Week 2

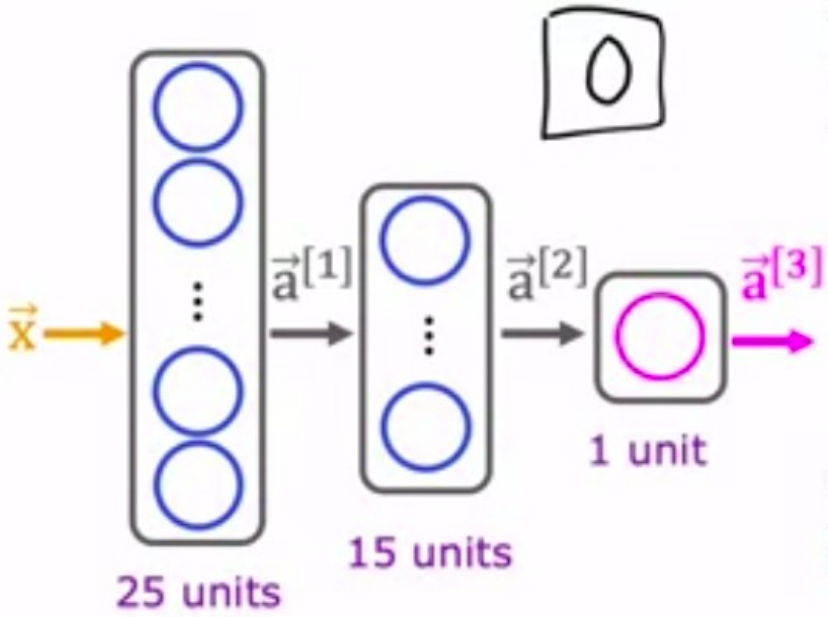
DeepLearning.AI

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)
```

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

Tensor Flow

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

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

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

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

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())
```

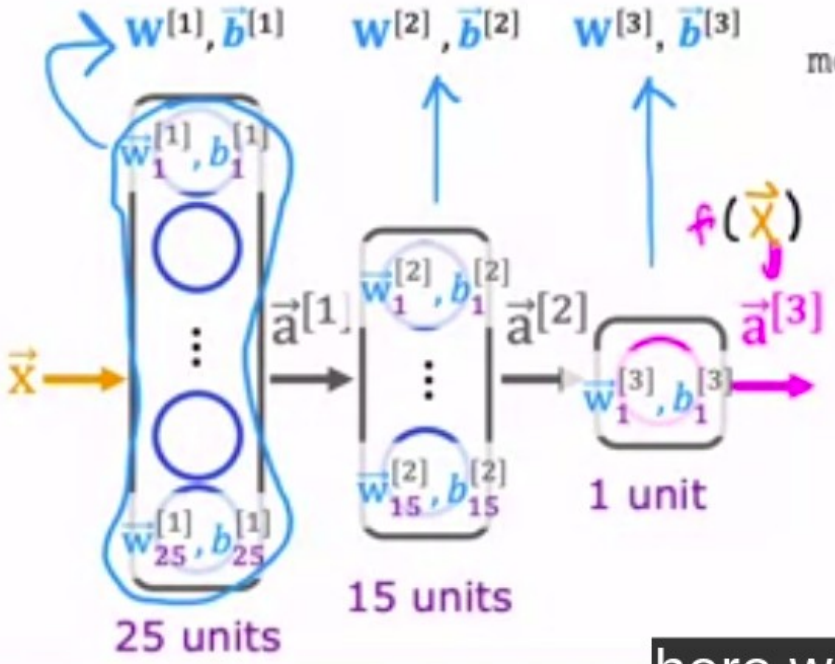
```
model.fit(X,y,epochs=100)
```

the context of training a neural network.

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'),
                                ])
```

here we have written w l and b l. Let's go on to step 2.

2. Loss and cost functions

handwritten digit classification problem

binary classification

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

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

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

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

Compare prediction vs. target

logistic loss

also known as binary cross entropy

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

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

K Keras

regression (predicting numbers and not categories)

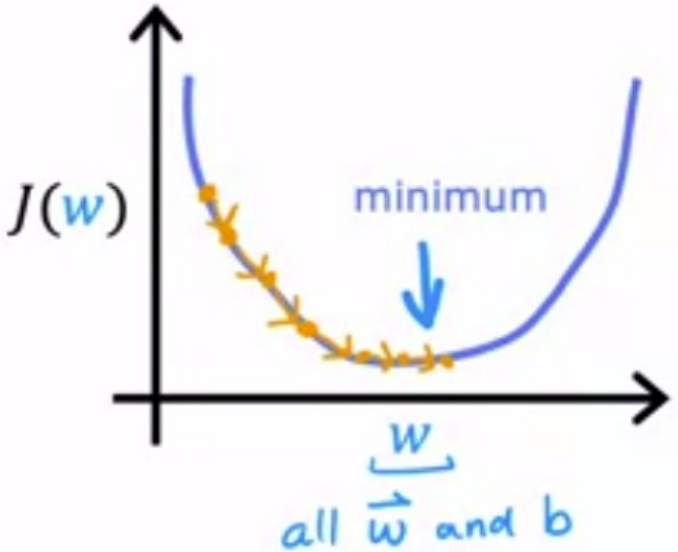
mean squared error

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

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

That's the loss function and the cost function.

3. Gradient descent



repeat {

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

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

} Compute derivatives for gradient descent using "back propagation"

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

enough that most developers will use these libraries,

Neural network libraries

Use code libraries instead of coding "from scratch"

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

which still does with
today's libraries,

Practice quiz: Neural Network Training

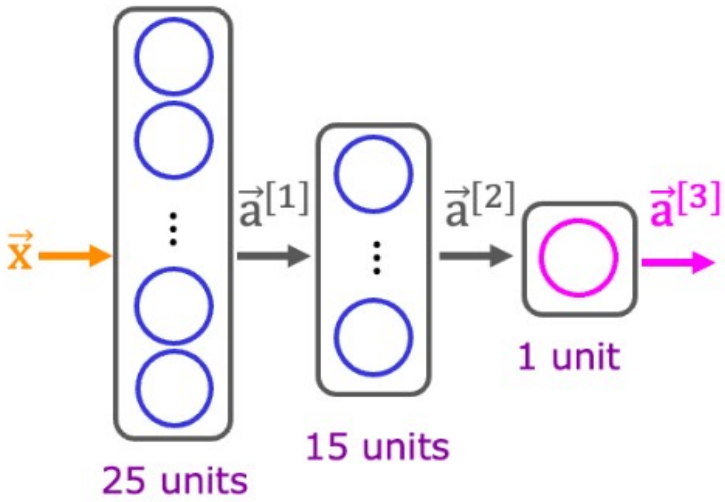
Graded Quiz • 30 min

Due Nov 27, 11:59 PM IST

1.

1 / 1 point

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

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

Here is some code that you saw in the lecture:

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

For which type of task would you use the binary cross entropy loss function?

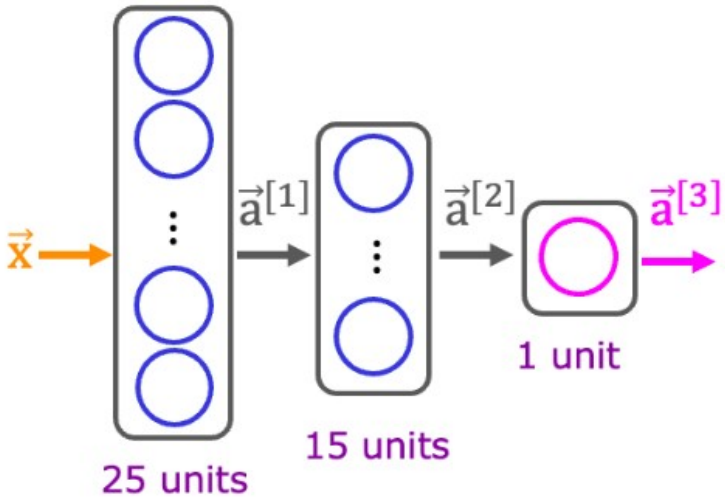
- ○ regression tasks (tasks that predict a number)
- ○ BinaryCrossentropy() should not be used for any task.
- ◉ binary classification (classification with exactly 2 classes)
- ○ A classification task that has 3 or more classes (categories)

Correct
Yes! Binary cross entropy, which we've also referred to as logistic loss, is used for classifying between two classes (two categories).

2.

1 / 1 point

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

Here is code that you saw in the lecture:

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

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

Which line of code updates the network parameters in order to reduce the cost?

- ○ model = Sequential([...])

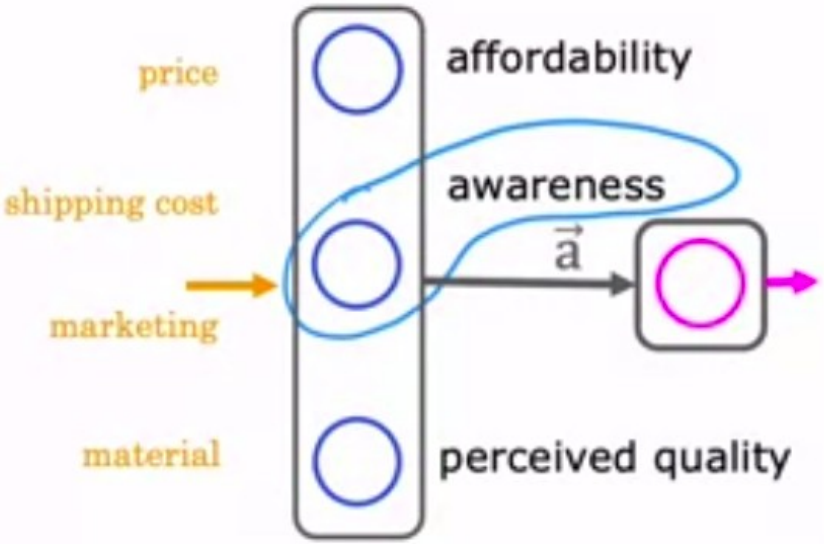
```
])


model.compile(loss=BinaryCrossentropy())

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

Which line of code updates the network parameters in order to reduce the cost?

- ○ model = Sequential([...])
- ○ None of the above -- this code does not update the network parameters.
- ◉ model.fit(X,y,epochs=100)
- ○ model.compile(loss=BinaryCrossentropy())

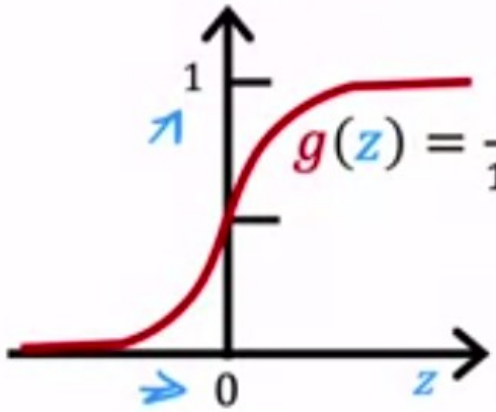
Correct
Yes! The third step of model training is to train the model on data in order to minimize the loss (and the cost)

Demand Prediction Example



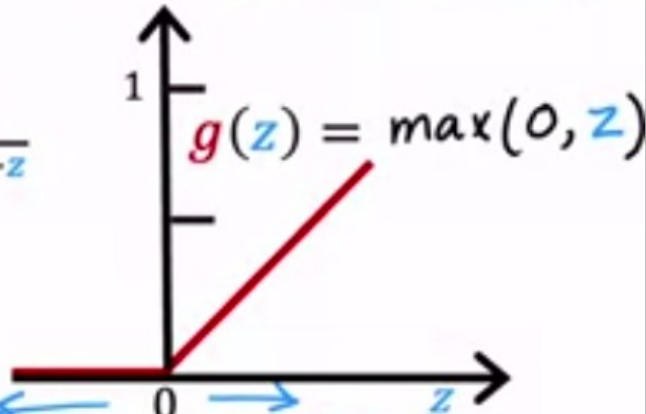
$$a_2^{[1]} = g(\overbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}^{z})$$

Sigmoid



$0 < g(z) < 1$

ReLU Rectified Linear Unit



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

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

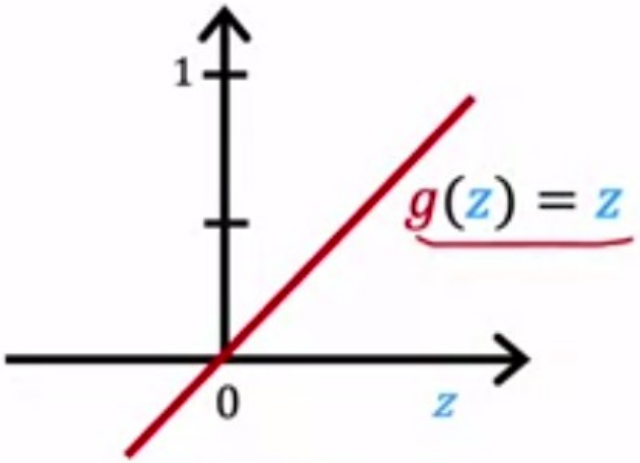
More generally you have a choice of what to use for g(z) and

