

# Neural Network: Logistic Regression

Text and Web Mining (IS6751)

School of Communication and Information

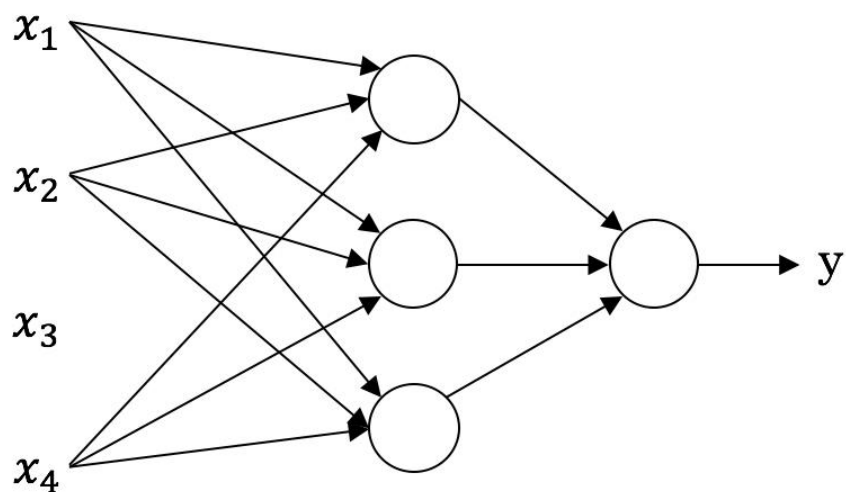
# Overview

- Introduce example applications of neural networks.
- Introduce Scalars, Vectors, and Matrixes and Matrix Operations.
- Introduce Logistic Regression (one neuron network) in detail.
  - Cover foundation concepts in Deep Learning

# Applications of Neural Networks

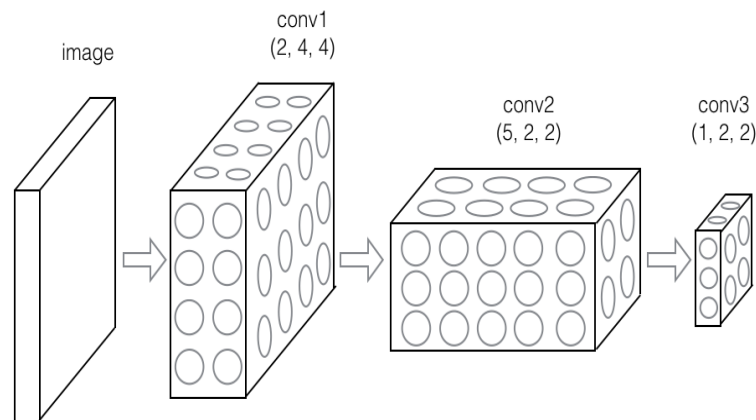
Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

# Neural Network examples



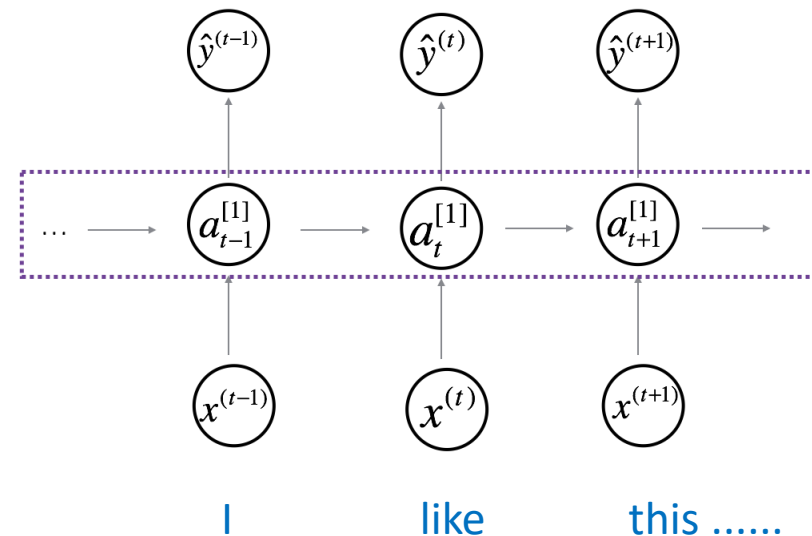
Standard NN

E.g.,  
input: a document vector  
output: sentiment polarity of the document.



Convolutional NN

E.g.,  
input: an image matrix  
output: category of the image.



Recurrent NN

E.g.,  
input: a sequence of words in a document  
output: sentiment polarity of the document

# Intro to Scalars, Vectors, and Matrixes: **Scalars**

- A scalar is a single number.
- Integers ( $a \in \mathbb{N}$ , e.g., 10), real numbers ( $x \in \mathbb{R}$ , e.g., 10.5), etc.
- We denote it with **italic** font:  $a, x$

# Vectors

- A vector is a 1-D array of numbers (a column vector):

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

- Can be integer, real, etc.
- Example notation for type and size:

$$\mathbf{x} = \mathbb{R}^n$$

- We denote it with **italic bold** font:  $\mathbf{a}$ ,  $\mathbf{x}$

[1, 2]

Note that a column vector is represented as a row vector in PyTorch.



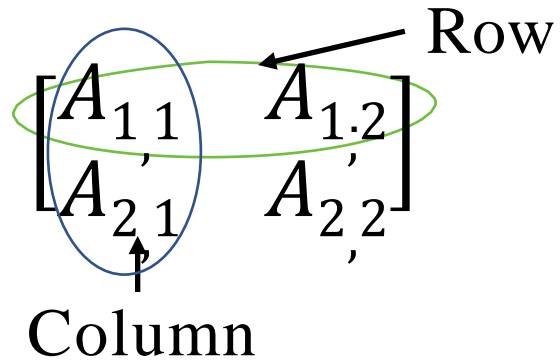
$$\mathbf{x} = \mathbb{R}^2$$

or

$$\mathbf{x} = \mathbb{N}^2$$

# Matrices

- A matrix is a 2-D array of numbers:



- Example notation for type and shape:

$$A = \mathbb{R}^{m \times n}$$

- We denote it with **uppercase letter with italic bold** font:  $A$ ,  $X$
- Vectors can be thought of as matrices that contain only one column.

