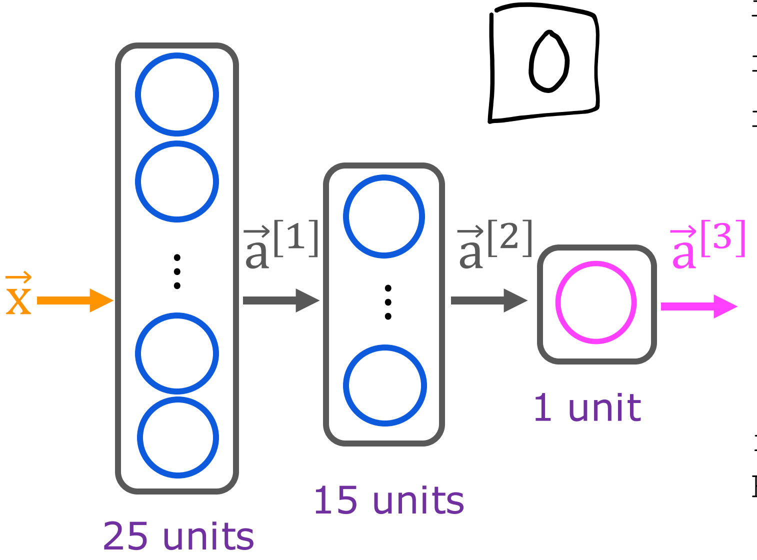
Neural Network Training

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?

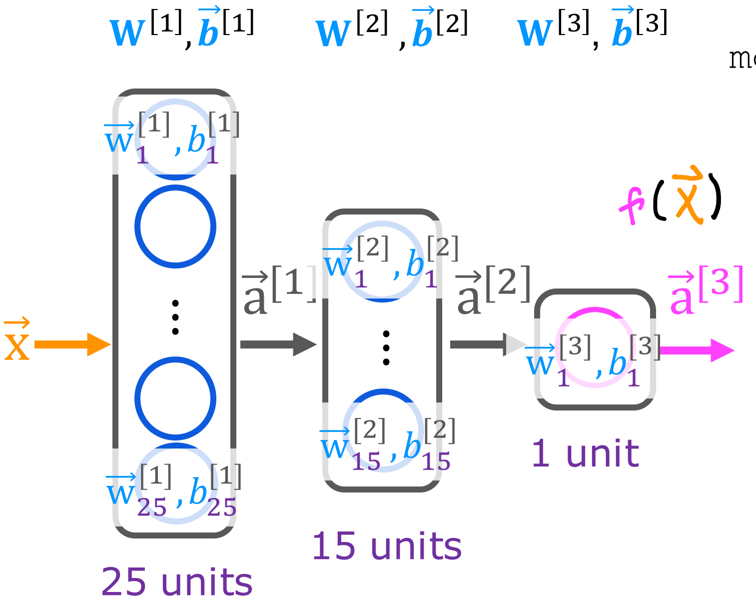
Model Training Steps

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

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

Mnist digit classification problem

binary classification

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

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
```

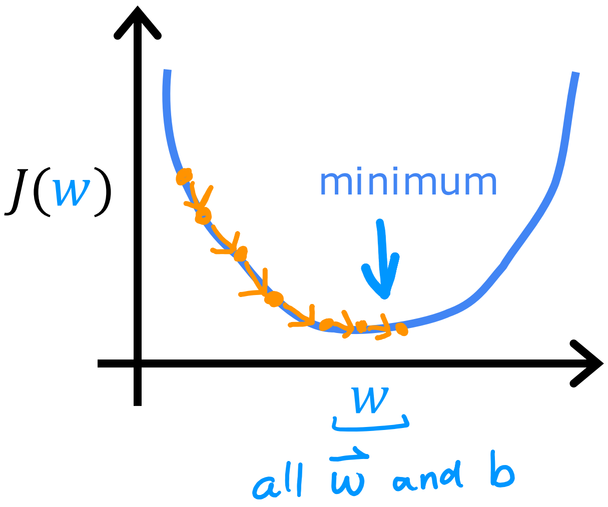
regression (predicting numbers and not categories)

mean squared error

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

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

3. Gradient descent



repeat {

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

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

