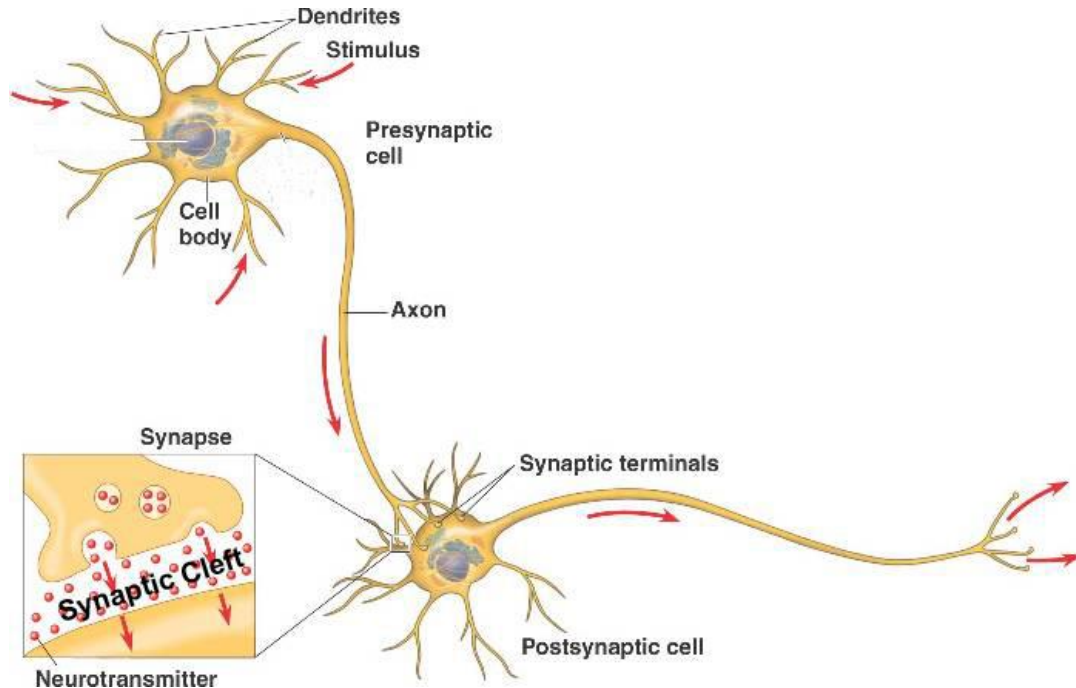


Artificial Neural Network

Liang Liang

Neural Network ? Neuron?



Artificial neural networks were proposed to mimic human neural networks

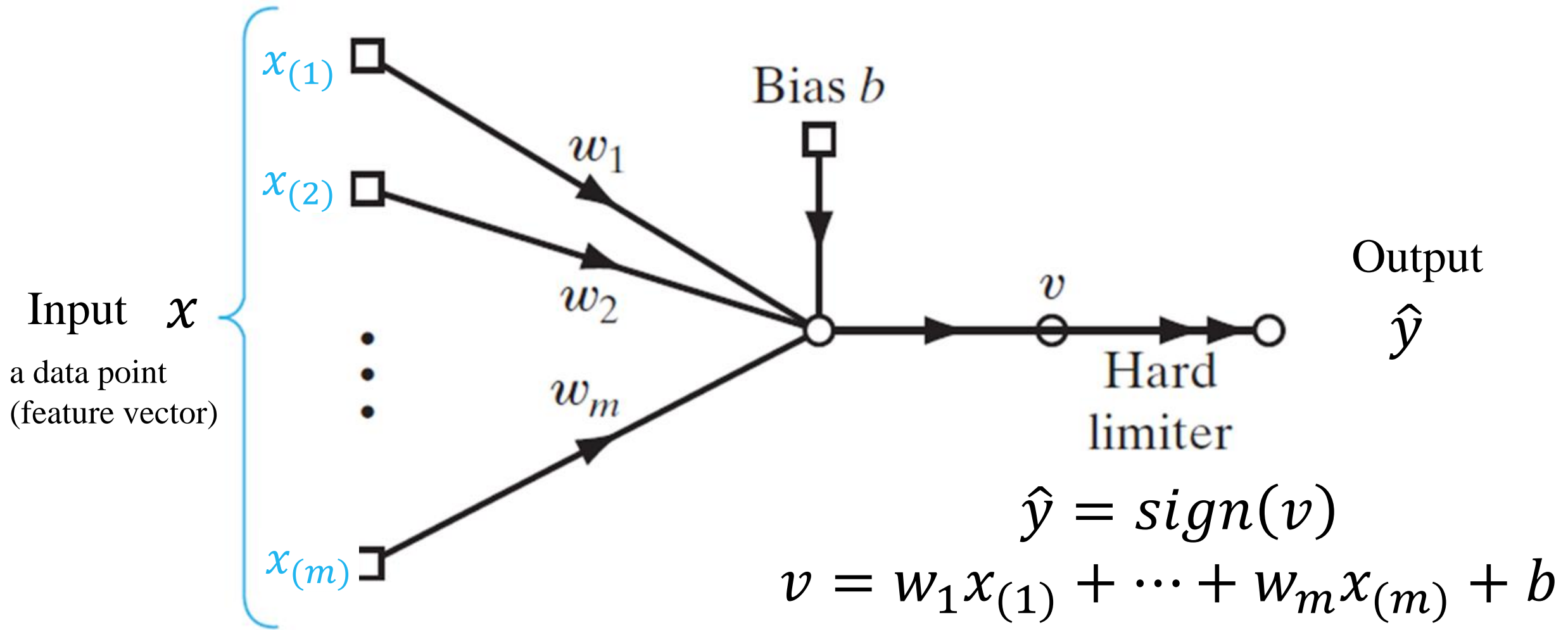
They could be similar, but NOT the same

In the field of Machine Learning,
Network refers to Artificial Neural Network
Unit refers to Neuron in an artificial neural network

