Neural Network: Multi-layer Neural Networks

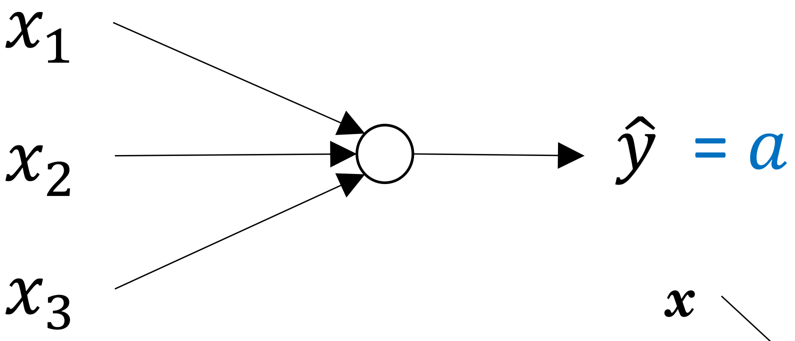
Text and Web Mining (IS6751)

School of Communication and Information

Overview

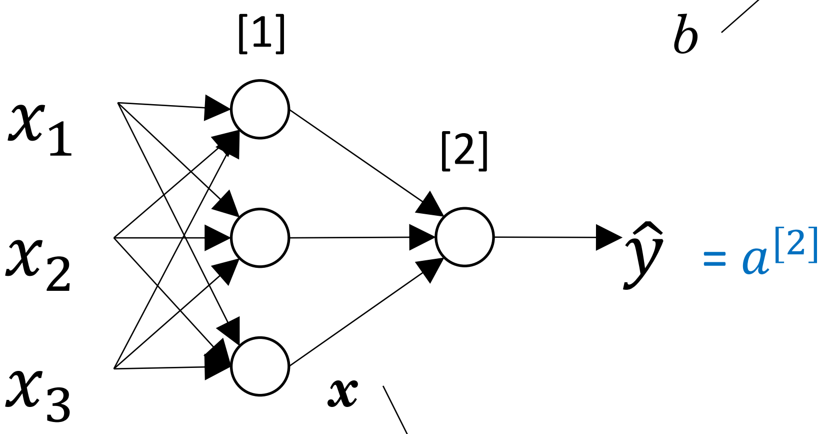
- Introduce Multi-Layer Neural Networks
  - 2 Layer Neural Network
  - Activation Functions
  - Forward and Backward Propagation
- Introduce Multi-class classification
  - Softmax Regression
- Introduce Hyperparameters, Bias, and Variance

What is a Neural Network?



One Neuron
Neural Network
(Logistic Regression)

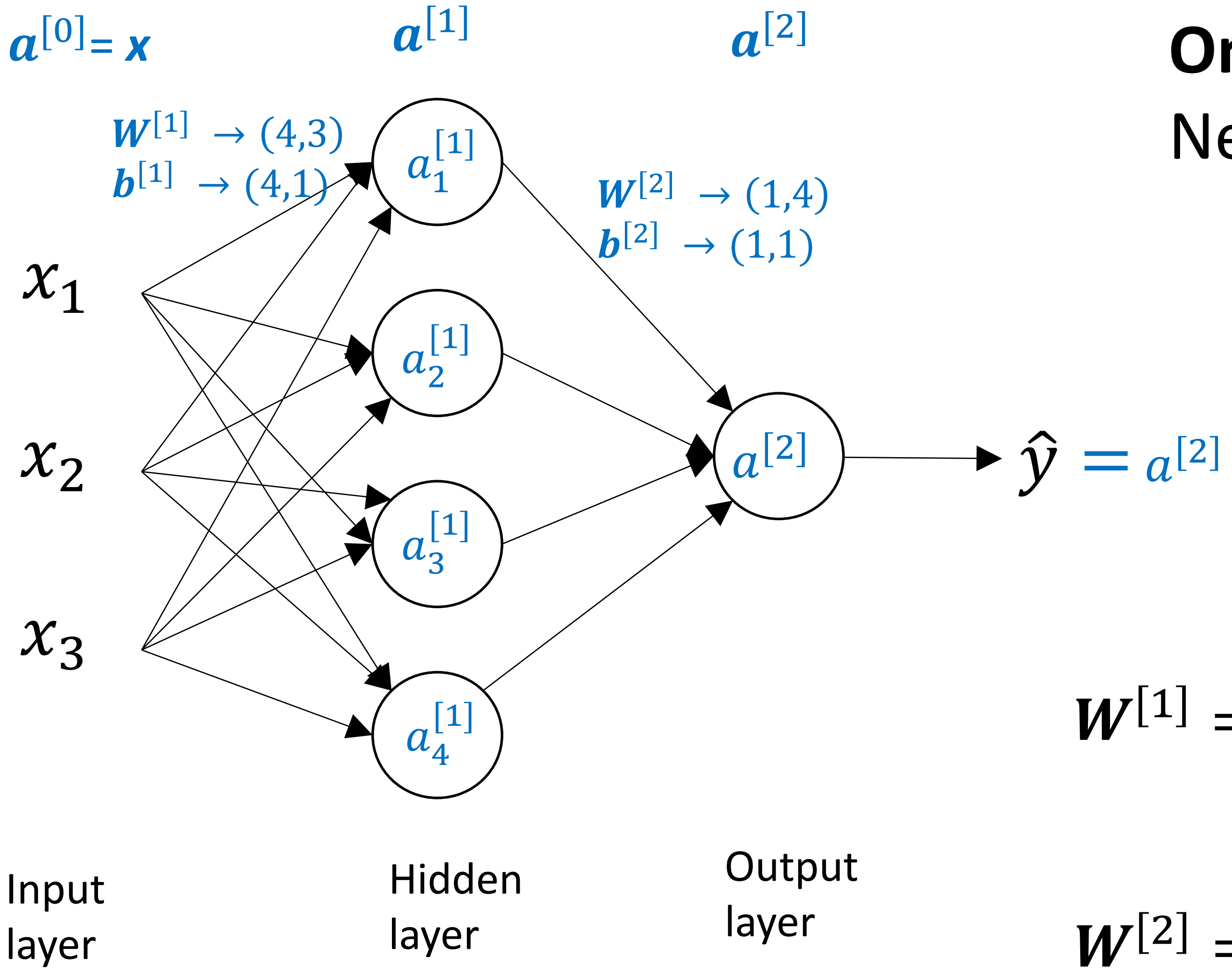
$$\boldsymbol{x}, \boldsymbol{w}, b \rightarrow \boxed{z = \boldsymbol{w}^T\boldsymbol{x} + b} \rightarrow \boxed{a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(a, y)}$$



One hidden layer
Neural Network

$$\boldsymbol{x}, \boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]} \rightarrow \boxed{\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}} \rightarrow \boxed{\boldsymbol{a}^{[1]} = \sigma(\boldsymbol{z}^{[1]})} \xrightarrow{\boldsymbol{W}^{[2]}, \boldsymbol{b}^{[2]}} \boxed{\boldsymbol{z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}} \rightarrow \boxed{\boldsymbol{a}^{[2]} = \sigma(\boldsymbol{z}^{[2]})} \rightarrow \boxed{\mathcal{L}(\boldsymbol{a}^{[2]}, \boldsymbol{y})}$$

Neural Network Representation: 2 Layer NN



One hidden layer

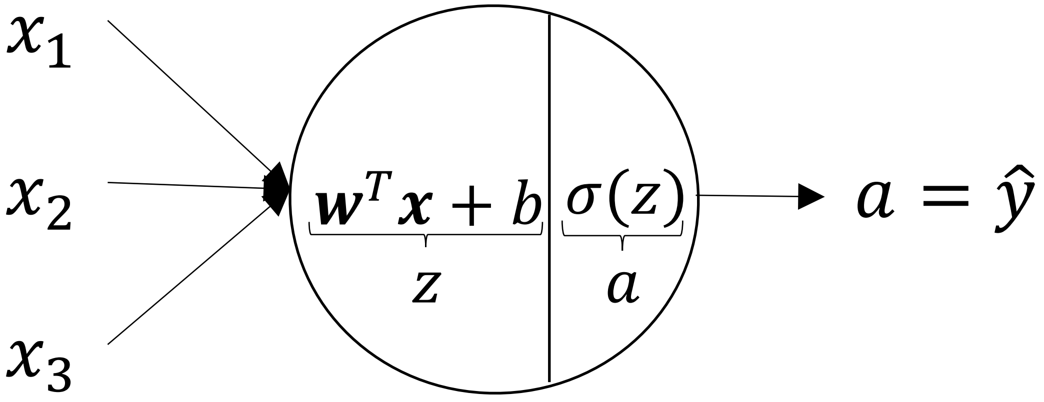
Neural Network

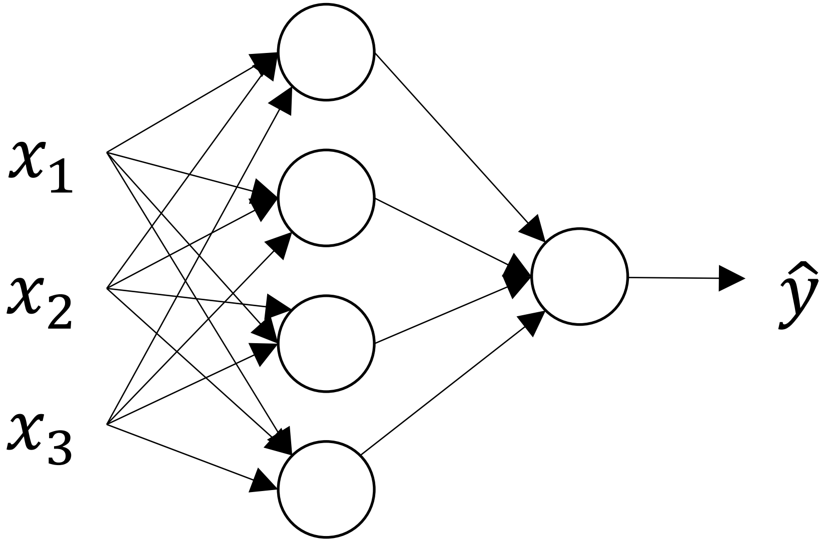
$$W^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$W^{[2]} = [w_{11}\, w_{12}\, w_{13}\, w_{14}]$$

