Machine learning

by - Andrew NG

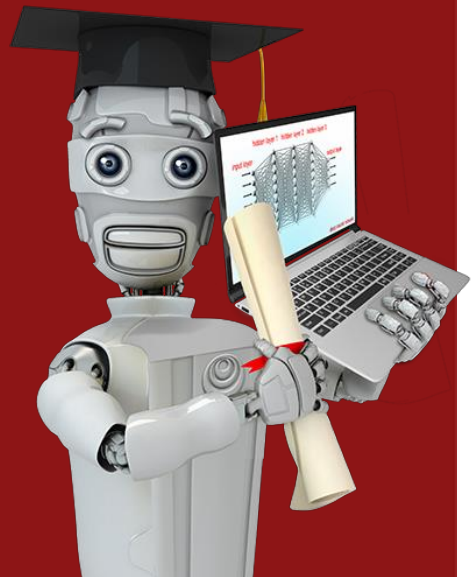
Sat 16 Jul

Week 2

Copyright Notice

DeepLearning.AI

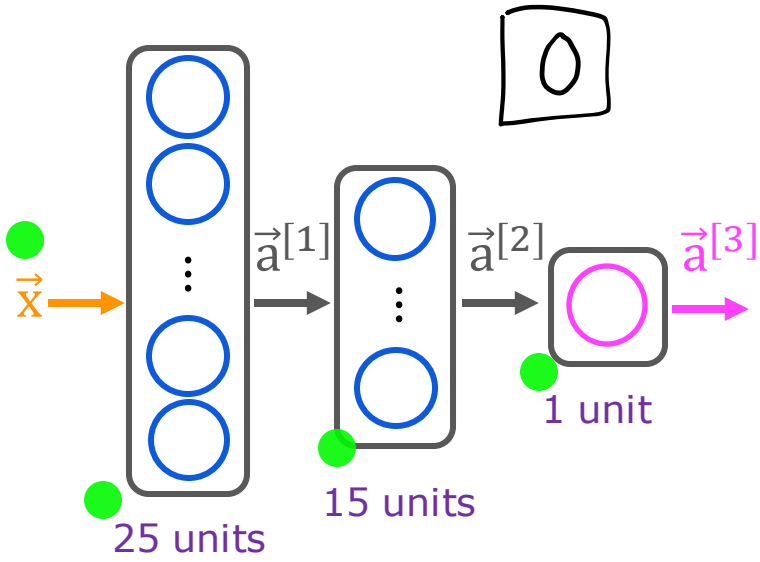
Stanford
ONLINE



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

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

0 1 2 3

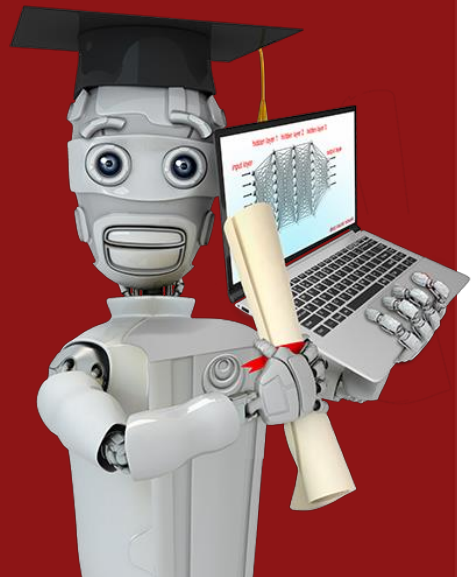
epochs: number of steps in gradient descent

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)

