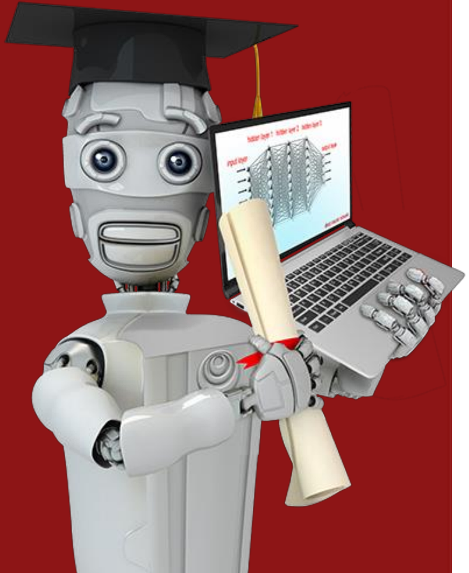
Copyright Notice

DeepLearning.AI

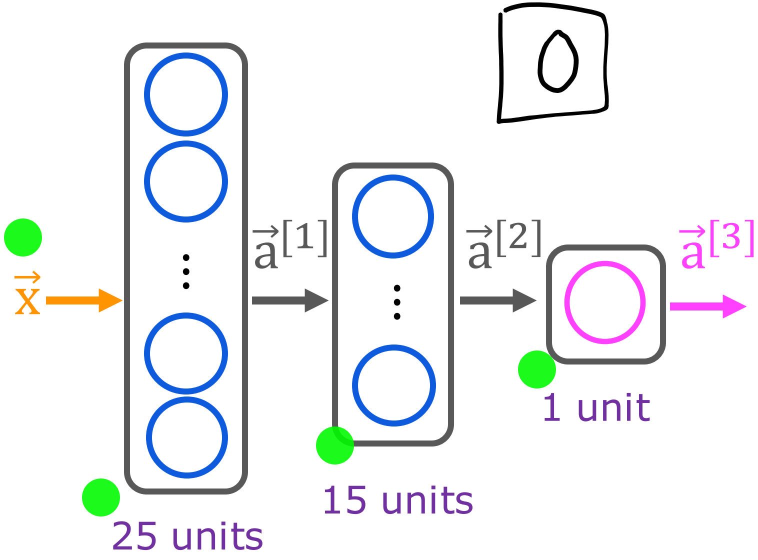
Stanford
ONLINE



Neural Network Training

TensorFlow implementation

Train a Neural Network in TensorFlow



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
  model = Sequential([
    Dense(units=25, activation='sigmoid’)
    Dense(units=15, activation='sigmoid’)
    Dense(units=1, activation='sigmoid’)
                  )]
```

①

```
from tensorflow.keras.losses import
BinaryCrossentropy
   model.compile(loss=BinaryCrossentropy())
```

②

```
   model.fit(X,Y,epochs=100)
```

③

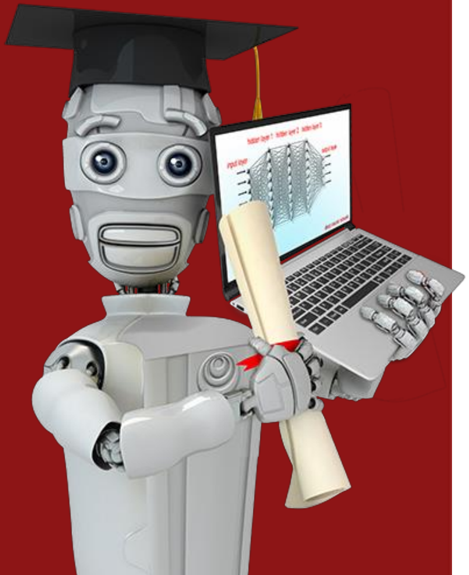
epochs: number of steps in gradient descent

Given set of (x,y) examples

How to build and train this in code?

DeepLearning.AI

Stanford
ONLINE



Neural Network Training

Training Details

Model Training Steps

TensorFlow

① specify how to compute output given input x and parameters w,b (define model)

$f_{\vec{w},b}(\vec{x}) = ?$

② specify loss and cost

$\boxed{L(\underline{f_{\vec{w},b}(\vec{x})}, \underline{y})}$ 1 example

$$J(\vec{w},b) = \frac{1}{m}\boxed{\sum_{i=1}^{m}} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$

③ Train on data to minimize $J(\underline{\vec{w},b})$

logistic regression

```
z = np.dot(w,x) + b

f_x = 1/(1+np.exp(-z))
```

logistic loss

```
loss = -y * np.log(f_x)
   -(1-y) * np.log(1-f_x)
```

```
w = w - alpha * dj_dw
b = b - alpha * dj_db
```

neural network

```
model = Sequential([
        Dense(...)
        Dense(...)
        Dense(...)      ])
```

binary cross entropy

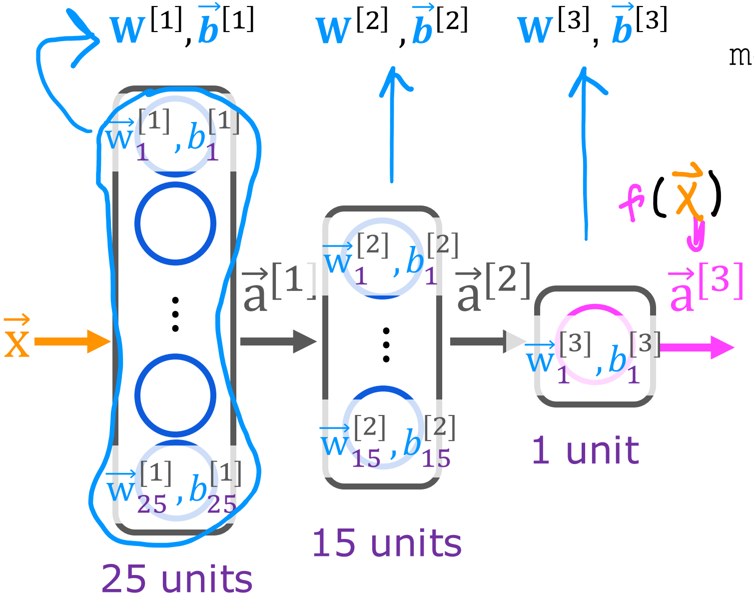
```
model.compile(
loss=BinaryCrossentropy())
```

```
model.fit(X,y,epochs=100)
```

1. Create the model

define the model

$f(\vec{x}) = ?$



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
  Dense(units=25, activation='sigmoid')
  Dense(units=15, activation='sigmoid')
  Dense(units=1, activation='sigmoid')
                    ])
```

2. Loss and cost functions

Mnist digit classification problem

binary classification

$$L(f(\vec{x}), y) = -y\log(f(\vec{x})) - (1-y)\log(1-f(\vec{x}))$$

compare prediction vs. target

logistic loss

also known as binary cross entropy

```
model.compile(loss= BinaryCrossentropy())
```

regression (predicting numbers and not categories)

mean squared error

```
model.compile(loss= MeanSquaredError())
```

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\vec{x}^{(i)}), y^{(i)})$$

$\mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]}$ $\vec{b}^{[1]}, \vec{b}^{[2]}, \vec{b}^{[3]}$

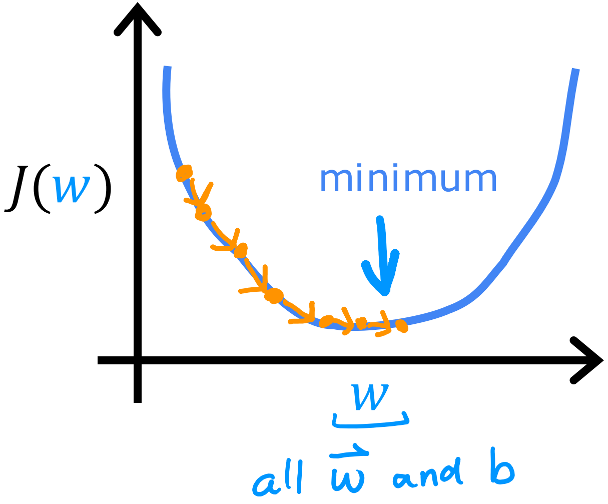
$f_{W,B}(\vec{x})$

```
from tensorflow.keras.losses import
  BinaryCrossentropy
```

Keras

```
from tensorflow.keras.losses import
  MeanSquaredError
```

3. Gradient descent



repeat {

$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{\mathrm{w}}, b)$$

$$b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

}

Compute derivatives for gradient descent using "backpropagation"

```
model.fit(X,y,epochs=100)
```

Neural network libraries

Use code libraries instead of coding ”from scratch”

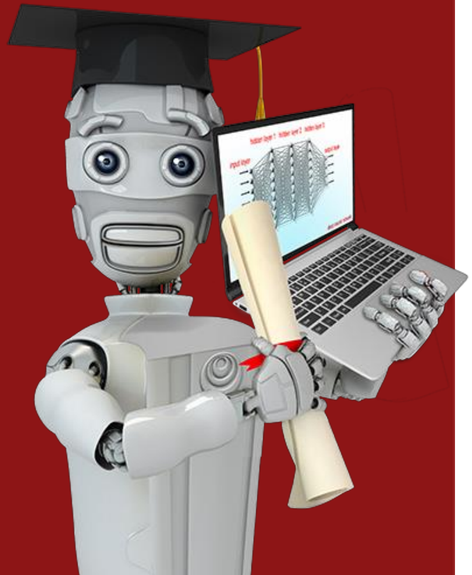
TensorFlow

PyTorch

Good to understand the implementation
(for tuning and debugging).

DeepLearning.AI

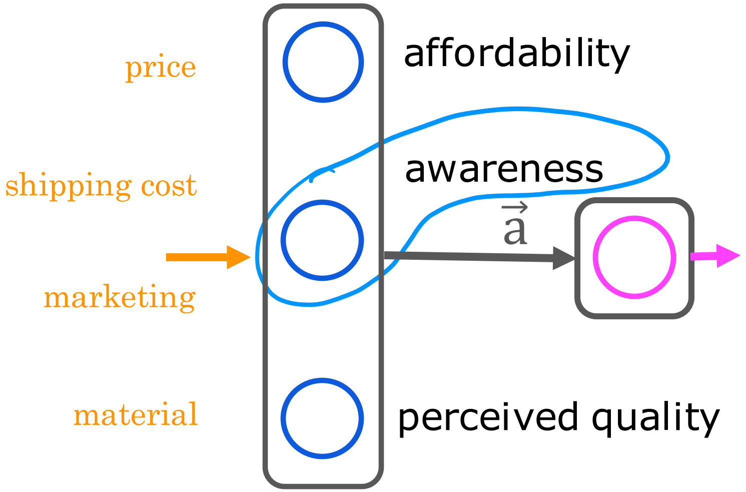
Stanford
ONLINE



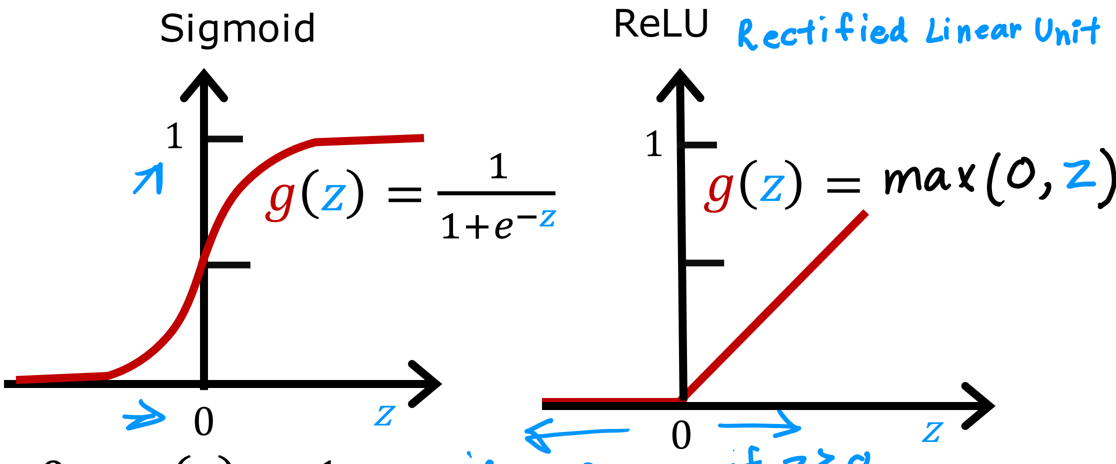
Activation Functions

Alternatives to the sigmoid activation

Demand Prediction Example



$$a_2^{[1]} = g(\overbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}^{z})$$