Examples of Activation Functions

"No activation function"

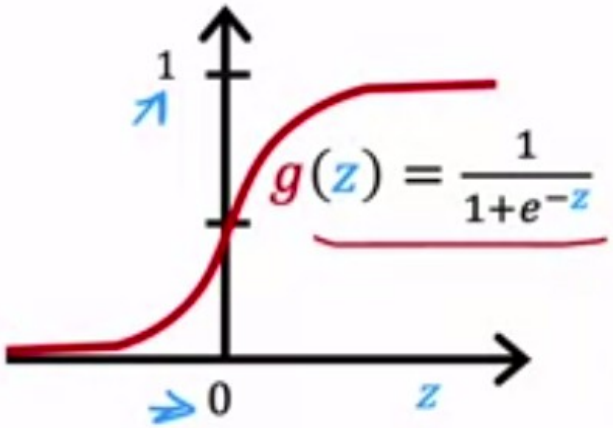
$$a_2^{[1]} = g(\underbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}_{z})$$

Linear activation function



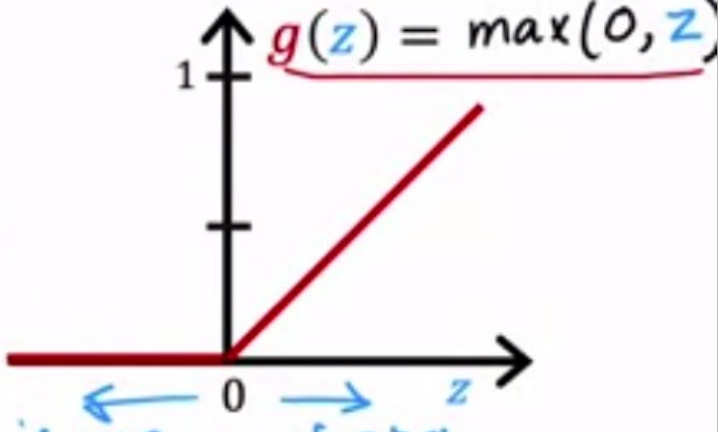
$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$

Sigmoid



$0 < g(z) < 1$

ReLU Rectified Linear Unit

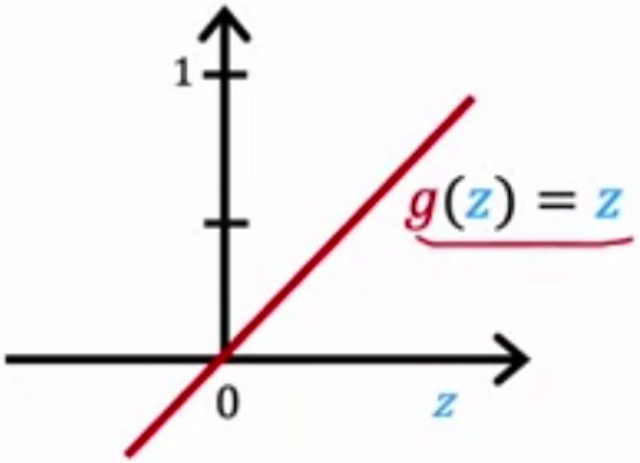


activation functions in neural networks.

Examples of Activation Functions

"No activation function"

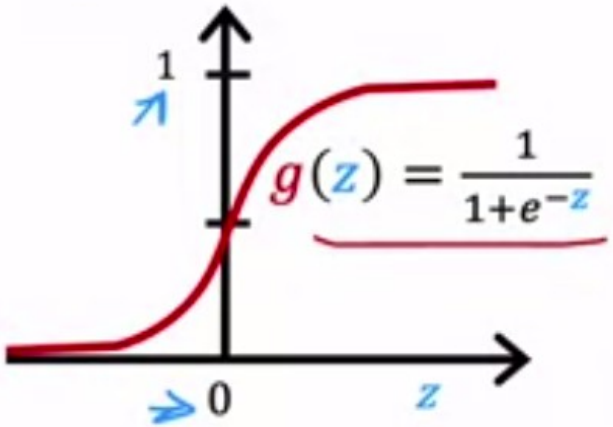
Linear activation function



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

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

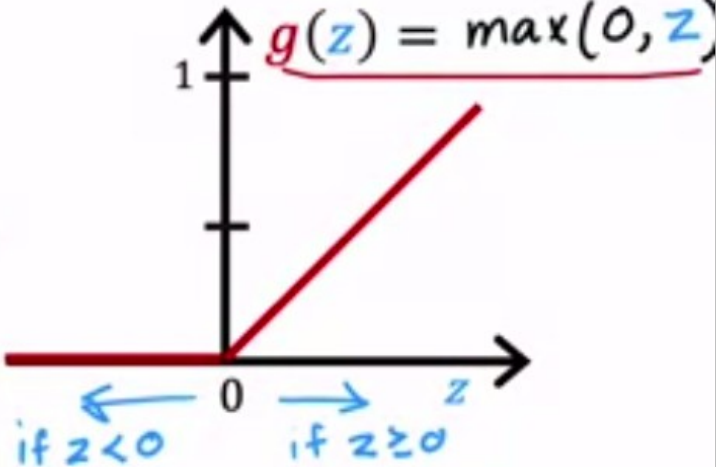
Sigmoid



$0 < g(z) < 1$

Later: softmax activation

ReLU Rectified Linear Unit



But with these activation functions you'll be able to build

DeepLearning.AI

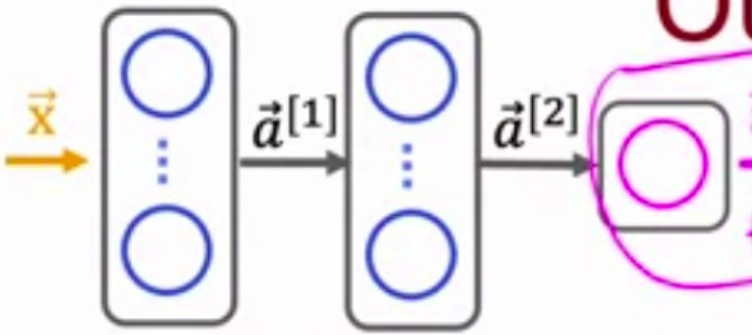
Stanford ONLINE

Activation Functions

Choosing activation functions

Let's take a look at how you can choose

Output Layer



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

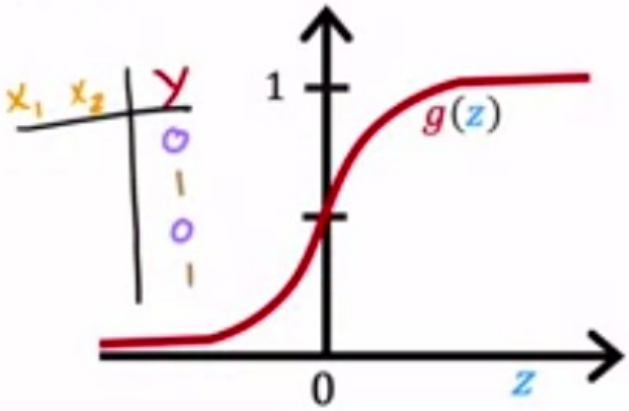
$f(\vec{x}) = a_1^{[3]} = g(z_1^{[3]})$

Choosing $g(z)$ for output layer?

Binary classification

Sigmoid

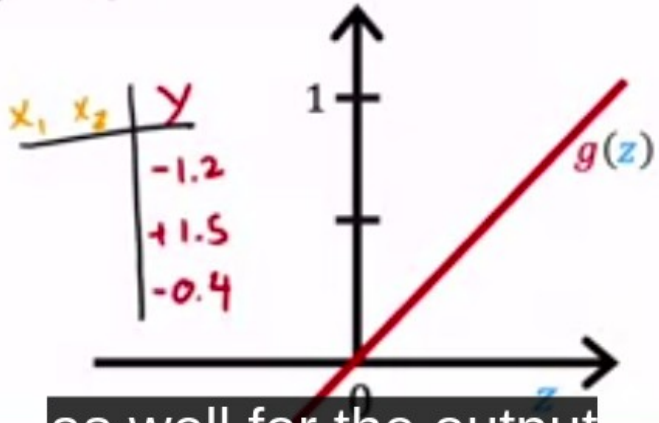
y=0/1



Regression

Linear activation function

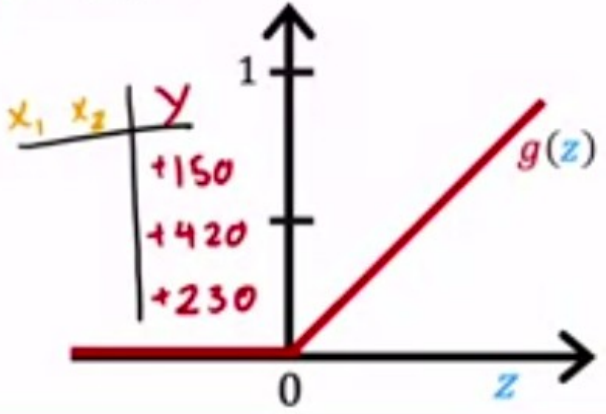
y = +/-



Regression

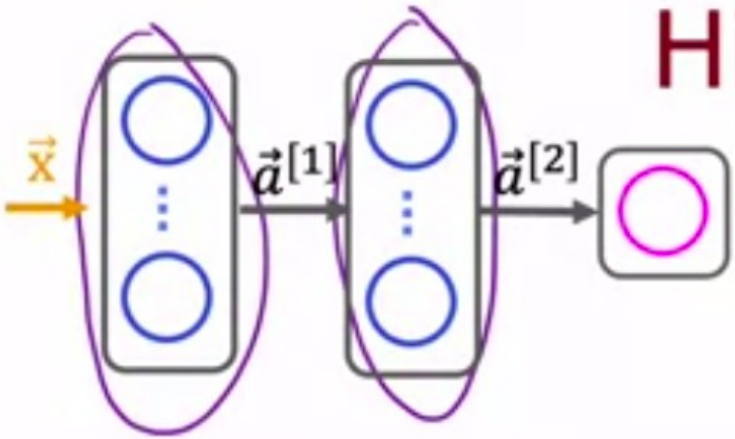
ReLU

= 0 or +



as well for the output layer of a neural network.

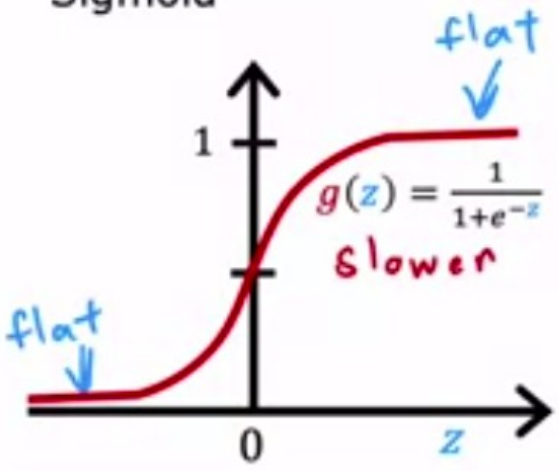
Hidden Layer

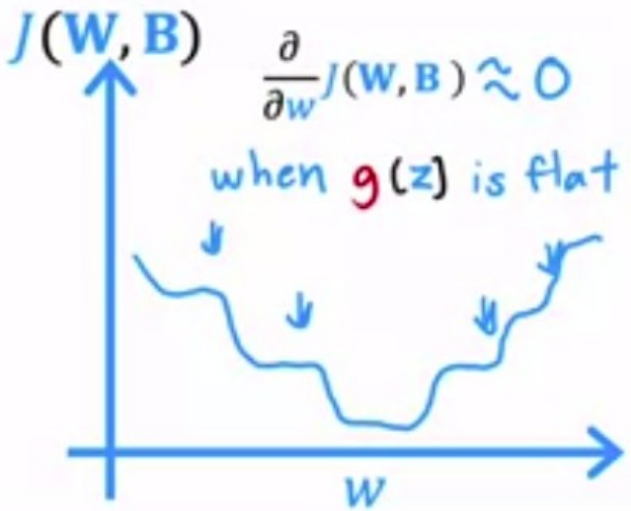


$\vec{a}^{[3]} = f(\vec{x})$

Choosing $g(z)$ for hidden layer

Sigmoid

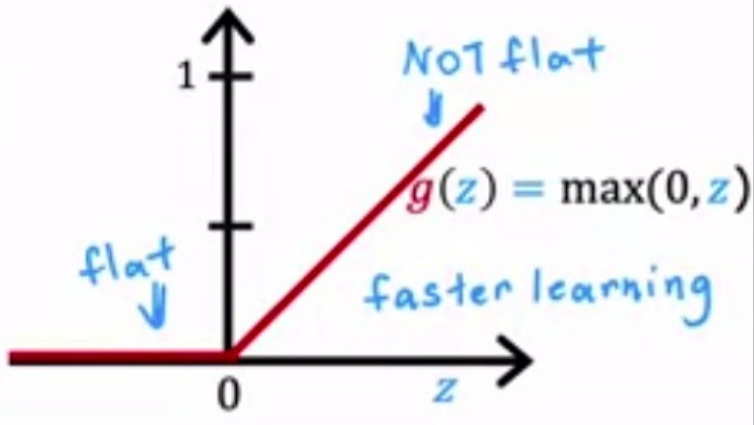




most common choice

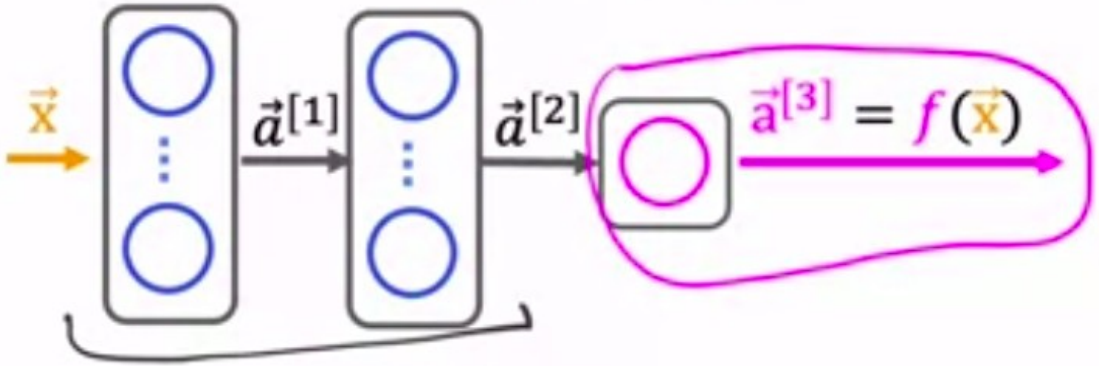
ReLU

faster



this is how I choose

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'

