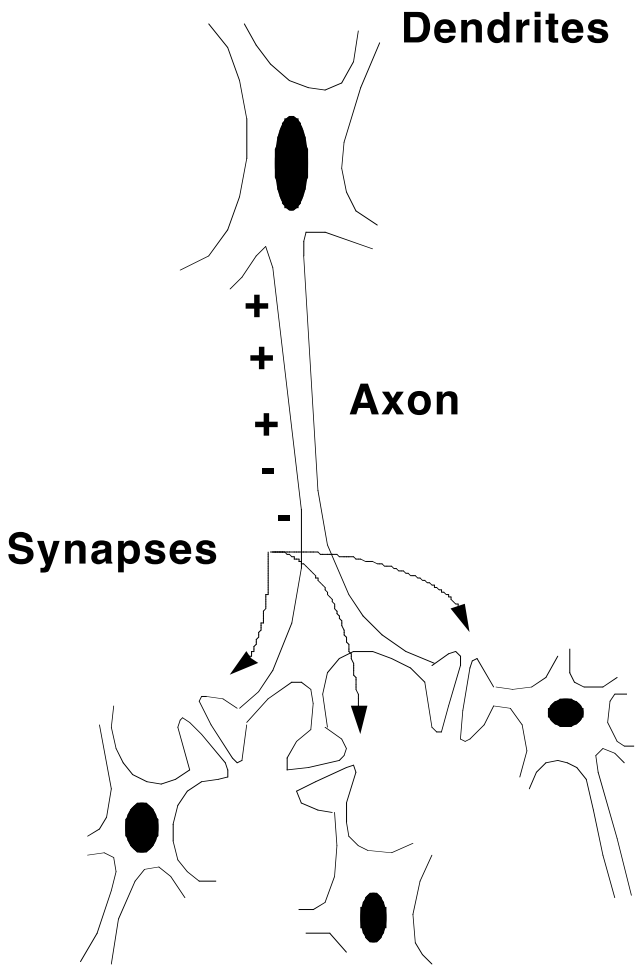
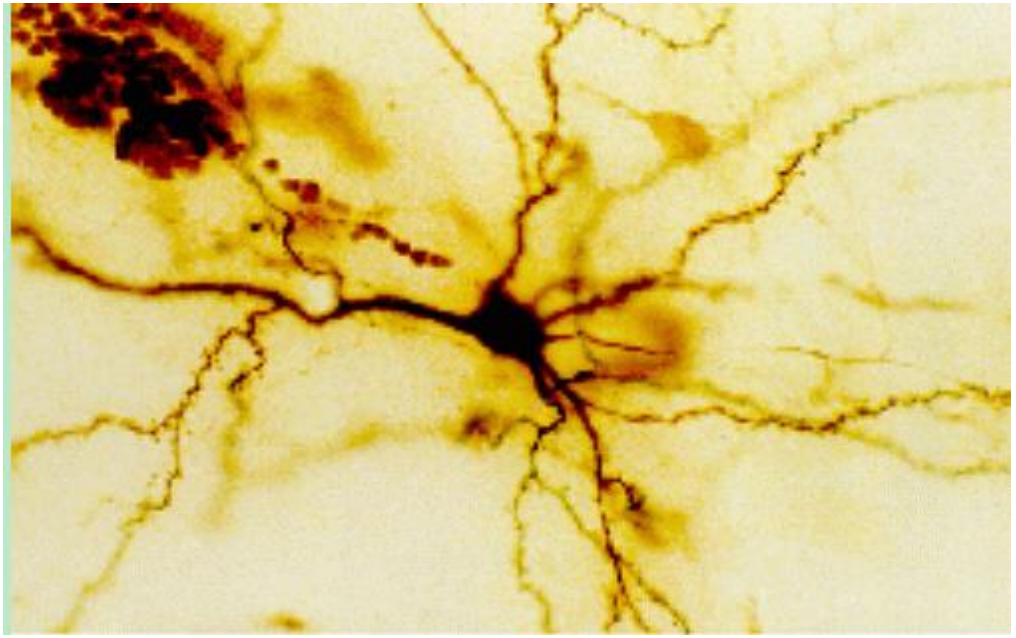
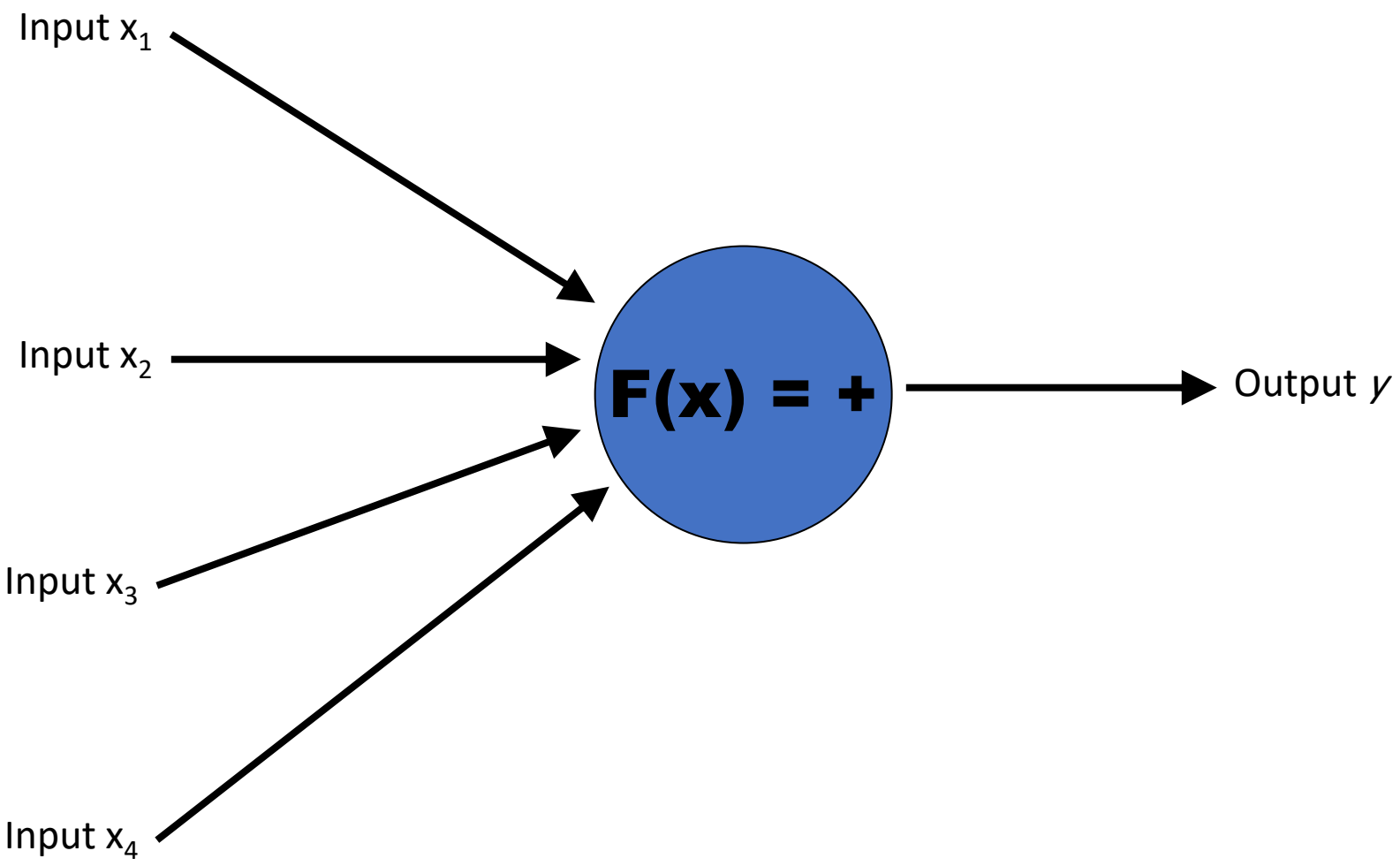
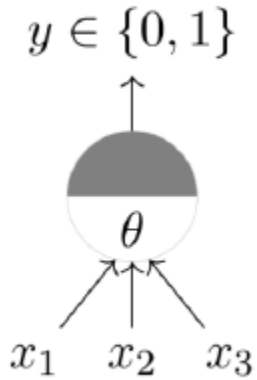
Neural Network

