

Week 2

Neural Network Training

Model Training Steps

Activation Functions

Examples of Activation Functions

How to Choose Activation Function?

Output layer

Hidden Layer

Why do we need Activation Functions?

Multiclass Classification

Examples

Logistic vs SoftMax regression

Cost function for SoftMax Regression

Neural Network with SoftMax Output Layer

Numerical Roundoff Errors

Additional Neural Network Concepts

Advanced Optimization

Adaptive moment estimation (Adam) Algorithm

TF Implementation

Additional Layer Type

Backpropagation

Computation Graph

Computation graph for Forward Propagation

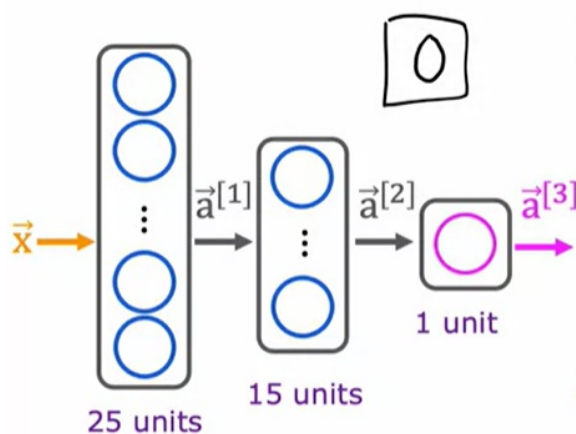
Computation graph for Backward Propagation

Why use Backward Prop?