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

logistic regression

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

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

neural network

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

2. specify loss and cost

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

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

logistic loss

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

binary cross entropy

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

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

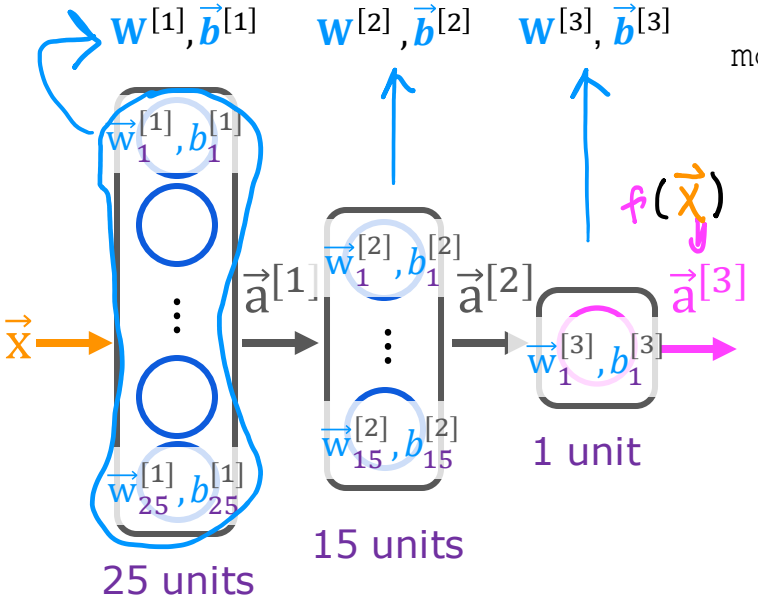
```
w = w - alpha * dj_dw
b = b - alpha * dj_db
```

```
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

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

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

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

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

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

Keras

regression
(predicting numbers
and not categories)

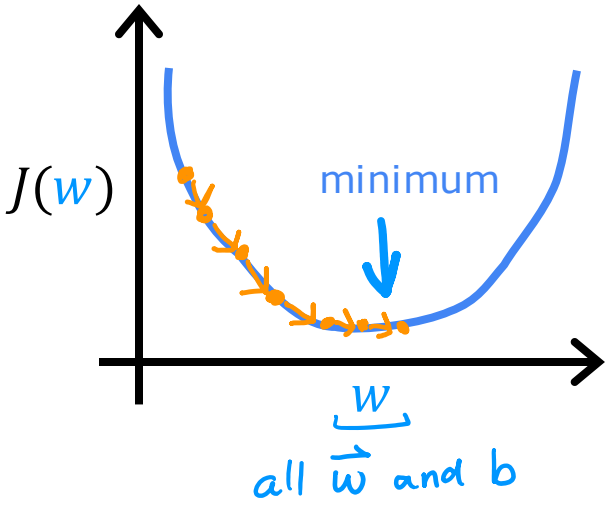
mean squared error

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

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

for regression

3. Gradient descent



repeat {

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

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

}

Compute derivatives for gradient descent using "back propagation"

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

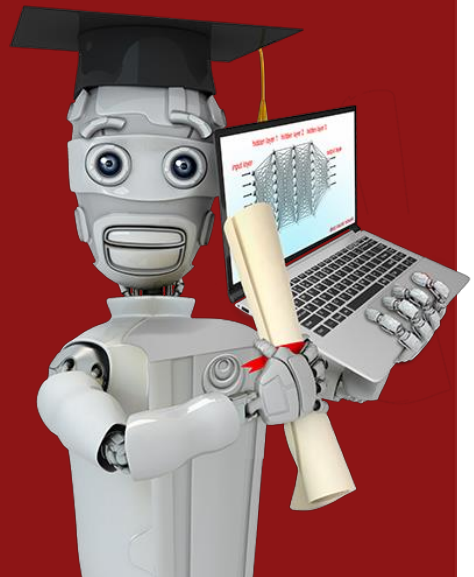
Neural network libraries

Use code libraries instead of coding ”from scratch”

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

DeepLearning.AI

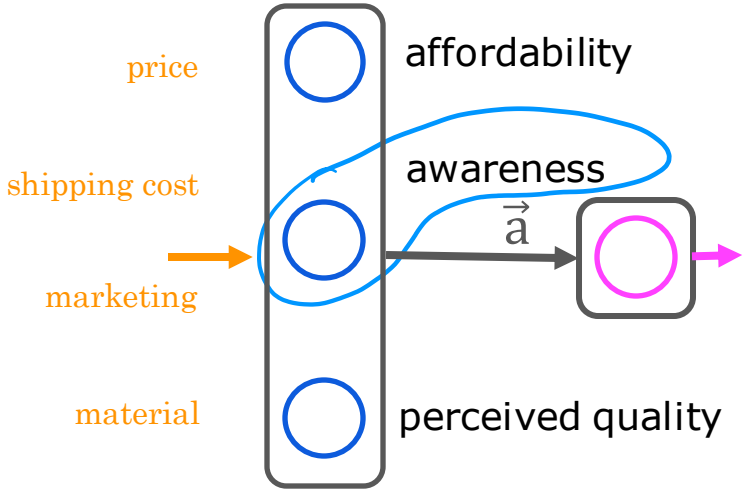
Stanford ONLINE



Activation Functions

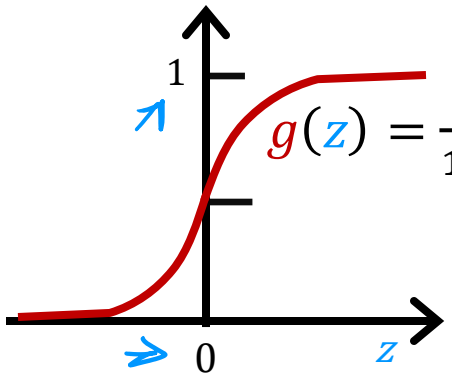
Alternatives to the sigmoid activation

Demand Prediction Example



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

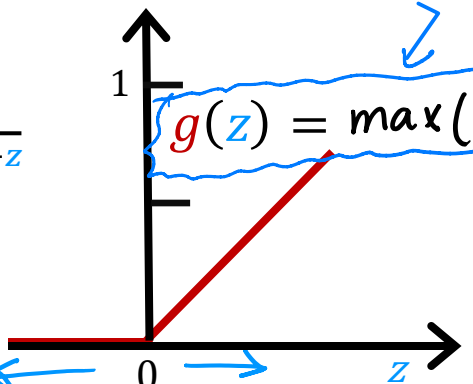