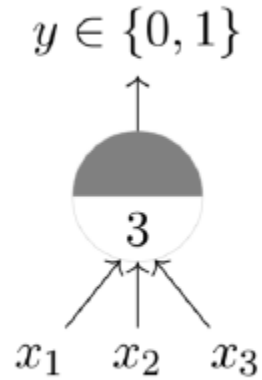

Dendrites
+
+
+
-
-
Axon
Synapses



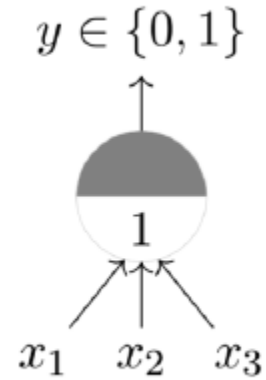

Input x_1
Input x_2
Input x_3
Input x_4
F(x) = +
Output *y*



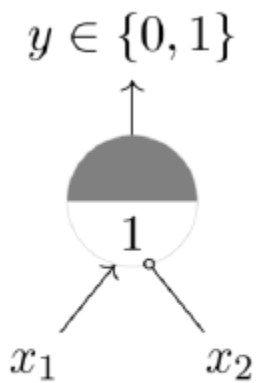
A McCulloch Pitts unit



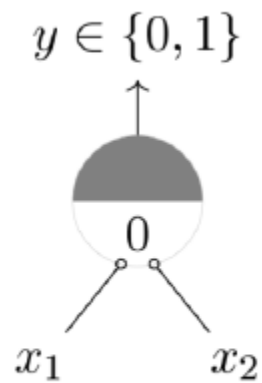
AND function



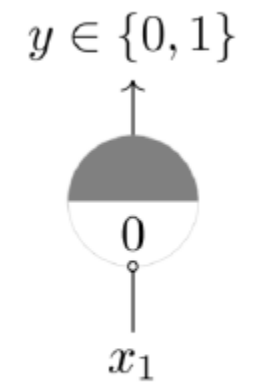
OR function



x_1 AND $!x_2$

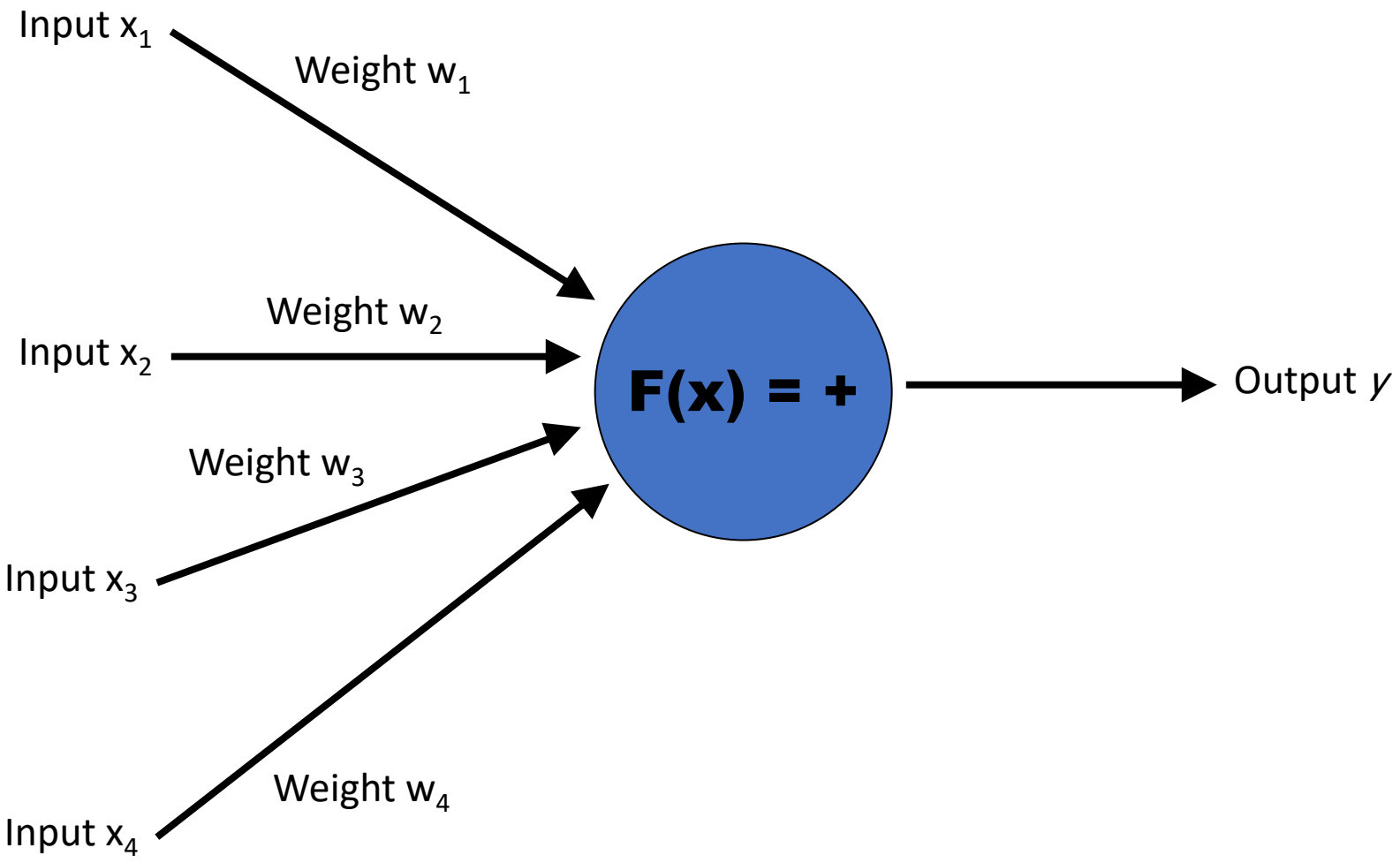


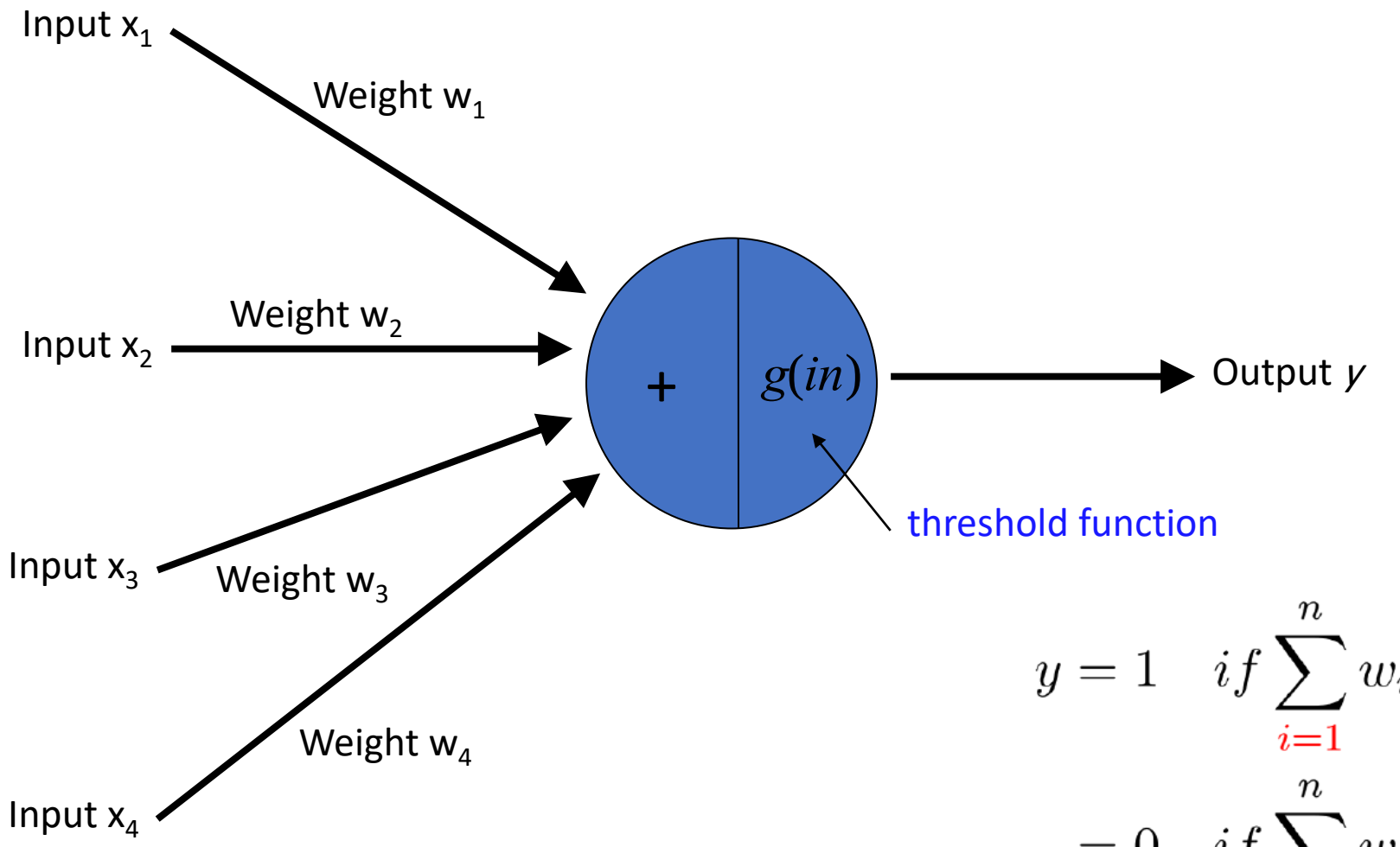
NOR function



NOT function

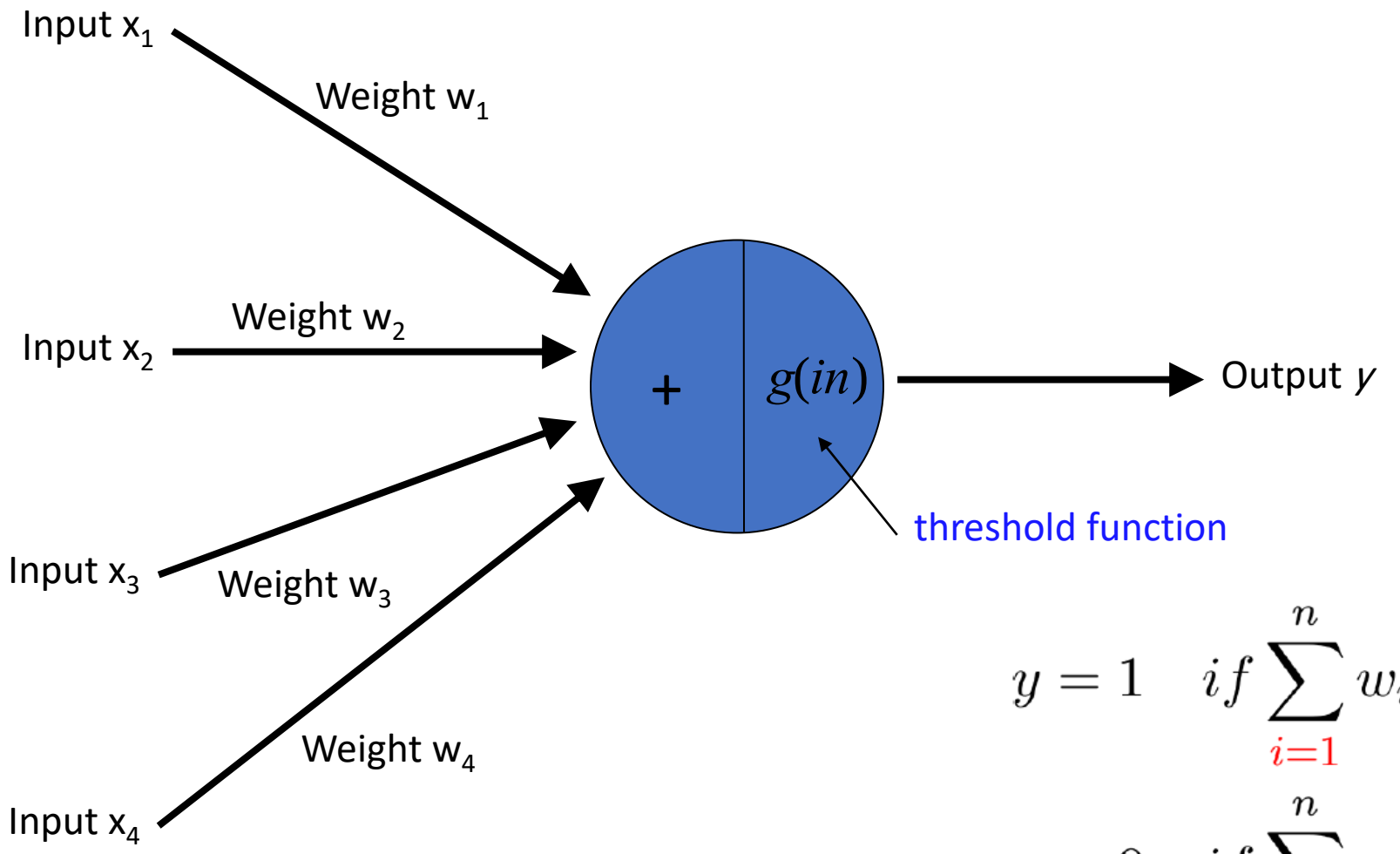
Perceptron


Input x_1
Weight w_1
Weight w_2
Input x_2
F(x) = +
Output *y*
Weight w_3
Input x_3
Weight w_4
Input x_4



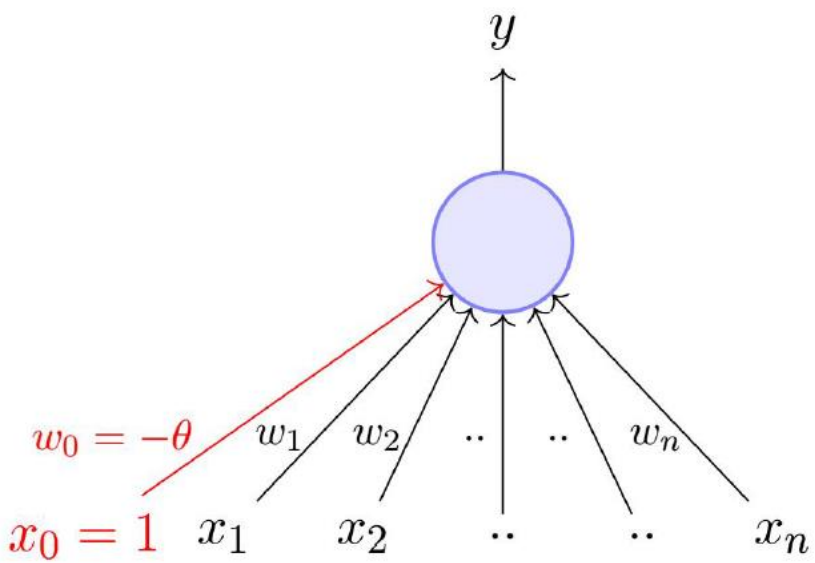