Neural Network Training

- `Model.compile` → states the loss function

Train a Neural Network in TensorFlow



Given set of (x, y) examples

How to build and train this in code?

```
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

Model Training Steps

- Recap

Model Training Steps TensorFlow

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

- Step 1 → Create the model (The first slide)
- Step 2 → specify loss

2. Loss and cost functions

handwritten digit classification problem → binary classification

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

Compare prediction vs. target

logistic loss

also known as binary cross entropy

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

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

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

regression

(predicting numbers and not categories) mean squared error

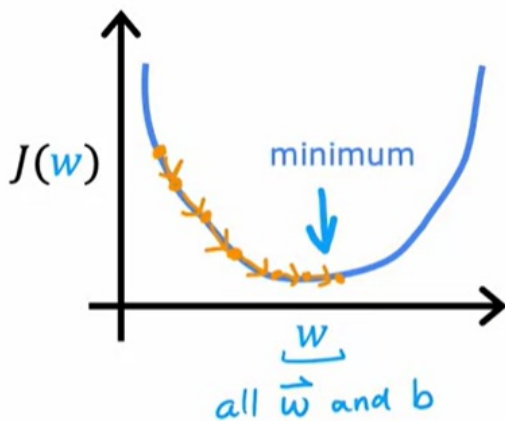
`model.compile(loss= MeanSquaredError())`

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

`from tensorflow.keras.losses import MeanSquaredError`

- Step 3 → Gradient descent (Minimize cost)

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 "back propagation"

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

Activation Functions

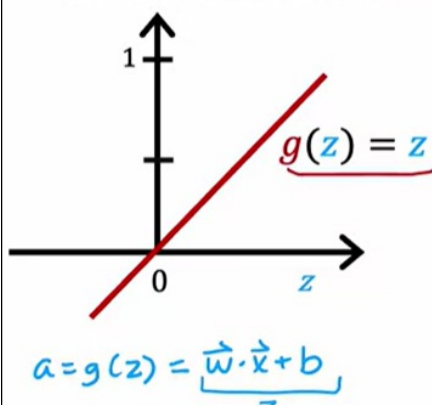
Examples of Activation Functions

- Linear Activation function
- Sigmoid function
- Rectified Linear Unit (ReLU)
- Later: SoftMax

Examples of Activation Functions

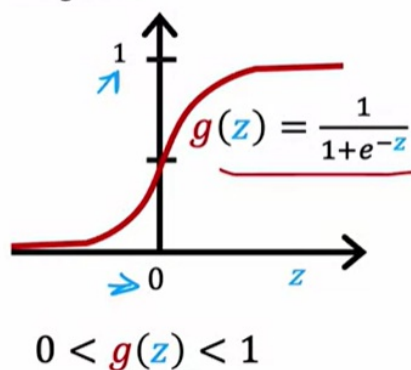
