Neural Network: Logistic Regression

Text and Web Mining (IS6751)

School of Communication and Information

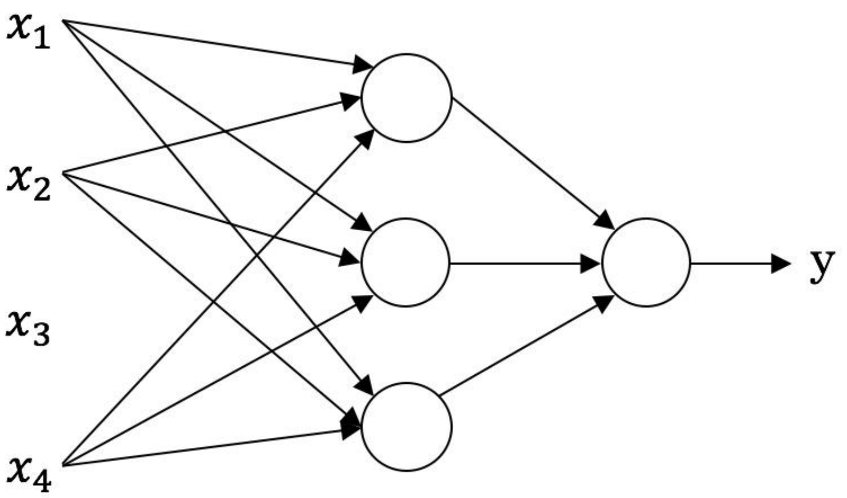
Overview

- 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

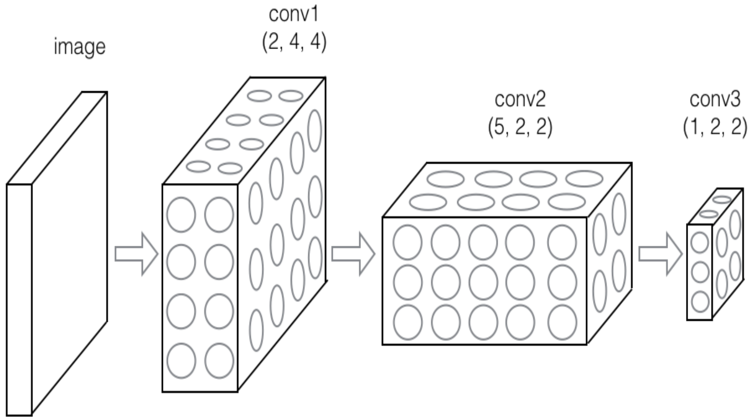
Applications of Neural Networks

Input(x)	Output (y)	Application	
Home features	Price	Real Estate	Standard NN
Ad, user info	Click on ad? (0/1)	Online Advertising	
Image	Object (1,…,1000)	Photo tagging	CNN/ Transformer
Audio	Text transcript	Speech recognition	RNN/ Transformer
English	Chinese	Machine translation	
Image, Radar info	Position of other cars	Autonomous driving 自动驾驶	Custom/ Hybrid

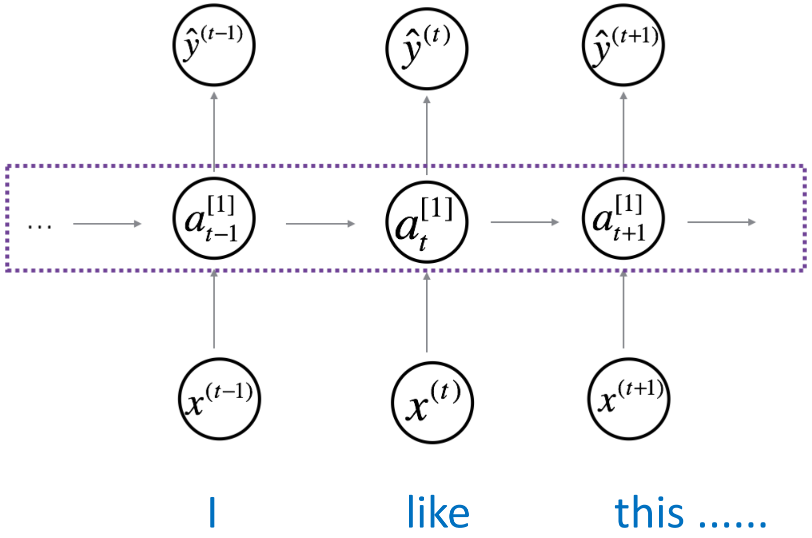
Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

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

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

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 ($a \in \mathrm{N}$, e.g., 10), real numbers ($x \in \mathbb{R}$, e.g., 10.5), etc.