Sigmoid



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

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

ReLU Rectified Linear Unit



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

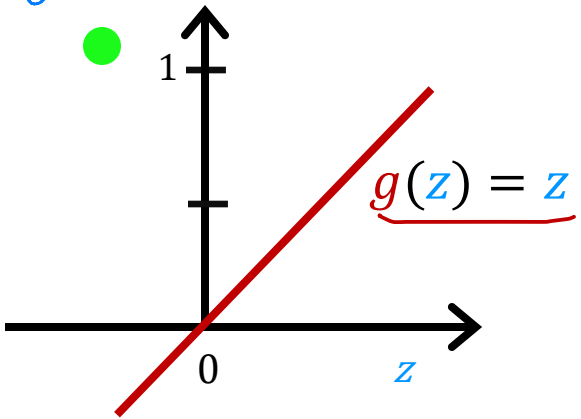
if $z < 0$, $g(z)$ is 0

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

Examples of Activation Functions

"No activation function"

1. Linear activation function

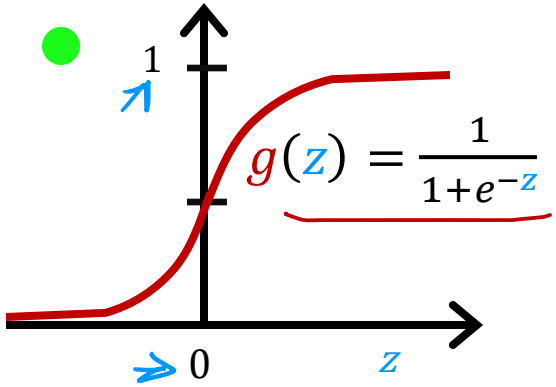


$a = g(z) = \vec{w} \cdot \vec{x} + b$

z

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

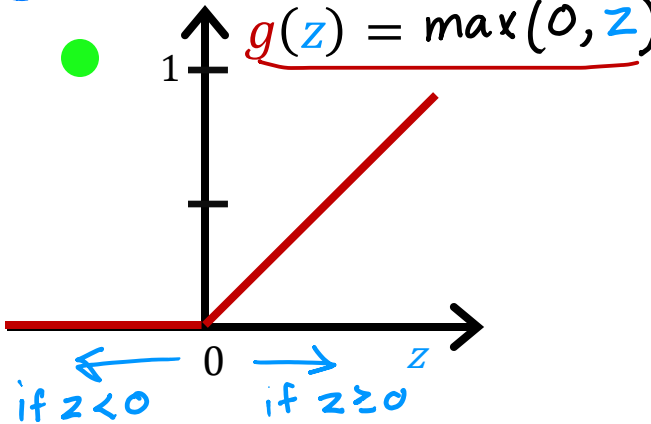
2. Sigmoid



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

Later: softmax activation

3. ReLU Rectified Linear Unit

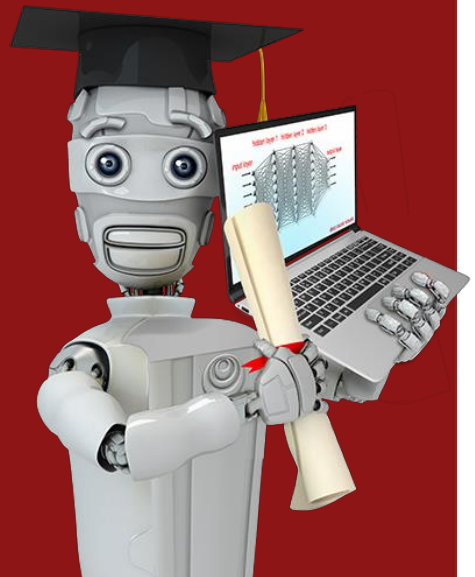


if $z < 0$: $g(z)$ is 0

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

DeepLearning.AI

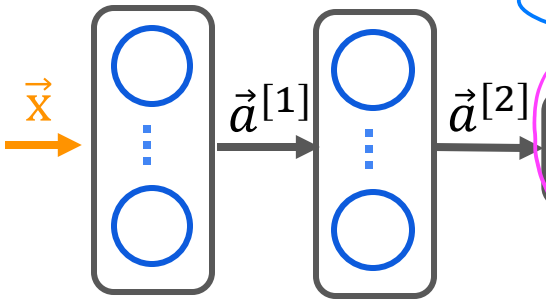
Stanford ONLINE



Activation Functions

Choosing activation functions

Output Layer



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

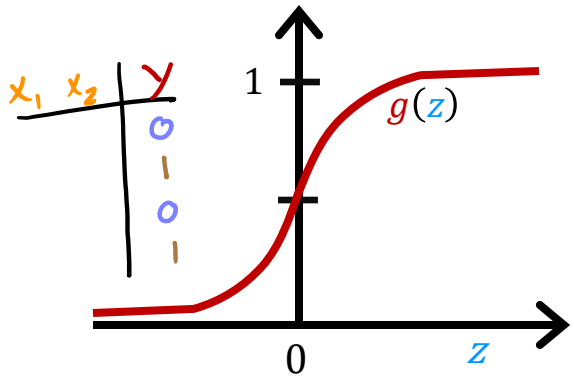
$f(\vec{x}) = a_1^{[3]} = g(z_1^{[3]})$

Choosing $g(z)$ for output layer?

Binary classification

Sigmoid

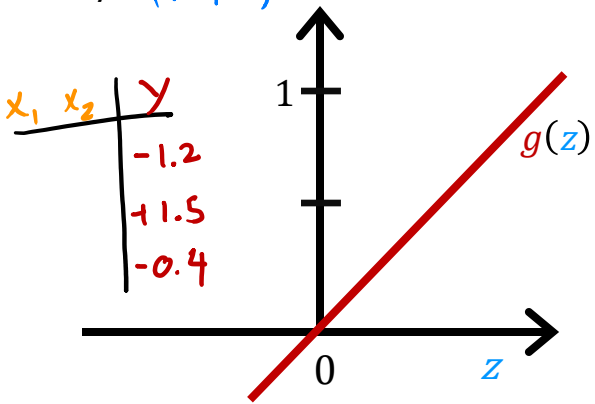
y=0/1



Regression

Linear activation function

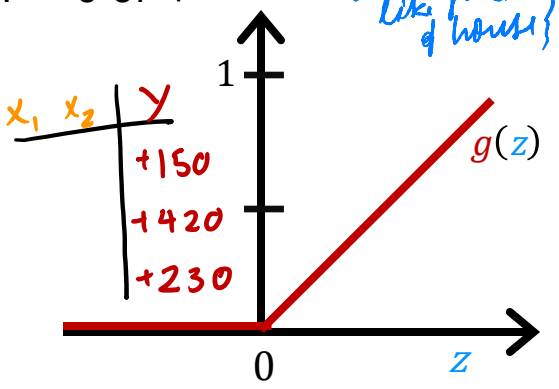
y = +/- {Profit}



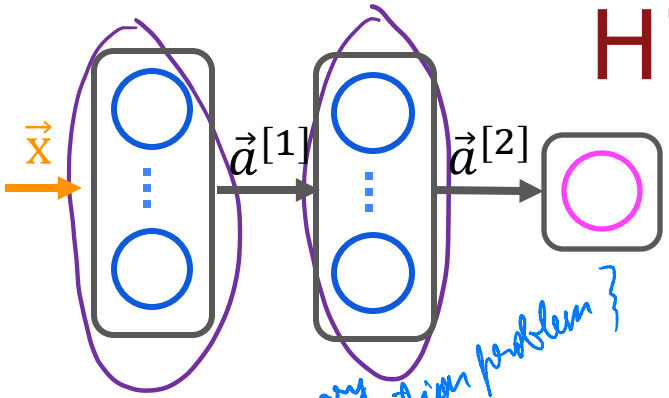
Regression

ReLU

Y = 0 or + {non-negative Like price of house}



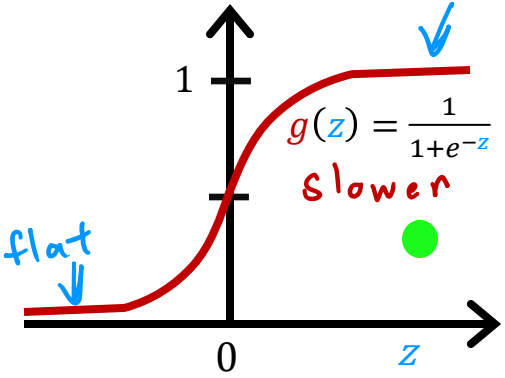
Hidden Layer

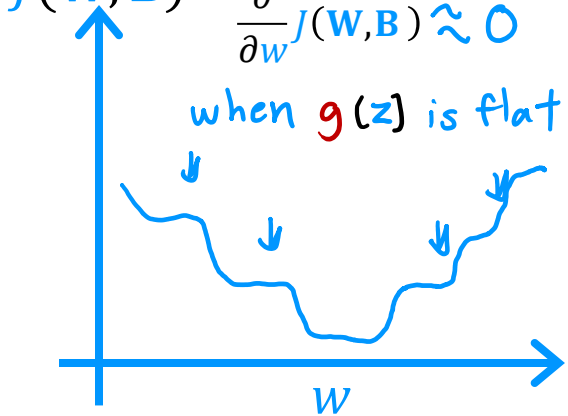


$\vec{a}^{[3]} = f(\vec{x})$

Choosing $g(z)$ for hidden layer

Sigmoid { binary classification problem }

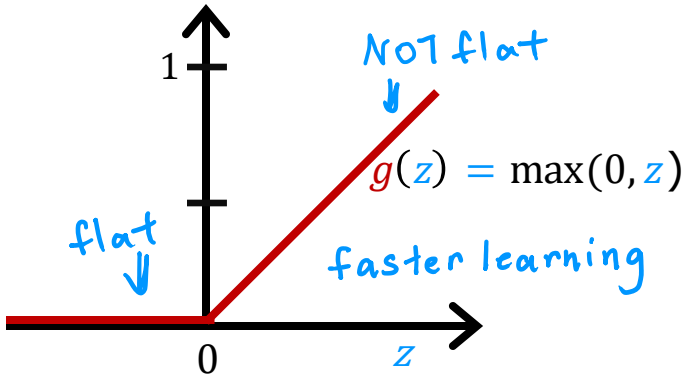




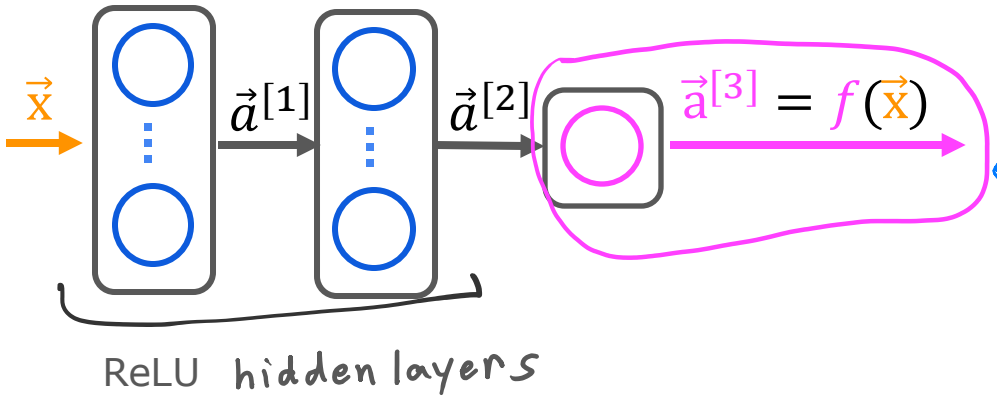
most common choice

ReLU

faster



Choosing Activation Summary



binary classification
activation='sigmoid'

regression y negative/positive
activation='linear'

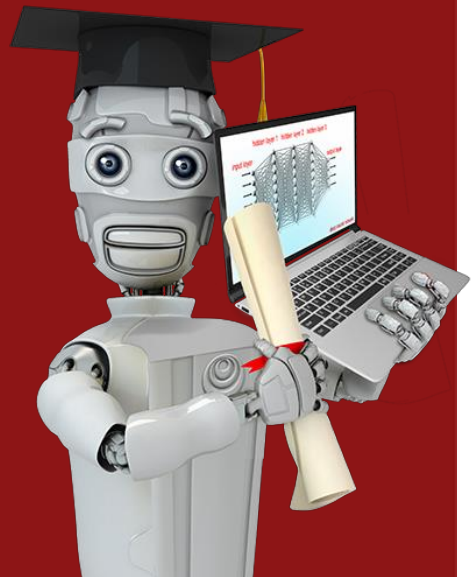
regression $y \geq 0$
activation='relu'