Neural Networks were 'invented' ~ 60 years ago

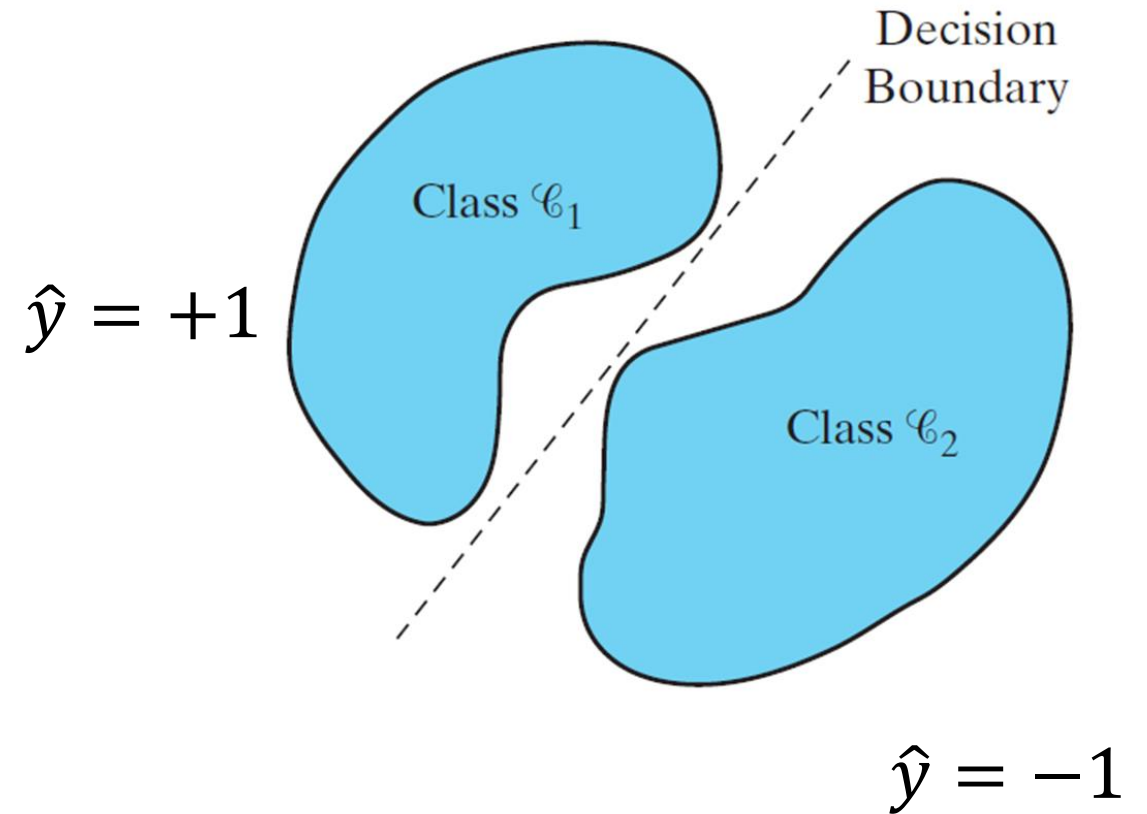
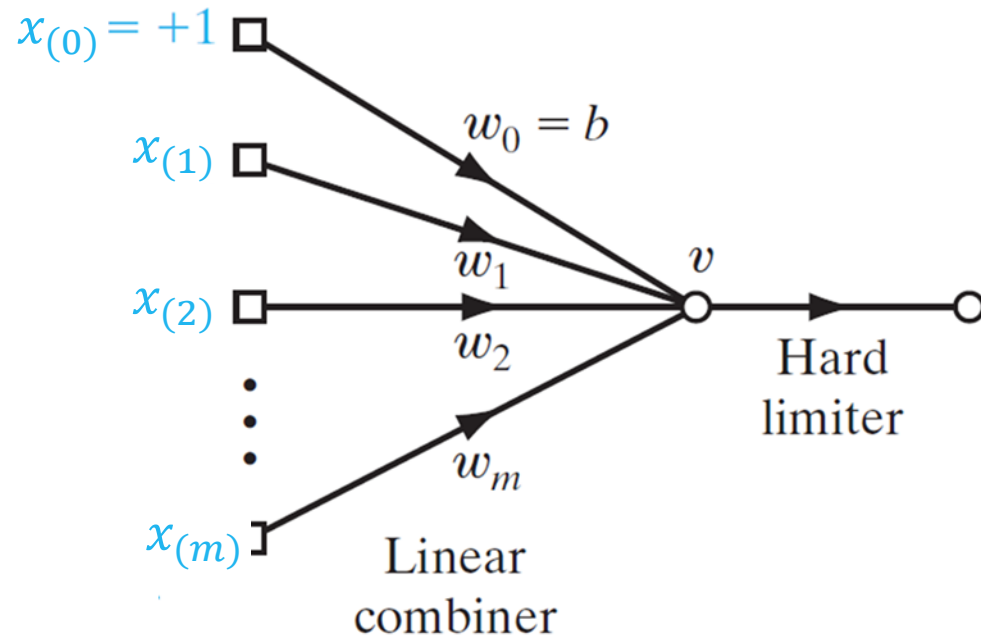
- McCulloch and Pitts (1943), introducing the idea of neural networks as computing machines.
- Hebb (1949), postulating the first rule for self-organized learning.
- Rosenblatt (1958), proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).
 - Rosenblatt's perceptron is the simplest form of a neural network used for the classification of patterns that are linearly separable