the ReLU activation function

DeepLearning.AI

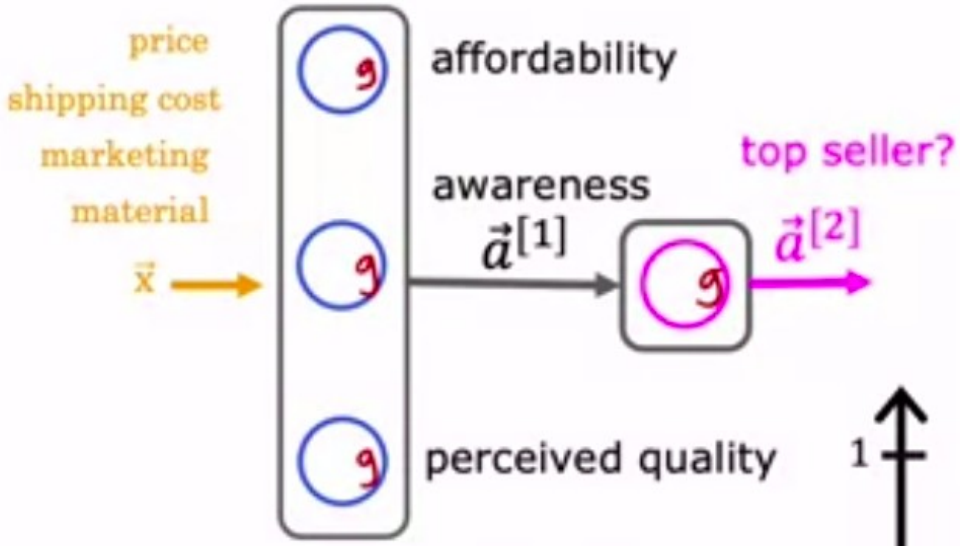
Stanford ONLINE

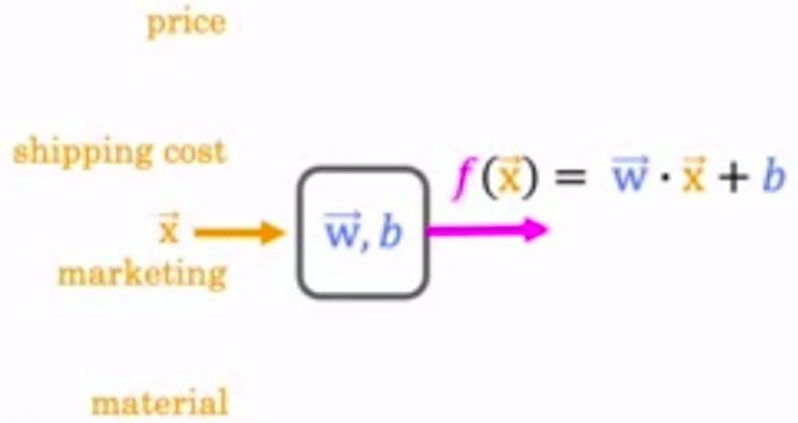
Activation Functions

Why do we need activation functions?

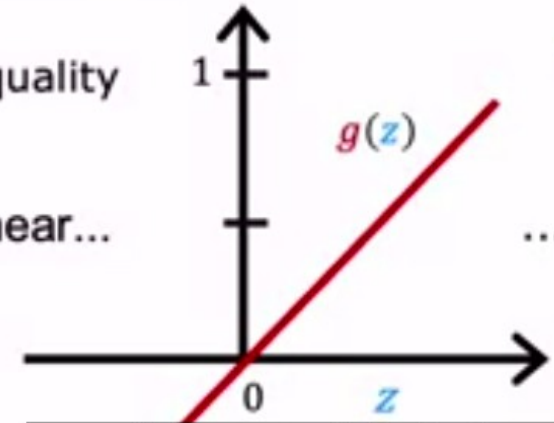
Let's take a look at why

Why do we need activation functions?





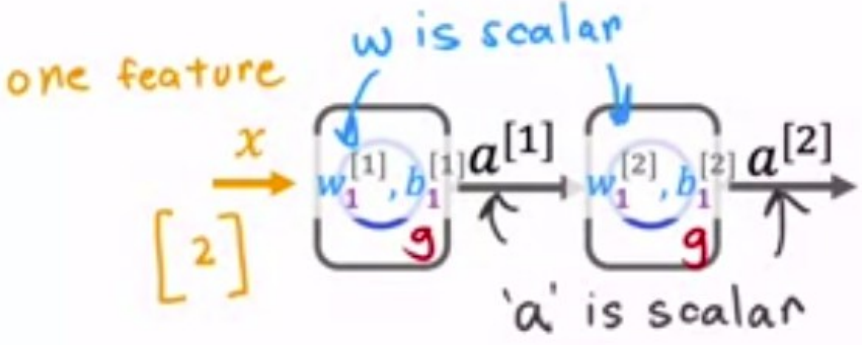
if all $g(z)$ are linear...



...no different than linear regression

So this would defeat the entire purpose of

Linear Example



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

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

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

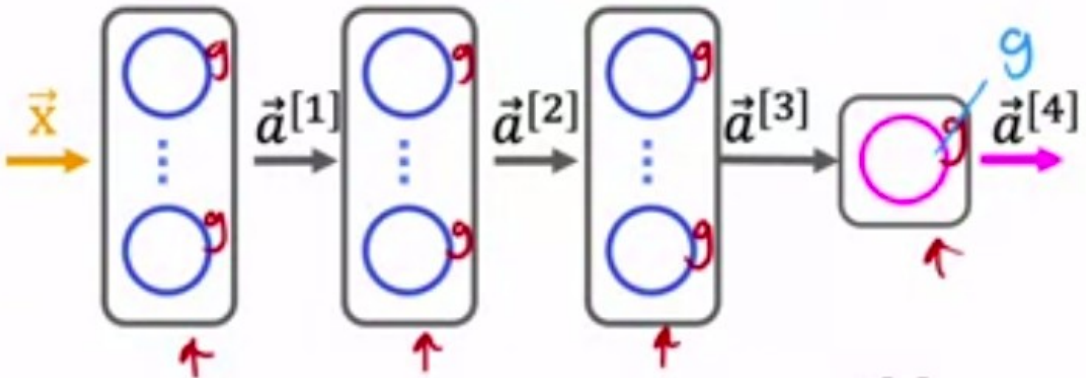
$$\vec{a}^{[2]} = \underbrace{(w_1^{[2]} w_1^{[1]})}_{w} x + \underbrace{w_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b}$$

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

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

anything more complex than
just a linear function.

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

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

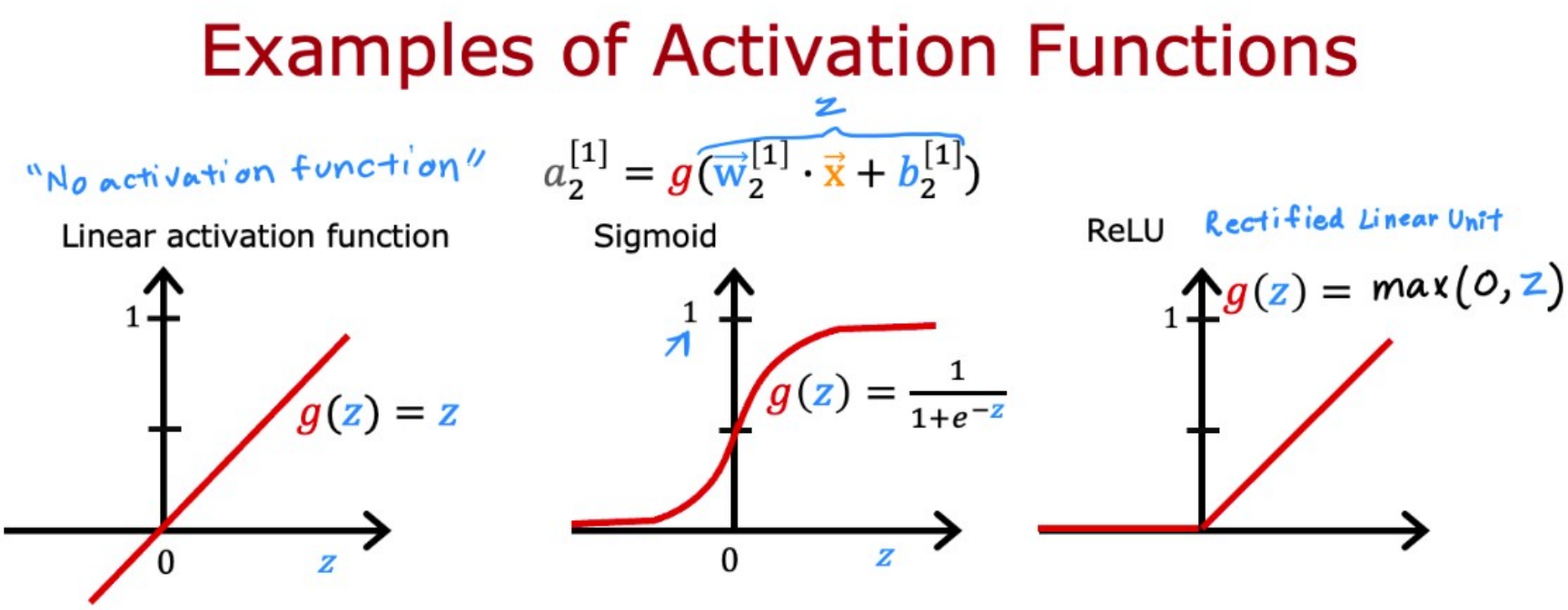
output activation is sigmoid
(hidden layers still linear)
↳ equivalent to logistic regression

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

the ReLU activation function should do just fine.

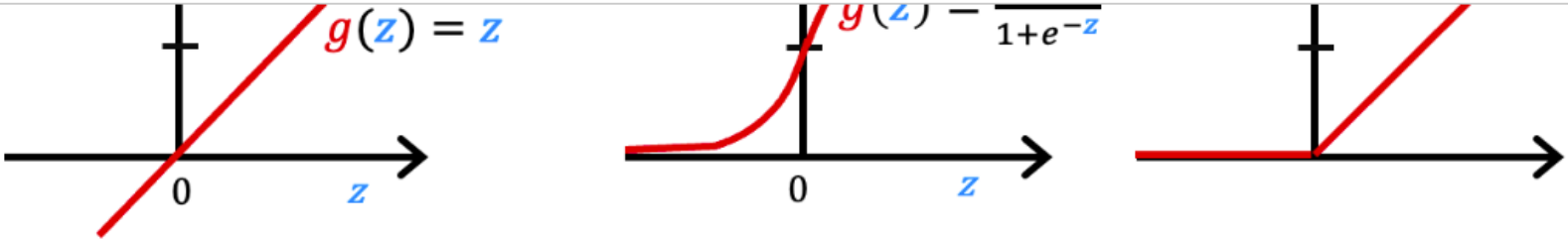
1.

1 / 1 point



Which of the following activation functions is the most common choice for the hidden layers of a neural network?

- (●) ReLU (rectified linear unit)



Which of the following activation functions is the most common choice for the hidden layers of a neural network?