Matrix  
representation in  
PyTorch:  
`[[1., 2.],  
 [3., 4.]]`



$$A = \mathbb{R}^{2 \times 2}$$

Matrix  
representation  
in PyTorch:  
`[[1.],  
 [2.]]`



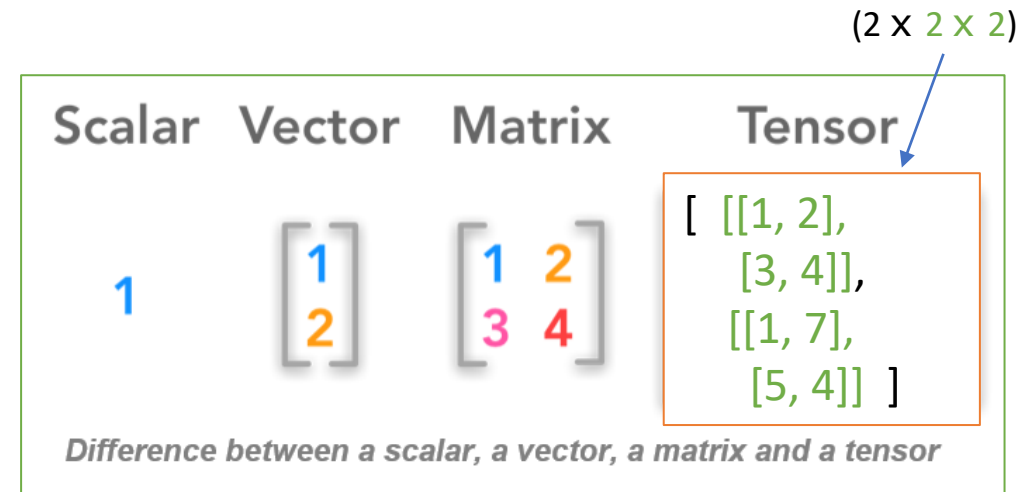
$$A = \mathbb{R}^{2 \times 1}$$

# Tensors

- A tensor is an array of numbers, that may have

- zero dimension, and be a scalar
- one dimension, and be a vector
- two dimensions, and be a matrix
- or more dimensions.

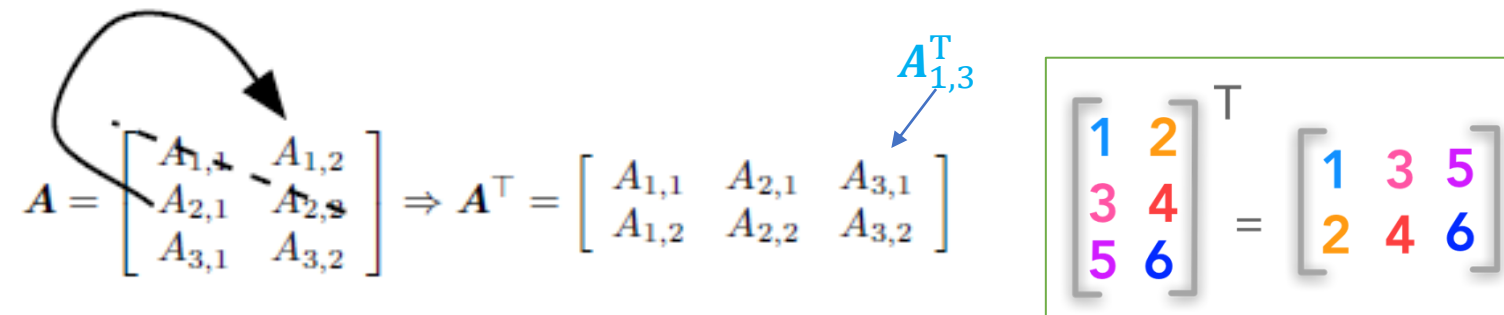
- While, technically, scalar, vector, and matrix are valid tensors, when we speak of tensors, we are generally speaking of the generalization of the concept of a matrix to  $N \geq 3$  dimensions.





# Matrix Transpose

$$(A^T)_{i,j} = A_{j,i}. \quad (2.3)$$



$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Figure 2.1: The transpose of the matrix can be thought of as a mirror image across the main diagonal.

# Matrix Addition

Matrices can be added if they have the same shape:

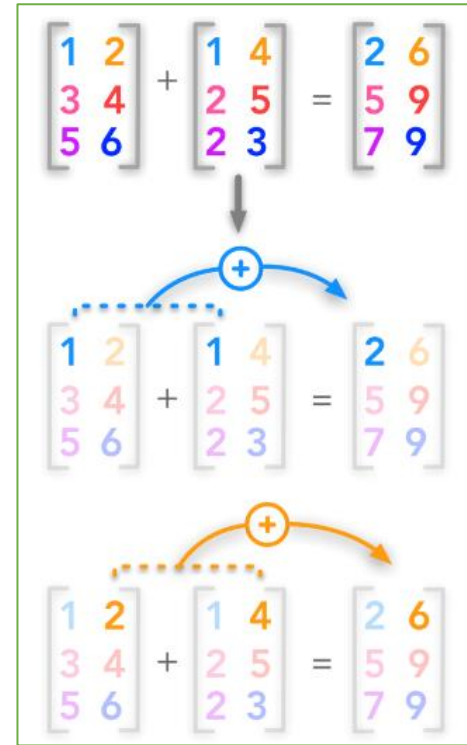
$$\mathbf{A} + \mathbf{B} = \mathbf{C}$$

Each cell of  $\mathbf{A}$  is added to the corresponding cell of  $\mathbf{B}$ :

$$\mathbf{A}_{ij} + \mathbf{B}_{ij} = \mathbf{C}_{ij}$$

$i$  is the row index and  $j$  the column index.

$$\begin{array}{ccc} \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} & + & \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \\ B_{3,1} & B_{3,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} + B_{1,1} & A_{1,2} + B_{1,2} \\ A_{2,1} + B_{2,1} & A_{2,2} + B_{2,2} \\ A_{3,1} + B_{3,1} & A_{3,2} + B_{3,2} \end{bmatrix} \\ (3 \times 2) & (3 \times 2) & (3 \times 2) \end{array}$$