Perceptron

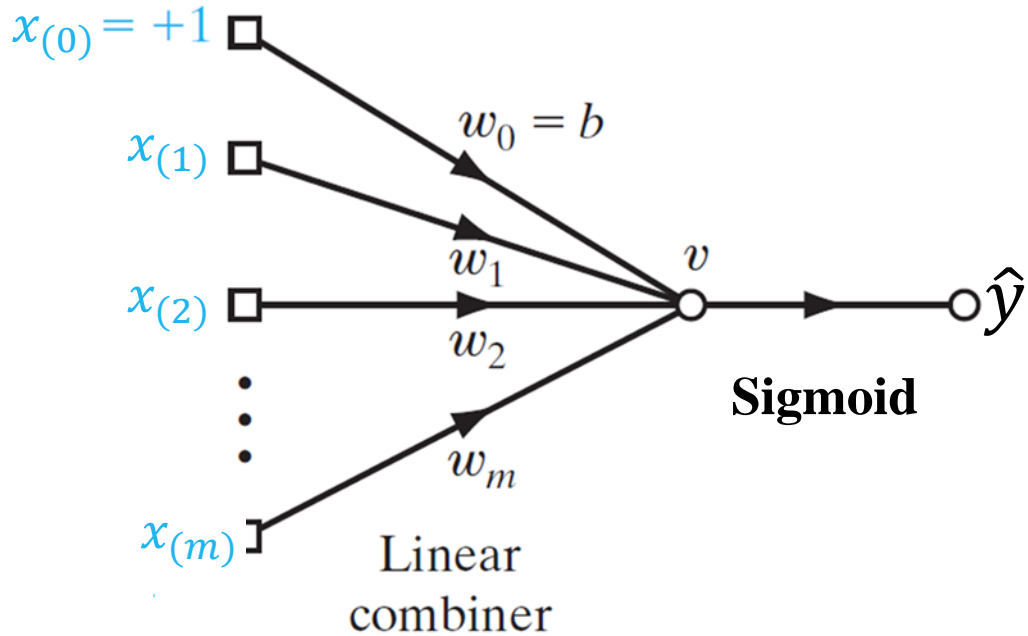


$$\text{sign}(100) = 1, \text{sign}(-10) = -1, \text{sign}(0) = 0$$

Perceptron



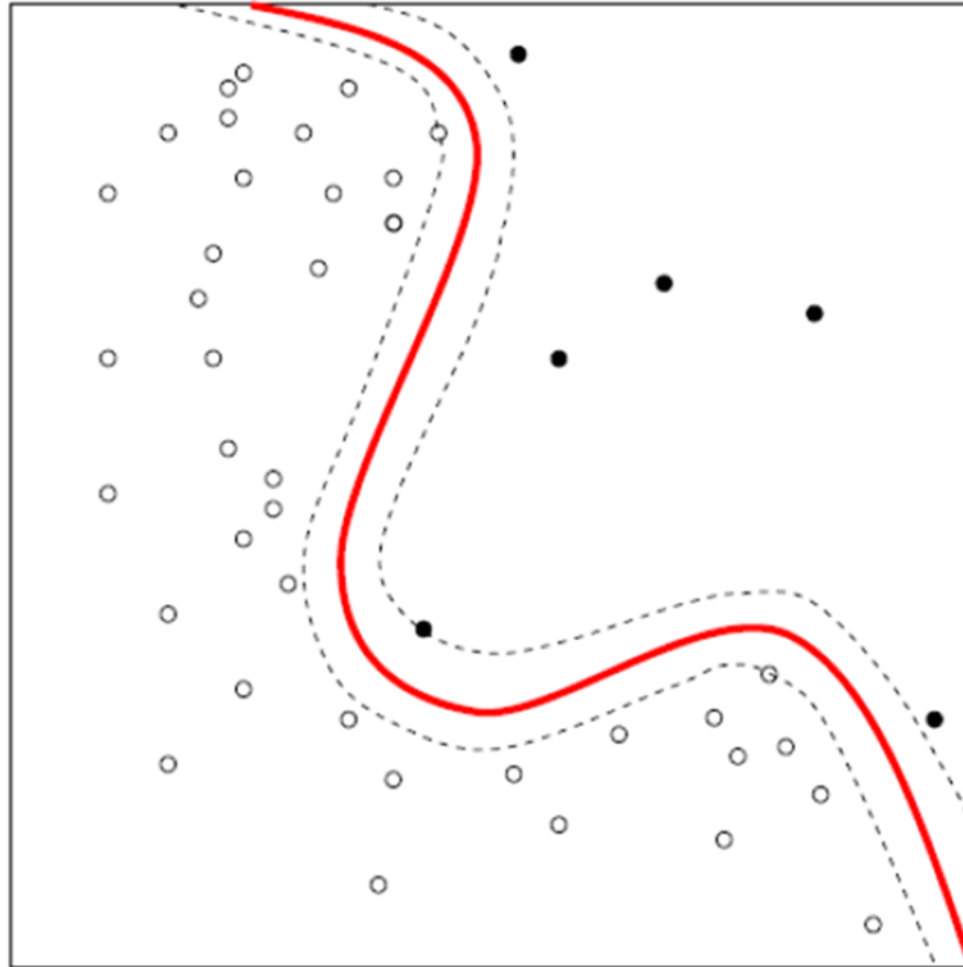
Logistic Regression Classifier: an enhanced perceptron



- Replace the sign function with Sigmoid

$$\hat{y} = s(v) = \frac{1}{1 + e^{-v}}$$

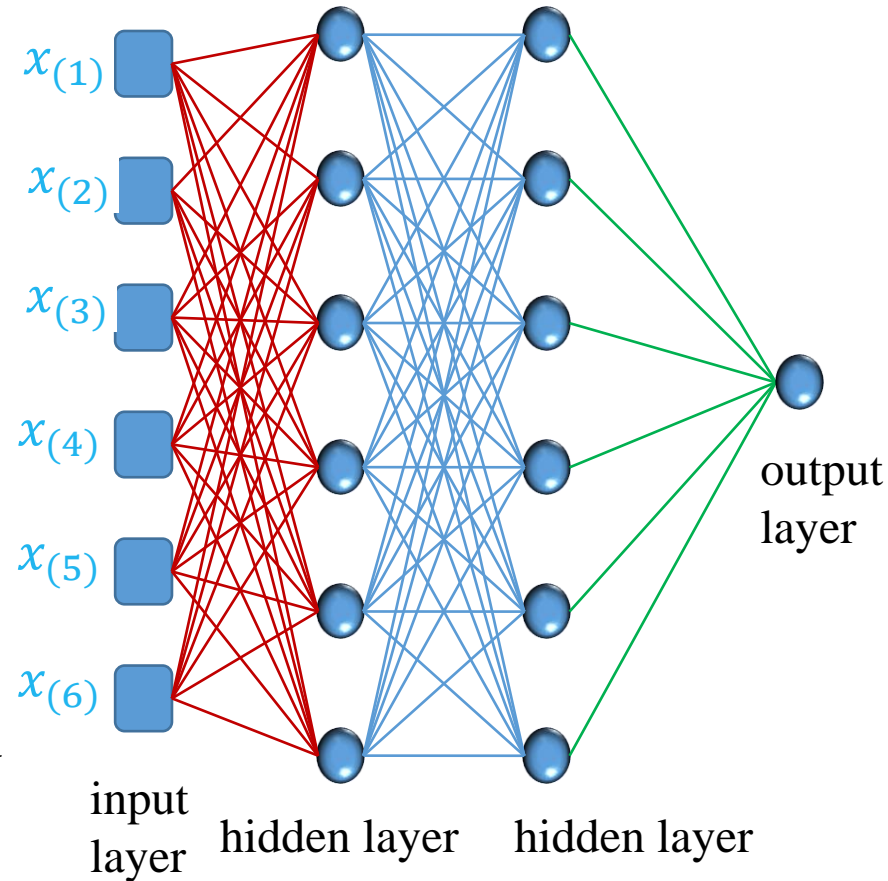
Limitation of Perceptron: it is a linear classifier
but, we may need a **nonlinear decision boundary**



