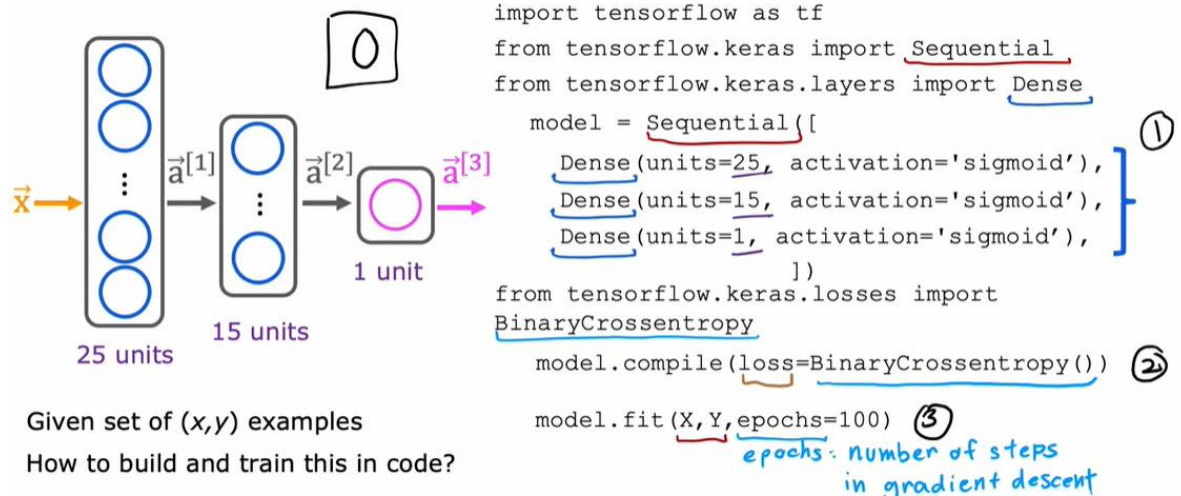


Neural Network Training

1. TensorFlow implementation

Train a Neural Network in TensorFlow



2. Training details

Model Training Steps TensorFlow

	logistic regression	neural network
① specify how to compute output given input x and parameters w, b (define model)	$z = \text{np.dot}(w, x) + b$ $f_x = 1 / (1 + \text{np.exp}(-z))$	<code>model = Sequential([Dense(...), Dense(...), Dense(...)])</code>
② specify loss and cost	logistic loss $\text{loss} = -y * \text{np.log}(f_x) - (1-y) * \text{np.log}(1-f_x)$	binary cross entropy <code>model.compile(loss=BinaryCrossentropy())</code>
③ Train on data to minimize $J(\vec{w}, b)$	$w = w - \text{alpha} * \text{dj_dw}$ $b = b - \text{alpha} * \text{dj_db}$	<code>model.fit(X, y, epochs=100)</code>

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

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

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

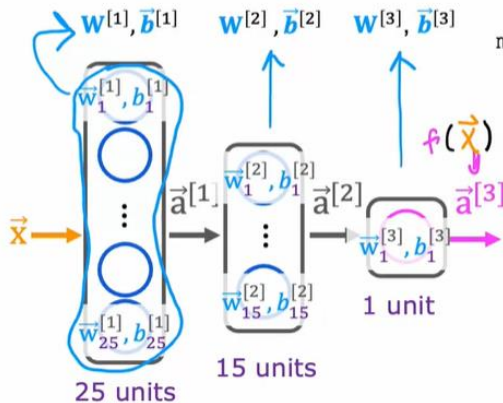
③

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

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

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

$\mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]}$ $\mathbf{B}^{[1]}, \mathbf{B}^{[2]}, \mathbf{B}^{[3]}$ $f_{\mathbf{W}, \mathbf{B}}(\vec{x})$

model.compile(loss= BinaryCrossentropy())

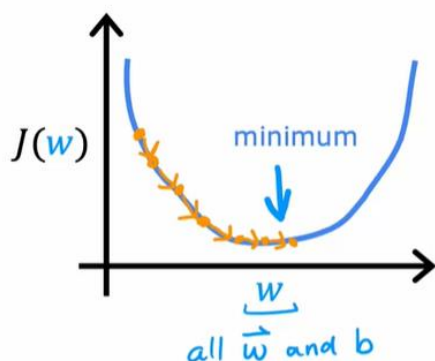
regression (predicting numbers and not categories) mean squared error

model.compile(loss= MeanSquaredError())

from tensorflow.keras.losses import BinaryCrossentropy **K** Keras

from tensorflow.keras.losses import MeanSquaredError

3. Gradient descent



repeat {

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

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

} Compute derivatives for gradient descent using "backpropagation"

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

Neural network libraries

Use code libraries instead of coding "from scratch"