- We denote it with italic font: a, x

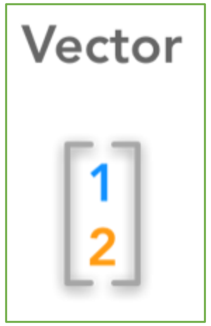
表示

Vectors

[1, 2]
Note that a column vector is represented as a row vector in PyTorch.

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

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$



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

or

$$\boldsymbol{x} = \mathrm{N}^2$$

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

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

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

Matrices

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

Row

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

Column

- Example notation for type and shape:

$$A = \mathbb{R}^{m \text{ x } n}$$

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

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

Matrix

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

$$A = \mathbb{R}^{2 \text{ x } 2}$$

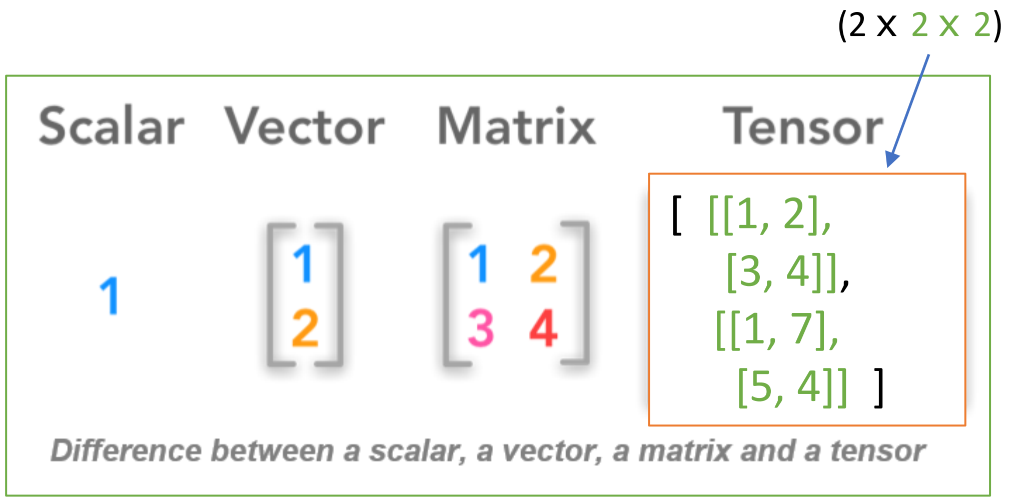
Matrix representation in PyTorch:
[[1.],
[2.]]

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

$$A = \mathbb{R}^{2 \text{ x } 1}$$

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.



Difference between a scalar, a vector, a matrix and a tensor

Matrix Transpose

$$(\boldsymbol{A}^\top)_{i,j} = A_{j,i}. \tag{2.3}$$

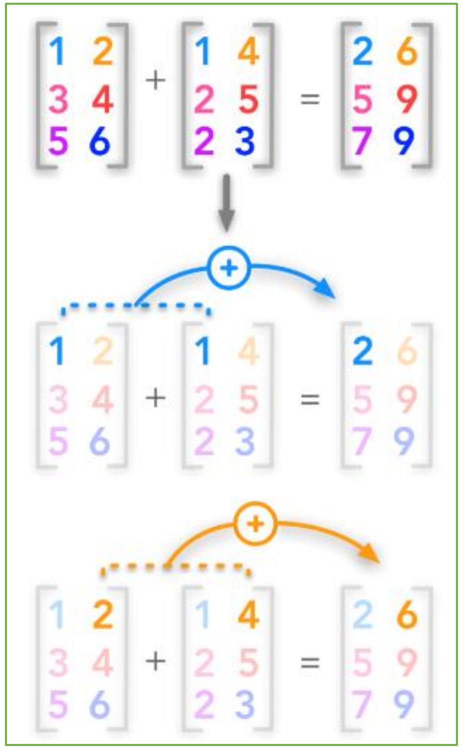
$\boldsymbol{A}^{\mathrm{T}}_{1,3}$

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

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

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:

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

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

$$\boldsymbol{A}_{i,j} + \boldsymbol{B}_{i,j} = \boldsymbol{C}_{i,j}$$

i is the row index and j the column index.

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

(3 x 2) (3 x 2) (3 x 2)

Broadcasting

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

X: (3 x 4) ***Y***: (3 x 4)

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

Z: (3 x 4)

- 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.

- PyTorch 和 Numpy（用于处理多维数组和矩阵的 Python 库）可以处理不同形状的数组上的操作。
- 较小的数组将被扩展以匹配较大数组的形状。

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

(3 x 2) + (3 x 1)

Note: (3 x 2) + (2 x 1) does not work.

is equivalent to (3 x 2) + (3 x 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 (3×1) matrix is converted to the right shape (3×2) by copying the first column. Numpy will do that automatically if the shapes can match.