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

$$0 < g(z) < 1$$

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

if $z<0$
$g(z)$ is 0

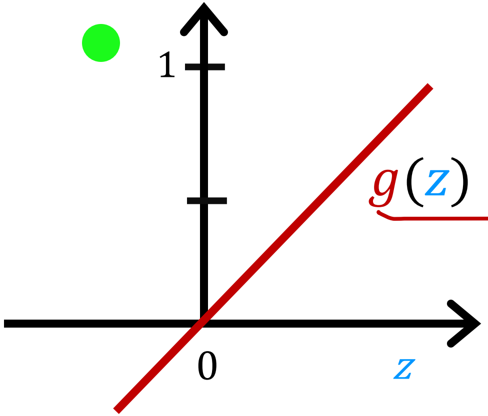
if $z \geq 0$
$g(z)$ is z

Examples of Activation Functions

$$a_2^{[1]} = g(\overbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}^{z})$$

"No activation function"

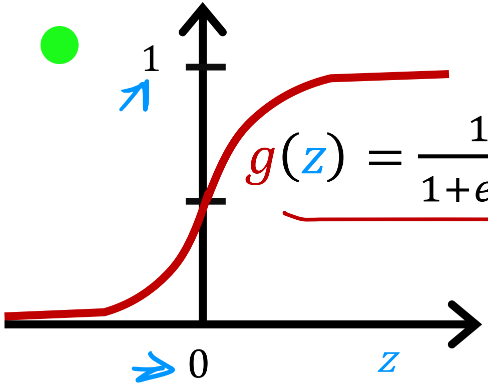
Linear activation function



$\underline{g(z) = z}$

$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$

Sigmoid



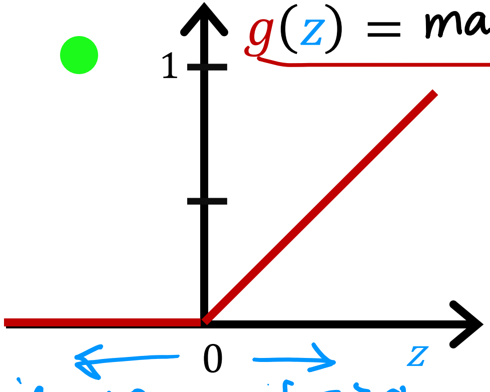
$\underline{g(z) = \frac{1}{1+e^{-z}}}$

$0 < g(z) < 1$

Later: softmax activation

ReLU Rectified Linear Unit

$\underline{g(z) = \max(0, z)}$

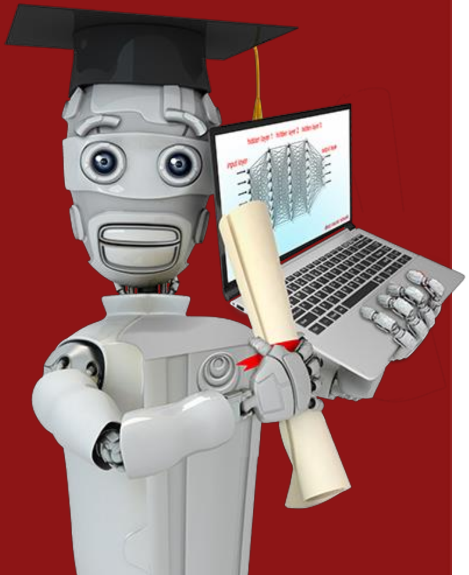


if $z<0$ $g(z)$ is 0

if $z \geq 0$ $g(z)$ is z

DeepLearning.AI

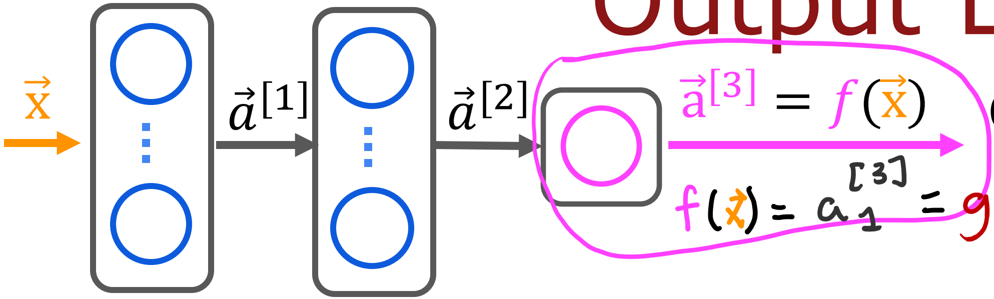
Stanford
ONLINE



Activation Functions

Choosing activation functions

Output Layer



$\vec{a}^{[3]} = f(\vec{x})$

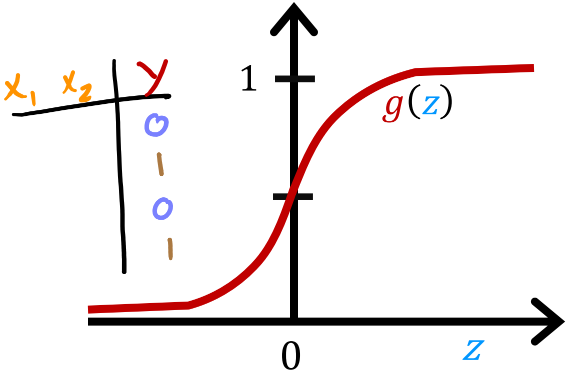
$f(\vec{x}) = a_1^{[3]} = g(z_1^{[3]})$

Choosing $g(z)$ for output layer?

Binary classification

Sigmoid

y=0/1

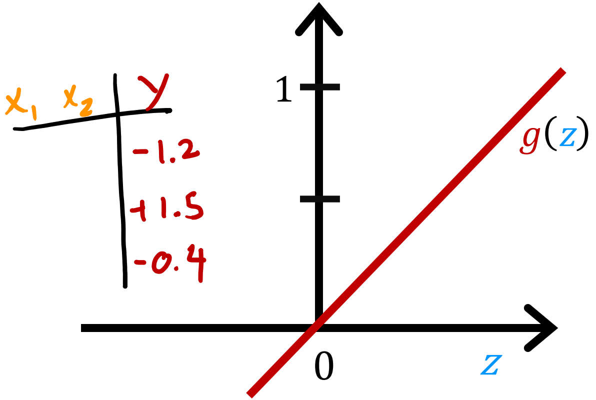


x_1	x_2	y
		0
		1
		0
		1

Regression

Linear activation function

y = +/-

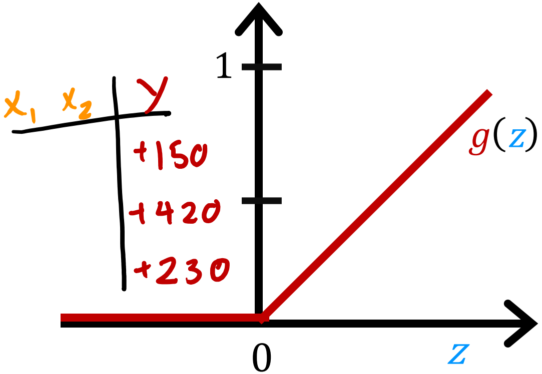


x_1	x_2	y
		-1.2
		+1.5
		-0.4

Regression

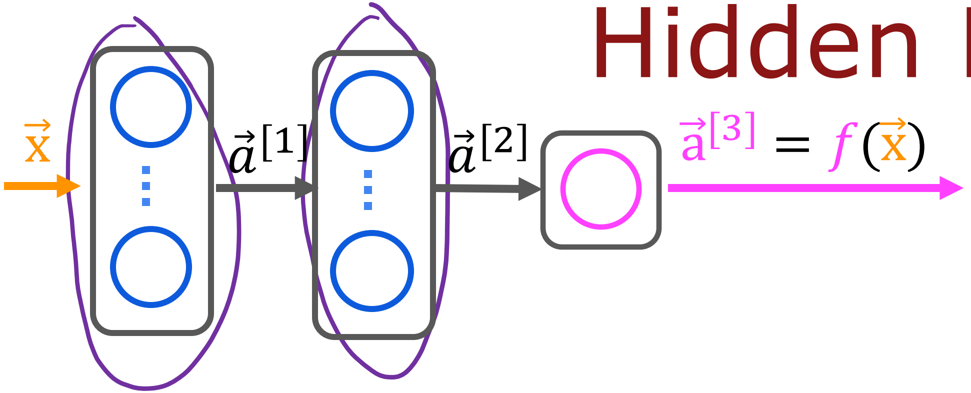
ReLU

Y = 0 or +



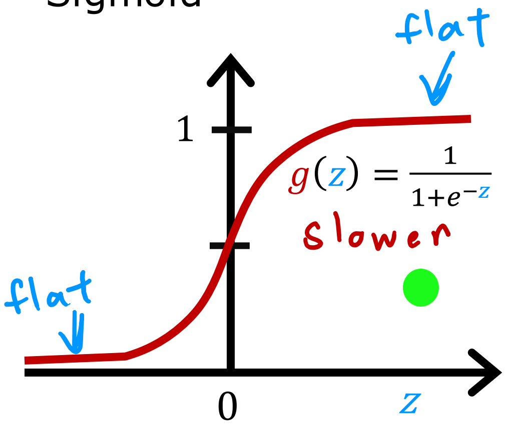
x_1	x_2	y
		+150
		+420
		+230

Hidden Layer

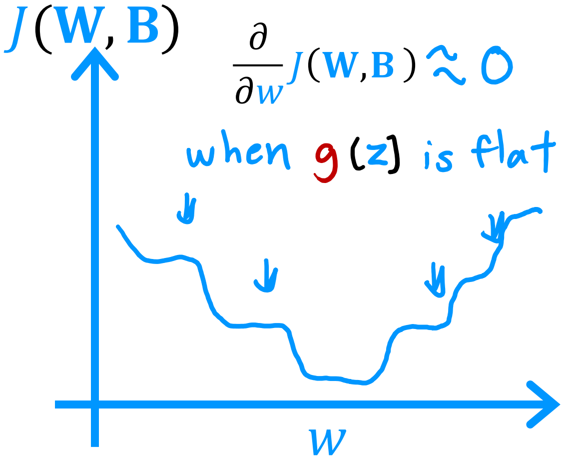


Choosing $g(z)$ for hidden layer

Sigmoid



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



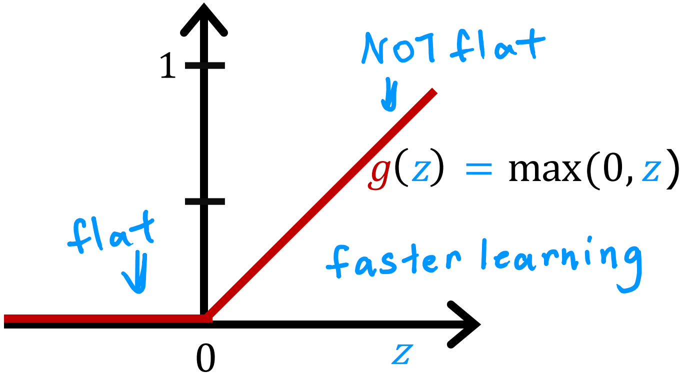
$\frac{\partial}{\partial w}J(\mathbf{W},\mathbf{B}) \approx 0$