# Broadcasting

- PyTorch and Numpy (a Python library for handling multi-dimensional arrays and matrices) can handle operations on arrays of different shapes.
- The smaller array will be extended to match the shape of the bigger one.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

$X: (3 \times 4)$        $Y: (3 \times 4)$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

$Z: (3 \times 4)$

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} + \begin{bmatrix} B_{1,1} \\ B_{2,1} \\ B_{3,1} \end{bmatrix} \quad (3 \times 2) + (3 \times 1)$$

Note:  $(3 \times 2) + (2 \times 1)$  does not work.

is equivalent to  $(3 \times 2) + (3 \times 2)$

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} + \begin{bmatrix} B_{1,1} & B_{1,1} \\ B_{2,1} & B_{2,1} \\ B_{3,1} & B_{3,1} \end{bmatrix} = \begin{bmatrix} A_{1,1} + B_{1,1} & A_{1,2} + B_{1,1} \\ A_{2,1} + B_{2,1} & A_{2,2} + B_{2,1} \\ A_{3,1} + B_{3,1} & A_{3,2} + B_{3,1} \end{bmatrix}$$

where the  $(3 \times 1)$  matrix is converted to the right shape  $(3 \times 2)$  by copying the first column. Numpy will do that automatically if the shapes can match.

```
x = torch.arange(12).view(3, 4)
y = torch.arange(4).view(1, 4)
z = torch.arange(3).view(3, 1)

print(x)
print(y)
print(z)
print(x + y)
print(x + z)

tensor([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]])
tensor([[0, 1, 2, 3]])
tensor([[0],
        [1],
        [2]])
tensor([[ 0,  2,  4,  6],
        [ 4,  6,  8, 10],
        [ 8, 10, 12, 14]])
tensor([[ 0,  1,  2,  3],
        [ 5,  6,  7,  8],
        [10, 11, 12, 13]])
```

# Matrix (Dot) Product

- The way to multiply matrices is to calculate the sum of the products between rows and columns.
- The matrix product, also called **dot product**, is calculated as follows:

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix} \times \begin{bmatrix} G \\ H \end{bmatrix} = \begin{bmatrix} A \times G + B \times H \\ C \times G + D \times H \\ E \times G + F \times H \end{bmatrix}$$

$(3 \times 2) \quad \times \quad (2 \times 1) = \quad (3 \times 1)$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 2 \\ 4 \end{bmatrix} =$$
$$\begin{bmatrix} 1 \times 2 + 2 \times 4 \\ 3 \times 2 + 4 \times 4 \\ 5 \times 2 + 6 \times 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 22 \\ 34 \end{bmatrix}$$

$$(3 \times 2) \times (2 \times 1) = (3 \times 1)$$

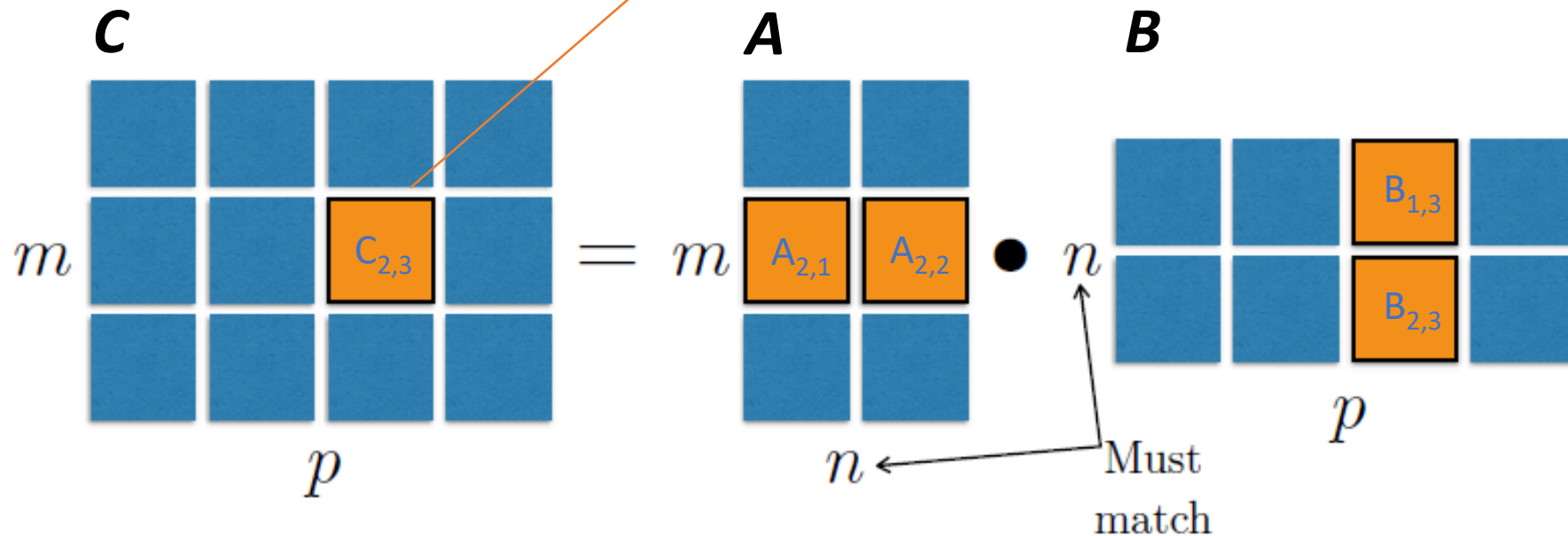
# Matrix (Dot) Product

$$C = AB.$$

The sum from  $K$  equals 1 to  $n$ , which is 2 in this example.

$$C_{i,j} = \sum_{k=1}^n A_{i,k} B_{k,j}$$

$$C_{2,3} = \sum_{k=1}^2 A_{2,k} B_{k,3} = A_{2,1} B_{1,3} + A_{2,2} B_{2,3}$$



- The number of columns of the first matrix must be equal to the number of rows of the second matrix. Thus, if the dimensions of the first matrix is  $(m \times n)$ , the second matrix need to be of shape  $(n \times p)$ . The resulting matrix will have the shape  $(m \times p)$ .