}

Compute derivatives for gradient descent using "back propagation"

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

Neural network libraries

Use code libraries instead of coding ”from scratch”

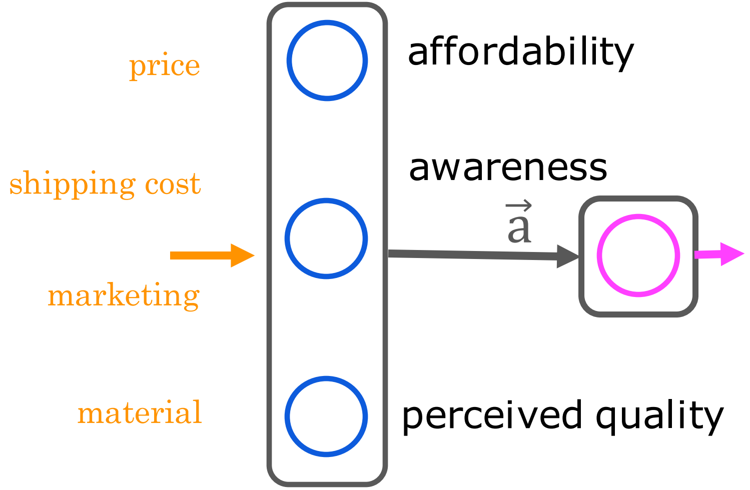
TensorFlow PyTorch

Good to understand the implementation

Activation Functions

Alternatives to the sigmoid activation

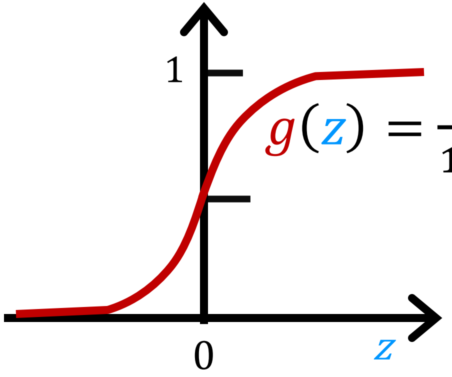
Demand Prediction Example



$$\overbrace{}^{z}$$

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

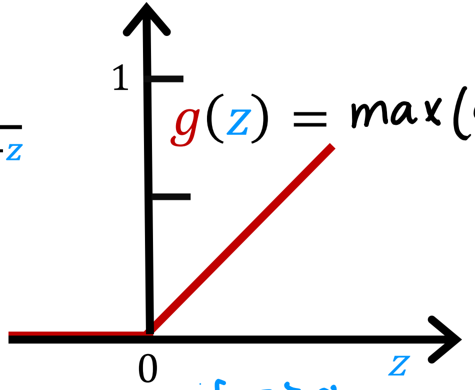
Sigmoid



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

$$0 < g(z) < 1$$

ReLU Rectified Linear Unit



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

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

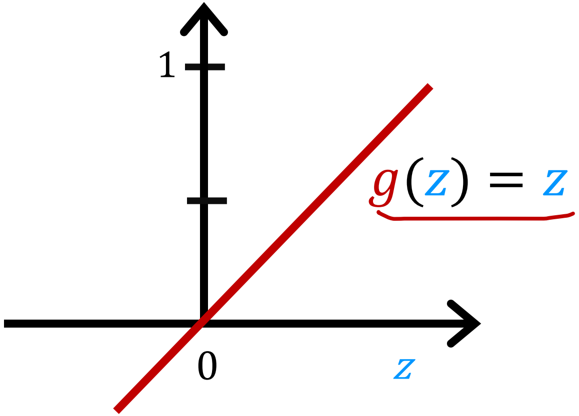
