First week

Advanced learning algorithms

Neural Networks
- inference (prediction)
- training

Practical advice for building machine learning systems

Decision Trees

决策树与神经网络相比。
decision trees compared to neural networks.

DeepLearning.AI Andrew Ng

Neural networks

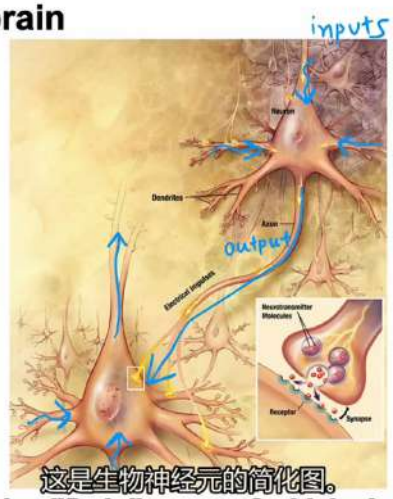
Origins: Algorithms that try to mimic the brain.

Used in the 1980's and early 1990's.
Fell out of favor in the late 1990's.

Resurgence from around 2005.

speech → images → text (NLP) → ...

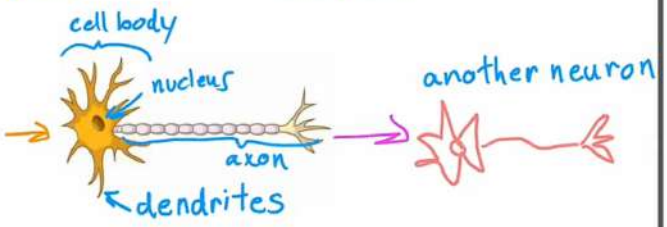
Neurons in the brain



这是生物神经元的简化图。

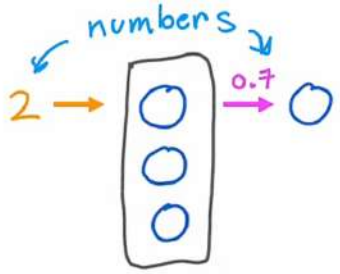
Biological neuron

inputs outputs

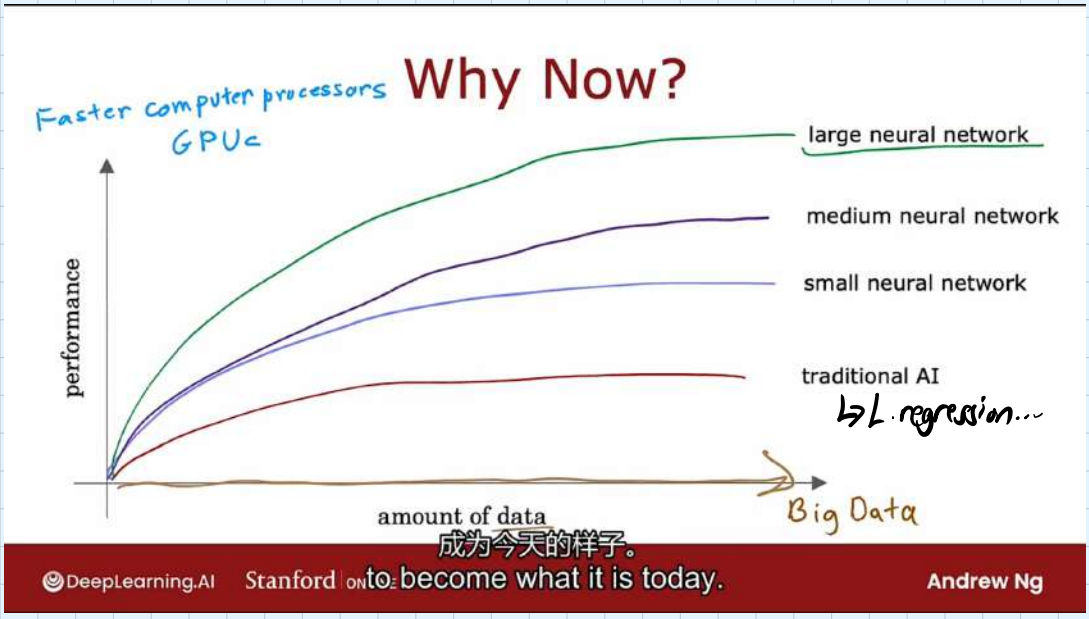


Simplified mathematical model of a neuron

inputs outputs



Why now ???



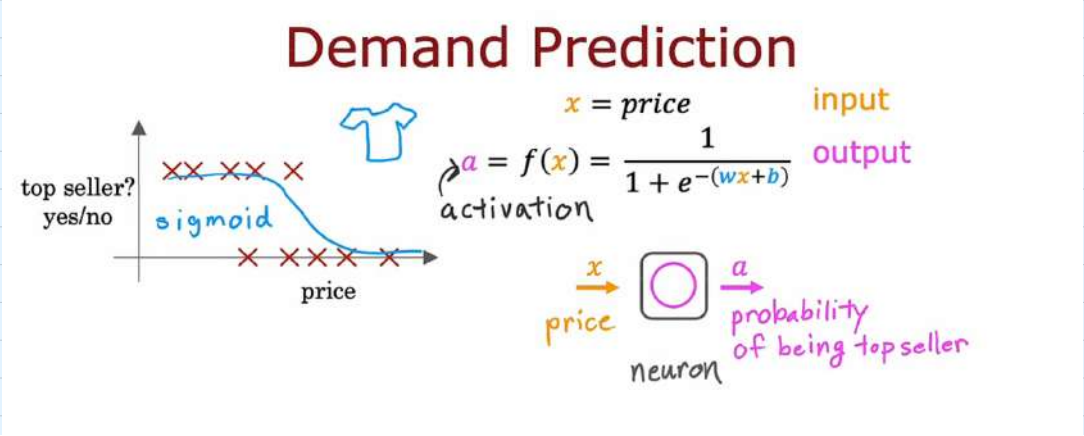
Old AI can't implement

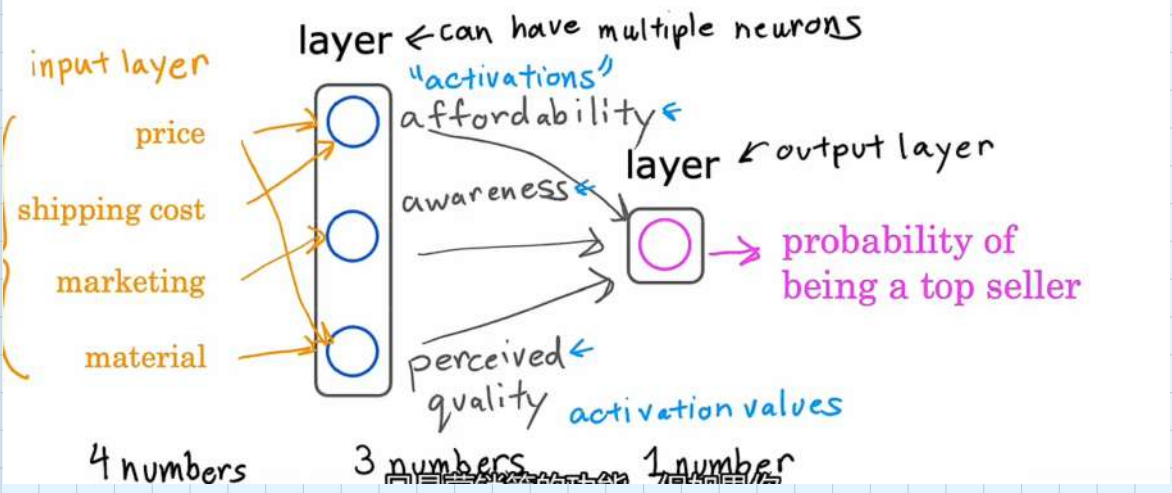
语音识别

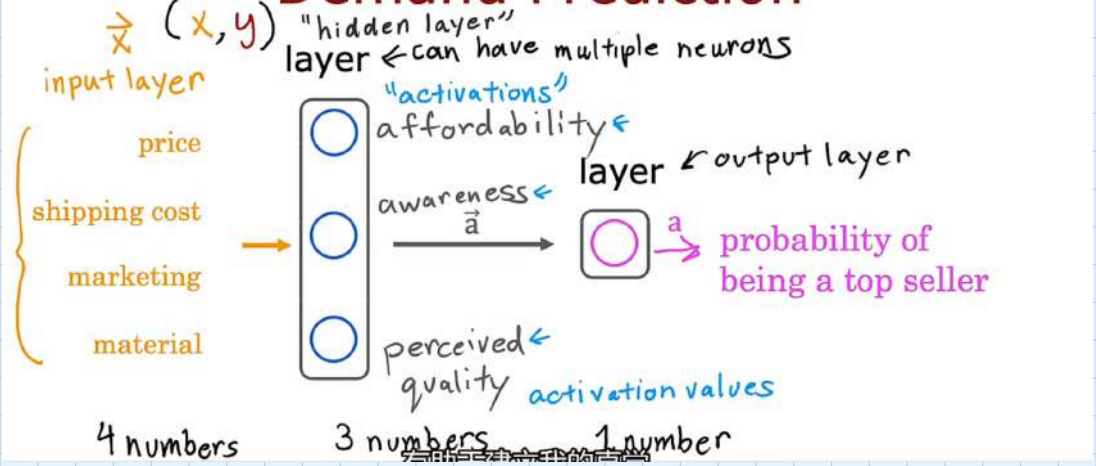
图象处理

NLP ...