# Element-wise (Hadamard) Product

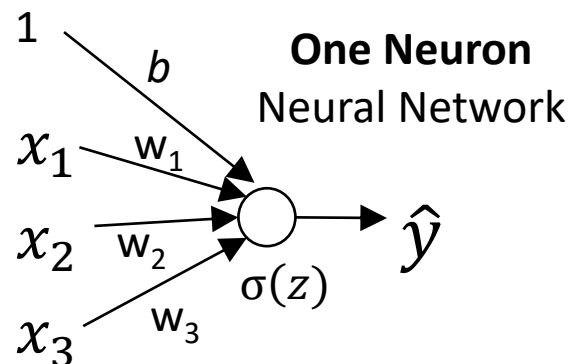
- The product of the individual elements is called the **element-wise product** or **Hadamard product**.
- For two matrices  $\mathbf{A}$  and  $\mathbf{B}$  of the same shape  $m \times n$ , the Hadamard product  $\mathbf{A} \circ \mathbf{B}$  (or  $\mathbf{A} \odot \mathbf{B}$ ) is a matrix of the same shape as the operands.
- For example, the Hadamard product for a  $3 \times 3$  matrix  $\mathbf{A}$  with a  $3 \times 3$  matrix  $\mathbf{B}$  is.

$$\begin{array}{ccc} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} & \circ & \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} b_{11} & a_{12} b_{12} & a_{13} b_{13} \\ a_{21} b_{21} & a_{22} b_{22} & a_{23} b_{23} \\ a_{31} b_{31} & a_{32} b_{32} & a_{33} b_{33} \end{bmatrix} \\ (3 \times 3) & & (3 \times 3) \end{array}$$

# Logistic Regression

E.g., The probability of Positive Class given a review document.

- Models the **probability** that an input belongs to a particular category.
  - Given  $\mathbf{x}$ , want to predict  $\hat{y} = P(y = 1 \mid \mathbf{x})$ , where  $0 \leq \hat{y} \leq 1$
  - $\mathbf{x} \in \mathbb{R}^{n_x}$ , where  $\mathbf{x} = x_1, x_2, \dots, x_{n_x}$
  - Parameters:  $\mathbf{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}$
  - $z = \mathbf{w}^T \mathbf{x} + b$
  - Output:  $\hat{y} = \sigma(z)$  where  $\sigma()$  is sigmoid function.



input      output

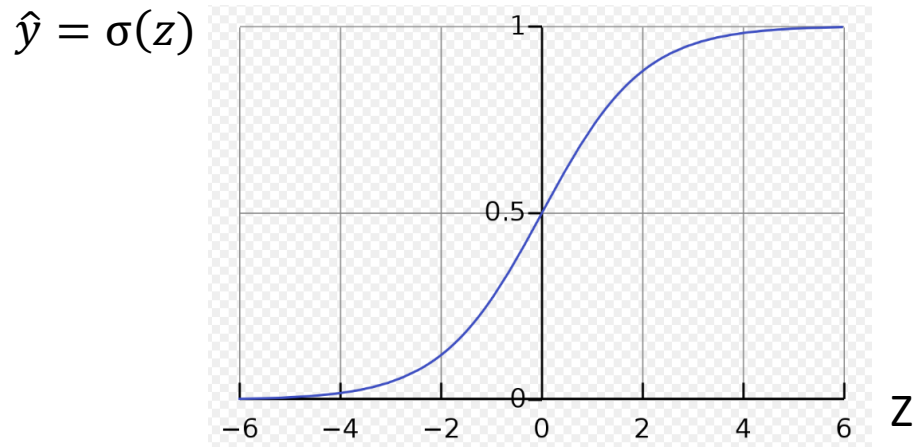
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$
$$\mathbf{w}^T = [w_1 \ w_2 \ w_3]$$

(1,3) (3,1)

$$z = [w_1 \ w_2 \ w_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b$$
$$= w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

# Logistic Regression

- Models the **probability** that an input belongs to a particular category.
  - Given  $\mathbf{x}$ , want to predict  $\hat{y} = P(y = 1 | \mathbf{x})$ , where  $0 \leq \hat{y} \leq 1$
  - $z = \mathbf{w}^T \mathbf{x} + b$
  - Output:  $\hat{y} = \sigma(z)$  where  $\sigma()$  is sigmoid function.



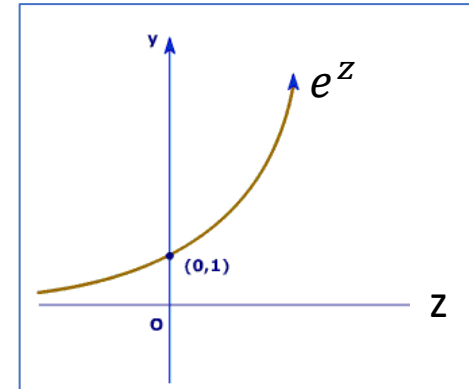
$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \frac{1}{e^z}}$$

If  $z$  is large,  $\sigma(z) \approx \frac{1}{1+0} = 1$

If  $z$  is a large negative value,  $\sigma(z) \approx \frac{1}{1+\infty} = 0$

If  $z$  is 0,  $\sigma(z) = \frac{1}{1+1} = 0.5$

In general, if  $\hat{y} \geq 0.5$ , predict positive class, otherwise negative class.