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

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())`

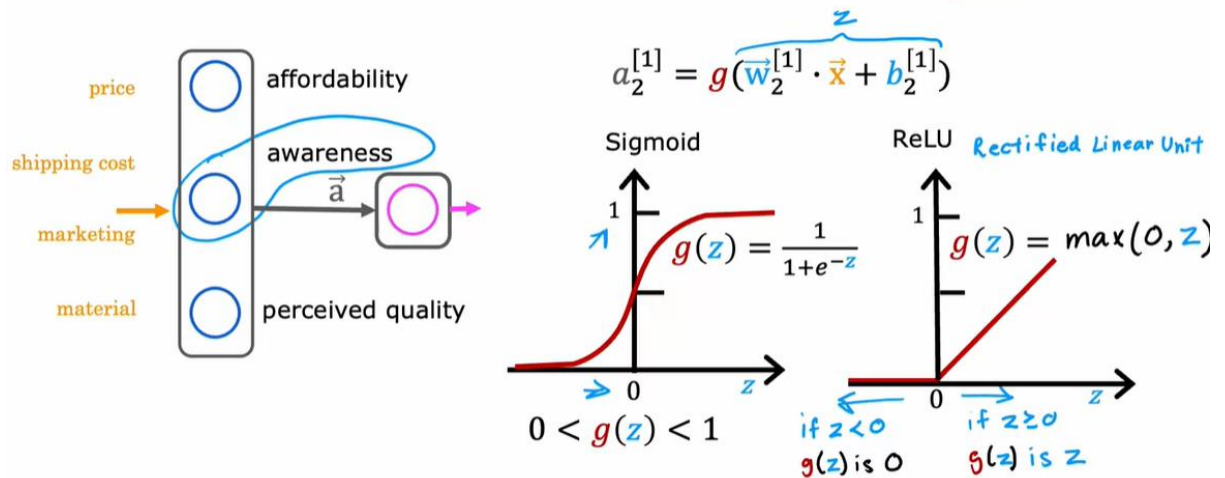
✓ **Correct**

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

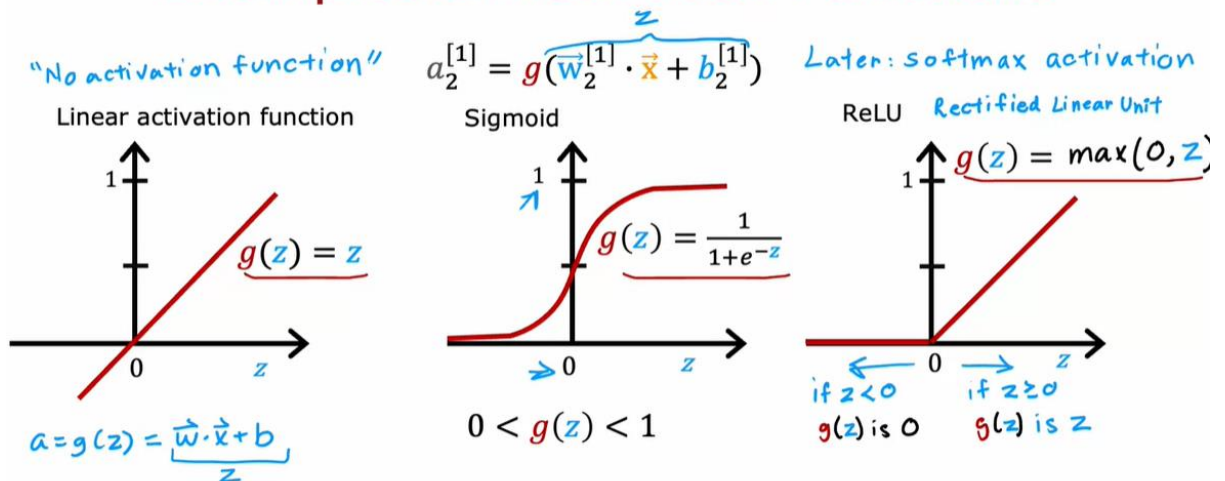
Activation Function

1. Alternatives to the sigmoid activation

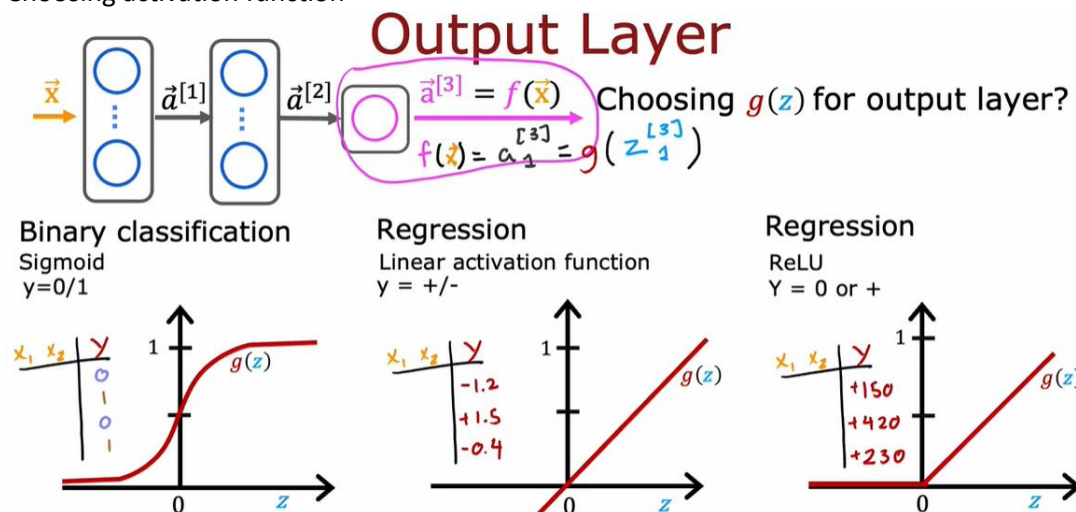
Demand Prediction Example

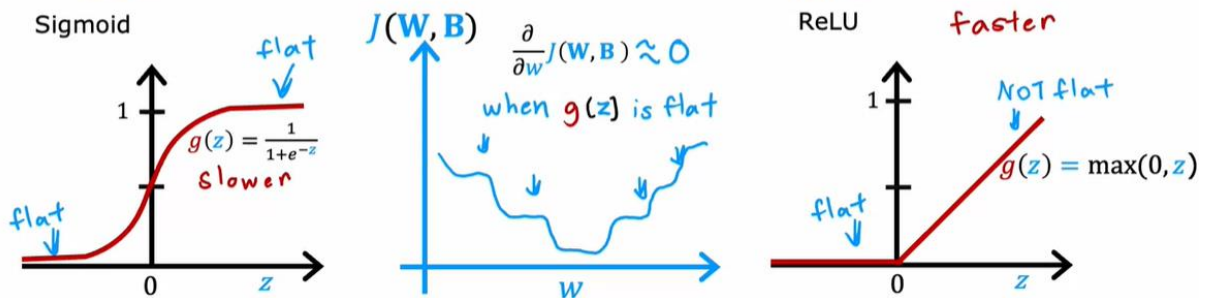
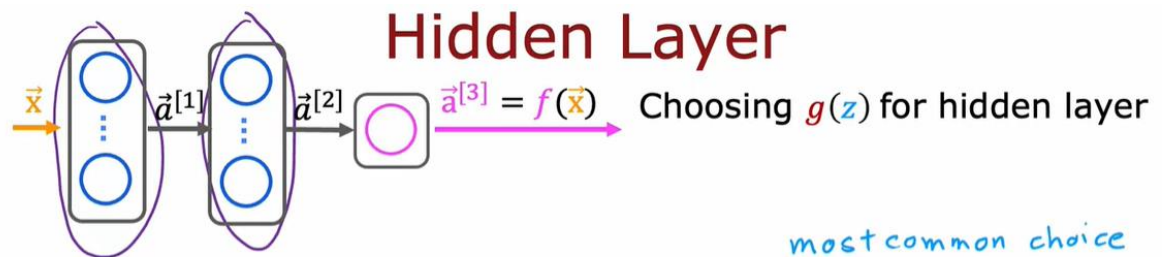


Examples of Activation Functions