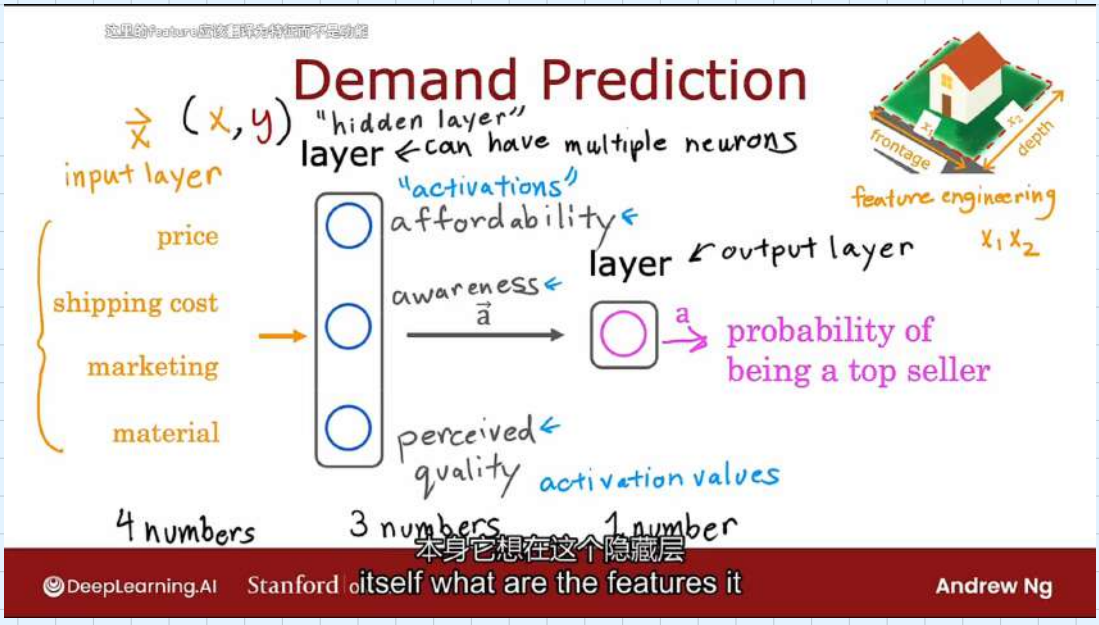
Demand prediction



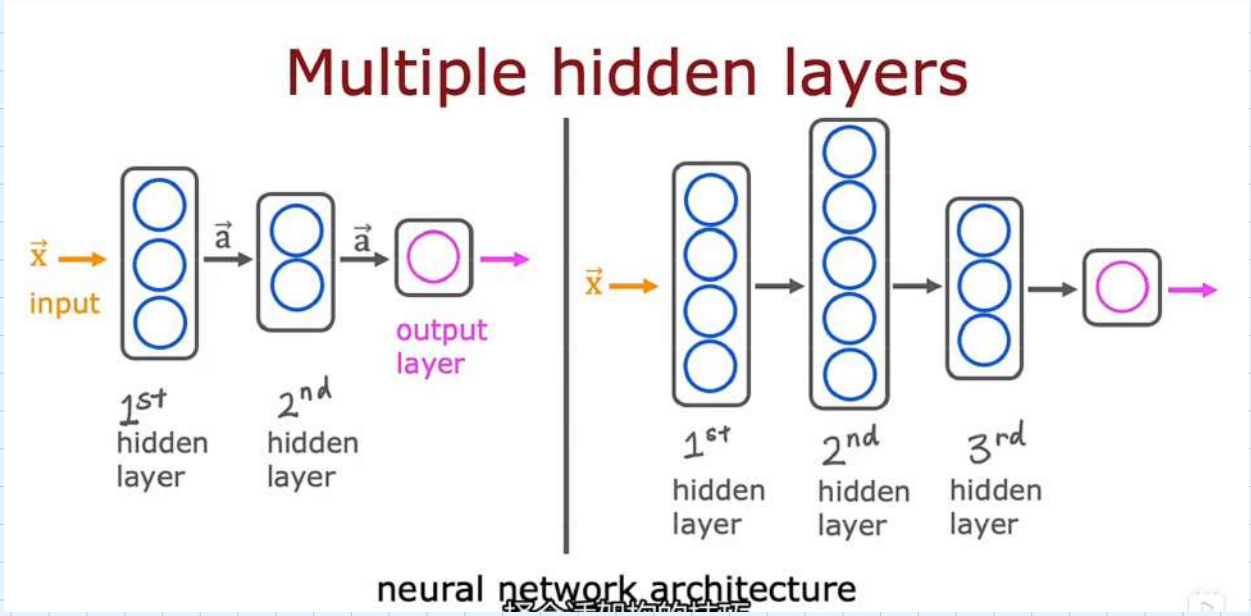




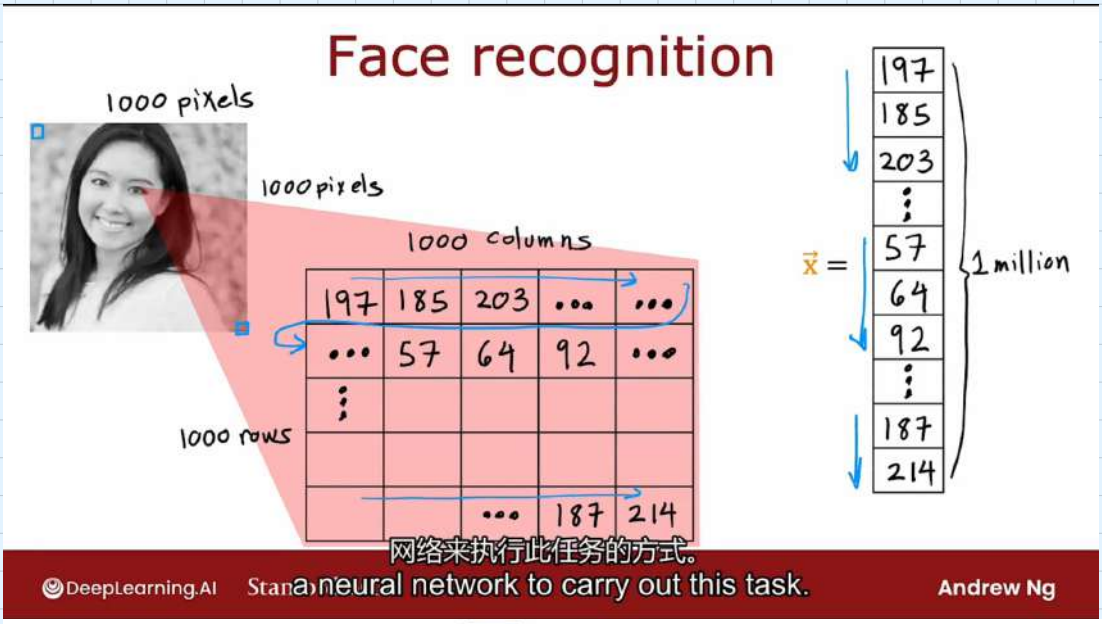
hidden layer make it stronger!

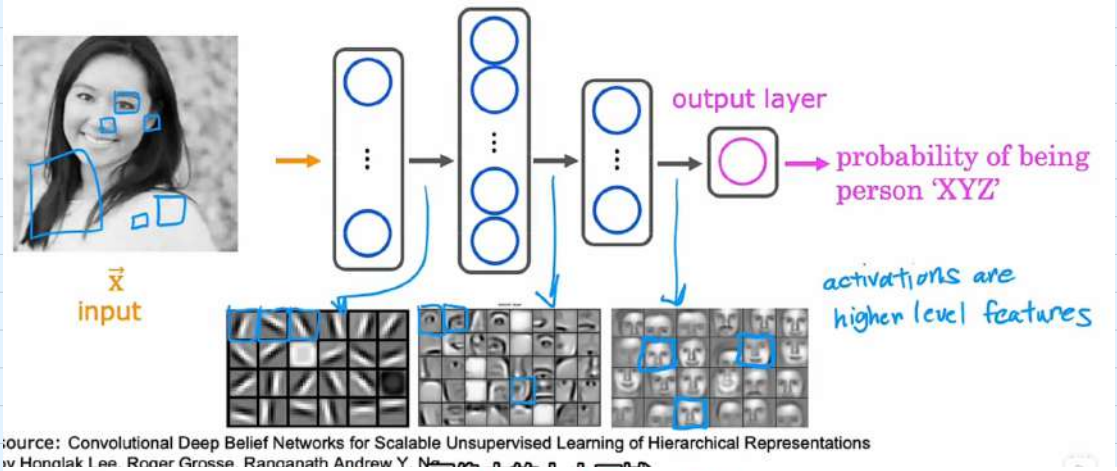


just like Logistic Regression

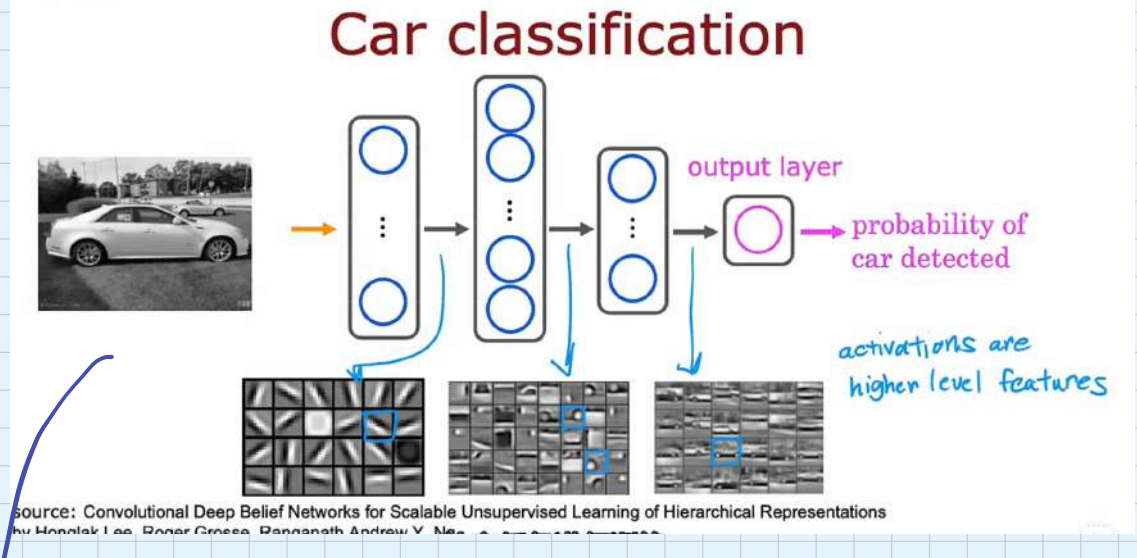


Face recognition





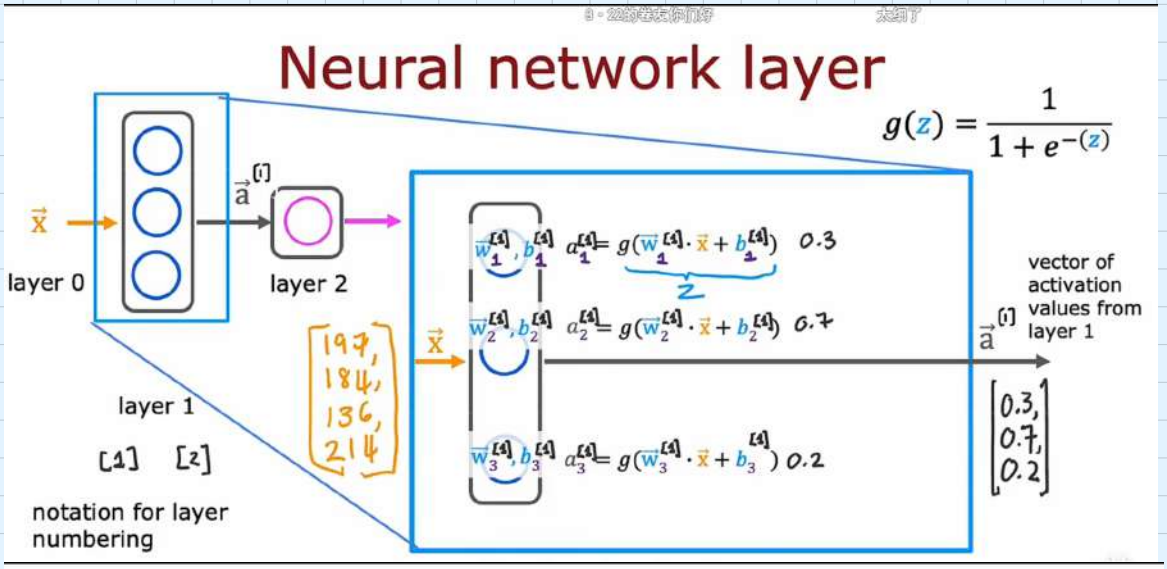
Source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