"No activation function"

Linear activation function

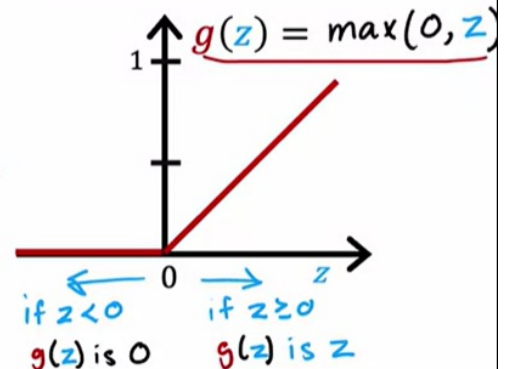


$$a_2^{[1]} = g(\overbrace{\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}}^z)$$

Sigmoid



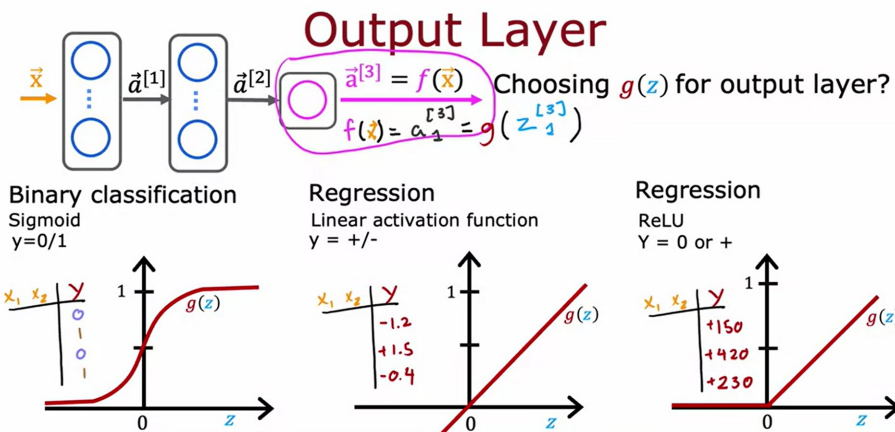
ReLU Rectified Linear Unit



How to Choose Activation Function?

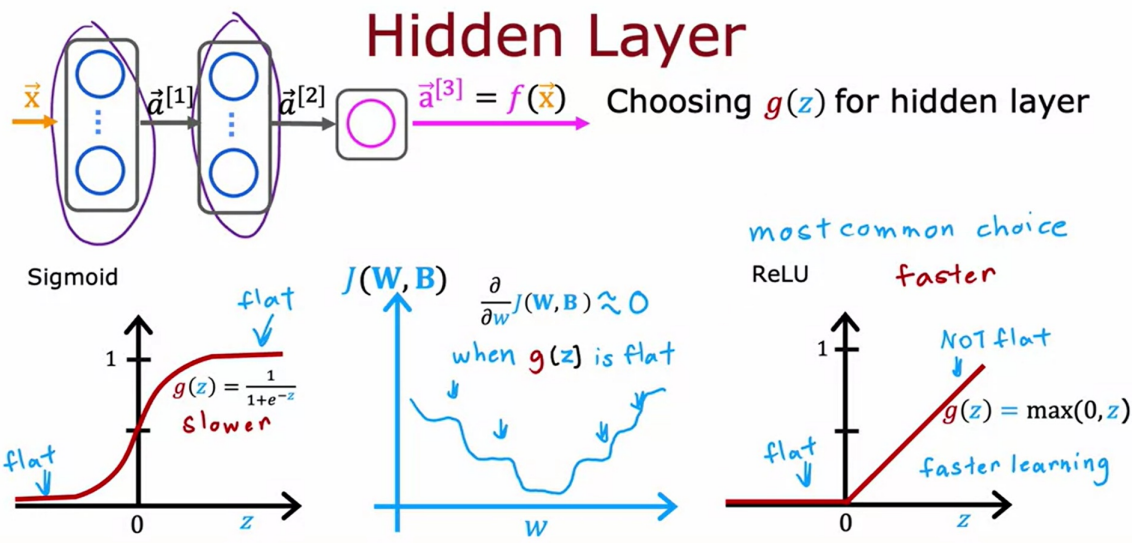
Output layer

- It depends on the type of output you want



Hidden Layer

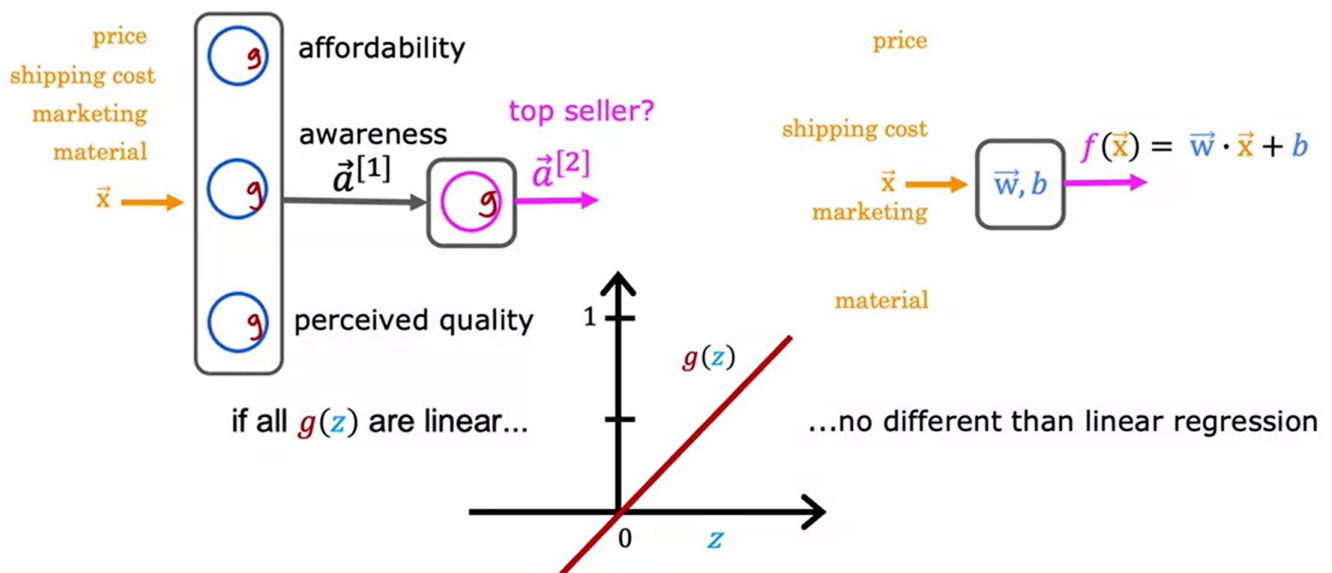
- When the output is flat, it will be slower to reach the minimum point



Why do we need Activation Functions?

- Why do they not work otherwise?
- In this case, having multiple layers will not make calculations any different

Why do we need activation functions?

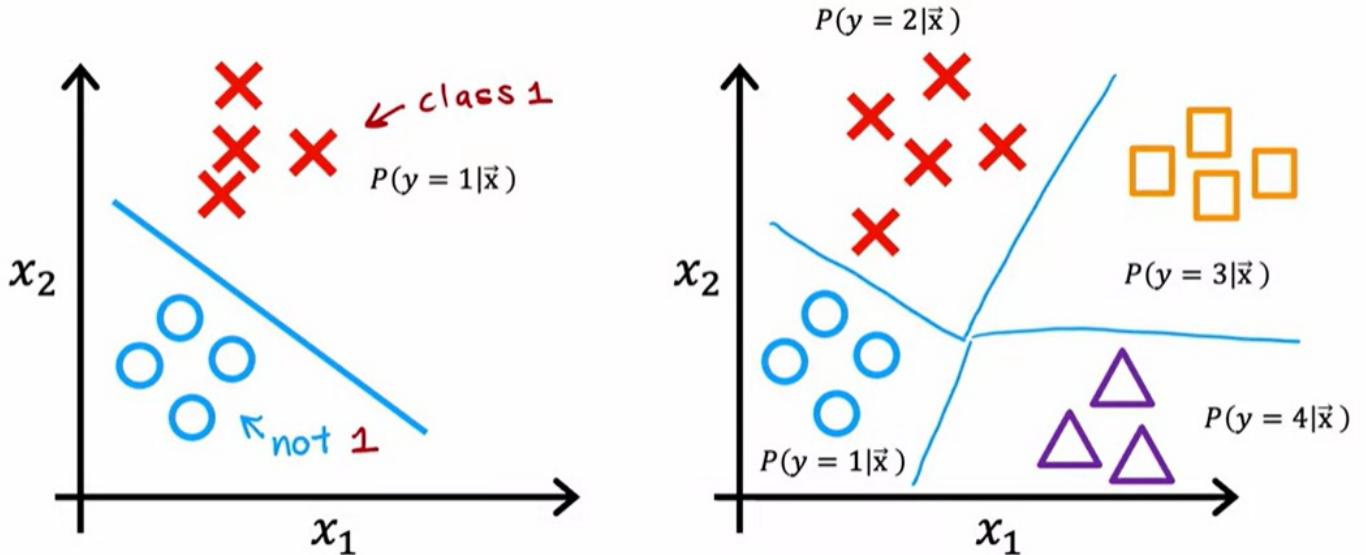


Multiclass Classification

Examples

- What's the chance that Y is = 1 OR $y=2$ OR $y=3$ OR $y=4$?

Multiclass classification example



Logistic vs SoftMax regression

- SoftMax model is a generalization of the logistic regression model!

Logistic regression
(2 possible output values)

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

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

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

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

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

$$\text{note: } a_1 + a_2 + \dots + a_N = 1$$

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

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