```
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'

DeepLearning.AI

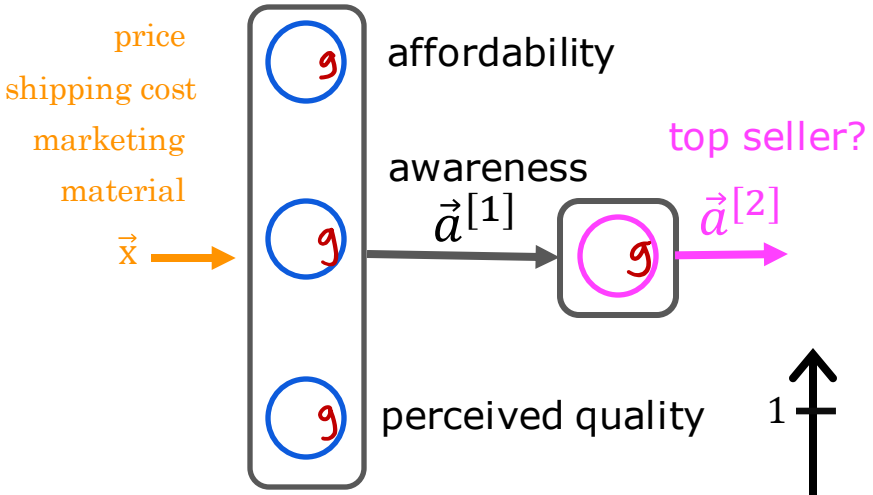
Stanford
ONLINE

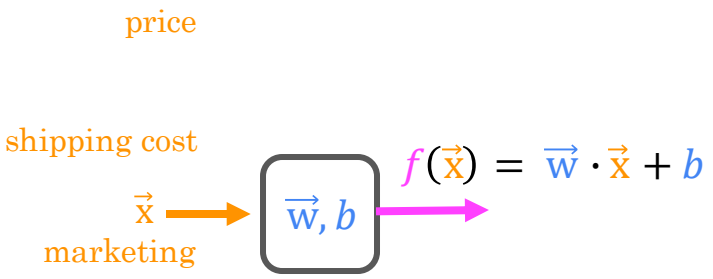


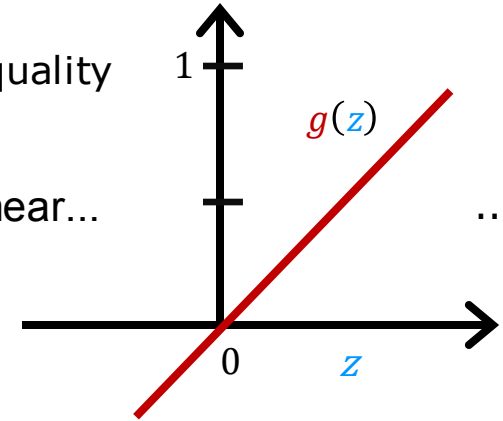
Activation Functions

Why do we need activation functions?

Why do we need activation functions?



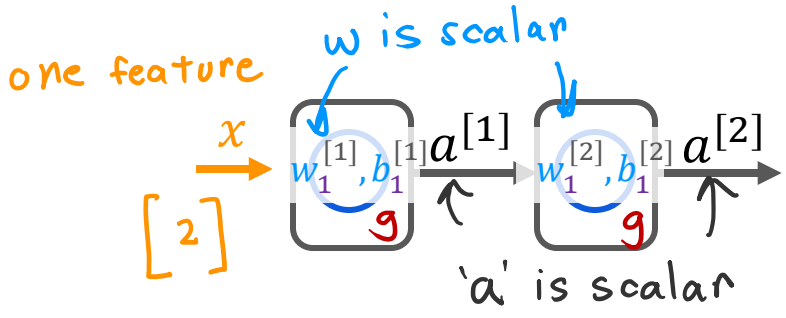




if all $g(z)$ are linear...

