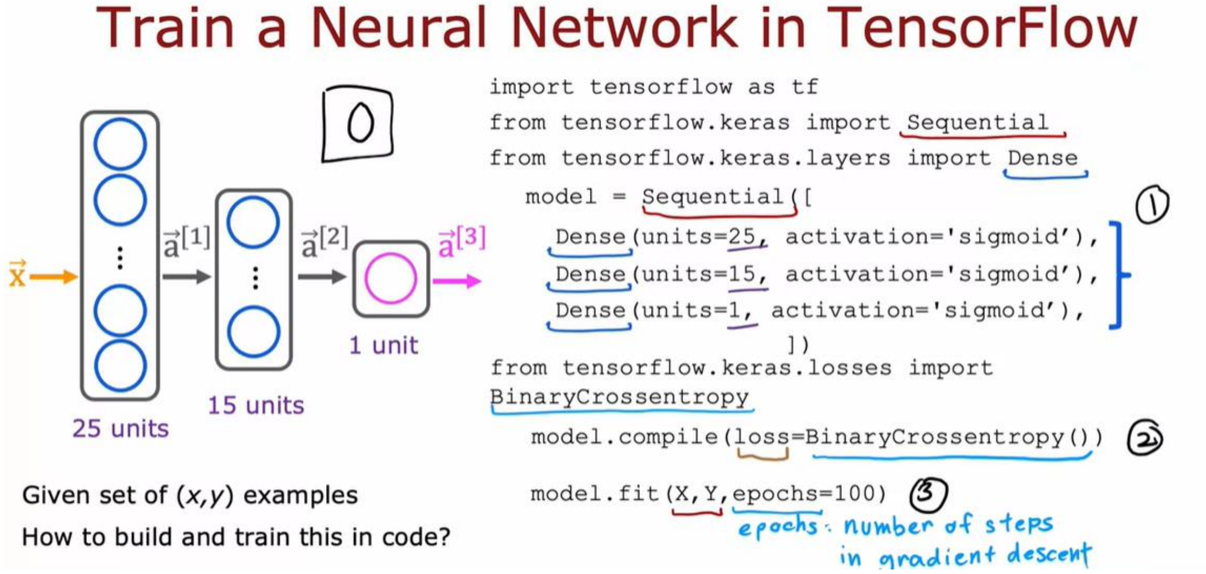
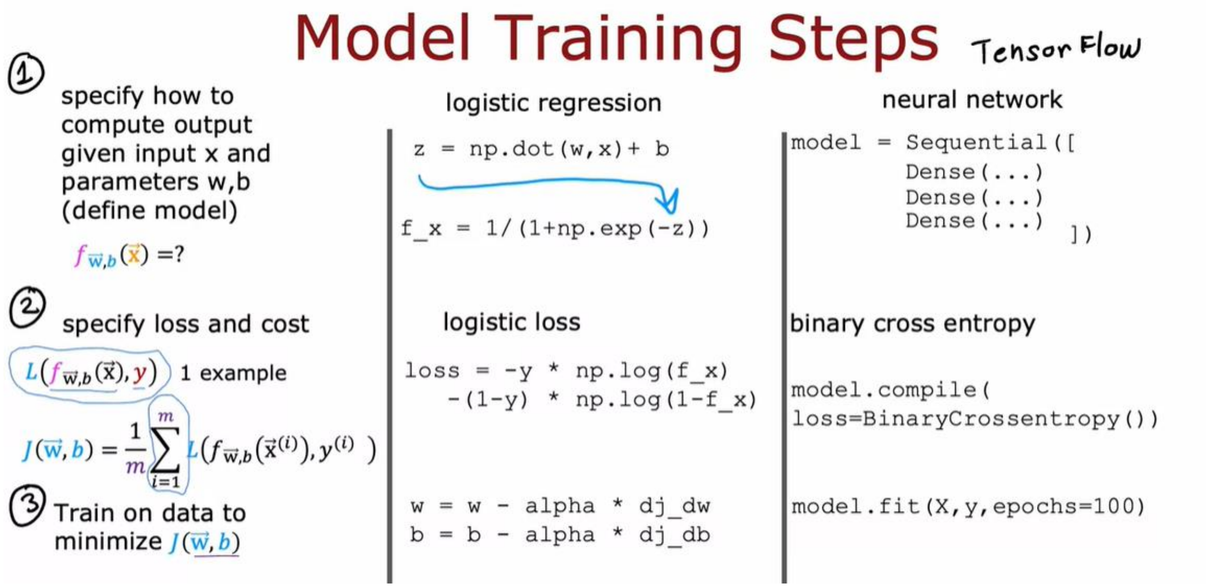
Neural Network Training

1. TensorFlow implementation



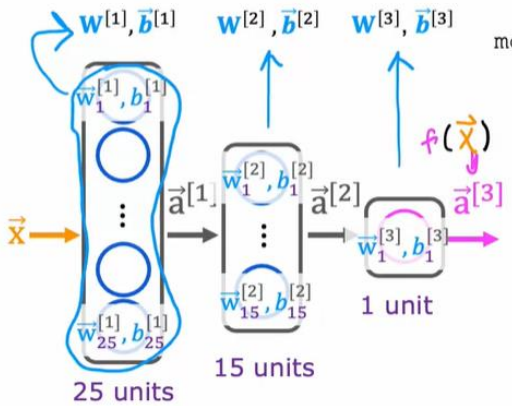
2. Training details



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

handwritten 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

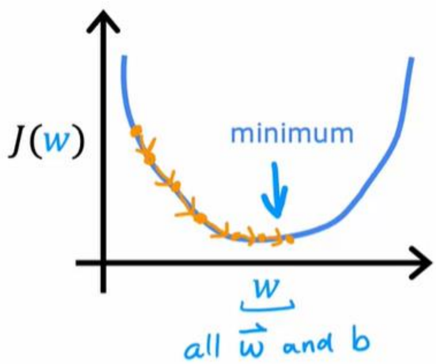
regression
(predicting numbers and not categories)

mean squared error

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

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

3. Gradient descent



repeat {

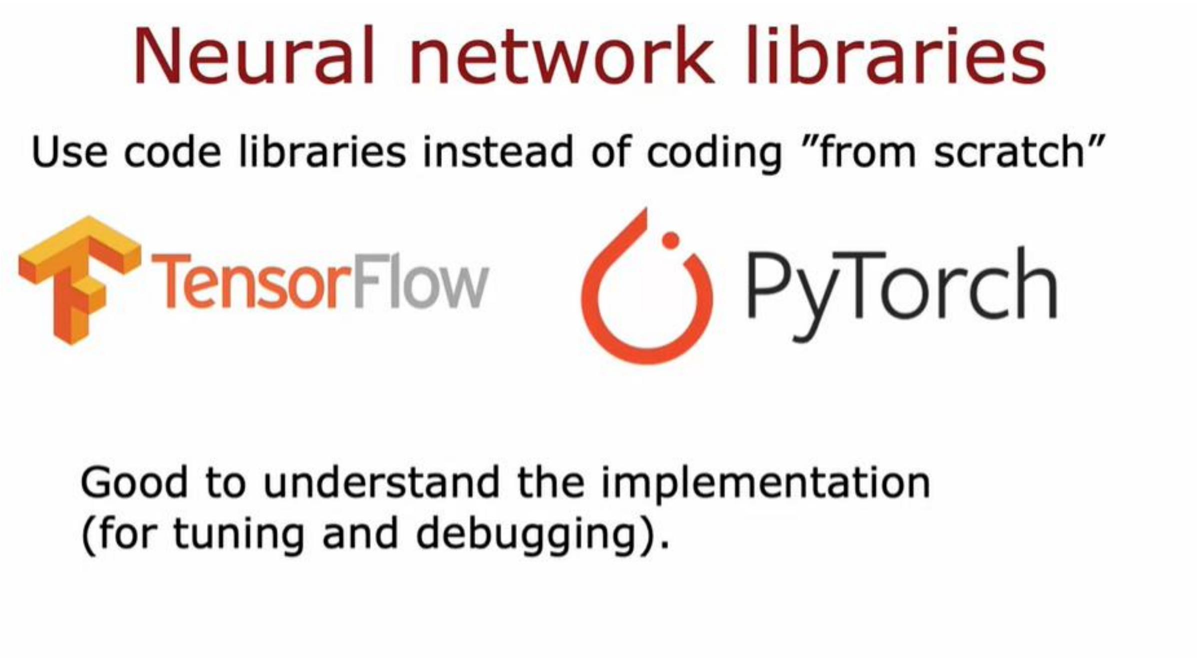
$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

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

}

Compute derivatives for gradient descent using "back propagation"

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



3. Practice quiz

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?