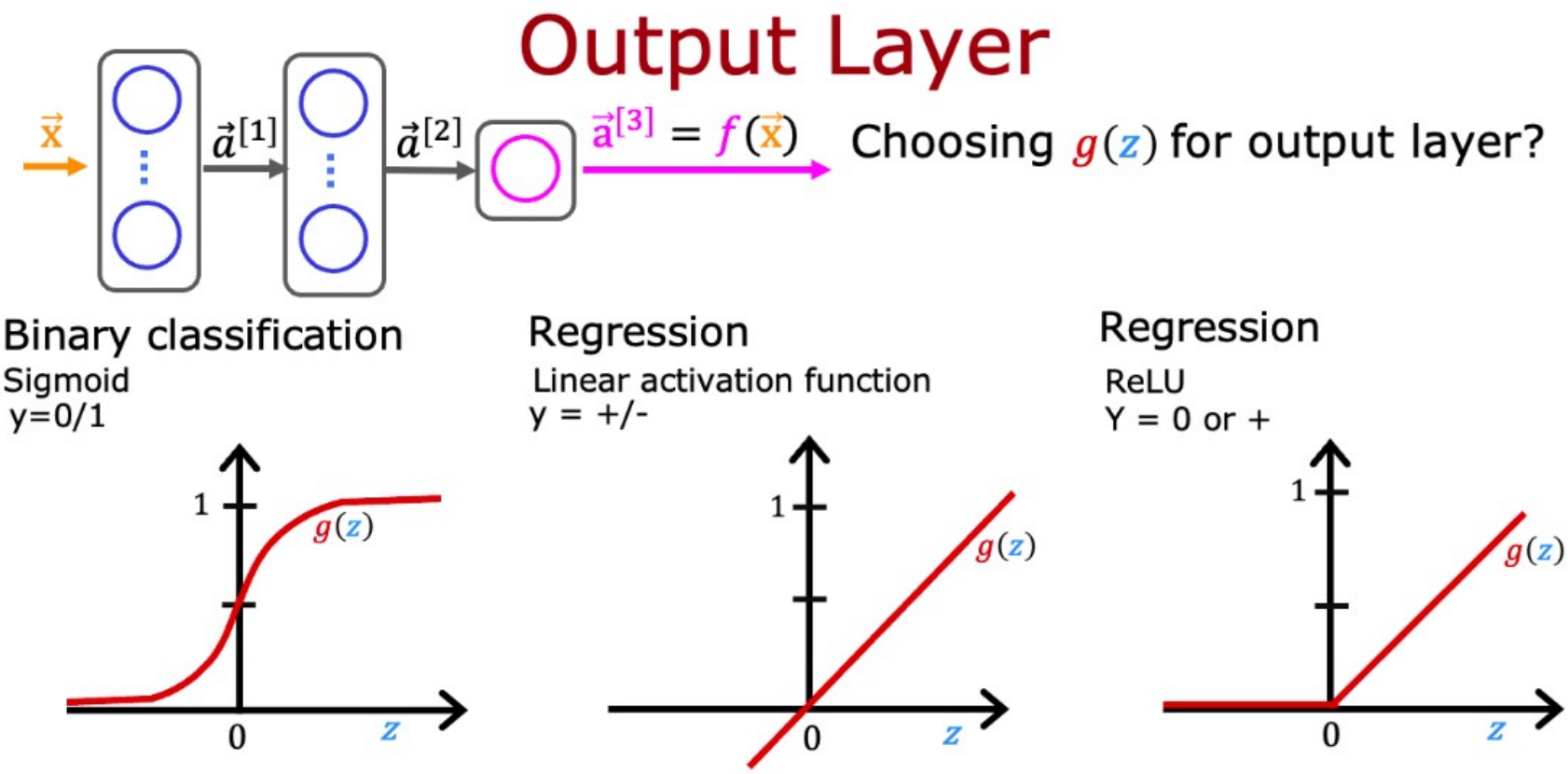
- (●) ReLU (rectified linear unit)
- ( ) Sigmoid
- ( ) Linear
- ( ) Most hidden layers do not use any activation function

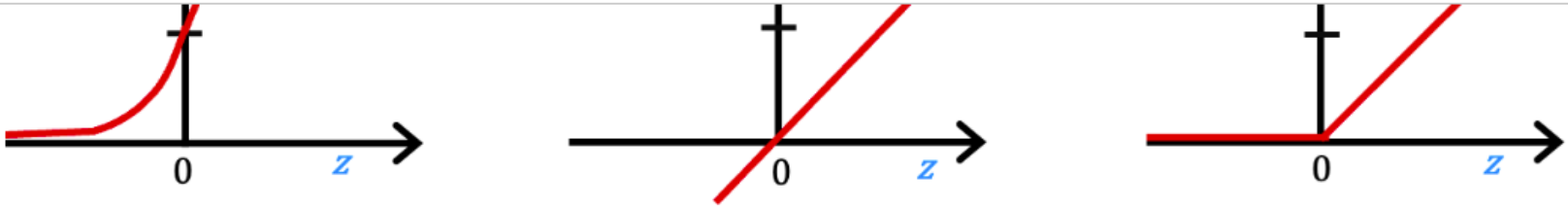
Correct

Yes! A ReLU is most often used because it is faster to train compared to the sigmoid. This is because the ReLU is only flat on one side (the left side) whereas the sigmoid goes flat (horizontal, slope approaching zero) on both sides of the curve.

2.

1 / 1 point


Output Layer
$\vec{x}$
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]} = f(\vec{x})$
Choosing $g(z)$ for output layer?
Binary classification
Sigmoid
y=0/1
1
$g(z)$
0
z
Regression
Linear activation function
y = +/-
1
$g(z)$
0
z
Regression
ReLU
Y = 0 or +
1
$g(z)$
0
z



For the task of predicting housing prices, which activation functions could you choose for the output layer? Choose the 2 options that apply.

- [ ] Sigmoid
- [x] linear

Correct
Yes! A linear activation function can be used for a regression task where the output can be both negative and positive, but it's also possible to use it for a task where the output is 0 or greater (like with house prices).

- [x] ReLU

Correct
Yes! ReLU outputs values 0 or greater, and housing prices are positive values.

- [x] linear

Correct
Yes! A linear activation function can be used for a regression task where the output can be both negative and positive, but it's also possible to use it for a task where the output is 0 or greater (like with house prices).

- [x] ReLU

Correct
Yes! ReLU outputs values 0 or greater, and housing prices are positive values.

3. True/False? A neural network with many layers but no activation function (in the hidden layers) is not effective; that's why we should instead use the linear activation function in every hidden layer.

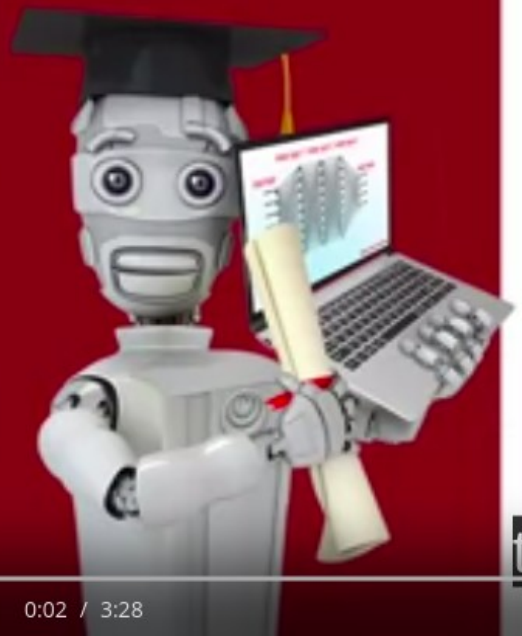
1 / 1 point

- (•) False
- ( ) True

Correct
Yes! A neural network with many layers but no activation function is not effective. A linear activation is the same as "no activation function".

DeepLearning.AI

Stanford
ONLINE



Multiclass Classification

Multiclass

Multiclass classification refers
to classification problems where

MNIST example

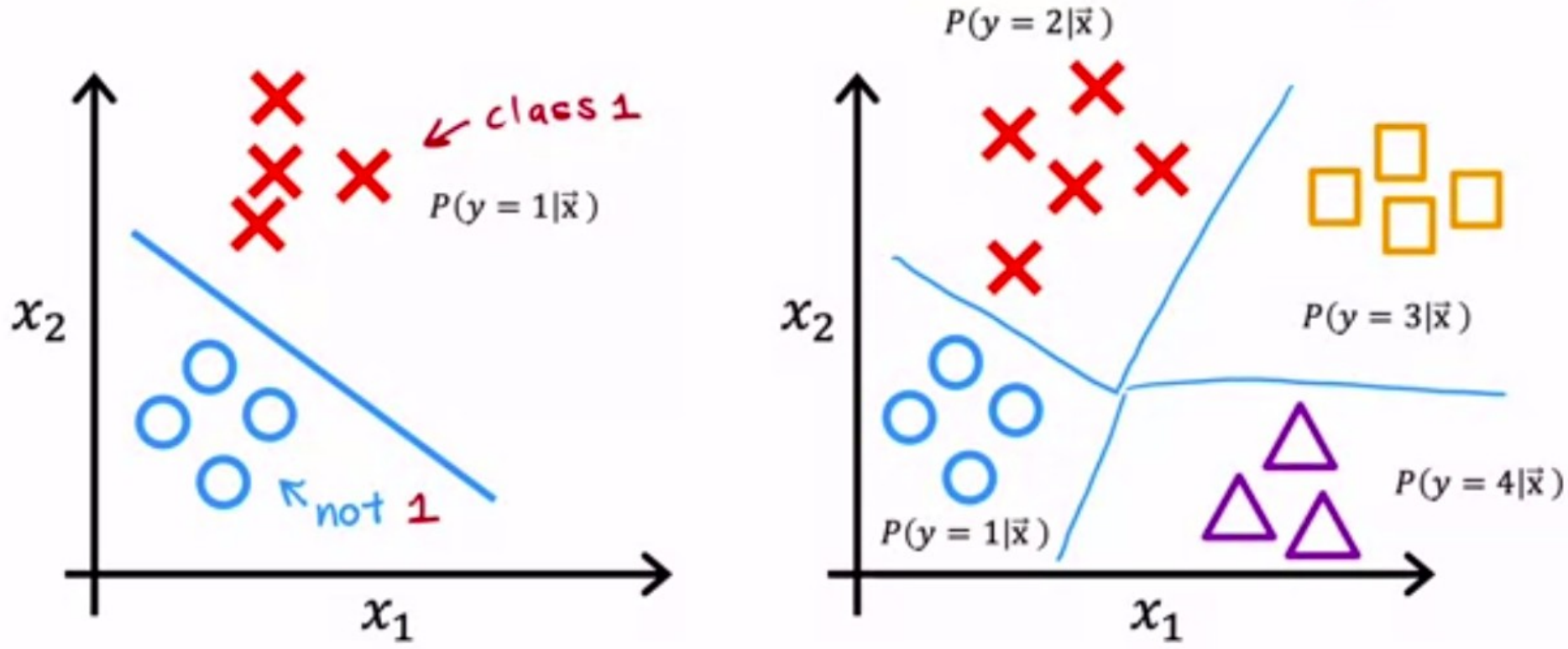
$y = 0$ 1 2 3 4 5 6 7 8 9

$y = 7$

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

but now y can take on more than just two possible values.

Multiclass classification example



exploded next to into four categories
rather than just two categories.

DeepLearning.AI

Stanford
ONLINE

Multiclass Classification

Softmax

a generalization of
logistic regression,

Question

What do you think a_4 is equal to?

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

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

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

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

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

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

Skip Continue

Question

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

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

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

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

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

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

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

Skip Continue

Question

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

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

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

- (•) 0.35
- ( ) 0.40
- ( ) -0.40

✓ **Correct**

This is correct. Please continue the video to see why!

