COMP 554 / CSDS 553 Advanced NLP

Faizad Ullah

Basics of Neural Networks

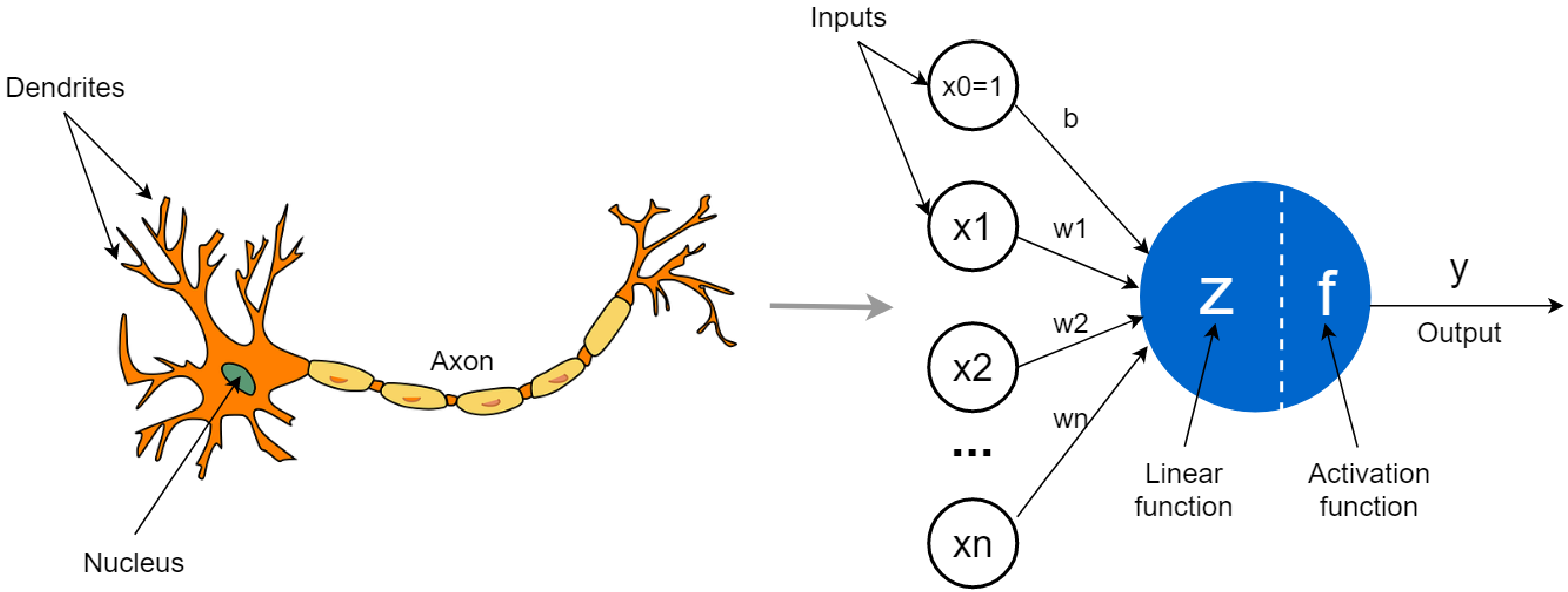
Four Components for ML Systems

- A feature representation of the input, i.e., $[x_1, x_2, ..., x_n]$
- A classification function that computes `\hat{y}`, the estimated class, via `p(y|x)`. Sigmoid, softmax, etc.
- An objective function that we want to optimize for learning, usually involving minimizing a loss function corresponding to error on training examples. Cross-entropy loss function, MAE, MSE, etc.
- An algorithm for optimizing the objective function. The gradient descent

Four Components for ML Systems

- Training: We train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.

- Testing: Given a test example x we compute $p(y|x)$ and return the higher probability label $y = 1$ or $y = 0$.

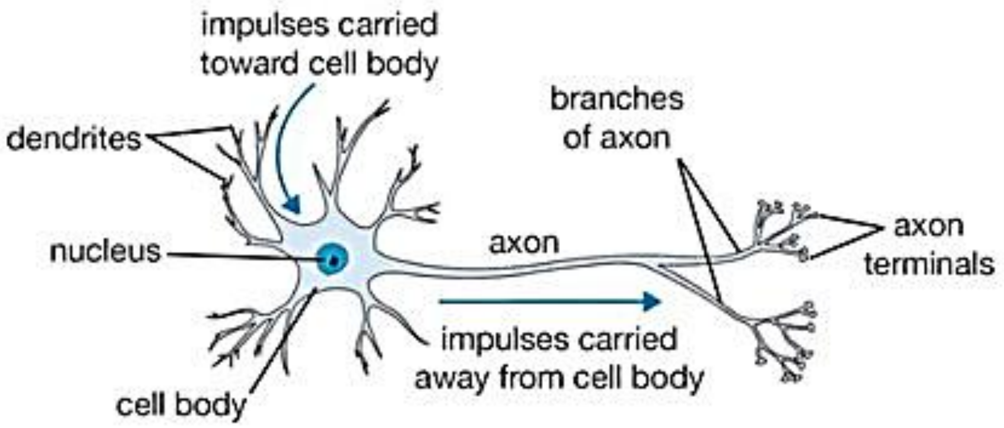


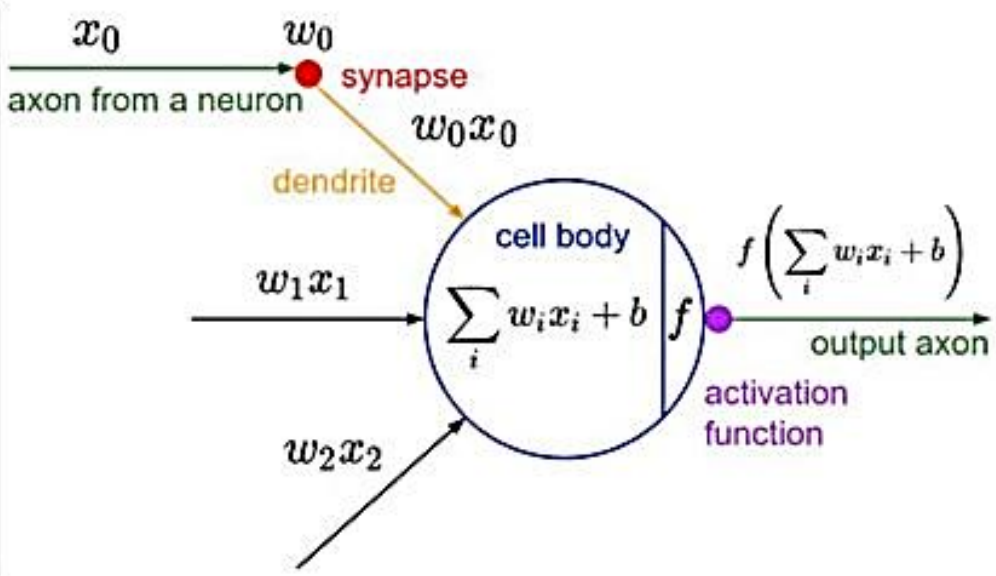
Dendrites receive signals from other neurons