if $z \geq 0$ $g(z)$ is z

Examples of Activation Functions

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

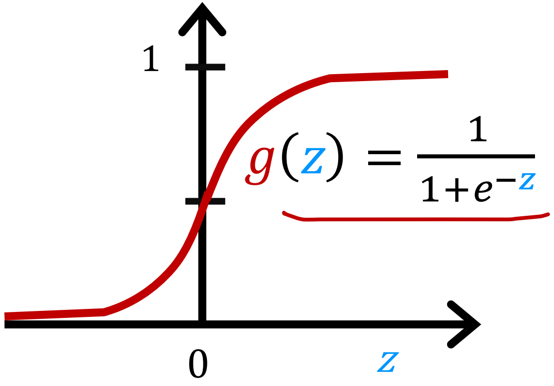
"No activation function"

Linear activation function



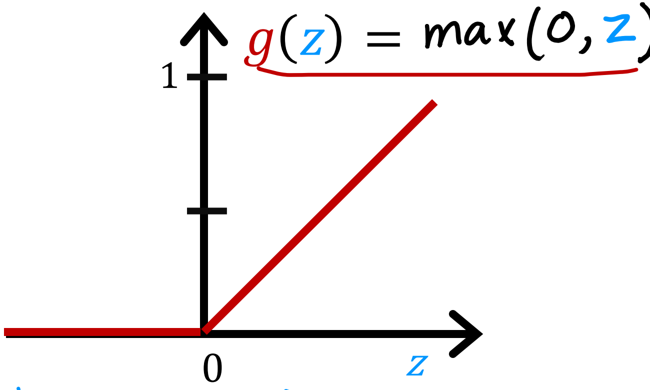
$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$

Sigmoid



$$0 < g(z) < 1$$

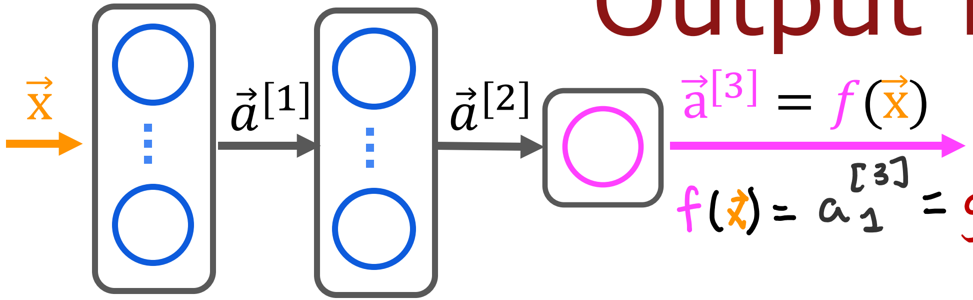
ReLU Rectified Linear Unit



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

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

Output Layer



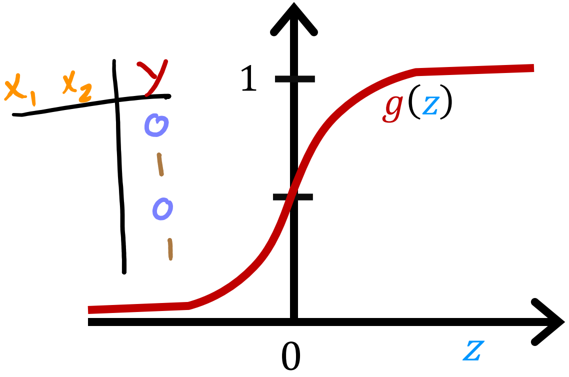
$f(\vec{x}) = a_1^{[3]} = g(z_1^{[3]})$

Choosing $g(z)$ for output layer?

Binary classification

Sigmoid

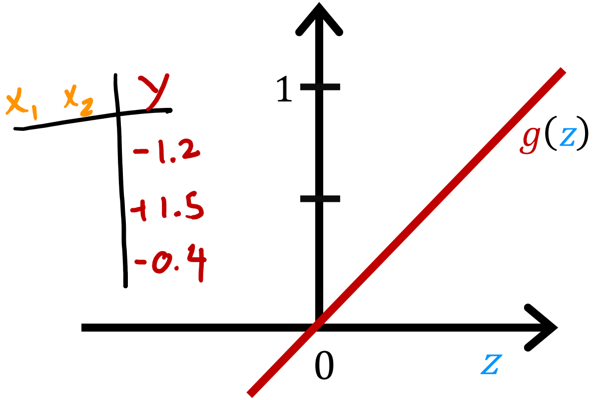
y=0/1



Regression

Linear activation function

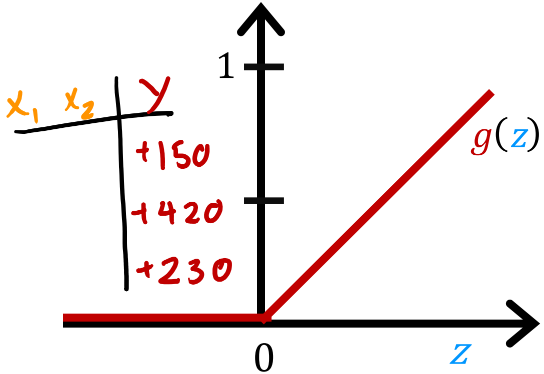
y = +/-