$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=1|\vec{x}) \quad 0.30$$

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

$$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=2|\vec{x}) \quad 0.20$$

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

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

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

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

Cost function for SoftMax Regression

- The smaller the number of $a_{j|j}$ (type of class), the greater the loss impact

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

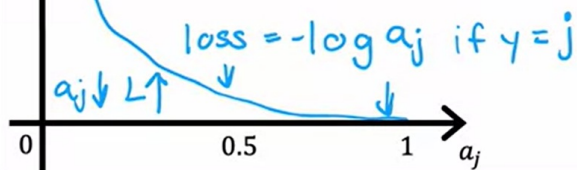
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} = P(y = 1|\vec{x})$$

$$\vdots$$

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

Crossentropy loss

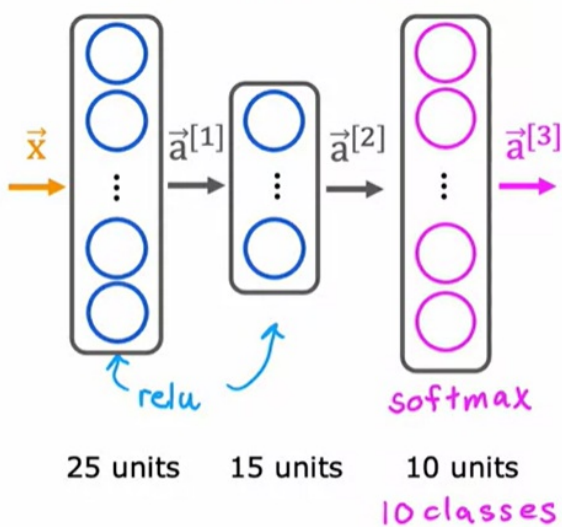
$$\text{loss}(a_1, \dots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ -\log a_2 & \text{if } y = 2 \\ \vdots \\ -\log a_N & \text{if } y = N \end{cases}$$



Neural Network with SoftMax Output Layer

- Output layers consists of several neurons

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

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

Numerical Roundoff Errors

- Tensorflow fix for logistic regression and softmax regression (though not fixing for logistic regression is generally acceptable)
- However, problem will get worse in softmax regression

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'), 'linear'
    Dense(units=1, activation='sigmoid')
])
```