when $g(z)$ is flat

most common choice

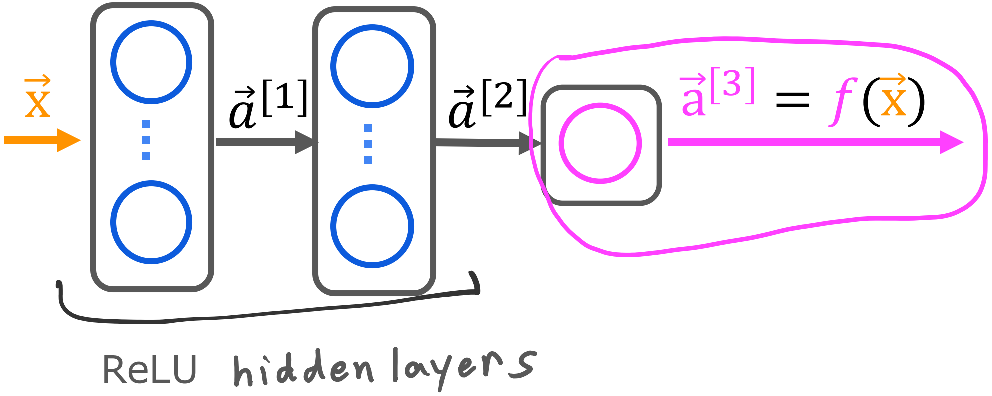
ReLU

faster



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

Choosing Activation Summary



binary classification

activation='sigmoid'

regression y negative/positive

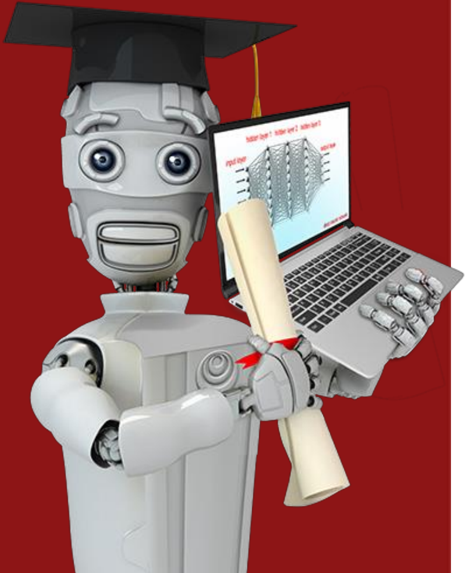
activation='linear'

regression y ≥ 0

activation='relu'

```
from tf.keras.layers import Dense
model = Sequential([
  Dense(units=25, activation='relu'),     layer1
  Dense(units=15, activation='relu'),     layer2
  Dense(units=1,  activation='sigmoid')   layer3
])
```

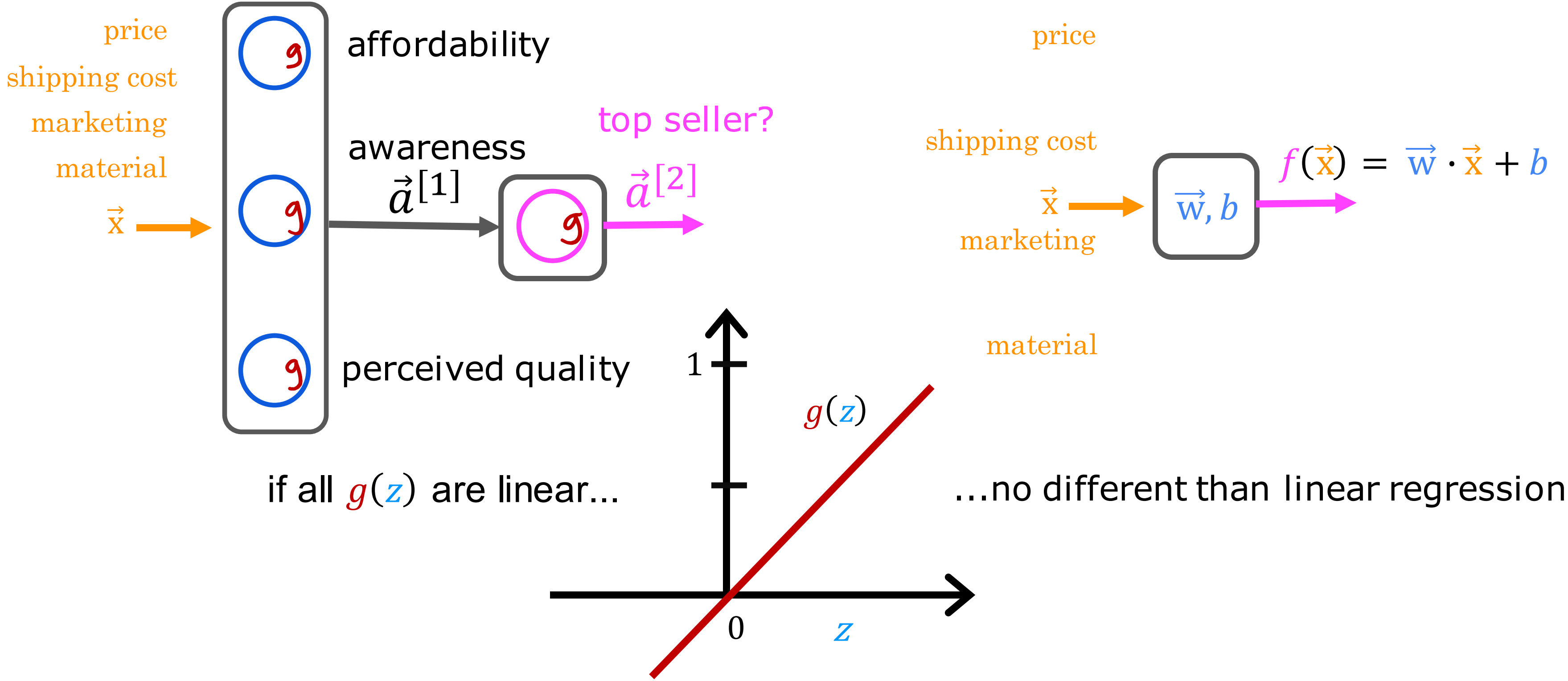
or 'linear'
or 'relu'



Activation Functions

Why do we need activation functions?

Why do we need activation functions?

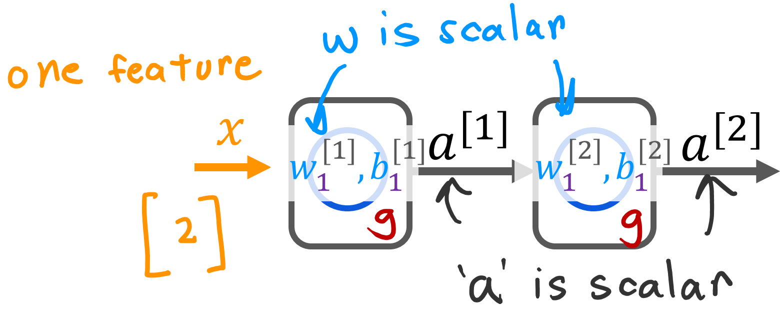


$f(\vec{x}) = \vec{w} \cdot \vec{x} + b$

if all $g(z)$ are linear...

...no different than linear regression

Linear Example



$$g(z) = z$$

$$a^{[1]} = \underbrace{w_1^{[1]} x + b_1^{[1]}}_{\downarrow}$$

$$a^{[2]} = w_1^{[2]} a^{[1]} + b_1^{[2]}$$

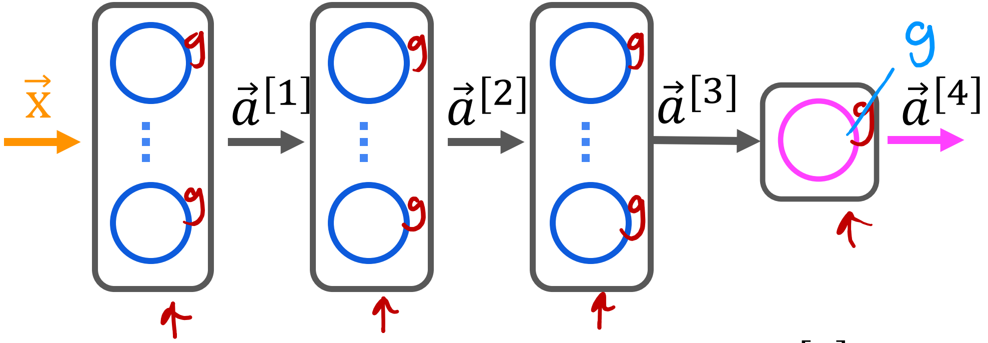
$$= w_1^{[2]} (w_1^{[1]} x + b_1^{[1]}) + b_1^{[2]}$$

$$\vec{a}^{[2]} = \underbrace{(\vec{w}_1^{[2]} \vec{w}_1^{[1]})}_{w} x + \underbrace{w_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b}$$

$$\vec{a}^{[2]} = w x + b$$

$f(x) = wx + b$ linear regression

Example



$g(z) = z$

$$\vec{a}^{[4]} = \vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]}$$

all linear (including output)
↳ equivalent to linear regression

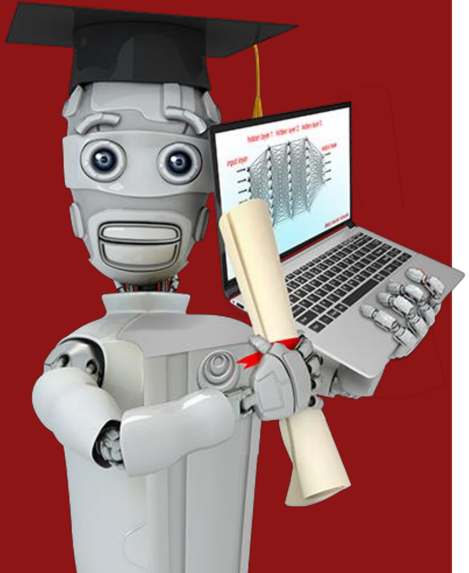
$$\vec{a}^{[4]} = \frac{1}{1+e^{-(\vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]})}}$$

output activation is sigmoid
(hidden layers still linear)
↳ equivalent to logistic regression

Don't use linear activations in hidden layers (use ReLU)