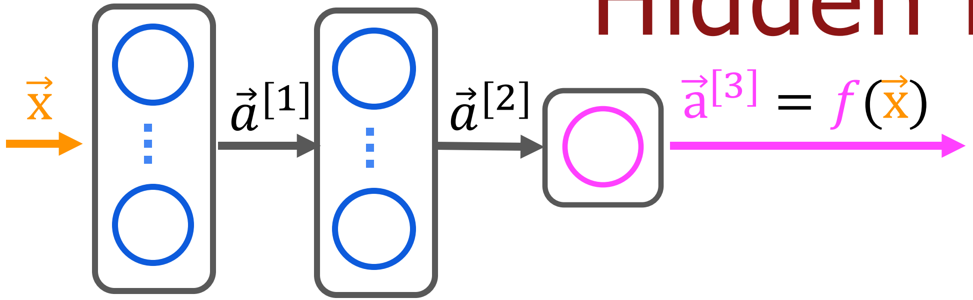
Regression

ReLU

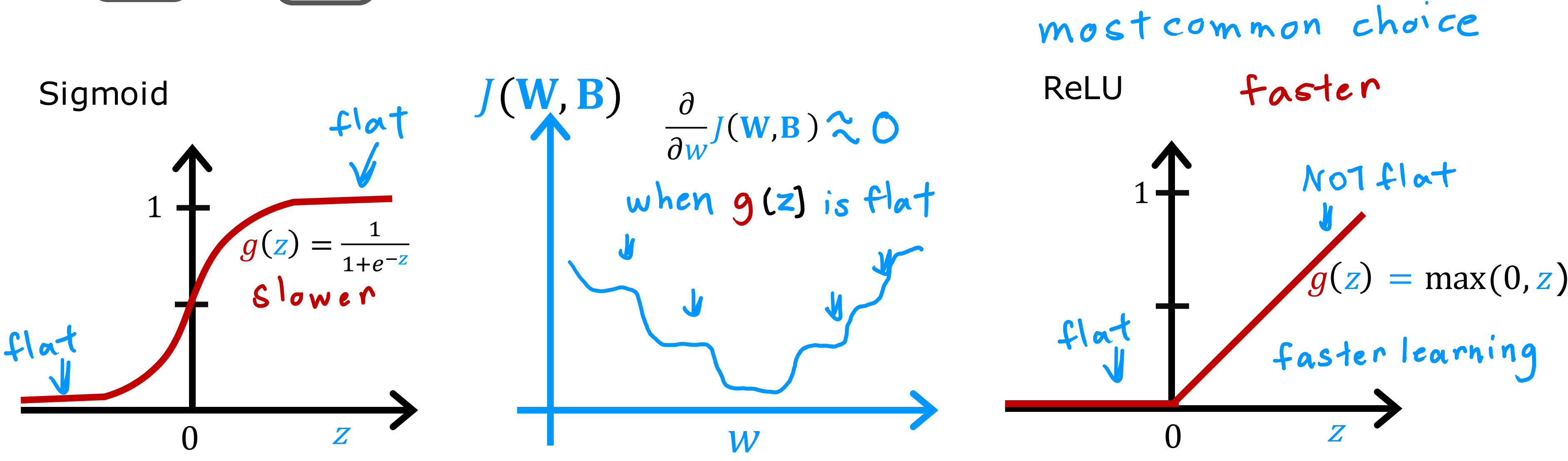
Y = 0 or +



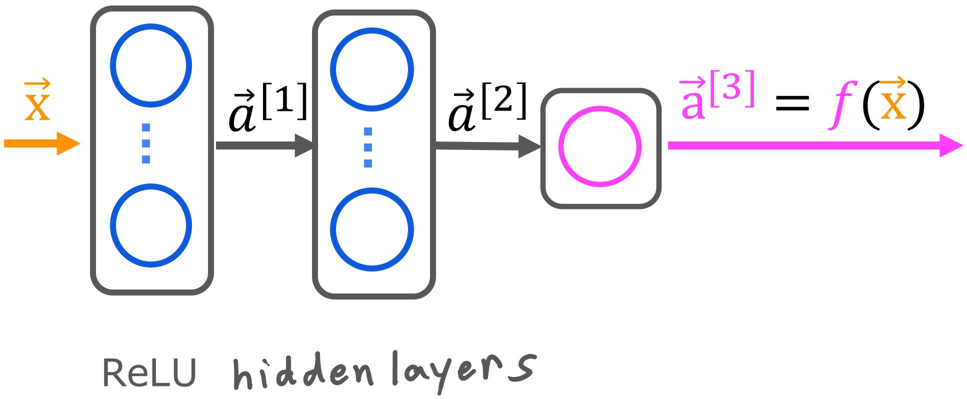
Hidden Layer



Choosing $g(z)$ for hidden layer



Choosing Activation Summary



binary classification
activation='sigmoid'

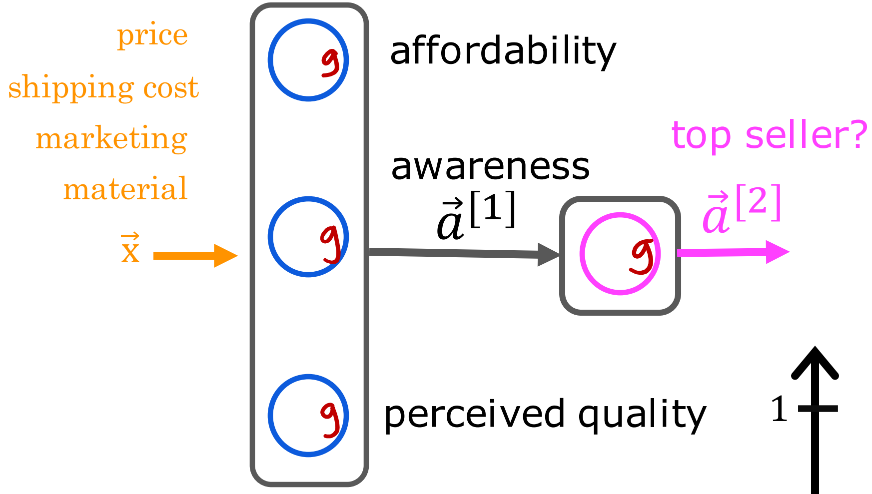
regression y negative/positive
activation='linear'

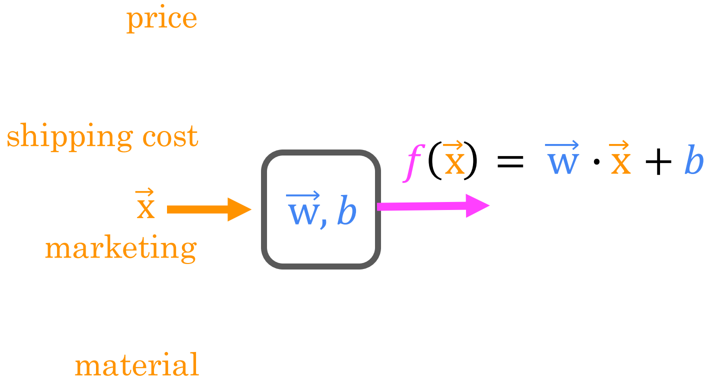
regression y ≥ 0
activation='relu'

```
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
])
```

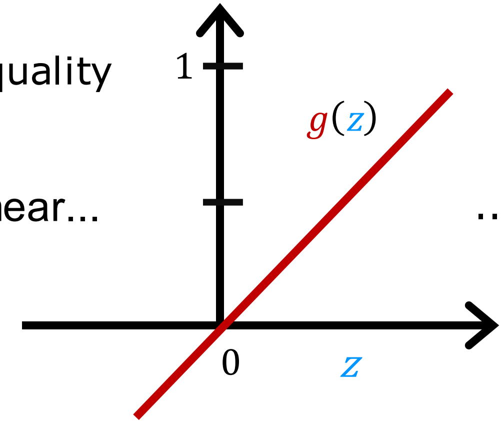
or 'linear'
or 'relu'

Why do we need activation functions?



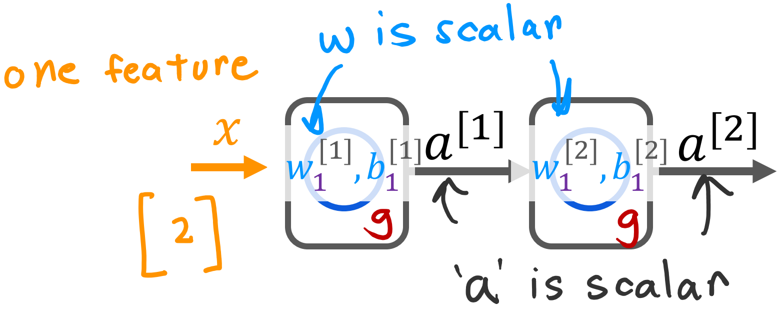


if all $g(z)$ are linear...



...no different than linear regression

Linear Example



$$g(z) = z$$

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