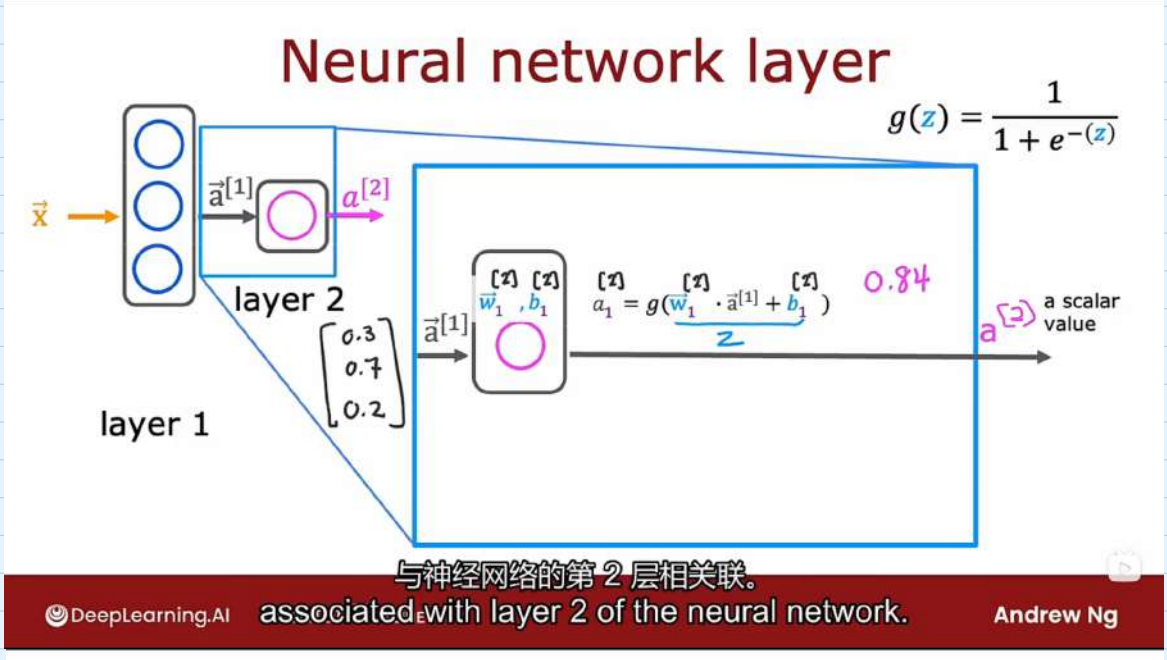


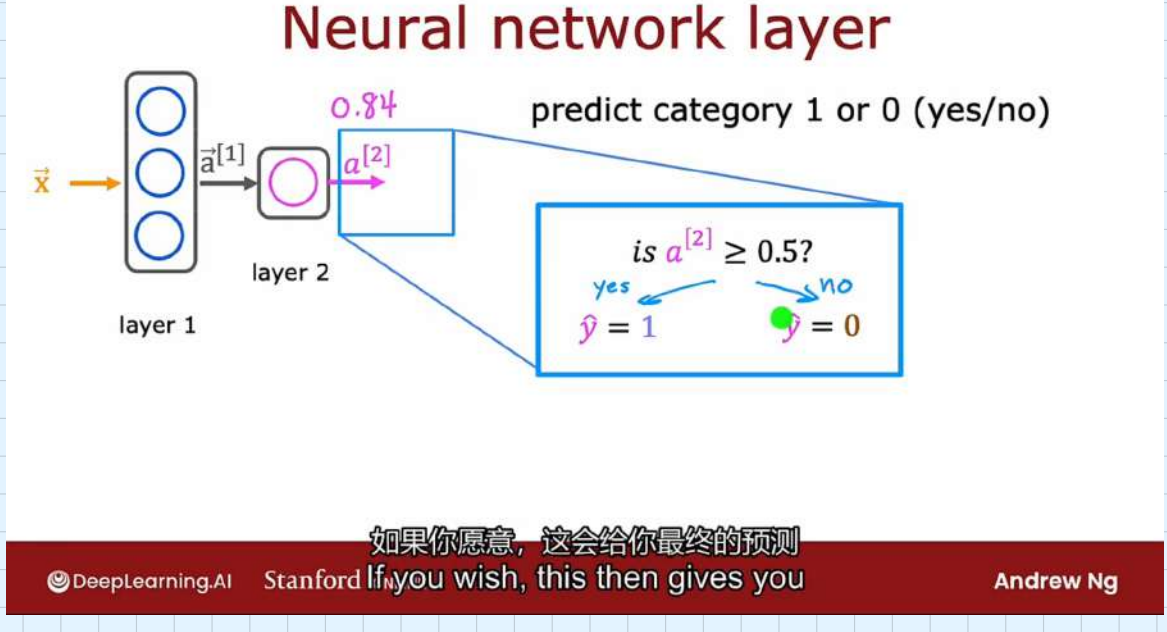
Source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

不管是什么进去，它会自动识别

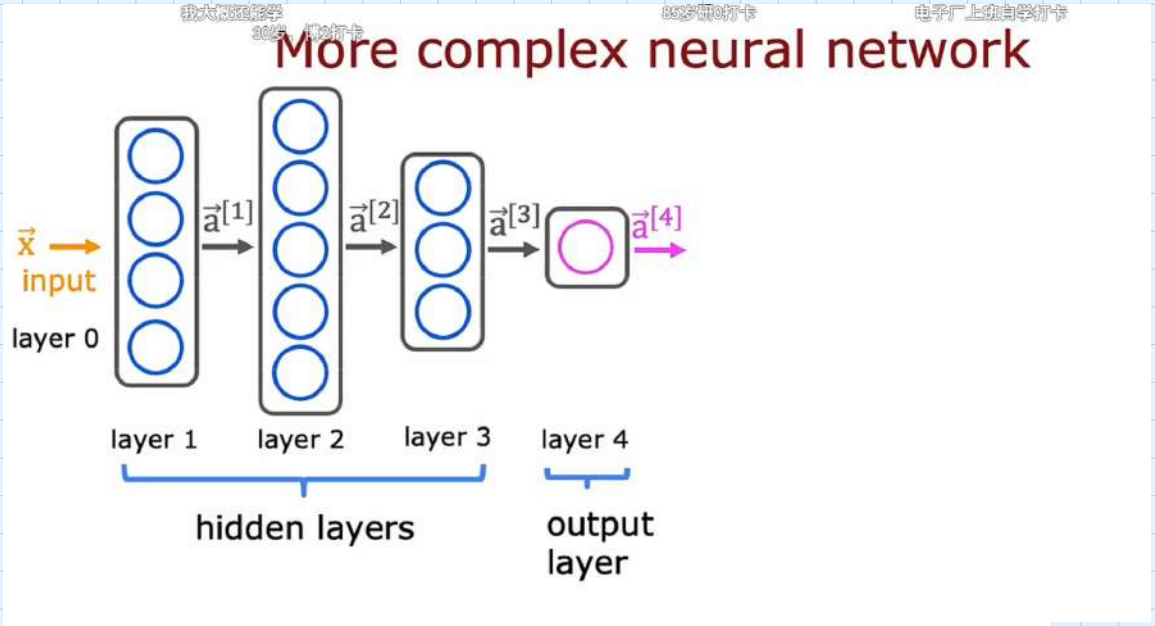
Layer

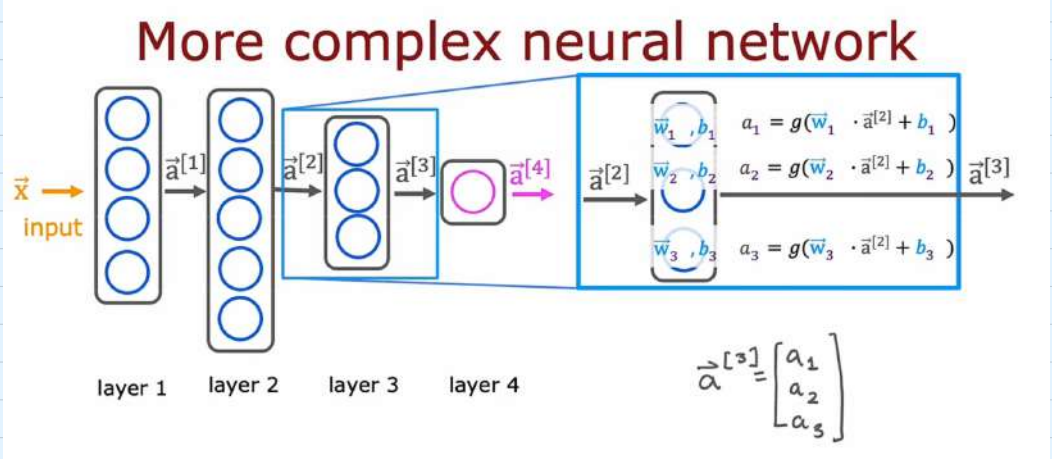

Neural network layer
$g(z) = \frac{1}{1+e^{-(z)}}$
layer 0
layer 2
layer 1
notation for layer numbering
vector of activation values from layer 1
0.3
0.7
0.2
197, 184, 136, 214

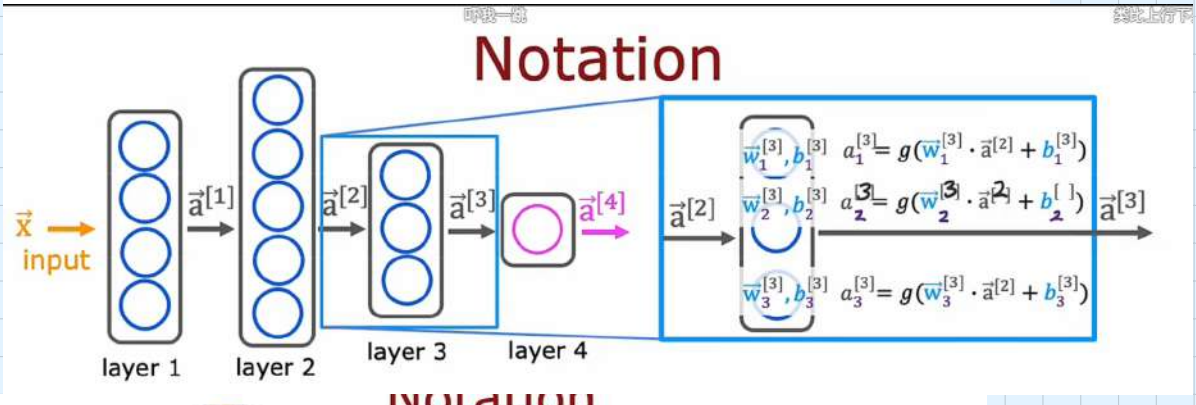

Neural network layer
$g(z) = \frac{1}{1+e^{-(z)}}$
layer 2
layer 1
0.84
a scalar value
0.3
0.7
0.2
与神经网络的第 2 层相关联。
associated with layer 2 of the neural network.
DeepLearning.AI
Andrew Ng

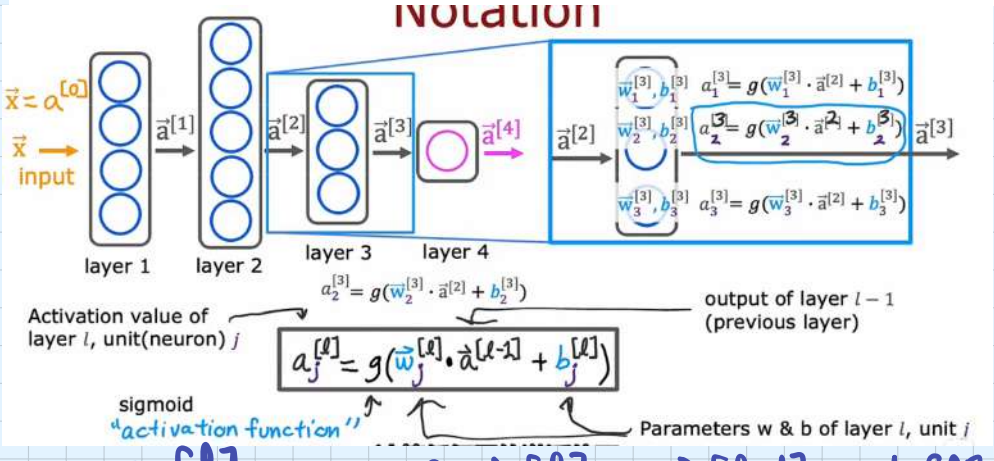

Neural network layer
0.84
predict category 1 or 0 (yes/no)
layer 2
layer 1
is $a^{[2]} \geq 0.5$?
yes
no
$\hat{y} = 1$
$\hat{y} = 0$
如果你愿意，这会给你最终的预测
If you wish, this then gives you
DeepLearning.AI
Stanford
Andrew Ng