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$



Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$

$N \leftarrow inputs \quad with \quad label \quad 0;$

Initialize **w** randomly;

while !*convergence* **do**

  Pick random $\mathbf{x} \in P \cup N$;

  if $\mathbf{x} \in P \quad and \quad \sum_{i=0}^{n} w_i * x_i < 0$ **then**

    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;

  end

  if $\mathbf{x} \in N \quad and \quad \sum_{i=0}^{n} w_i * x_i \geq 0$ **then**

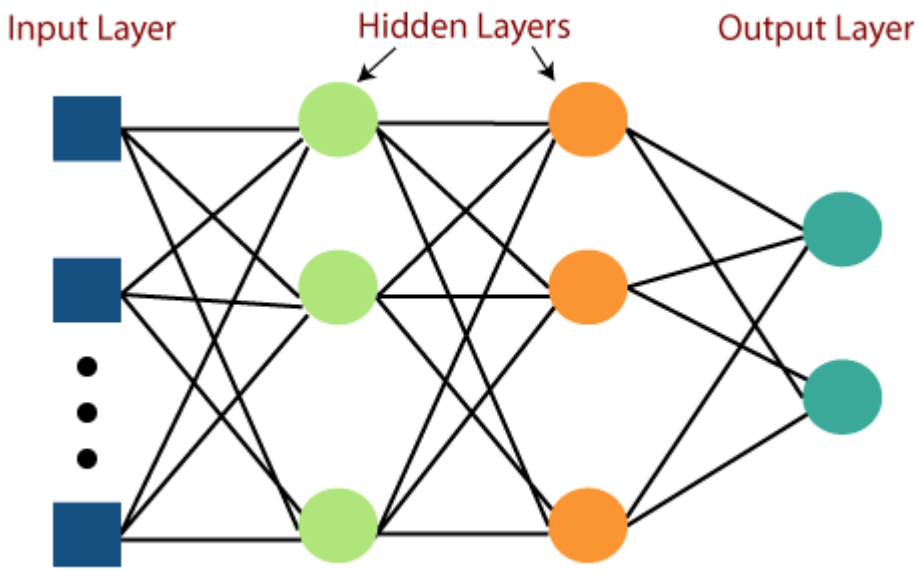
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;

  end

end

//the algorithm converges when all the inputs are classified correctly

Multi-layer Perceptron (MLP)