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

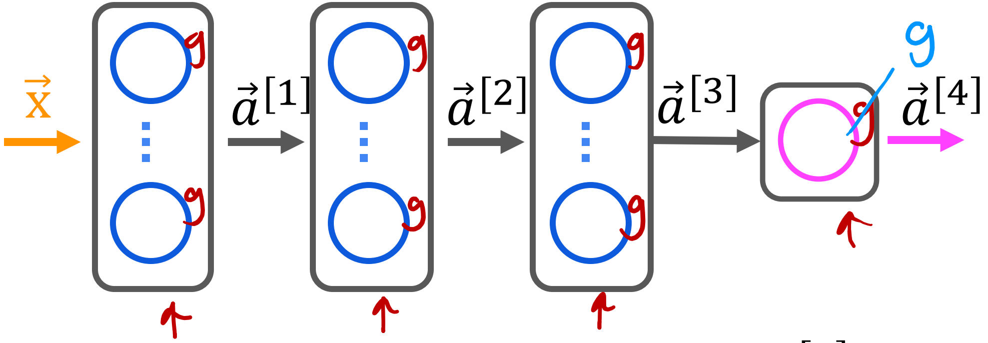
$$= w_1^{[2]} (w_1^{[1]} x + b_1^{[1]}) + b_1^{[2]}$$

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

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

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

Example



$g(z) = z$

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

all linear (including output)
↳ equivalent to linear regression

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

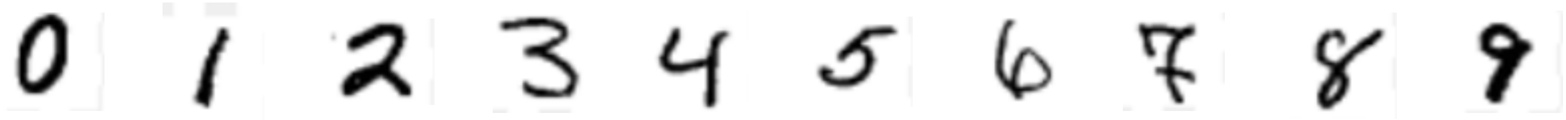
output activation is sigmoid
(hidden layers still linear)
↳ equivalent to logistic regression

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

Multiclass Classification

Multiclass

MNIST example



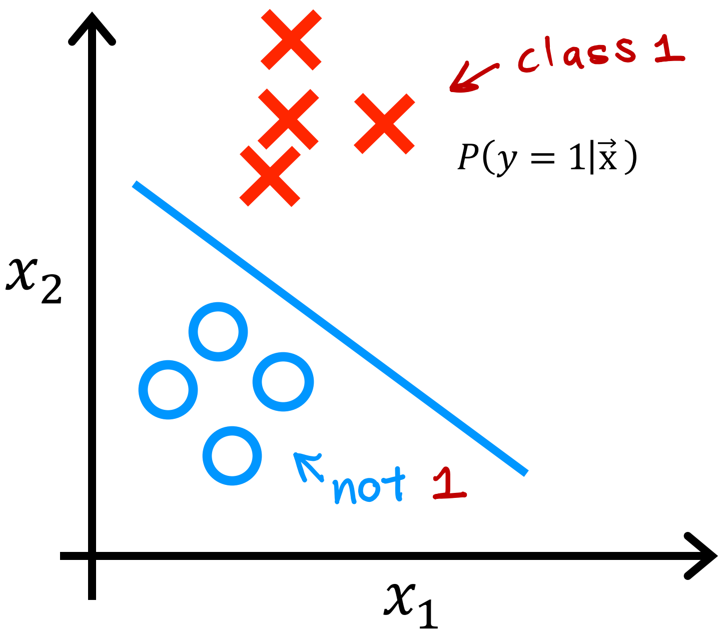
$y = 0$ 1 2 3 4 5 6 7 8 9

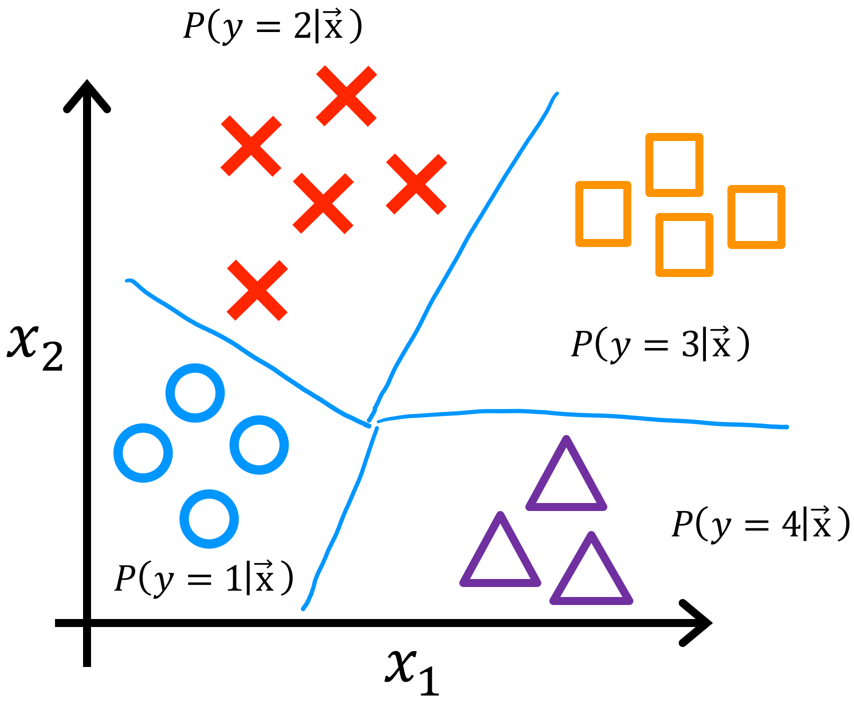