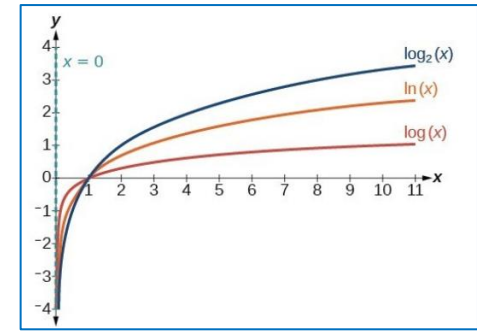
# Logistic Regression cost function

Given  $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$ , want  $\hat{y}^{(i)} \approx y^{(i)}$ . E.g.,  $\{(\mathbf{x}^{(1)}, 0), (\mathbf{x}^{(2)}, 1), \dots, (\mathbf{x}^{(m)}, 1)\}$

$$\hat{y}^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}, \text{ where } z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$$

**Loss (error) function:** *minimize* the loss function,  
called **Cross Entropy Loss Function** (or *negative log likelihood*)

$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$



If  $y^{(i)}=1$ ,  $\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$  : want  $\log(\hat{y}^{(i)})$  large, want  $\hat{y}^{(i)}$  large (close to 1)

If  $y^{(i)}=0$ ,  $\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$  : want  $\log(1 - \hat{y}^{(i)})$  large, want  $\hat{y}^{(i)}$  small (close to 0)

**Cost function:** find  $\mathbf{w}$  and  $b$  values that minimize the cost function

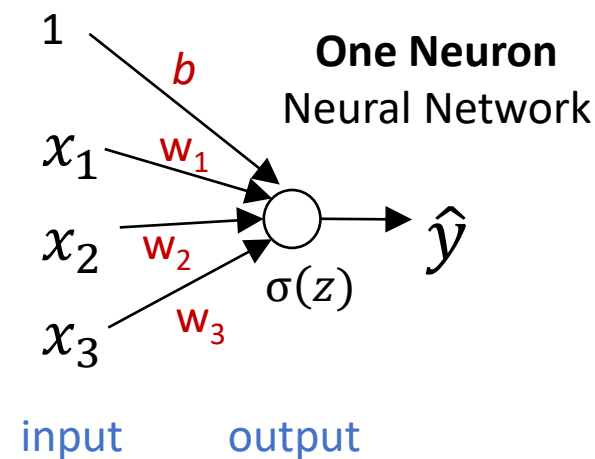
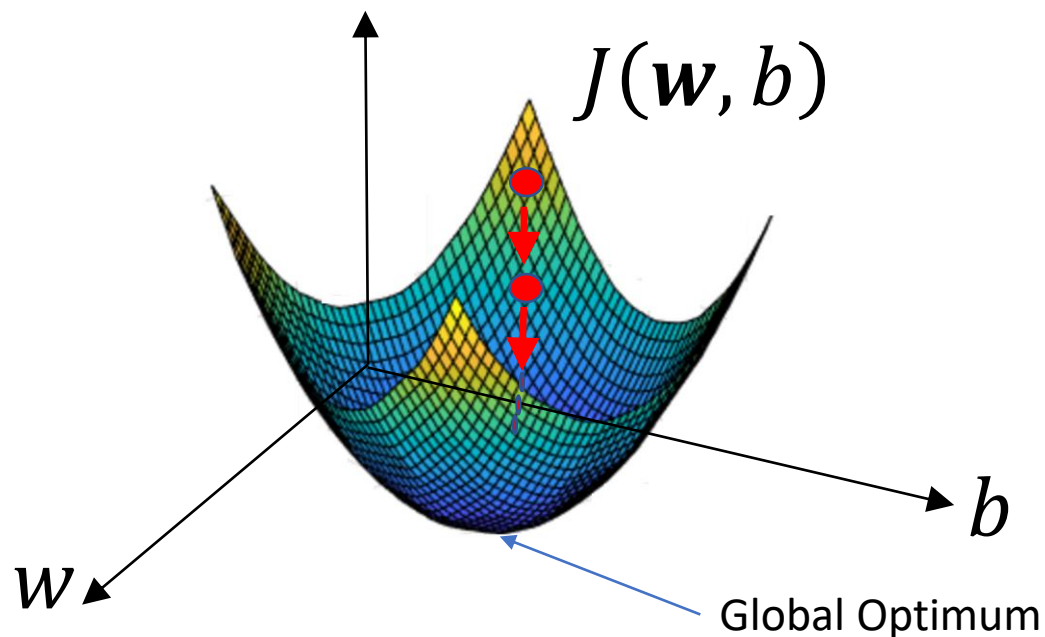
$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

# Gradient Descent

Recap:  $\hat{y}^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$ , where  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

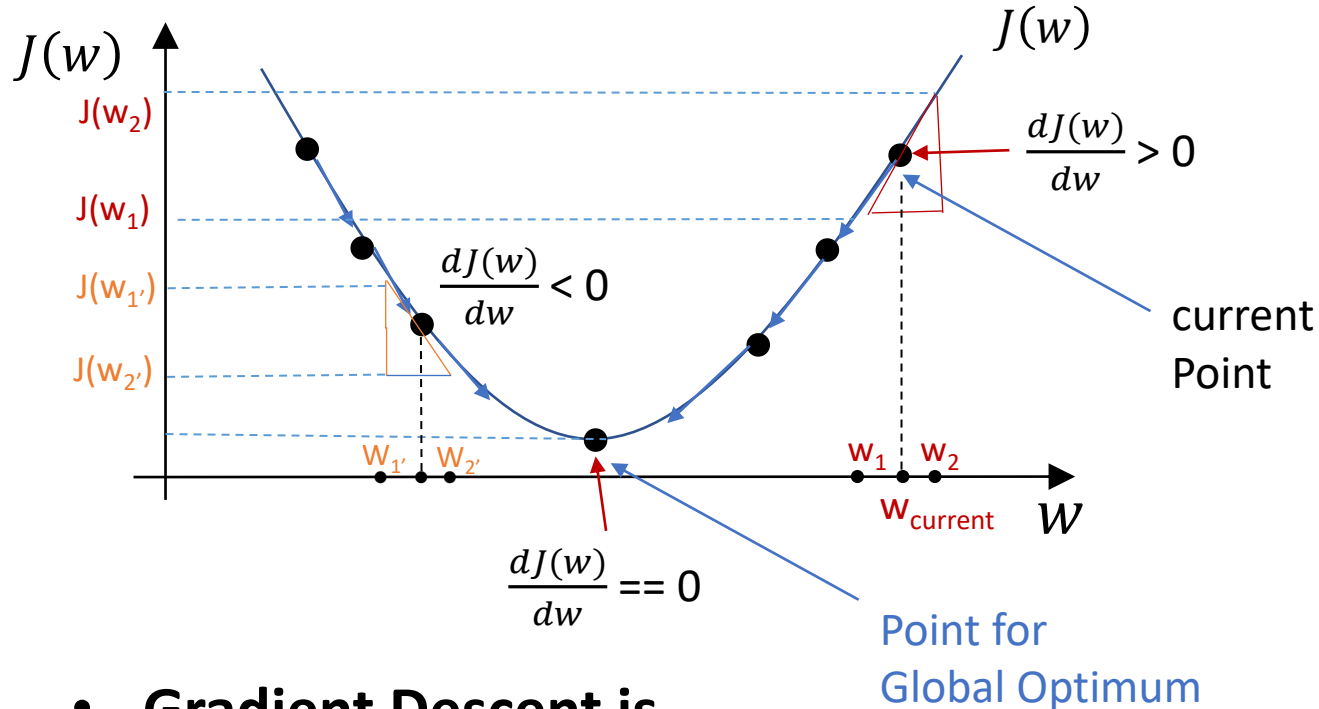
Want to find  $\mathbf{w}, b$  that minimize  $J(\mathbf{w}, b)$