Complex neural network



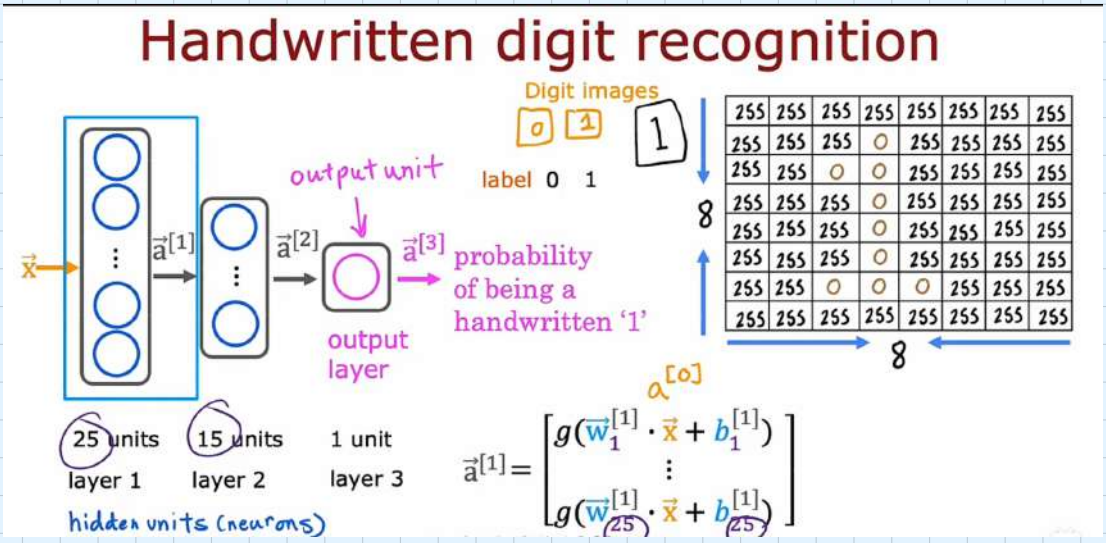






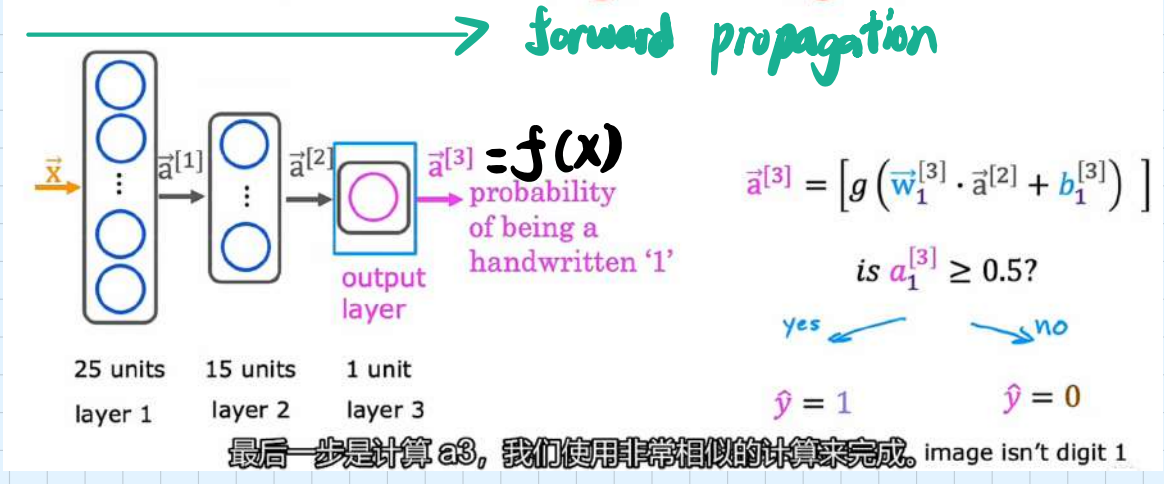
$$\Rightarrow \quad a_j^{[l]} = g\left(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]}\right)$$

Handwritten digit recognition

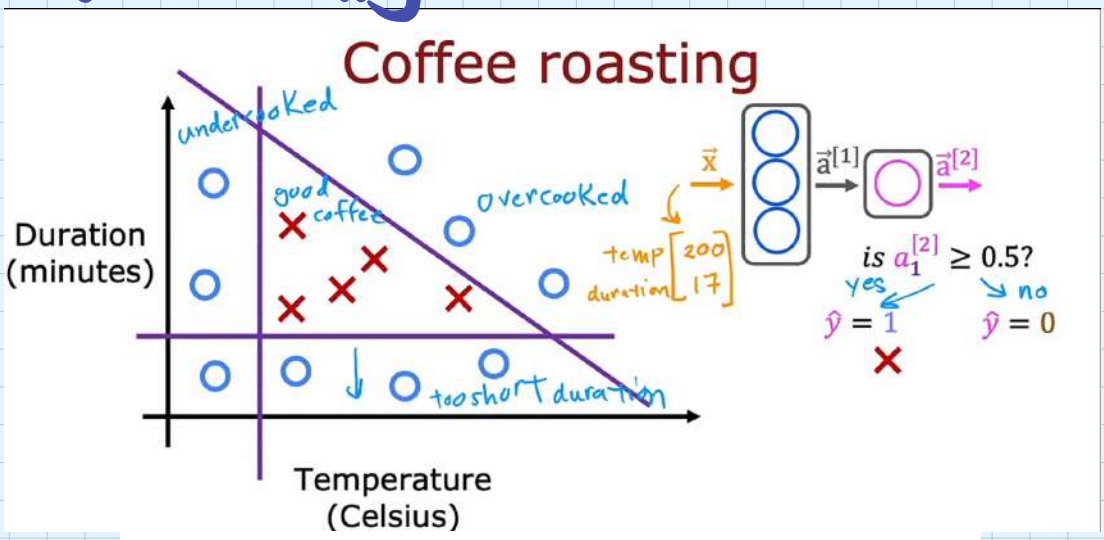


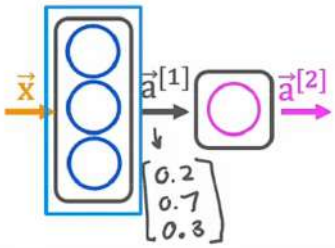
compute $\vec{a}^{[2]}$

Layer 2 $\Rightarrow$ $\vec{a}^{[2]} = \begin{bmatrix} g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}) \\ \vdots \\ g(\vec{w}_{15}^{[2]} \cdot \vec{a}^{[1]} + b_{15}^{[2]}) \end{bmatrix}$

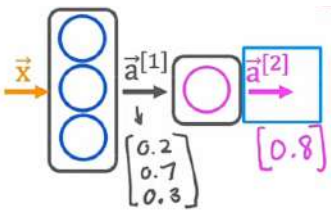


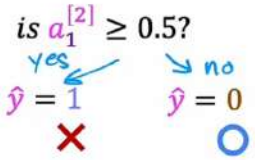
Tensorflow
↝ Coffee roasting






```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

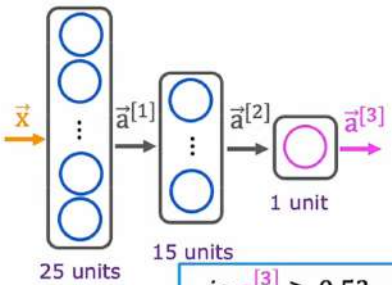




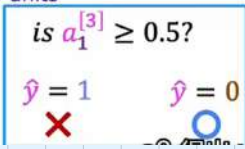

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)

layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

```
if a2 >= 0.5:
  yhat = 1
else:
  yhat = 0
```

Model for digit classification




```
x = np.array([[0.0,...245,...240...0]])
layer_1 = Dense(units=25, activation='sigmoid')
a1 = layer_1(x)

layer_2 = Dense(units=15, activation='sigmoid')
a2 = layer_2(a1)

layer_3 = Dense(units=1, activation='sigmoid')
a3 = layer_3(a2)
```




```
if a3 >= 0.5:
  yhat = 1
else:
  yhat = 0
```

Data in TensorFlow

Feature vectors

temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

```
x = np.array([[200.0, 17.0]])
[[200.0, 17.0]]
```

Why? two?

Note about numpy arrays

3 columns, 2 rows

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

2 x 3 matrix

```
x = np.array([[1, 2, 3],
              [4, 5, 6]])
[[1, 2, 3],
 [4, 5, 6]]
```

2D array

2 x 3

$$\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix}$$

4 rows, 2 columns

4 x 2 matrix

```
x = np.array([[0.1, 0.2],
              [-3.0, -4.0,],
              [-0.5, -0.6,],
              [7.0, 8.0,]])
[[0.1, 0.2],
 [-3.0, -4.0,],
 [-0.5, -0.6,],
 [7.0, 8.0,]]
```

4 x 2

2 x 1

```
x = np.array([[200, 17]])   → [200 17]      1 x 2
x = np.array([[200],        → [200]
              [17]])            [17]      2 x 1
```

matrix used in TF

2D array

```
x = np.array([200,17])
```

1D "vector"

~ 1D array

More flexible

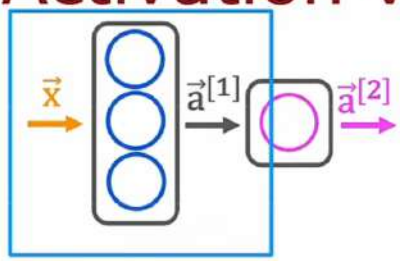
Feature vectors

temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...	...	...

```
x = np.array([[200.0, 17.0]])
[[200.0, 17.0]]
```

→ [200.0 17.0] 1 x 2

Activation vector




```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

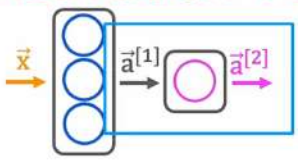
[[0.2, 0.7, 0.3]] 1 x 3 matrix

```
tf.Tensor([[0.2 0.7 0.3]], shape=(1, 3), dtype=float32)
```

```
a1.numpy()
```

```
array([[1.4661001, 1.125196 , 3.2159438]], dtype=float32)
```

Activation vector




```
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

[[0.8]] ← 1 x 1

```
tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

```
a2.numpy()
```

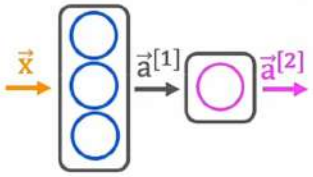
```
array([[0.8]], dtype=float32)
```

两种表示方式

Tensor
Numpy
} 可互换

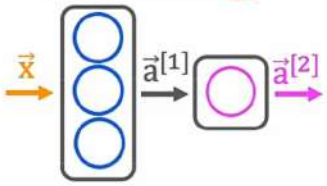
Building a neural network

What you saw earlier




```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation="sigmoid")
a1 = layer_1(x)

layer_2 = Dense(units=1, activation="sigmoid")
a2 = layer_2(a1)
```

Building a neural network architecture




```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])
```

layer1
layer2

		y
200	17	1
120	5	0
425	20	0
212	18	1

```
x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

4 x 2

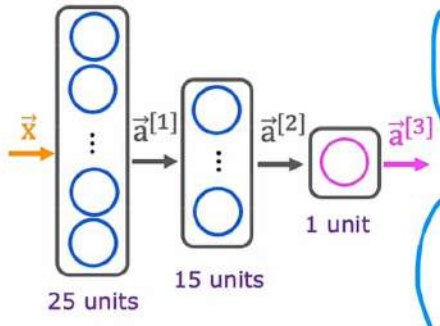
targets `y = np.array([1,0,0,1])`

```
model.compile(...)
model.fit(x,y)
```

← more about this next week!

```
model.predict(x_new)
```

Digit classification model

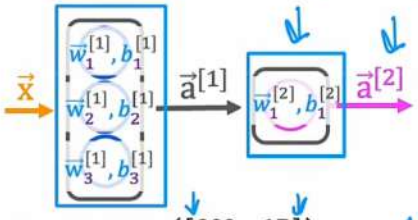



```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)
x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])
model.fit(x,y)
model.predict(x_new)
```

← more about this next week!

Forward prop in a single layer (just using Python & Numpy)

forward prop (coffee roasting model)



$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$

```
w2_1 = np.array([-7, 8])
b2_1 = np.array([3])
z2_1 = np.dot(w2_1,a1)+b2_1
a2_1 = sigmoid(z1_1)
```

$w_1^{[2]}$ w2_1

```
x = np.array([200, 17])
```

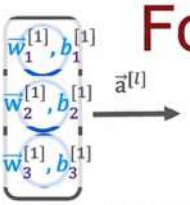
1D arrays

$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$ $a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$ $a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$

```
w1_1 = np.array([1, 2])      w1_2 = np.array([-3, 4])     w1_3 = np.array([5, -6])
b1_1 = np.array([-1])        b1_2 = np.array([1])         b1_3 = np.array([2])
z1_1 = np.dot(w1_1,x)+ b     z1_2 = np.dot(w1_2,x)+b      z1_3 = np.dot(w1_3,x)+b
a1_1 = sigmoid(z1_1)         a1_2 = sigmoid(z1_2)         a1_3 = sigmoid(z1_3)
```

General implement of forward propagation

Forward prop in NumPy



$\vec{w}_1^{[1]} = \begin{bmatrix}1\\2\end{bmatrix}$ $\vec{w}_2^{[1]} = \begin{bmatrix}-3\\4\end{bmatrix}$ $\vec{w}_3^{[1]} = \begin{bmatrix}5\\-6\end{bmatrix}$

```
W = np.array([
    [1, -3, 5]
    [2, 4, -6]])
```

2 by 3

$b_1^{[l]} = -1$ $b_2^{[l]} = 1$ $b_3^{[l]} = 2$