...no different than linear regression

Linear Example



$$g(z) = z$$

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

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

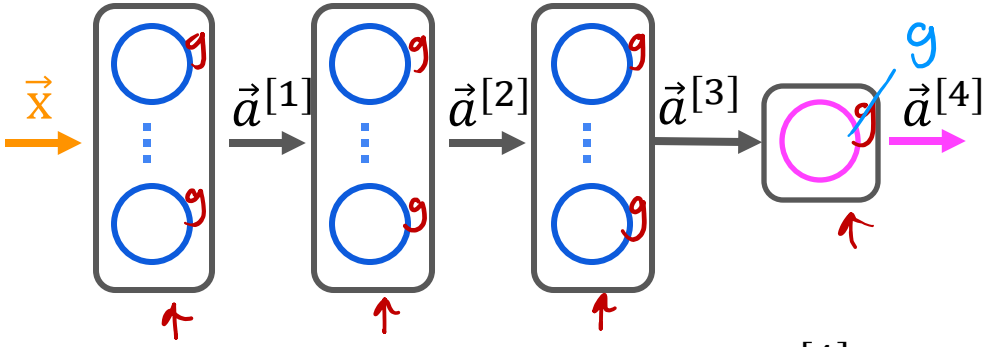
$$= w_1^{[2]} \left(w_1^{[1]} x + b_1^{[1]} \right) + b_1^{[2]}$$

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

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

$f(x) = wx + b$ 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

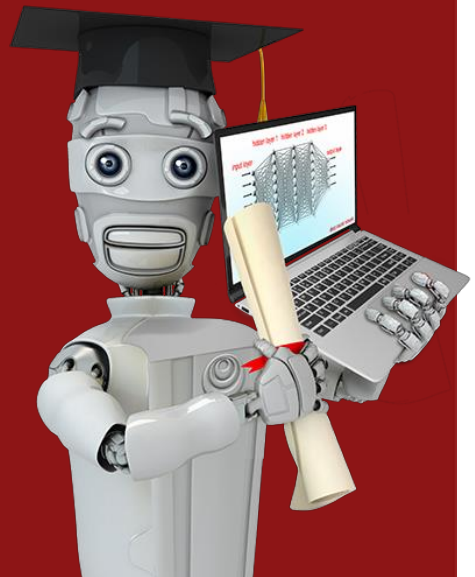
$$\vec{a}^{[4]} = \frac{1}{1+e^{-(\vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]})}}$$

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

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

DeepLearning.AI

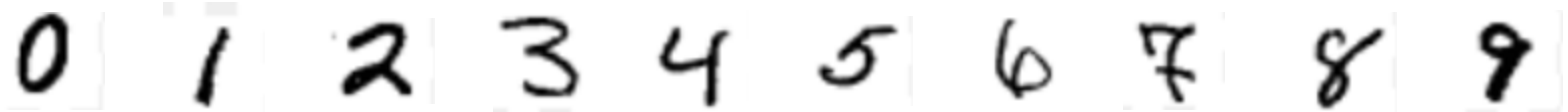
Stanford
ONLINE



Multiclass Classification

Multiclass

MNIST example

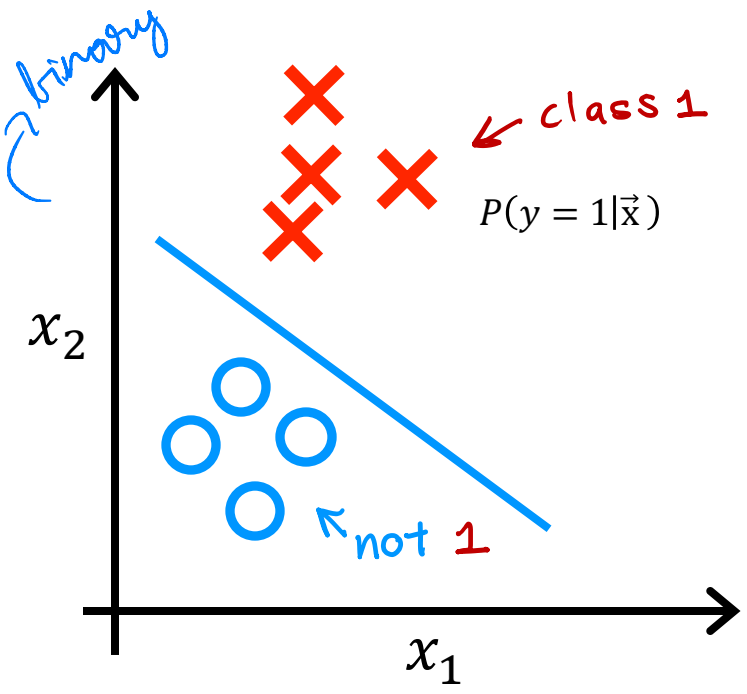


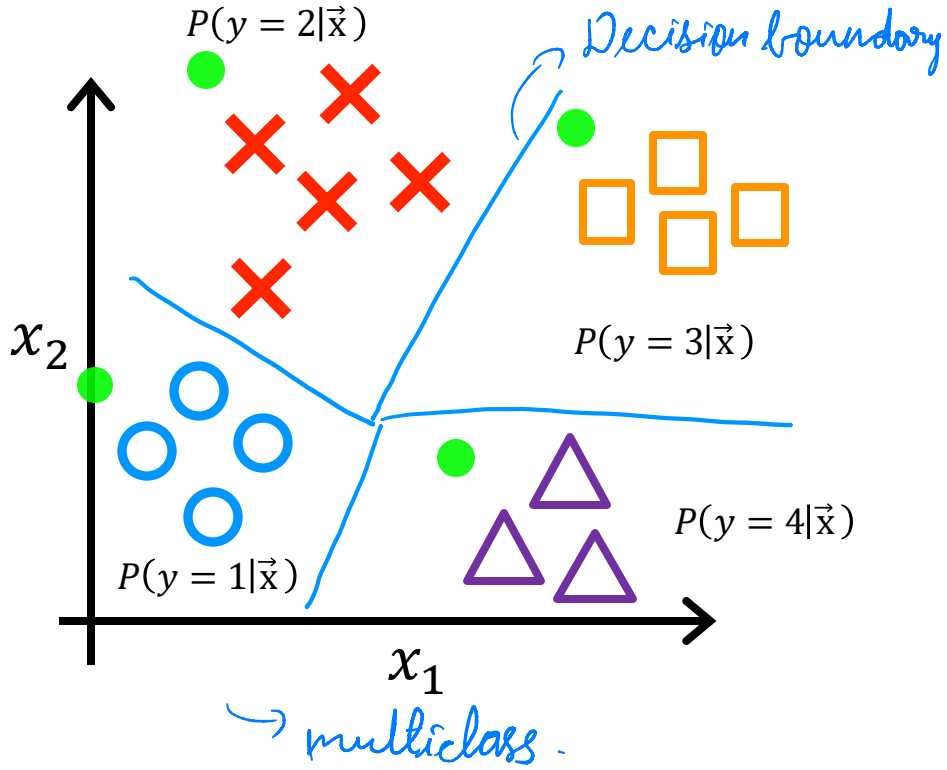
$y = 0$ 1 2 3 4 5 6 7 8 9

$y = 7$

multiclass classification problem:
target y can take on more than two possible values