Original loss

$$\text{loss} = -y \log(\hat{a}) - (1 - y) \log(1 - \hat{a})$$

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

More accurate loss (in code)

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

logit: z

Predict step changes:

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)
```

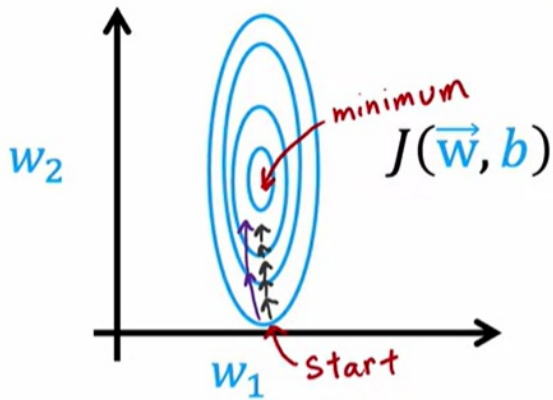
Additional Neural Network Concepts

Advanced Optimization

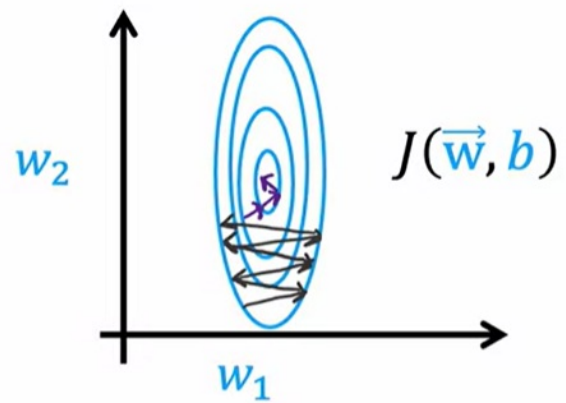
Adaptive moment estimation (Adam) Algorithm

- Having different learning rates for each feature
- If weight increase s in 1 direction, increase learning rate (if going 1 direction down the hill, go faster, if zigzagging, go slower)

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 .

TF Implementation

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

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

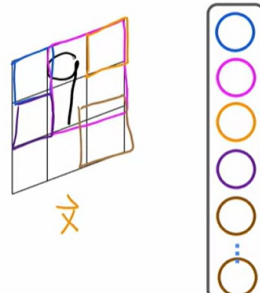
$$\alpha = 10^{-3} = 0.001$$

fit