```
b = np.array([-1, 1, 2])
```

$\vec{a}^{[0]} = \vec{x}$

```
a_in = np.array([-2, 4])
```

```
def dense(a_in,W,b, g):
    units = W.shape[1]      [0,0,0]
    a_out = np.zeros(units)
    for j in range(units):  0,1,2
        w = W[:,j]
        z = np.dot(w,a_in) + b[j]
        a_out[j] = g(z)
    return a_out
```

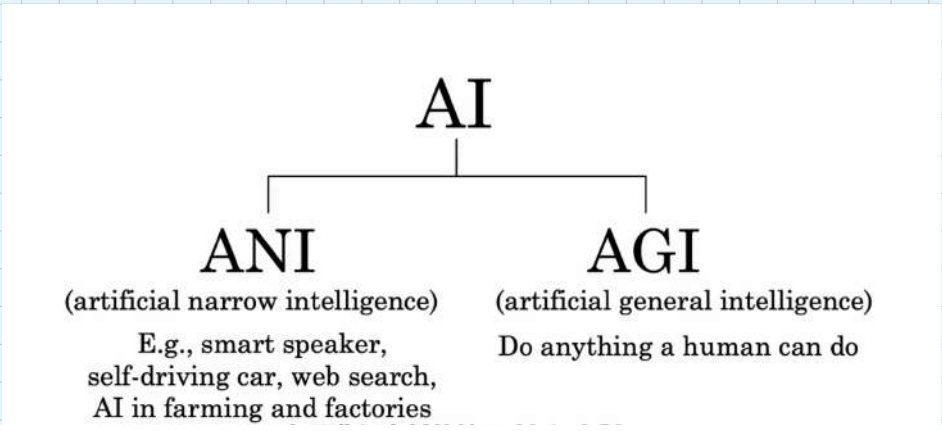
capital W refers to a matrix

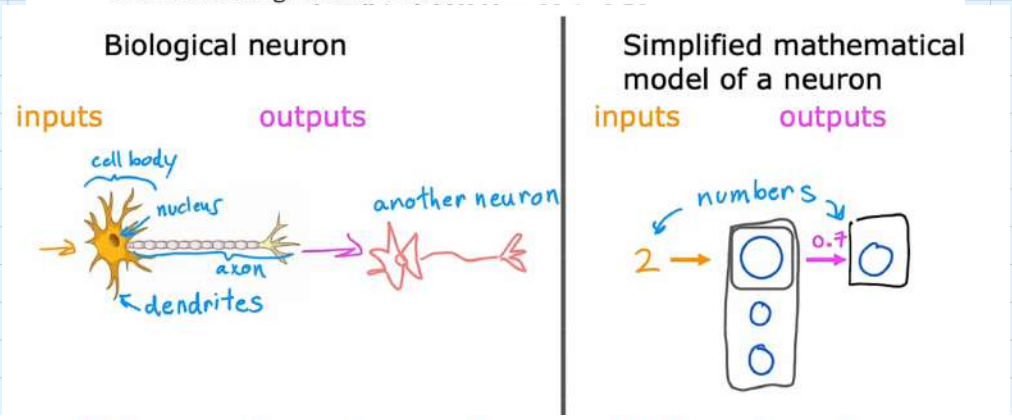
$a^{[1]}$ $a^{[1]}$

```
def sequential(x):
    a1 = dense(x, W1, b1)
    a2 = dense(a1, W2, b2)
    a3 = dense(a2, W3, b3)
    a4 = dense(a3, W4, b4)
    f_x = a4
    return f_x
```

→ basic physical meanings behind equation!

Speculations on AGZ (artificial general intelligence)



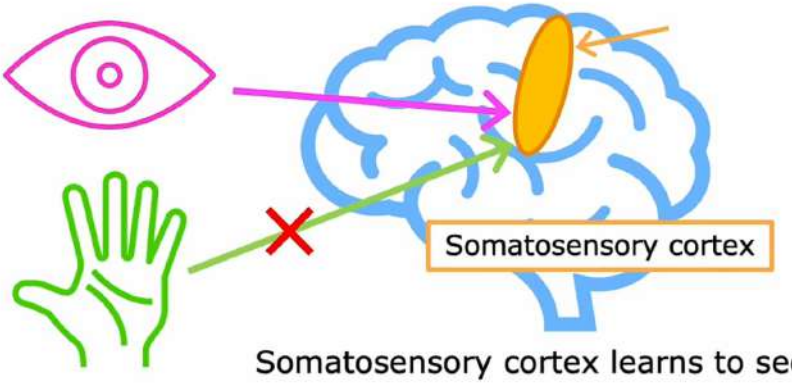


Neural network and the brain

Can we mimic the human brain?

We have (almost) no idea how the brain works

The "one learning algorithm" hypothesis



Somatosensory cortex learns to see

Sensor representations in the brain

Seeing with your tongue

Human echolocation (sonar)

Haptic belt: Direction sense

Implanting a 3rd eye

How neural networks are implemented efficiency
↳ caused matrix multiplication

For loops vs. vectorization

```
x = np.array([200, 17])

W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([-1, 1, 2])

def dense(a_in,W,b):
  a_out = np.zeros(units)
  for j in range(units):
    w = W[:,j]
    z = np.dot(w,x) + b[j]
    a[j] = g(z)
 return a
```

vectorized

```
X = np.array([[200, 17]])          2D array
W = np.array([[1, -3, 5],          same
              [-2, 4, -6]])
B = np.array([[-1, 1, 2]])         1x3 2D array

def dense(A_in,W,B):               all 2D arrays
  Z = np.matmul(A_in,W) + B        matrix multiplication
  A_out = g(Z)
  return A_out

[[1,0,1]]
```

[1,0,1]

matrix multiplication

Dot products

example: $\begin{bmatrix}1\\2\end{bmatrix} \cdot \begin{bmatrix}3\\4\end{bmatrix}$

$z = (1 \times 3) + (2 \times 4)$

$3 + 8$

11

in general: $\begin{bmatrix}\uparrow\\\vec{a}\\\downarrow\end{bmatrix} \cdot \begin{bmatrix}\uparrow\\\vec{w}\\\downarrow\end{bmatrix}$

$z = \vec{a} \cdot \vec{w}$

transpose: $\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}$, $\vec{a}^T = [1 \quad 2]$

vector vector multiplication: $\underset{1\times 2}{[\leftarrow \vec{a}^T \rightarrow]} \begin{bmatrix}\uparrow\\\vec{w}\\\downarrow\end{bmatrix}_{2\times 1}$

$z = \vec{a}^T \vec{w}$

equivalent

useful for understanding matrix multiplication

Vector matrix multiplication

$\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}$

$\vec{a}^T = [1 \quad 2]$ $\quad W = \begin{bmatrix}3 & 5\\4 & 6\end{bmatrix}$

$z = \vec{a}^T W$ (1 by 2) $\quad [\leftarrow \vec{a}^T \rightarrow] \begin{bmatrix}\uparrow & \uparrow\\\vec{w}_1 & \vec{w}_2\\\downarrow & \downarrow\end{bmatrix}$

$z = [\vec{a}^T \vec{w}_1 \quad a^T \vec{w}_2]$

$(1 * 3) + (2 * 4)$ $\quad$ $(1 * 5) + (2 * 6)$

$3 + 8$ $\quad$ $5 + 12$

11 $\quad$ 17

$z = [11 \quad 17]$

matrix matrix multiplication

$$A = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} \leftarrow & \vec{a}_1^T & \rightarrow \\ \leftarrow & \vec{a}_2^T & \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

rows columns

row 1 col 1 row 1 col 2

$$= \begin{bmatrix} \vec{a}_1^T \vec{w}_1 & \vec{a}_1^T \vec{w}_2 \\ \vec{a}_2^T \vec{w}_1 & \vec{a}_2^T \vec{w}_2 \end{bmatrix}$$

row 2 col 1 row 2 col 2

$(-1 \times 3) + (-2 \times 4)$
$-3 + -8$
-11

$(-1 \times 5) + (-2 \times 6)$
$-5 + -12$
-17

$$= \begin{bmatrix} 11 & 17 \\ -11 & -17 \end{bmatrix}$$

general rules for matrix multiplication
↳ next video!

Rules

Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

3 x 2 2 x 4

can only take dot products of vectors that are same length

[0.1 0.2] length 2

$\begin{bmatrix} 5 \\ 6 \end{bmatrix}$ length 2

3 by 4 matrix
↳ same # rows as A^T
↳ same # columns as W

In numpy

Matrix multiplication in NumPy

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

```
A=np.array([1,-1,0.1],
           [2,-2,0.2]])

AT=np.array([1,2],
            [-1,-2],
            [0.1,0.2])

AT=A.T
```
↳ transpose

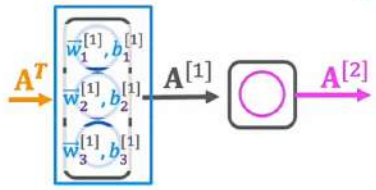
```
W=np.array([3,5,7,9],
           [4,6,8,0])
```

```
Z = np.matmul(AT,W)
```
or
```
Z = AT @ W
```

result

```
[[11,17,23,9],
 [-11,-17,-23,-9],
 [1.1,1.7,2.3,0.9]
]
```

Dense layer vectorized




```
A
AT = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([[-1, 1, 2]])
          a_in
def dense(AT,W,b,g):
  z = np.matmul(AT,W) + b
  a_out = g(z)        a_in
  return a_out
[[1,0,1]]
```

$A^T = [200 \quad 17]$ (1 × 2)

$W = \begin{bmatrix} 1 & -3 & 5 \\ -2 & 4 & -6 \end{bmatrix}$ (2 × 3)

$\vec{b} = [-1 \quad 1 \quad 2]$ (1 × 3)

$Z = A^T W + B$

$[165 \quad -531 \quad 900]$

$z_1^{[1]} \quad z_2^{[1]} \quad z_3^{[1]}$

$A = g(Z)$

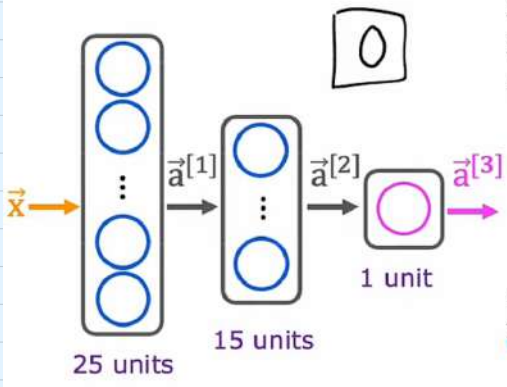
$[1 \quad 0 \quad 1]$

Optional Lab

Second Week

TensorFlow implementation

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)
```

epochs: number of steps in gradient descent

Training detail

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

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

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

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

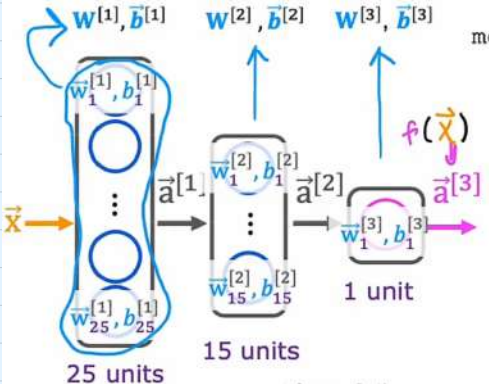
```
model.compile(
loss=BinaryCrossentropy())
```

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

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

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

Compare prediction vs. target

logistic loss

also known as binary cross entropy

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

$\mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]}$ $\vec{b}^{[1]}, \vec{b}^{[2]}, \vec{b}^{[3]}$

$f_{\mathbf{W},\mathbf{B}}(\vec{x})$

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

```
from tensorflow.keras.losses import
  BinaryCrossentropy
```

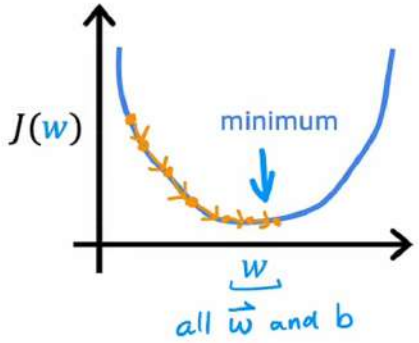
K Keras

regression (predicting numbers and not categories) — mean squared error

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

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

3. Gradient descent



all $\vec{w}$ and b

repeat {

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

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

} Compute derivatives for gradient descent using "back propagation"

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

nowadays，成熟了，用度较

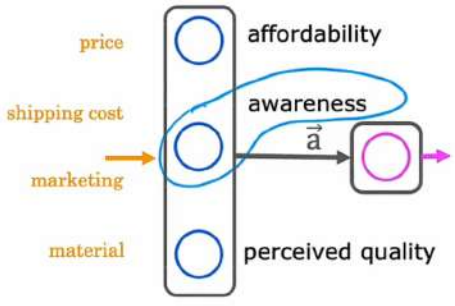
能用早之

PYTORCH → 主流

Alternative to the Sigmoid activation

Try probability

Demand Prediction Example



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

ReLU

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

注 activation function

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

"Softmax activation"

Linear activation function

$g(z) = z$

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

Sigmoid

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

$0 < g(z) < 1$

ReLU Rectified Linear Unit

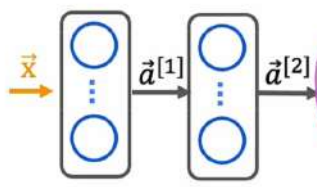
$g(z) = \max(0, z)$

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

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

Choosing activation functions

Output Layer 一看便迷



$\vec{a}^{[3]} = f(\vec{x})$

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

Choosing $g(z)$ for output layer?