x  $y = 7$

multiclass classification problem:

target y can take on more than two possible values

Multiclass classification example





Multiclass Classification

Softmax

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

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

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

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

$$\text{loss} = -y \log a_1 - (1 - y) \log(1 - a_1)$$

if y=1 (under $-y \log a_1$); if y=0 (under $\log(1 - a_1)$, which is a_2)

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

Softmax regression

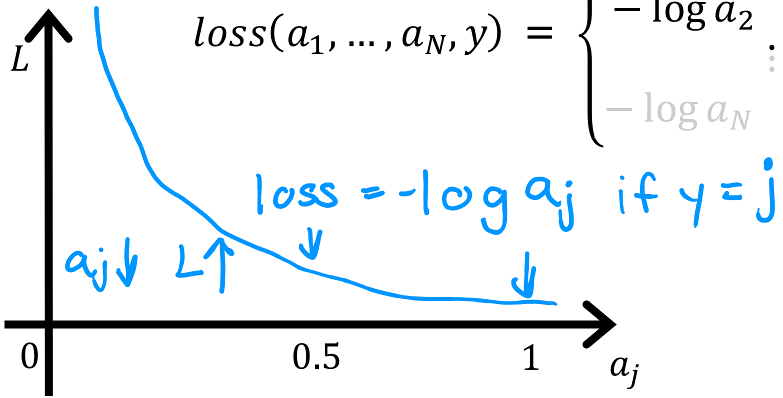
$$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = 1|\vec{x})$$

$$\vdots$$

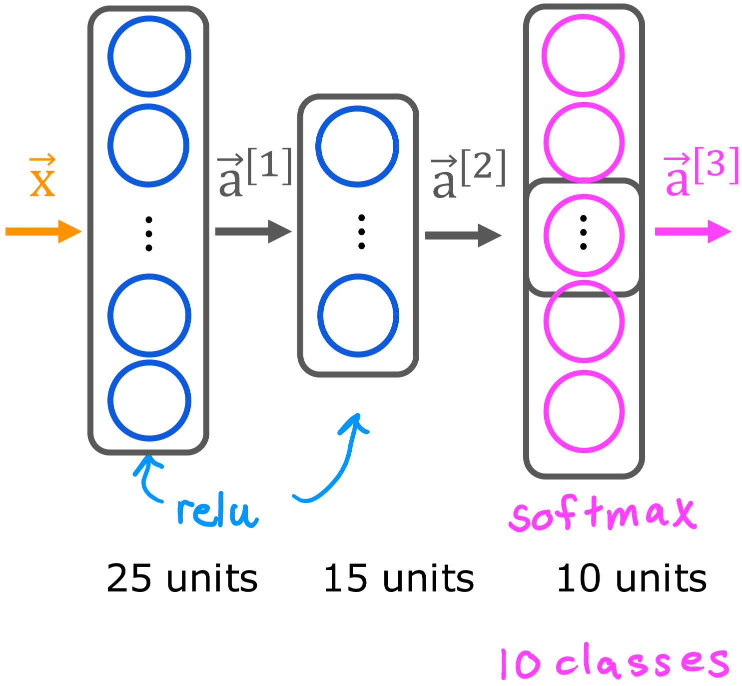
$$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = N|\vec{x})$$