A set of connected perceptrons is a neural network

medical diagnosis

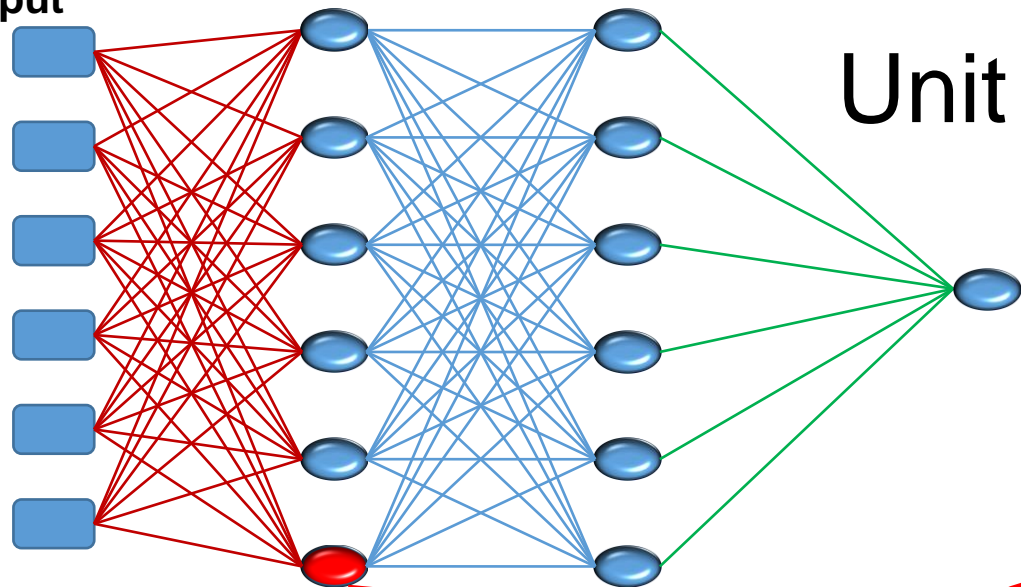
Input:
feature vector
of a patient



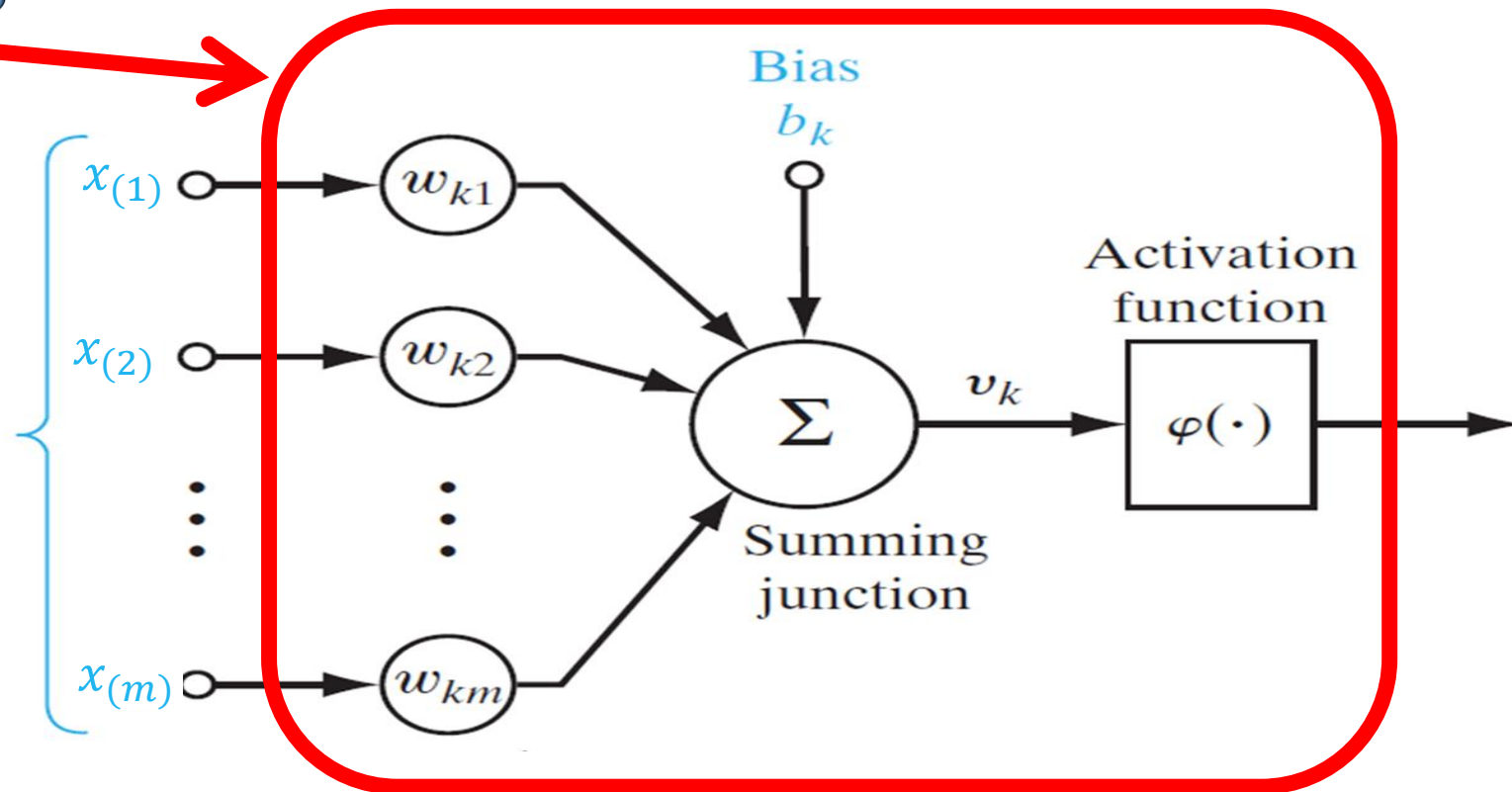
1: disease
0: normal

This network is also called
Multi-Layer Perceptron
(MLP)

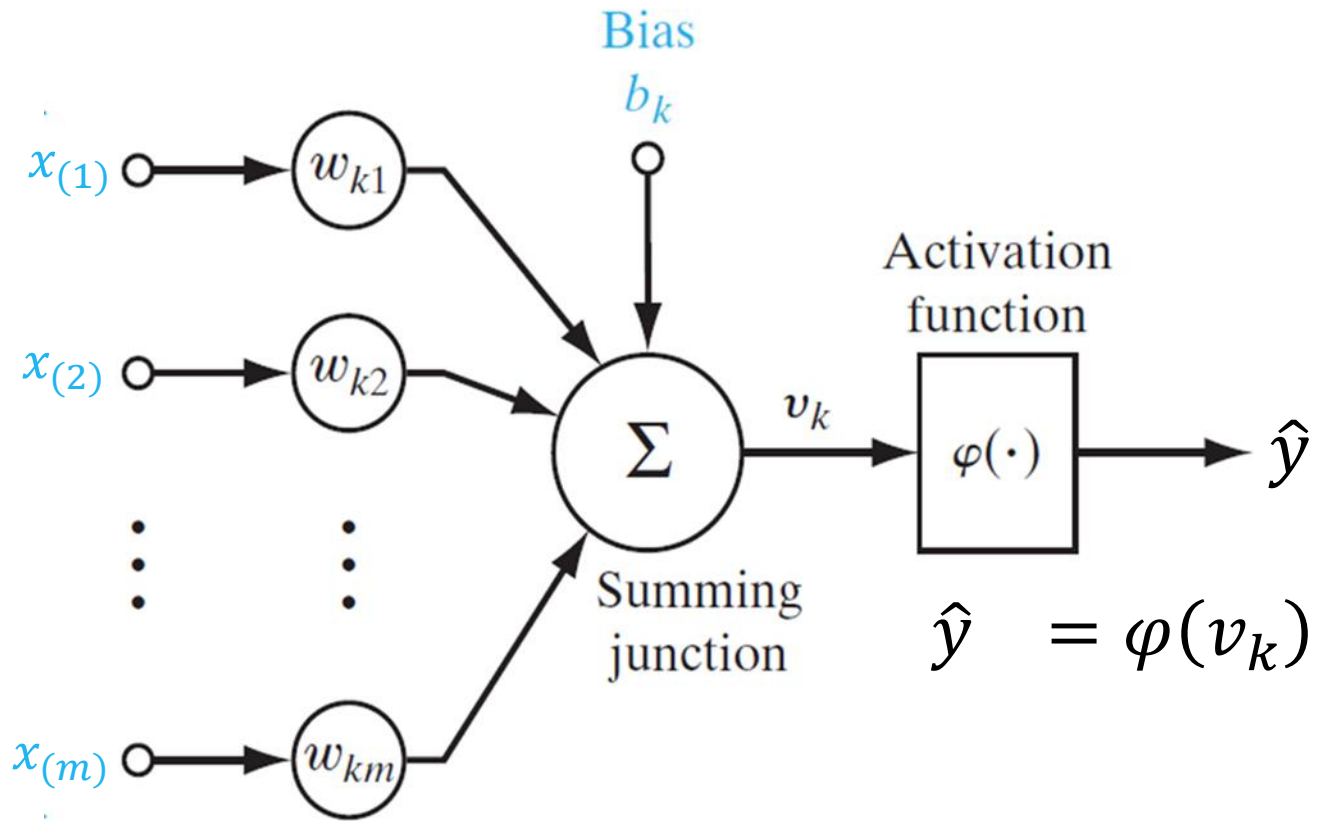
Unit in a Network



One Unit (Perceptron)