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

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

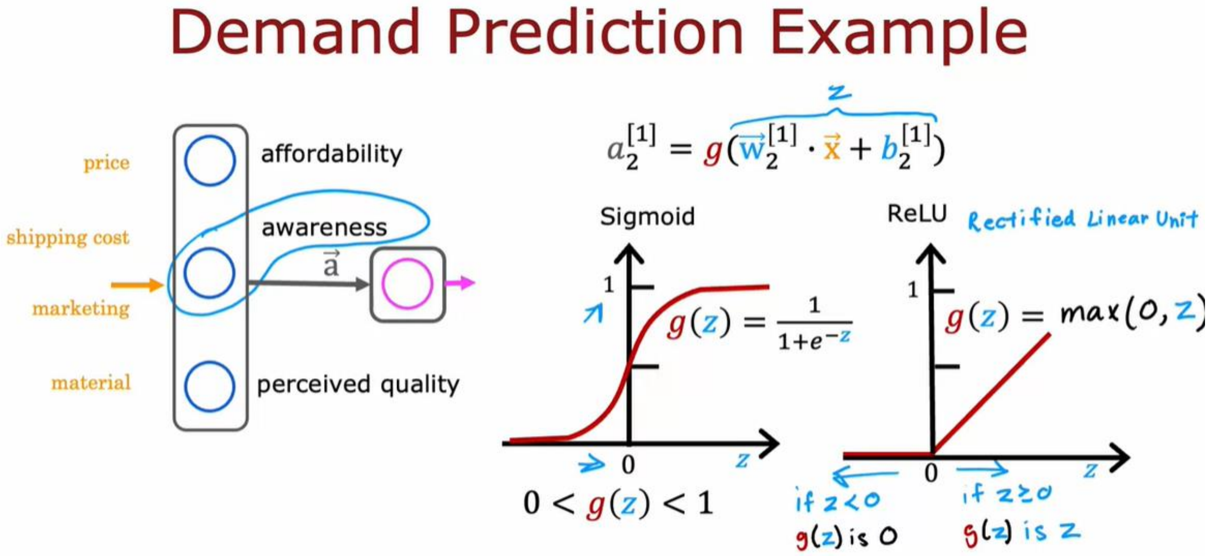
Which line of code updates the network parameters in order to reduce the cost?

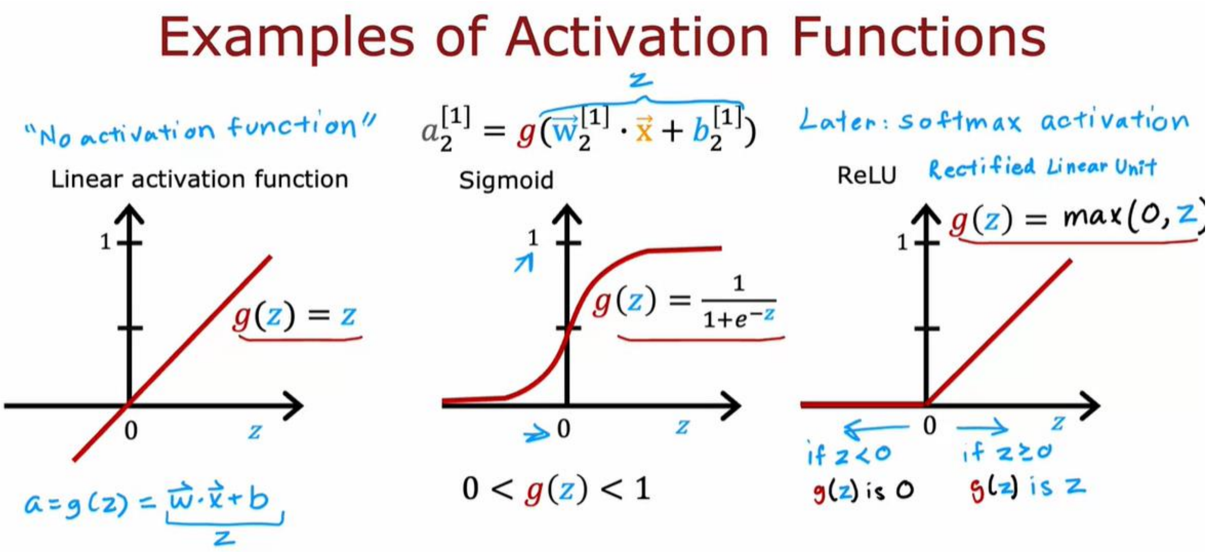
- model.fit(X,y,epochs=100)
- model = Sequential([...])
- None of the above -- this code does not update the network parameters.
- model.compile(loss=BinaryCrossentropy())

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

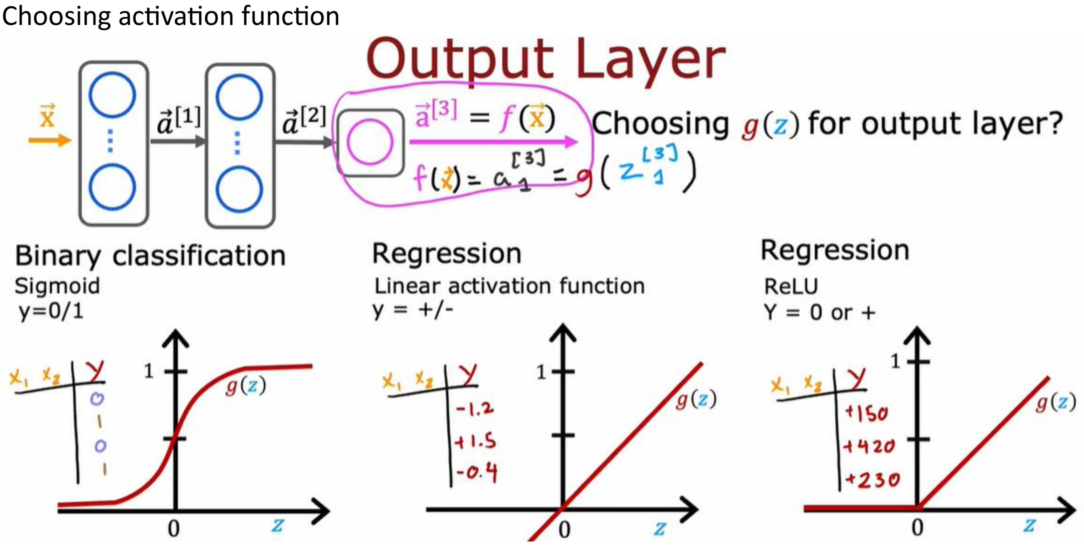
Activation Function

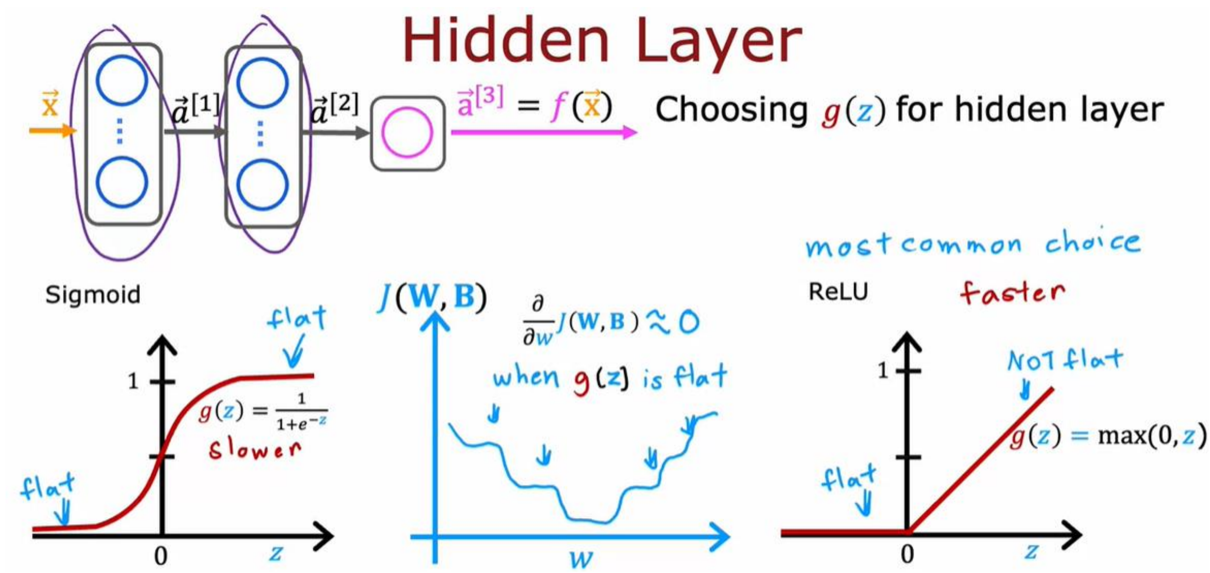
1. Alternatives to the sigmoid activation