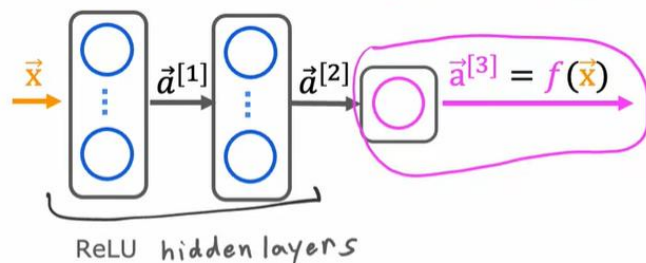


2. Choosing activation function





Choosing Activation Summary



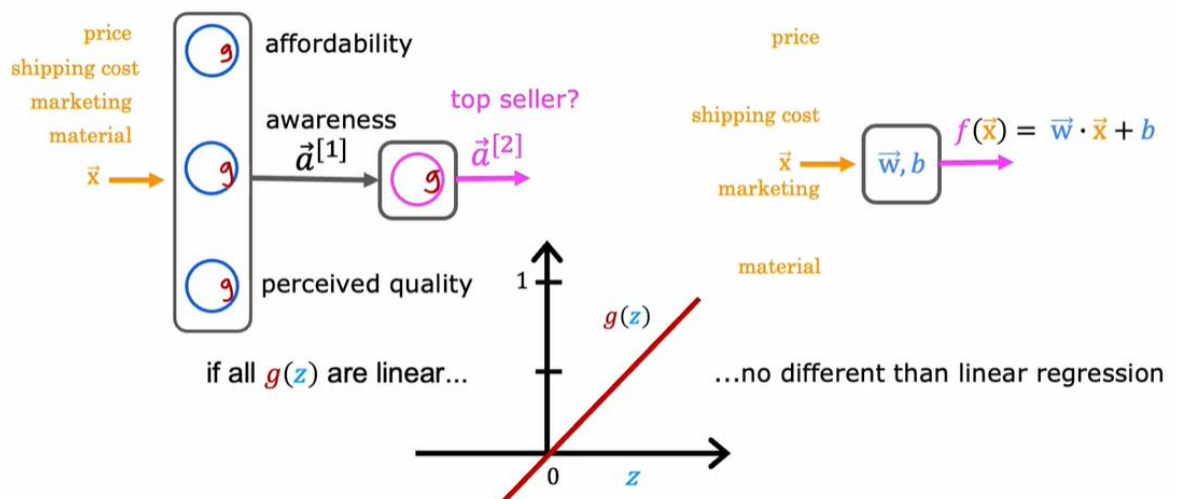
binary classification
 activation='sigmoid'
 regression y negative/
 activation='linear' positive
 regression $y \geq 0$
 activation='relu'

```
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
])
```

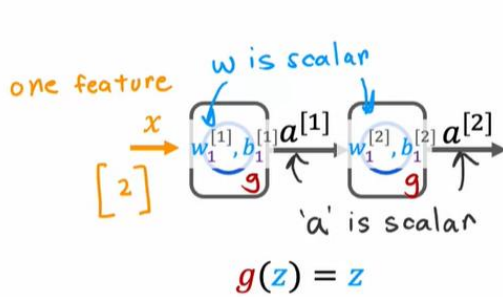
or 'linear'
 or 'relu'

3. Why do we need activation function

Why do we need activation functions?