Skip Continue

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

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

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

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

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

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

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

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

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

Softmax regression

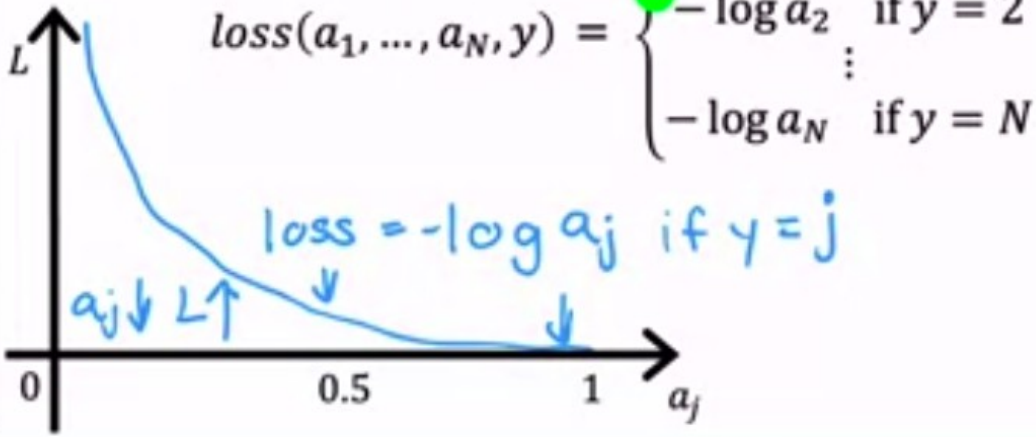
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$$

$$\vdots$$

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

Crossentropy loss

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



Cost

Logistic regression

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

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

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

a_2

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

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

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

Softmax regression

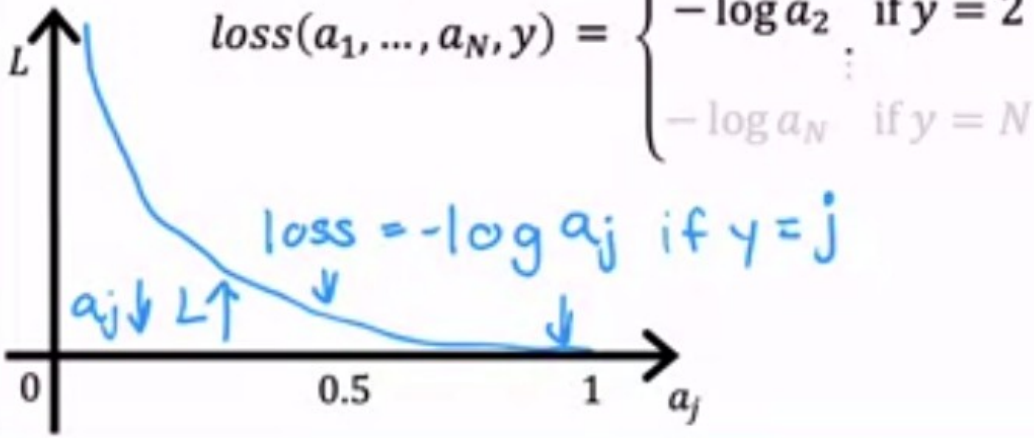
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$$

$$\vdots$$

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

Crossentropy loss

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



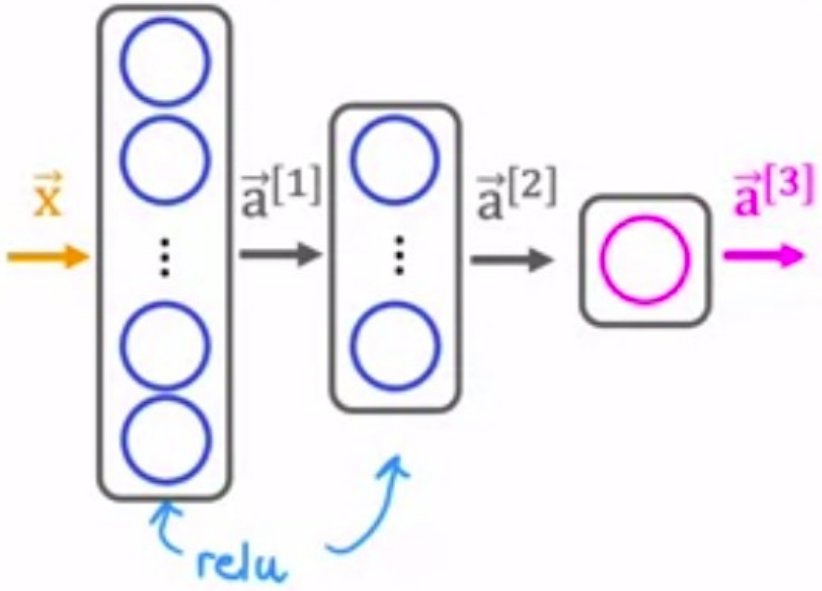
DeepLearning.AI

Stanford
ONLINE

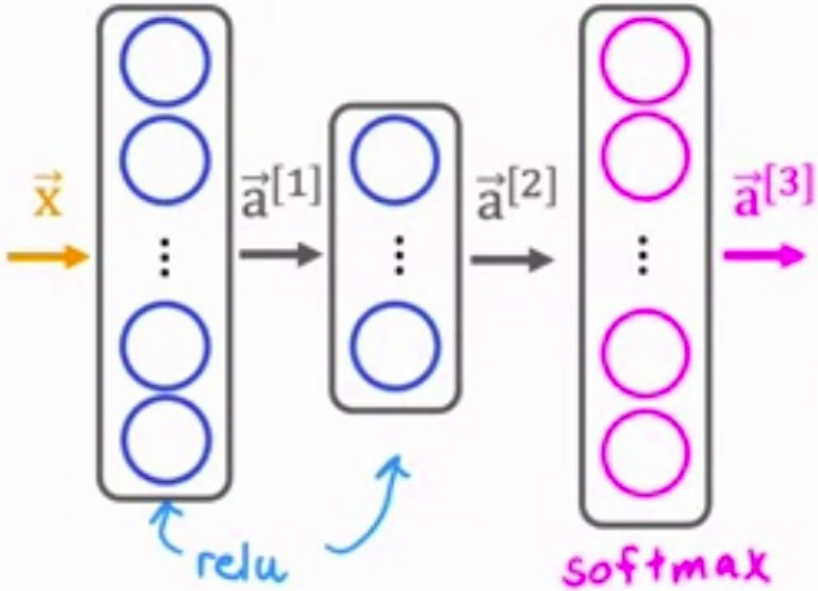
Multiclass Classification

Neural Network with Softmax output

Neural Network with Softmax output



Neural Network with Softmax output



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

$$\vdots$$

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

logistic regression

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

softmax

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

If you want to implement the neural network that I've shown here on

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

DeepLearning.AI

Stanford
ONLINE

Multiclass Classification

Improved implementation of softmax

Numerical Roundoff Errors

option 1

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

option 2

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

jupyter Numerical roundoff errors (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [1]:
```
x1 = 2.0 / 10000
print(f"{x1:.18f}") # print 18 digits to the right of decimal point
```

```
0.000200000000000000
```

In [2]:
```
x2 = 1 + (1/10000) - (1 - 1/10000)
print(f"{x2: .18f}")
```

```
 0.000199999999999978
```

In [ ]:

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

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

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

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)

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

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

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

'linear'

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

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')
  ])
```

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

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

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

```
model.compile(loss=SparseCategoricalCrossEntropy(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

x

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

yes
no
yes

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

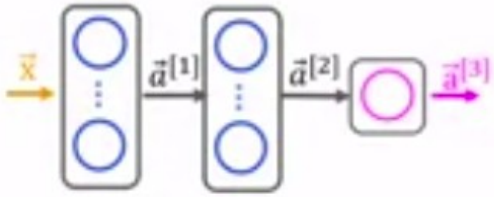
no
no
yes

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

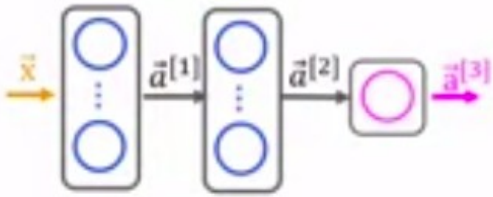
yes
yes
no

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

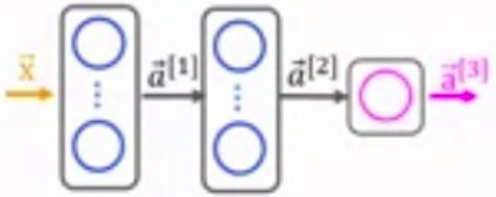
Multi-label Classification



car

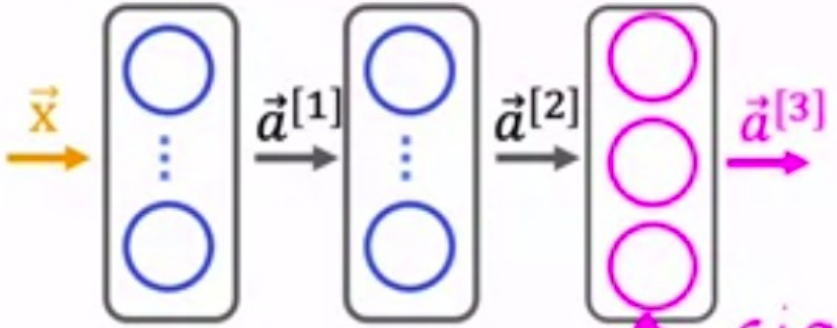


bus



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

Practice quiz: Multiclass Classification

Graded Quiz • 30 min

Due Nov 27, 11:59 PM IST

1.

1 / 1 point

Softmax regression (4 possible outputs)

✗ $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$ $\quad a_1 = \dfrac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

✗ ○ □ △

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

○ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$ $\quad a_2 = \dfrac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

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

□ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $\quad a_3 = \dfrac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

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

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

□ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $\quad a_3 = \dfrac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

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

△ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$ $\quad a_4 = \dfrac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

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

For a multiclass classification task that has 4 possible outputs, the sum of all the activations adds up to 1. For a multiclass classification task that has 3 possible outputs, the sum of all the activations should add up to ….

- ○ Less than 1

$$\triangle\ z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \qquad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

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

For a multiclass classification task that has 4 possible outputs, the sum of all the activations adds up to 1. For a multiclass classification task that has 3 possible outputs, the sum of all the activations should add up to ….

- ○ Less than 1
- ○ More than 1
- ◉ 1
- ○ It will vary, depending on the input x.

Correct

Yes! The sum of all the softmax activations should add up to 1. One way to see this is that if $e^{z_1} = 10, e^{z_2} = 20, e^{z_3} = 30$, then the sum of $a_1 + a_2 + a_3$ is equal to $\frac{e^{z_1}+e^{z_2}+e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$ which is 1.

2.

1 / 1 point

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

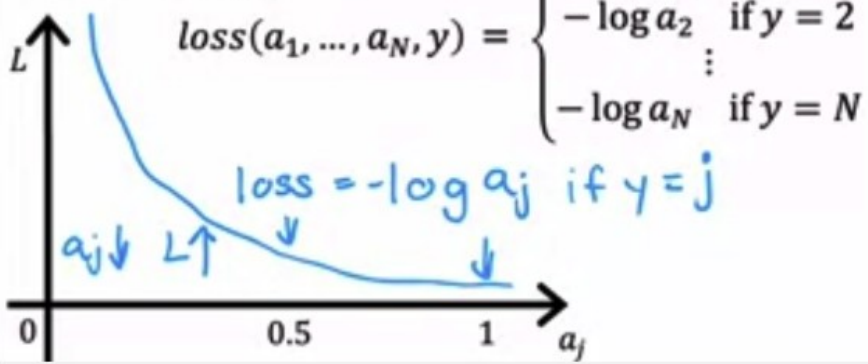
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$$

$$\vdots$$

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

Crossentropy loss

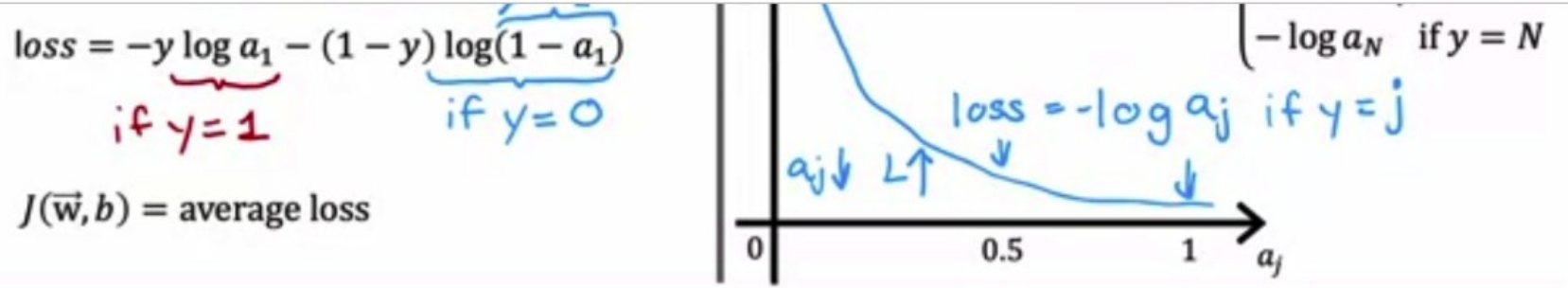
$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \\ -\log a_N & \text{if } y=N \end{cases}$$



Practice quiz: Multiclass Classification

Graded Quiz • 30 min

Due Nov 27, 11:59 PM IST



For multiclass classification, the cross entropy loss is used for training the model. If there are 4 possible classes for the output, and for a particular training example, the true class of the example is class 3 (y=3), then what does the cross entropy loss simplify to? [Hint: This loss should get smaller when a_3 gets larger.]

- z_3
- $\frac{-log(a_1)+-log(a_2)+-log(a_3)+-log(a_4)}{4}$
- z_3/(z_1+z_2+z_3+z_4)
- $-log(a_3)$

Correct

Correct. When the true label is 3, then the cross entropy loss for that training example is just the negative of the log of the activation for the third neuron of the softmax. All other terms of the cross entropy loss equation ($-log(a_1), -log(a_2), and\ -log(a_4)$) are ignored

3.