Crossentropy loss

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



Neural Network with Softmax output



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

$$\vdots$$

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

logistic regression

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

softmax

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

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.

Multi-label Classification

x

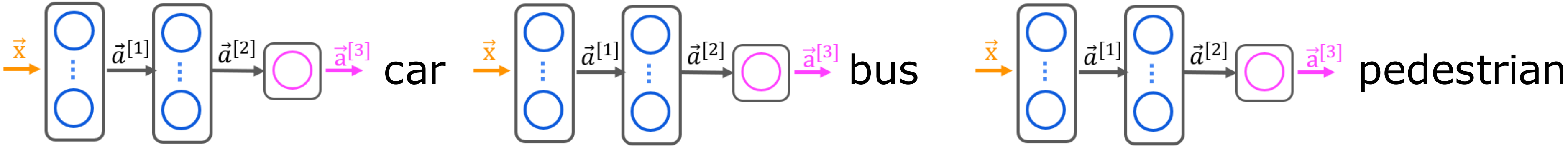
Is there a car? yes | no | yes

Is there a bus? no | no | yes

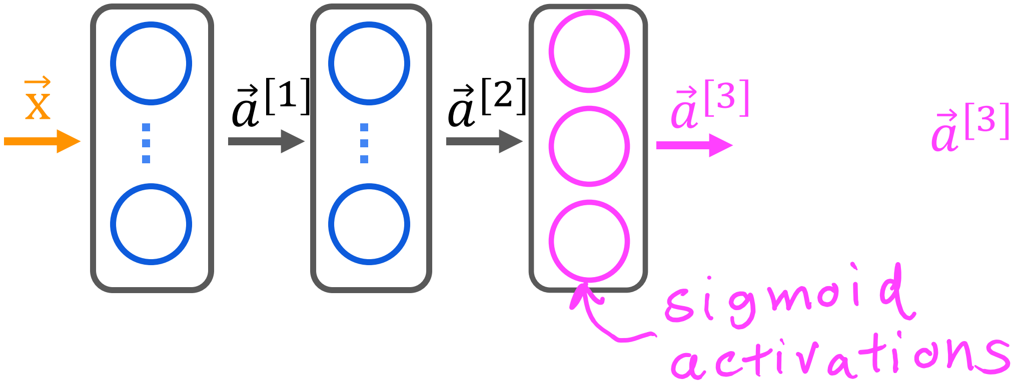
Is there a pedestrian yes | yes | no

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

Multiple classes



Alternatively, train one neural network with three outputs

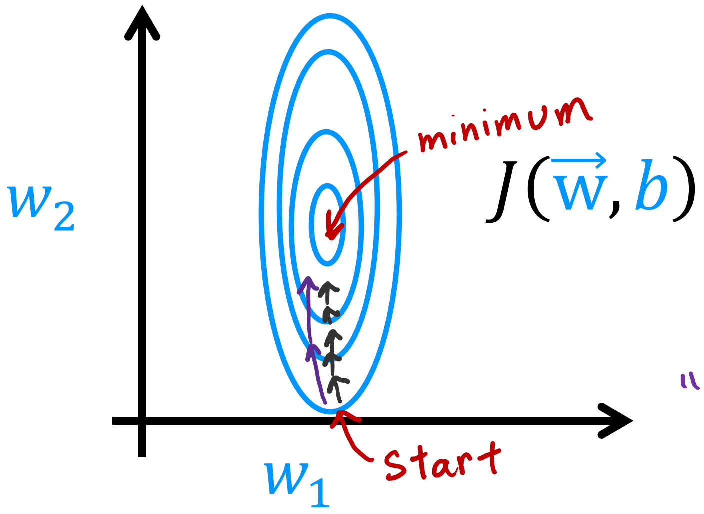