Activation function

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

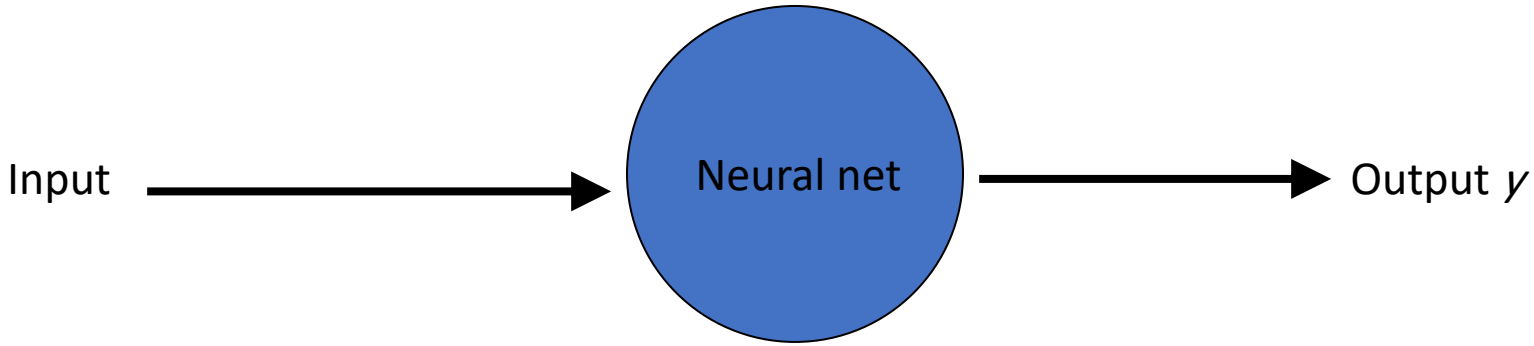
Table 2.1 Transfer Functions

[Which activation function for each network]

[More information]