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

Additional Layer Type

Convolutional Layer

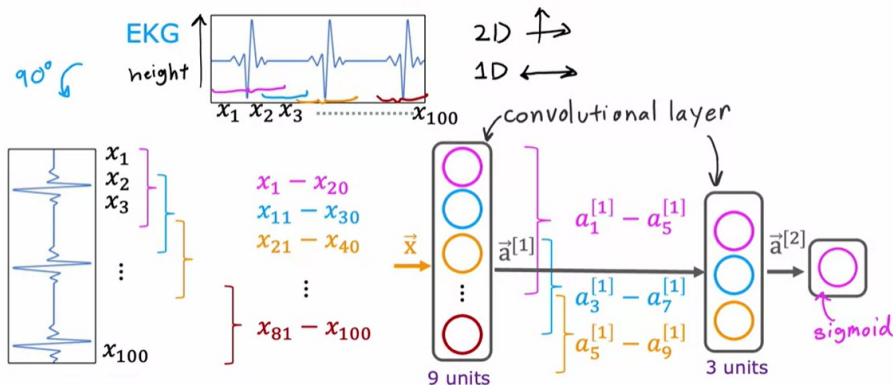


Each Neuron only looks at part of the previous layer's inputs.

Why?

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

Convolutional Neural Network

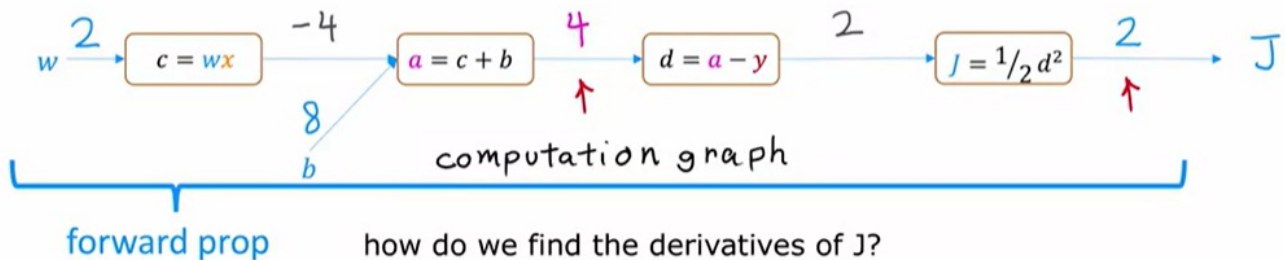
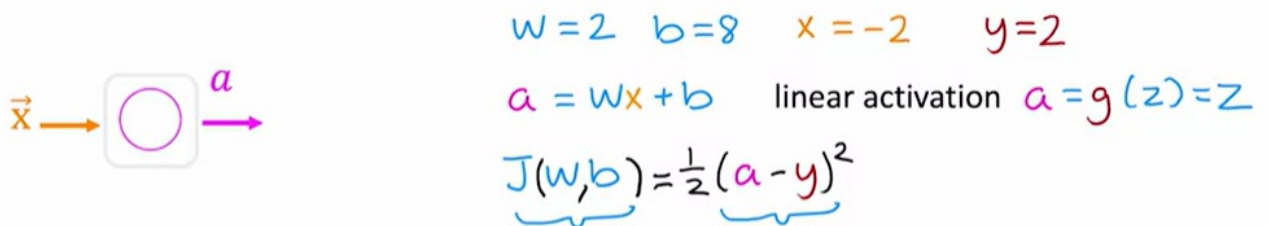


Backpropagation

Computation Graph

Computation graph for Forward Propagation

Small Neural Network Example



how do we find the derivatives of J ?

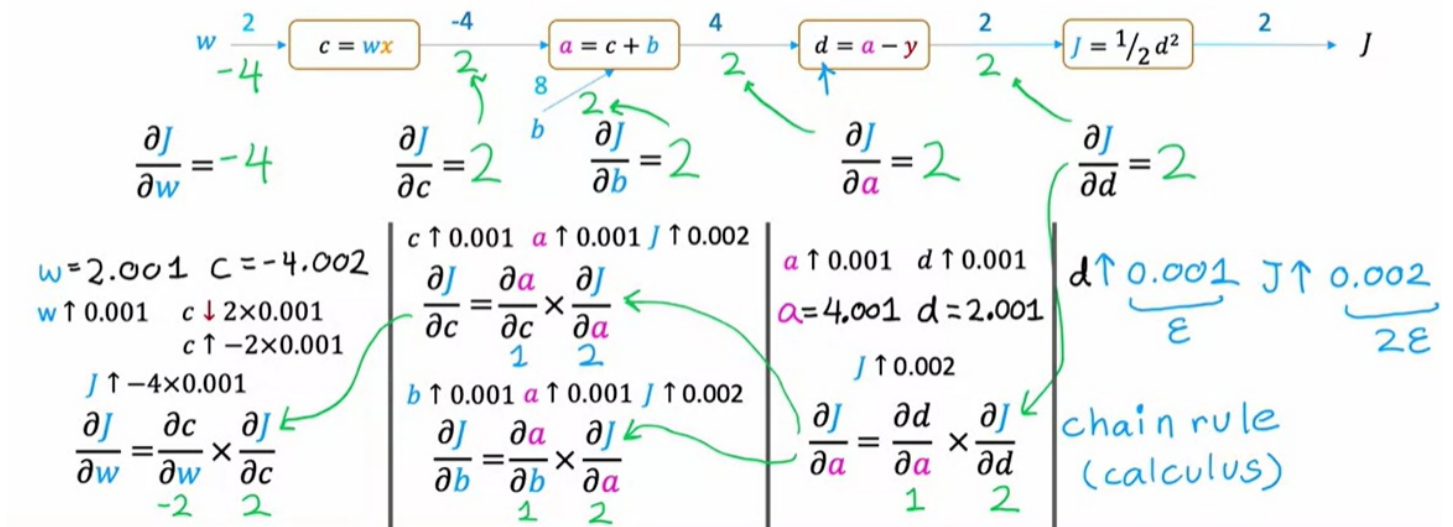
Computation graph for Backward Propagation

- Computing the derivatives at each step is part of backward propagation
- Using the constant derived from

Computing the Derivatives

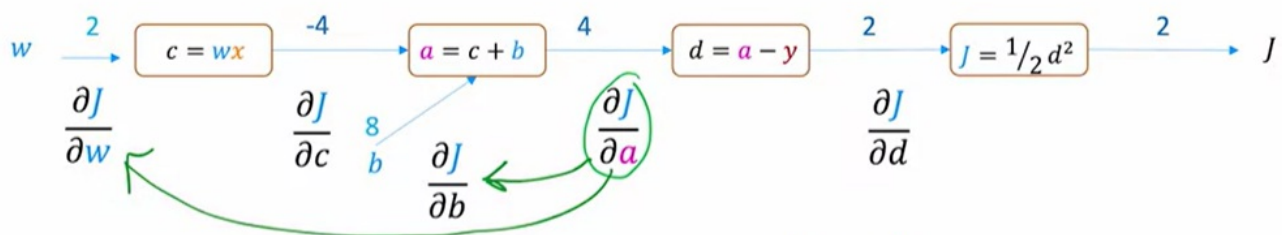
$w = 2$ $b = 8$ $x = -2$ $y = 2$ $a = wx + b$ $J = \frac{1}{2}(a - y)^2$

Forward prop →
 ← Back prop



Why use Backward Prop?

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