Linear Example



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

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

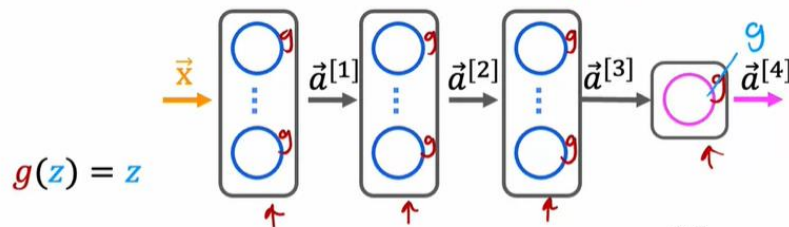
$$= w_1^{[2]} (w_1^{[1]} x + b_1^{[1]}) + b_1^{[2]}$$

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

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

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

Example



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

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

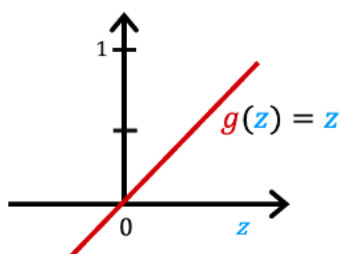
4. Practice quiz

1.

Examples of Activation Functions

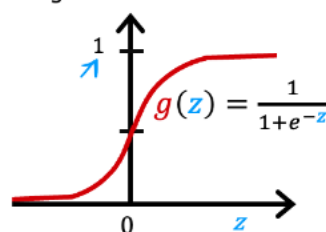
"No activation function"

Linear activation function

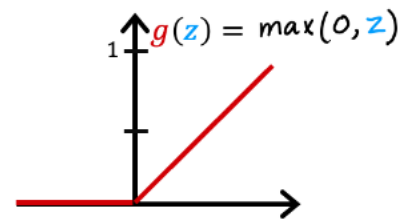


$$a_2^{[1]} = g(\underbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}_z)$$

Sigmoid



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.

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

☒ 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).

☒ 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
- ☒ False

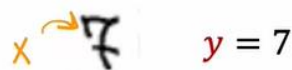
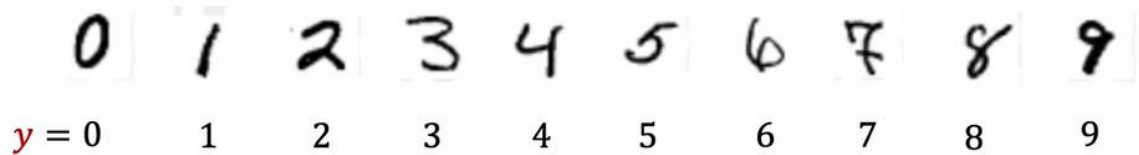
✓ **Correct**

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

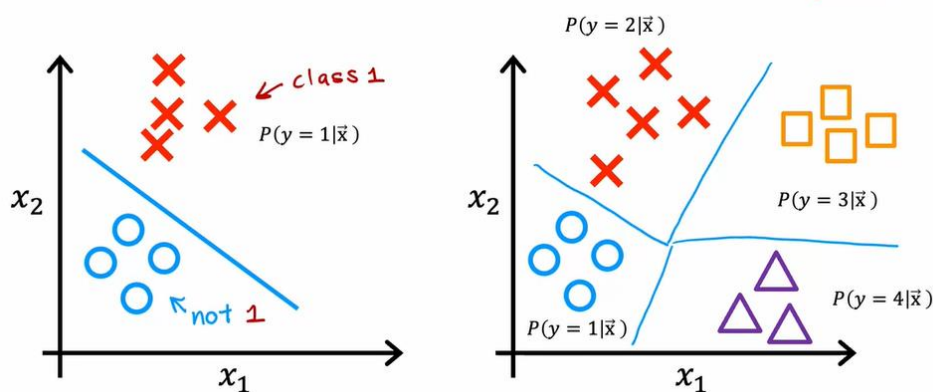
1. Multiclass

MNIST example



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

Multiclass classification example



2. Softmax

Logistic regression
(2 possible output values)
 $z = \vec{w} \cdot \vec{x} + b$

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

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

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

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

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

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

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

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

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

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

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

$$\text{blue } \circ z_2 = \vec{w}_2 \cdot \vec{x} + b_2$$

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

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

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

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

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

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

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

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

Cost

Logistic regression

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

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

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

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

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

Softmax regression

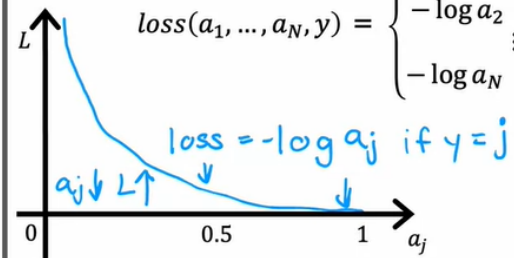
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = 1|\vec{x})$$