$$b^{[2]} = [b_1]$$

Neural Network Representation





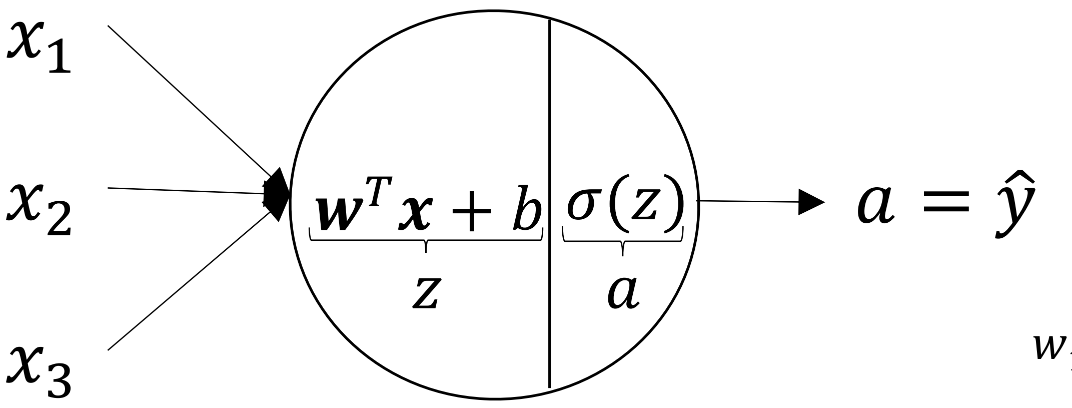
Computing One Neuron Network's Output

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

$$a = \sigma(z)$$

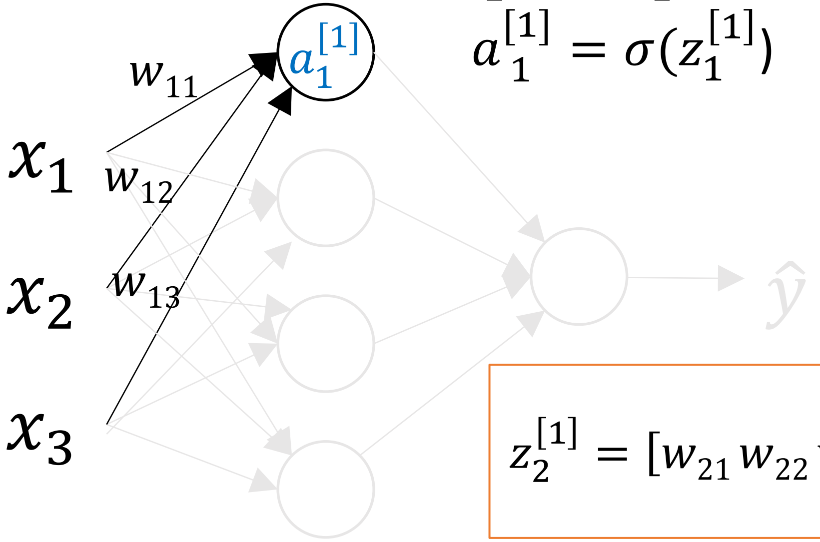
Computing a Neural Network's Output

Neural Network Representation



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

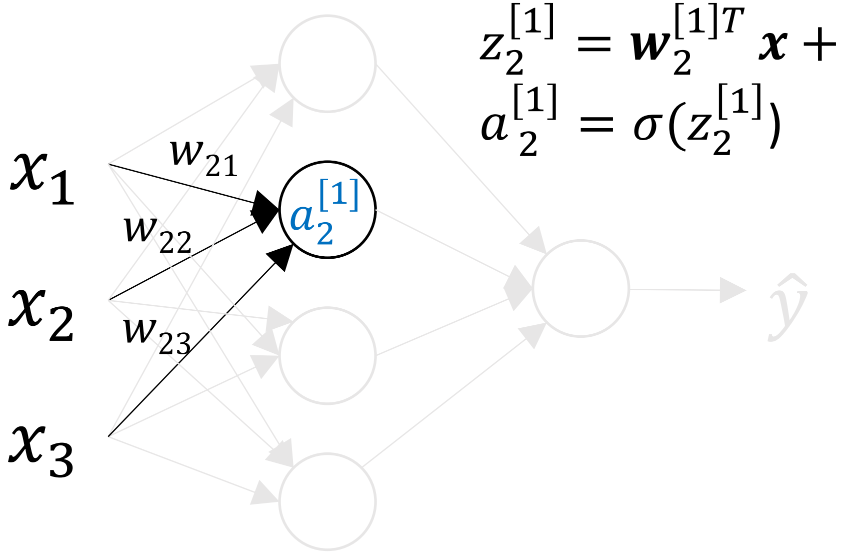
$$a = \sigma(z)$$



$$\boxed{z_1^{[1]} = [w_{11}\, w_{12}\, w_{13}] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b_1^{[1]}}$$

$$z_1^{[1]} = \boldsymbol{w}_1^{[1]T}\,\boldsymbol{x} + b_1^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$



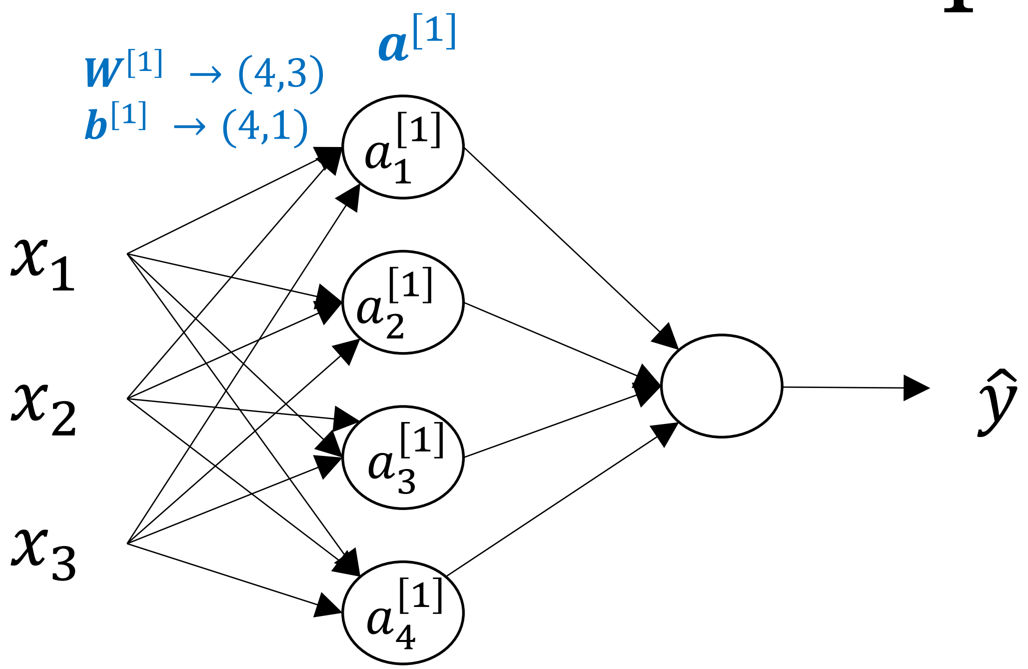
$$\boxed{z_2^{[1]} = [w_{21}\, w_{22}\, w_{23}] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b_2^{[1]}}$$

$$z_2^{[1]} = \boldsymbol{w}_2^{[1]T}\,\boldsymbol{x} + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

w_{11}

Neural Network Representation (cont.)



$$z_1^{[1]} = \boldsymbol{w}_1^{[1]T}\,\boldsymbol{x} + b_1^{[1]}, \qquad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = \boldsymbol{w}_2^{[1]T}\,\boldsymbol{x} + b_2^{[1]}, \qquad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = \boldsymbol{w}_3^{[1]T}\,\boldsymbol{x} + b_3^{[1]}, \qquad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = \boldsymbol{w}_4^{[1]T}\,\boldsymbol{x} + b_4^{[1]}, \qquad a_4^{[1]} = \sigma(z_4^{[1]})$$