Soma processes the information

Axons transmit the output

Synapses are the connections to other neurons





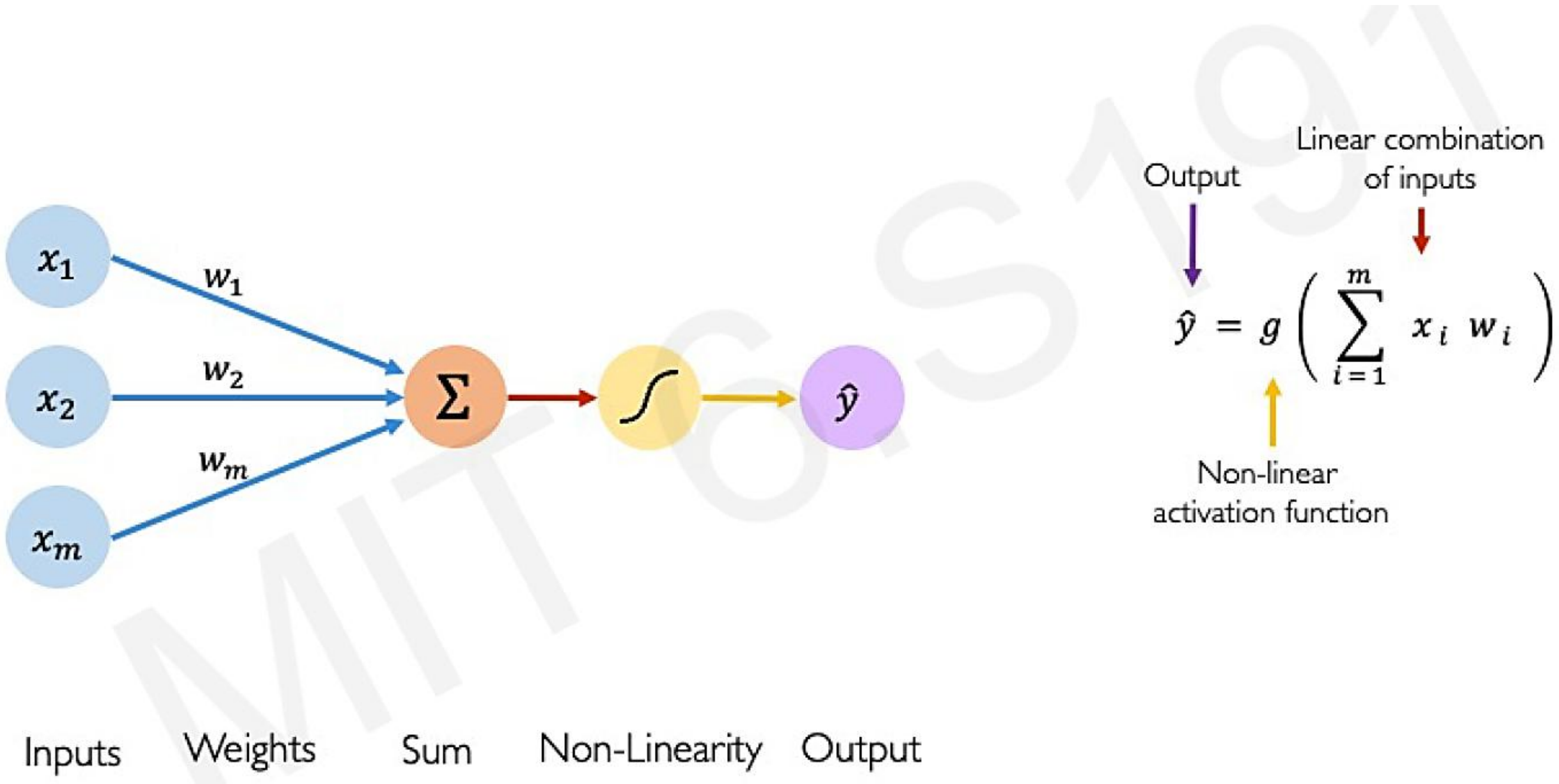
Dendrites receive signals from other neurons

Soma processes the information

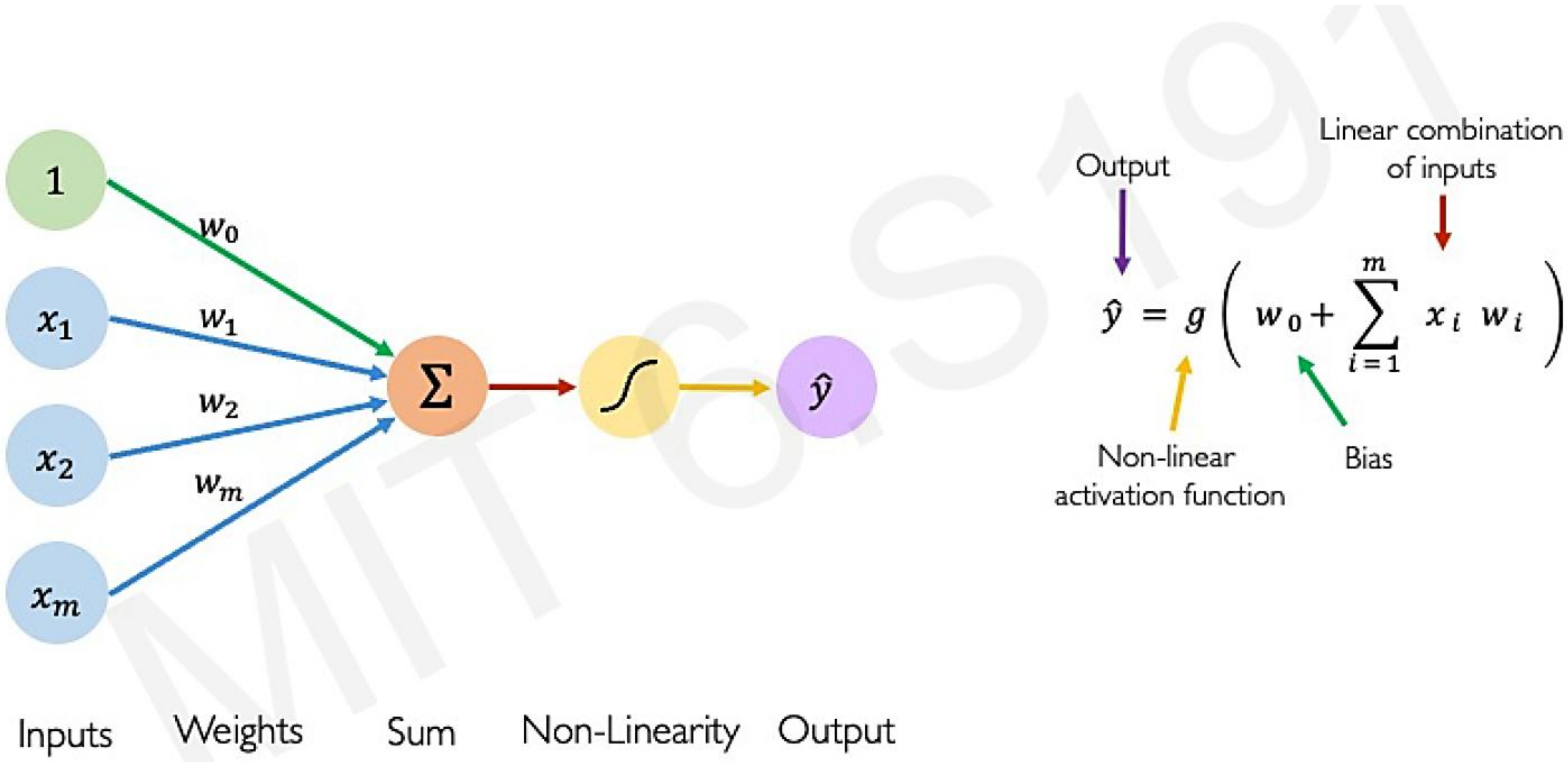
Axons transmit the output

Synapses are the connections to other neurons

Perceptron

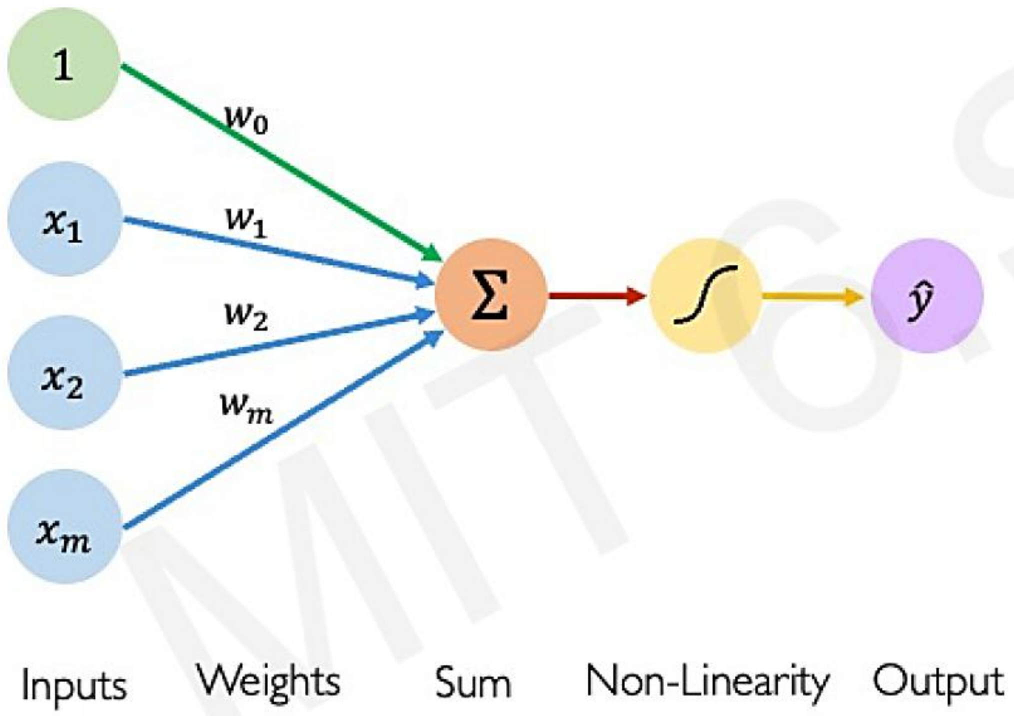

x_1
x_2
x_m
w_1
w_2
w_m
Σ
$\hat{y}$
Inputs
Weights
Sum
Non-Linearity
Output
Output
Linear combination of inputs
$\hat{y} = g\left(\sum_{i=1}^{m} x_i \, w_i\right)$
Non-linear activation function

