an Analogy

- Many different types of neurons
- The computations are not simply linear combinations of inputs transformed by the same activation functions
- Synapses are more complicated than a single weight
- The neurons don’t output a real number: instead, they “fire” spikes at a (somewhat) regular rate

Perceptron Learning

Learning in a Perceptron

Learning some pattern in a perceptron is equal to finding the right weight adjustments of a perceptron that minimize the objective function. The weights represent the learning in a perceptron.

Perceptron Learning Rule

- Assume some random weights and random bias for perceptron

- Do until all the samples are classified correctly or maximum number of epochs reached:

  For each training sample $(x^{(n)}, y^{(n)})$

  a. Compute the output of perceptron, o

  b. Update each weight, w_j, of the perceptron as follows:

$$w_j = w_j + \eta \left(y^{(n)} - o^{(n)}\right) x_j^{(n)}$$

Note: keep $x_j^{(n)} = 1$ for w_o (bias input)

Perceptron Learning Rule

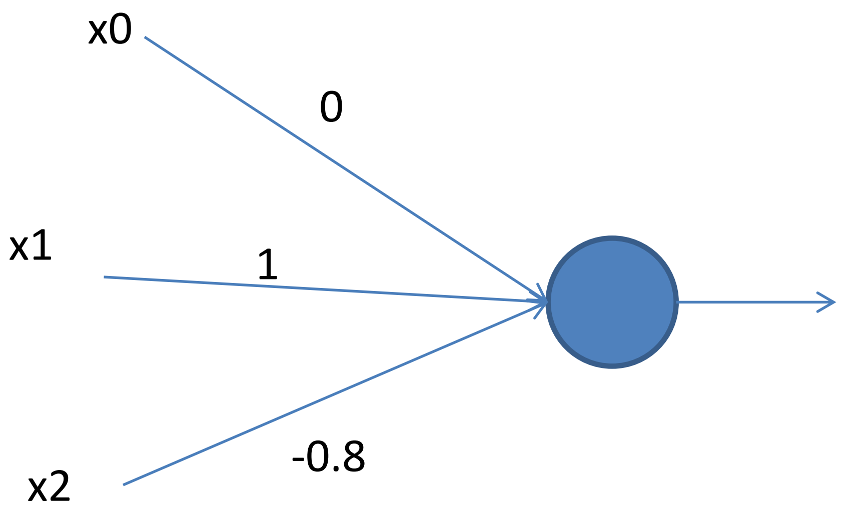
- In the two scenarios where the perceptron predicts the class label correctly, the weights remain unchanged

- false positive(target is -1 but predicted as +1):
  - a > 0 but we want a to be negative to get correct prediction
  - So, we need to reduce the weights of positive inputs and increase the weights of negative inputs

Perceptron Learning Rule

- false negative(target is +1 but predicted as -1):
  - a < 0 but we want a to be positive to get correct prediction
  - So, we need to increase the weights of positive inputs and decrease the weights of negative inputs

Numerical Example(linear separable)



x1	x2	Output
1	2	1
-1	2	0
0	-1	0

Issues with Thresholded Perceptron

- It will converge only if data is linearly separable

- It stops the learning immediately if all the samples are classified correctly during learning. The linear separator may not have best generalization capability.

Objective based Perceptrons

Objective based perceptron learning

- To make the learning of decision boundary better in both linear and non-linear data, we can make perceptron learning based on objective function.

- We use squared loss as objective function for perceptron learning:

$$E = \frac{1}{2}\sum_{n=1}^{N}\left(y^{(n)} - o^{(n)}\right)^{2}$$

Objective based perceptron learning

- We use squared loss as objective function for perceptron learning:

Represent $w_0 x_0 + w_1 x_1$ in vector notation as wx

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\left(y^{(n)} - f\left(wx^{(n)}\right)\right)^2$$

- Use gradient descent algorithm to compute weight vector w. To compute the gradient of E wrt weights(EW), the function f must be differentiable.

$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

Perceptron Learning - BGD

- Assume some random weights and random bias for perceptron

- Repeat the following until the error is below threshold or maximum number of epochs reached:

  a. Pick a training smaple($x^{(n)}$, $y^{(n)}$) and compute the output of perceptron, $o^{(n)}$