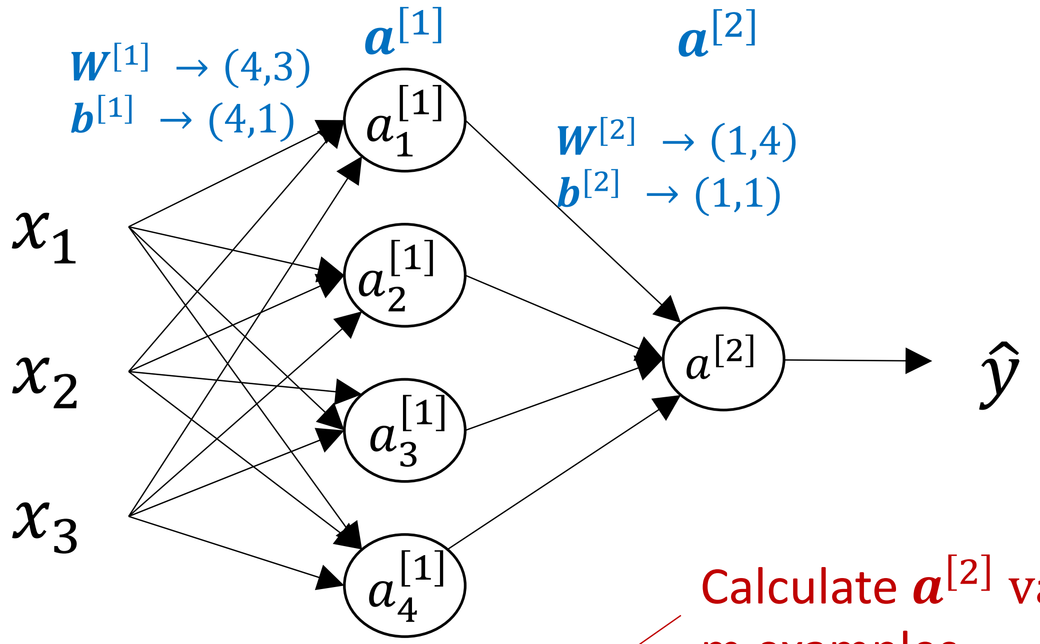
$$\boldsymbol{z}^{[1]} = \underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}}_{\boldsymbol{W}^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1^{[1]} \\ w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2^{[1]} \\ w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + b_3^{[1]} \\ w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix},$$

$$\boldsymbol{a}^{[1]} = \sigma(\boldsymbol{z}^{[1]}) = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

$$\underset{(4,1)}{\boldsymbol{z}^{[1]}} = \underset{(4,3)}{\boldsymbol{W}^{[1]}}\,\underset{(3,1)}{\boldsymbol{x}} + \underset{(4,1)}{\boldsymbol{b}^{[1]}}$$

$$\underset{(4,1)}{\boldsymbol{a}^{[1]}} = \sigma(\underset{(4,1)}{\boldsymbol{z}^{[1]}})$$

Neural Network Representation learning



Given an input $\boldsymbol{x}$:

$$\underset{(4,1)}{\boldsymbol{z}^{[1]}} = \underset{(4,3)}{\boldsymbol{W}^{[1]}}\underset{(3,1)}{\boldsymbol{x}} + \underset{(4,1)}{\boldsymbol{b}^{[1]}}$$

$$\underset{(4,1)}{\boldsymbol{a}^{[1]}} = \sigma(\underset{(4,1)}{\boldsymbol{z}^{[1]}})$$

$$\underset{(1,1)}{\boldsymbol{z}^{[2]}} = \underset{(1,4)}{\boldsymbol{W}^{[2]}}\underset{(4,1)}{\boldsymbol{a}^{[1]}} + \underset{(1,1)}{\boldsymbol{b}^{[2]}}$$

$$\underset{(1,1)}{\boldsymbol{a}^{[2]}} = \sigma(\underset{(1,1)}{\boldsymbol{z}^{[2]}})$$

Calculate $\boldsymbol{a}^{[2]}$ values for m examples

```
for i = 1 to m
```

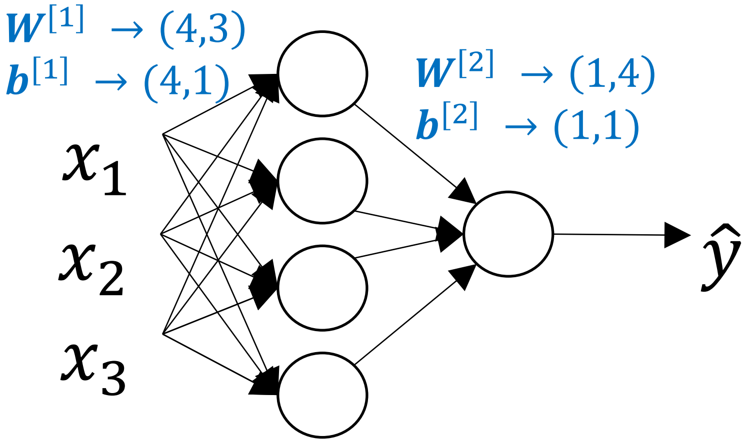
$$\boldsymbol{z}^{[1](i)} = \boldsymbol{W}^{[1]}\boldsymbol{x}^{(i)} + \boldsymbol{b}^{[1]}$$

$$\boldsymbol{a}^{[1](i)} = \sigma(\boldsymbol{z}^{[1](i)})$$

$$\boldsymbol{z}^{[2](i)} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1](i)} + \boldsymbol{b}^{[2]}$$

$$\boldsymbol{a}^{[2](i)} = \sigma(\boldsymbol{z}^{[2](i)})$$

Vectorizing across multiple examples



$$\underset{(3,m)}{\boldsymbol{X}} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(m)} \\ | & | & & | \end{bmatrix}$$

$$\underset{(4,m)}{\boldsymbol{Z}^{[1]}} = \begin{bmatrix} | & | & & | \\ \boldsymbol{z}^{[1](1)} & \boldsymbol{z}^{[1](2)} & \cdots & \boldsymbol{z}^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$\underset{(4,m)}{\boldsymbol{Z}^{[1]}} = \underset{(4,3)}{\boldsymbol{W}^{[1]}} \underset{(3,m)}{\boldsymbol{X}} + \underset{(4,1)}{\boldsymbol{b}^{[1]}}$$

$$\boldsymbol{A}^{[1]} = \sigma(\boldsymbol{Z}^{[1]})$$

$$\underset{(1,m)}{\boldsymbol{Z}^{[2]}} = \underset{(1,4)}{\boldsymbol{W}^{[2]}} \underset{(4,m)}{\boldsymbol{A}^{[1]}} + \underset{(1,1)}{\boldsymbol{b}^{[2]}}$$

$$\boldsymbol{A}^{[2]} = \sigma(\boldsymbol{Z}^{[2]})$$

```
for i = 1 to m
```

$$\boldsymbol{z}^{[1](i)} = \boldsymbol{W}^{[1]} \boldsymbol{x}^{(i)} + \boldsymbol{b}^{[1]}$$

$$\boldsymbol{a}^{[1](i)} = \sigma(\boldsymbol{z}^{[1](i)})$$

$$\boldsymbol{z}^{[2](i)} = \boldsymbol{W}^{[2]} \boldsymbol{a}^{[1](i)} + \boldsymbol{b}^{[2]}$$

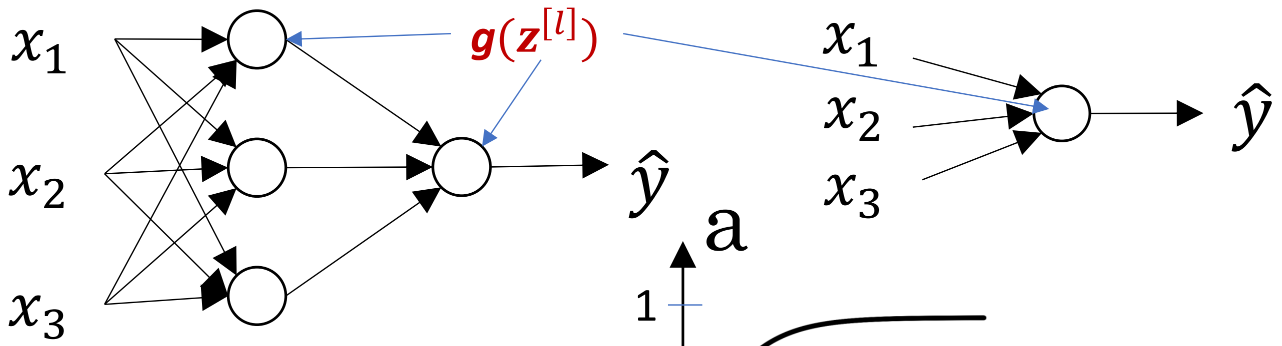
$$\boldsymbol{a}^{[2](i)} = \sigma(\boldsymbol{z}^{[2](i)})$$

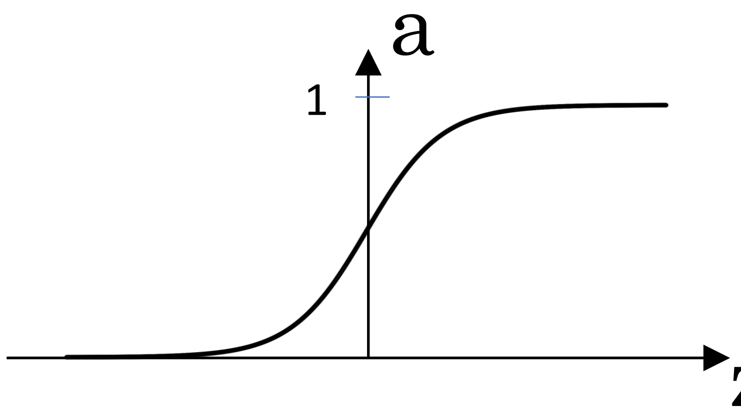
Vectorizing across multiple examples works faster than *for* loop.