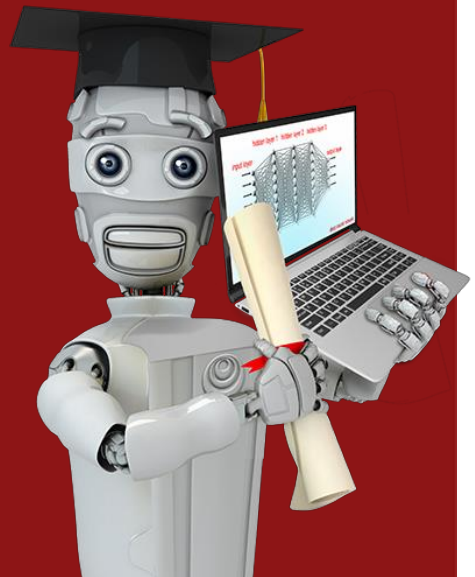
Multiclass classification example





DeepLearning.AI

Stanford
ONLINE



Multiclass Classification

Softmax

regression algo

logistic regression to multiclass

Logistic regression (2 possible output values)

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

$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$ 0.71

$a_2 = 1 - a_1 = P(y=0|\vec{x})$ 0.29

Softmax regression (N possible outputs)

$y = 1, 2, 3, \ldots, N$

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

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

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

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

Softmax regression (4 possible outputs)

$y = 1, 2, 3, 4$

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

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=1|\vec{x})$$ 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

If $N = 2$ then it becomes logistic regression

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

$$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=3|\vec{x})$$ 0.15

$a_1 = \frac{e^{(z_1)}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

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

$$a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=4|\vec{x})$$ 0.35

$a_1 + a_2 + a_3 + a_4 = 1$

Cost

Logistic regression

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

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

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

$$loss = -y \log a_1 - (1 - y) \log(1 - a_1)$$

if $y = 1$ if $y = 0$

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

Softmax regression

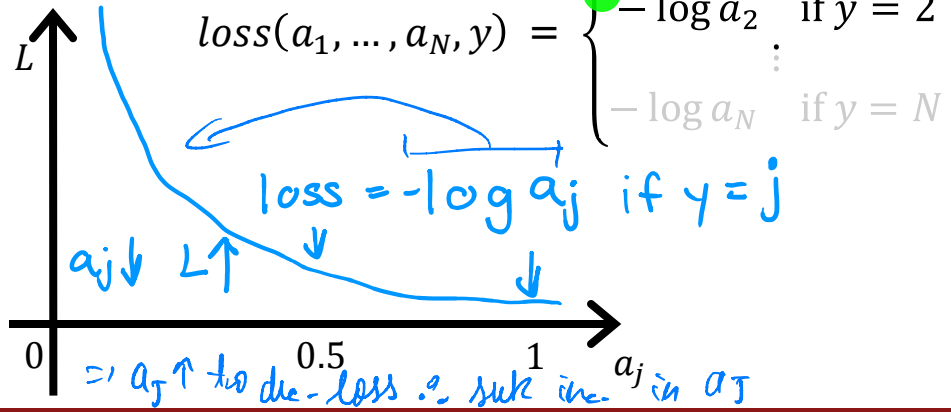
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = 1|\vec{x})$$

$$\vdots$$

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

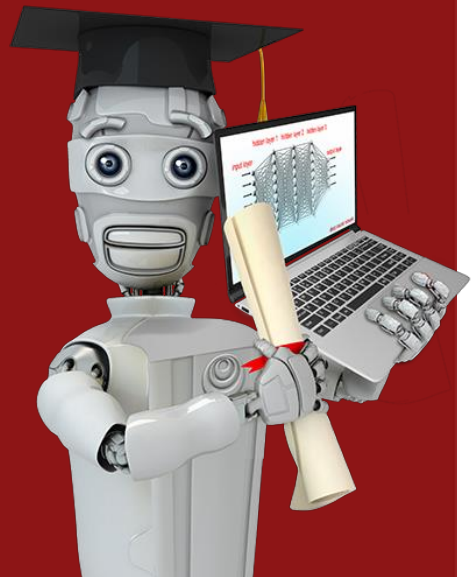
Crossentropy loss

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



DeepLearning.AI

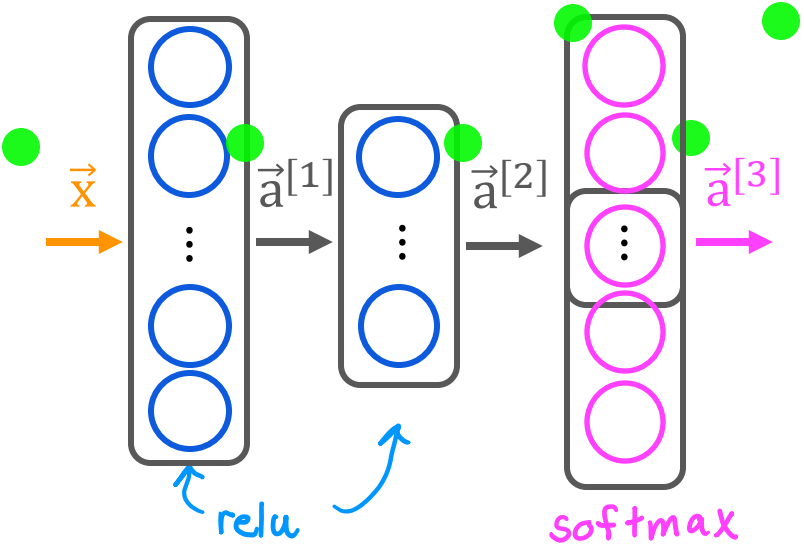
Stanford
ONLINE



Multiclass Classification

Neural Network with Softmax output

Neural Network with Softmax output



$$z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]} \qquad a_1 = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}} = P(y = 1 | \vec{x})$$

$$\vdots$$

$$z_{10}^{[3]} = \vec{w}_{10}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]} \qquad a_{10} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}} = P(y = 10 | \vec{x})$$

logistic regression

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

a_1 depends on only z_1

softmax → a_1 depends on all z simultaneously

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

MNIST with softmax

1. specify the model

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

2. specify loss and cost

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

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

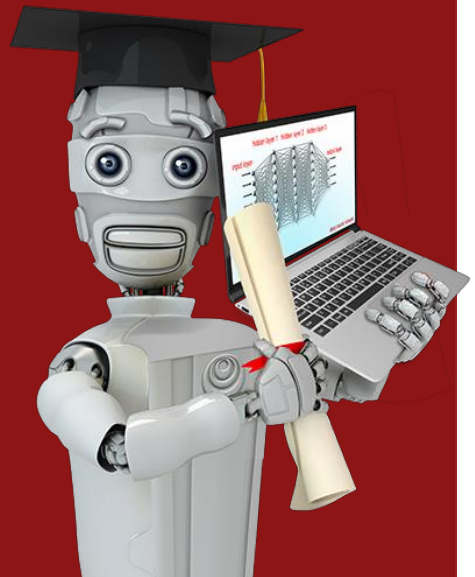
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
  Dense(units=25, activation='relu')
  Dense(units=15, activation='relu')
  Dense(units=10, activation='softmax')
                  )]
from tensorflow.keras.losses import
  SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy() )
model.fit(X,Y,epochs=100)
```

Note: better (recommended) version later.

Don't use the version shown here! Okay :)

DeepLearning.AI

Stanford
ONLINE



Multiclass Classification

Improved implementation of softmax

Numerical Roundoff Errors

option 1

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

option 2

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

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

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

More accurate loss (in code)

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

```
model = Sequential([
    Dense(units=25, activation='relu')
    Dense(units=15, activation='relu')
    Dense(units=10, activation='sigmoid')
])
model.compile(loss=BinaryCrossEntropy() )
```

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

'linear'

logit: z

More numerically accurate implementation of softmax

Softmax regression

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

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

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

'linear'

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

More Accurate

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

to tell tensor-flow to modify it.