Binary classification

Sigmoid

y=0/1

x_1	x_2	y
		0
		1
		0
		1

Regression

Linear activation function

y = +/-

x_1	x_2	y
		-1.2
		+1.5
		-0.4

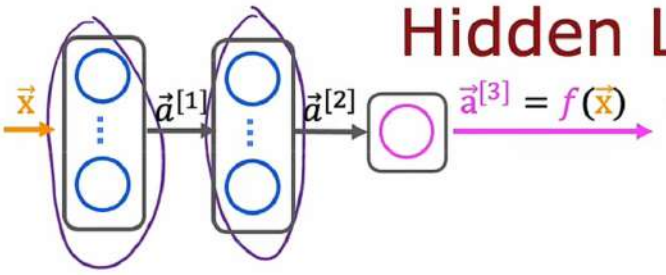
Regression

ReLU

y = 0 or +

x_1	x_2	y
		+150
		+420
		+230

Hidden Layer

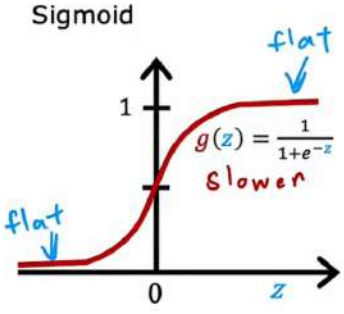


Choosing $g(z)$ for hidden layer

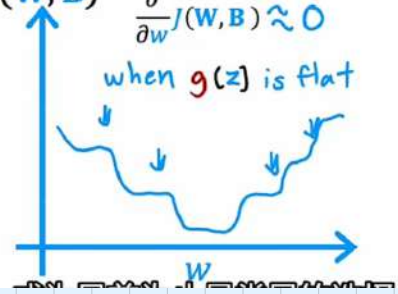
ReLU

most common choice

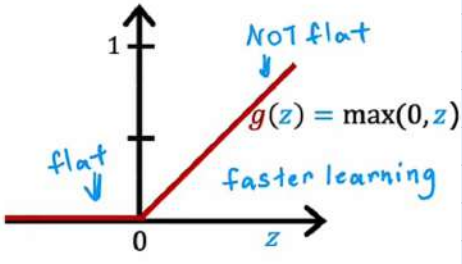
Sigmoid



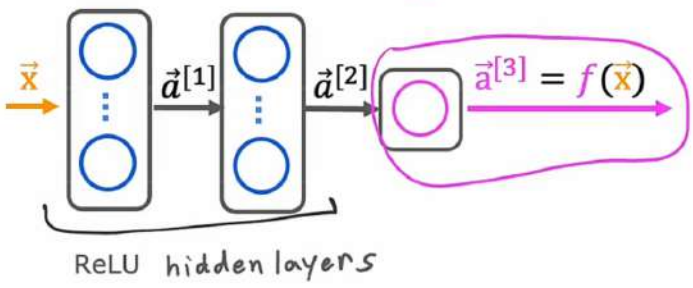
$J(W,B)$ $\frac{\partial}{\partial w} J(W,B) \approx 0$ when $g(z)$ is flat



ReLU faster



Choosing Activation Summary



ReLU hidden layers

binary classification
activation='sigmoid'

regression y negative/positive
activation='linear'

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'

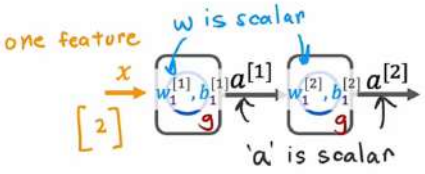
hidden → ReLU (SoftMax)
output → depends on y

또 뭐가 있지? tanh activation function or LeakyReLU or Swish

Why do we need Activation function

Why do we need activation functions?

Linear Example



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

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

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

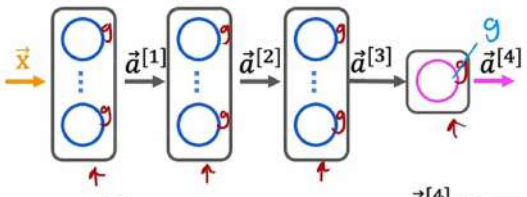
$$\vec{a}^{[2]} = (\underbrace{\vec{w}_1^{[2]} \vec{w}_1^{[1]}}_{w}) x + \underbrace{w_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b}$$

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

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

没卵用

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

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

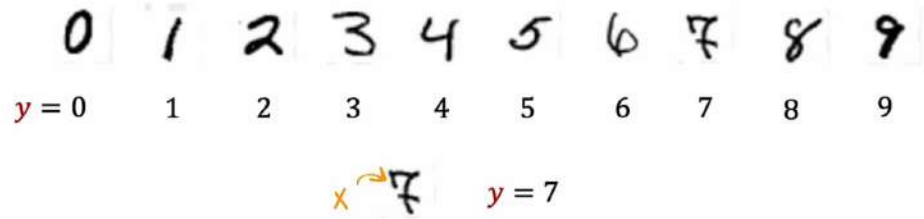
output activation is sigmoid
(hidden layers still linear)
↳ equivalent to logistic regression

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

ReLU 激活函数就可以了。

Multi Class

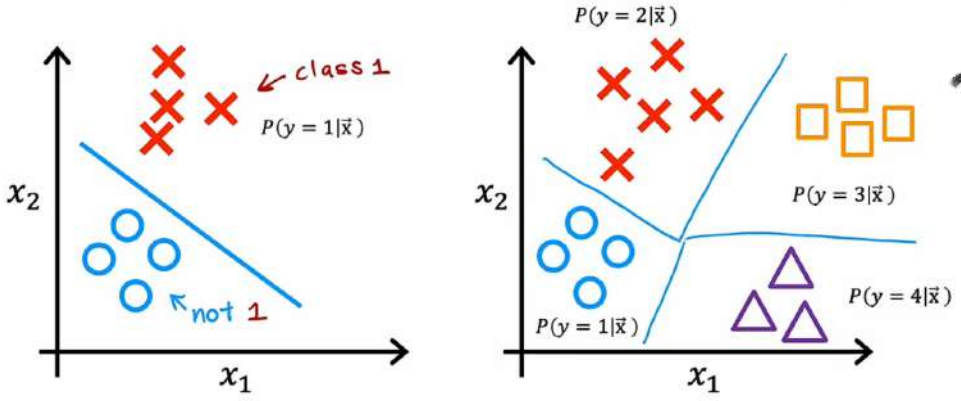
MNIST example



x → 7 $y = 7$

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

Multiclass classification example



Binary cannot solve
∴ use Softmax algorithm

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) $y = 1, 2, 3, 4$

$z_1 = \vec{w}_1 \cdot \vec{x} + b_1$ $\quad a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 1|\vec{x})$ 0.30

$z_2 = \vec{w}_2 \cdot \vec{x} + b_2$ $\quad a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 2|\vec{x})$ 0.20

$z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $\quad a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 3|\vec{x})$ 0.15

$z_4 = \vec{w}_4 \cdot \vec{x} + b_4$ $\quad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 4|\vec{x})$ 0.35

Cost

Logistic regression

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

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

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

$loss = -y \log a_1 - (1 - y) \log(1 - a_1)$

if $y = 1$ if $y = 0$

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

Softmax regression

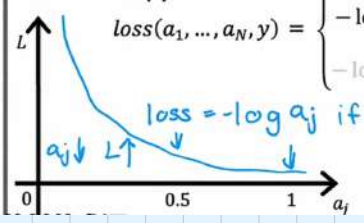
$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = 1|\vec{x})$

$\vdots$

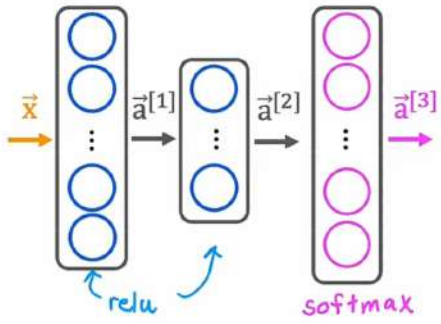
$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y = N|\vec{x})$

Crossentropy loss

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



Neural Network with Softmax output



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

$$\vdots$$

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

logistic regression

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

softmax

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

MNIST with softmax

① specify the model

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

② specify loss and cost

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

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

```
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)
```

Note: better (recommended) version later.

Don't use the version shown here!

Improved implementation of softmax (better way)

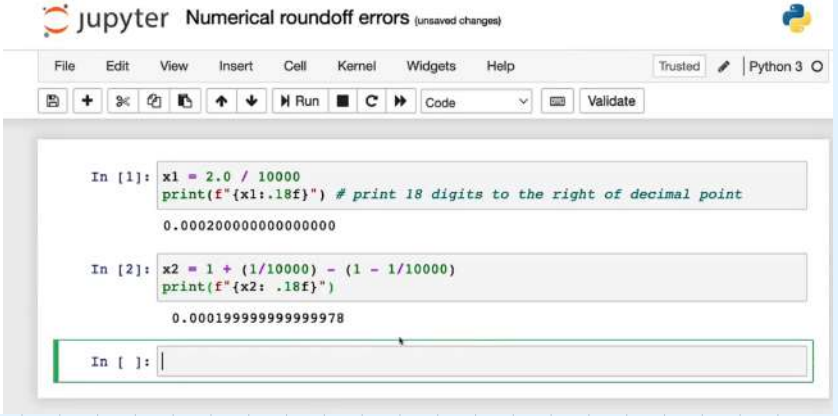
Numerical Roundoff Errors

option 1

$$x = \frac{2}{10{,}000}$$

option 2

$$x = \left(1 + \frac{1}{10{,}000}\right) - \left(1 - \frac{1}{10{,}000}\right) =$$



Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

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

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

```
model = Sequential([
  Dense(units=25, activation='relu')
  Dense(units=15, activation='relu')
  Dense(units=10, activation='sigmoid')
model.compile(loss=BinaryCrossEntropy() )
model.compile(loss=BinaryCrossEntropy(from_logits=True) )
```

'linear'

In softmax 上问题

More numerically accurate implementation of softmax

Softmax regression

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

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

More Accurate

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

```
model = Sequential([
  Dense(units=25, activation='relu')
  Dense(units=15, activation='relu')
  Dense(units=10, activation='softmax')
model.compile(loss=SparseCategoricalCrossEntropy() )
model.compile(loss=SparseCrossEntropy(from_logits=True) )
```

'linear'

→ become linear 无法算

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 $a_1, \ldots a_{10}$ 而 $z_1, \ldots z_{10}$