$$\underset{(4,m)}{\boldsymbol{A}^{[1]}} = \begin{bmatrix} | & | & & | \\ \boldsymbol{a}^{[1](1)} & \boldsymbol{a}^{[1](2)} & \cdots & \boldsymbol{a}^{[1](m)} \\ | & | & & | \end{bmatrix}$$

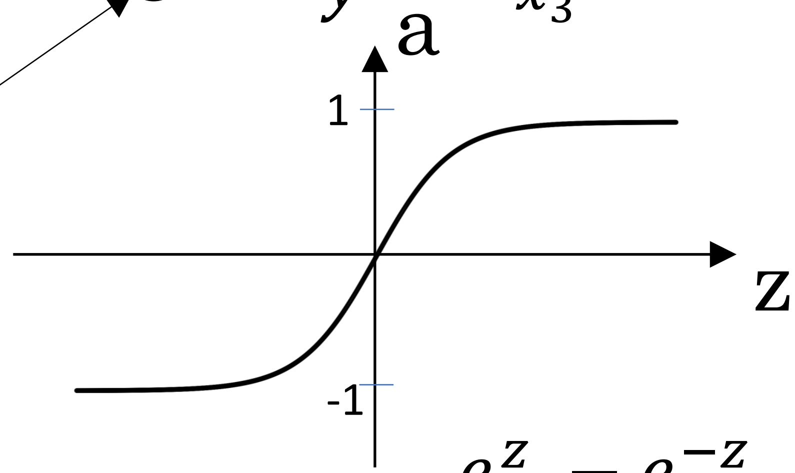
$$\underset{(1,m)}{\boldsymbol{A}^{[2]}} = \begin{bmatrix} a^{[2](1)} & a^{[2](2)} & \cdots & a^{[2](m)} \end{bmatrix}$$

Activation functions

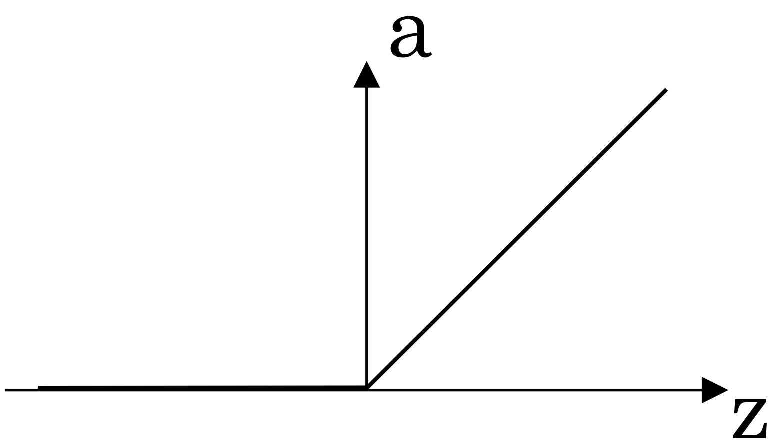




sigmoid: $a = g(z) = \dfrac{1}{1+e^{-z}}$

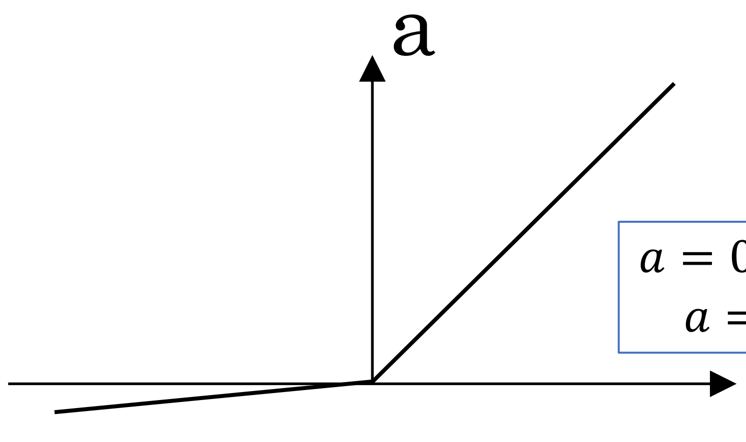


tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



ReLu: $a = \max(0, z)$

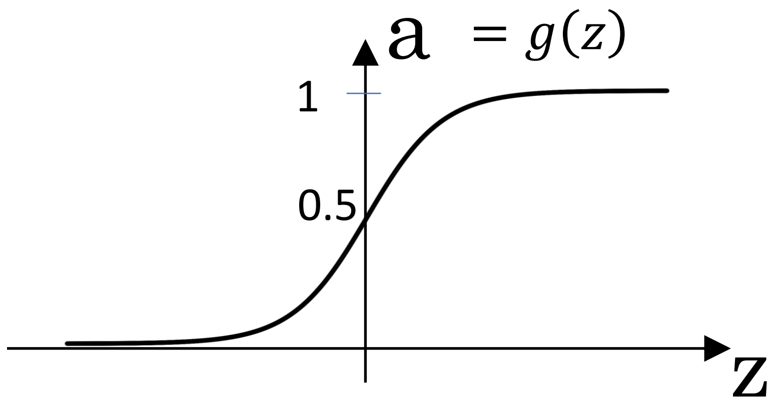
$a = 0 \; if \; z < 0$
$a = z \; if \; z \geq 0$



$a = 0.01z \; if \; z < 0$
$a = z \; if \; z \geq 0$

Leaky ReLu: $a = \max(0.01z, z)$

Sigmoid activation function



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

$$g'(z) = \frac{d}{dz}g(z) = \text{Slope of g}(z)\text{ at z}$$
$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$
$$= g(z)(1 - g(z))$$
$$= a(1-a)$$

When z=10, g(z) (or a) ≈ 1

$\frac{d}{dz}g(z) \approx$ 1(1-1) $\approx$ 0

When z=-10, g(z) ≈ 0

$\frac{d}{dz}g(z) \approx$ 0(1-0) $\approx$ 0

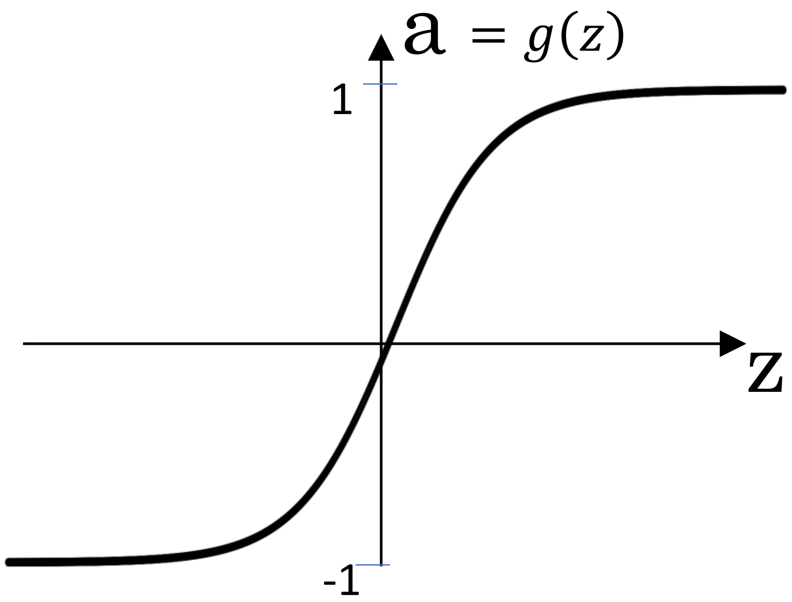
When z=0, g(z) = 0.5

$\frac{d}{dz}g(z)$ = 0.5(1-0.5) = $\frac{1}{4}$

- When Z value close to +/- 10, slope of g(z) becomes close to zero (e.g., 10^{-10}). So, it will slow down learning process, i.e., Gradient Decent process.

$$\frac{\partial \mathcal{L}(a,y)}{\partial w_1} = dw_1 = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right)\cdot a(1-a)\cdot x_1 \quad ; \qquad w_1 = w_1 - \alpha \mathbf{dw_1}$$

Tanh activation function



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

$$g'(z) = \frac{d}{dz}g(z) = \text{Slope of } g(z) \text{ at } z$$
$$= 1 - (\tanh(z))^2$$
$$= 1 - a^2$$

- Compared to Sigmoid, **Tanh** provides higher slope values when *z* value is between -10 and 10. So learning process is faster than sigmoid.