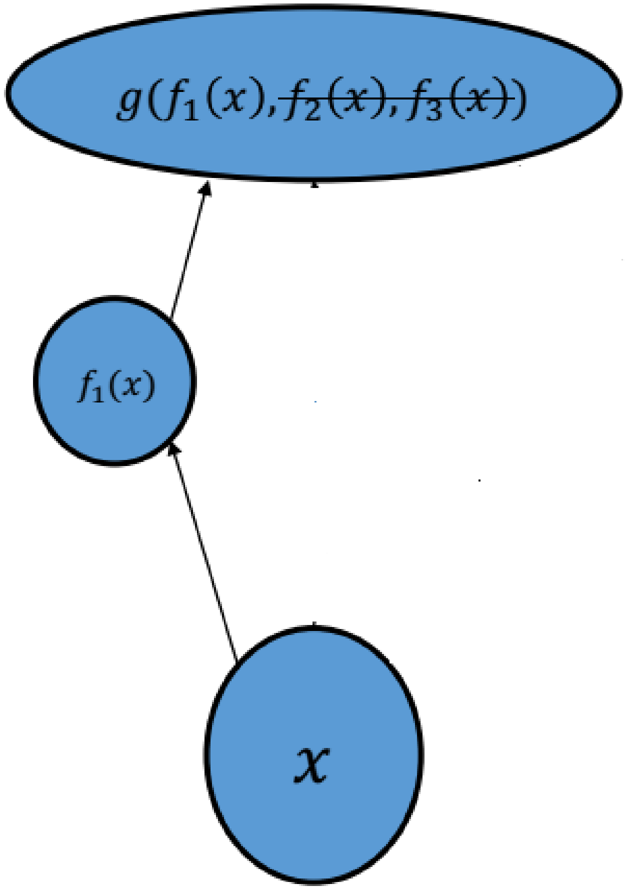
  b. Update each weight, w_j, of the perceptron as follows:

$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

$$w_j = w_j + \eta \sum_{n=1}^{N} \left(y^{(n)} - o^{(n)}\right) x_j^{(n)} g'(a)^{(n)}$$

Note: keep $x_j^{(n)}$ = 1 for w_o (bias input)

Computing EW: Chain Rule



$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f_1}\frac{\partial f_1}{\partial x}$$

Error derivatives of weights

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial o} \quad \frac{\partial o}{\partial a} \quad \frac{\partial a}{\partial w_j}$$

$$\frac{\partial a}{\partial w_j} = \frac{\partial \; \sum_k w_k x_k}{\partial w_j} = x_j$$

$$\frac{\partial o}{\partial a} = g'(a)$$

Error derivatives of weights

$$\frac{\partial E}{\partial o} = \frac{1}{2}\frac{\partial}{\partial o^{(n)}}(y - o)^2$$

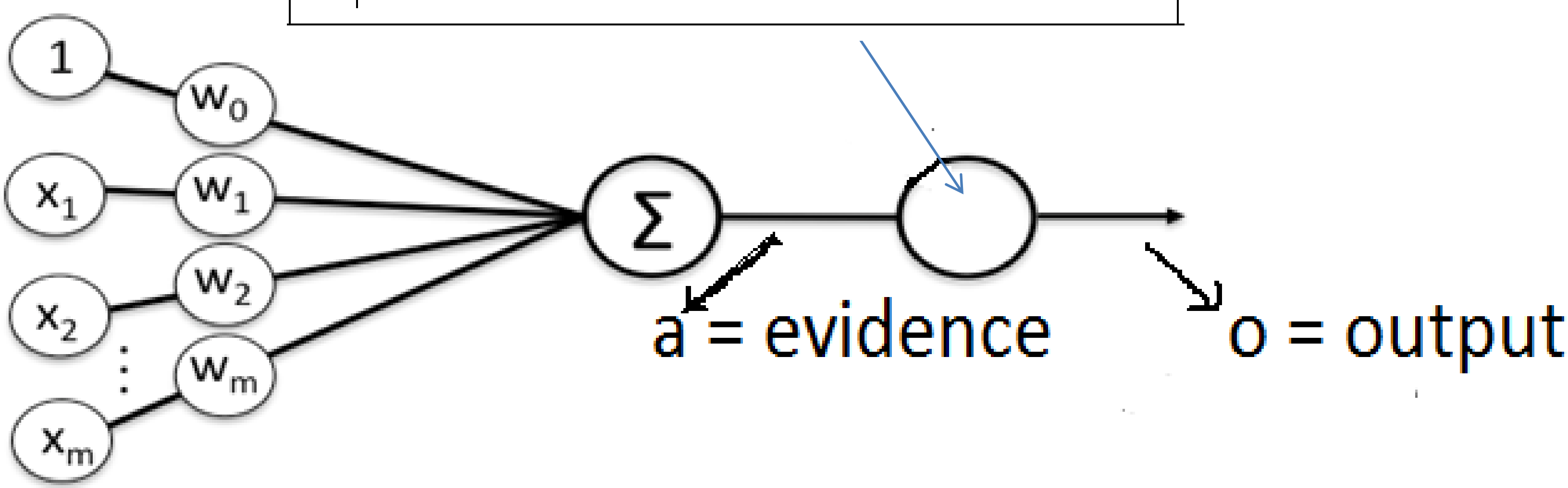
$$\frac{\partial E}{\partial o} = \frac{1}{2}2 * (y - o) * \frac{\partial}{\partial o^{(n)}}(y - o)$$

$$\frac{\partial E}{\partial o} = (y - o) * -1$$

Understanding activation functions

Common Activation Functions

Unit step	$g(a) = \begin{cases} 1 \text{ if } a \geq 0 \\ -1 \text{ otherwise.} \end{cases}$
	$g(a) = \begin{cases} 1 \text{ if } a \geq 0 \\ 0 \text{ otherwise.} \end{cases}$
Linear	$g(a) = a$
Logistic (sigmoid)	$g(a) = 1 / (1 + \exp(-a))$
Hyperbolic tangent	$g(a) = \frac{\exp(2a) - 1}{\exp(2a) + 1}$
ReLU	$g(a) = \max(0, a)$



Two perspectives of activation functions

- Model perspective: The interpretation of activation function curves based on - evidence vs outcome

- Learning perspective: Some activation functions provide much smoother learning compared to others

Issues of objective perceptron

Issues of Perceptron-I

- Computing Gradient takes longer time: If the dataset is very large then It can take several hours to compute a single gradient of the error over dataset.