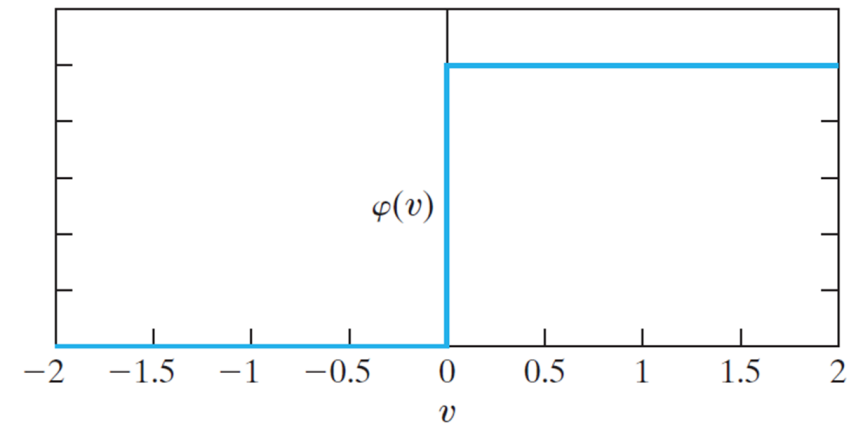
Unit in a Network: the activation function



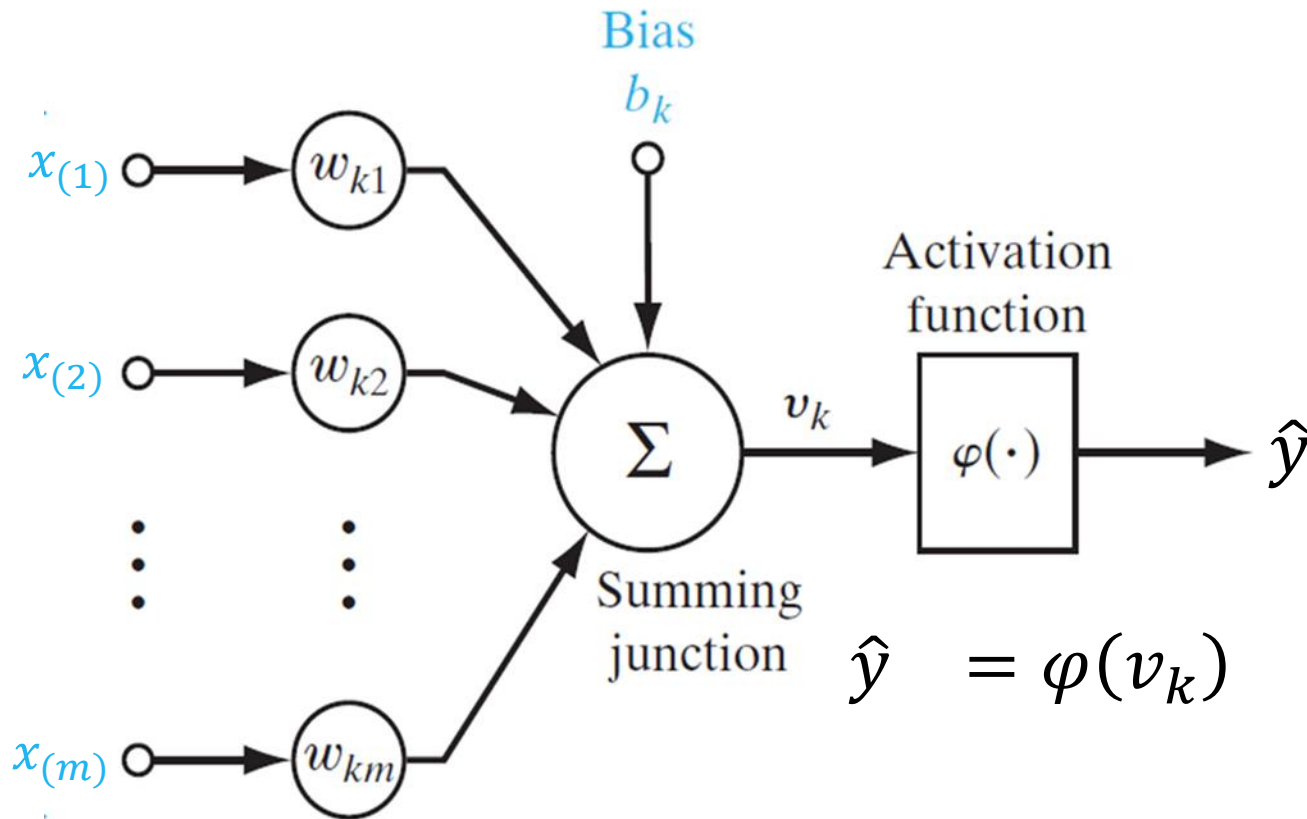
k is the index of the unit in the layer

Threshold function

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$



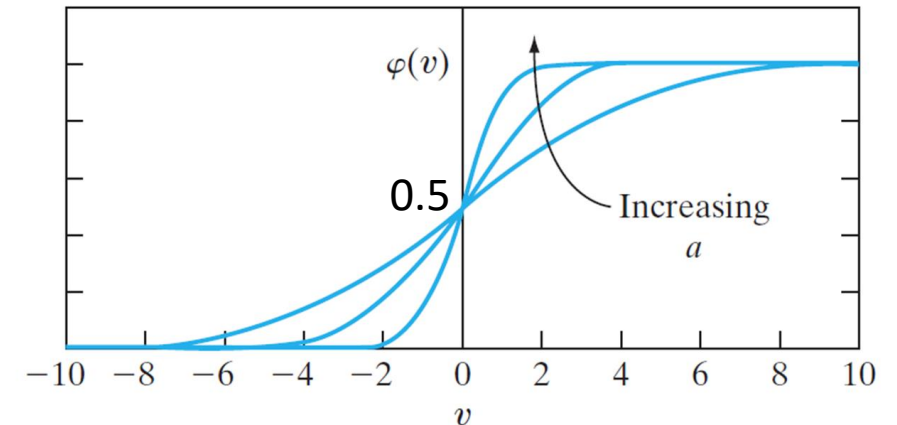
Unit in a Network: the activation function



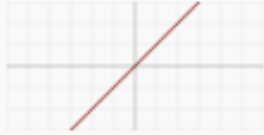

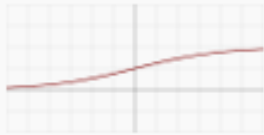
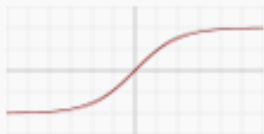
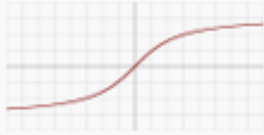
k is the index of the unit in the layer

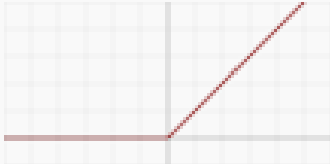
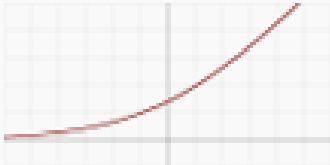
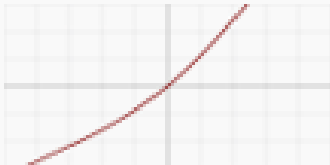
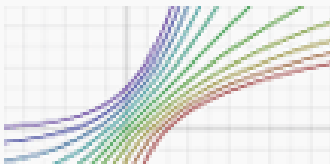
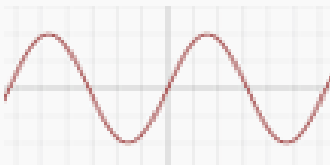
Sigmoid function