Perceptron



1
w_0
x_1
w_1
x_2
w_2
x_m
w_m
Σ
$\hat{y}$
Inputs
Weights
Sum
Non-Linearity
Output
Output
Linear combination of inputs
$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$
Non-linear activation function
Bias
MIT 6.S191

Perceptron

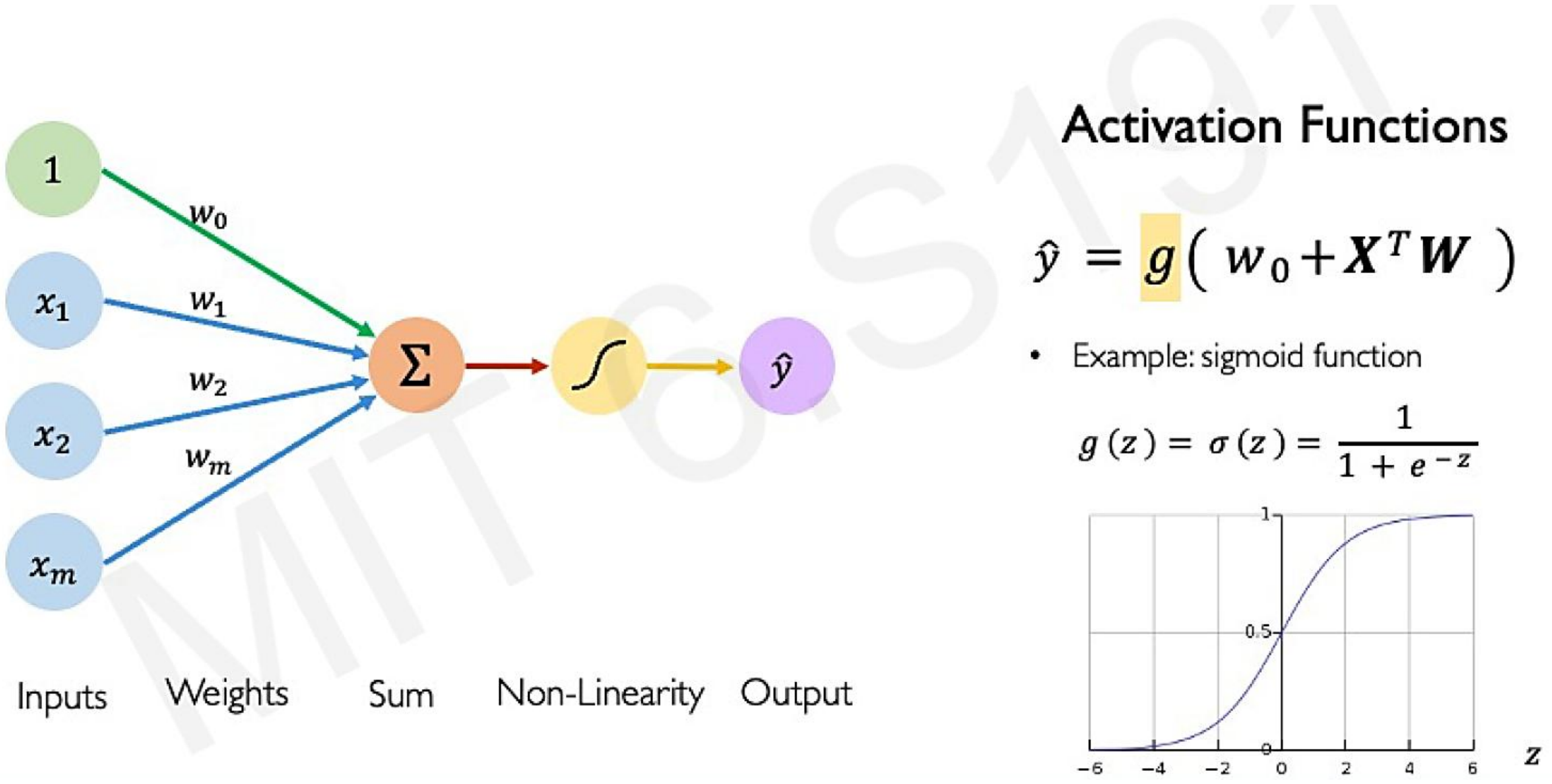


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

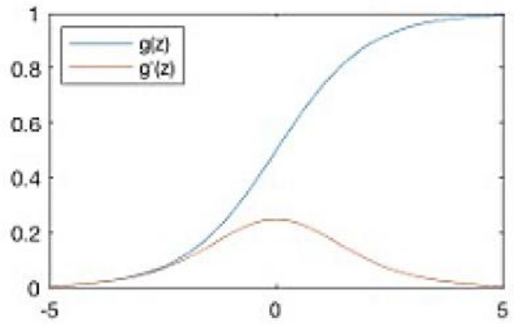
Perceptron



1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
$\hat{y}$
Inputs
Weights
Sum
Non-Linearity
Output
Activation Functions
$\hat{y} = g(\, w_0 + \boldsymbol{X}^T \boldsymbol{W} \,)$
• Example: sigmoid function
$g(z) = \sigma(z) = \dfrac{1}{1 + e^{-z}}$
z
MIT 6.S191

Activation Functions

Sigmoid Function



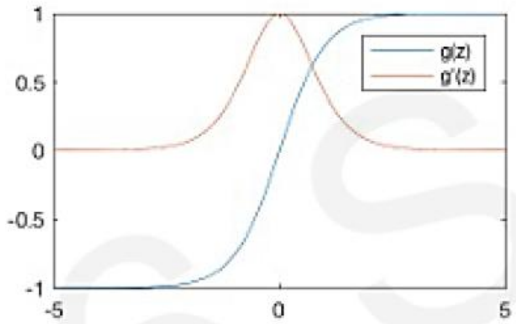
$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

```
tf.math.sigmoid(z)
```

```
torch.sigmoid(z)
```

Hyperbolic Tangent



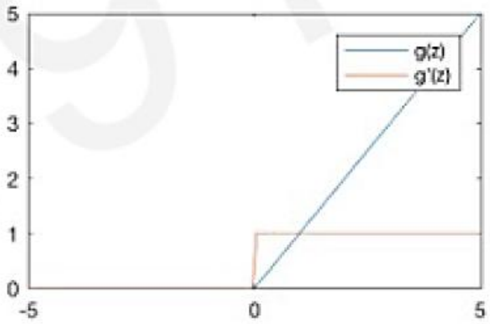
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

```
tf.math.tanh(z)
```

```
torch.tanh(z)
```

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

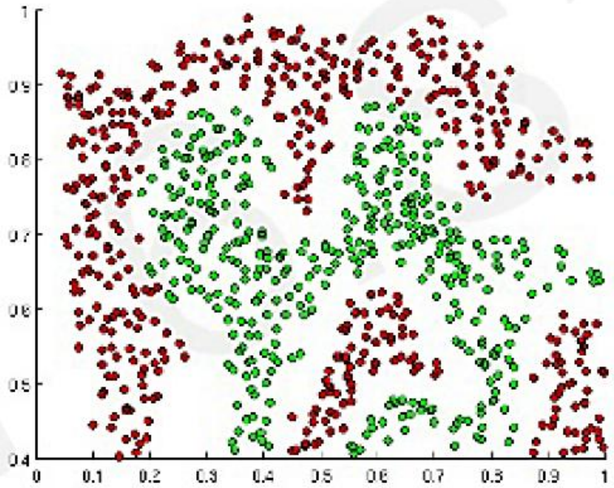
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

```
tf.nn.relu(z)
```

```
torch.nn.ReLU(z)
```

Activation Functions and Non-Linearities

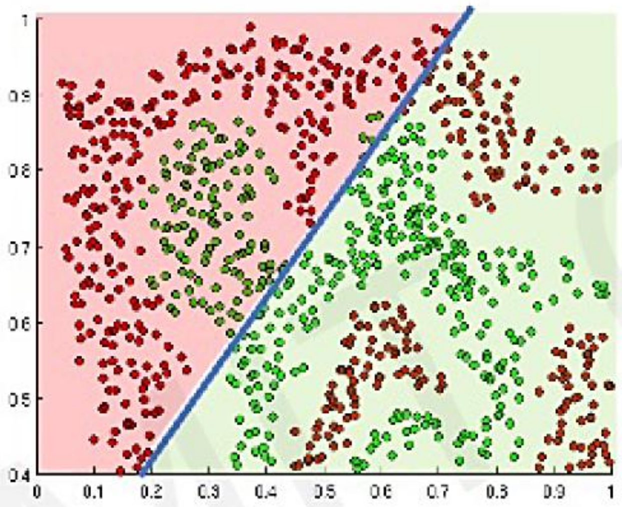
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



What if we wanted to build a neural network to distinguish green vs red points?