```
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

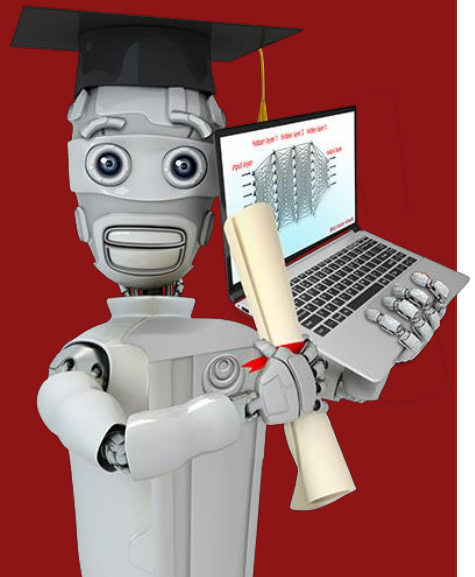
```
logit = model(X)
```

predict

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

DeepLearning.AI

Stanford
ONLINE



Multi-label Classification

Classification with multiple outputs (Optional)

Multi-label Classification

x

Is there a car?
Is there a bus?
Is there a pedestrian

yes
no
yes

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

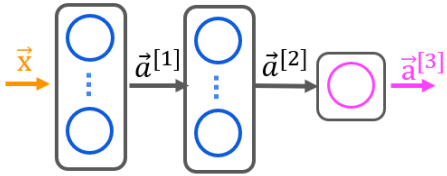
no
no
yes

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

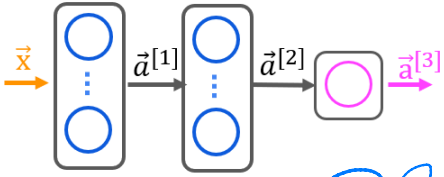
yes
yes
no

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

Multiple classes

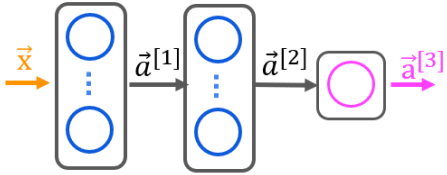


car



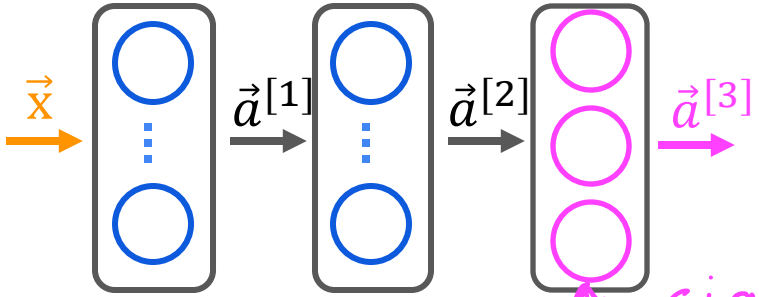
bus

OR



pedestrian

Alternatively, train one neural network with three outputs



$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix}$$

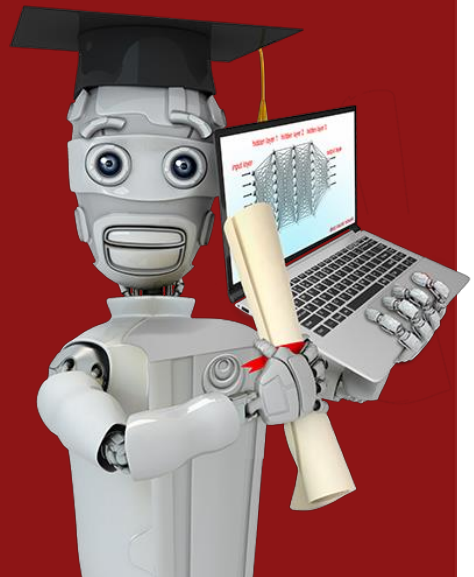
car

bus

pedestrian

DeepLearning.AI

Stanford
ONLINE



Additional Neural Network Concepts

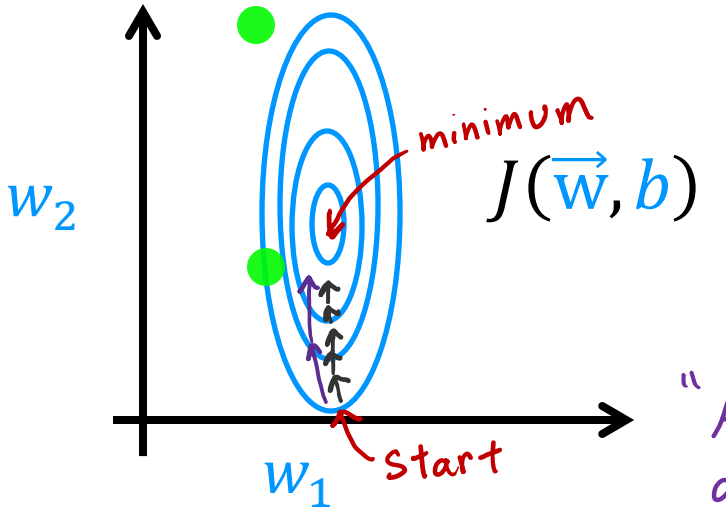
Advanced Optimization

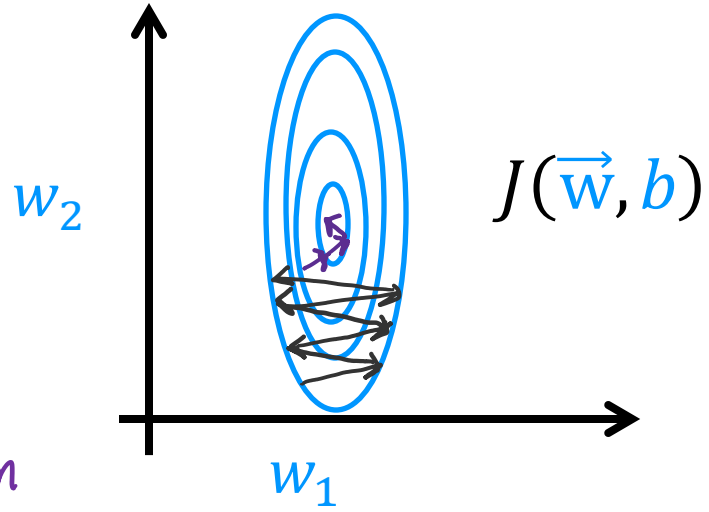
Adam algo is faster than gradient descent.