When z=10, g(z) (or a) ≈ 1

$\frac{d}{dz}g(z) \approx$ 1-1 $\approx$ 0

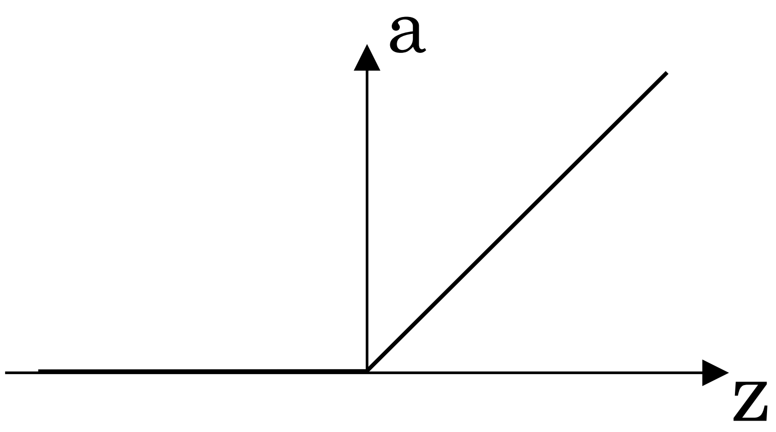
When z=-10, g(z) $\approx$ -1

$\frac{d}{dz}g(z) \approx$ 1-1 $\approx$ 0

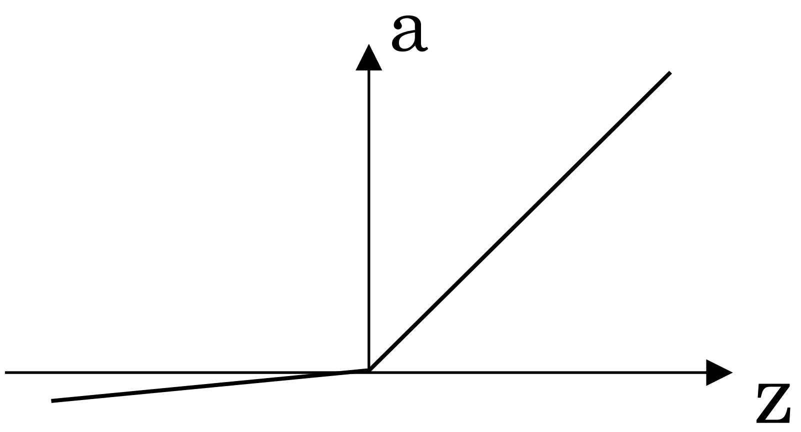
When z=0, g(z) = 0

$\frac{d}{dz}g(z)$= 1-0 = 1

ReLU and Leaky ReLU



ReLU



Leaky ReLU

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

$$\boxed{\begin{array}{c} g(z) = 0\ if\ z < 0 \\ g(z) = z\ if\ z \geq 0 \end{array}}$$

$$g'(z) = \begin{array}{l} 0\ if\ z < 0 \\ 1\ if\ z \geq 0 \end{array}$$

$$g(z) = \max(0.01\text{z}, z)$$

$$\boxed{\begin{array}{c} g(z) = 0.01z\ if\ z < 0 \\ g(z) = z\ if\ z \geq 0 \end{array}}$$

$$g'(z) = \begin{array}{ll} 0.01 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{array}$$

- In ReLU, Slope is always 1 when *z* is larger than or equal to 0. ReLU is a popular activation function since learning process is faster.

- In Leaky ReLU, Slope becomes non-zero when *z* is less than 0.

Computing gradients:

$$x_1, x_2 \rightarrow \bigcirc \rightarrow \hat{y} = a$$

Logistic regression(revisit)

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

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

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

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

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

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

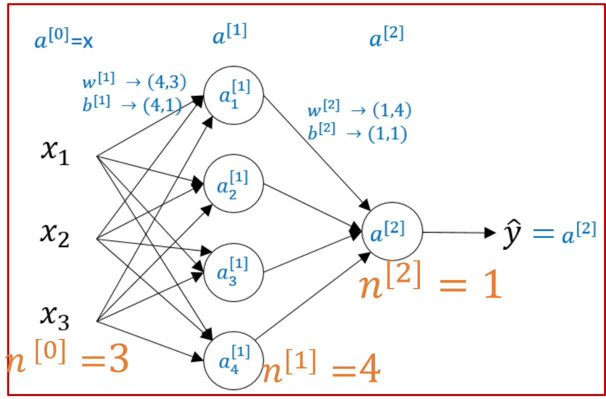
$$\frac{\partial \mathcal{L}(a,y)}{\partial w_1} = dw_1 = \frac{\partial \mathcal{L}(a,y)}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1} = \frac{\partial \mathcal{L}(a,y)}{\partial z} \cdot \frac{\partial z}{\partial w_1} = (a - y) \cdot x_1$$

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

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

Neural network gradients

Shape:
$\boldsymbol{a}^{[1]}, \boldsymbol{z}^{[1]}, \boldsymbol{dz}^{[1]}, \boldsymbol{da}^{[1]}$: $(n^{[1]}, 1)$ -> (4,1)
$\boldsymbol{a}^{[2]}, \boldsymbol{z}^{[2]}, \boldsymbol{dz}^{[2]}, \boldsymbol{da}^{[2]}$: $(n^{[2]}, 1)$ -> (1,1)



$$\begin{matrix} \boldsymbol{W}^{[2]} \\ \boldsymbol{b}^{[2]} \end{matrix}$$

$$\begin{matrix} \boldsymbol{x} \\ \boldsymbol{W}^{[1]} \\ \boldsymbol{b}^{[1]} \end{matrix} \rightarrow \boxed{\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}} \rightarrow \boxed{\boldsymbol{a}^{[1]} : g^{[1]}\,(\boldsymbol{z}^{[1]})} \rightarrow \boxed{\boldsymbol{z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}} \rightarrow \boxed{\boldsymbol{a}^{[2]} = \sigma(\boldsymbol{z}^{[2]})} \rightarrow \boxed{\mathcal{L}(\boldsymbol{a}^{[2]}, \boldsymbol{y})}$$

$$\boldsymbol{dW}^{[1]} = \begin{bmatrix} dw_{11}\, dw_{12}\, dw_{13} \\ dw_{21} dw_{22} dw_{23} \\ dw_{31}\, dw_{32}\, dw_{33} \\ dw_{41}\, dw_{42}\, dw_{43} \end{bmatrix} \quad (4,3)$$

$$\boldsymbol{dW}^{[2]} = [dw_{11}\, dw_{12}\, dw_{13}\, dw_{14}] \quad (1, 4)$$

$$\boldsymbol{db}^{[2]} = [db_1] \; (1, 1)$$

$$\boldsymbol{db}^{[1]} = \begin{bmatrix} db_1 \\ db_2 \\ db_3 \\ db_4 \end{bmatrix} \quad (4,1)$$

$$\boxed{\begin{aligned} \boldsymbol{W}^{[1]} &= \boldsymbol{W}^{[1]} - \alpha\, \boldsymbol{dW}^{[1]} \\ \boldsymbol{b}^{[1]} &= \boldsymbol{b}^{[1]} - \alpha\, \boldsymbol{db}^{[1]} \\ \boldsymbol{W}^{[2]} &= \boldsymbol{W}^{[2]} - \alpha\, \boldsymbol{dW}^{[2]} \\ \boldsymbol{b}^{[2]} &= \boldsymbol{b}^{[2]} - \alpha\, \boldsymbol{db}^{[2]} \end{aligned}}$$

$$\boldsymbol{W}^{[1]} = \begin{bmatrix} w_{11}\, w_{12}\, w_{13} \\ w_{21}\, w_{22}\, w_{23} \\ w_{31}\, w_{32}\, w_{33} \\ w_{41}\, w_{42}\, w_{43} \end{bmatrix} \qquad \boldsymbol{b}^{[1]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

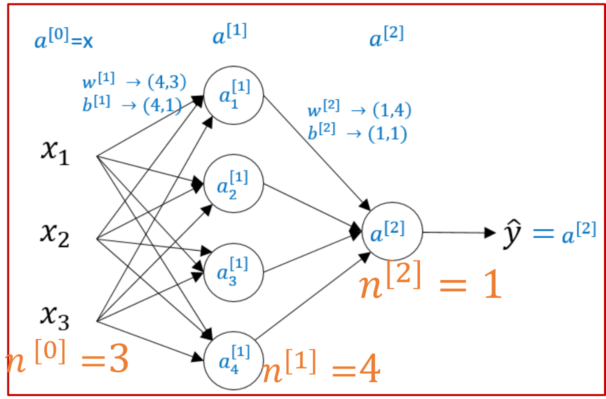
$$\boldsymbol{W}^{[2]} = [w_{11}\, w_{12}\, w_{13}\, w_{14}]$$

$$\boldsymbol{b}^{[2]} = [b_1]$$

Note: Need to estimate ***21*** parameters

Neural network gradients

Shape:
$\boldsymbol{a}^{[2]}, \boldsymbol{z}^{[2]}, \boldsymbol{dz}^{[2]}, \boldsymbol{da}^{[2]}$: $(n^{[2]}, 1)$ -> (1,1)
$\boldsymbol{a}^{[1]}, \boldsymbol{z}^{[1]}, \boldsymbol{dz}^{[1]}, \boldsymbol{da}^{[1]}$: $(n^{[1]}, 1)$ -> (4,1)



$$\boldsymbol{x},\ \boldsymbol{W}^{[1]},\ \boldsymbol{b}^{[1]} \rightarrow \boxed{\boldsymbol{z}^{[1]} = \boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}} \rightarrow \boxed{\boldsymbol{a}^{[1]} : g^{[1]}(\boldsymbol{z}^{[1]})} \rightarrow \boxed{\boldsymbol{z}^{[2]} = \boldsymbol{W}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}} \rightarrow \boxed{\boldsymbol{a}^{[2]} = \sigma(\boldsymbol{z}^{[2]})} \rightarrow \boxed{\mathcal{L}(\boldsymbol{a}^{[2]}, \boldsymbol{y})}$$