Activation Functions and Non-Linearities

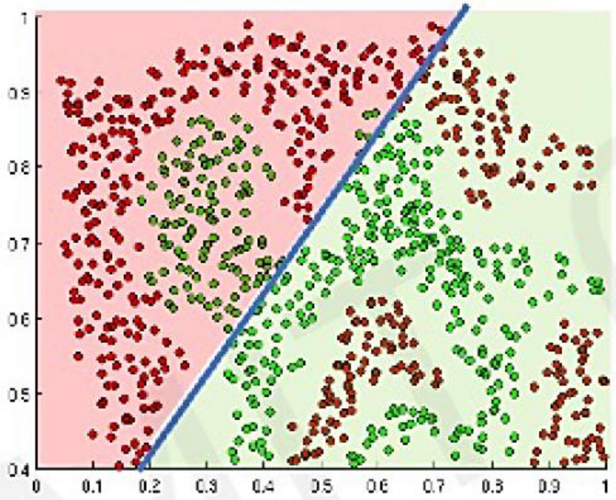
*The purpose of activation functions is to **introduce non-linearities** into the network*



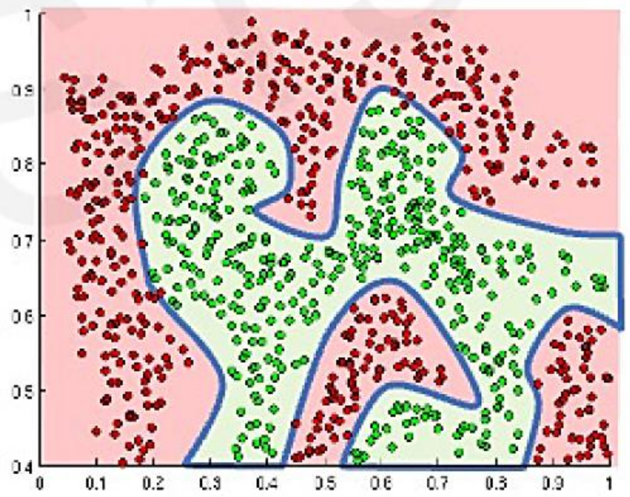
Linear activation functions produce linear decisions no matter the network size

Activation Functions and Non-Linearities

The purpose of activation functions is to ***introduce non-linearities*** *into the network*

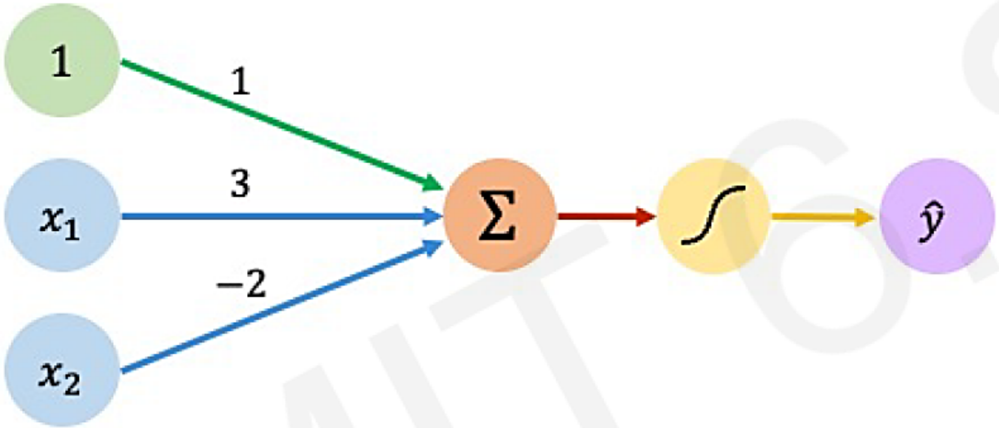


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

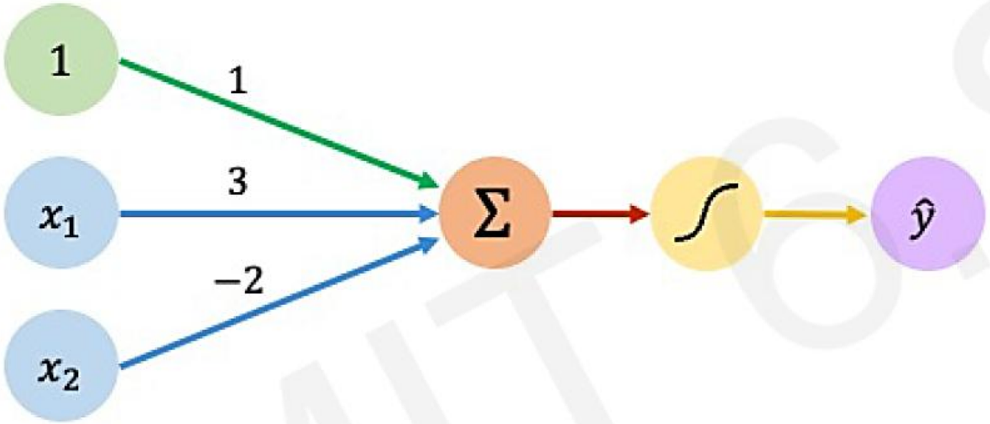
The Perceptron Example

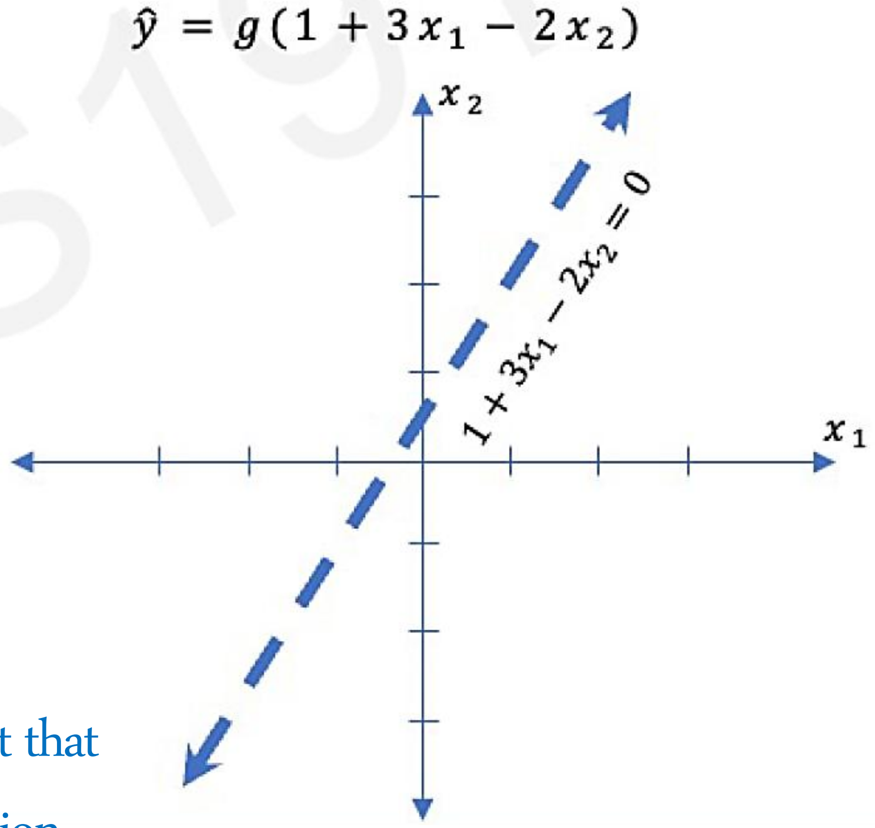


We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned} \hat{y} &= g(w_0 + \boldsymbol{X}^T \boldsymbol{W}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}}) \end{aligned}$$