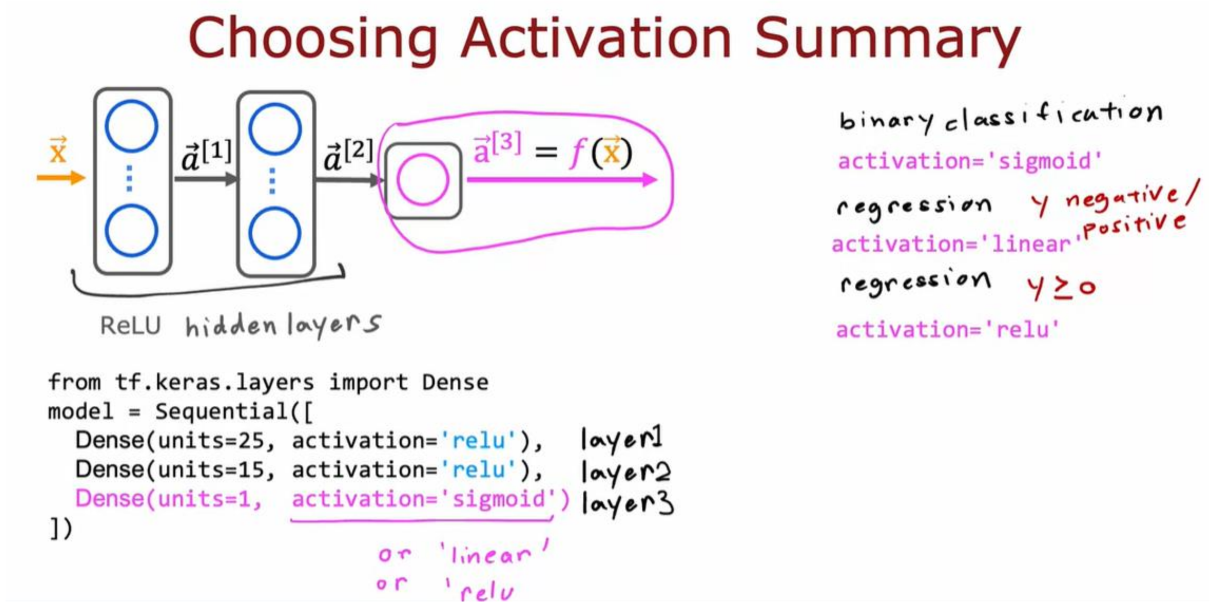




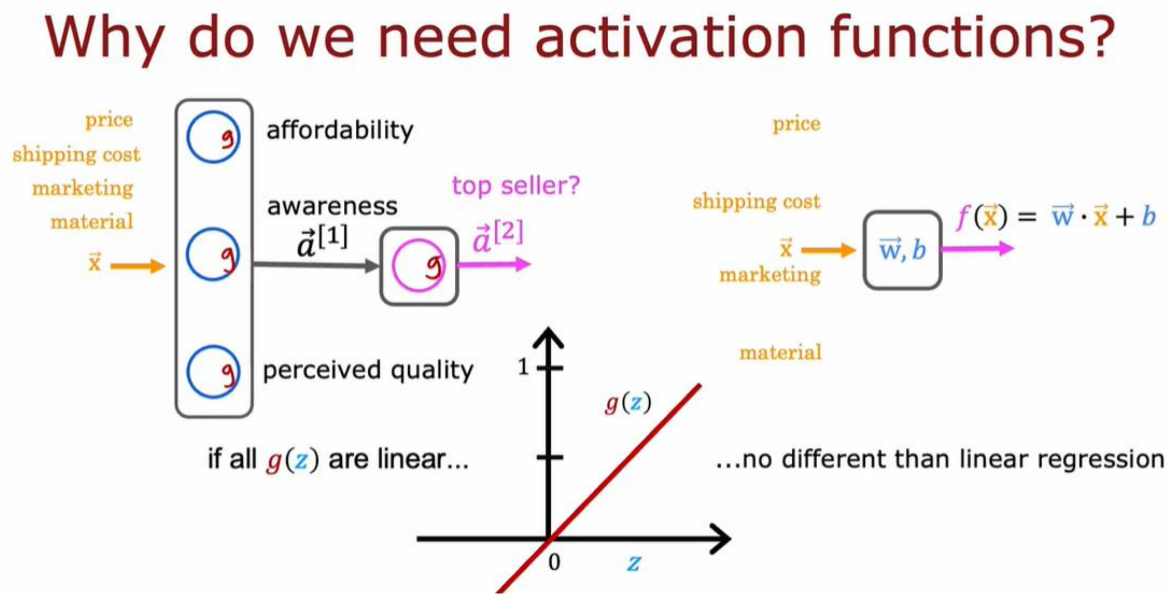
2. Choosing activation function

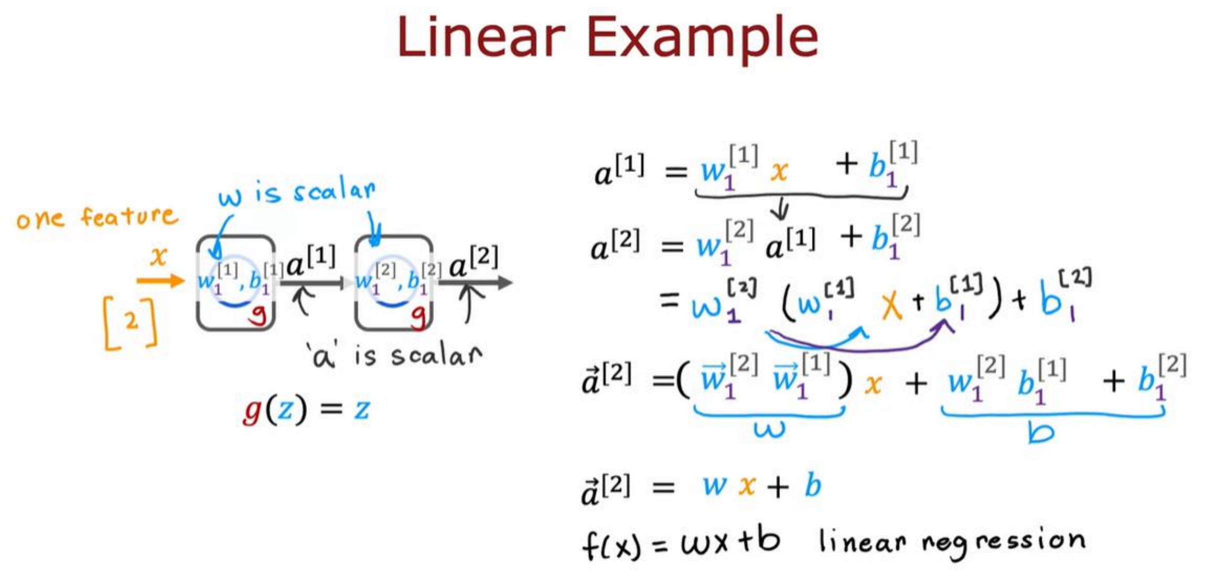


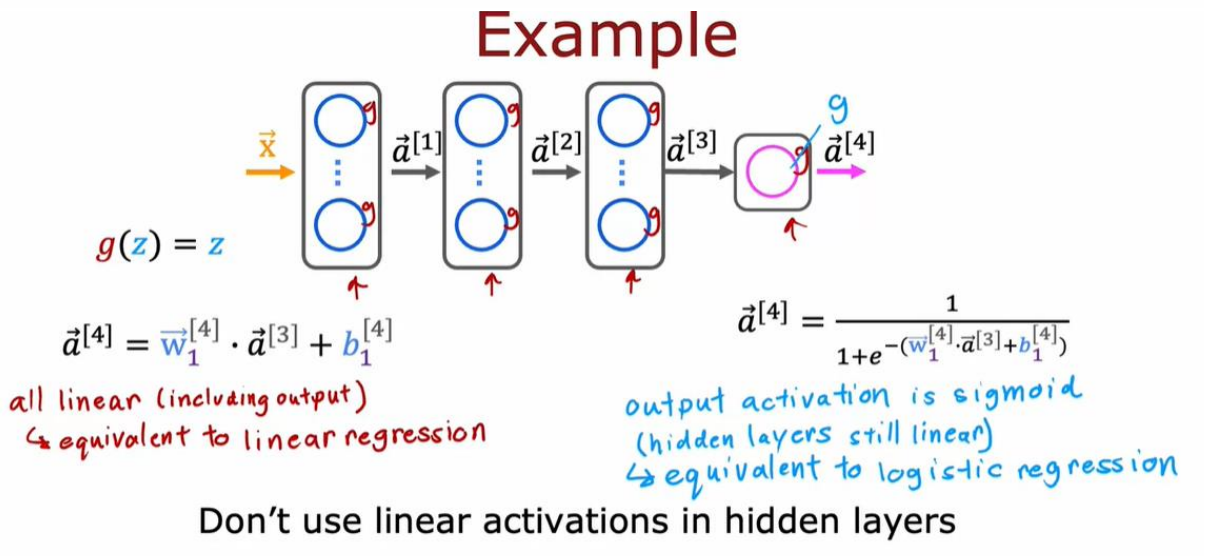




3. Why do we need activation function

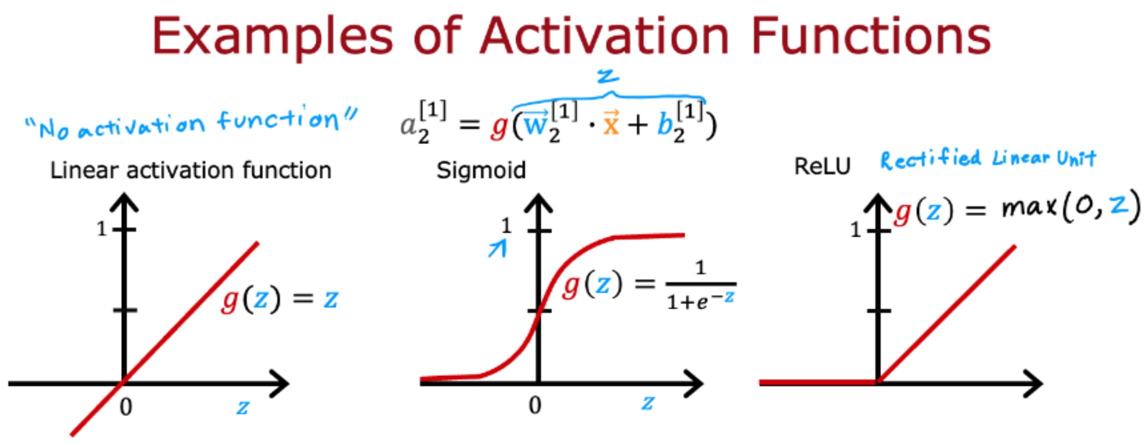






4. Practice quiz

1.



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

- (selected) 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.

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