$\boldsymbol{W}^{[2]},\ \boldsymbol{b}^{[2]}$ → $\boldsymbol{z}^{[2]}$

$$d\boldsymbol{a}^{[2]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{a}^{[2]}} = -\frac{\boldsymbol{y}}{\boldsymbol{a}^{[2]}} + \frac{1-\boldsymbol{y}}{1-\boldsymbol{a}^{[2]}}$$

$$\mathrm{d}\boldsymbol{z}^{[2]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[2]}} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{a}^{[2]}} \cdot \frac{\partial \boldsymbol{a}^{[2]}}{\partial \boldsymbol{z}^{[2]}} = \left(-\frac{\boldsymbol{y}}{\boldsymbol{a}^{[2]}} + \frac{1-\boldsymbol{y}}{1-\boldsymbol{a}^{[2]}}\right) \cdot \boldsymbol{a}^{[2]}(1-\boldsymbol{a}^{[2]}) = \boldsymbol{a}^{[2]} - \boldsymbol{y}$$

$$\mathrm{d}\boldsymbol{a}^{[1]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{a}^{[1]}} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[2]}} \cdot \frac{\partial \boldsymbol{z}^{[2]}}{\partial \boldsymbol{a}^{[1]}} = \boldsymbol{W}^{[2]T}\mathbf{dz}^{[2]}$$

(4,1)(1,1)−> (4,1)

$$\mathrm{d}\boldsymbol{z}^{[1]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[1]}} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{a}^{[1]}} \cdot \frac{\partial \boldsymbol{a}^{[1]}}{\partial \boldsymbol{z}^{[1]}} = \boldsymbol{W}^{[2]T}\mathbf{dz}^{[2]} * g^{[1]\prime}(\boldsymbol{z}^{[1]})$$

Elementwise product

(4,1) ∗ (4,1) −> (4,1)

$$\begin{bmatrix} da_1^{[1]} \\ da_2^{[1]} \\ da_3^{[1]} \\ da_4^{[1]} \end{bmatrix} * \begin{bmatrix} g^{[1]\prime}(z_1^{[1]}) \\ g^{[1]\prime}(z_2^{[1]}) \\ g^{[1]\prime}(z_3^{[1]}) \\ g^{[1]\prime}(z_4^{[1]}) \end{bmatrix}$$

$$\mathbf{d}\boldsymbol{W}^{[1]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[1]}} \cdot \frac{\partial \boldsymbol{z}^{[1]}}{\partial \boldsymbol{W}^{[1]}} = \mathbf{dz}^{[1]}\boldsymbol{x}^{T}$$

(4,1)(1,3) -> (4,3)

$$\mathbf{d}\boldsymbol{b}^{[1]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[1]}} \cdot \frac{\partial \boldsymbol{z}^{[1]}}{\partial \boldsymbol{b}^{[1]}} = \mathbf{dz}^{[1]}$$

-> (4,1)

$$\boxed{[dw_{11}\, dw_{12}\, dw_{13}\, dw_{14}]}$$

(1,1)(1,4)−> (1,4)

$$\mathbf{d}\boldsymbol{W}^{[2]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[2]}} \cdot \frac{\partial \boldsymbol{z}^{[2]}}{\partial \boldsymbol{w}^{[2]}} = (\boldsymbol{a}^{[2]} - \boldsymbol{y})\boldsymbol{a}^{[1]T} = \mathbf{dz}^{[2]}\boldsymbol{a}^{[1]T}$$

$$\boxed{[db]}$$

$$\boldsymbol{db}^{[2]} = \frac{\partial \mathcal{L}(\boldsymbol{a}^{[2]},\boldsymbol{y})}{\partial \boldsymbol{z}^{[2]}} \cdot \frac{\partial \boldsymbol{z}^{[2]}}{\partial \boldsymbol{b}^{[2]}} = (\boldsymbol{a}^{[2]} - \boldsymbol{y}) \cdot 1 = \mathbf{dz}^{[2]}$$

(1,1)

Summary of gradient descent

Computing gradients:

$$dz^{[2]} = a^{[2]} - y$$

(1, 1)

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

(1,1) (1,4) -> (1,4)

$$db^{[2]} = dz^{[2]}$$

(1, 1)

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$(4,1) * (4,1) -> (4,1)$

$$dW^{[1]} = dz^{[1]} x^T$$

$(4,1)(1,3) -> (4,3)$

$$db^{[1]} = dz^{[1]}$$

(4, 1)

Vectorizing across multiple examples:

Note: need to use the average $dW^{[2]}$ value.

$$dZ^{[2]} = A^{[2]} - Y$$

(1,m)

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$ (1,m) (m, 4) -> (1,4)

sum of each row

$$db^{[2]} = \frac{1}{m} torch.sum(dZ^{[2]}, dim = 1, keepdim{=}True)$$ (1, 1)

(1,m)

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$ $(4, m) * (4, m) -> (4,m)$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$ $(4, m)(m, 3) -> (4,3)$

$$db^{[1]} = \frac{1}{m} torch.sum(dZ^{[1]}, dim = 1, keepdim{=}True)$$

$(4, m)$

(4, 1)

Forward and backward propagation

Forward propagation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}\left(Z^{[L]}\right) = \hat{Y}$$

L = # of Layers

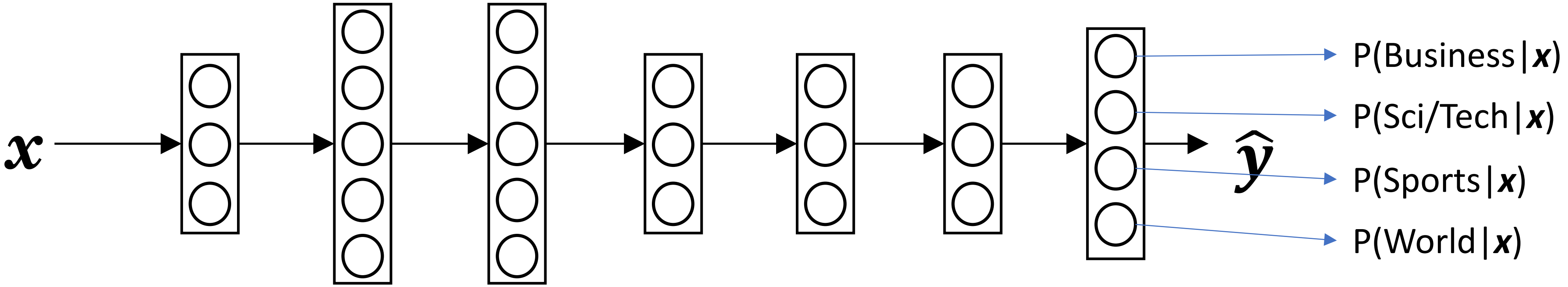
Backward propagation

$$dZ^{[L]} = A^{[L]} - Y$$
$$dW^{[L]} = \frac{1}{m} dZ^{[L]} A^{[L-1]^T}$$
$$db^{[L]} = \frac{1}{m} torch.\mathrm{sum}(\mathbf{d}\mathbf{Z}^{[L]}, dim = 1, keepdim{=}True)$$
$$dZ^{[L-1]} = dW^{[L]^T} dZ^{[L]} * g'^{[L-1]}(Z^{[L-1]})$$
$$\vdots$$
$$dZ^{[1]} = dW^{[2]^T} dZ^{[2]} * g'^{[1]}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} A^{[0]^T}$$
$$db^{[1]} = \frac{1}{m} torch.\mathrm{sum}(\mathbf{d}\mathbf{Z}^{[1]}, dim = 1, keepdim{=}True)$$

Multi-class classification: Softmax regression

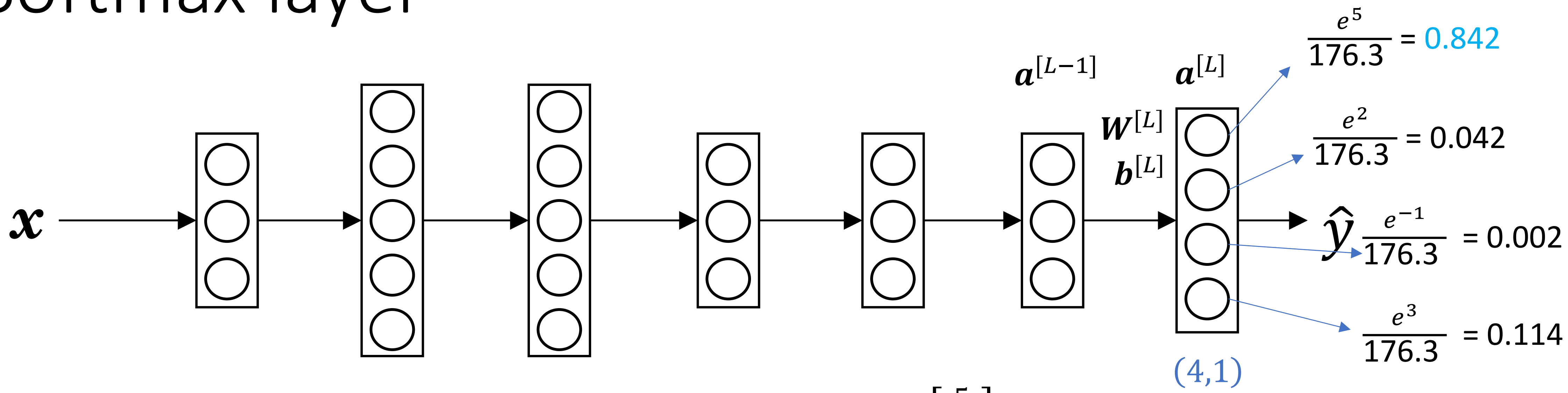
Recognizing News categories: ***Business, Sci/Tech, Sports,*** and ***World***

C = #classes = 4 (i.e., 0, 1, 2, 3)



$\hat{\boldsymbol{y}}$ is $(4,1)$.