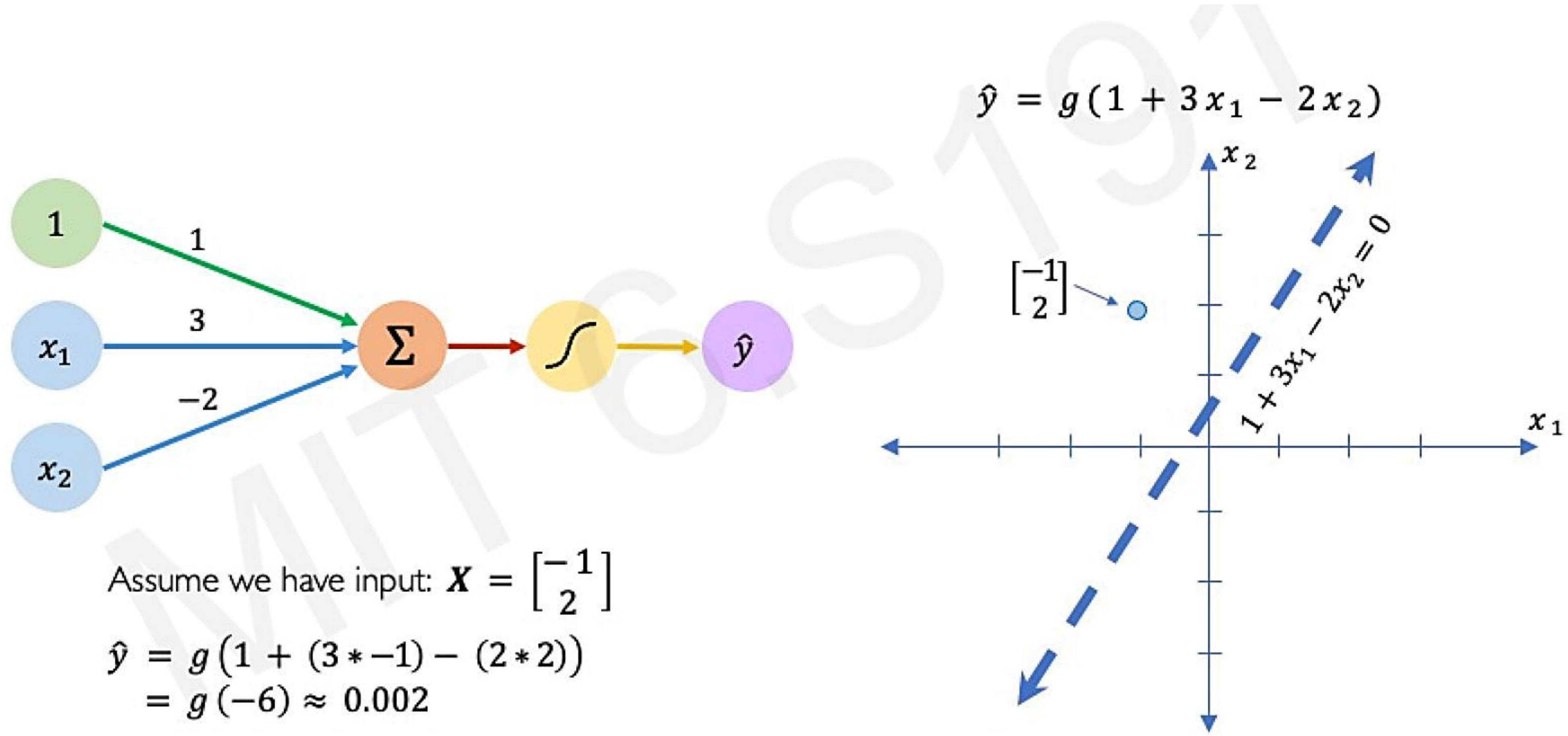
The Perceptron Example



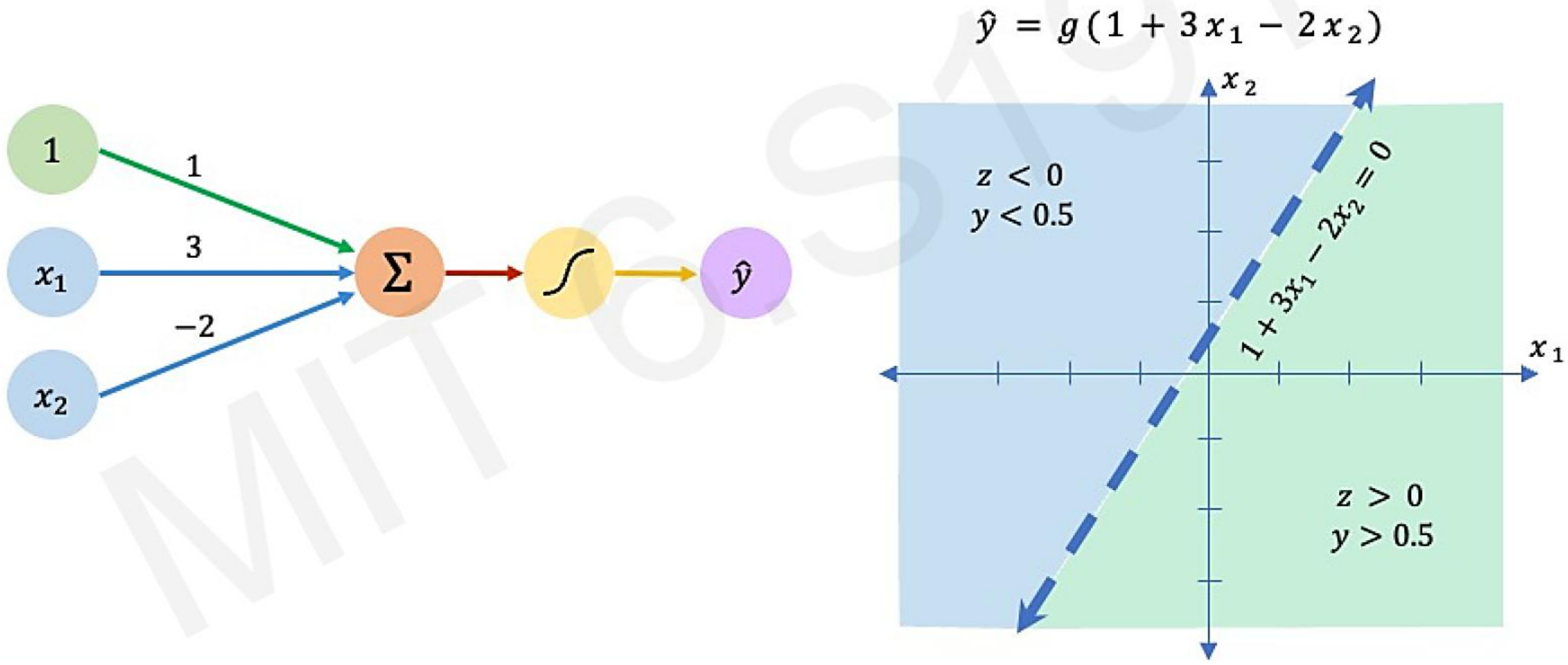


The weight w_i represents how important that input feature is to the classification decision