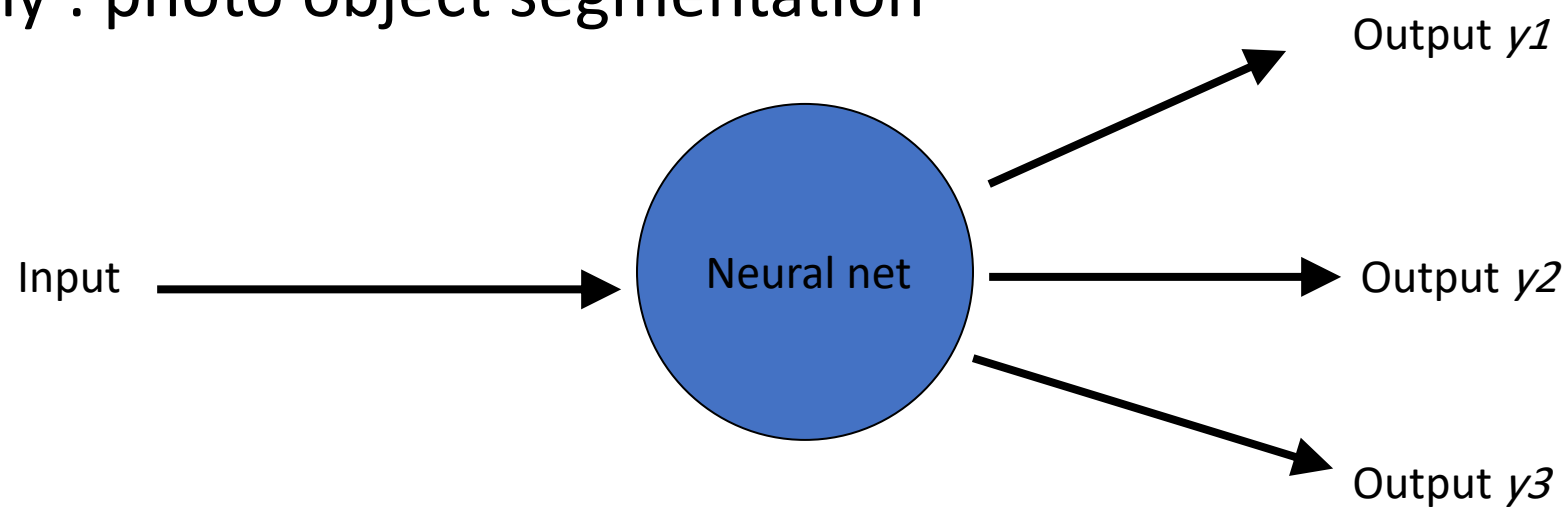
Different input - output

One to one : dog or cat



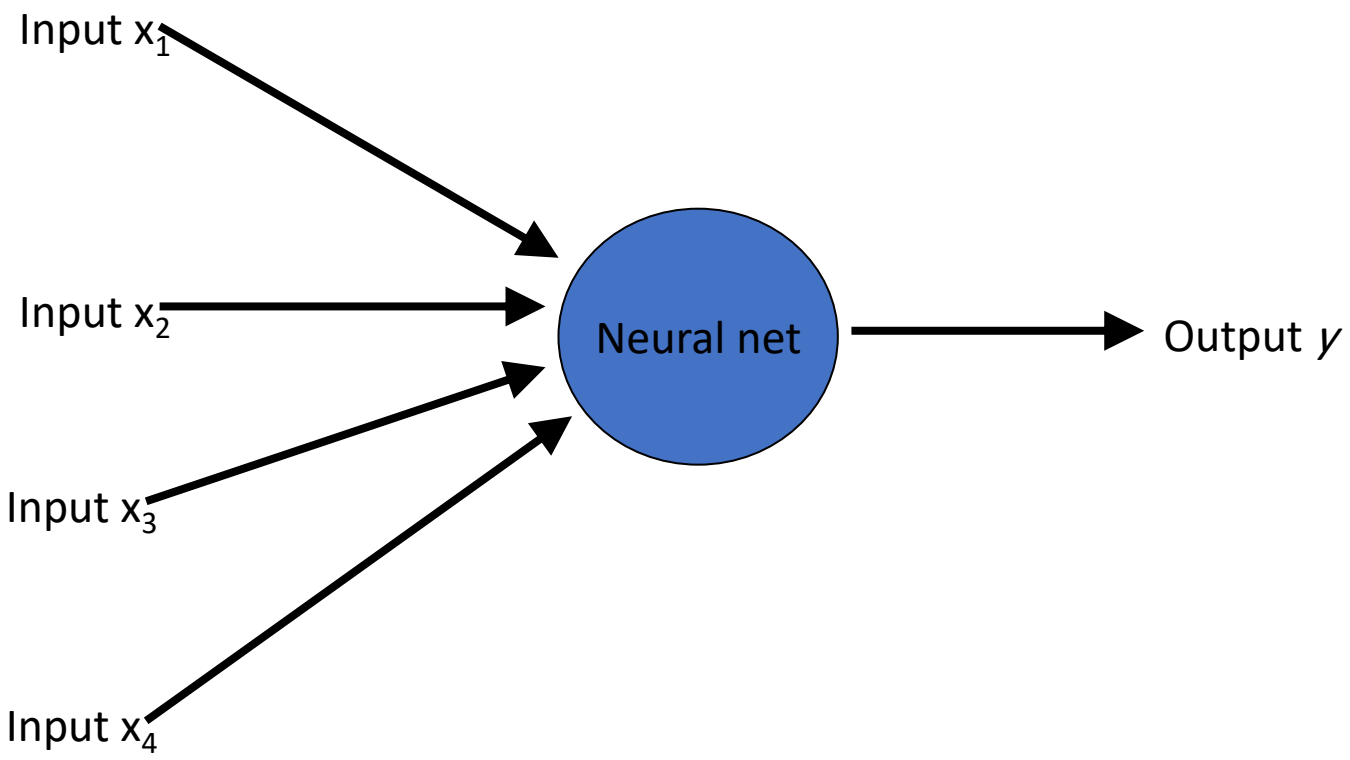
Different input - output

One to many : photo object segmentation



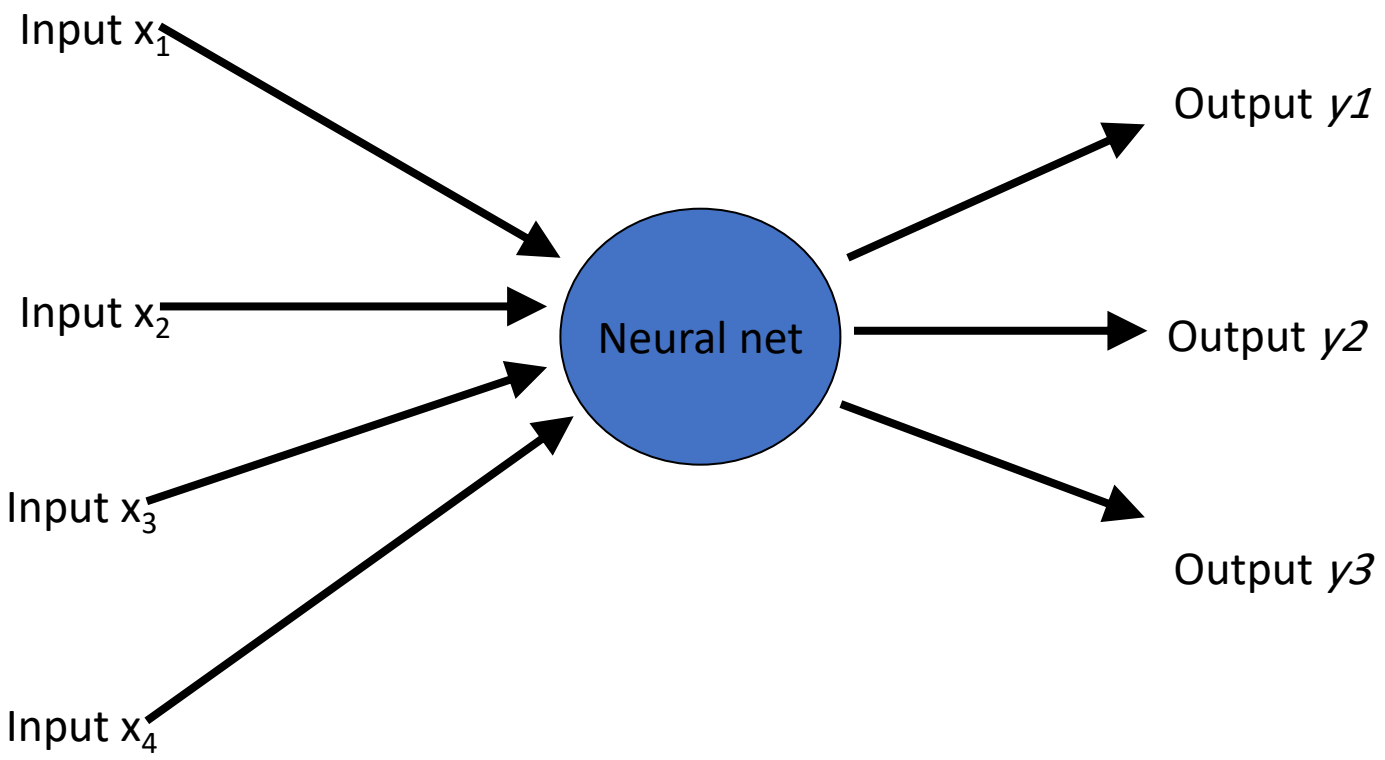
Different input - output

many to one : binary classifications



Different input - output

many to many : multiclass classifications



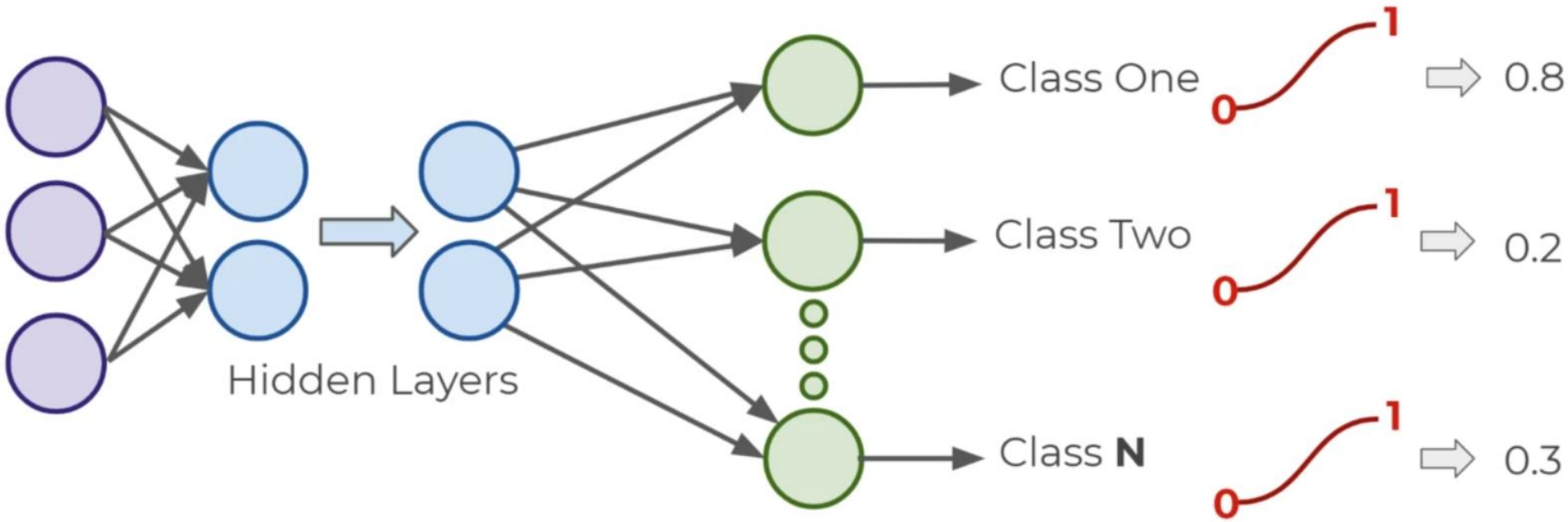
one hot encoding

- Mutually Exclusive Classes

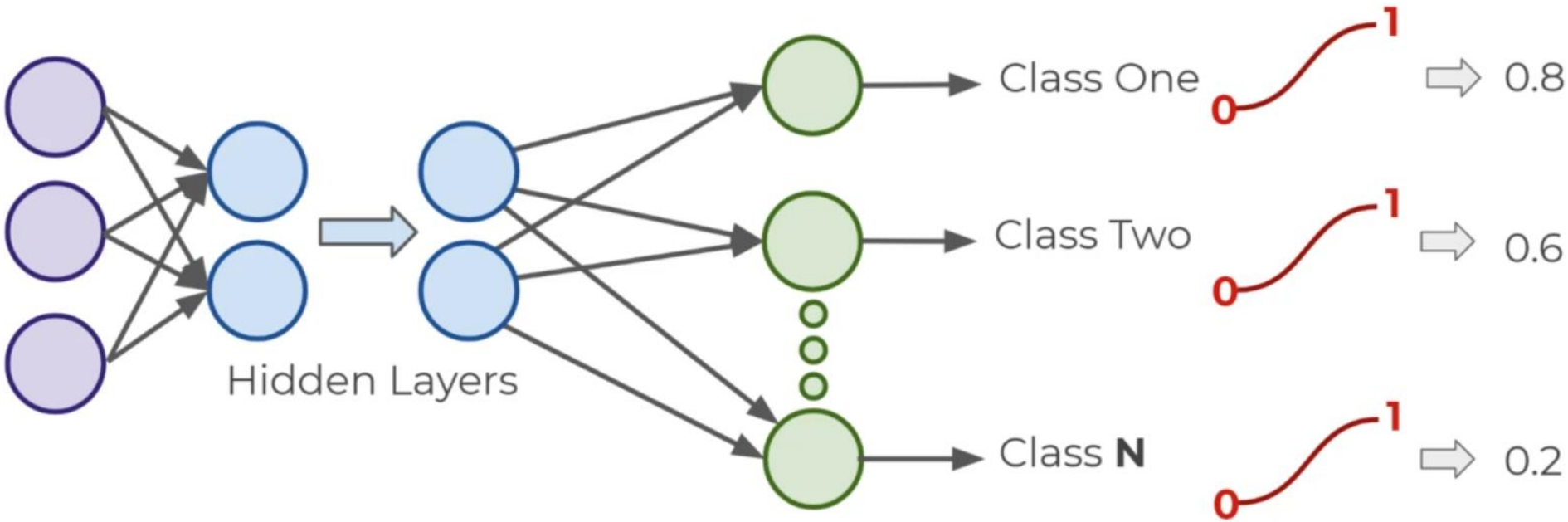
Data Point 1	RED
Data Point 2	GREEN
Data Point 3	BLUE
...	...
Data Point N	RED

	RED	GREEN	BLUE
Data Point 1	1	0	0
Data Point 2	0	1	0
Data Point 3	0	0	1
...	...	...	...
Data Point N	1	0	0

- Sigmoid Function for Non-Exclusive Classes



- Sigmoid Function for Non-Exclusive Classes