DeepLearning.AI

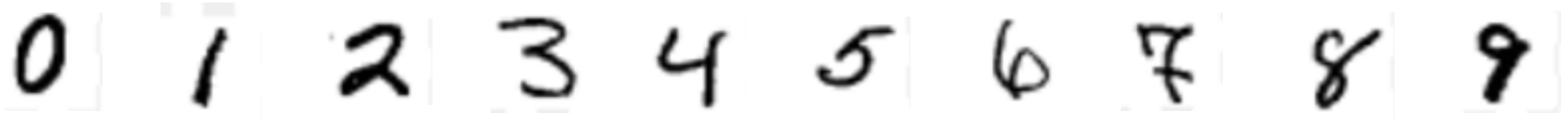
Stanford
ONLINE



Multiclass Classification

Multiclass

MNIST example



$y = 0$ 1 2 3 4 5 6 7 8 9

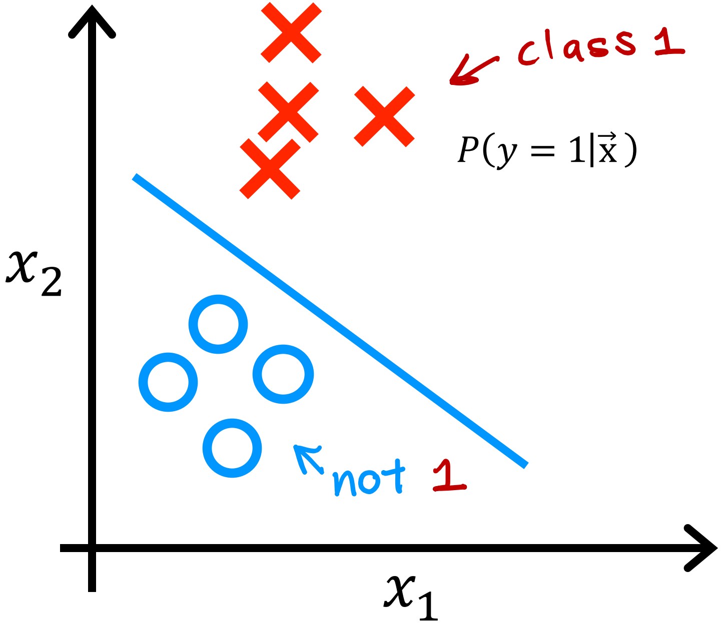
x

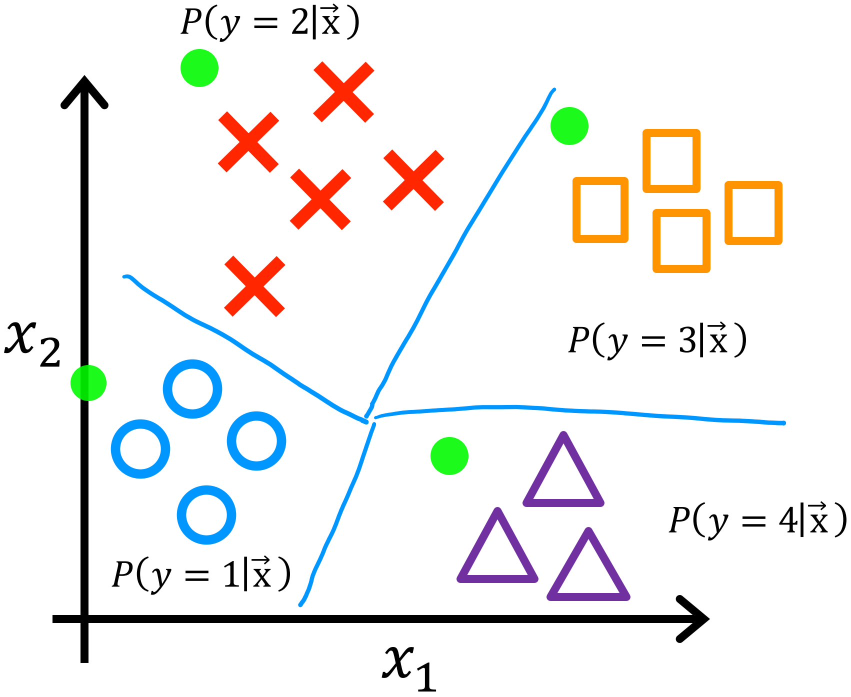
$y = 7$

multiclass classification problem:

target y can take on more than two possible values

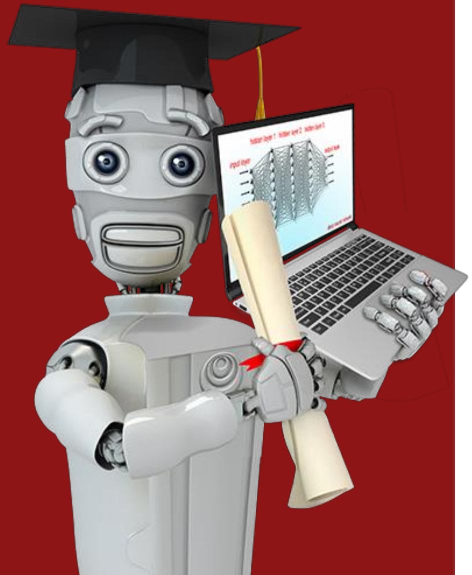
Multiclass classification example





DeepLearning.AI

Stanford
ONLINE



Multiclass Classification

Softmax

Logistic regression
(2 possible output values)

$z = \vec{w} \cdot \vec{x} + b$

0.71

× $a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y = 1|\vec{x})$

○ $a_2 = 1 - a_1 = P(y = 0|\vec{x})$

0.29

Softmax regression
(N possible outputs) $y = 1, 2, 3, \dots, N$

$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad j = 1, \dots, N$

parameters $w_1, w_2, \dots, w_N$
$b_1, b_2, \dots, b_N$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y = j|\vec{x})$$

note: $a_1 + a_2 + \dots + a_N = 1$

Softmax regression (4 possible outputs) $y = 1, 2, 3, 4$

× $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$

$$a_1 = \frac{e^{z_1}}{\underset{×}{e^{z_1}} + \underset{○}{e^{z_2}} + \underset{□}{e^{z_3}} + \underset{△}{e^{z_4}}}$$

$= P(y = 1|\vec{x})$ 0.30

○ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$

$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$= P(y = 2|\vec{x})$ 0.20

□ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$

$$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$= P(y = 3|\vec{x})$ 0.15

△ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$

$$a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

$= P(y = 4|\vec{x})$ 0.35

Cost

Logistic regression

$$z = \vec{w} \cdot \vec{x} + b$$

$$a_1 = g(z) = \frac{1}{\underbrace{1 + e^{-z}}} \quad = P(y = 1|\vec{x})$$

$$a_2 = \underbrace{1 - a_1} \quad = P(y = 0|\vec{x})$$

$$\text{loss} = -y \underbrace{\log a_1}_{\text{if } y=1} - (1-y) \underbrace{\log\overbrace{(1 - a_1)}^{a_2}}_{\text{if } y=0}$$

$$J(\vec{w}, b) = \text{average loss}$$

Softmax regression

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} \quad = P(y = 1|\vec{x})$$

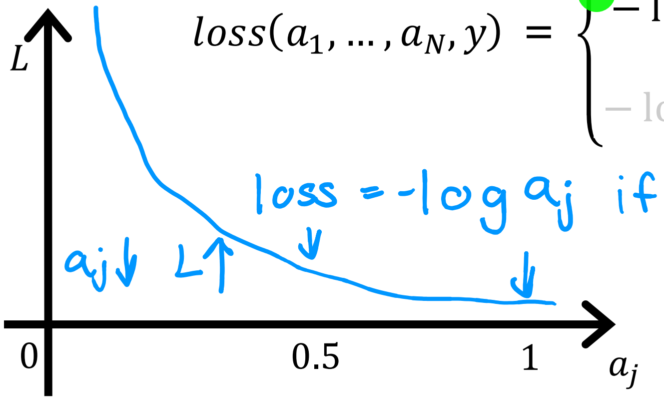
$$\vdots$$

$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} \quad = P(y = N|\vec{x})$$

Crossentropy loss

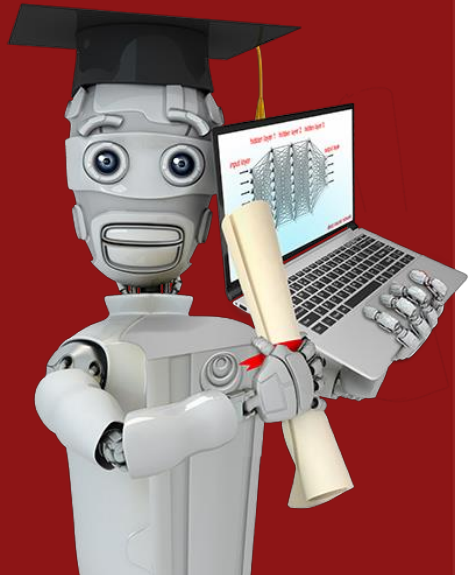
$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ -\log a_2 & \text{if } y = 2 \\ \quad \vdots & \\ -\log a_N & \text{if } y = N \end{cases}$$

$$\text{loss} = -\log a_j \quad \text{if } y = j$$



DeepLearning.AI

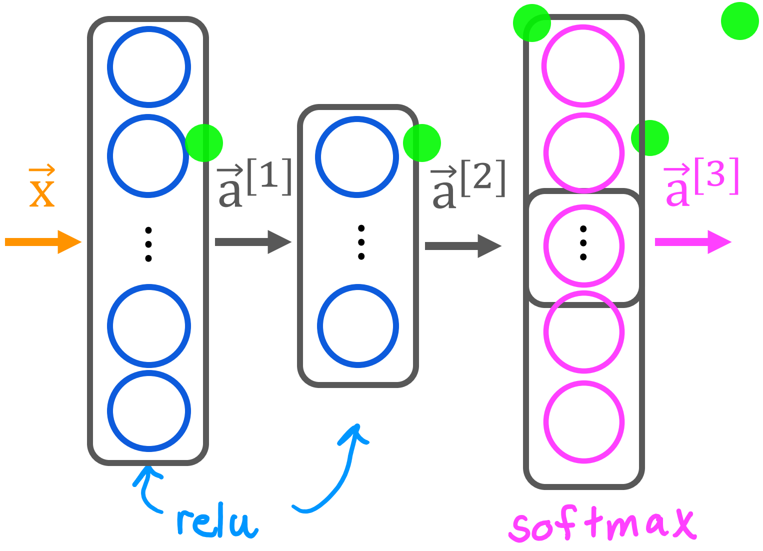
Stanford
ONLINE



Multiclass Classification

Neural Network with Softmax output

Neural Network with Softmax output



25 units 15 units 10 units

10 classes

$$z_1 = \vec{w}_1 \cdot \vec{a}^{[2]} + b_1 \qquad a_1 = \frac{e^{z_1}}{e^{z_1} + \cdots + e^{z_{10}}}$$

$$= P(y = 1|\vec{x})$$

$$\vdots$$

$$z_{10} = \vec{w}_{10} \cdot \vec{a}^{[2]} + b_{10} \qquad a_{10} = \frac{e^{z_{10}}}{e^{z_1} + \cdots + e^{z_{10}}}$$

$$= P(y = 10|\vec{x})$$

logistic regression

$$a_1^{[3]} = g\left(z_1^{[3]}\right) \qquad a_2^{[3]} = g\left(z_2^{[3]}\right)$$

softmax

$$\vec{a}^{[3]} = \left(a_1^{[3]}, \ldots a_{10}^{[3]}\right) = g\left(z_1^{[3]}, \ldots, z_{10}^{[3]}\right)$$

MNIST with softmax

① specify the model

$f_{\vec{w},b}(\vec{x}) = ?$

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='relu')
    Dense(units=15, activation='relu')
    Dense(units=10, activation='softmax')
                    )]
```

② specify loss and cost

$L(f_{\vec{w},b}(\vec{x}), y)$