$$\vdots$$

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

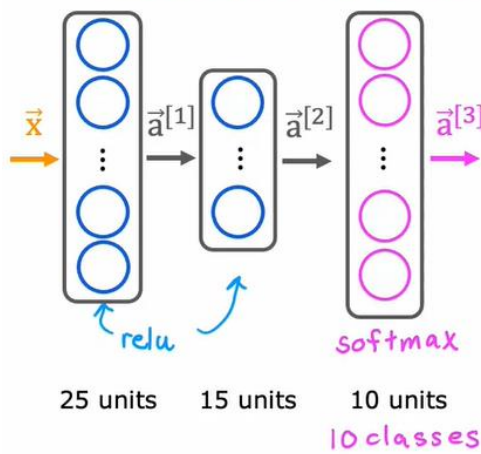
Crossentropy loss

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



3. Neural network with softmax output

Neural Network with Softmax output



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

$$\vdots$$

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

logistic regression

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

softmax

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

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')
])
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy() )
model.fit(X,Y,epochs=100)
```

② specify loss and cost

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

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

Note: better (recommended) version later.

Don't use the version shown here!

4. 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
0.00020000000000000000
```

```
In [2]: x2 = 1 + (1/10000) - (1 - 1/10000)
print(f"{x2: .18f}")
0.00019999999999999978
```

In []:

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

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

Logistic regression:

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

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

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

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

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

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

More Accurate

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

```
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)

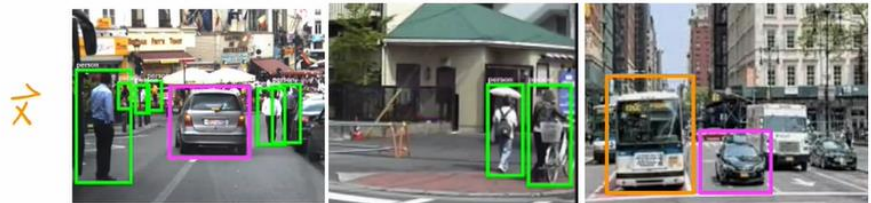
```

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

5. Classification with multiple outputs

Multi-label Classification



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

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

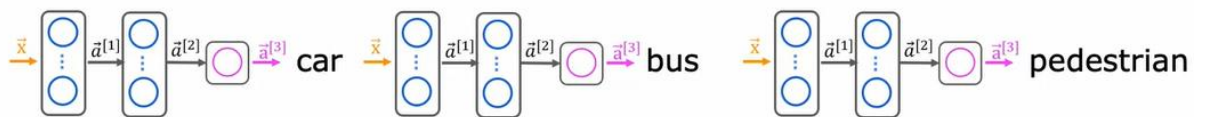
no
no
yes

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

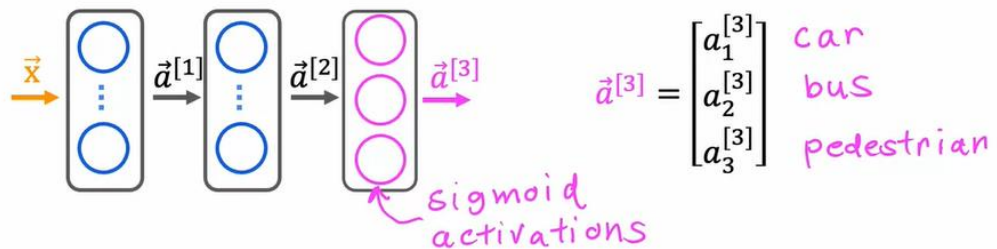
yes
yes
no

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

Multi-label Classification



Alternatively, train one neural network with three outputs



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

- ☒ 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.]

- ☒ $-\log(a_3)$
- ☐ $z_3/(z_1+z_2+z_3+z_4)$
- ☐ 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...