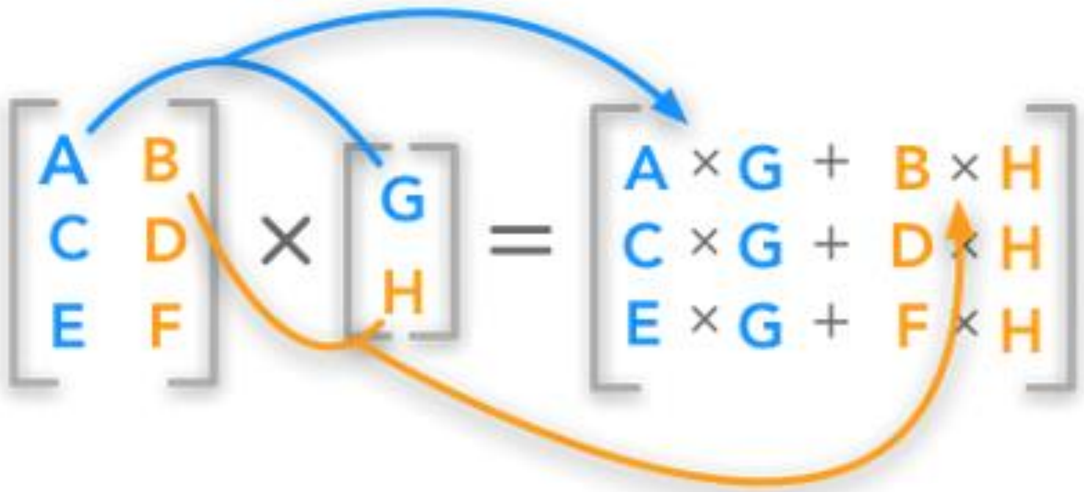
```
x = torch.arange(12).view(3, 4)
y = torch.arange(4).view(1, 4)
z = torch.arange(3).view(3, 1)

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:



(3x2) x (2x1) = (3x1)

Row * Col

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

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

(3x2) x (2x1) = (3x1)

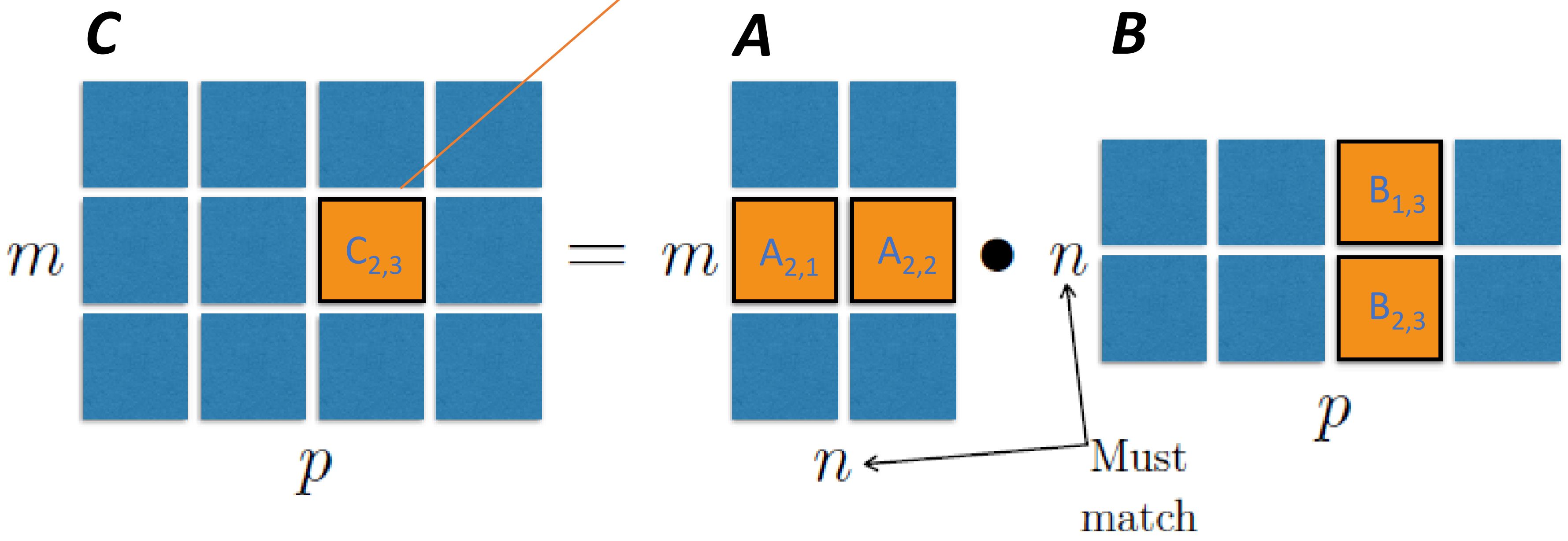
Matrix (Dot) Product

$$C = AB.$$

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

$$C_{i,j} = \sum_{k=1}^{n} \boldsymbol{A}_{i,k} \boldsymbol{B}_{k,j}$$

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



第一个矩阵的列数必须等于第二个矩阵的行数。因此，如果第一个矩阵的尺寸为 (m×n)，则第二个矩阵的形状必须是 (n×p)。结果矩阵的形状将为 (m×p)。

- 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×n), the second matrix need to be of shape (n×p). The resulting matrix will have the shape (m×p).

Element-wise (Hadamard) Product

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

- 各个元素的乘积称为元素乘积或 Hadamard 乘积。
- 对于两个形状为 m × n 的矩阵 A 和 B，Hadamard 乘积 A ∘ B（或 A ⊙ B）是与操作数形状相同的矩阵。
- 例如，3 × 3 矩阵 A 与 3 × 3 矩阵 B 的 Hadamard 乘积为。