Softmax layer



$$\boldsymbol{z}^{[L]} = \boldsymbol{W}^{[L]}\boldsymbol{a}^{[L-1]} + \boldsymbol{b}^{[L]}$$

Activation Function: $\boldsymbol{t}^{[L]} = e^{(\boldsymbol{z}^{[L]})}$

$$\boldsymbol{a}^{[L]} = g^{[L]}\left(\boldsymbol{z}^{[L]}\right) = \frac{e^{(\boldsymbol{z}^{[L]})}}{\sum_{i=1}^{4} t_i} ,$$

$$a_i^{[L]} = \frac{t_i}{\sum_{i=1}^{4} t_i}$$

$$\boldsymbol{z}^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

0.842 + 0.042 + 0.002 + 0.114 = **1**

$$\boldsymbol{t}^{[L]} = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}, \quad \sum_{i=1}^{4} t_i = 176.3$$

$$a_i^{[L]} = \frac{t_i}{176.3}$$

Trying a softmax classifier: Understanding softmax

$$\boldsymbol{z}^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \qquad \boldsymbol{t}^{[L]} = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$$

, when C= 4

"soft max"

$$\boldsymbol{a}^{[L]} = g^{[L]}(\boldsymbol{z}^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(e^5+e^2+e^{-1}+e^3) \\ e^{-1}/(e^5+e^2+e^{-1}+e^3) \\ e^3/(e^5+e^2+e^{-1}+e^3) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

Softmax regression generalizes logistic regression to C classes :

- If C=2, softmax reduces to logistic regression, $\boldsymbol{a}^{[L]} = \begin{bmatrix} 0.842 \\ 0.158 \end{bmatrix}$

Loss function for multi-class classification

(4,1)

$$\boldsymbol{y}^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \leftarrow y_2$$

(4,1)

$$\hat{\boldsymbol{y}}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \leftarrow \hat{y}_2$$

Loss function for binary classification:

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

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{j=1}^{4} y_j log \hat{y}_j$$

Make it small

$$-y_2 log \hat{y}_2 = -log \hat{y}_2 = -log 0.2 \approx 0.699$$

Make $\hat{y}_2$ big.

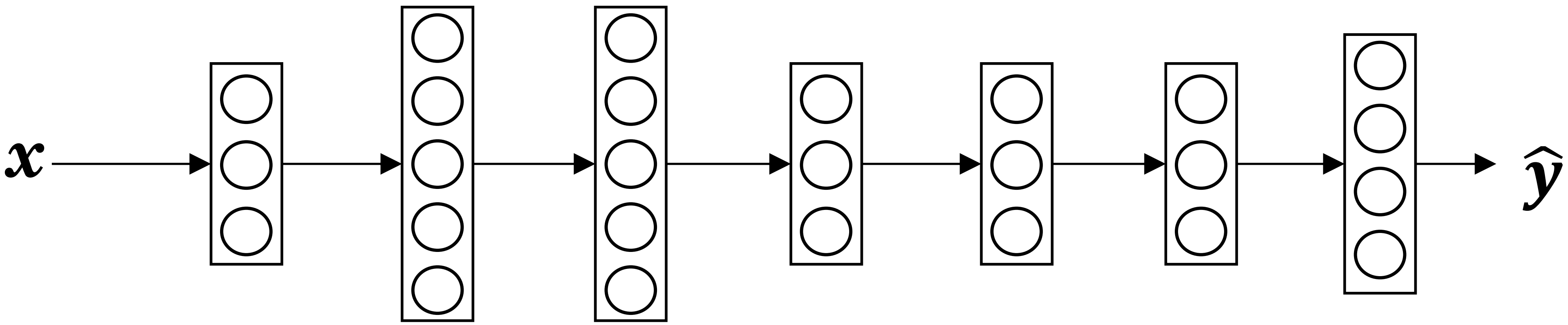
$$J(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)})$$

$$\boldsymbol{Y} = [y^{(1)} y^{(2)} y^{(3)}, \ldots, y^{(m)}] \; ; \; \hat{\boldsymbol{Y}} = [\hat{y}^{(1)} \hat{y}^{(2)} \hat{y}^{(3)}, \ldots, \hat{y}^{(m)}]$$

$$= \begin{bmatrix} 0 & 0 & & \\ 1 & 0 & \cdots\cdots & \\ 0 & 1 & & \\ 0 & 0 & & \end{bmatrix} \quad = \begin{bmatrix} 0.3 & 0.1 & & \\ 0.2 & 0.1 & & \\ 0.1 & 0.7 & \cdots\cdots & \\ 0.4 & 0.2 & & \end{bmatrix}$$

(4,m) (4,m)

Summary of softmax classifier



$\boldsymbol{z}^{[L]} \Rightarrow \boldsymbol{a}^{[L]} \Rightarrow \hat{\boldsymbol{y}} \Rightarrow \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$

(4,1)

(4,1)

Backprop: $\boldsymbol{dz}^{[L]} = \hat{\boldsymbol{y}} - \boldsymbol{y}$

$$\frac{\partial \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})}{\partial \boldsymbol{z}^{[L]}}$$

What are hyperparameters?

Parameters: $\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}, \boldsymbol{W}^{[2]}, \boldsymbol{b}^{[2]}, \boldsymbol{W}^{[3]}, \boldsymbol{b}^{[3]} \ldots$

- Hyperparameters: affect the values of parameters but cannot be learned by training.
  - Learning rate
  - The number of iterations (epoch)
  - The number of hidden layers
  - The number of hidden units in each layer
  - Choice of activation functions
  - Additional ones
    - Momentum: how to update parameters
    - Mini-batch size: use a subset of the whole training data for updating parameters.
      - What if m = 5,000,000 ?
      - -> 5,000 mini-batches of 1,000 each (i.e., batch size = 1,000)
    - Regularizations: e.g., drop out rate
    - Etc.