```
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy() )
```

③ Train on data to minimize $J(\vec{w}, b)$

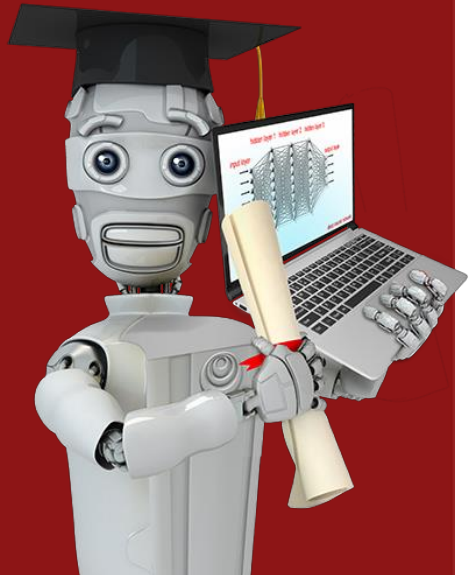
```
model.fit(X,Y,epochs=100)
```

Note: better (recommended) version later.

Don't use the version shown here!

DeepLearning.AI

Stanford
ONLINE



Multiclass Classification

Improved implementation of softmax

Numerical Roundoff Errors

option 1

$$x = \frac{2}{10{,}000}$$

option 2

$$x = \underbrace{\left(1 + \frac{1}{10{,}000}\right)}_{} - \underbrace{\left(1 - \frac{1}{10{,}000}\right)}_{} =$$

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

$1 + \frac{1}{10,000}$ $1 - \frac{1}{10,000}$

Logistic regression:

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

```
model = Sequential([
  Dense(units=25, activation='relu')
  Dense(units=15, activation='relu')
  Dense(units=10, activation='sigmoid')
```

~~sigmoid~~ 'linear'

~~model.compile(loss=BinaryCrossEntropy())~~

```
model.compile(loss=BinaryCrossEntropy(from_logits=True) )
```

Original loss

$$loss = -y\log(a) - (1-y)\log(1-a)$$

More accurate loss (in code)

$$loss = -y\log\left(\frac{1}{1+e^{-z}}\right) - (1-y)\log(1-\frac{1}{1+e^{-z}})$$

logit: z

More numerically accurate implementation of softmax

Softmax regression

$$(a_1, \dots, a_{10}) = g(z_1, \dots, z_{10})$$

$$\text{Loss} = L(\vec{a}, y) = \begin{cases} -\log \boxed{a_1} & \text{if } y = 1 \\ \vdots & \\ -\log \boxed{a_{10}} & \text{if } y = 10 \end{cases}$$

`model = Sequential([`

`Dense(units=25, activation='relu')`

`Dense(units=15, activation='relu')`

`Dense(units=10, activation='`~~`softmax`~~`')`

'linear'

~~`model.compile(loss=SparseCategoricalCrossEntropy()`~~` )`

More Accurate

$$L(\vec{a}, y) = \begin{cases} -\log \boxed{\dfrac{e^{\boxed{z_1}}}{e^{z_1} + \cdots + e^{z_{10}}}} \text{if } y = 1 \\ \vdots \\ -\log \boxed{\dfrac{e^{\boxed{z_{10}}}}{e^{z_1} + \cdots + e^{z_{10}}}} \text{if } y = 10 \end{cases}$$

`model.compile(loss=SparseCrossEntropy(from_logits=True) )`

MNIST (more numerically accurate)

model

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
  Dense(units=25, activation='relu')
  Dense(units=15, activation='relu')
  Dense(units=10, activation='linear’) )]
```

loss

```
from tensorflow.keras.losses import
  SparseCategoricalCrossentropy
model.compile(...,loss=SparseCategoricalCrossentropy(from_logits=True) )
```

fit

```
model.fit(X,Y,epochs=100)
```

predict

```
logits = model(X)
f_x = tf.nn.softmax(logits)
```

not $a_1 \ldots a_{10}$

is $z_1 \ldots z_{10}$

logistic regression
(more numerically accurate)

model

```
model = Sequential([
  Dense(units=25, activation='sigmoid')
  Dense(units=15, activation='sigmoid')
  Dense(units=1, activation='linear')
from tensorflow.keras.losses import
  BinaryCrossentropy
```

loss

```
model.compile(..., BinaryCrossentropy(from_logits=True)) )

model.fit(X,Y,epochs=100)
```

fit

```
logit = model(X)
```

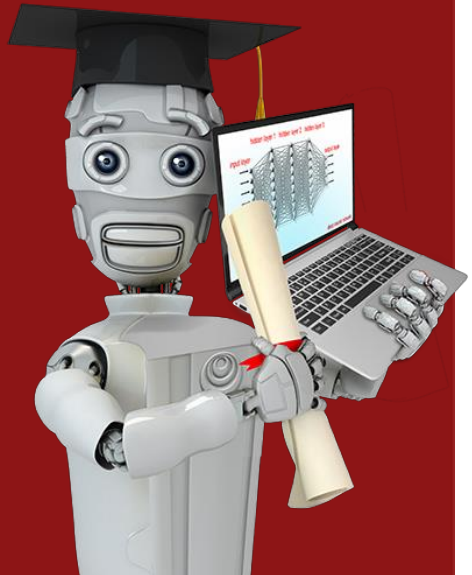
z

predict

```
f_x = tf.nn.sigmoid(logit)
```

DeepLearning.AI

Stanford
ONLINE



Multi-label Classification

Classification with multiple outputs (Optional)

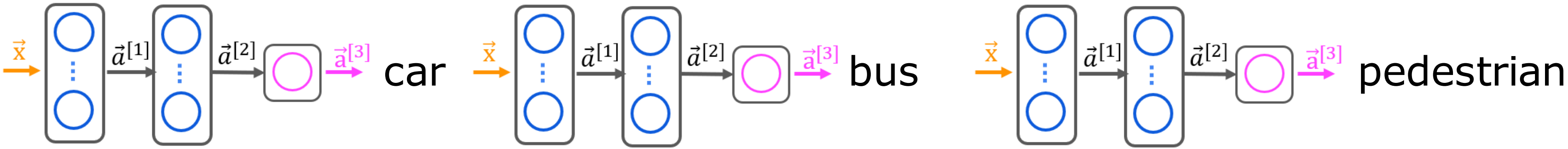
Multi-label Classification

$\vec{x}$

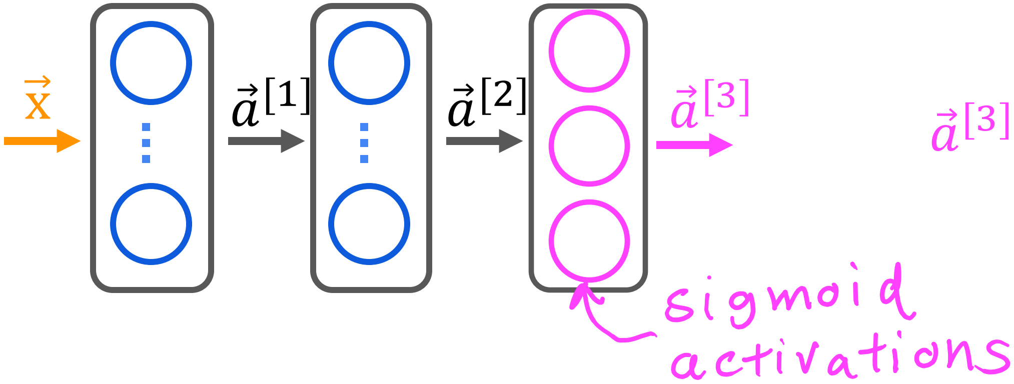
Is there a car?	yes	no	yes
Is there a bus?	no	no	yes
Is there a pedestrian	yes	yes	no

$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ $y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

Multiple classes



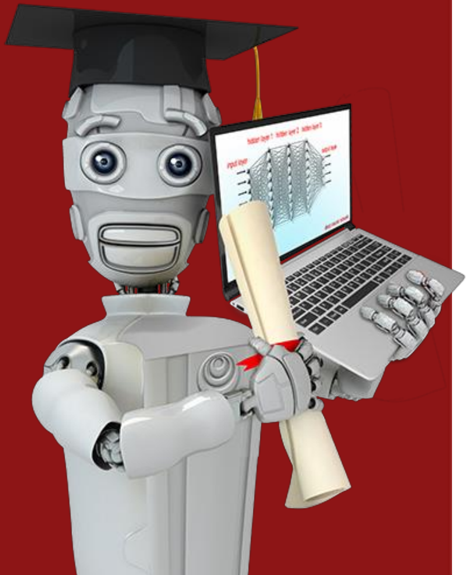
Alternatively, train one neural network with three outputs



$$\vec{a}^{[3]} = \begin{array}{cl} \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix} & \begin{array}{l} \text{car} \\ \text{bus} \\ \text{pedestrian} \end{array} \end{array}$$