Softmax activation function

- این تابع فعالساز در طبقه‌بندی‌های چندکلاسه استفاده می‌شود.
- زمانی که احتیاج داشته باشیم در خروجی احتمال عضویت بیشتر دو کلاس را پیش‌بینی کنیم، می‌توانیم به سراغ این تابع برویم.
- تابع سافت‌مکس تمامی مقادیر یک بردار با طول K را به بازه‌ی صفر تا ۱ می‌برد، به طوری که جمع تمامی مقادیر این بردار با هم ۱ می‌شود.
- این تابع برای نورون‌های لایه‌ی خروجی استفاده می‌شود؛ زیرا در شبکه‌های عصبی در آخرین لایه (خروجی) به طبقه‌بندی ورودی‌ها در کلاس‌های مختلف نیاز داریم.

which activation function?

- تابع سیگموید در مسائل طبقه‌بندی معمولاً خیلی خوب عمل می‌کند.
- توابع سیگموید و تانژانت هایپربولیک، به‌دلیل مشکل محوشدگی گرادیان، در بعضی مواقع استفاده نمی‌شوند.
- تابع فعالساز واحد یک‌سوشده‌ی خطی (ReLU/ Rectified Linear Unit) فقط در لایه‌های نهان استفاده می‌شود.
- اگر با مشکل مرگ نورون در شبکه مواجه هستیم، تابع Leaky ReLU

(f(x)= a = max(0.01x, x)) می‌تواند گزینه‌ی بسیار خوبی باشد. (وقتی مقدار کمتر از صفر است.)

Loss/cost function

معیاری برای سنجش مناسب بودن مدل از نظر قابلیت و توانایی در پیشگویی مقدارهای جدید است.

- توابع زیان مربوط به الگوریتم‌های دسته‌بندی
- توابع زیان مربوط به رگرسیون

تابع زیان مسائل رگرسیون

- تابع زیان هوبر یا میانگین خطای قدرمطلق هموار شده

$$L_\delta(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 & |y-\hat{y}| > \delta \end{cases}$$

- تابع زیان Log-Cosh

$$L_{log-cosh}(y,\hat{y}) = \sum_{i=1}^{n} \log(\cosh(y-\hat{y}))$$

- تابع زیان چندکی (Quantile Loss)

$$L_\gamma(y,\hat{y}) = \sum_{i \in \{i;y_i<\hat{y}\}} (1-\gamma)|y_i-\widehat{y_i}| + \sum_{i \in \{i;y_i \ge \hat{y}_i\}} (\gamma)|y_i-\widehat{y_i}|$$