Setting up your ML application: Train/validation/test sets

Applied ML is a highly iterative process

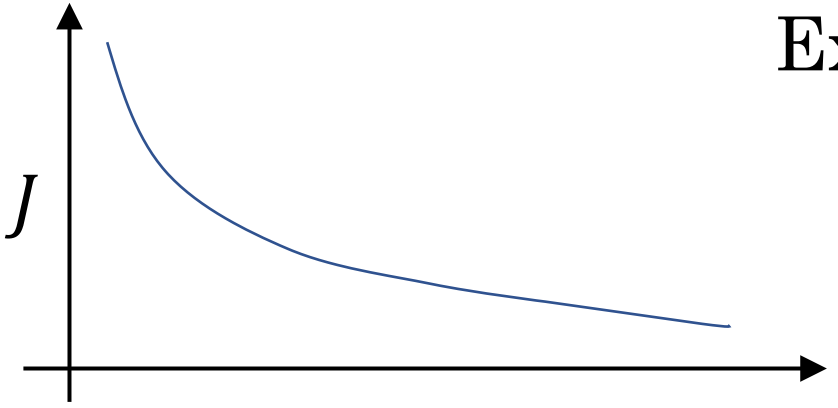
layers

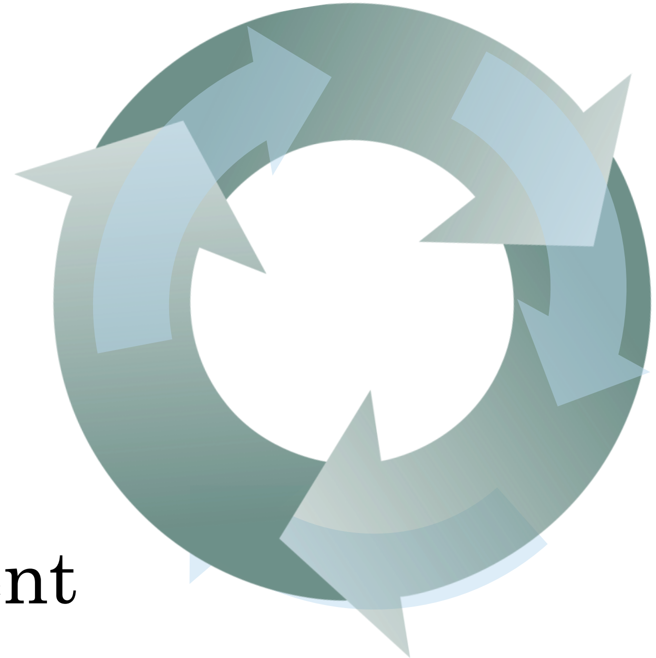
hidden units

learning rates

activation functions

…





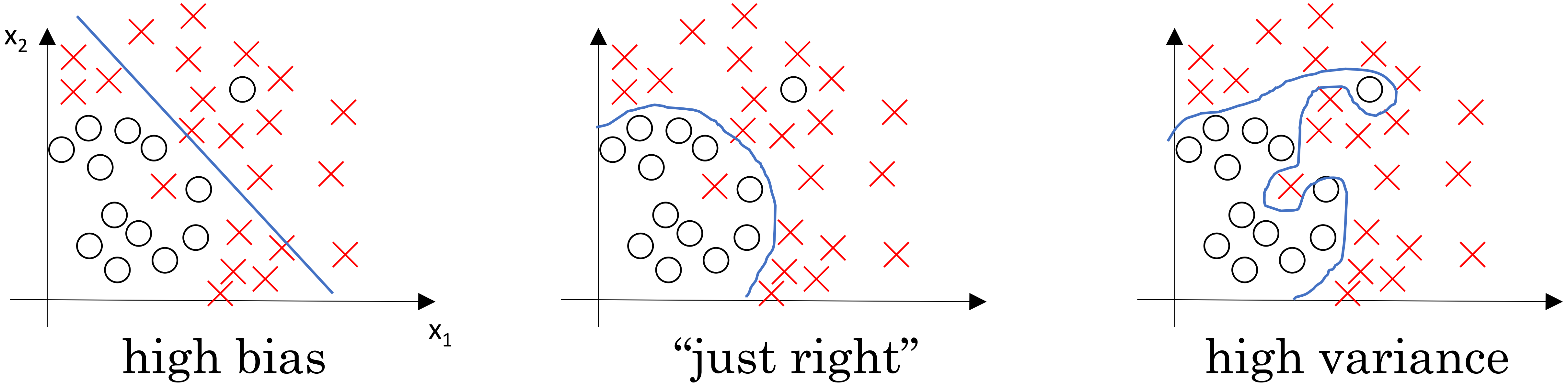
Train/valid/test sets

- Data

Training Set	Validation set	Test Set

- Previous era: 100, 1000, 10,000 data points
  - “70 (train) / 30 (test)”% or “60 (train) / 20 (validation)/ 20 (test)”%
- Big data era: 1,000,000 data points
  - 98 (train) / 1 (validation)/ 1 (test) %
  - 99.5 (train) / 0.4 (validation)/ 0.1 (test) %

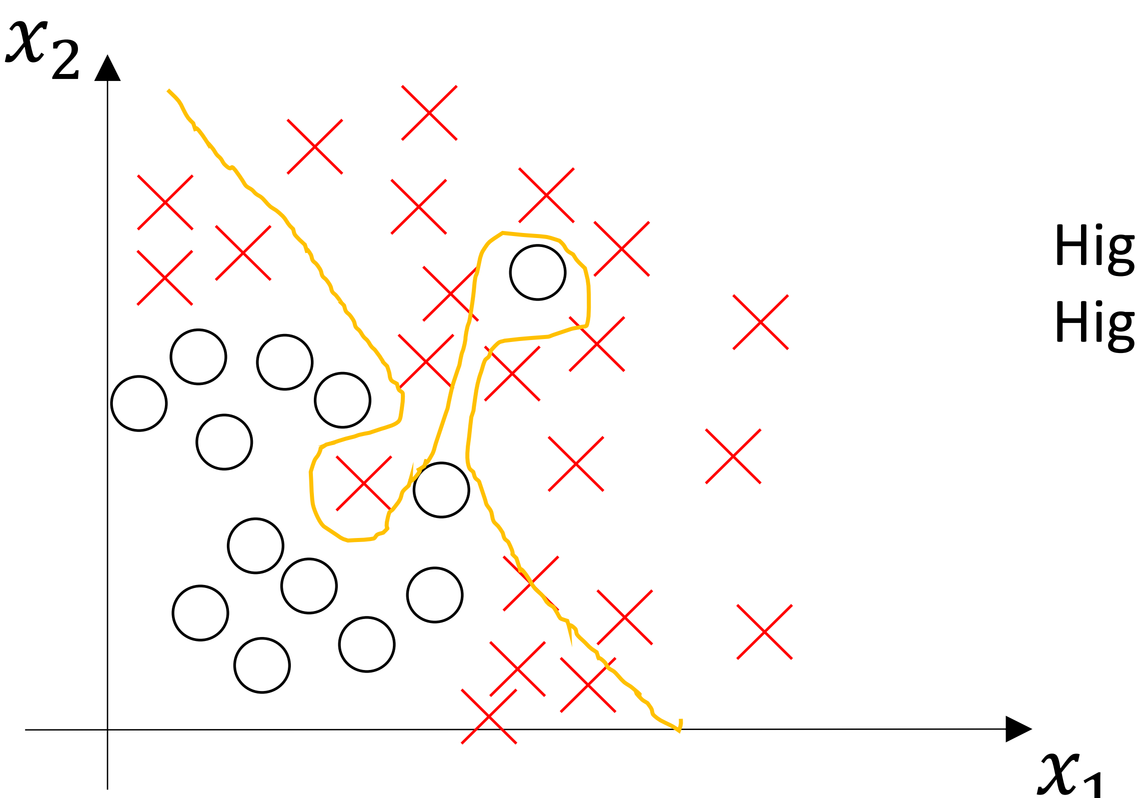
Bias and Variance



Underfitting of data

Overfitting to noisy data

High bias and high variance



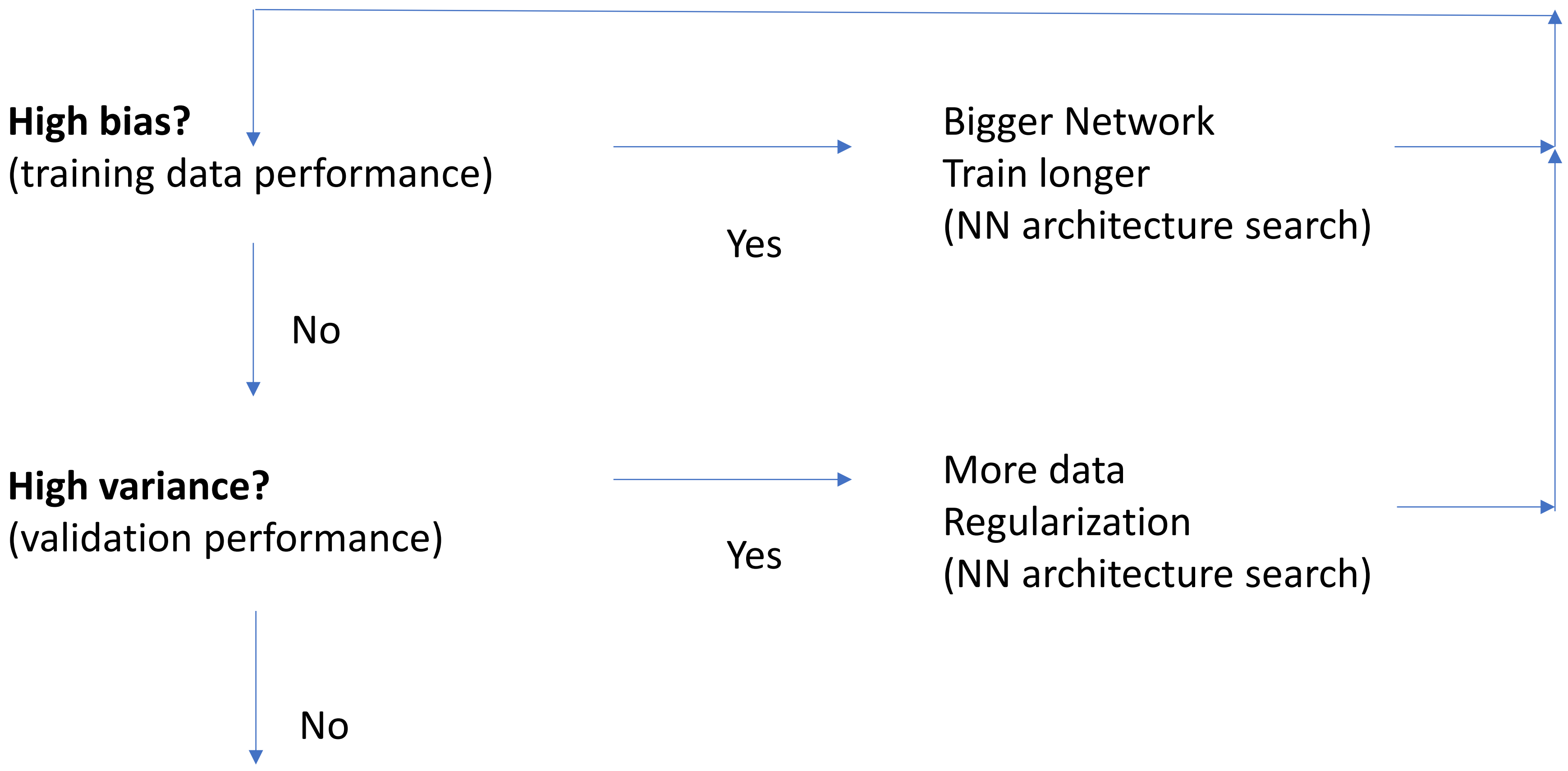
High bias
High variance

Bias and Variance

Sentiment classification

Train set error:	1%	15%	15%	0.5%
Valid set error:	11%	16%	30%	1%
When Human error (Optimal error) ≈ 0%,	High variance	high bias	high bias high variance	**low bias low variance**
When Human error (Optimal error) ≈ 15%,		**low bias low variance**	low bias high variance	

Basic recipe for machine learning



Summary of today's lecture

- Learned Multi-Layer Neural Networks
  - 2 Layer Neural Network
  - Vectorizing across multiple examples
  - Activation Functions
  - Forward and Backward Propagation
    - Used Computation Graph and Gradient Descent
- Learned Multi-class classification
  - Softmax and Loss functions for multi-class classification
- Learned Hyperparameters, Bias, and Variance

Reference

- Neural Network and Deep Learning, Andrew Ng, https://www.coursera.org/learn/neural-networks-deep-learning