- ☒ a 'linear' activation
- ☐ a 'softmax' activation

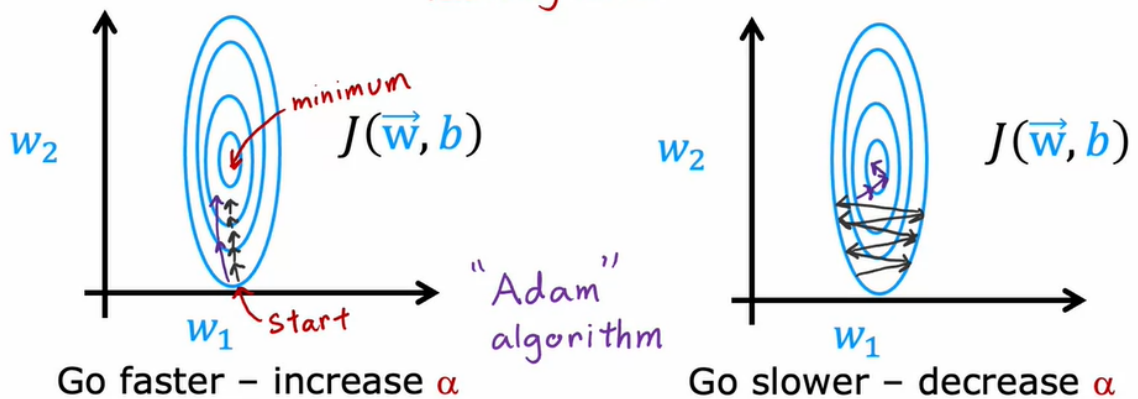
✓ **Correct**

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

1. Advance optimization

Gradient Descent

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

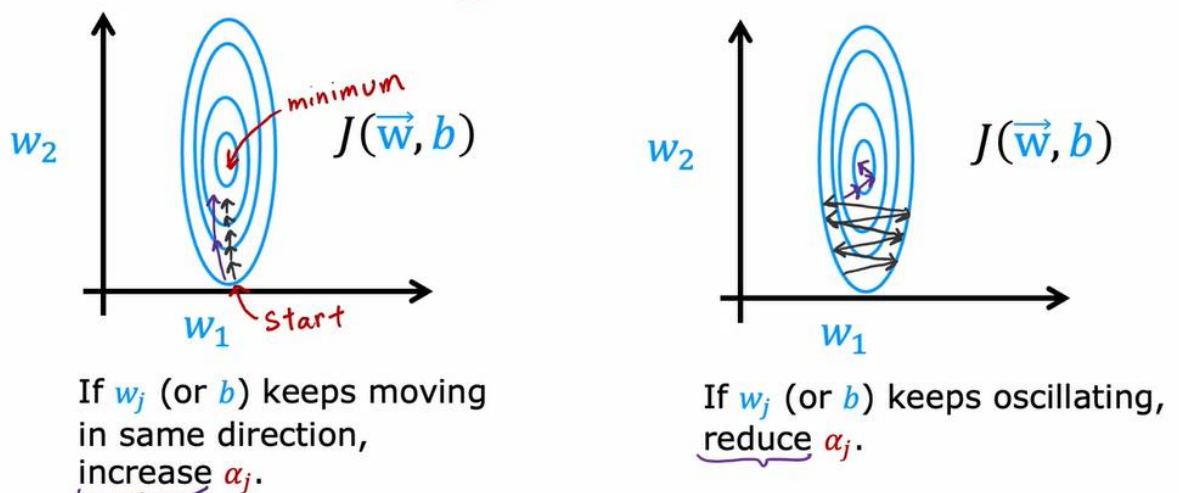


Adam Algorithm Intuition

Adam: Adaptive Moment estimation *not just one α*

$$\begin{aligned} w_1 &= w_1 - \underbrace{\alpha_1}_{\text{adaptive}} \frac{\partial}{\partial w_1} J(\vec{w}, b) \\ &\vdots \\ w_{10} &= w_{10} - \underbrace{\alpha_{10}}_{\text{adaptive}} \frac{\partial}{\partial w_{10}} J(\vec{w}, b) \\ b &= b - \underbrace{\alpha_{11}}_{\text{adaptive}} \frac{\partial}{\partial b} J(\vec{w}, b) \end{aligned}$$

Adam Algorithm Intuition



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

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

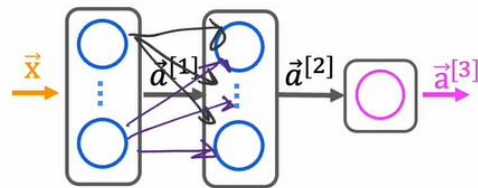
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

2. Additional layer types

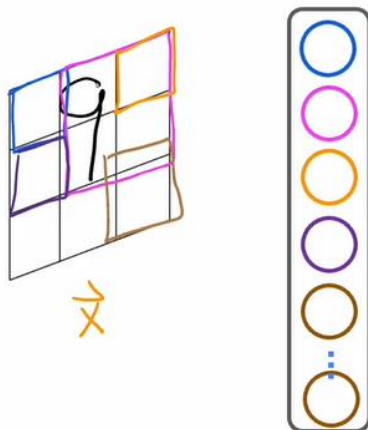
Dense Layer



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

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

Convolutional Layer

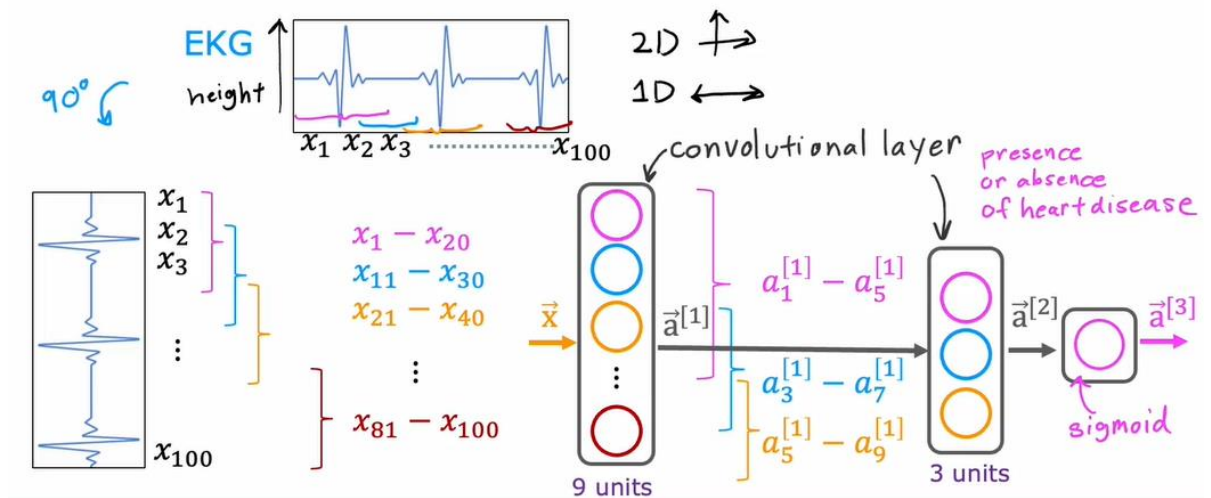


Each neuron only looks at part of the previous layer's outputs.

Why?

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

Convolutional Neural Network



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?