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\, b_{11} & a_{12}\, b_{12} & a_{13}\, b_{13} \\ a_{21}\, b_{21} & a_{22}\, b_{22} & a_{23}\, b_{23} \\ a_{31}\, b_{31} & a_{32}\, b_{32} & a_{33}\, b_{33} \end{bmatrix}$$

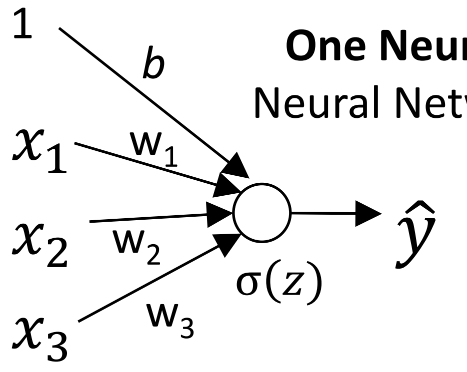
(3x3) ○ (3x3) = (3x3)

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 $\boldsymbol{x}$, want to predict $\hat{y} = P(y = 1 \mid \boldsymbol{x})$, where $0 \leq \hat{y} \leq 1$
  - $\boldsymbol{x} \in \mathbb{R}^{n_x}$, where $\boldsymbol{x} = x_1, x_2, \ldots, x_{n_x}$
  - Parameters: $\boldsymbol{w} \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$
  - $z = \boldsymbol{w}^T \boldsymbol{x} + b$
  - Output: $\hat{y} = \sigma(z)$ where σ() is sigmoid function.

One Neuron
Neural Network



input output

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$\boldsymbol{w}^T = [w_1 \; w_2 \; w_3]$$

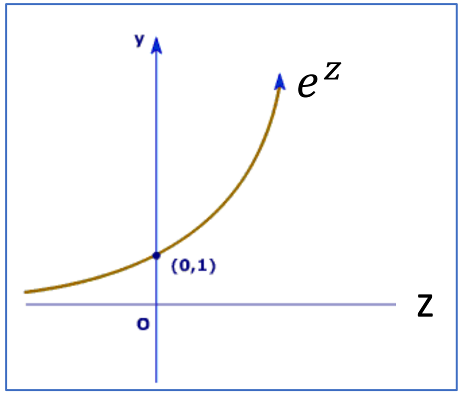
(1,3) (3,1)

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

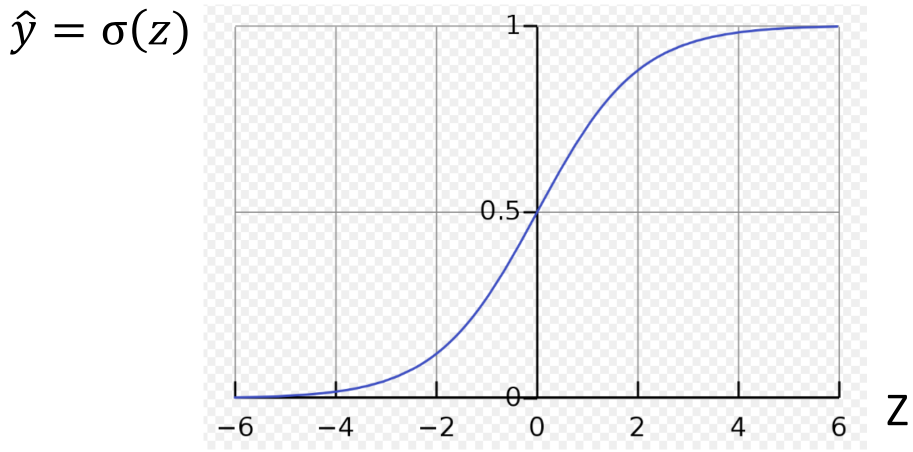
$$= w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Logistic Regression

- Models the ***probability*** that an input belongs to a particular category.
  - Given $\boldsymbol{x}$, want to predict $\hat{y} = P(y = 1 \mid \boldsymbol{x})$, where $0 \le \hat{y} \le 1$