↑ z

Classification with multiple outputs 打多label

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

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

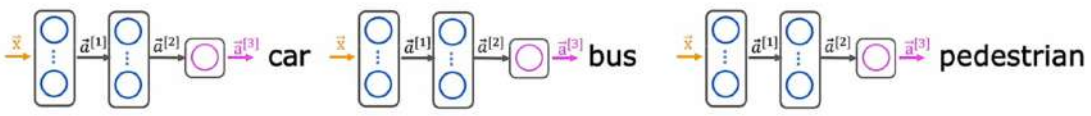
yes
yes
no

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

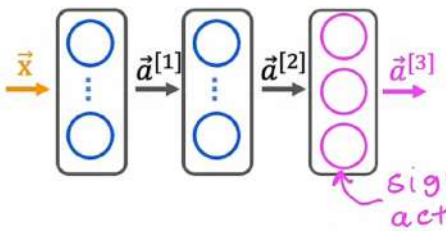
三个二分类问题

two ways → 1) build three neuron network
2) train simultaneously

Multiple classes



Alternatively, train one neural network with three outputs



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

Multi-label vs multiclass 差别

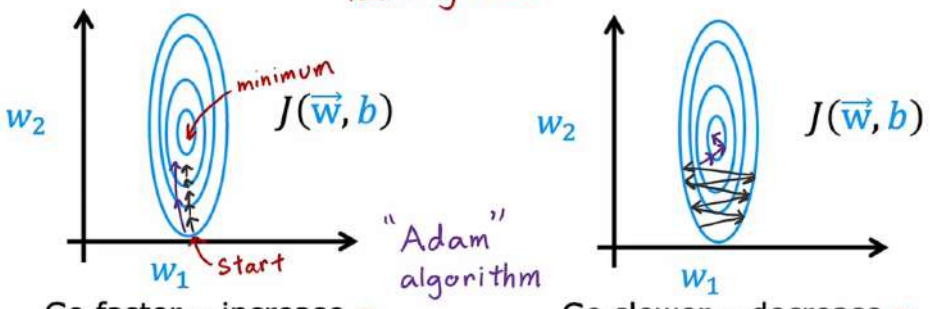
Advanced Optimization 高级优化方法

↳ train NN most faster than GD

Gradient Descent

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

α: learning rate



Go faster – increase α

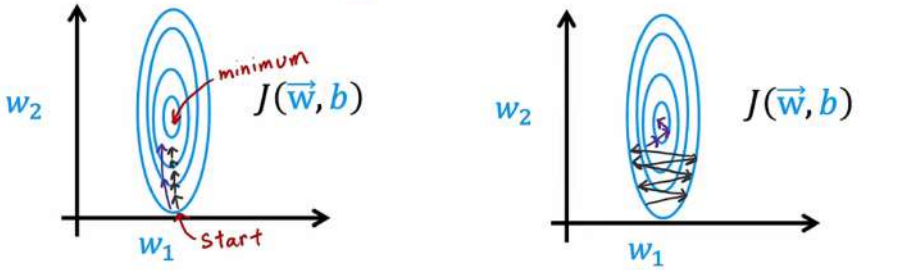
Go slower – decrease α

Adam Algorithm Intuition

Adam: Adaptive Moment estimation not just one α

$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{w}, b)$$
$$\vdots$$
$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$
$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{w}, b)$$

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.

~:深度学习

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

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

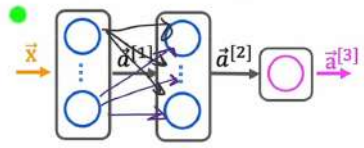
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

Additional Layer Types

Dense Layer



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

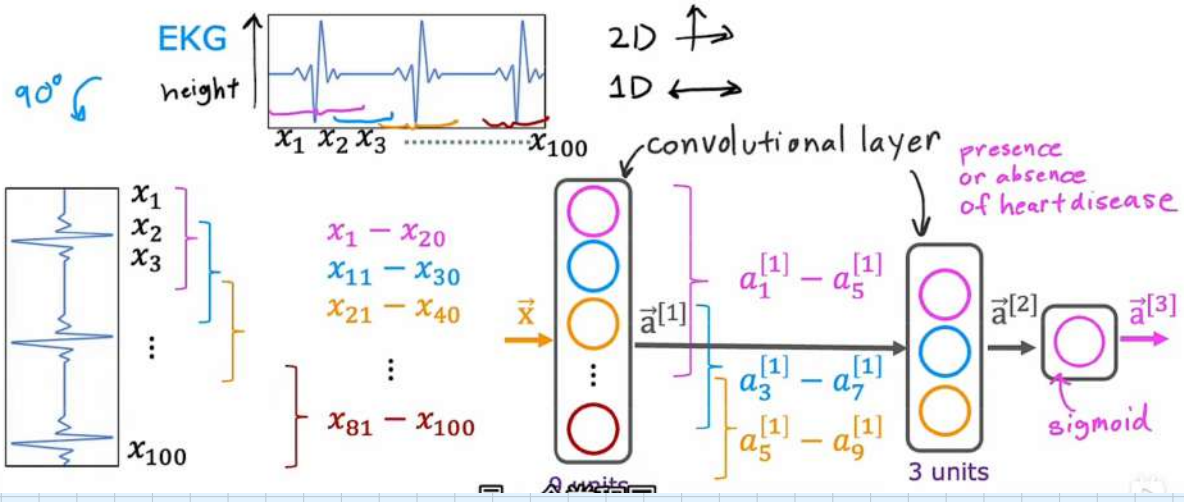
$$\vec{a}_1^{[2]} = g\left(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}\right)$$

Convolutional Layer

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

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

Convolutional Neural Network



Learn in Deep Learning

Optional Lab ~> see Jupyter

可选 ↝ What is a derivative 5.1

可选 ↝ Computation Graph 5.2

可选 ↝ Larger Neural Network Example 5.3

→ 这里快速带过

Third week

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(\vec{w}, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

But it makes unacceptably large errors in predictions. What do you try next?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$
- Try decreasing λ
- Try increasing λ

诊断

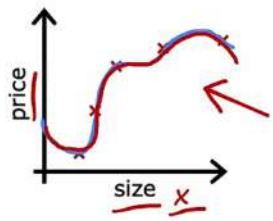
Machine learning diagnostic

Diagnostic: A test that you run to gain insight into what is/isn't working with a learning algorithm, to gain guidance into improving its performance.

Diagnostics can take time to implement but doing so can be a very good use of your time.

Evaluating a model

Evaluating your model



Model fits the training data well but will fail to generalize to new examples not in the training set.

$x = x_1$

$$f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + \cdots + w_nx^n + b$$

w_nx^4

just x

$f(\vec{x})$

- x_1 = size in feet²
- x_2 = no. of bedrooms
- x_3 = no. of floors
- x_4 = age of home in years

Evaluating your model

Dataset:

	size	price	
70%	2104	400	training set
	1600	330	
	2400	369	
	1416	232	
	3000	540	
	1985	300	
	1534	315	
30%	1427	199	test set
	1380	212	
	1494	243	

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
$\vdots$
$(x^{(m_{train})}, y^{m_{train}})$

m_{train} = no. training examples = 7

$(x_{test}^{(1)}, y_{test}^{(1)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. test examples = 3

Train/test procedure for linear regression (with squared error cost)

Fit parameters by minimizing cost function $J(\vec{w}, b)$

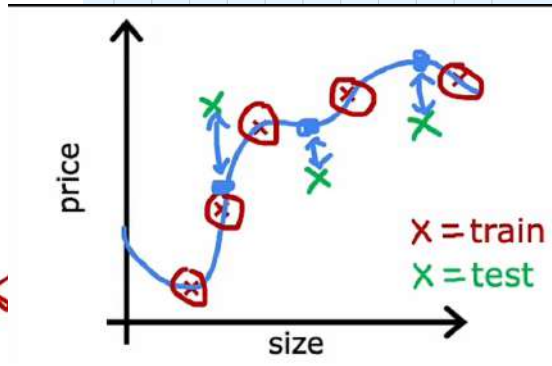
$$J(\vec{w}, b) = \min_{\vec{w}, b} \left[\frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2 \right]$$

Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right) - y_{test}^{(i)} \right)^2 \right]$$

Compute training error:

$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right) - y_{train}^{(i)} \right)^2 \right]$$



$J_{train}(\vec{w}, b)$ will be low

$J_{test}(\vec{w}, b)$ will be high

Train/test procedure for classification problem

Fit parameters by minimizing $J(\vec{w}, b)$ to find $\vec{w}, b$

E.g.,

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)}) \right) + (1 - y^{(i)}) \log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

Compute test error:

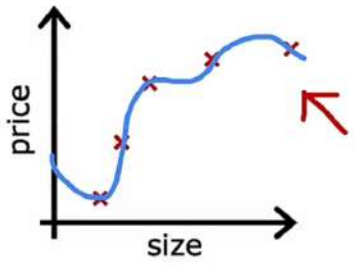
$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right) \right) + \left(1 - y_{test}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right) \right) \right]$$

Compute train error:

$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y_{train}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right) \right) + \left(1 - y_{train}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right) \right) \right]$$

Model selection / Cross-validation 交叉验证

Model selection (choosing a model)



$$f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$$

Once parameters $\vec{w}, b$ are fit to the training set, the training error $J_{train}(\vec{w}, b)$ is likely lower than the actual generalization error.

$J_{test}(\vec{w}, b)$ is better estimate of how well the model will generalize to new data than $J_{train}(\vec{w}, b)$.

Model selection (choosing a model)

d=1 1. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$ → $w^{<1>},b^{<1>}$ → $J_{test}(w^{<1>},b^{<1>})$

d=2 2. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + b$ → $w^{<2>},b^{<2>}$ → $J_{test}(w^{<2>},b^{<2>})$

d=3 3. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + w_3x^3 + b$ → $w^{<3>},b^{<3>}$ → $J_{test}(w^{<3>},b^{<3>})$

⋮

d=10 10. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + \cdots + w_{10}x^{10} + b$ → $J_{test}(w^{<10>},b^{<10>})$

Choose $w_1x_1 + \cdots + w_5x^5 + b$ d=5 $J_{test}(w^{<5>},b^{<5>})$

How well does the model perform? Report test set error $J_{test}(w^{<5>},b^{<5>})$?

The problem is $J_{test}(w^{<5>},b^{<5>})$ is likely to be an optimistic estimate of generalization error. Ie: An extra parameter d (degree of polynomial) was chosen using the test set.

w,b

Training/cross validation/test set

validation set
development set
→ dev set

size	price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

training set 60% → $(x^{(1)},y^{(1)}) \ \vdots \ (x^{(m_{train})},y^{(m_{train})})$ $m_{train} = 6$

cross validation 20% → $(x_{cv}^{(1)},y_{cv}^{(1)}) \ \vdots \ (x_{cv}^{(m_{cv})},y_{cv}^{(m_{cv})})$ $m_{cv} = 2$

test set 20% → $(x_{test}^{(1)},y_{test}^{(1)}) \ \vdots \ (x_{test}^{(m_{test})},y_{test}^{(m_{test})})$ $m_{test} = 2$

最简单的算法，但交叉验证很重要

Training/cross validation/test set

Training error: $J_{train}(\vec{w},b) = \frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2\right]$

Cross validation error: $J_{cv}(\vec{w},b) = \frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}}\left(f_{\vec{w},b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2\right]$ (validation error, dev error)

Test error: $J_{test}(\vec{w},b) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}}\left(f_{\vec{w},b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)}\right)^2\right]$

Model selection

d=1 1. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$ → $w^{<1>},b^{<1>}$ → $J_{cv}(w^{<1>},b^{<1>})$

d=2 2. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + b$ → $J_{cv}(w^{<2>},b^{<2>})$

d=3 3. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + w_3x^3 + b$

⋮

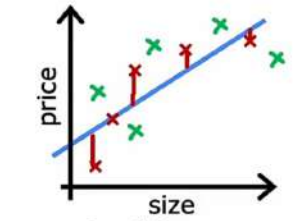
d=10 10. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + \cdots + w_{10}x^{10} + b$ → $J_{cv}(w^{<10>},b^{<10>})$

→ Pick $w_1x_1 + \cdots + w_4x^4 + b$ (J_{cv} $w^{<4>},b^{<4>}$)

Estimate generalization error using test the set: $J_{test}(w^{<4>},b^{<4>})$

Diagnosing bias and variance 偏差&方差

Bias/variance

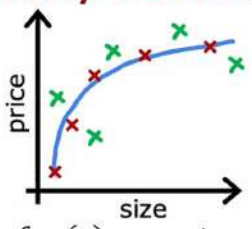


$f_{\vec{w},b}(x) = w_1x + b$

High bias (underfit)

J_{train} is high

J_{cv} is high

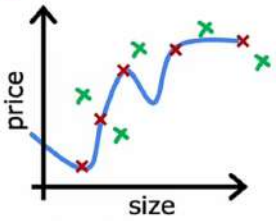


$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$

"Just right"

J_{train} is low

J_{cv} is low



$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

High variance (overfit)