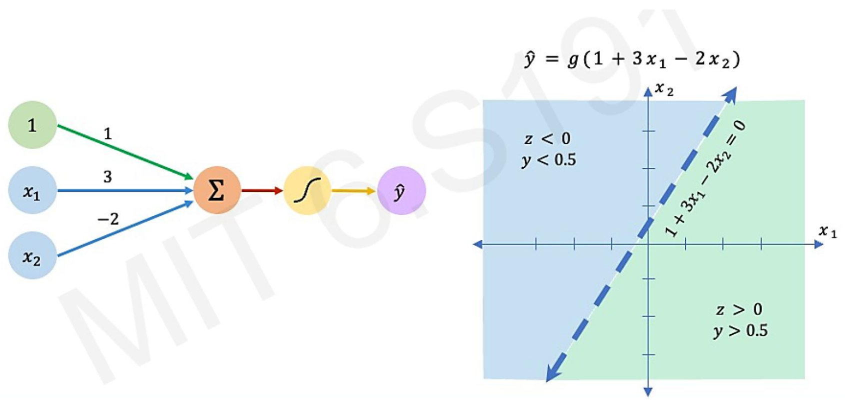
The Perceptron Example

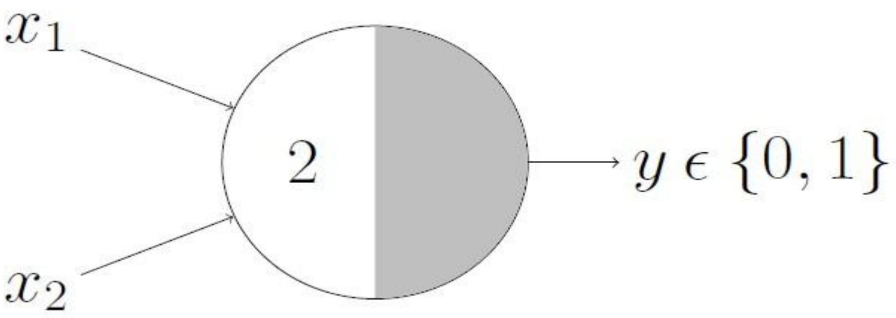


The Perceptron Example



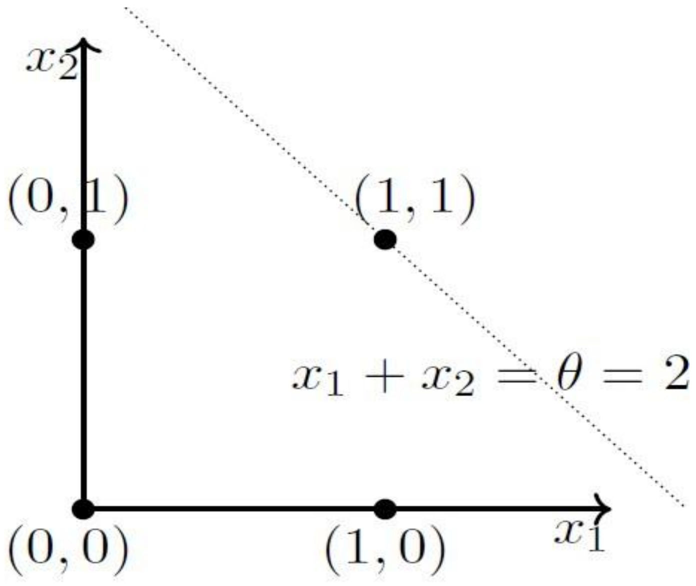
Boolean Functions



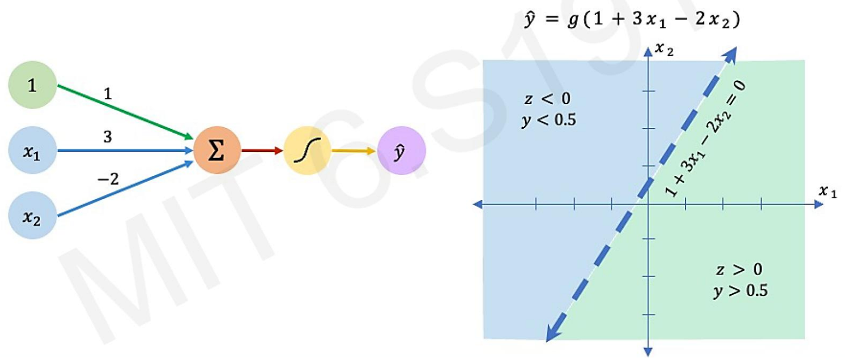


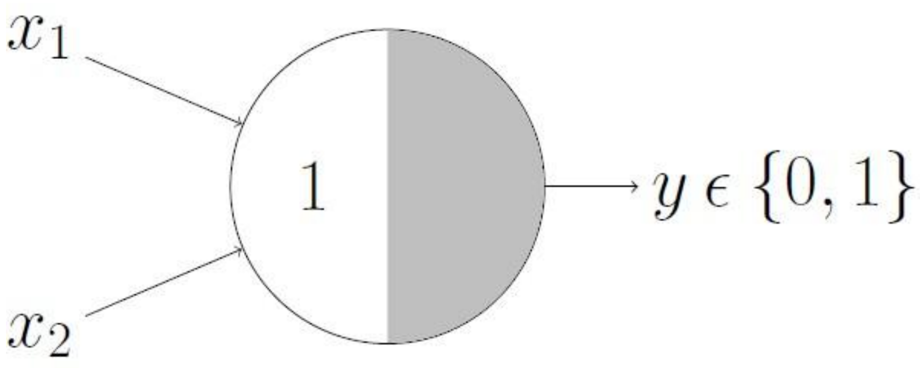
AND function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$



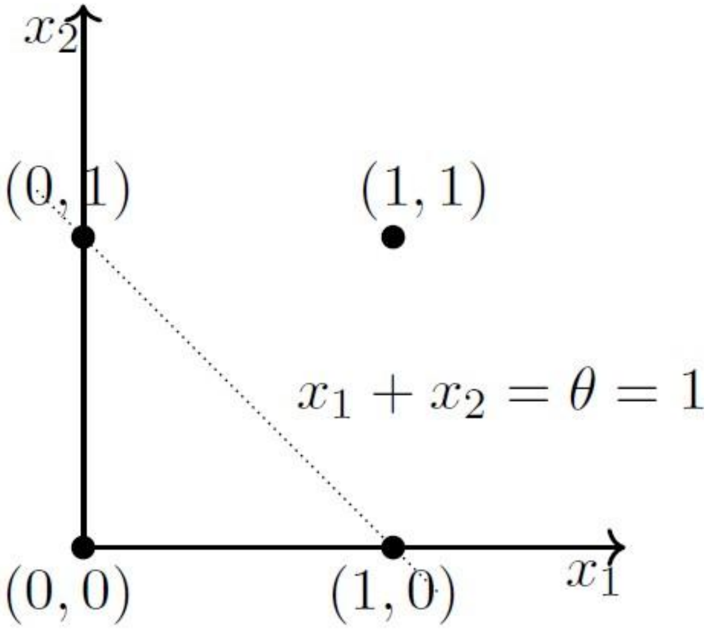
Boolean Functions





OR function

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$



Classification with Logistic Regression

A Document

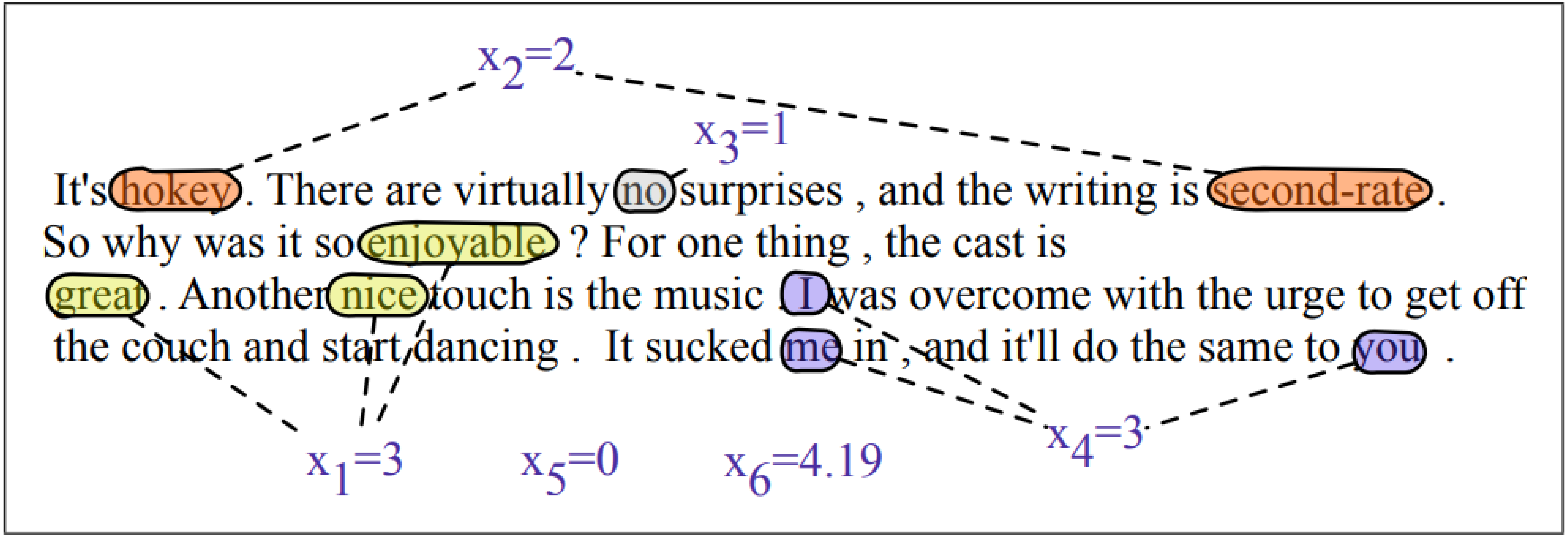


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\ln(\text{word count of doc})$	$\ln(66) = 4.19$