  - $z = \boldsymbol{w}^T \boldsymbol{x} + b$
  - 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, $\sigma(z) = \frac{1}{1+1} = 0.5$

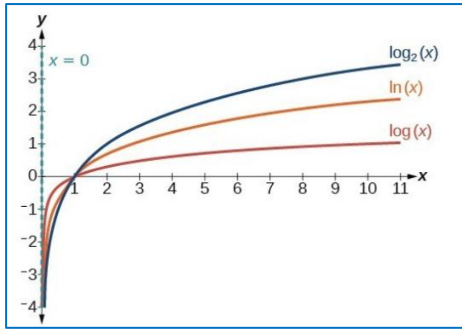
In general, if $\hat{y}$ >= 0.5, predict positive class, otherwise negative class.

Logistic Regression cost function

Given $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

E.g., $\{(\boldsymbol{x}^{(1)}, 0), (\boldsymbol{x}^{(2)}, 1), \ldots, (\boldsymbol{x}^{(m)}, 1)\}$

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



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 ***w*** and *b* values that minimize the cost function

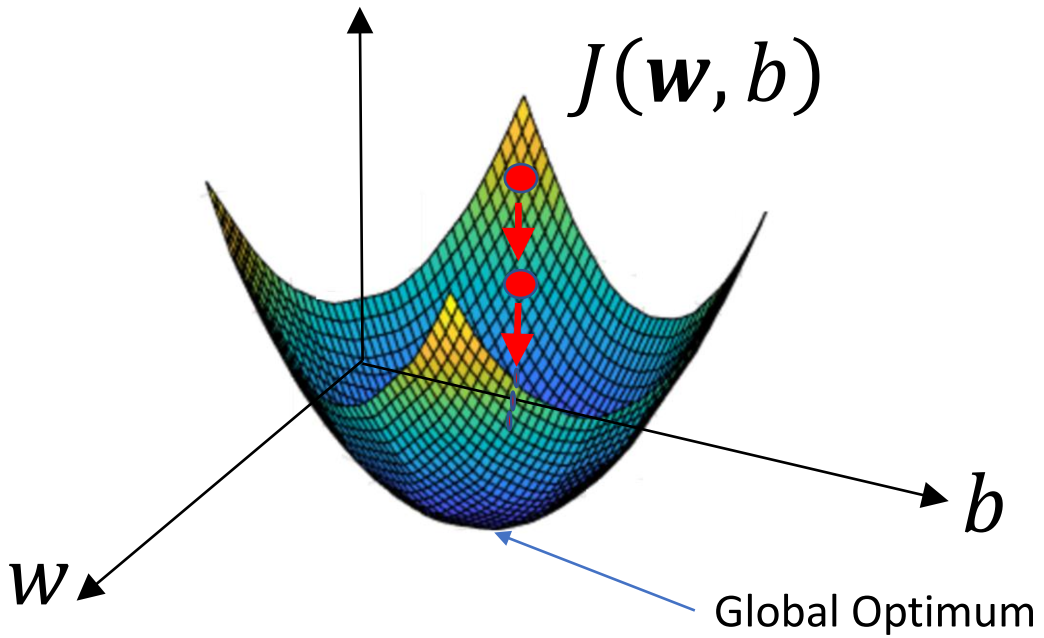
$$J(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

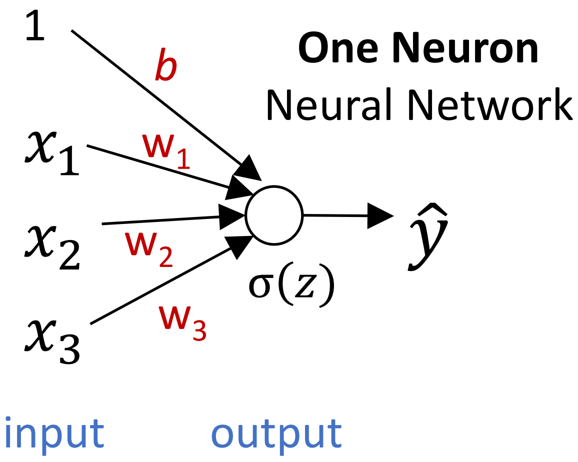
Gradient Descent

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

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

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

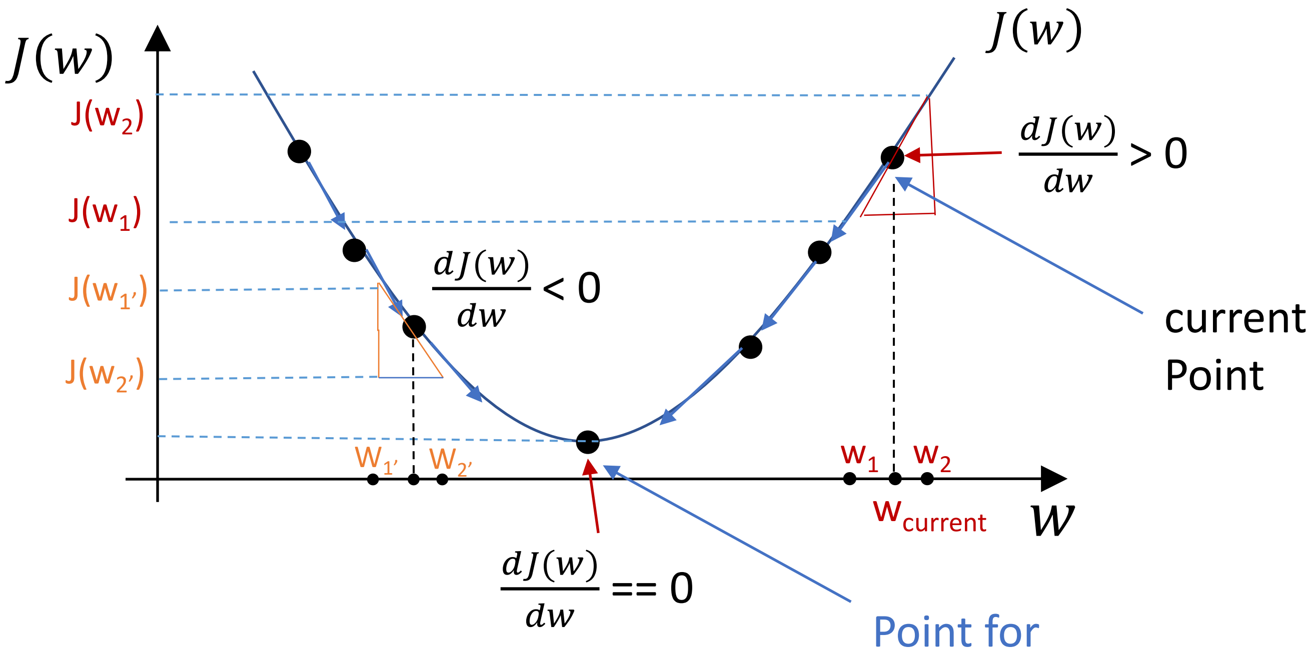




Gradient Descent

Slope = Height / Width
= $J(w_2) - J(w_1) / w_2 - w_1$

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



Repeat {