تابع زیان مسائل دسته بندی

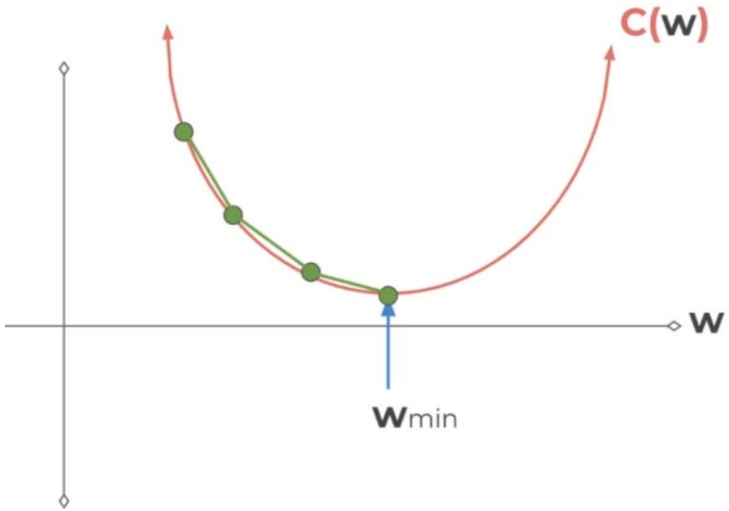
- تابع زیان Hinge

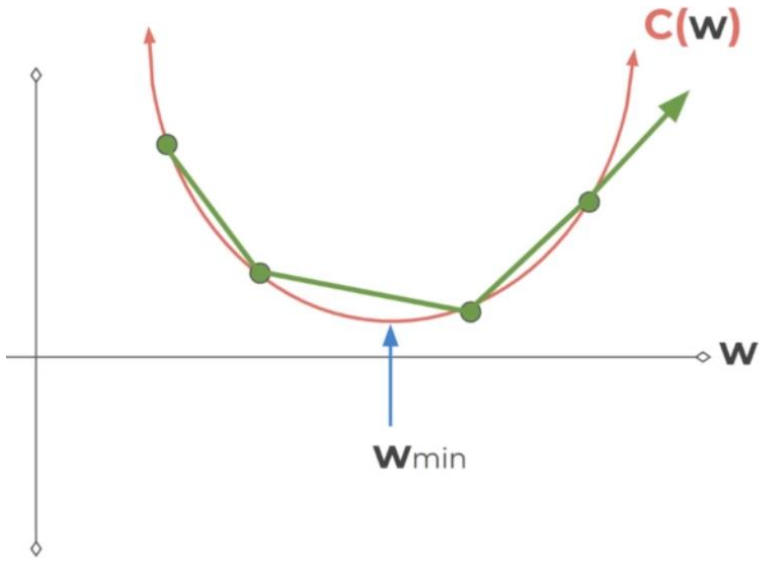
$$\text{loss} = \max(0, 1 - (y * y'))$$

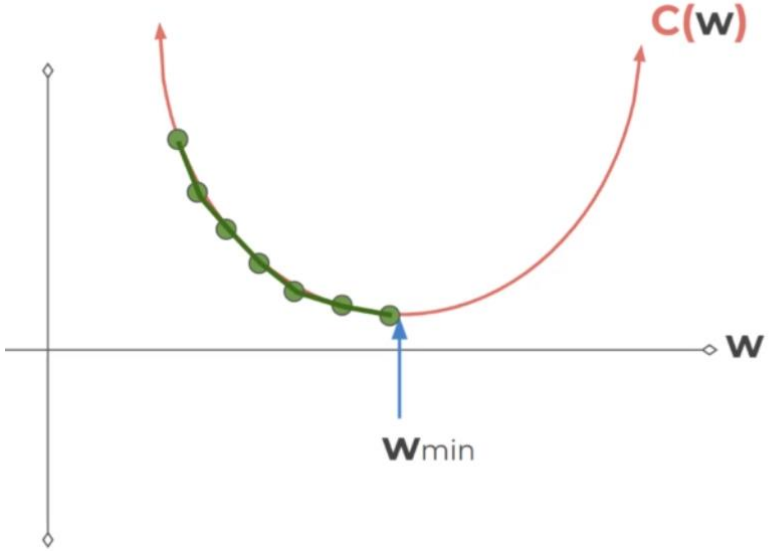
- تابع زیان cross-entropy

$$\text{cross-entropy} = \sum_i p_i \cdot log_2 \frac{1}{\hat{p}_i}$$

gradient decent

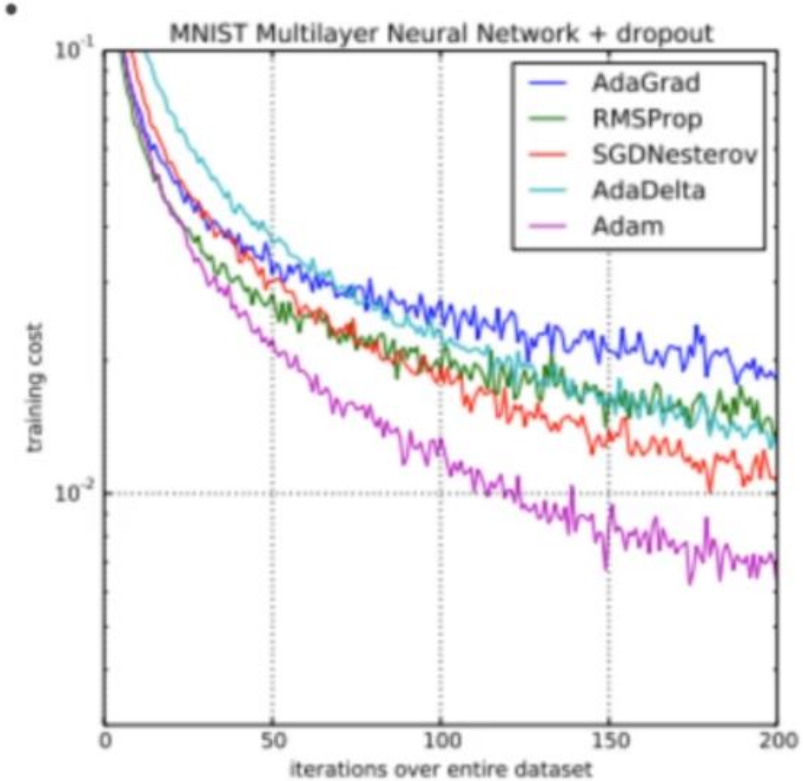






$$\nabla f(x_1,x_2,\ldots,x_n)=(\frac{\partial f}{\partial x_1},\frac{\partial f}{\partial x_2},\ldots,\frac{\partial f}{\partial x_n})$$

- Adam versus other gradient descent algorithms:



pip install tensorflow

pip install tensorflow-gpu