# Gradient Descent

$$\begin{aligned}\text{Slope} &= \text{Height} / \text{Width} \\ &= J(w_2) - J(w_1) / w_2 - w_1\end{aligned}$$

When slope is positive, decrease  $w$  value, and vice versa.



- **Gradient Descent is an iterative optimization algorithm for finding the minimum of a function.** It takes steps proportional to the *negative* of the gradient of the function at the current point.

Repeat {

$$w = w - \alpha \frac{dJ(w)}{dw}$$

}

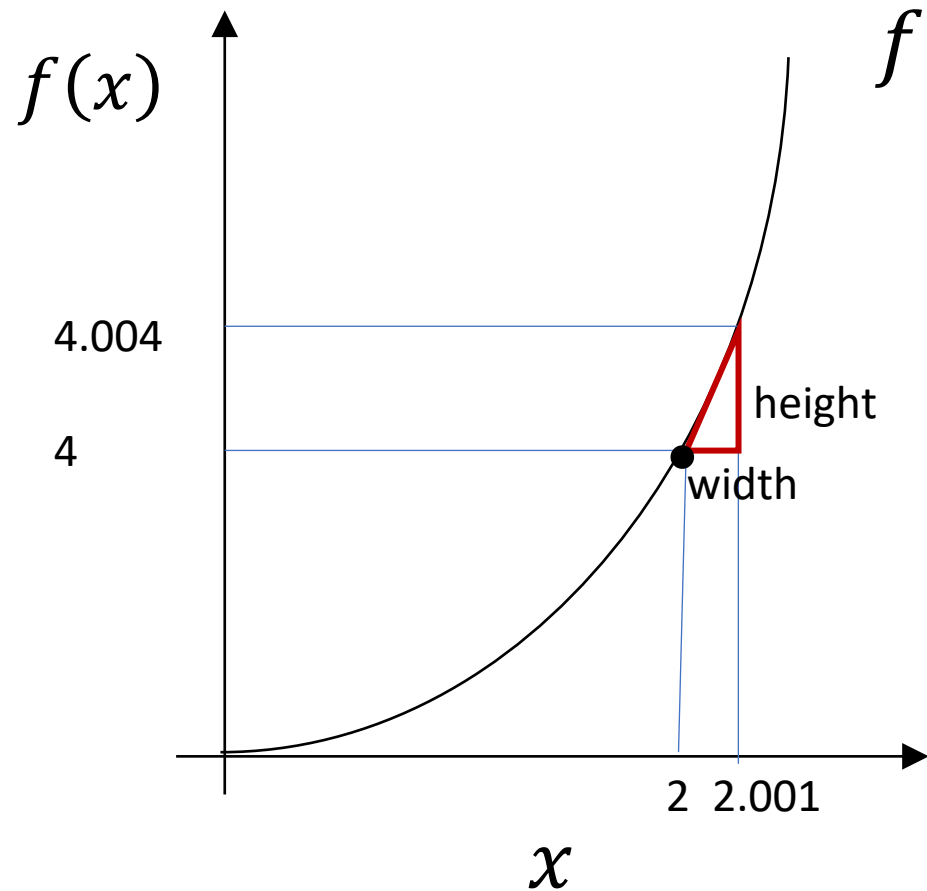
Slope (derivative) of  $J(w)$  with respect to  $w$

Learning rate (e.g., 0.1)

$$w = w - \alpha \frac{dJ(w,b)}{dw}$$

$$b = b - \alpha \frac{dJ(w,b)}{db}$$

# Intuition about derivatives



$$f(x) = x^2$$

$$x = 2; f(x) = 4$$

$$x = 2.001; f(x) \approx 4.004$$

Slope (derivative) of  $f(x)$  at  $x=2$  is

$\frac{0.004}{0.001} = 4$  : So, when  $x$  increases by 0.001,  $f(x)$  increases 0.004. So, the change rate of  $f(x)$  is 4 when  $x$  is 2.

$$\frac{df(x)}{dx} = \frac{dx^2}{dx} = 2x$$

$$F(x)=3x: \frac{df(x)}{dx} = \frac{d3x}{dx} = 3$$

$$F(x)=x^2: \frac{df(x)}{dx} = \frac{dx^2}{dx} = 2x$$

$$F(x)=x^3: \frac{df(x)}{dx} = \frac{dx^3}{dx} = 3x^2$$

$$F(x)=\log x: \frac{df(x)}{dx} = \frac{d\log x}{dx} = \frac{1}{x}$$

Some Basic Derivatives

$$\frac{d}{dx}(c) = 0$$

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(x^n) = nx^{n-1}$$

Another example, the change rate of  $f(x)$  is 8 when  $x$  is 4:

$$\frac{4.001^2 - 4^2}{0.001} = \frac{0.008}{0.001} = 8.$$

# Computation Graph

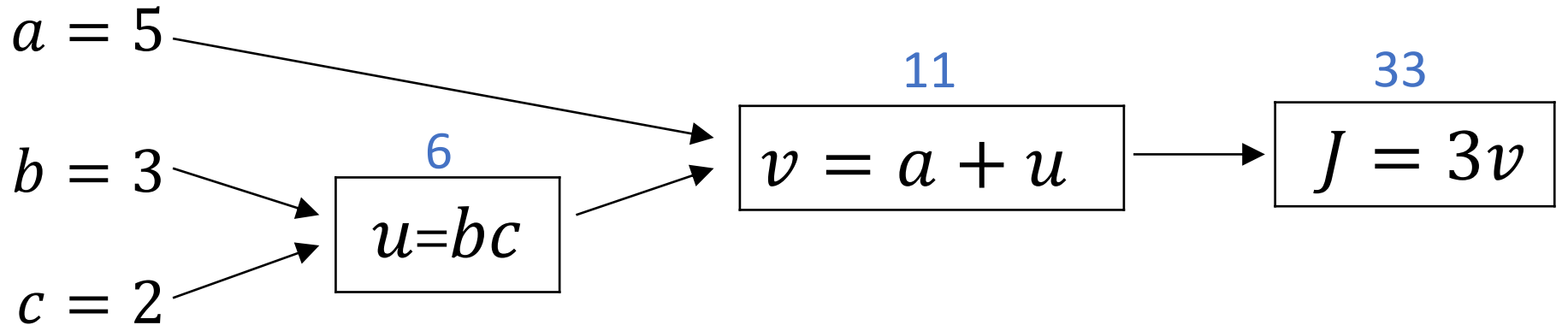
- Used for **computing derivatives** in deep learning platforms, such as PyTorch.

$$J(a, b, c) = 3(a + bc)$$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



$$J(a, b, c) = 3(a + bc) = 3(5 + 3 \cdot 2) = 33$$

# Computing derivatives

**chain rule** example:

$$\frac{dy}{dx} = \frac{dy}{dsomething} \frac{dsomething}{dx}$$

$$\begin{aligned} y &= 5(x^3 + 7)^3 \\ &= 5(\text{something})^3 \\ y' &= 15(\text{something})^2 \times \frac{d(\text{something})}{dx} \\ &= 15(x^3 + 7)^2 \times (3x^2 + 0) \\ &= 45x^2(x^3 + 7)^2 \end{aligned}$$

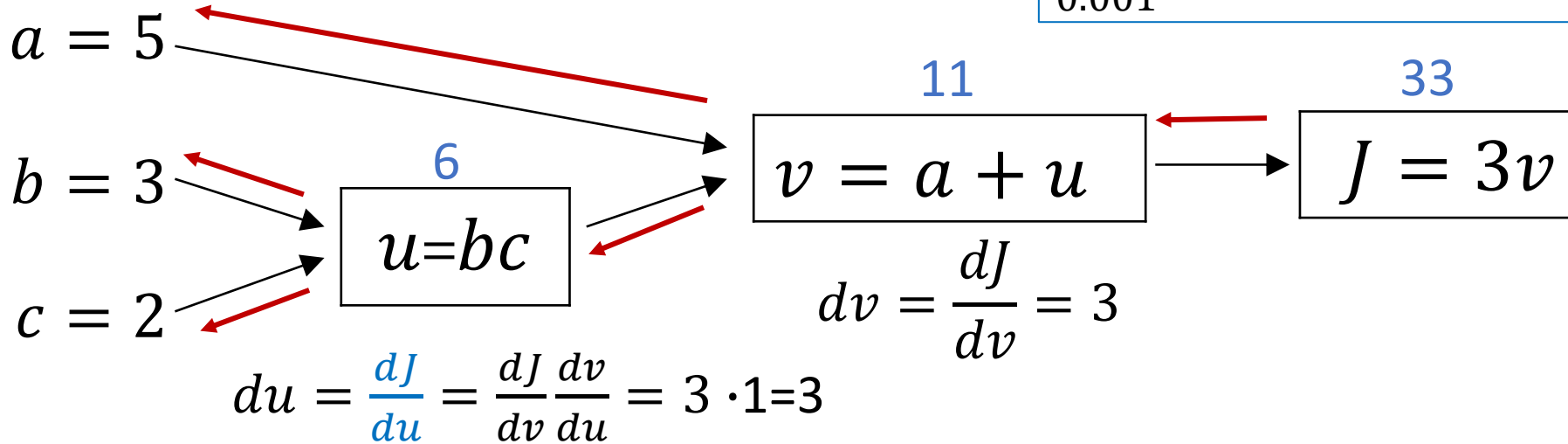
Use chain rule:  $J(a, b, c) = 3(a + bc)$

$$\frac{dJ}{da}$$

When  $a=5$ , get  $v=11$  and  $J=33$   
When  $a=5.001$ , get  $v=11.001$  and  $J=33.003$

So, when  $a$  increases by 0.001,  $J$  increases 0.003.  
 $\frac{0.003}{0.001} = 3$ , so we can say that  $\frac{dJ}{da} = 3$

$$da = \frac{dJ}{da} = \frac{dJ}{dv} \frac{dv}{da} = 3 \cdot 1 = 3$$



$$db = \frac{dJ}{db} = \frac{dJ}{dv} \frac{dv}{du} \frac{du}{db} = \frac{dJ}{du} \frac{du}{db} = 3 \cdot 2 = 6$$

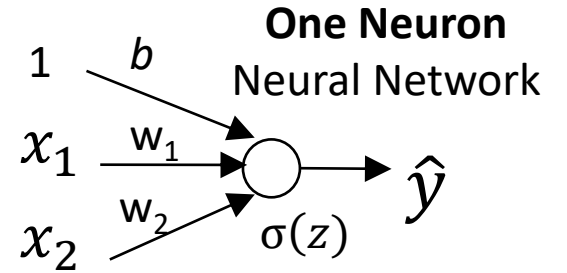
$$dc = \frac{dJ}{dc} = \frac{dJ}{dv} \frac{dv}{du} \frac{du}{dc} = \frac{dJ}{du} \frac{du}{dc} = 3 \cdot 3 = 9$$