1 / 1 point

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

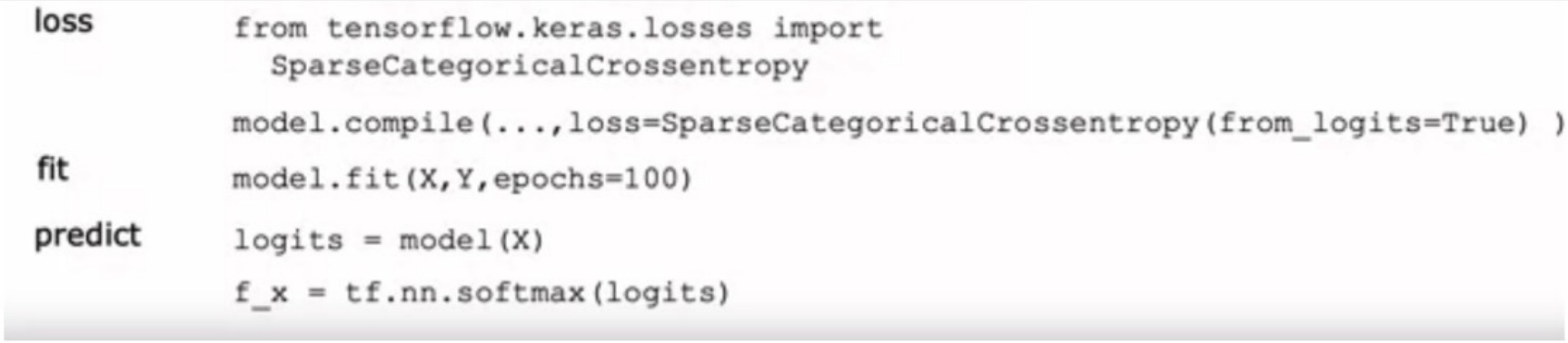
```
from tensorflow.keras.losses import
  SparseCategoricalCrossentropy
model.compile(...,loss=SparseCategoricalCrossentropy(from_logits=True) )
```

fit

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

predict

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



For multiclass classification, the recommended way to implement softmax regression is to set from_logits=True in the loss function, and also to define the model's output layer with…

- (●) a 'linear' activation
- ( ) a 'softmax' activation

Correct
Yes! Set the output as linear, because the loss function handles the calculation of the softmax with a more numerically stable method.

DeepLearning.AI

Stanford
ONLINE

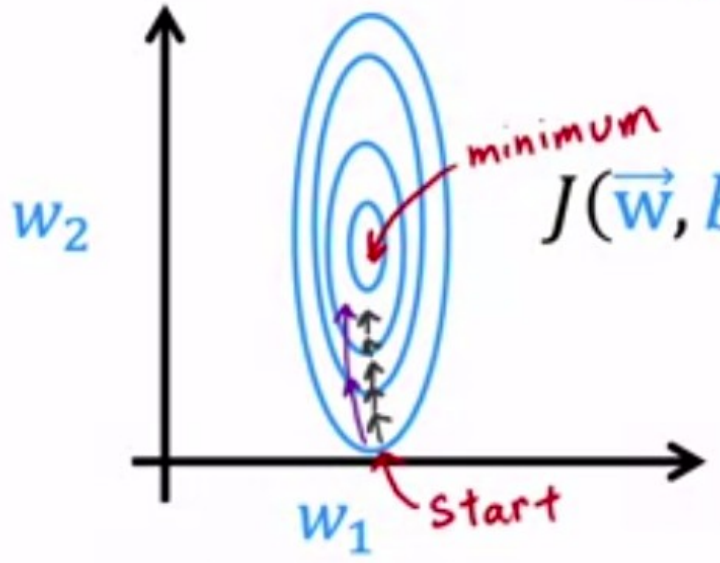
Additional Neural Network Concepts

Advanced Optimization

Gradient Descent

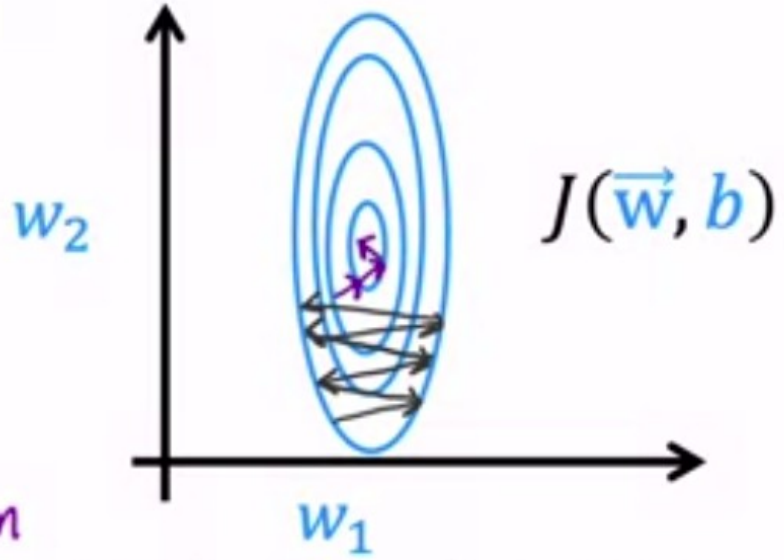
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

learning rate



"Adam" algorithm

Go faster – increase α



Go slower – decrease α

a smaller learning rate Alpha.

Adam Algorithm Intuition

Adam: Adaptive Moment estimation

not just one α

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

$$\vdots$$

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

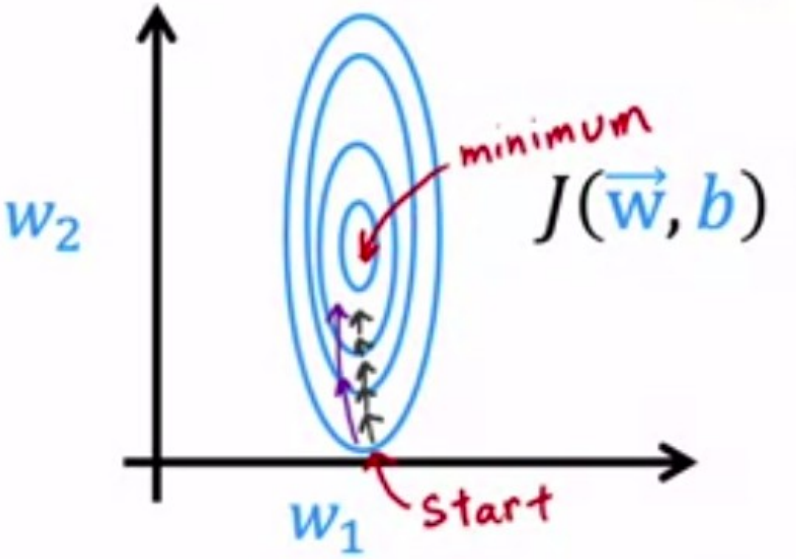
$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{w}, b)$$

as well as I'll call it
Alpha_11 for the parameter b.

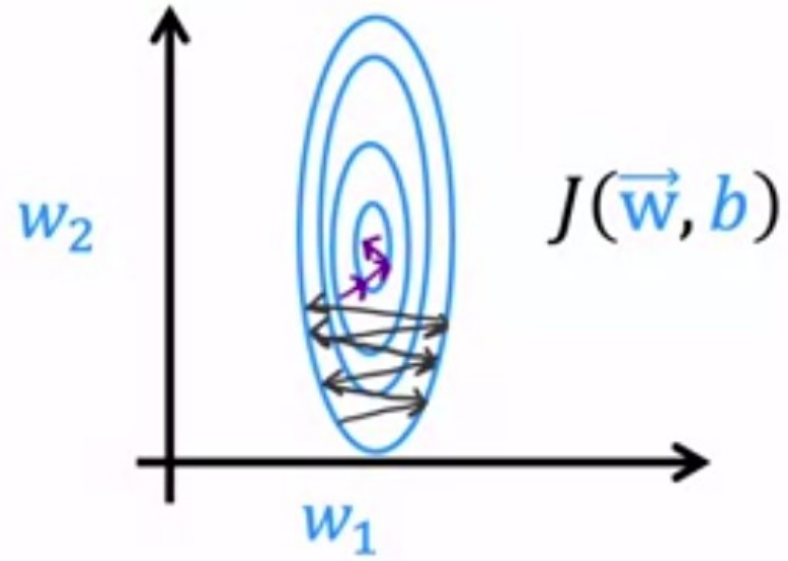
MNIST Adam

but in codes this is
how you implement it.

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.

you may learn more about the details

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)
```

if you can get somewhat faster learning.

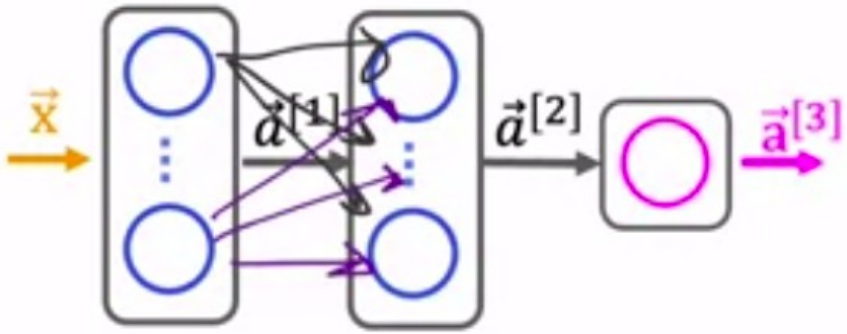
DeepLearning.AI

Stanford
ONLINE

Additional Neural Network Concepts

Additional Layer Types

Dense Layer

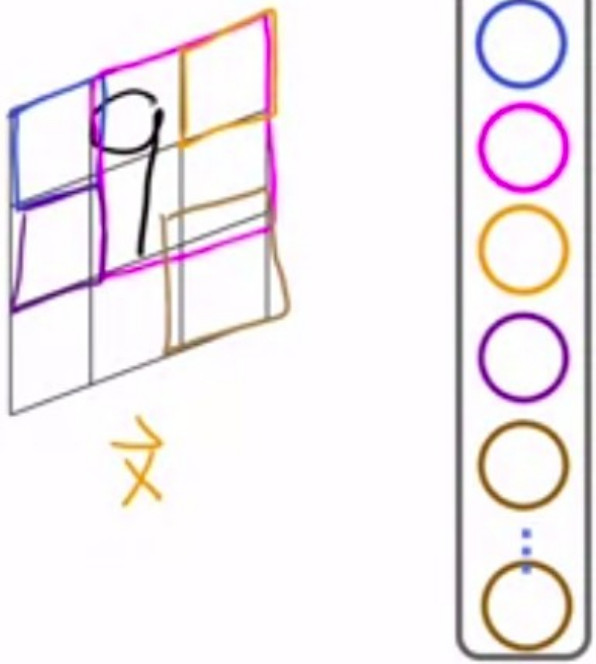


Each neuron output is a function of all the activation outputs of the previous layer.

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

someone designing a neural network may choose to use a different type of layer.

Convolutional Layer



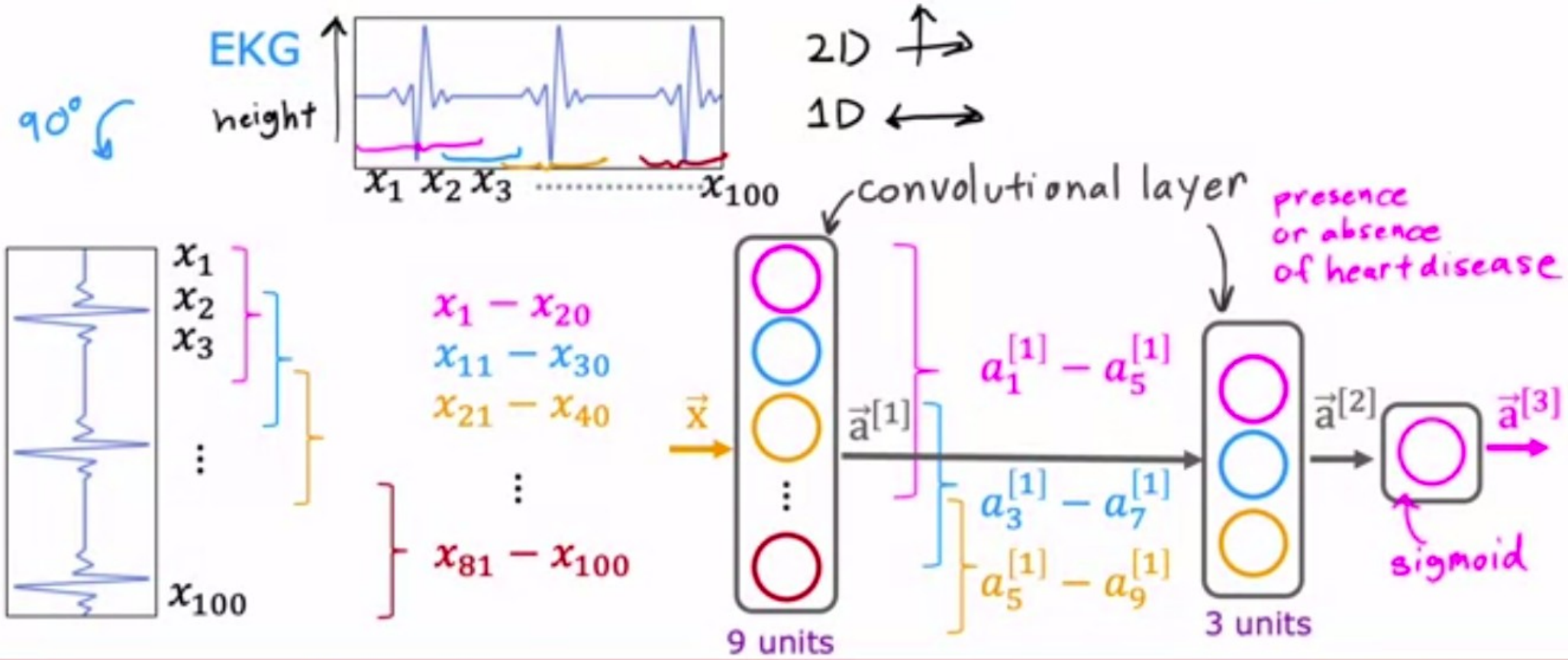
Each Neuron only looks at part of the previous layer's inputs.

Why?

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

of how to get convolutional layers to work and popularized their use.

Convolutional Neural Network



1.

1 / 1 point

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)
```

```
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)
```