Weights and Biases

- Let’s assume the 6 weights corresponding to the 6 features are

```
w = [2.5,-5.0,-1.2,0.5,2.0,0.7]
```

```
b = [0.1]
```

The weight w_i represents how important that input feature is to the classification decision

Classification with Logistic Regression

$$
\begin{aligned}
p(+|x) = P(y=1|x) &= \sigma(\mathbf{w}\cdot\mathbf{x}+b) \\
&= \sigma([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1) \\
&= \sigma(.833) \\
&= 0.70 \\
p(-|x) = P(y=0|x) &= 1-\sigma(\mathbf{w}\cdot\mathbf{x}+b) \\
&= 0.30
\end{aligned}
$$

Processing many examples at once

$$\textbf{foreach} \quad x^{(i)} \quad \text{in input } [x^{(1)}, x^{(2)}, \ldots, x^{(m)}]$$
$$y^{(i)} = \sigma(\mathbf{w} \cdot \mathbf{x}^{(\mathbf{i})} + b)$$

$$\begin{aligned}
P(y^{(1)} = 1|x^{(1)}) &= \sigma(\mathbf{w} \cdot \mathbf{x}^{(\mathbf{1})} + b) \\
P(y^{(2)} = 1|x^{(2)}) &= \sigma(\mathbf{w} \cdot \mathbf{x}^{(\mathbf{2})} + b) \\
P(y^{(3)} = 1|x^{(3)}) &= \sigma(\mathbf{w} \cdot \mathbf{x}^{(\mathbf{3})} + b)
\end{aligned}$$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_f^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_f^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \ldots & x_f^{(3)} \\ \ldots & & & \end{bmatrix}$$

$$\hat{y}^{(1)} = [x_1^{(1)}, x_2^{(1)}, \ldots, x_f^{(1)}] \cdot [w_1, w_2, \ldots, w_f] + b$$

$$\underset{(m \times 1)}{\mathbf{y}} = \underset{(m \times f)}{\mathbf{X}} \; \underset{(f \times 1)}{\mathbf{w}} + \underset{(m \times 1)}{\mathbf{b}}$$

Cost Function

Cost Function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ ?

Cost Function

$L(\hat{y}, y)$ = How much $\hat{y}$ differs from the true y.

A good cost function?

1. Punish incorrect answers with high cost

y=1, $h\theta(x) \longrightarrow 0$

y=0, $h\theta(x) \longrightarrow 1$

2. Reward correct answers with a low cost

y=1, $h\theta(x) \longrightarrow 1$

y=0, $h\theta(x) \longrightarrow 0$

The cross-entropy loss function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

L(ŷ, y) = How much ŷ differs from the true y.

- Rewards $\mathrm{y} = 1, h_{\theta}(x)$ = 1, with cost = 0
- Rewards $\mathrm{y} = 1, h_{\theta}(x)$ → 0 even further with asymptotically smaller costs

y=1

cost = $\log(h_{\theta}(x))$

- Rewards $\mathrm{y} = 1, h_{\theta}(x)$ = 1, with cost = 0
- Punishes $\mathrm{y} = 1, h_{\theta}(x)$ → 0 with asymptotically higher costs

y=1

cost = $-\log(h_{\theta}(x))$

- Rewards $\mathrm{y} = 0, h_\theta\,(x)$ = 0, with cost = 0
- Rewards $\mathrm{y} = 0, h_\theta\,(x)$ → 1 even further with asymptotically smaller costs

y=0

cost = $\log(1 - h_\theta\,(x))$

- Rewards $\mathrm{y} = 0, h_\theta\,(x)$ = 0, with cost = 0
- Punishes $\mathrm{y} = 0, h_\theta\,(x)$ → 1 with asymptotically higher costs

y=0

cost = $-\log(1 - h_\theta\,(x))$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$$\boldsymbol{cost(h_\theta(x), y) = -y\, log\big(h_\theta(x)\big) - (1-y)\, log\big(1-h_\theta(x)\big)}$$

$$L_{\mathrm{CE}}(\hat{y}, y) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1-y) \log (1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

To fit parameters θ:

$$\min_\theta J(\theta)$$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ gives $p(y = 1|x; \theta)$

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y},y) &= -[y\log\sigma(\mathbf{w}\cdot\mathbf{x}+b)+(1-y)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \\
&= -[\log\sigma(\mathbf{w}\cdot\mathbf{x}+b)] \\
&= -\log(.70) \\
&= .36
\end{aligned}
$$

$$
\begin{aligned}
L_{\mathrm{CE}}(\hat{y},y) &= -[y\log\sigma(\mathbf{w}\cdot\mathbf{x}+b)+(1-y)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \\
&= -[\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}+b))] \\
&= -\log(.30) \\
&= 1.2
\end{aligned}
$$

Gradient Descent

Gradient Descent

The goal is to find the set of weights which minimizes the loss function, averaged over all examples:

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta), y^{(i)})$$

Want $\min_\theta J(\theta)$:

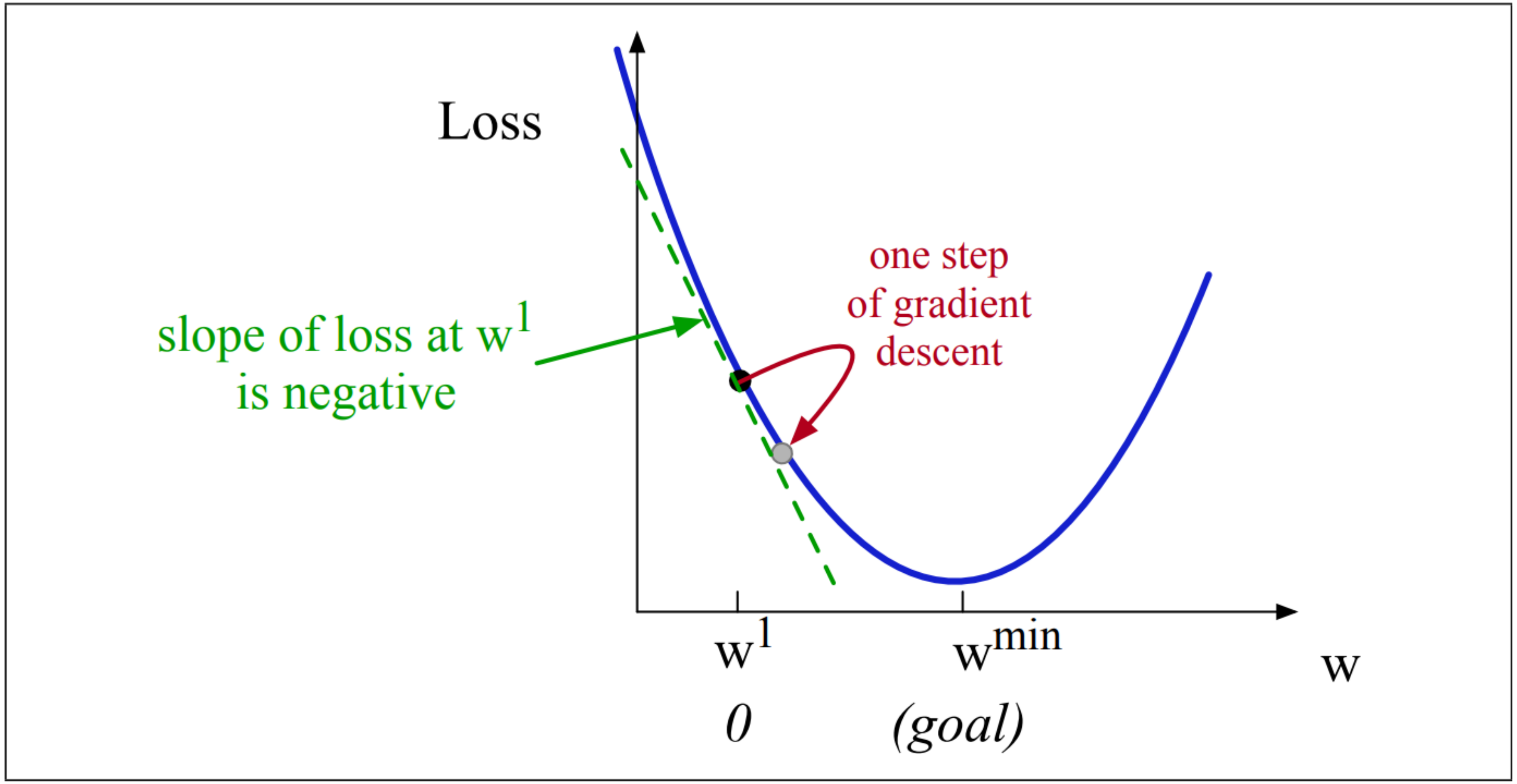
Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\}$ (simultaneously update all θ_j)