DeepLearning.AI

Stanford
ONLINE



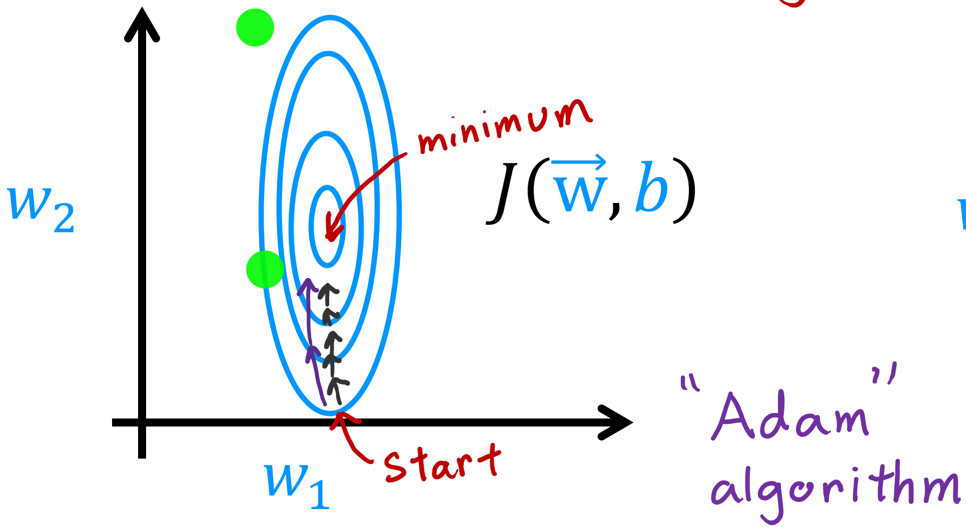
Additional Neural Network Concepts

Advanced Optimization

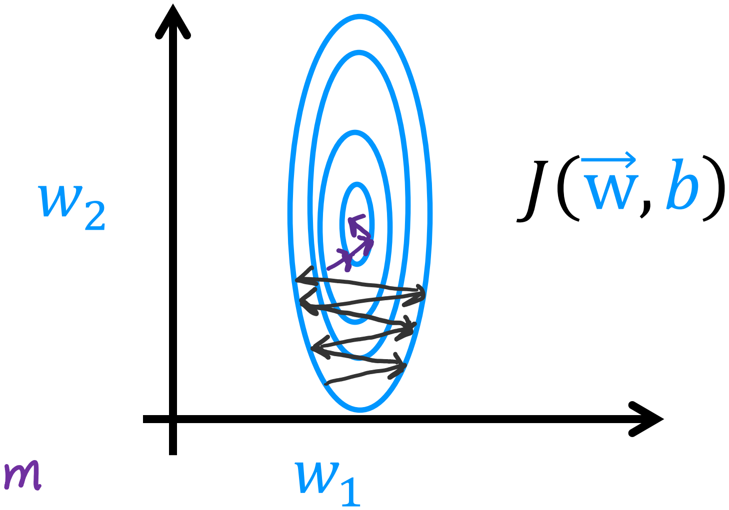
Gradient Descent

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

learning rate



Go faster – increase α



Go slower – decrease α

Adam Algorithm Intuition

Adam: Adaptive Moment estimation

not just one α

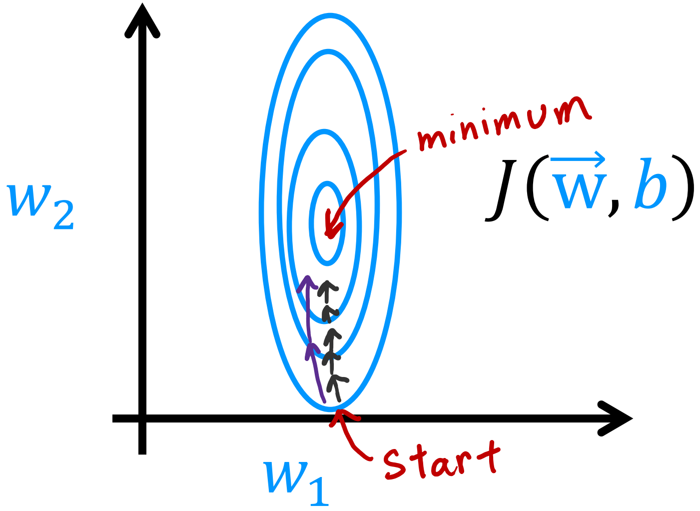
$$w_1 = w_1 - \underbrace{\alpha_1}_{} \frac{\partial}{\partial w_1} J(\vec{\mathrm{w}}, b)$$

$$\vdots$$

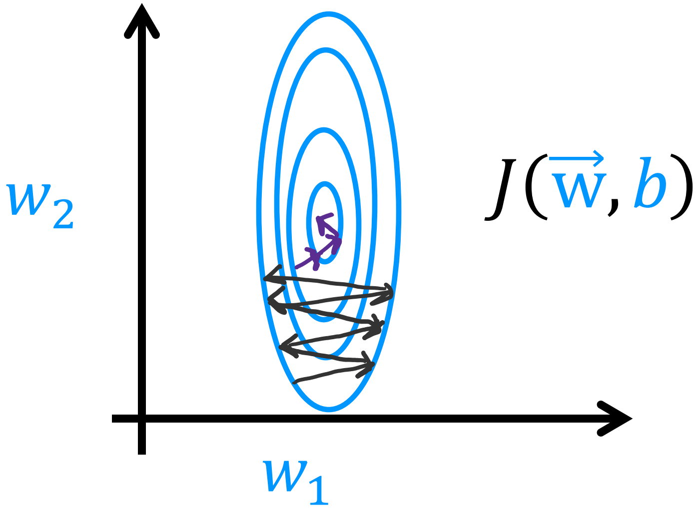
$$w_{10} = w_{10} - \underbrace{\alpha_{10}}_{} \frac{\partial}{\partial w_{10}} J(\vec{\mathrm{w}}, b)$$

$$b = b - \underbrace{\alpha_{11}}_{} \frac{\partial}{\partial b} J(\vec{\mathrm{w}}, b)$$

Adam Algorithm Intuition



If w_j (or b) keeps moving in same direction, $\underbrace{\text{increase}}$ α_j.



If w_j (or b) keeps oscillating, $\underbrace{\text{reduce}}$ α_j.

MNIST Adam

model

```
model = Sequential([
            tf.keras.layers.Dense(units=25, activation='sigmoid')
            tf.keras.layers.Dense(units=15, activation='sigmoid')
            tf.keras.layers.Dense(units=10, activation='linear')
 ])
```

compile

$\alpha = 10^{-3} = 0.001$

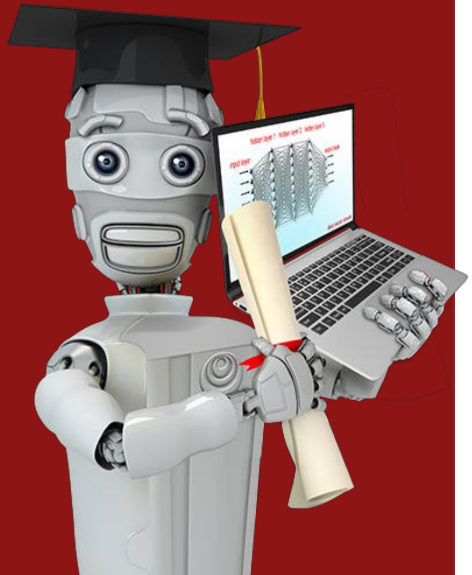
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

```
model.fit(X,Y,epochs=100)
```

DeepLearning.AI

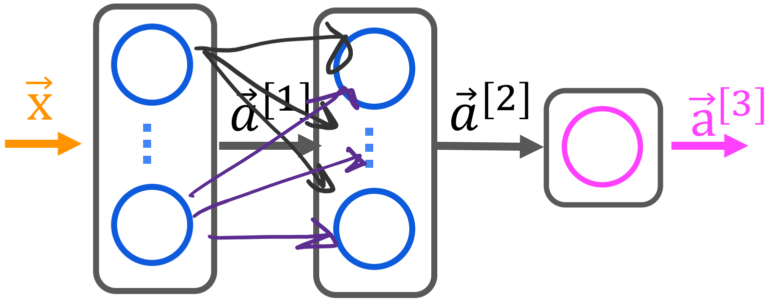
Stanford
ONLINE



Additional Neural Network Concepts

Additional Layer Types

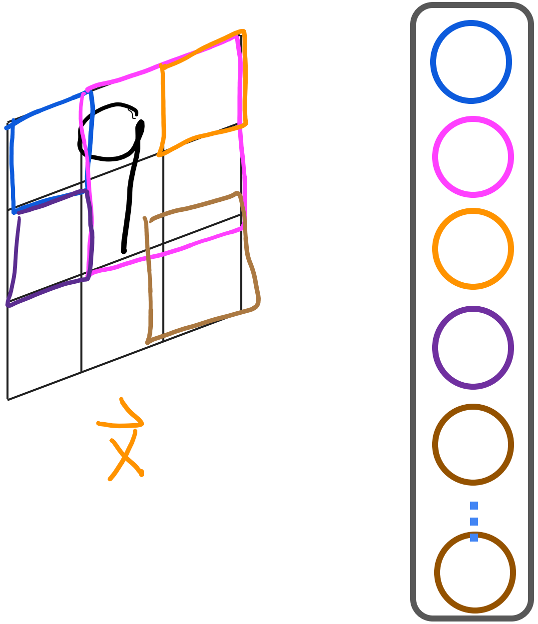
Dense Layer



Each neuron output is a function of
$\underbrace{\text{all}}$ the activation outputs of the previous layer.

$$\vec{a}_1^{[2]} = g\left(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}\right)$$

Convolutional Layer

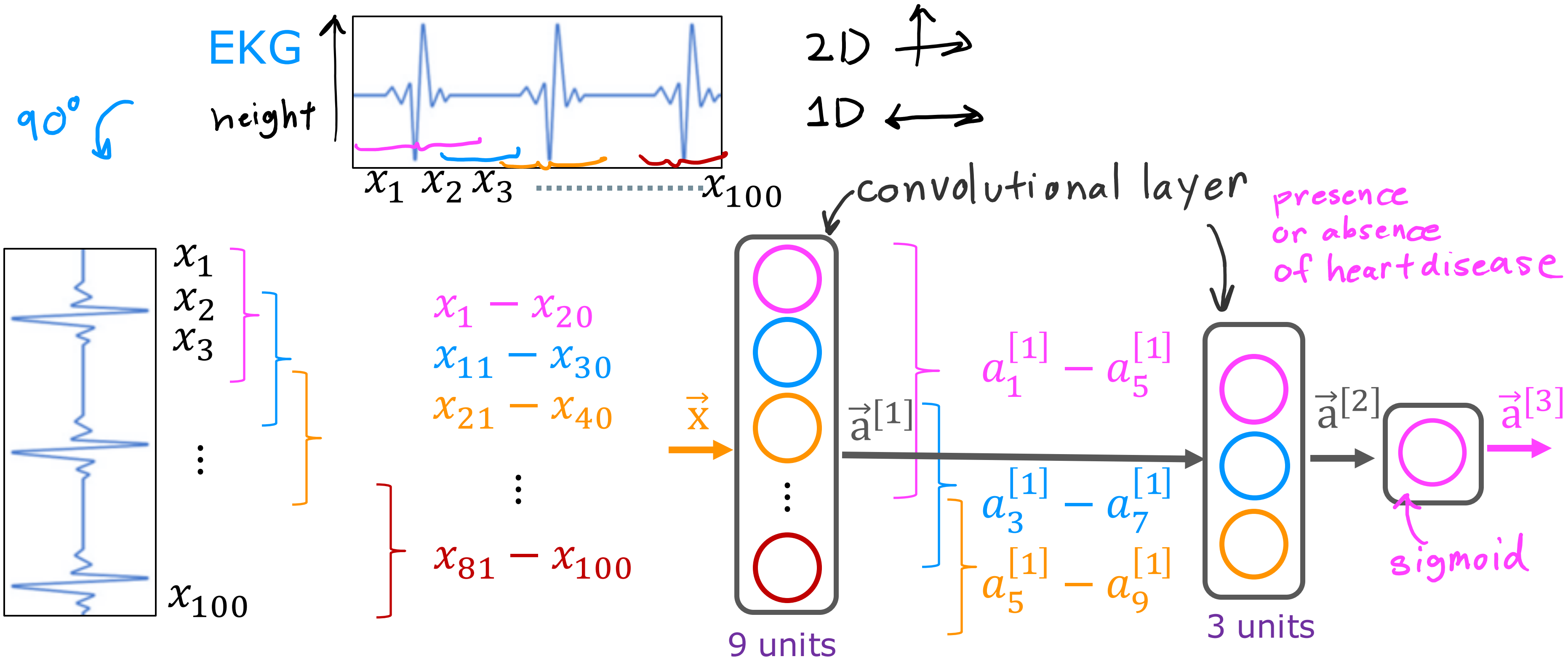


Each Neuron only looks at part of the previous layer's inputs.

Why?

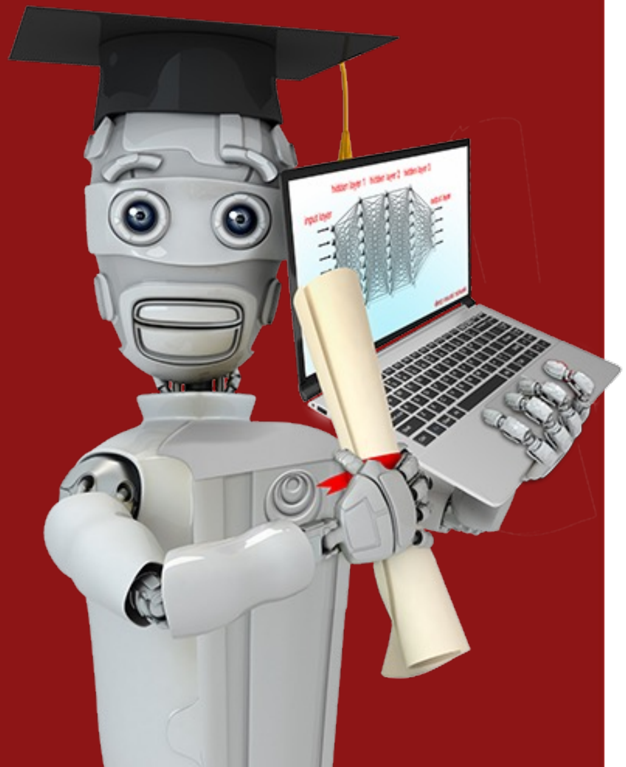
- Faster computation
- Need less training data (less prone to overfitting)

Convolutional Neural Network



DeepLearning.AI

Stanford
ONLINE



Backprop Intuition (Optional)

What is a derivative?