- [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.

- [ ] Sigmoid

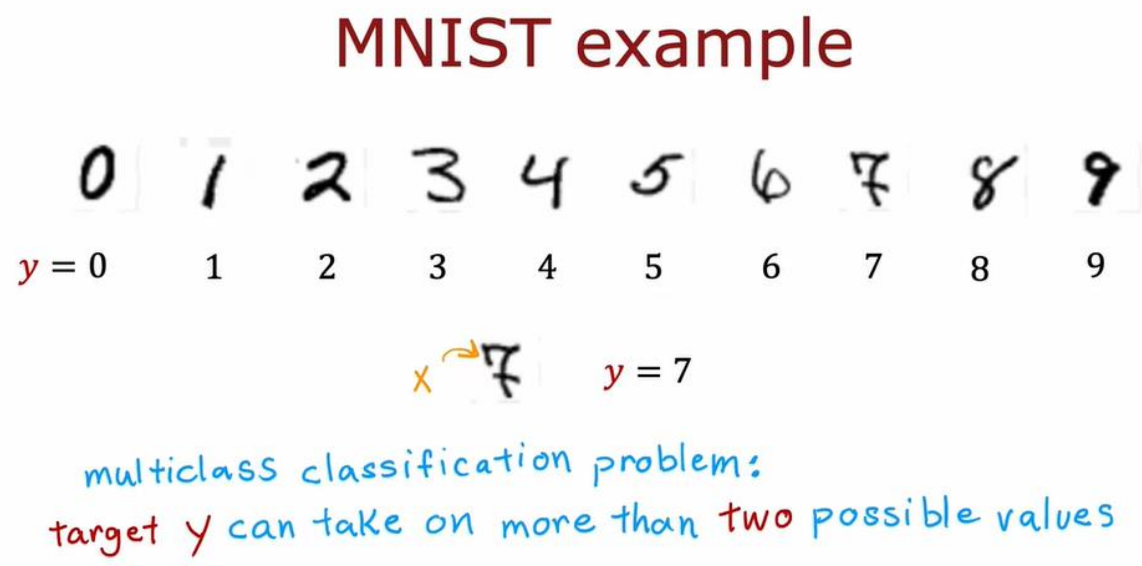
3. True/False? A neural network with many layers but no activation function (in the hidden layers) is not effective; that's why we should instead use the linear activation function in every hidden layer.

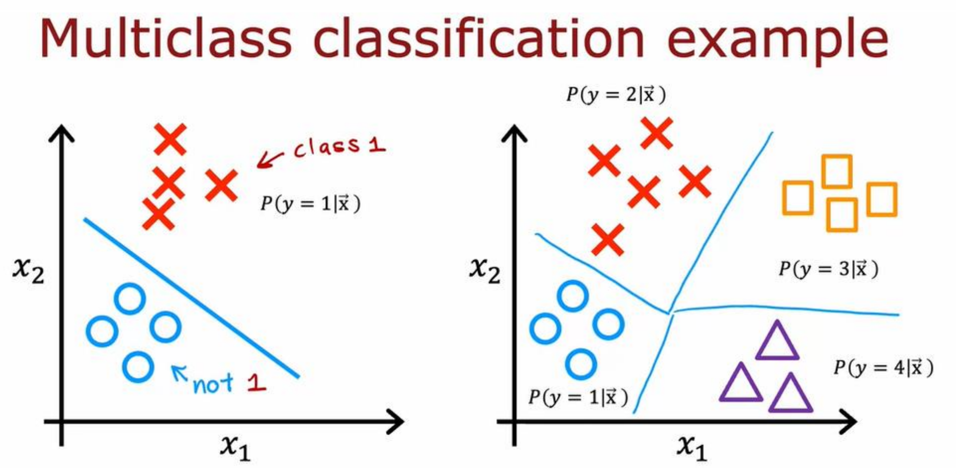
- True
- (selected) False

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".