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

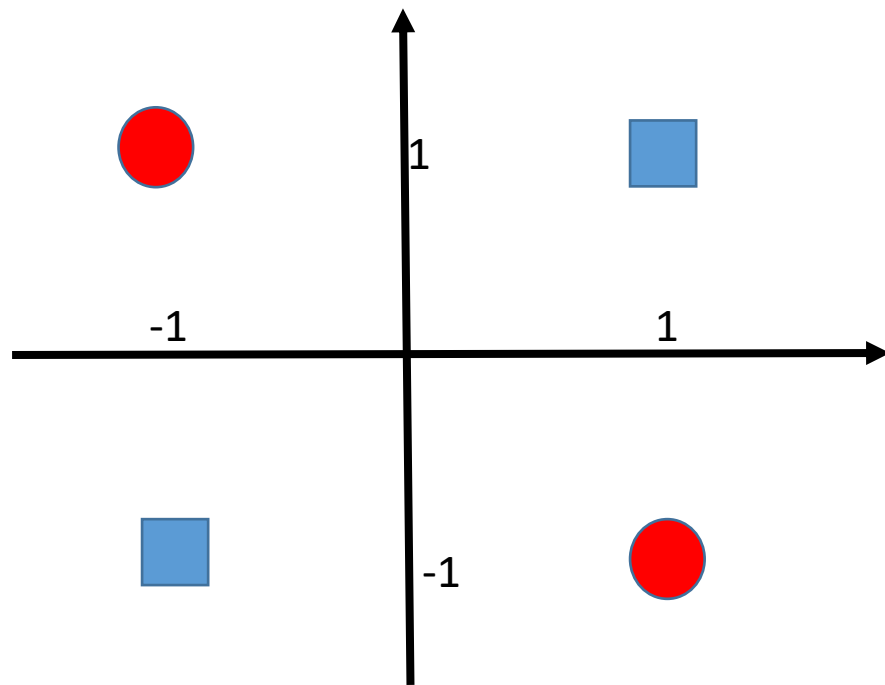






More Activation Functions, Notation: $f(x) \sim \varphi(v)$

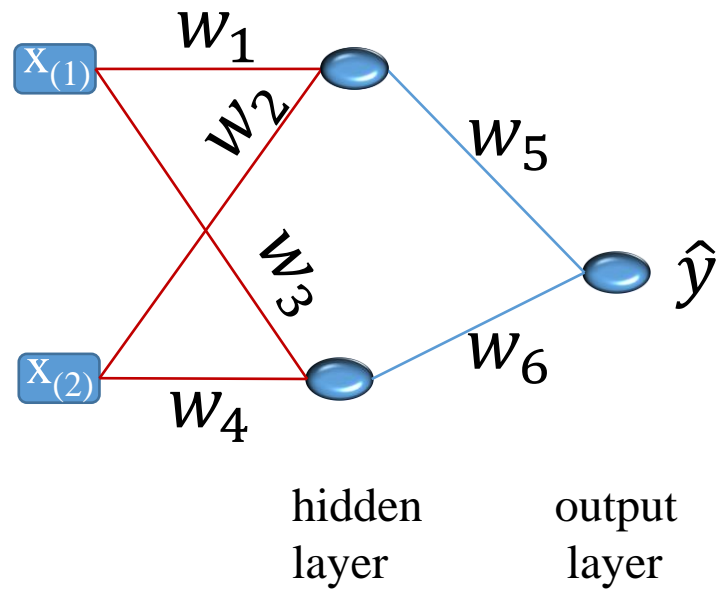
	Name	Plot	Equation	Derivative
Linear	Identity		$f(x) = x$	$f'(x) = 1$
	Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Sigmoid	Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
	<u>TanH</u>		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
	<u>ArcTan</u>		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$

Rectified Linear (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
<u>SoftPlus</u>		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$
<u>SoftExponential</u>		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1 - \alpha(x + \alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(\alpha + x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$

the XOR problem



Input vector x	Desired response y
$(-1, -1)$ 	-1
$(-1, +1)$ 	+1
$(+1, -1)$ 	+1
$(+1, +1)$ 	-1



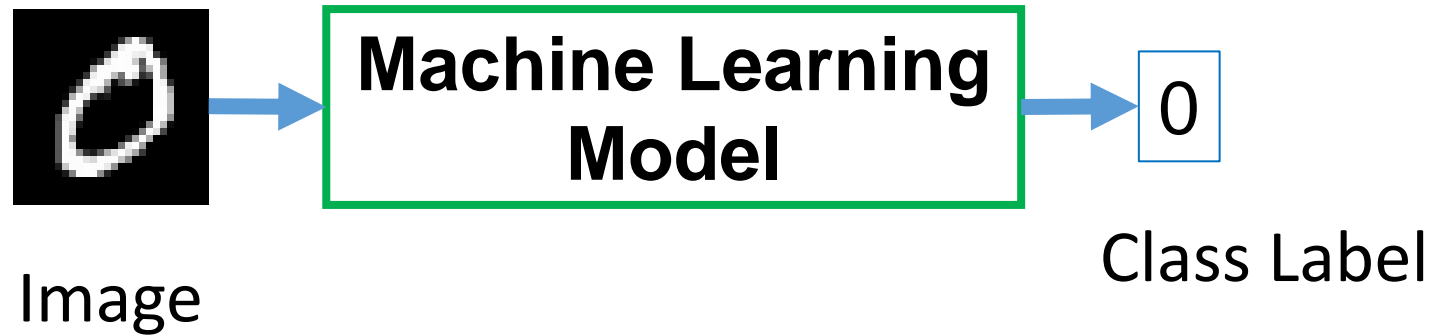
This neural network can solve the XOR problem
activation function is sign

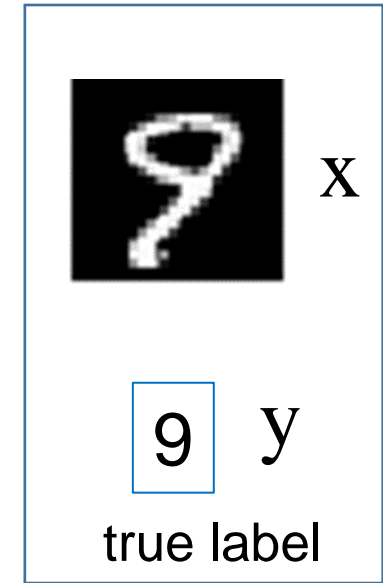
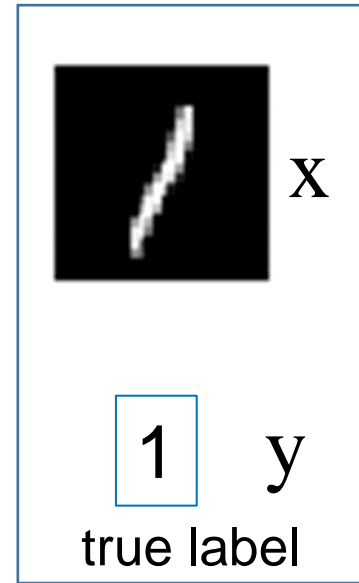
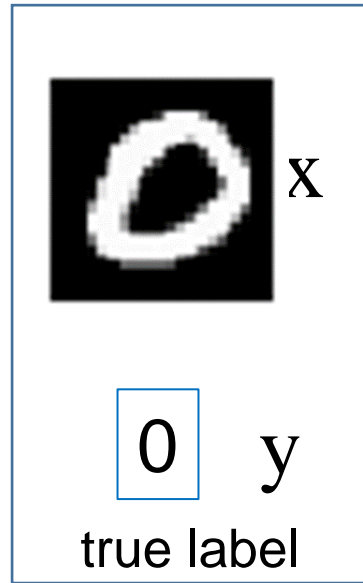
Expressive Capabilities of Neural Networks $\hat{y} = f(x)$