# Logistic regression recap

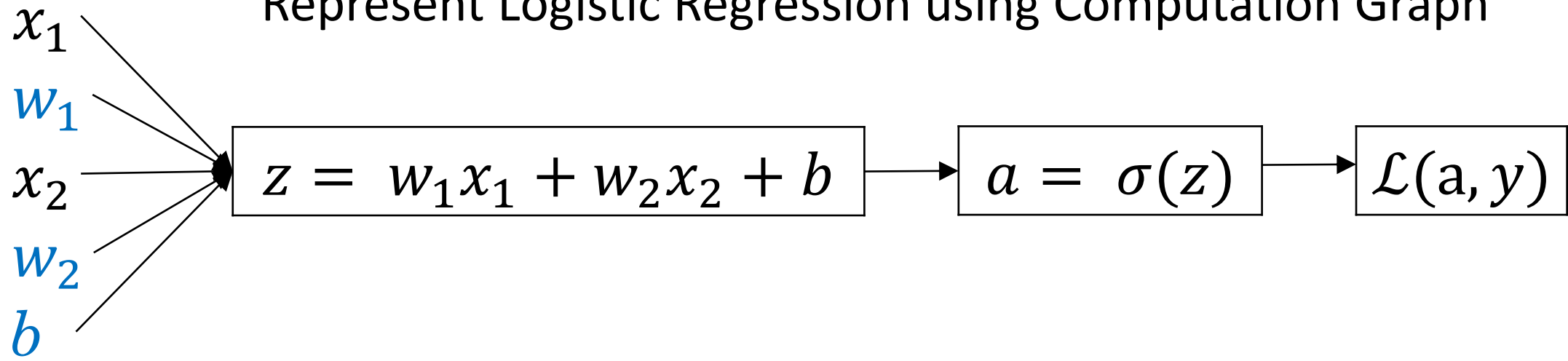
$$z = \mathbf{w}^T \mathbf{x} + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$



Represent Logistic Regression using Computation Graph



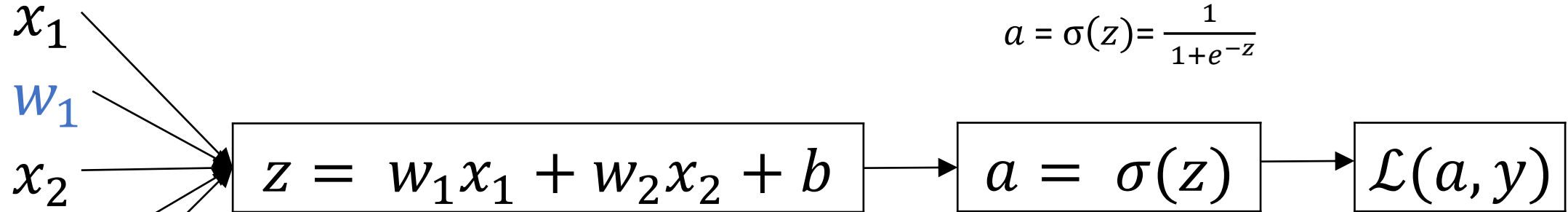
# Logistic regression derivatives

$$F(x)=\log x: \frac{df(x)}{dx} = \frac{d\log x}{dx} = \frac{1}{x}$$

Goal: estimate  $w_1$ ,  $w_2$ , and  $b$ .

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

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



$$\begin{aligned} dz &= \frac{\partial \mathcal{L}(a, y)}{\partial z} \\ &= \frac{\partial \mathcal{L}(a, y)}{\partial a} \cdot \frac{\partial a}{\partial z} \\ &= \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) \cdot a(1-a) \\ &= a - y \end{aligned}$$

$$\begin{aligned} da &= \frac{\partial \mathcal{L}(a, y)}{\partial a} \\ &= -\frac{y}{a} + \frac{1-y}{1-a} \end{aligned}$$

Using Gradient Descent:

$$\begin{aligned} w_1 &= w_1 - \alpha dw_1 \\ w_2 &= w_2 - \alpha dw_2 \\ b &= b - \alpha db \end{aligned}$$

$$\frac{\partial \mathcal{L}(a, y)}{\partial w_1} = dw_1 = \frac{\partial \mathcal{L}(a, y)}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1} = \frac{\partial \mathcal{L}(a, y)}{\partial z} \cdot \frac{\partial z}{\partial w_1} = (a - y) \cdot x_1$$

$$\frac{\partial \mathcal{L}(a, y)}{\partial w_2} = dw_2 = \frac{\partial \mathcal{L}(a, y)}{\partial z} \cdot \frac{\partial z}{\partial w_2} = (a - y) \cdot x_2 \quad \frac{\partial \mathcal{L}(a, y)}{\partial b} = db = \frac{\partial \mathcal{L}(a, y)}{\partial z} \cdot \frac{\partial z}{\partial b} = (a - y) \cdot 1 = (a - y)$$



# Logistic regression on $m$ examples

$$J=0, dw_1=0, dw_2=0, db=0$$

For  $i = 1$  to  $m$

$$z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += - [y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)} \quad \# \text{ use derivatives}$$

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

$$J /= m; dw_1 /= m; dw_2 /= m; db /= m$$

$$J = J / m$$

$epoch = n$  # e.g.,  $n=20$

For  $j = 1$  to  $epoch$

$J=0, dw_1=0, dw_2=0, db=0$

For  $i = 1$  to  $m$

.....

$J /= m; dw_1 /= m; dw_2 /= m; db /= m$

$w_1 = w_1 - \alpha dw_1; w_2 = w_2 - \alpha dw_2$

$b = b - \alpha db$

Update

$$\begin{aligned} w_1 &= w_1 - \alpha dw_1 \\ w_2 &= w_2 - \alpha dw_2 \\ b &= b - \alpha db \end{aligned}$$

# Summary of today's lecture

- Introduced example applications of neural networks.
- Introduced Scalars, Vectors, and Matrixes and Matrix Operations.
- Covered Logistic Regression (one neuron network) in detail.
  - z score (from  $\mathbf{w}^T \mathbf{x} + b$ )
  - Sigmoid function
  - Cost or Loss function
  - Gradient Descent
    - Used for finding  $\mathbf{w}$ ,  $b$  values that minimize the cost function
  - Computation Graph
    - Used for computing derivatives



# Reference

- Neural Network and Deep Learning, Andrew Ng,  
<https://www.coursera.org/learn/neural-networks-deep-learning>
- *Deep Learning*, Ian Goodfellow et al., 2016
  - Chapter 2