Gradient Descent

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

learning rate





"Adam" algorithm

Go faster – increase α

Go slower – decrease α

Adam Algorithm Intuition

Adam: Adaptive Moment estimation

not just one α

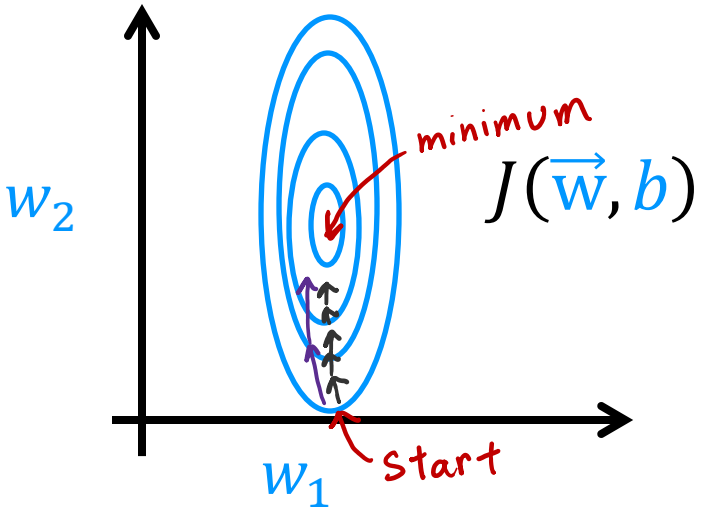
$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{w}, b)$$

$$\vdots$$

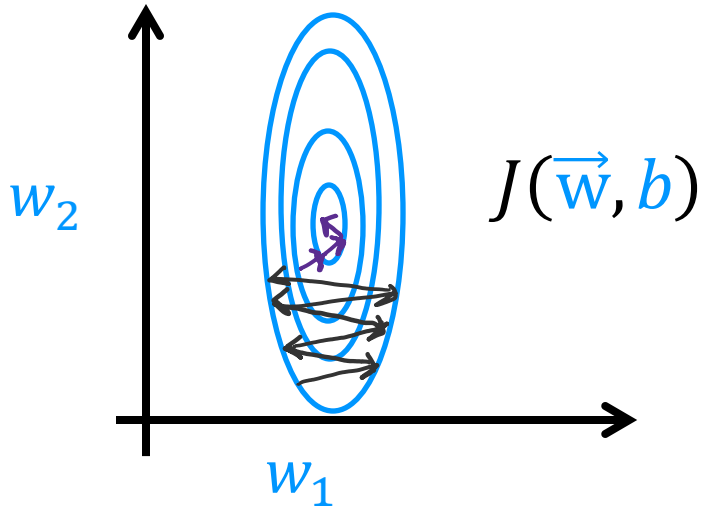
$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$

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

Adam Algorithm Intuition



If w_j (or b) keeps moving in same direction, increase α_j.



If w_j (or b) keeps oscillating, 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

needs initial α.

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

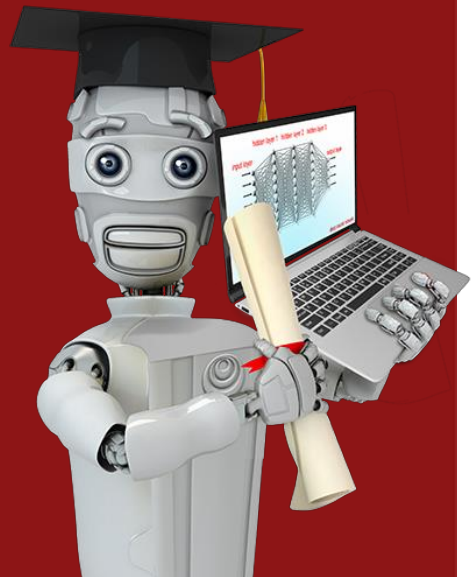
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

DeepLearning.AI

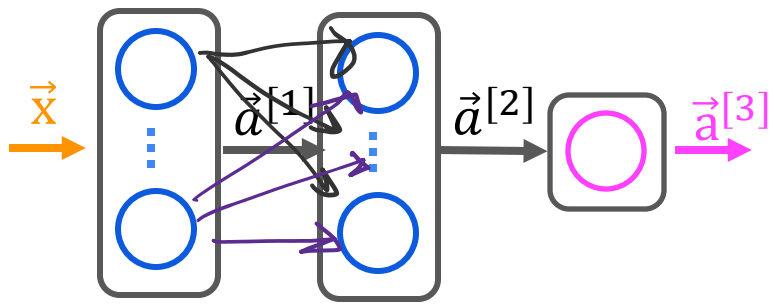
Stanford
ONLINE



Additional Neural Network Concepts

Additional Layer Types

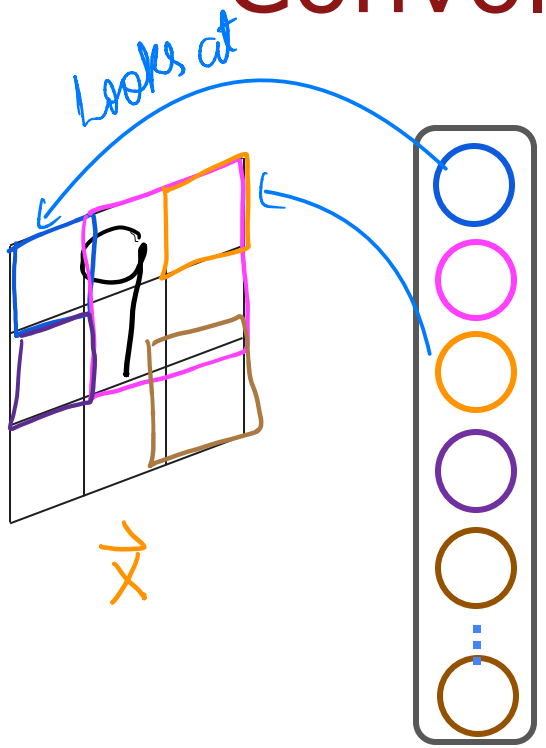
Dense Layer



Each neuron output is a function of 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

has multiple Convolutional layer