- Boolean functions:
 - Every Boolean function can be represented by a network with a single hidden layer
 - It might require a very large number of units in the hidden layer
- Continuous functions: (**Universal Approximation Theorem**)
 - Every bounded continuous function can be approximated with arbitrary small error, by a network with one hidden layer
 - Activation functions need to be locally bounded and piecewise continuous
- Deep network vs shallow network

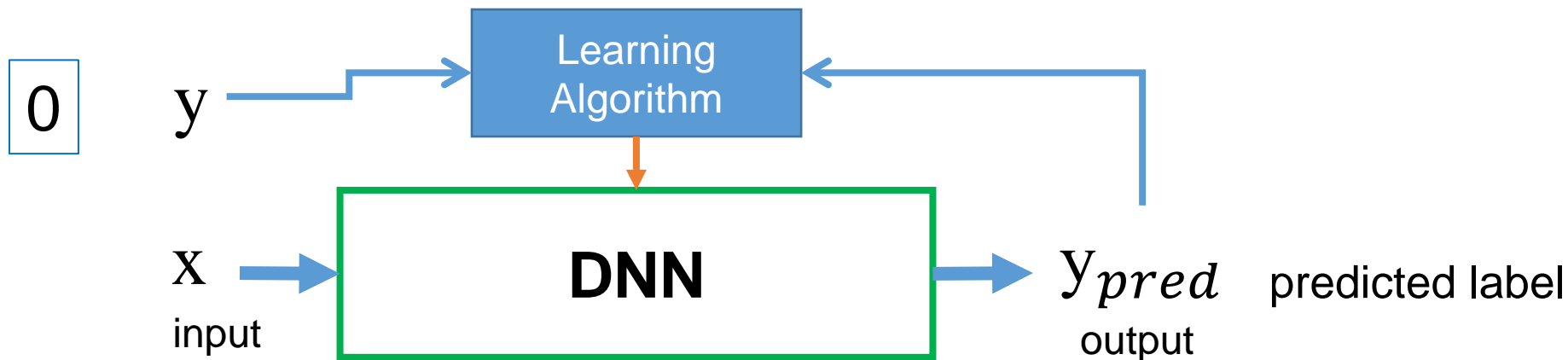
A continuous function can be approximated by a deep network or a shallow network. The deep network usually uses a smaller number of units.

- Handwritten digit recognition

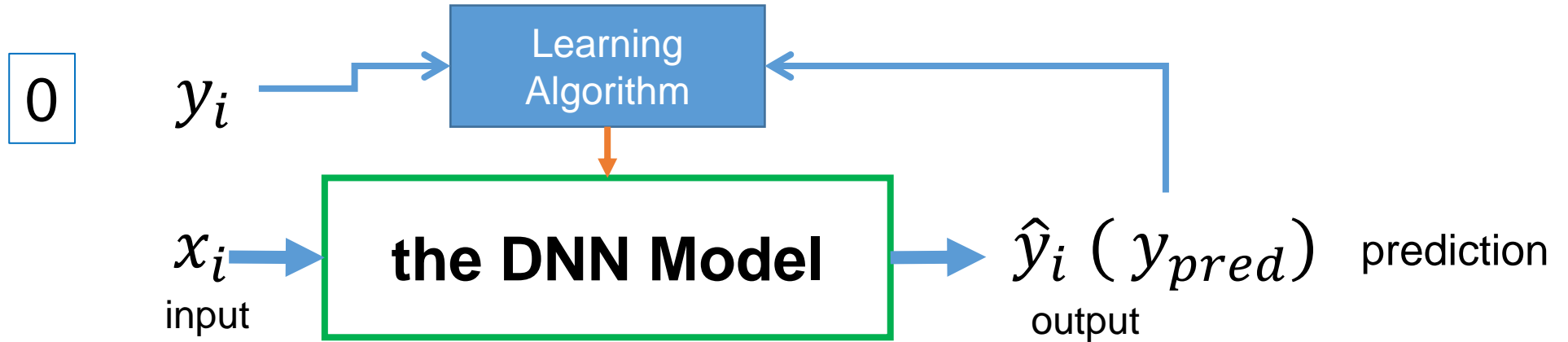




During the training stage, the ML model will see many images and the true labels of the images. Give an input x , if the output y_{pred} is not equal to the true label y , then a Learning Algorithm will adjust the ML model. After many...many adjustments (learning), the ML model can make correct predictions.



Loss function:



Loss function:

$L(\hat{y}_i, y_i)$ measures the difference between the prediction \hat{y}_i and the ground truth y_i , where i is the index of a data sample (e.g., an image)

The learning algorithm will minimize the loss function, in order to find the best (trainable) parameters of the machine learning model