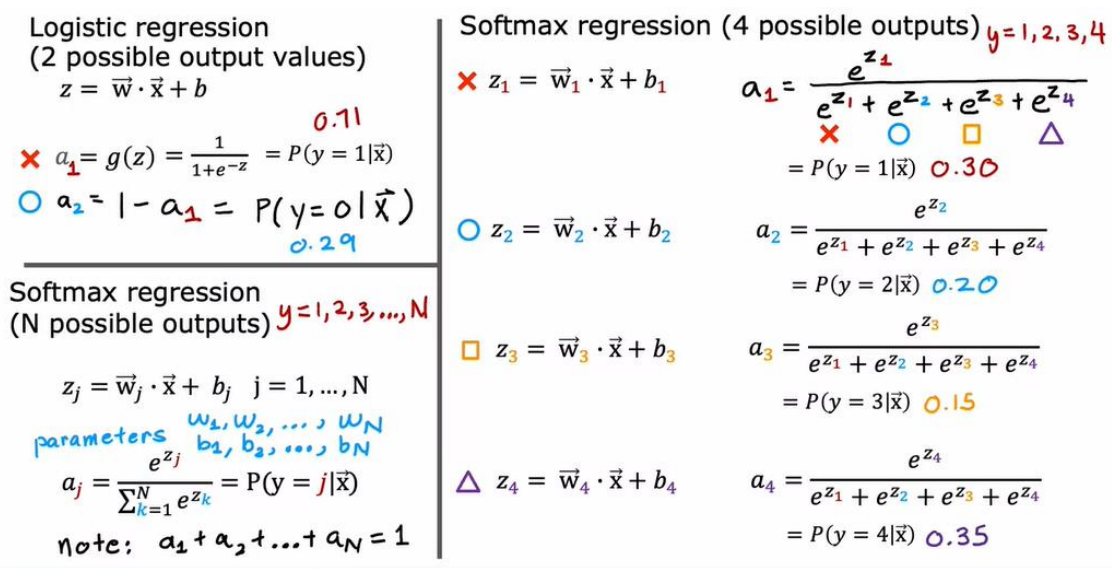
Multiclass classification

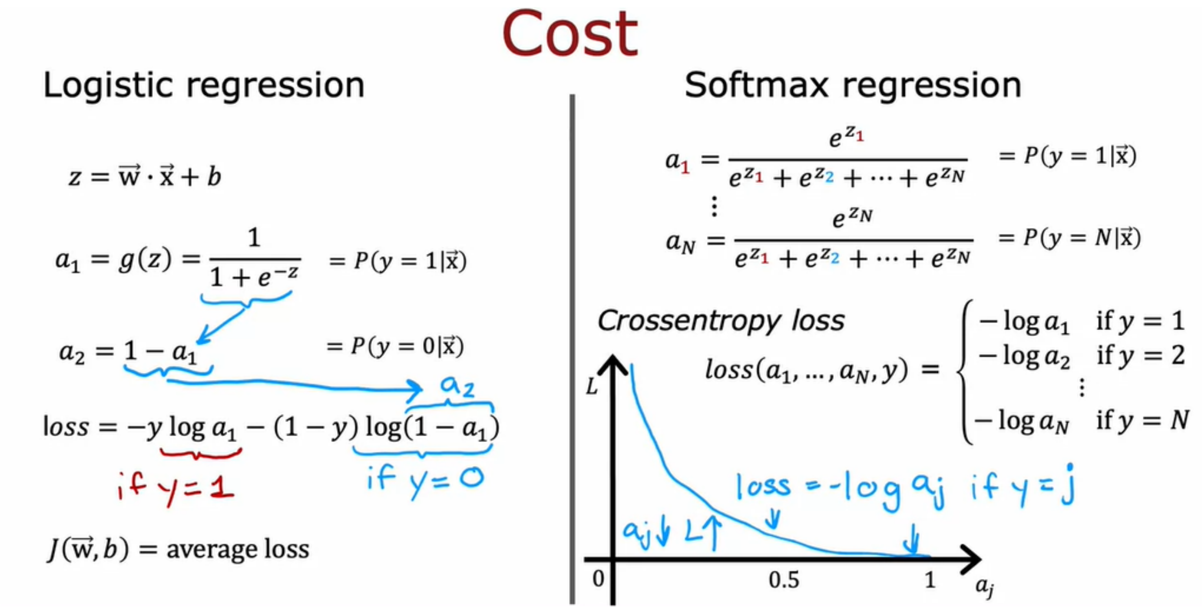
1. Multiclass



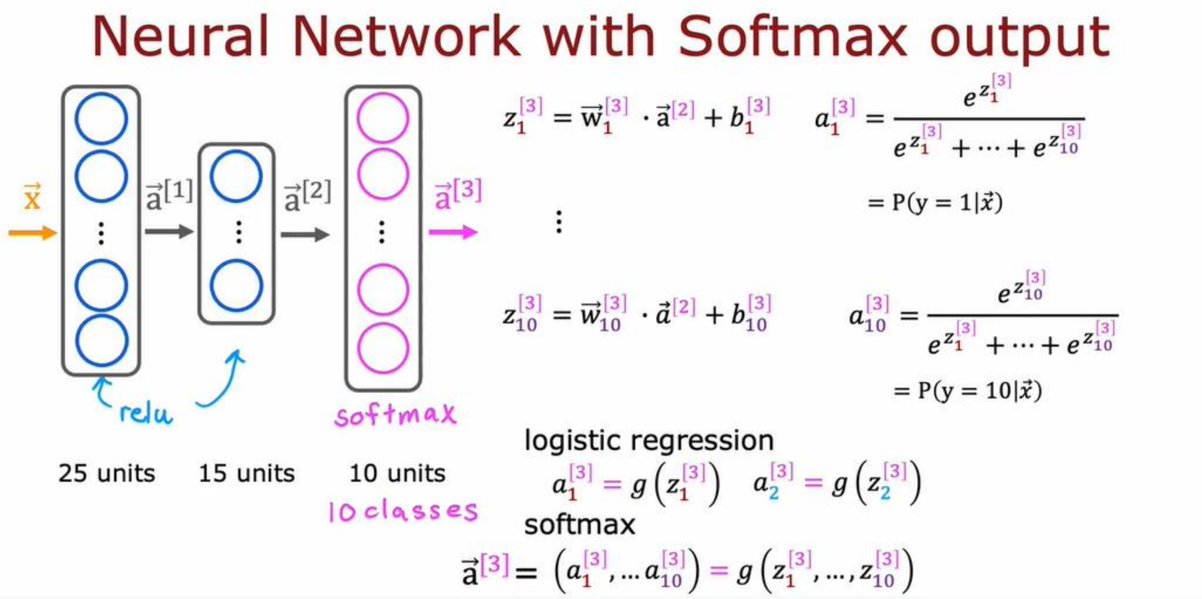


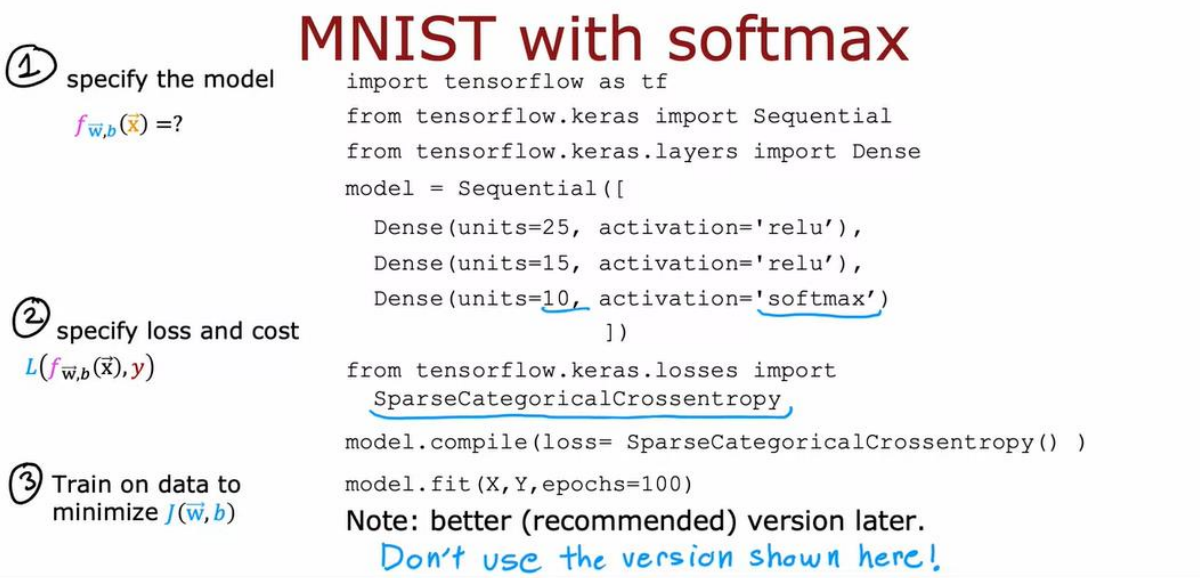
2. Softmax



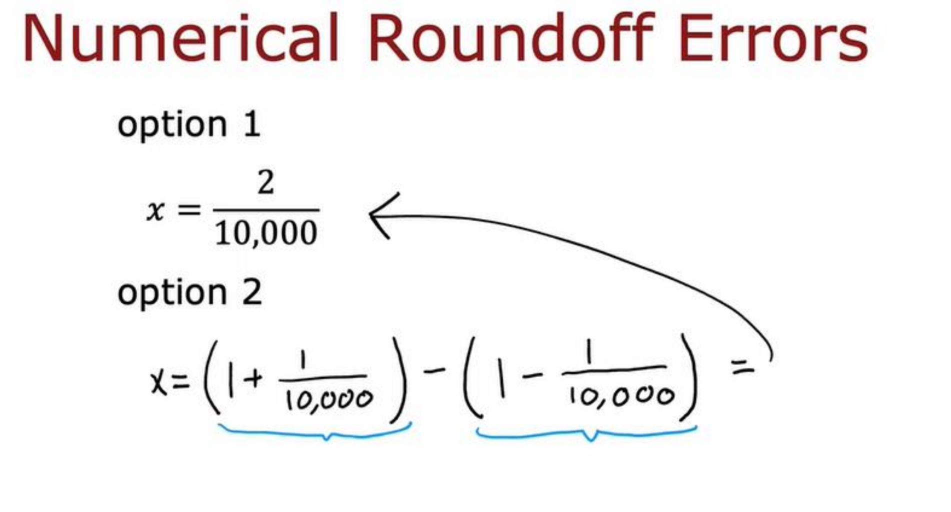


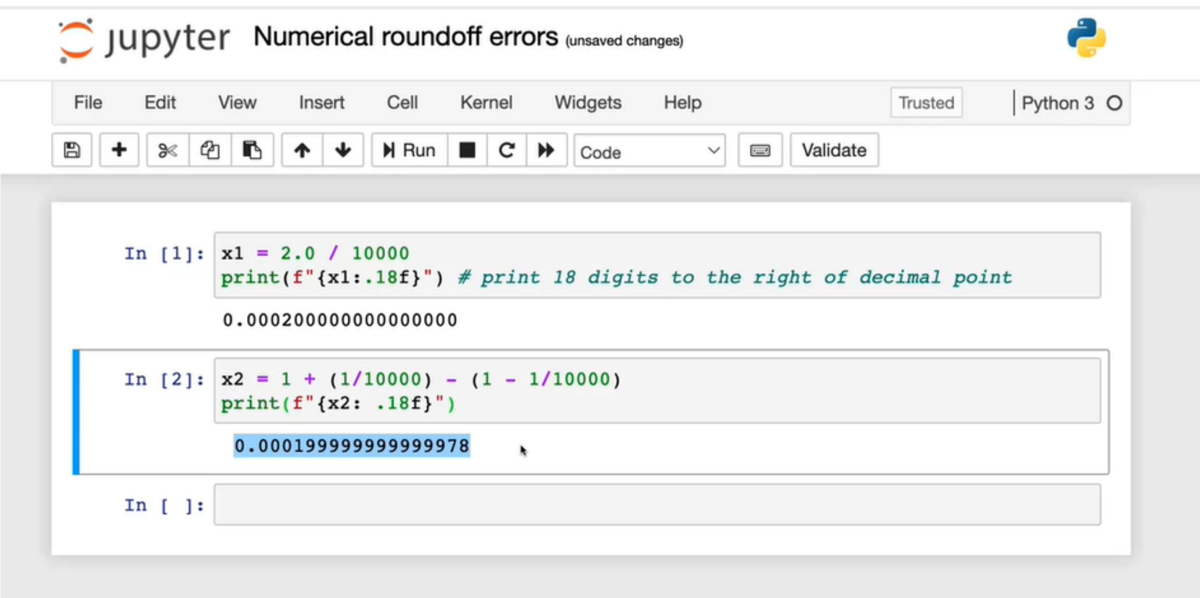
3. Neural network with softmax output





4. Improved implementation of softmax





Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

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

Logistic regression:

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

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

'linear'

Original loss

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

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

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

More accurate loss (in code)

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

logit: z

More numerically accurate implementation of softmax

Softmax regression

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

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

```
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 $a_1 \dots a_{10}$

is $z_1 \dots z_{10}$

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)