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

Gradient Descent



function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ
where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$

$\theta \leftarrow 0$ # (or small random values)
repeat til done   # see caption
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
    1. Optional (for reporting):   # How are we doing on this tuple?
      Compute $\hat{y}^{(i)} = f(x^{(i)};\theta)$ # What is our estimated output $\hat{y}$?
      Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
    2. $g \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$ # How should we move θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead
return θ

Batch Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\}$ (simultaneously update all θ_j)

$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y] x_j$$
$$= (\hat{y} - y) x_j$$

(see SLP3 for derivation)

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(x)^{(i)} - y^{(i)}\right) x^{(i)}$$

Batch Gradient Descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

$$\nabla L(f(x;\boldsymbol{\theta}), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\boldsymbol{\theta}), y) \\ \frac{\partial}{\partial w_2} L(f(x;\boldsymbol{\theta}), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\boldsymbol{\theta}), y) \\ \frac{\partial}{\partial b} L(f(x;\boldsymbol{\theta}), y) \end{bmatrix}$$

Algorithm looks identical to linear regression!

Gradient Descent Example

$$
\begin{aligned}
x_1 &= 3 \quad \text{(count of positive lexicon words)} \\
x_2 &= 2 \quad \text{(count of negative lexicon words)}
\end{aligned}
$$

$$
\begin{aligned}
w_1 = w_2 = b &= 0 \\
\eta &= 0.1
\end{aligned}
$$

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$$

$$\nabla_{w,b}L = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(\mathbf{w}\cdot\mathbf{x}+b)-y)x_1 \\ (\sigma(\mathbf{w}\cdot\mathbf{x}+b)-y)x_2 \\ \sigma(\mathbf{w}\cdot\mathbf{x}+b)-y \end{bmatrix} = \begin{bmatrix} (\sigma(0)-1)x_1 \\ (\sigma(0)-1)x_2 \\ \sigma(0)-1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

So, after one step of gradient descent, the weights have shifted to be: w1 = .15, w2 = .1, and b = .05

Softmax

Softmax

- For a vector z of dimensionality K, the softmax is defined as:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \quad 1 \leq i \leq K$$

- The softmax of an input vector $z = [z_1, z_2, ..., z_K]$ is thus a vector itself:

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, ..., \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

Softmax

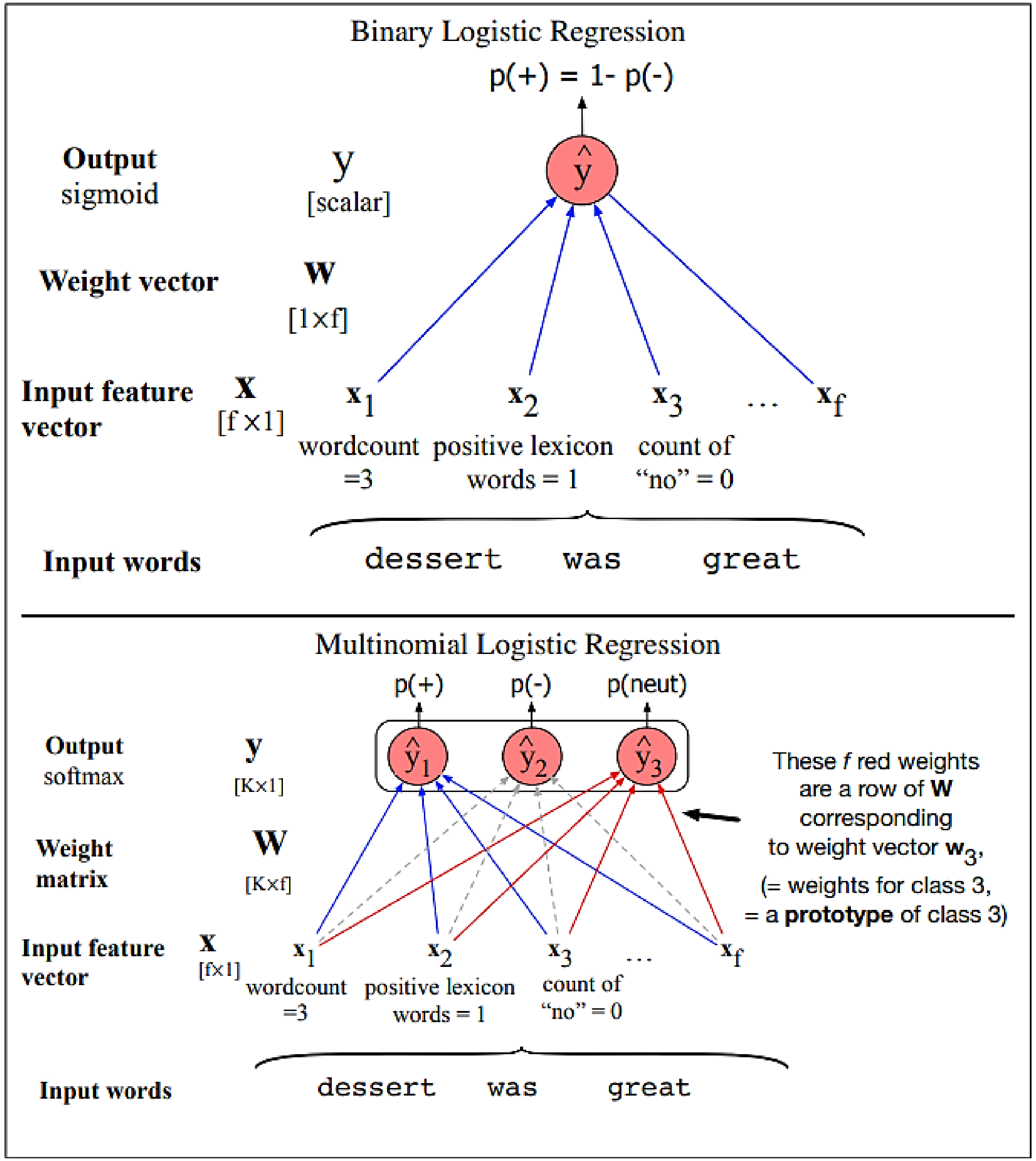
$$\mathrm{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K}\exp(z_i)}, \ldots, \frac{\exp(z_K)}{\sum_{i=1}^{K}\exp(z_i)}\right]$$

$$\mathbf{z} = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

The resulting (rounded) softmax(z) is:

$$[0.05, 0.09, 0.01, 0.1, 0.74, 0.01]$$

A neural network can be viewed as a series of logistic regression classifiers stacked on top of each other



Sources

- https://web.stanford.edu/~jurafsky/slp3/2.pdf
- https://web.stanford.edu/~jurafsky/slp3/3.pdf
- **Machine Learning for Intelligent Systems**, Kilian Weinberger, Cornell, Lectures 3-6, https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecture note03.html
- **Prof. Mitesh M. Khapra** (https://www.cse.iitm.ac.in/~miteshk/) on NPTEL's (http://nptel.ac.in/) Deep Learning course (https://onlinecourses.nptel.ac.in/noc18_cs41/preview)
- **Perceptrons. An Introduction to Computational Geometry. Marvin Minsky and Seymour Papert. M.I.T. Press, Cambridge, Mass., 1969.** https://science.sciencemag.org/content/165/3895/780