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

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

}

Learning rate (e.g., 0.1)

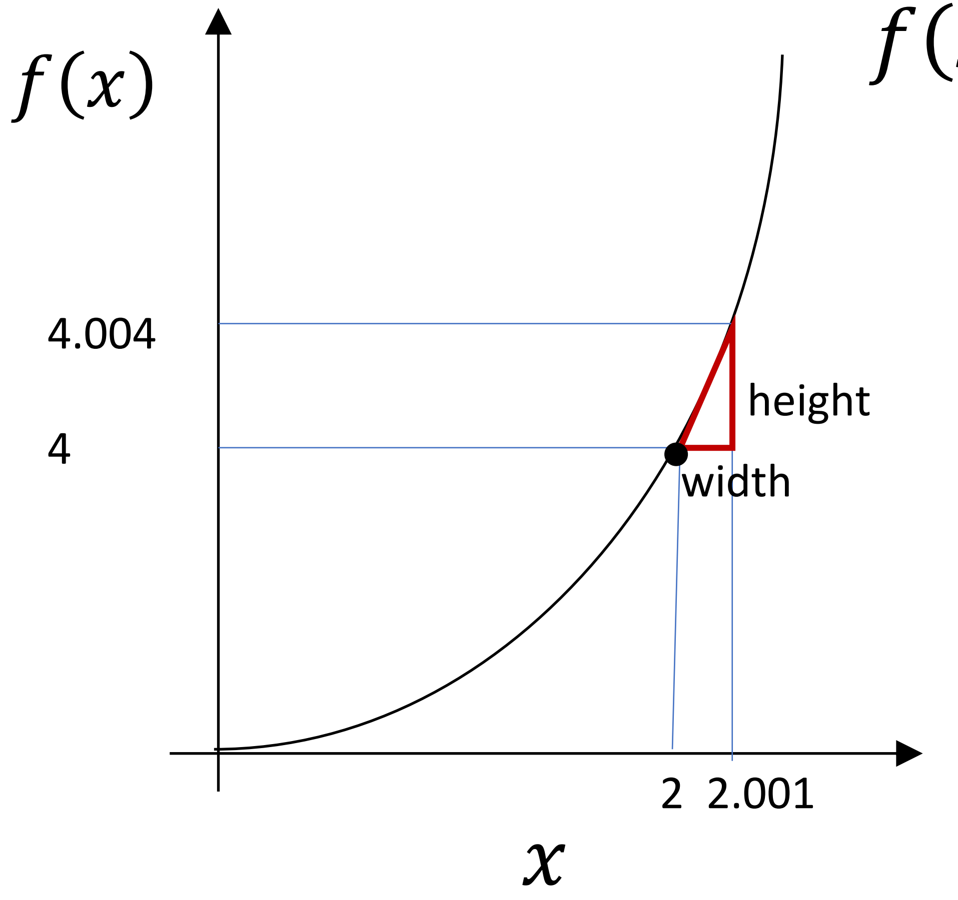
$$w = w - \alpha \frac{dJ(w,b)}{dw}$$

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

- **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. 梯度下降是一种用于寻找函数最小值的迭代优化算法。其步长与当前点处函数梯度的负值成比例。

Intuition about derivatives

关于衍生品的直觉



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

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)=logx: $\frac{df(x)}{dx} = \frac{dlogx}{dx} = \frac{1}{x}$

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:

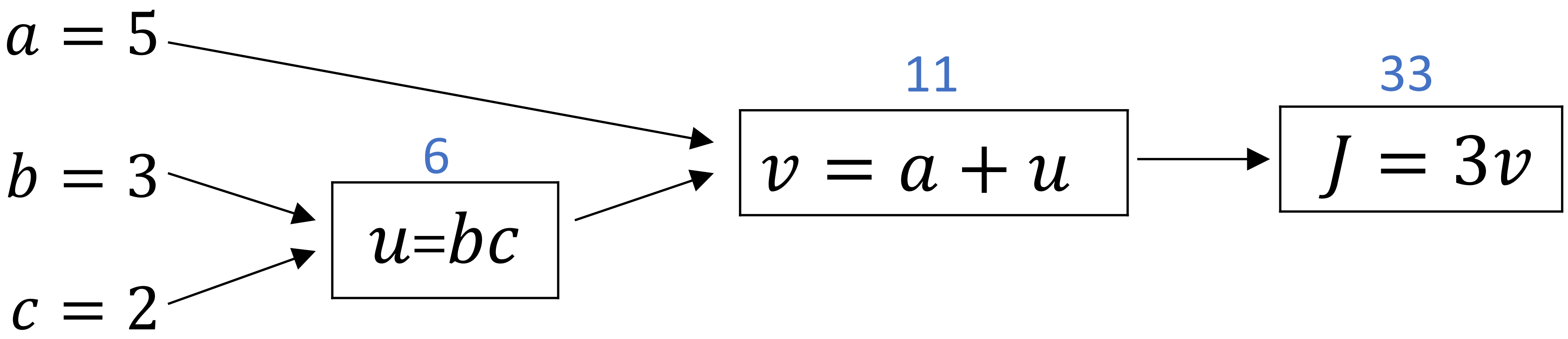
$\frac{4.001^2-4^2}{0.001} = \frac{0.008}{0.001} = 8$.

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*2) = 33

Computing derivatives

chain rule example:

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

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

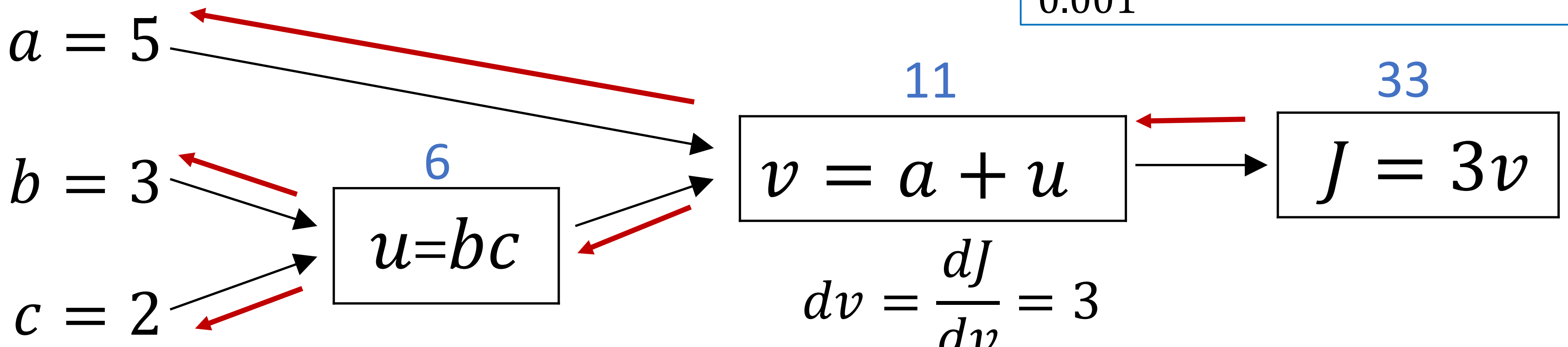
Use chain rule: $J(a, b, c) = 3\,(a + bc)$

$\frac{dJ}{da}$

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 $\frac{dJ}{da} = 3$

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



$$dv = \frac{dJ}{dv} = 3$$