Derivative Example

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

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

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

$$w = 3 + 0.001 \quad 0.002$$

$$J(w) = w^2 = 9.006001$$
$$\begin{array}{r} 9.012004 \\ \hline 9.012 \end{array}$$

$$\epsilon = 0.002$$

$$\epsilon = 0.001$$

$$\text{If } w \uparrow 0.001$$

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

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

$$6 \times \epsilon$$

$$6 \times 0.002 = 0.012$$

Informal Definition of Derivative

If $w \uparrow \epsilon$ causes $J(w) \uparrow k \times \epsilon$ 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

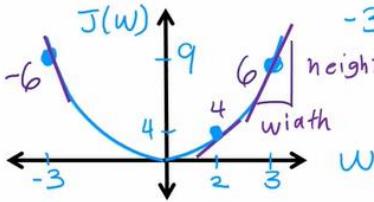
If the derivative is large, then this update step will make a large update to w_j

More Derivative Examples

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

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

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



$-3 + 0.001$

$\downarrow 0.006$

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

Using code to get derivatives (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3

Run Stop Restart
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

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

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

$J(w) = w = 2$ $\frac{\partial}{\partial w} J(w) = 1$ $w \uparrow \epsilon$ $J(w) = 2.001$
 $J(w) \uparrow 1 \times \epsilon$

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

$\downarrow 0.006$

-0.25×0.001

$0.5 - 0.00025$

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

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

A note on derivative notation

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

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

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

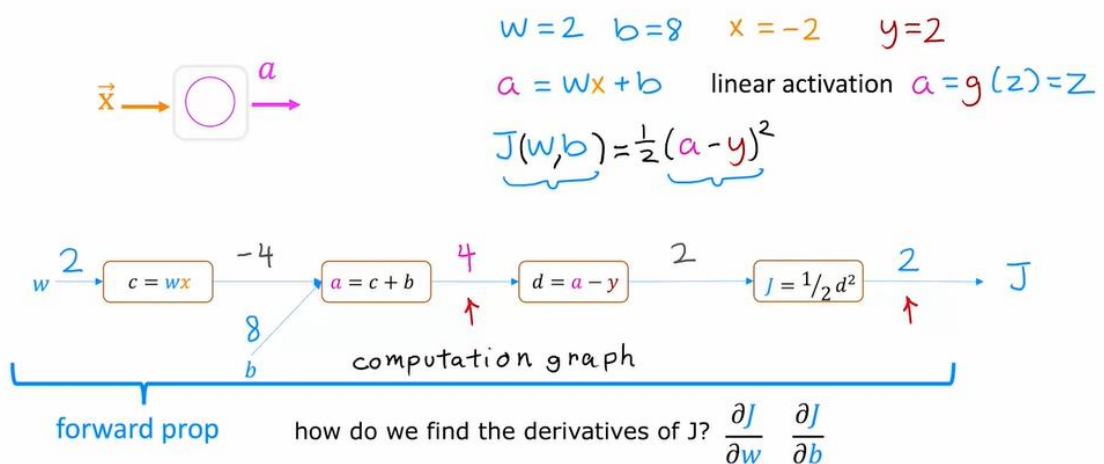
$$\partial \frac{\partial}{\partial w_i} J(w_1, w_2, \dots, w_n) \quad \frac{\partial J}{\partial w_i}$$

"partial derivative"

