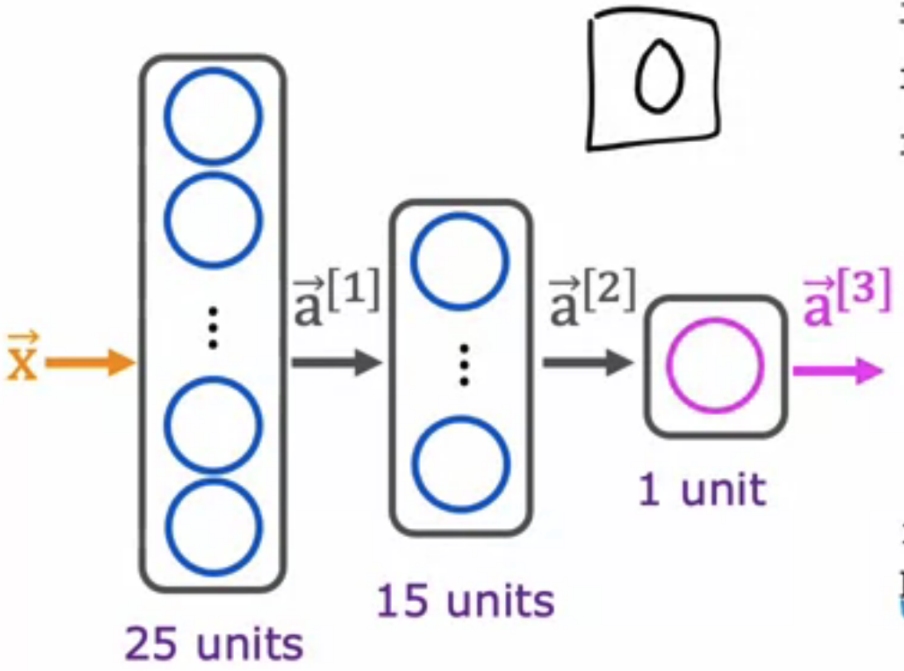
# Train a Neural Network in TensorFlow




0

```
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?

# Model Training Steps

TensorFlow

1. specify how to compute output given input x and parameters w,b (define model)

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

2. specify loss and cost

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

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

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)
```

# 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
```

K Keras

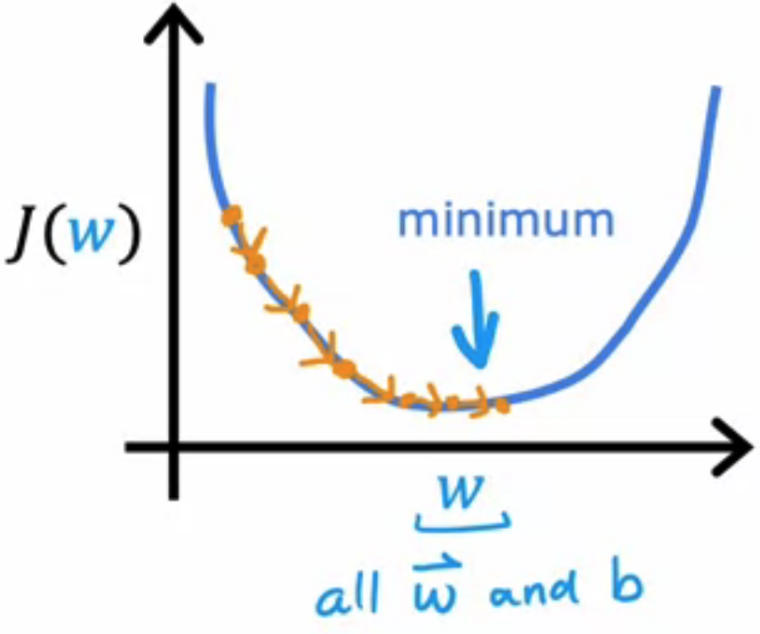
regression (predicting numbers and not categories)

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 "backpropagation"

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