Binary Classifier: Linear + Sigmoid

a data point x



Binary
Classifier

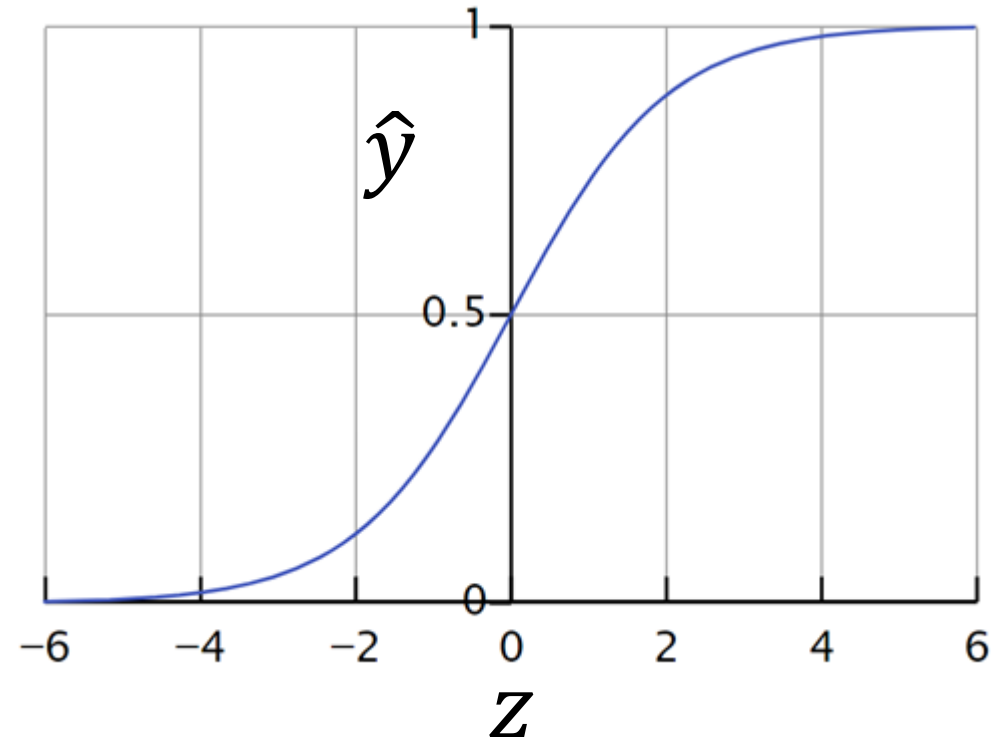
$\hat{y} > 0.5$, it is a cat

$\hat{y} < 0.5$, it is not a cat

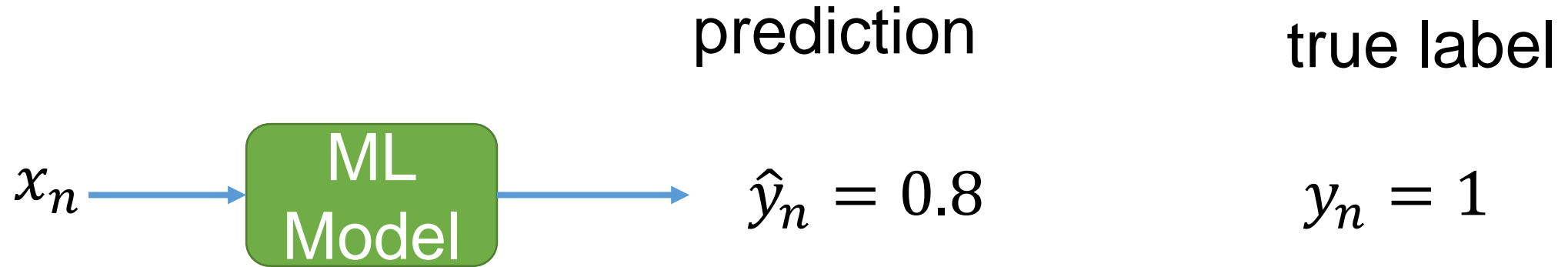
\hat{y} predicted soft label

Linear function $z = z(x)_1$

Sigmoid function $\hat{y} = \frac{1}{1 + e^{-z}}$



Binary Cross-Entropy (BCE) loss for binary-class classification



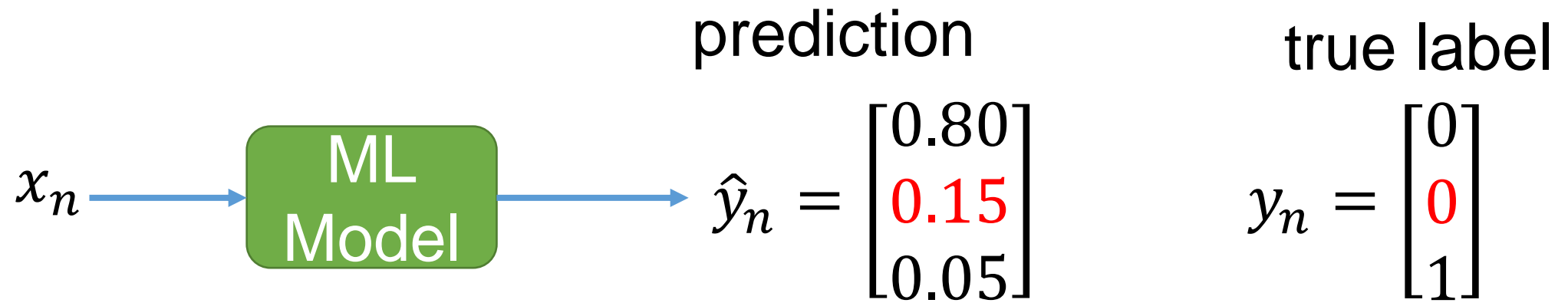
The binary cross-entropy (BCE) loss of this sample is

$$L(y_n, \hat{y}_n) = -[1 \times \log(0.8) + \textcolor{red}{0} \times \log(1 - 0.8)]$$

\log is the natural log

y_n could be 0 or 1

Cross-Entropy(CE) loss for multi-class classification



The cross-entropy (CE) loss of this sample is

$$L(y_n, \hat{y}_n) = -[0 \times \log(0.80) + 0 \times \log(0.15) + 1 \times \log(0.05)]$$

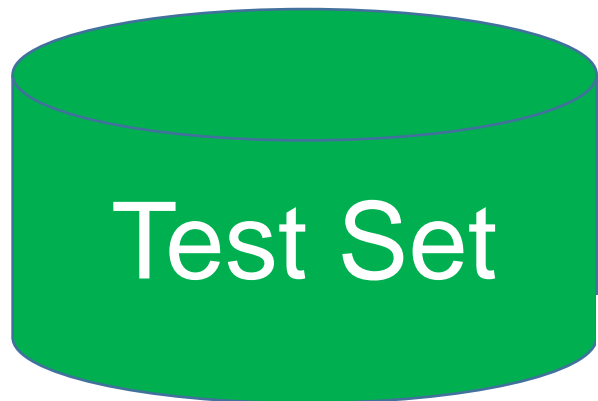
\log is the natural log

y_n is a one-hot vector

The cross-entropy loss of a sample is $L(y_n, \hat{y}_n)$

The cross-entropy loss of the training samples is

$$Loss = \frac{1}{N} \sum_{n=1}^N L(y_n, \hat{y}_n)$$



Let's use 10 test images

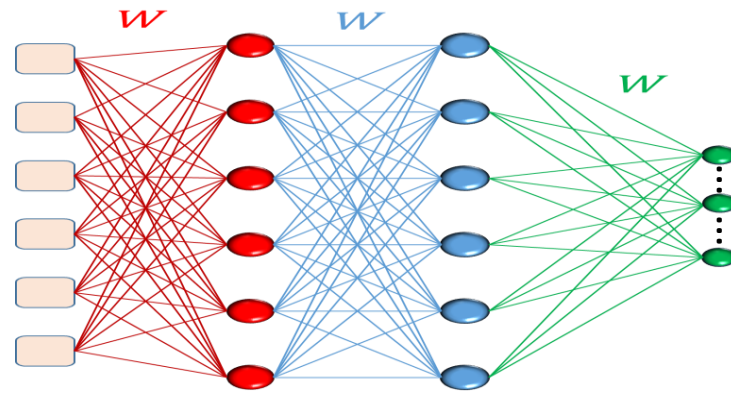


True Label	0	1	2	3	4	5	6	7	8	9
Prediction	0	1	2	3	4	5	6	9	6	9



classification accuracy = $8/10=80\%$

MLP



Define the model

```
1 model = Sequential()
2 model.add(Dense(units=256, activation='relu', input_shape=(784,)))
3 model.add(Dense(units=256, activation='relu'))
4 model.add(Dense(units=10, activation='softmax'))
5 model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.01), metrics=['accuracy'])
6 model.summary()
```

note: it is better to use `sparse_categorical_crossentropy`

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	200960
dense_1 (Dense)	(None, 256)	65792
dense_2 (Dense)	(None, 10)	2570
Total params: 269,322		
Trainable params: 269,322		
Non-trainable params: 0		

Mini-batch Stochastic Gradient Descent (SGD)

- Initialization: initialize the network parameters using random numbers
one epoch (the network sees all of the training data points)
- Randomly shuffle data points in training set and divide them into mini-batches
a mini-batch is a subset of the training set. Assume there are **M** mini-batches
- for **m** in $[1, 2, \dots, M]$:
 - N_m is called batch_size
 - assume data points in current mini-batch are $\{(x_i, y_i), i = 1, \dots, N_m\}$
 - Forward Pass: compute output \hat{y} for every data point in this mini-batch
 - Backward Pass: compute some derivatives
 - Update parameters of the network
- Run many epochs until convergence

Mini-batch Stochastic Gradient Descent (SGD)

- Initialization: initialize the network parameters using random numbers
one epoch (the network sees all of the training data points)

In Keras:
`model.fit(x_train, y_train)`

- Run many epochs until convergence