$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix} \begin{matrix} \text{car} \\ \text{bus} \\ \text{pedestrian} \end{matrix}$$

Gradient Descent

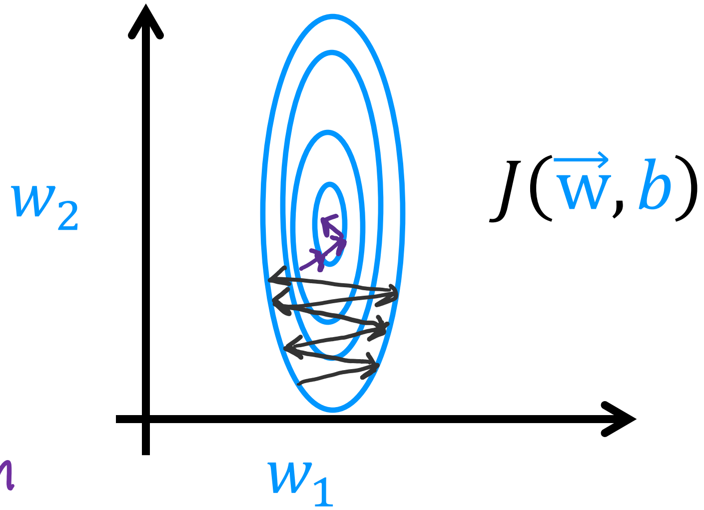
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

learning rate



Go faster – increase α

"Adam" algorithm



Go slower – decrease α

Adam Algorithm Intuition

Adam: Adaptive Moment estimation   not just one α

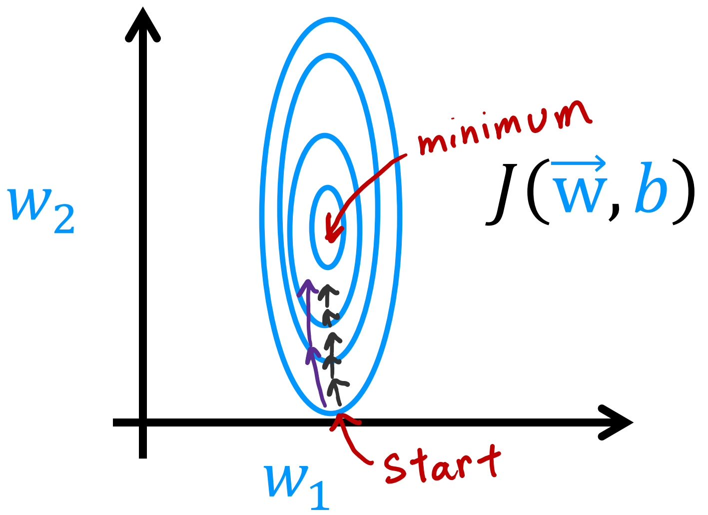
$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{\mathrm{w}}, b)$$

$$\vdots$$

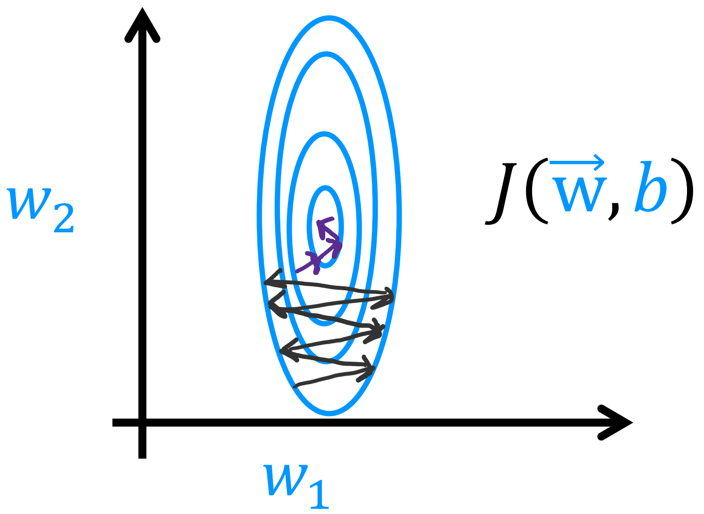
$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{\mathrm{w}}, b)$$

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

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.

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)
```