```

5. Classification with multiple outputs

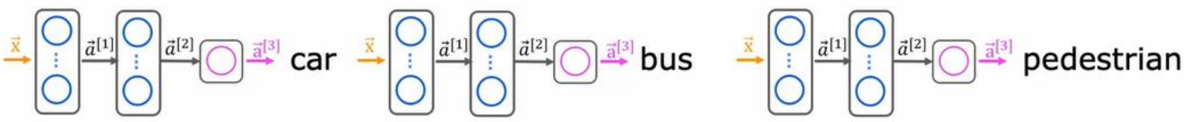
Multi-label Classification

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

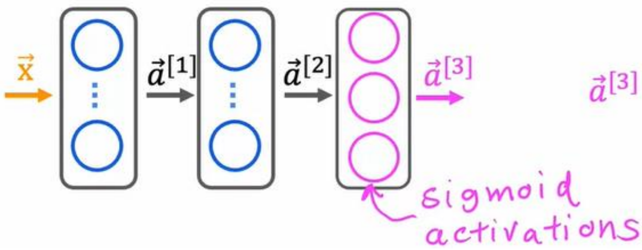
Is there a car?	yes	no	yes
Is there a bus?	no	no	yes
Is there a pedestrian	yes	yes	no

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

Multi-label Classification



Alternatively, train one neural network with three outputs



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

6. Practice quiz

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 ….

- (selected) 1
- More than 1
- It will vary, depending on the input x.
- Less than 1

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.

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.]

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

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

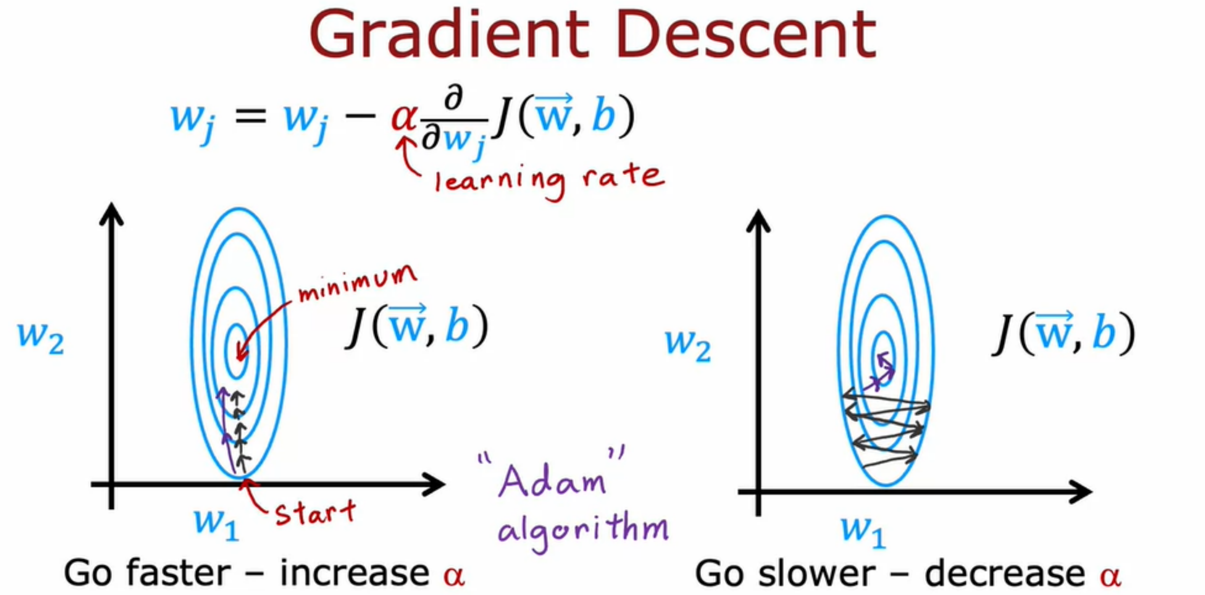
For multiclass classification, the recommended way to implement softmax regression is to set from_logits=True in the loss function, and also to define the model's output layer with…

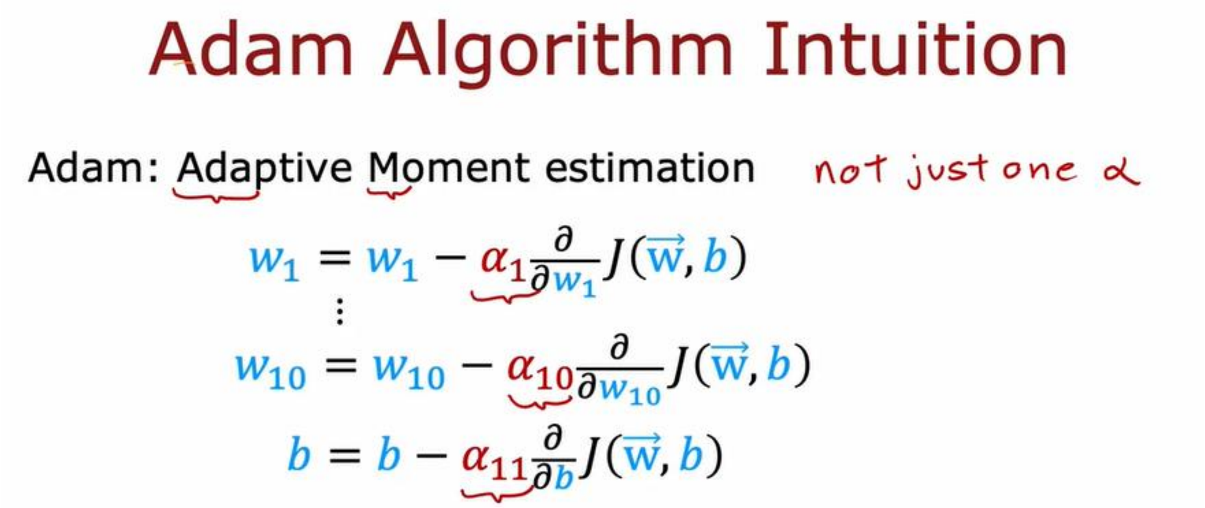
- (selected) a 'linear' activation
- a 'softmax' activation