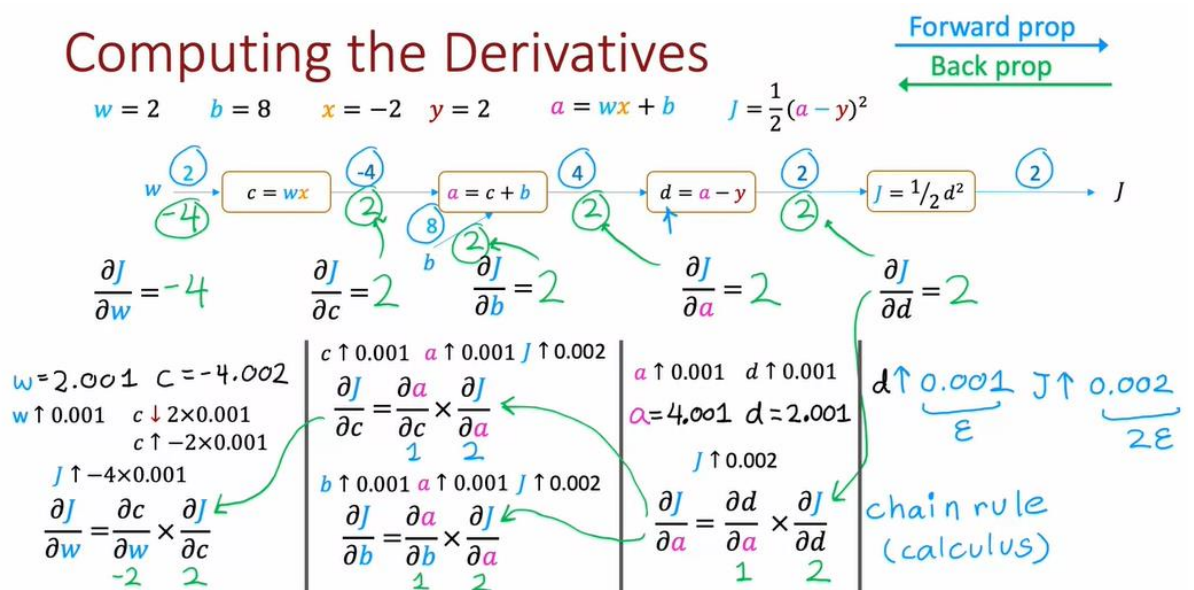
notation used
in these courses

2. Computation graph

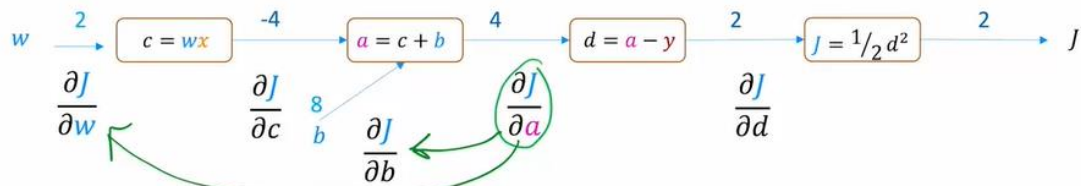
Small Neural Network Example



Computing the Derivatives



Backprop is an efficient way to compute derivatives



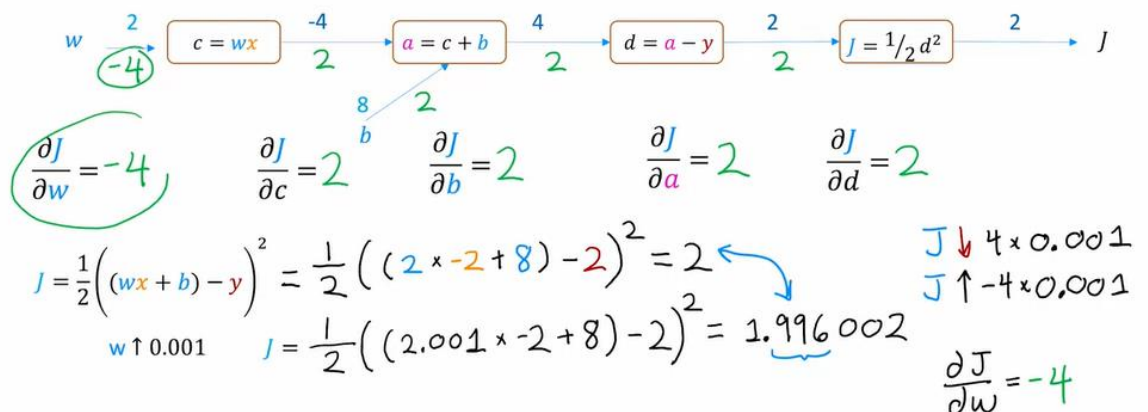
Compute $\frac{\partial J}{\partial a}$ once and use it to compute both $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$.

If N nodes and P parameters, compute derivatives in roughly $N + P$ steps rather than $N \times P$ steps.

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

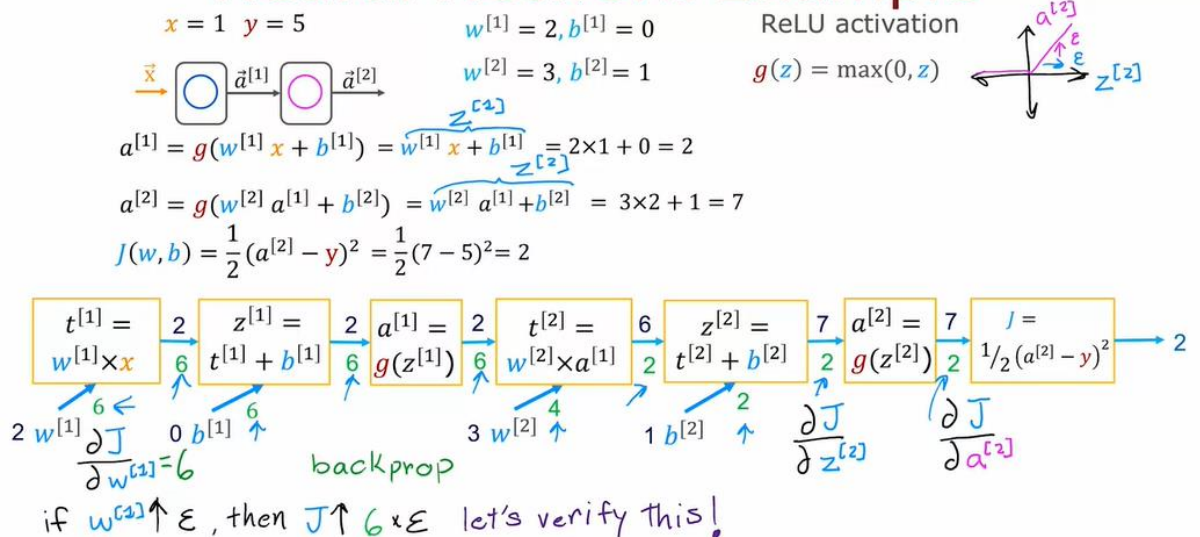
Computing the Derivatives

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



3. Larger neural network example

Neural Network Example



Neural Network Example

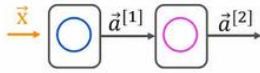
$$x = 1 \quad y = 5$$

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

ReLU activation

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

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



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

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

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

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

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

N nodes $\square \rightarrow \square \rightarrow \square$

P parameters
 $w_1, b_1, w_2, b_2 \dots$

inefficient way

$N \times P$

efficient way (backprop)

$N + P$