  Solution: Use the stochastic approximation of the gradient using a single sample or a group of samples. It has faster convergence rate and having chance of avoiding local minima.

Perceptron Learning with SqError(SGD)

- Assume some random weights and random bias for perceptron

- Repeat the following until the error is below threshold or maximum number of epochs reached:

  Shuffle the train data and repeat the following until the end of epoch

  a. Pick a training smaple($x^{(n)}$, $y^{(n)}$) and compute the output of perceptron, $o^{(n)}$

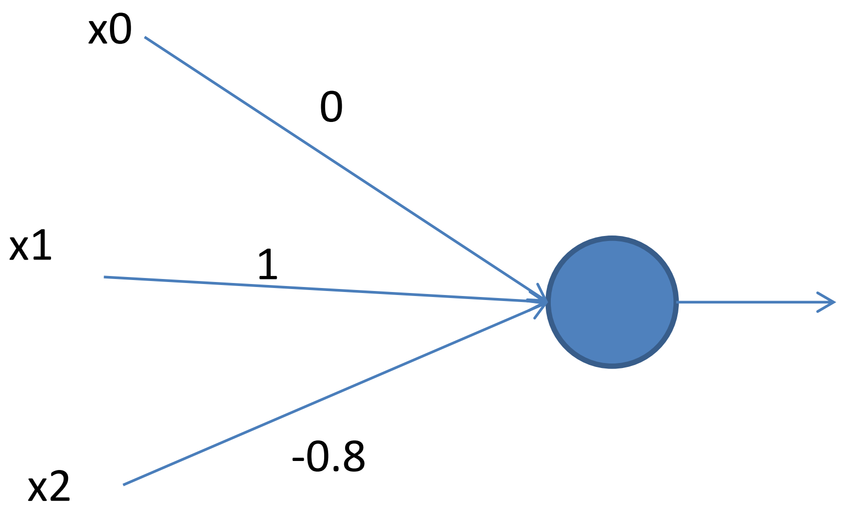
  b. Update each weight,w_j,of the perceptron as follows:

$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

$$w_j = w_j + \eta \left(y^{(n)} - o^{(n)}\right) x_j^{(n)} \, g'(a)^{(n)}$$

Note: keep $x_j^{(n)}$ = 1 for w_o (bias input)

Numerical Example



x1	x2	Output
1	2	1
-1	2	0
0	-1	0

Issues of Perceptron-II

- Learning is slower: The perceptron learning is slower due to objective

Solution: The squared error objective may lead to slower learning in case of very bad errors. Use cross-entropy objective function to make faster learning.

When does squared error leads to slower learning?

- What happens if $y = 1$ and $o=0.1$ with initial random weights?

- What happens if $y = 0$ and $o=0.9$ with initial random weights?

- The weight increment will be very small since $o(1-o)$ gives very small multiplier. But in reality, for big errors the improvement is expected to be much higher.

cross-entropy objective function

E = $- \frac{1}{N} \sum_{n=1}^{N} [y^{(n)} \log o^{(n)} + (1 - y^{(n)}) \log(1 - o^{(n)})]$

Intuitive meaning

- Since the output is always between 0 to 1, the above error is always positive
- The cost tends to be close to zero if the output is approaching with actual value

Perceptron Learning with Xentropy(SGD)

- Assume some random weights and random bias for perceptron

- Repeat the following until the error is below threshold or maximum number of epochs reached:

  Shuffle the train data and repeat the following until the end of epoch

  a. Pick a training smaple($x^{(n)}$, $y^{(n)}$) and compute the output of perceptron, $o^{(n)}$

  b. Update each weight,w_j,of the perceptron as follows:

$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

$$w_j = w_j + \eta \left(y^{(n)} - o^{(n)}\right) x_j^{(n)}$$

Note: keep $x_j^{(n)}$ = 1 for w_o (bias input)

Error derivatives of weights

$$\frac{\partial E}{\partial o} = \frac{1}{2}\frac{\partial}{\partial o^{(n)}}(y-o)^2$$

$$\frac{\partial E}{\partial o} = \frac{1}{2}2*(y-o)*\frac{\partial}{\partial o^{(n)}}(y-o)$$

$$\frac{\partial E}{\partial o} = (y-o)*-1$$

Error derivatives of weights

$$\frac{\partial E}{\partial o} = \frac{\partial}{\partial o} y \log o + (1-y)\log(1-o)$$

$$\frac{\partial E}{\partial o} = -\left(\frac{y}{o}\right) - \left(\frac{1-y}{(1-o)}\right)$$

$$\frac{\partial E}{\partial o} = \left(\frac{o-y}{o(1-o)}\right)$$