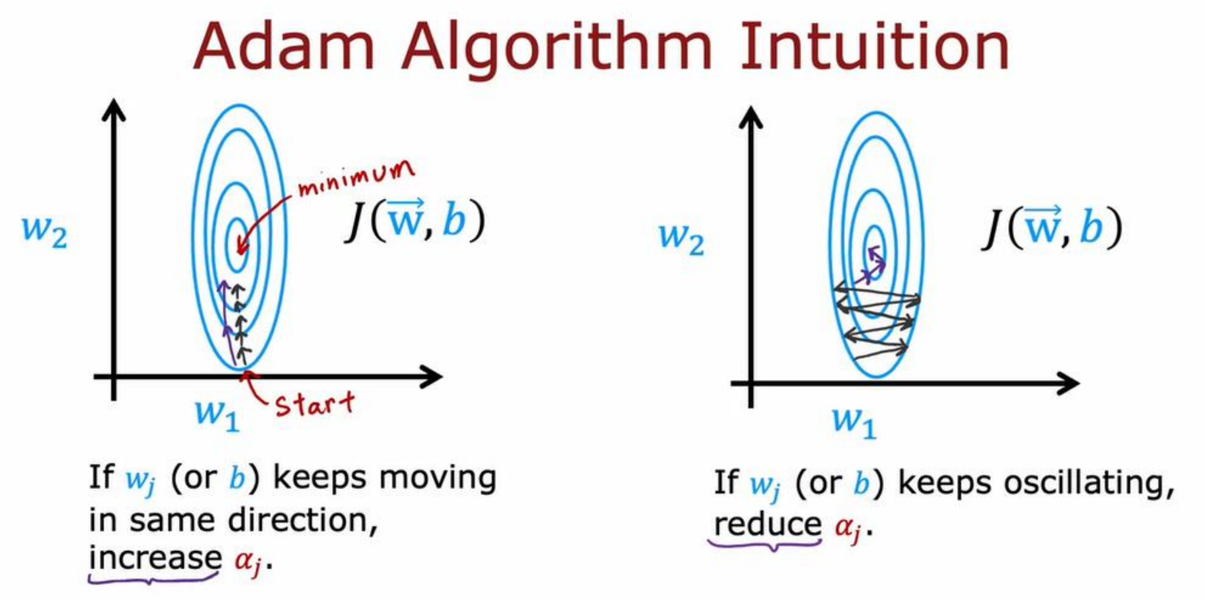
Correct
Yes! Set the output as linear, because the loss function handles the calculation of the softmax with a more numerically stable method.

Additional Neural Network Concepts

1. Advance optimization







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

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

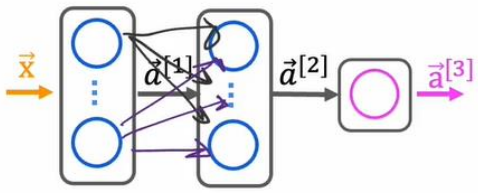
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

2. Additional layer types

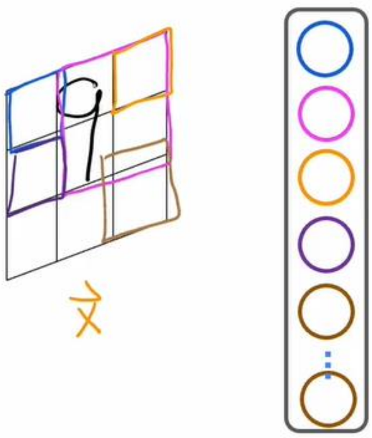
Dense Layer



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

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

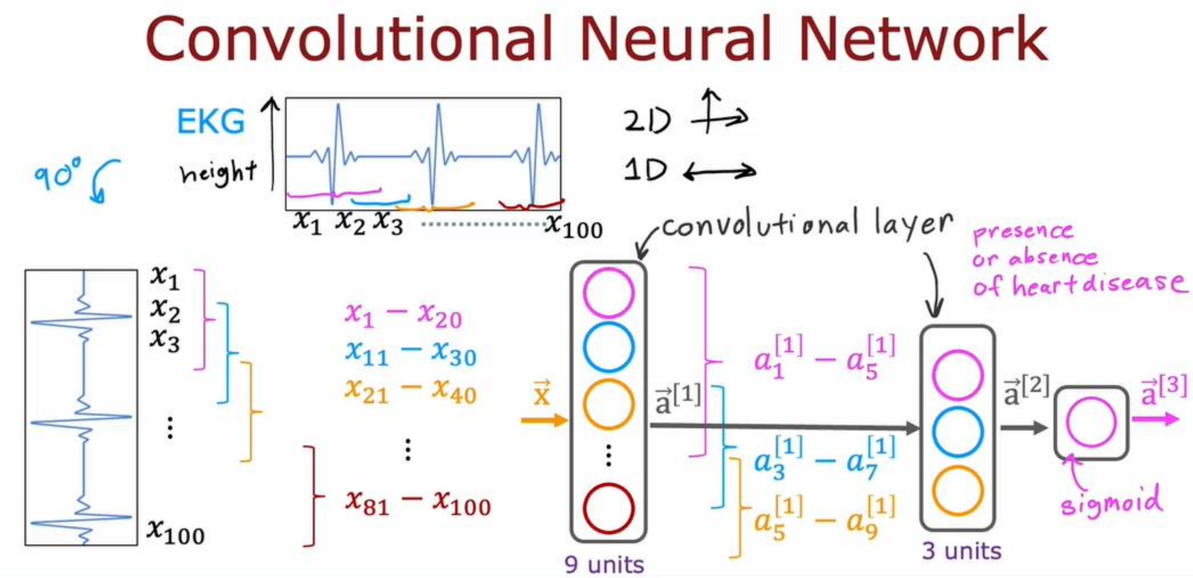
Convolutional Layer



Each neuron only looks at part of the previous layer's outputs.

Why?

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



3. Practice quiz

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 Adam optimizer works only with Softmax outputs. So if a neural network has a Softmax output layer, TensorFlow will automatically pick the Adam optimizer.
- ○ 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.
- ○ The call to model.compile() uses the Adam optimizer by default
- ◉ When calling model.compile, set optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3).

Correct
Correct. Set the optimizer to Adam.

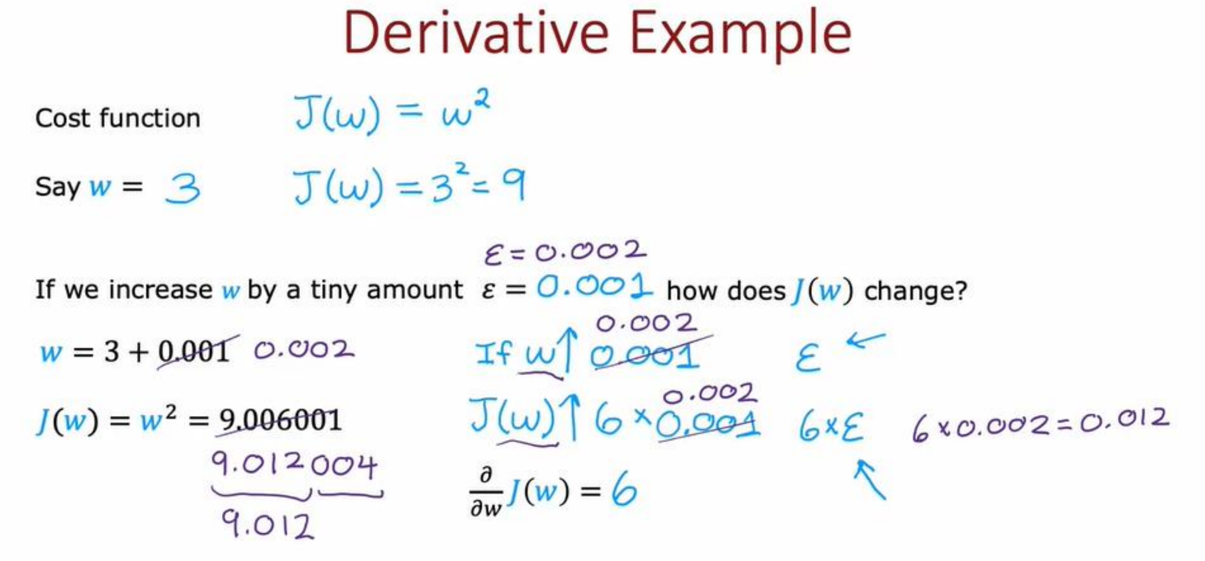
The lecture covered a different layer type where each single neuron of the layer does not look at all the values of the input vector that is fed into that layer. What is this name of the layer type discussed in lecture?