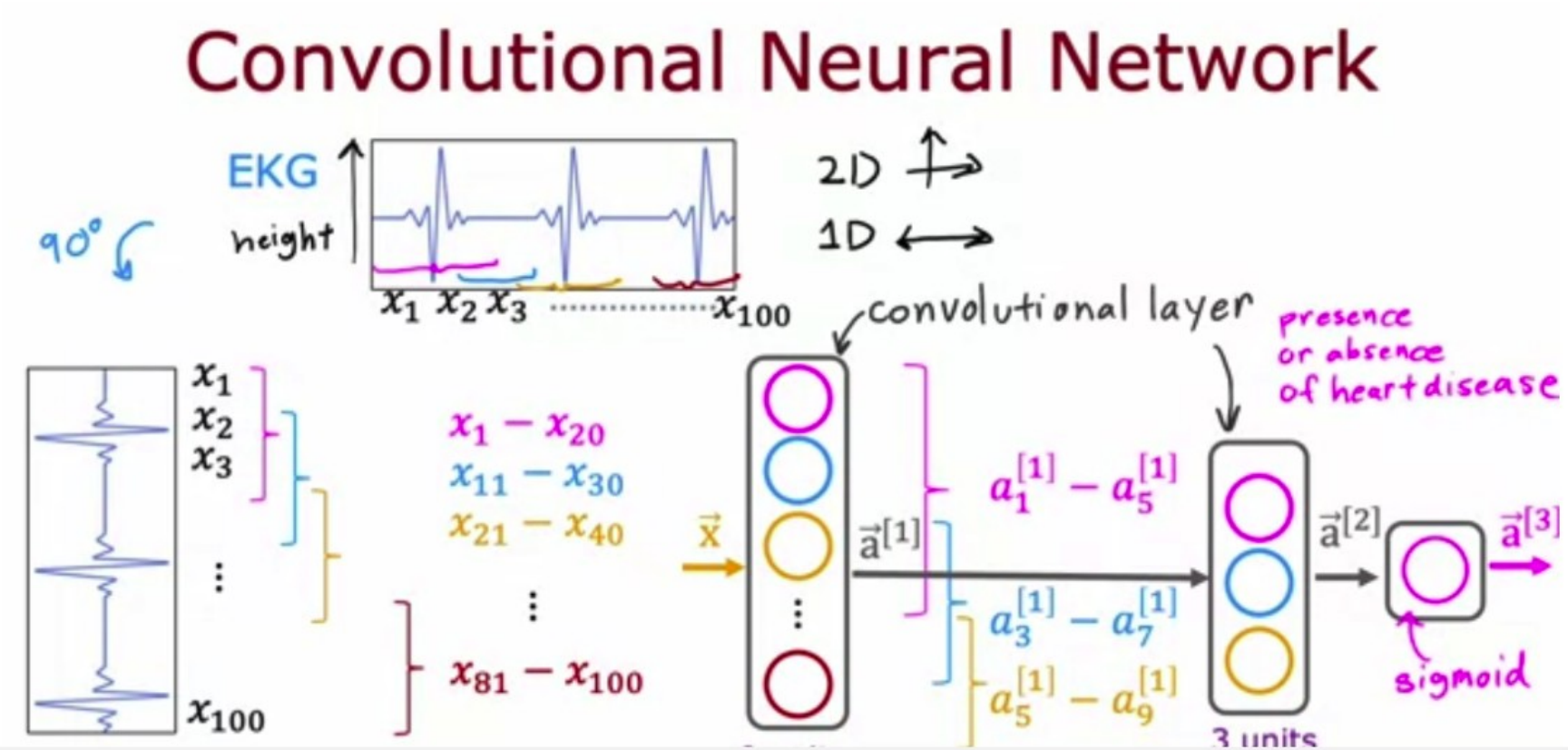
The Adam optimizer is the recommended optimizer for finding the optimal parameters of the model. How do you use the Adam optimizer in TensorFlow?

- ○ The call to model.compile() will automatically pick the best optimizer, whether it is gradient descent, Adam or something else. So there's no need to pick an optimizer manually.
- ◉ When calling model.compile, set optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3).
- ○ The call to model.compile() uses the Adam optimizer by default
- ○ The Adam optimizer works only with Softmax outputs. So if a neural network has a Softmax output layer, TensorFlow will automatically pick the Adam optimizer.

Correct
Correct. Set the optimizer to Adam.

2.

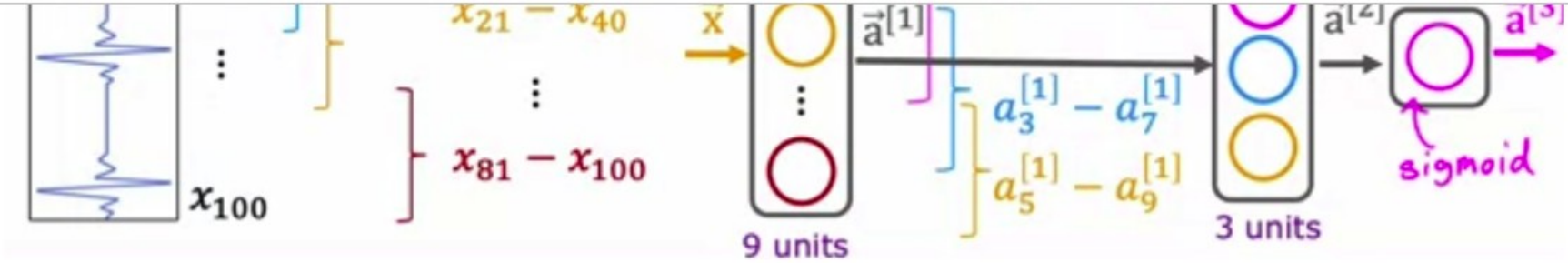
1 / 1 point


Convolutional Neural Network
EKG
height
90°
2D
1D
x_1 x_2 x_3 ..... x_{100}
convolutional layer
presence or absence of heart disease
x_1
x_2
x_3
x_{100}
$x_1 - x_{20}$
$x_{11} - x_{30}$
$x_{21} - x_{40}$
$x_{81} - x_{100}$
$\vec{x}$
$\vec{a}^{[1]}$
$a_1^{[1]} - a_5^{[1]}$
$a_3^{[1]} - a_7^{[1]}$
$a_5^{[1]} - a_9^{[1]}$
3 units
$\vec{a}^{[2]}$
$\vec{a}^{[3]}$
sigmoid

Practice quiz: Additional Neural Network Concepts

Graded Quiz • 30 min

Due Mar 26, 11:59 PM IST



The lecture covered a different layer type where each single neuron of the layer does not look at all the values of the input vector that is fed into that layer. What is this name of the layer type discussed in lecture?

- (●) convolutional layer
- ( ) Image layer
- ( ) A fully connected layer
- ( ) 1D layer or 2D layer (depending on the input dimension)

Correct

Correct. For a convolutional layer, each neuron takes as input a subset of the vector that is fed into that layer.

DeepLearning.AI

Stanford
ONLINE

Backprop Intuition (Optional)

What is a derivative?

Derivative Example

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

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

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

$\varepsilon = 0.002$

$w = 3 + 0.001$ 0.002

$J(w) = w^2 = 9.006001$

9.012004

9.012

If $w \uparrow$ 0.002 0.001 ε

$J(w) \uparrow 6 \times 0.002$ 0.001 $6 \times \varepsilon$ $6 \times 0.002 = 0.012$

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

Informal Definition of Derivative

If $w \uparrow \varepsilon$ causes $J(w) \uparrow k \times \varepsilon$ then

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

Gradient descent

repeat {

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

learning rate

}

If derivative is small, then this update step will make a small update to w_j

If the derivative is large, then this update step will make a large update to w_j

More Derivative Examples

$w = 3$ $\quad J(w) = w^2 = 9$ $\quad w \uparrow 0.001$ $\quad J(w) = J(3.001) = 9.006001$

$J(w) \uparrow 6 \times 0.001$ $\quad \frac{\partial}{\partial w} J(w) = 6$

$w = 2$ $\quad J(w) = w^2 = 4$ $\quad w \uparrow 0.001$ $\quad J(w) = J(2.001) = 4.004001$

$J(w) \uparrow 4 \times 0.001$ $\quad \frac{\partial}{\partial w} J(w) = 4$

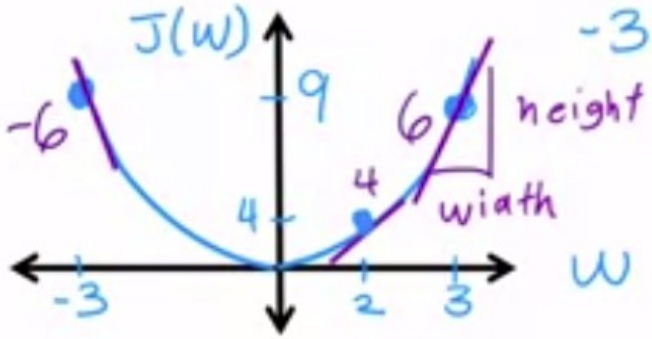
$J(w) = w^2$

$\downarrow 0.006$

$w = -3$ $\quad J(w) = w^2 = 9$ $\quad w \uparrow 0.001$ $\quad J(w) = J(-2.999) = 8.994001$

$J(w) \downarrow 6 \times 0.001$ $\quad \frac{\partial}{\partial w} J(w) = -6$

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



-3 + 0.001

Calculus	w	$\partial J(w)/\partial w$
$\frac{\partial}{\partial w} J(w) = 2w$	3	$2 \times 3 = 6$
	2	$2 \times 2 = 4$
	-3	$2 \times -3 = -6$

Even More Derivative Examples

$w = 2$

- $J(w) = w^2 = 4$ — $\frac{\partial}{\partial w} J(w) = 2w = 4$ — $w \uparrow 0.001$ (ε) — $J(w) = 4.004001$, $J(w) \uparrow 4 \times \varepsilon$
- $J(w) = w^3 = 8$
- $J(w) = w = 2$
- $J(w) = \frac{1}{w} = \frac{1}{2} = 0.5$

jupyter Using code to get derivatives (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [1]:
```
import sympy
```

In [2]:
```
J, w = sympy.symbols('J,w')
```

In [27]:
```
J = w**3
J
```

Out[27]: w^3

In [28]:
```
dJ_dw = sympy.diff(J,w)
dJ_dw
```

Out[28]: $3w^2$

In [29]:
```
dJ_dw.subs([(w,2)])
```

Out[29]: 12

In [ ]:

jupyter Using code to get derivatives (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [2]: J, w = sympy.symbols(J,w)

In [3]:
```
J = w**2
J
```

Out[3]: w^2

In [5]:
```
dJ_dw = sympy.diff(J,w)
dJ_dw
```

Out[5]: $2w$

In [6]:
```
dJ_dw.subs([(w,2)])
```

Out[6]: 4

In [ ]:

jupyter Using code to get derivatives (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [2]:
```
J, w = sympy.symbols('J,w')
```

In [7]:
```
J=w**3 #J = w**2
J
```

Out[7]: w^3

In [8]:
```
dJ_dw = sympy.diff(J,w)
dJ_dw
```

Out[8]: $3w^2$

In [9]:
```
dJ_dw.subs([(w,2)])
```

Out[9]: 12

In [ ]:

jupyter Using code to get derivatives (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [2]:
```
J, w = sympy.symbols('J,w')
```

In [10]:
```
J=w #J=w**3 #J = w**2
J
```

Out[10]: w

In [11]:
```
dJ_dw = sympy.diff(J,w)
dJ_dw
```

Out[11]: 1

In [12]:
```
dJ_dw.subs([(w,2)])
```

Out[12]: 1

In [ ]:

jupyter Using code to get derivatives (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [2]:
```
J, w = sympy.symbols('J,w')
```

In [13]:
```
J=1/w #J=w #J=w**3 #J = w**2
J
```

Out[13]: $\frac{1}{w}$

In [14]:
```
dJ_dw = sympy.diff(J,w)
dJ_dw
```

Out[14]: $-\frac{1}{w^2}$

In [15]:
```
dJ_dw.subs([(w,2)])
```

Out[15]: $-\frac{1}{4}$

In [ ]:

jupyter Using code to get derivatives (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code Validate