Derivative Example

Cost function $J(w) = w^2$

Say $w = 3$ $\quad J(w) = 3^2 = 9$

$\varepsilon = 0.002$

If we increase w by a tiny amount $\varepsilon = 0.001$ how does $J(w)$ change?

$w = 3 + \cancel{0.001}\ 0.002$

$J(w) = w^2 = \cancel{9.006001}$

$\underbrace{9.012}_{9.012}\underbrace{004}$

If $w \uparrow \overset{0.002}{\cancel{0.001}}$ $\qquad \varepsilon$ ←

$J(w) \uparrow 6 \times \overset{0.002}{\cancel{0.001}}$ $\qquad 6 \times \varepsilon \qquad 6 \times 0.002 = 0.012$

$\frac{\partial}{\partial w} J(w) = 6$

Informal Definition of Derivative

If $\underline{w \uparrow \varepsilon}$ causes $\underline{J(w) \uparrow}\; \underset{\overset{\uparrow}{6}}{\boxed{k \times \varepsilon}}$ then

$$\frac{\partial}{\partial w} J(w) = k \rightarrow 6$$

Gradient descent

```
repeat {
    $w_j = w_j - \overset{\text{learning rate}}{\alpha} \frac{\partial}{\partial w_j} J(\vec{w}, b)$
}
```

If derivative is $\underline{\text{small,}}$ then this update step will make a $\underline{\text{small}}$ update to w_j

If the derivative is $\underline{\text{large,}}$ then this update step will make a $\underline{\text{large}}$ update to w_j

More Derivative Examples

$w = 3$ $J(w) = w^2 = 9$ $w \uparrow 0.001$ $J(w) = J(3.001) = 9.006001$ $\frac{\partial}{\partial w} J(w) = 6$

$J(w) \uparrow 6 \times 0.001$

$w = 2$ $J(w) = w^2 = 4$ $w \uparrow 0.001$ $J(w) = J(2.001) = 4.004001$ $\frac{\partial}{\partial w} J(w) = 4$

$J(w) \uparrow 4 \times 0.001$

$J(w) = w^2$

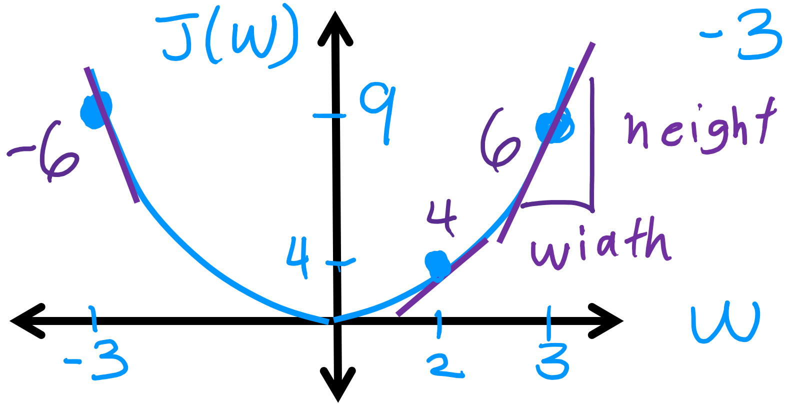
$\downarrow 0.006$

$w = -3$ $J(w) = w^2 = 9$ $w \uparrow 0.001$ $J(w) = J(-2.999) = 8.994001$ $\frac{\partial}{\partial w} J(w) = -6$

$-3 + 0.001$

$J(w) \downarrow 6 \times 0.001$

$J(w) \uparrow -6 \times 0.001$



Calculus

$\frac{\partial}{\partial w} J(w) = 2w$

w	$\partial J(w)/\partial w$
3	$2 \times 3 = 6$
2	$2 \times 2 = 4$
-3	$2 \times -3 = -6$

Even More Derivative Examples

$w = 2$

$J(w) = w^2 = 4$ $\quad \frac{\partial}{\partial w} J(w) = 2w = 4$ $\quad w \uparrow \underbrace{0.001}_{\varepsilon}$ $\quad J(w) = 4.004001$

$J(w) \uparrow 4 \times \varepsilon$

$J(w) = w^3 = \boxed{8}$ $\quad \frac{\partial}{\partial w} J(w) = 3w^2 = \boxed{12}$ $\quad w \uparrow \varepsilon$ $\quad J(w) = 8.012006$

$J(w) \uparrow \boxed{12} \times \varepsilon$

$J(w) = w = 2$ $\quad \frac{\partial}{\partial w} J(w) = \boxed{1}$ $\quad w \uparrow \varepsilon$ $\quad J(w) = 2.001$

$J(w) \uparrow \boxed{1} \times \varepsilon$

-0.25×0.001

$J(w) = \frac{1}{w} = \frac{1}{2} = \boxed{0.5}$ $\quad \frac{\partial}{\partial w} J(w) = \frac{-1}{w^2} = \boxed{\frac{-1}{4}}$ $\quad w \uparrow \varepsilon$ $\quad 0.5 - 0.00025$

$w = \frac{1}{2.001}$

$J(w) = 0.49975$

$J(w) \uparrow \boxed{-\frac{1}{4}} \times \varepsilon$

$\frac{\partial}{\partial w} J(w) \quad w \uparrow \varepsilon \quad J(w) \uparrow \boxed{k} \times \varepsilon$

A note on derivative notation

If $J(w)$ is a function of one variable (w),

d $$\frac{d}{dw}J(w)$$

If $J(w_1, w_2, \ldots, w_n)$ is a function of more than one variable,

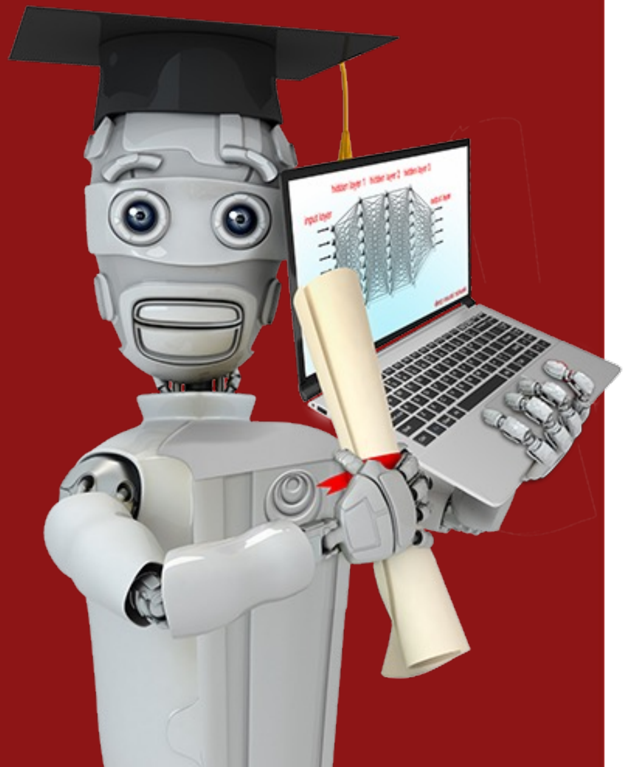
∂ $$\frac{\partial}{\partial w_i}J(w_1, w_2, \ldots, w_n) \qquad \frac{\partial J}{\partial w_i} \quad \text{or} \quad \frac{\partial}{\partial w_i}J$$

"partial derivative"

notation used in these courses

DeepLearning.AI

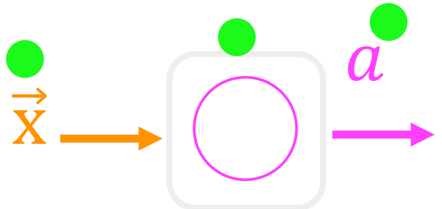
Stanford
ONLINE



Backprop Intuition (Optional)

Computation Graph

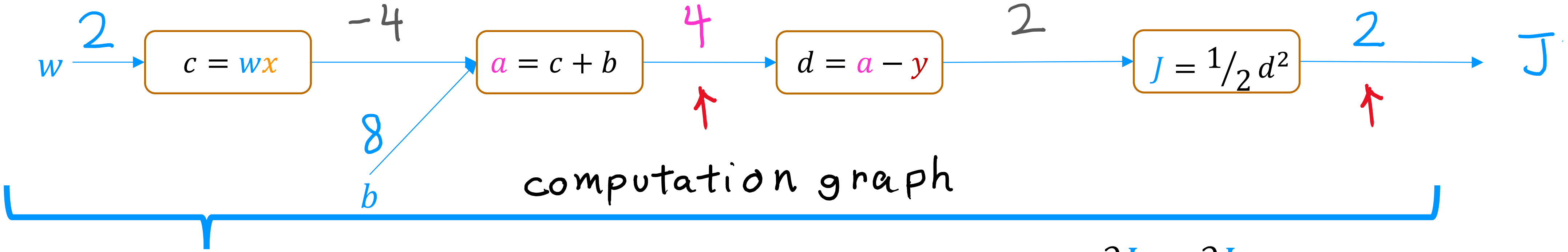
Small Neural Network Example



$w = 2$ $b = 8$ $x = -2$ $y = 2$

$a = wx + b$ linear activation $a = g(z) = z$

$$\underbrace{J(w,b)}_{} = \frac{1}{2}\underbrace{(a-y)}_{}^{2}$$



forward prop

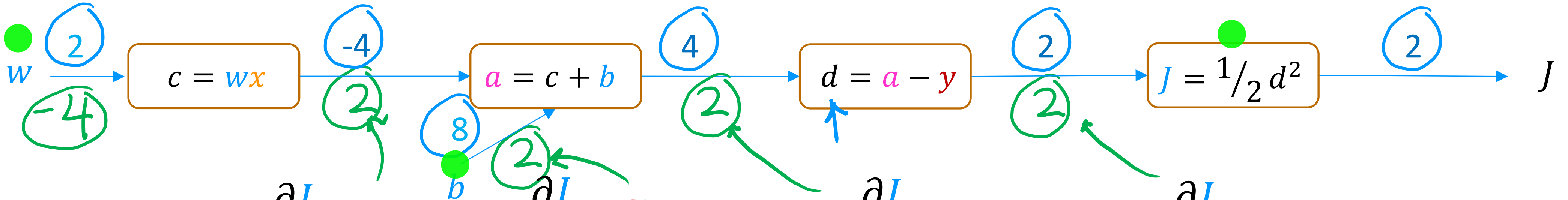
how do we find the derivatives of J? $\frac{\partial J}{\partial w}$ $\frac{\partial J}{\partial b}$