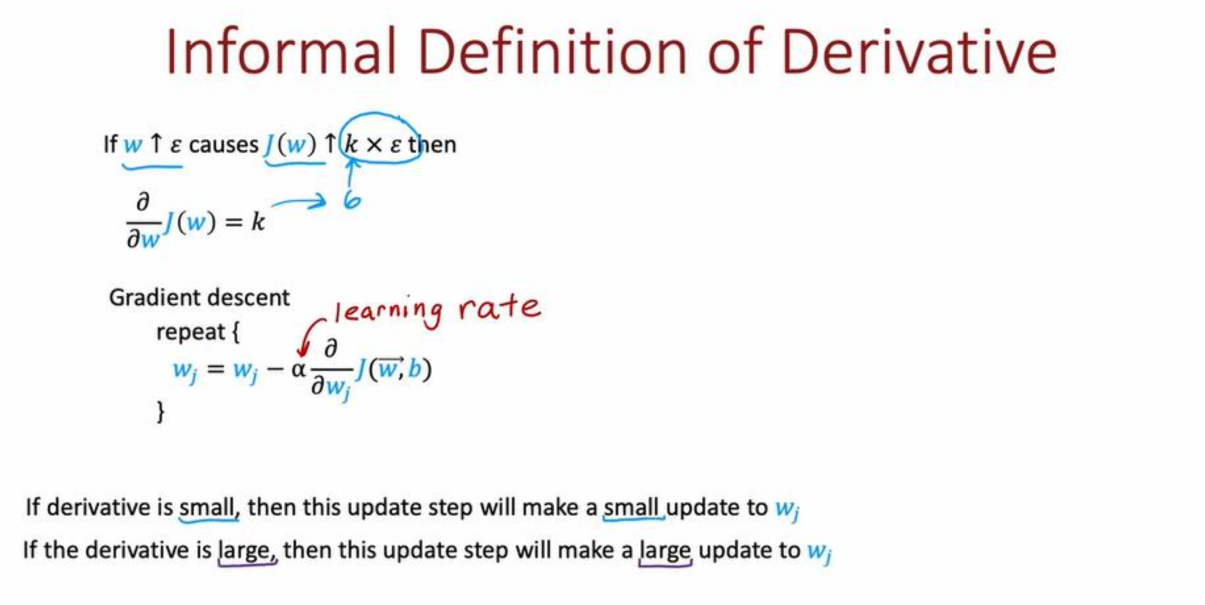
- ○ Image layer
- ○ A fully connected layer
- ◉ convolutional layer
- ○ 1D layer or 2D layer (depending on the input dimension)

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

Back Propagation

1. What is a derivative?





More Derivative Examples

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

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

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

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

$J(w) = w^2$

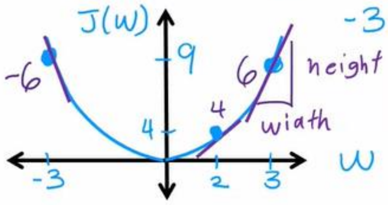
$\downarrow 0.006$

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

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

$-3 + 0.001$

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



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

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 [ ]:
```

Even More Derivative Examples

$w = 2$

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

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

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

$J(w) \uparrow 12 \times \varepsilon$

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

$J(w) \uparrow 1 \times \varepsilon$

-0.25×0.001

$0.5 - 0.00025$

$J(w) = \frac{1}{w} = \frac{1}{2} = 0.5$ $\quad \frac{\partial}{\partial w} J(w) = \frac{-1}{w^2} = \frac{-1}{4}$ $\quad w \uparrow \varepsilon$ $\quad J(w) = 0.49975$

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

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

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

A note on derivative notation

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

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

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

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

"partial derivative"

notation used in these courses

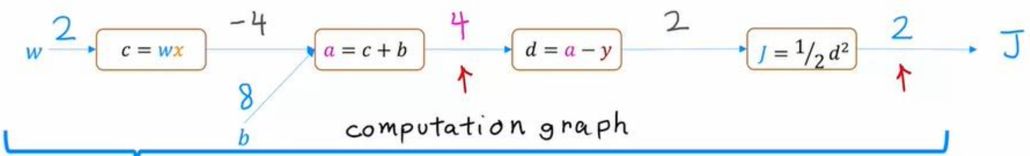
2. Computation graph

Small Neural Network Example

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

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

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



forward prop

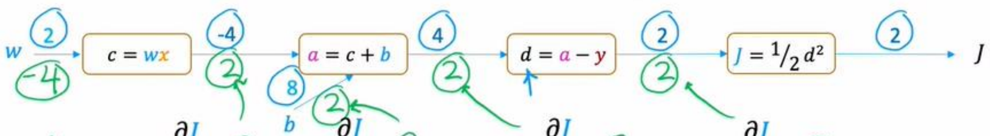
how do we find the derivatives of J? $\frac{\partial J}{\partial w} \quad \frac{\partial J}{\partial b}$

Computing the Derivatives

Forward prop →

← Back prop

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



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

$w=2.001 \quad c=-4.002$

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

$c \uparrow -2\times 0.001$

$J \uparrow -4\times 0.001$

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

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

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

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

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

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

$a=4.001 \quad d=2.001$

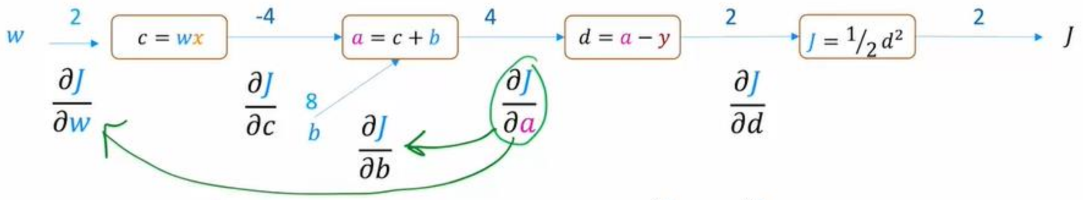
$J \uparrow 0.002$

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

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

chain rule (calculus)

Backprop is an efficient way to compute derivatives



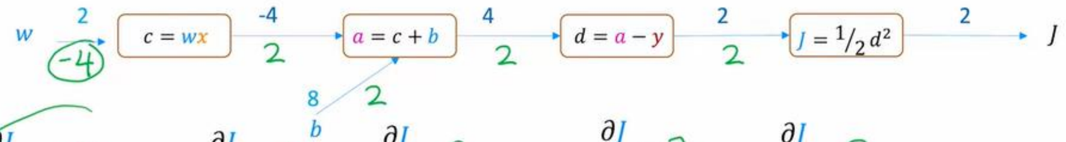
Compute $\frac{\partial J}{\partial a}$ once and use it to compute both $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$.

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

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

J ↓ 4 × 0.001
J ↑ −4 × 0.001

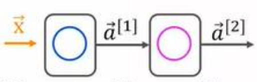
$\frac{\partial J}{\partial w} = -4$

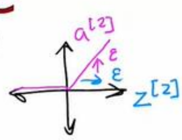
3. Larger neural network example

Neural Network Example

$x = 1$ $y = 5$ $w^{[1]} = 2, b^{[1]} = 0$ ReLU activation

$w^{[2]} = 3, b^{[2]} = 1$ $g(z) = \max(0, z)$

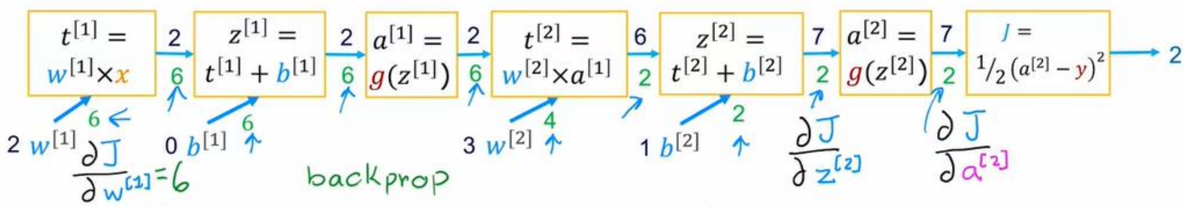




$a^{[1]} = g(w^{[1]}x + b^{[1]}) = \overbrace{w^{[1]}x + b^{[1]}}^{z^{[1]}} = 2 \times 1 + 0 = 2$

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

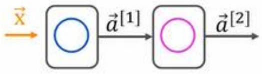
$J(w, b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(7 - 5)^2 = 2$



if $w^{[1]}$ ↑ ε, then J ↑ 6 × ε let's verify this!

Neural Network Example

$x = 1$ $y = 5$ $w^{[1]} = 2, b^{[1]} = 0$ ReLU activation



$w^{[2]} = 3,\ b^{[2]} = 1$ $g(z) = \max(0, z)$

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

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

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

$$\frac{(2.003)^2}{2}$$

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

$J \uparrow 6 \times 0.001$

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

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

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

N nodes

P parameters

$w_1, b_1, w_2, b_2 \dots$

inefficient way

$N \times P$

efficient way (backprop)

$N + P$