In [1]:
```
import sympy
```

In [2]:
```
J, w = sympy.symbols('J,w')
```

In [13]:
```
J=1/w #J=w #J=w**3 #J = w**2
J
```

Out[13]: $\frac{1}{w}$

In [14]:
```
dJ_dw = sympy.diff(J,w)
dJ_dw
```

Out[14]: $-\frac{1}{w^2}$

In [15]:
```
dJ_dw.subs([(w,2)])
```

Out[15]: $-\frac{1}{4}$

In [ ]:

Even More Derivative Examples

$w = 2$

$J(w) = w^2 = 4$	$\frac{\partial}{\partial w} J(w) = 2w = 4$	$w \uparrow 0.001$ (ε)	$J(w) = 4.004001$ $J(w) \uparrow 4 \times \varepsilon$
$J(w) = w^3 = 8$	$\frac{\partial}{\partial w} J(w) = 3w^2 = 12$	$w \uparrow \varepsilon$	$J(w) = 8.012006$ $J(w) \uparrow 12 \times \varepsilon$
$J(w) = w = 2$	$\frac{\partial}{\partial w} J(w) = 1$	$w \uparrow \varepsilon$	$J(w) = 2.001$ $J(w) \uparrow 1 \times \varepsilon$
$J(w) = \frac{1}{w} = \frac{1}{2} = 0.5$	$\frac{\partial}{\partial w} J(w) = \frac{-1}{w^2} = \frac{-1}{4}$	$w \uparrow \varepsilon$ $w = \frac{1}{2.001}$	-0.25×0.001 $0.5 - 0.00025$ $J(w) = 0.49975$ $J(w) \uparrow -\frac{1}{4} \times \varepsilon$

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

A note on derivative notation

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

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

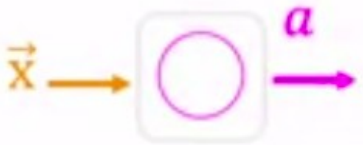
If $J(w_1, w_2, \ldots, w_n)$ is a function of more than one variable,

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

"partial derivative"

notation used in these courses

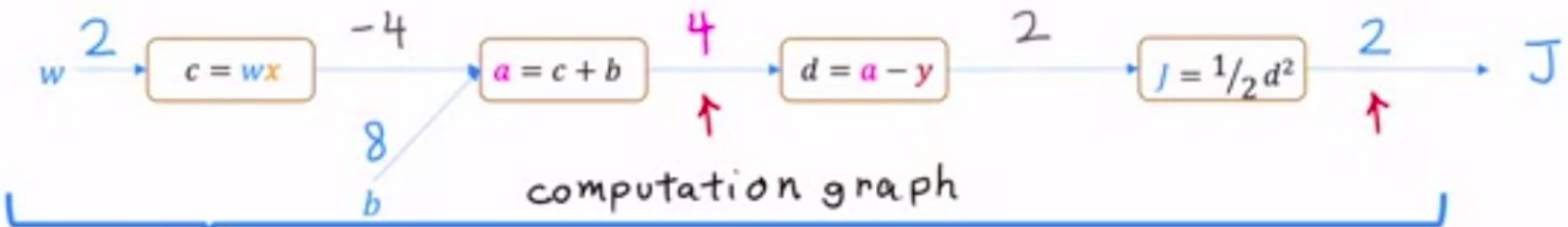
Small Neural Network Example



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

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

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



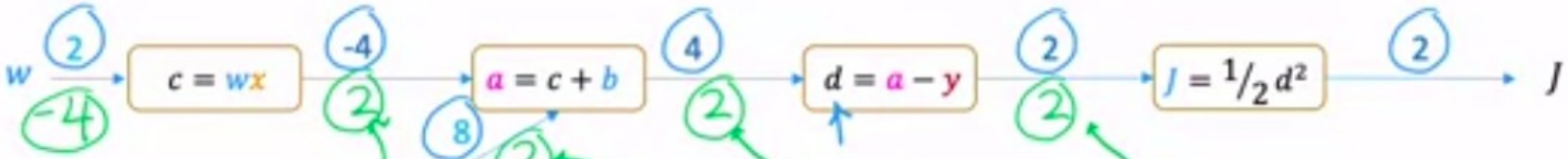
how do we find the derivatives of J? $\frac{\partial J}{\partial w}$ $\frac{\partial J}{\partial b}$

Computing the Derivatives

Forward prop →

Back prop ←

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



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

$w = 2.001 \quad c = -4.002$

$w \uparrow 0.001 \quad c \downarrow 2 \times 0.001$

$c \uparrow -2 \times 0.001$

$J \uparrow -4 \times 0.001$

$$\frac{\partial J}{\partial w} = \frac{\partial c}{\partial w} \times \frac{\partial J}{\partial c}$$

$-2 \quad 2$

$c \uparrow 0.001 \quad a \uparrow 0.001 \quad J \uparrow 0.002$

$$\frac{\partial J}{\partial c} = \frac{\partial a}{\partial c} \times \frac{\partial J}{\partial a}$$

$1 \quad 2$

$b \uparrow 0.001 \quad a \uparrow 0.001 \quad J \uparrow 0.002$

$$\frac{\partial J}{\partial b} = \frac{\partial a}{\partial b} \times \frac{\partial J}{\partial a}$$

$1 \quad 2$

$a \uparrow 0.001 \quad d \uparrow 0.001$

$a = 4.001 \quad d = 2.001$

$J \uparrow 0.002$

$$\frac{\partial J}{\partial a} = \frac{\partial d}{\partial a} \times \frac{\partial J}{\partial d}$$

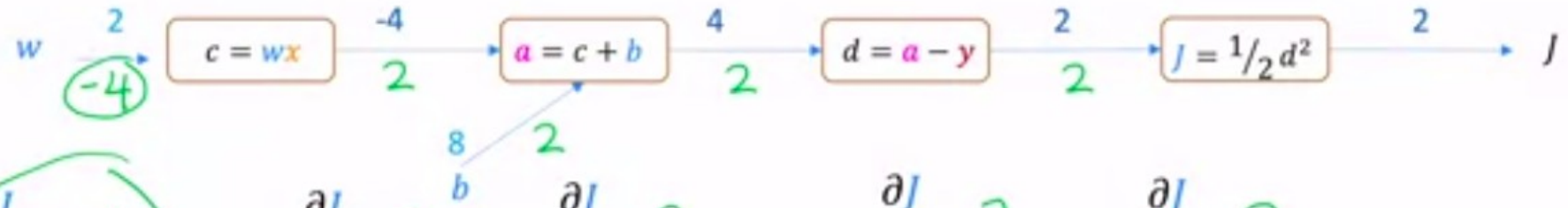
$1 \quad 2$

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

chain rule (calculus)

Computing the Derivatives

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



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

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

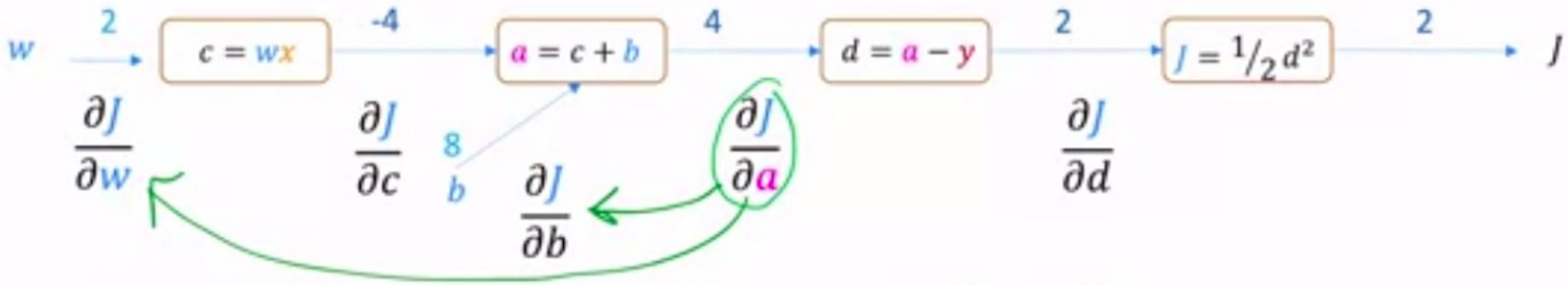
$w \uparrow 0.001 \qquad J = \frac{1}{2}\left((2.001 \times -2 + 8) - 2\right)^2 = 1.996002$

$J \downarrow 4 \times 0.001$

$J \uparrow -4 \times 0.001$

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

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

Neural Network Example

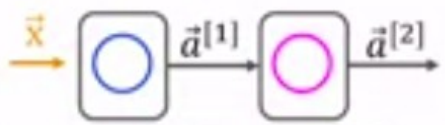
$x = 1 \quad y = 5$

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

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

ReLU activation

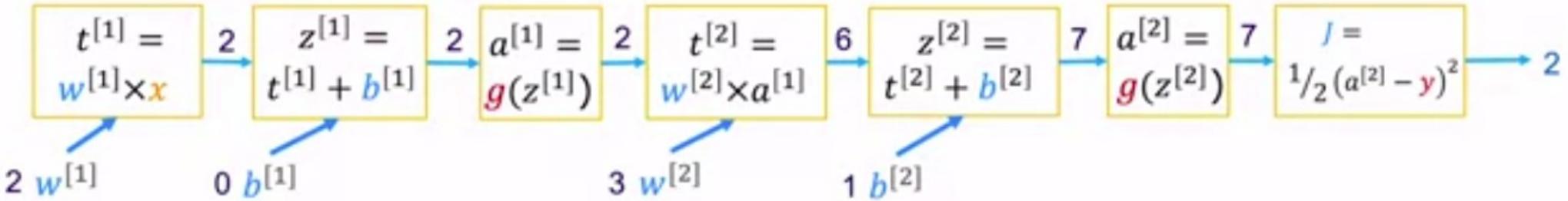
$g(z) = \max(0, z)$



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

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

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



Neural Network Example

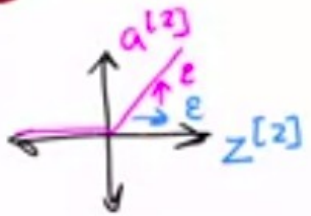
$x = 1 \quad y = 5$

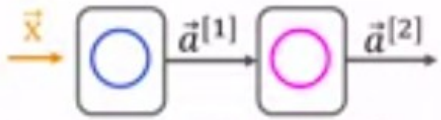
$w^{[1]} = 2, b^{[1]} = 0$

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

ReLU activation

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

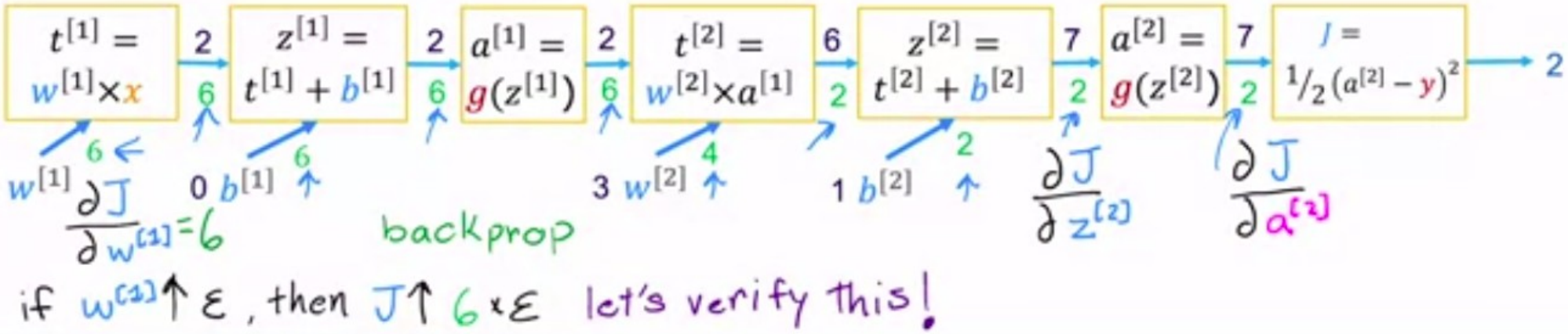




$a^{[1]} = g(w^{[1]} x + b^{[1]}) = \overbrace{w^{[1]} x + b^{[1]}}^{z^{[1]}} = 2 \times 1 + 0 = 2$

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

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



Neural Network Example

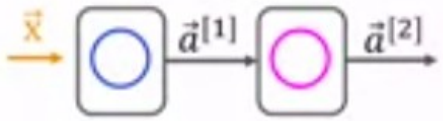
$x = 1 \quad y = 5$

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

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

ReLU activation

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



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

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

$J(w, b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(7 - 5)^2 = 2$ → $\frac{(2.003)^2}{2} = 2.006005$

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

$J \uparrow 6 \times 0.001$

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

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

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

N nodes

P parameters

$w_1, b_1, w_2, b_2 \ldots$

inefficient way

$N \times P$

efficient way (backprop)

$N + P$