Computing the Derivatives

Forward prop →

Back prop ←

$w = 2$ $b = 8$ $x = -2$ $y = 2$ $a = wx + b$ $J = \frac{1}{2}(a - y)^2$



$\frac{\partial J}{\partial w} = ?$ -4 $\frac{\partial J}{\partial c} = ?$ 2 $\frac{\partial J}{\partial b} = ?$ 2 $\frac{\partial J}{\partial a} = ?$ 2 $\frac{\partial J}{\partial d} = 2$

w=2.001 c=-4.002

w ↑ 0.001 c ↓ 2×0.001

c ↑ −2×0.001

J ↑ −4×0.001

$$\frac{\partial J}{\partial w} = \underset{-2}{\frac{\partial c}{\partial w}} \times \underset{2}{\frac{\partial J}{\partial c}}$$

c ↑ 0.001 a ↑ 0.001 J ↑ 0.002

$$\frac{\partial J}{\partial c} = \underset{1}{\frac{\partial a}{\partial c}} \times \underset{2}{\frac{\partial J}{\partial a}}$$

b ↑ 0.001 a ↑ 0.001 J ↑ 0.002

$$\frac{\partial J}{\partial b} = \underset{1}{\frac{\partial a}{\partial b}} \times \underset{2}{\frac{\partial J}{\partial a}}$$

a ↑ 0.001 d ↑ 0.001

a=4.001 d=2.001

J ↑ 0.002

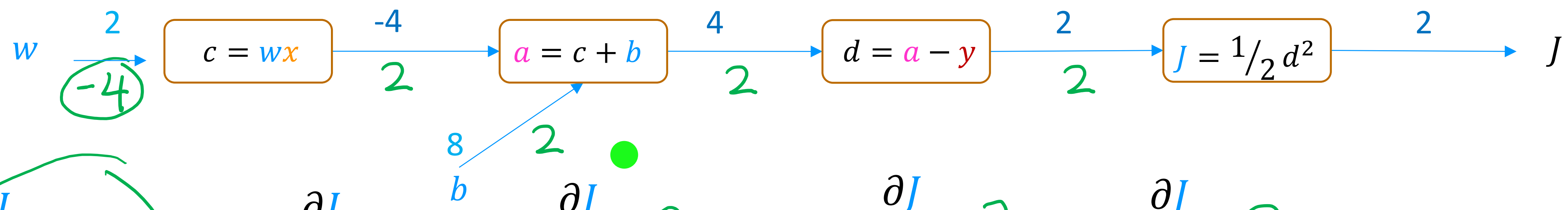
$$\frac{\partial J}{\partial a} = \underset{1}{\frac{\partial d}{\partial a}} \times \underset{2}{\frac{\partial J}{\partial d}}$$

$d \uparrow \underbrace{0.001}_{\varepsilon} \quad \overset{?}{J} \uparrow \underbrace{0.002}_{2\varepsilon}$

chain rule
(calculus)

Computing the Derivatives

$w = 2$ $b = 8$ $x = -2$ $y = 2$ $a = wx + b$ $J = \frac{1}{2}(a-y)^2$



$\frac{\partial J}{\partial w} = -4$ $\frac{\partial J}{\partial c} = 2$ $\frac{\partial J}{\partial b} = 2$ $\frac{\partial J}{\partial a} = 2$ $\frac{\partial J}{\partial d} = 2$

$$J = \frac{1}{2}\left((wx+b)-y\right)^2 = \frac{1}{2}\left((2 \times -2 + 8) - 2\right)^2 = 2$$

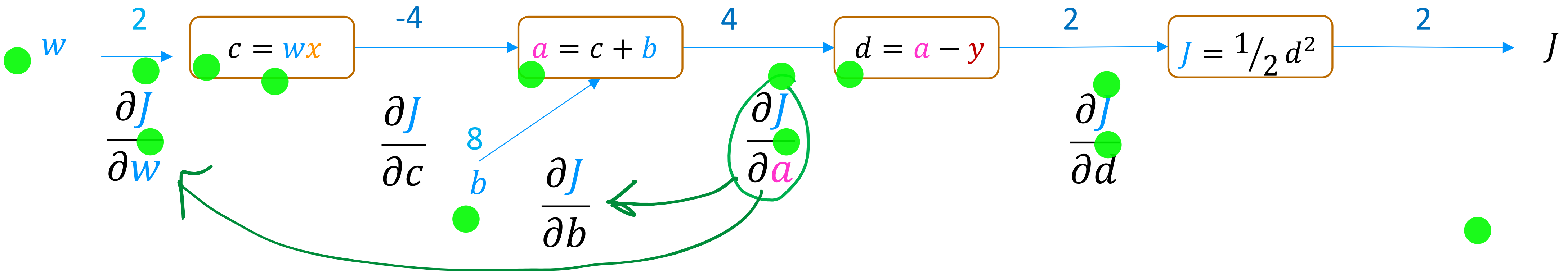
w ↑ 0.001 $J = \frac{1}{2}\left((2.001 \times -2 + 8) - 2\right)^2 = 1.\underbrace{996}002$

$J \downarrow 4 \times 0.001$

$J \uparrow -4 \times 0.001$

$\frac{\partial J}{\partial w} = -4$

Backprop is an efficient way to compute derivatives



Compute $\frac{\partial J}{\partial a}$ once and use it to compute both $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$.

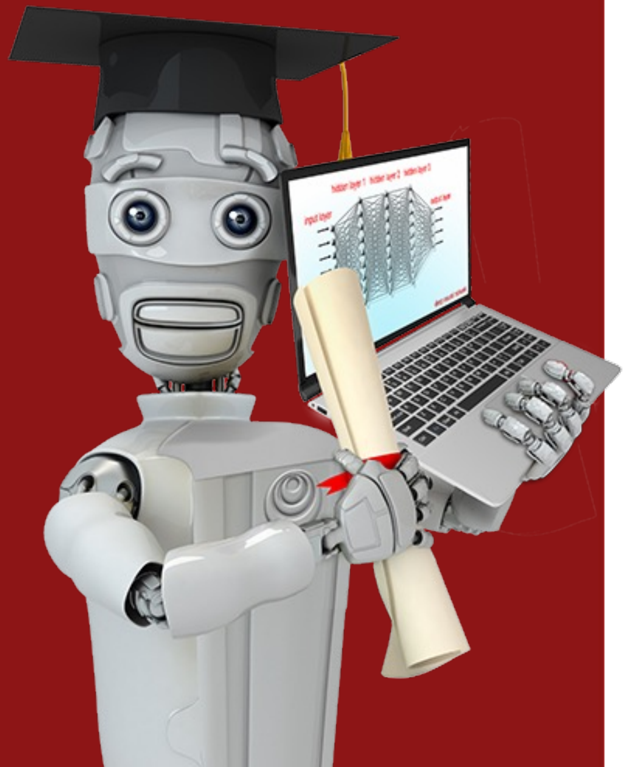
If N nodes and P parameters, compute derivatives

in roughly $N + P$ steps rather than $N \times P$ steps.

N	P	N+P	N×P
10,000	100,000	1.1×10^5	10^9

DeepLearning.AI

Stanford
ONLINE



Backprop Intuition (Optional)

Larger Neural Network Example

Neural Network Example

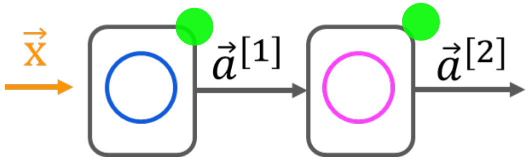
$x = 1 \quad y = 5$

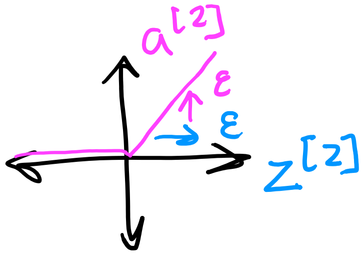
$w^{[1]} = 2, b^{[1]} = 0$

$w^{[2]} = 3, b^{[2]} = 1$

ReLU activation

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

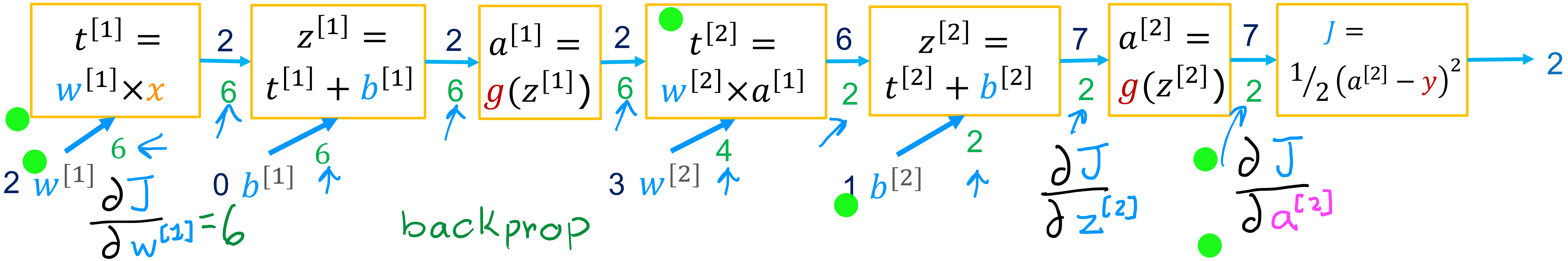




$$a^{[1]} = g(w^{[1]}x + b^{[1]}) = \overbrace{w^{[1]}x + b^{[1]}}^{z^{[1]}} = 2\times1 + 0 = 2$$

$$a^{[2]} = g(w^{[2]}a^{[1]} + b^{[2]}) = \overbrace{w^{[2]}a^{[1]} + b^{[2]}}^{z^{[2]}} = 3\times2 + 1 = 7$$

$$J(w,b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(7-5)^2 = 2$$



backprop

if $w^{[1]} \uparrow \varepsilon$, then $J \uparrow 6 \times \varepsilon$ let's verify this!

Neural Network Example

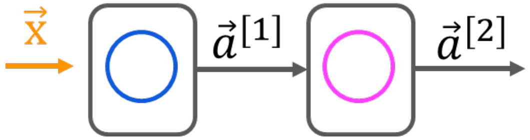
$x = 1$ $y = 5$

$w^{[1]} = 2, b^{[1]} = 0$

$w^{[2]} = 3, b^{[2]} = 1$

ReLU activation

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



$$a^{[1]} = g(w^{[1]}x + b^{[1]}) = w^{[1]}x + b^{[1]} = \overset{2.001}{\cancel{2}}\times 1 + 0 = \overset{2.001}{\cancel{2}}$$

$$a^{[2]} = g(w^{[2]}a^{[1]} + b^{[2]}) = w^{[2]}a^{[1]} + b^{[2]} = 3\times \overset{2.001}{\cancel{2}} + 1 = \cancel{7}\ 7.003$$

$$J(w,b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(\underbrace{\overset{7.003}{\cancel{7}} - 5}_{\frac{(2.003)^2}{2}})^2 = \cancel{2}\ \ 2\boxed{.006}005$$

$w^{[1]} \uparrow 0.001$

$J \uparrow 6 \times 0.001$

$$\frac{\partial J}{\partial w^{[1]}} = 6$$

$$\frac{\partial J}{\partial w^{[1]}} \quad \frac{\partial J}{\partial b^{[1]}}$$

$$\frac{\partial J}{\partial w^{[2]}} \quad \frac{\partial J}{\partial b^{[2]}}$$

N nodes □→□→□

P parameters

$w_1, b_1, w_2, b_2 \dots$

inefficient way

$N \times P$

efficient way (backprop)

$N + P$