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

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

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

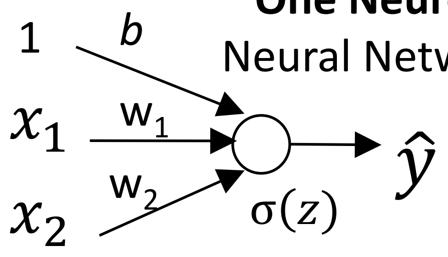
Logistic regression recap

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

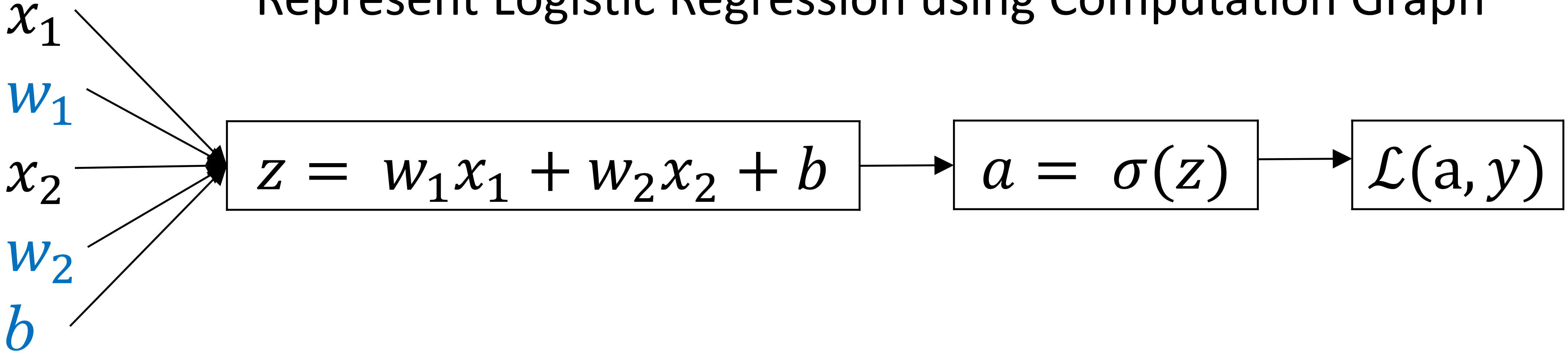
$$\hat{y} = a = \sigma(z)$$

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

One Neuron
Neural Network



Represent Logistic Regression using Computation Graph



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

Logistic regression derivatives

Goal: estimate w_1 , w_2 , and b.

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

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

$$x_1,\ w_1,\ x_2,\ w_2,\ b \longrightarrow \boxed{z = w_1x_1 + w_2x_2 + b} \longrightarrow \boxed{a = \sigma(z)} \longrightarrow \boxed{\mathcal{L}(a,y)}$$

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

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

Using Gradient Descent:

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

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

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

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

Logistic regression on m examples

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

For i = 1 to m

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

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

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

$dz^{(i)} = a^{(i)} - y^{(i)}$ *# use derivatives*

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

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

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

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

$J = J / m$

Update

$w_1 = w_1 - \alpha dw_1$
$w_2 = w_2 - \alpha dw_2$
$b = b - \alpha db$

$epoch = n$ # e.g., n=20
For $j = 1$ to $epoch$
  J=0, dw_1=0, dw_2=0, db=0
  For $i = 1$ to m
    ……
  $J /= m; dw_1 /= m; dw_2 /= m; db /= m$
  $w_1 = w_1 - \alpha dw_1$; $w_2 = w_2 - \alpha dw_2$
  $b = b - \alpha db$

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 $\boldsymbol{w}^T\boldsymbol{x} + b$)
  - Sigmoid function
  - Cost or Loss function
  - Gradient Descent
    - Used for finding $\boldsymbol{w}$, 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