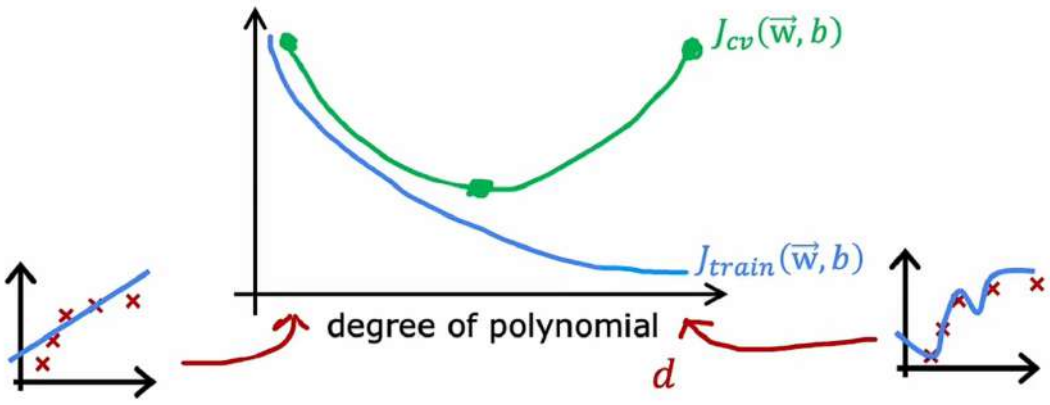
J_{train} is low

J_{cv} is high

→ High Variance

→ High bias

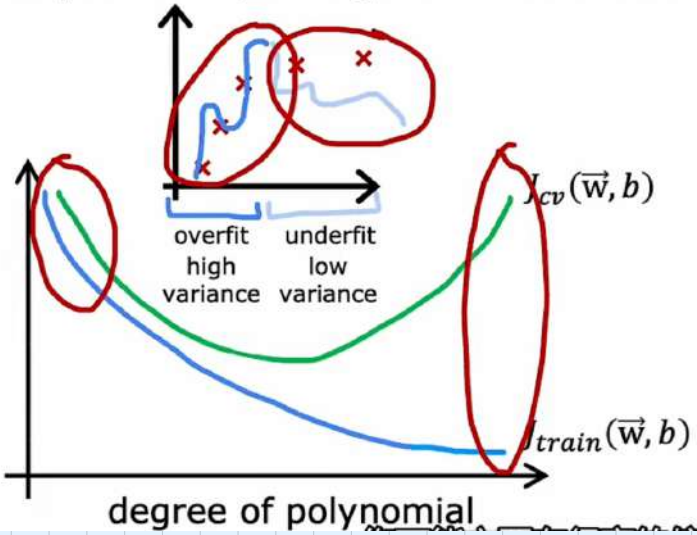
Understanding bias and variance



所以最好的是 Just right

Diagnosing bias and variance

How do you tell if your algorithm has a bias or variance problem?



High bias (underfit)

J_{train} will be high

($J_{train} \approx J_{cv}$)

High variance (overfit)

$J_{cv} \gg J_{train}$

(J_{train} may be be low)

High bias and high variance

J_{train} will be high

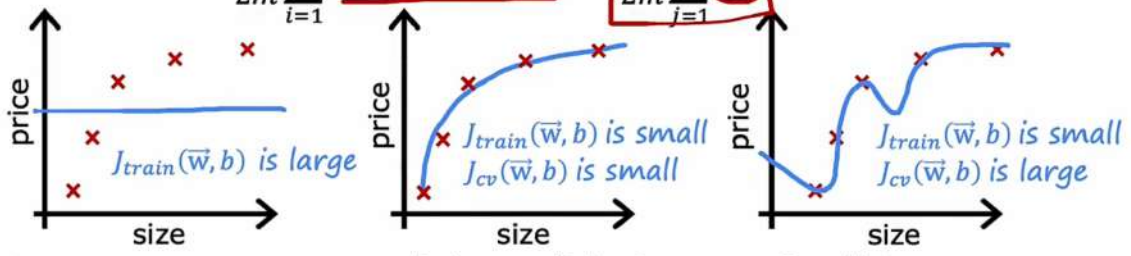
and $J_{cv} \gg J_{train}$

Regularization and bias variance 影响入取值

Linear regression with regularization

Model: $f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$



Large λ
High bias (underfit)
$\lambda = 10{,}000$ $w_1 \approx 0, w_2 \approx 0$
$f_{\vec{w},b}(\vec{x}) \approx b$

Intermediate λ

Small λ
High variance (overfit)
$\lambda = 0$

λ

Choosing the regularization parameter λ

Model: $f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

1. Try $\lambda = 0$ → $\min_{\vec{w},b} J(\vec{w},b)$ → $w^{<1>}, b^{<1>}$ → $J_{cv}(w^{<1>}, b^{<1>})$
2. Try $\lambda = 0.01$ → $w^{<2>}, b^{<2>}$ → $J_{cv}(w^{<2>}, b^{<2>})$
3. Try $\lambda = 0.02$ → $J_{cv}(w^{<3>}, b^{<3>})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $J_{cv}(w^{<5>}, b^{<5>})$

⋮

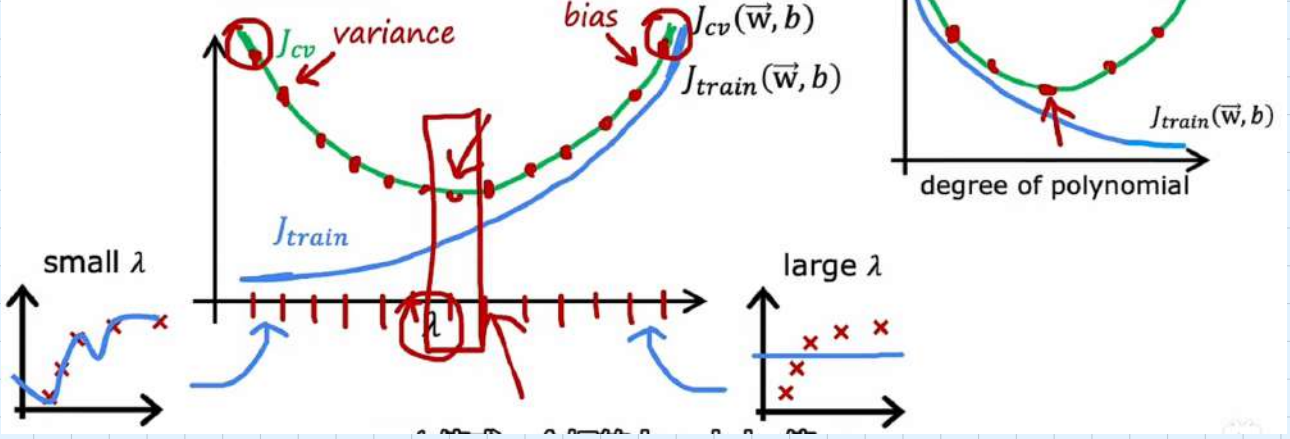
12. Try $\lambda \approx 10$ → $w^{<12>}, b^{<12>}$ → $J_{cv}(w^{<12>}, b^{<12>})$

Pick $w^{<5>}, b^{<5>}$

Report test error: $J_{test}(w^{<5>}, b^{<5>})$

Bias and variance as a function of regularization parameter λ

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$



Establishing a baseline level of performance

↳ 要找(决定)

Speech recognition example

又例如，要大量数据支持

Human level performance : 10.6%

Training error J_{train} : 10.8% — 0.2%

Cross validation error J_{cv} : 14.8% — 4.0%

→ 大

Establishing a baseline level of performance

What is the level of error you can reasonably hope to get to?

- Human level performance
- Competing algorithms performance
- Guess based on experience

Bias/variance examples

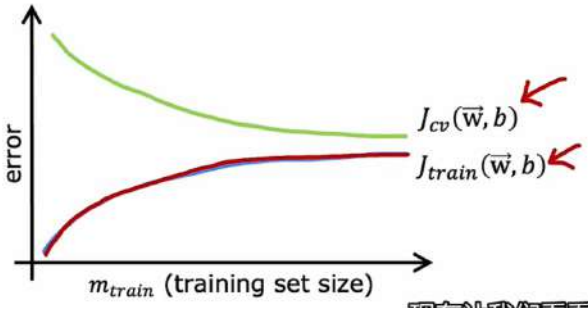
Baseline performance	10.6%	10.6%	10.6%
Training error (J_{train})	10.8% (0.2%)	15.0% (4.4%)	15.0% (4.4%)
Cross validation error (J_{cv})	14.8% (4.0%)	15.5% (0.5%)	19.7% (4.7%)
	high variance	high bias	high bias high variance

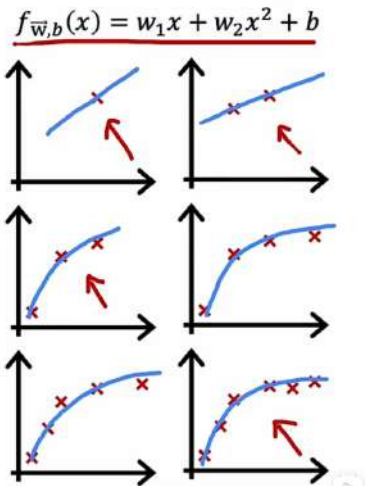
Learning curves (visualize) 并不常用

Learning curves

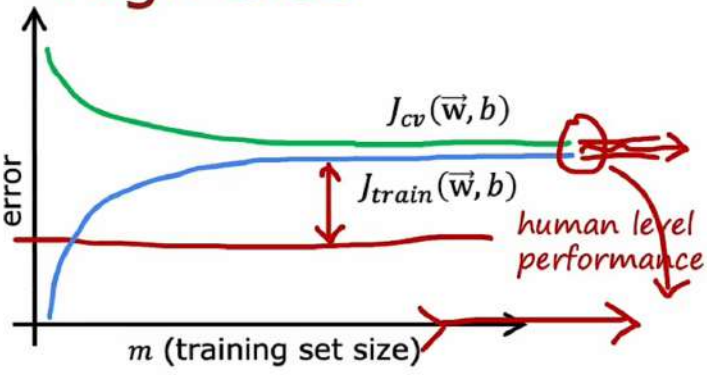
J_{train} = training error

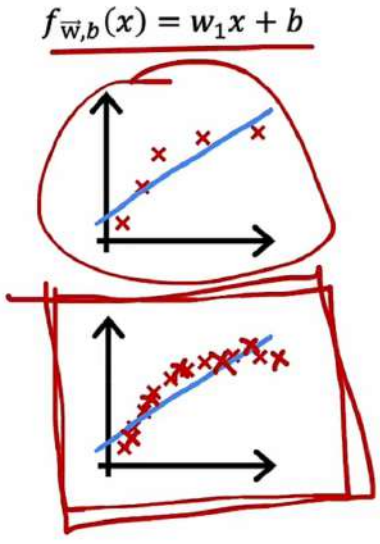
J_{cv} = cross validation error





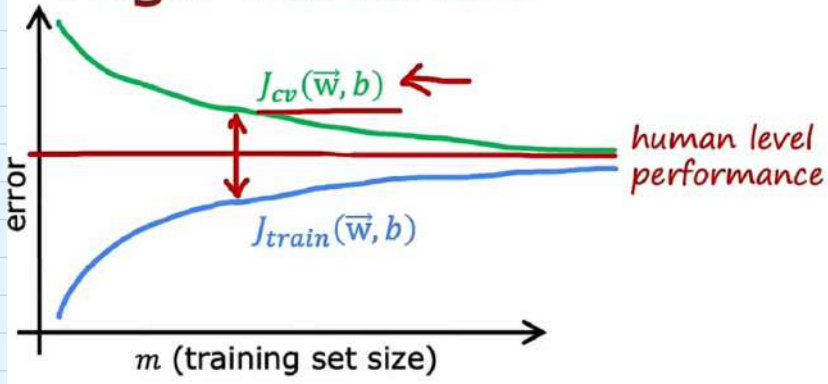
High bias

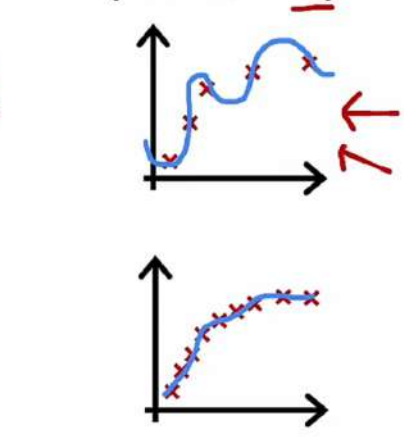




if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much.

High variance





if a learning algorithm suffers from high variance, getting more training data is likely to help.

Deciding what to try next revisited

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(\vec{w}, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

But it makes unacceptably large errors in predictions. What do you try next?

→ Get more training examples	fixes high variance
→ Try smaller sets of features $x, x^2, \ldots$	fixes high variance
→ Try getting additional features	fixes high bias
→ Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$	fixes high bias
→ Try decreasing λ	fixes high bias
→ Try increasing λ	fixes high variance

Bias and variance in neural networks

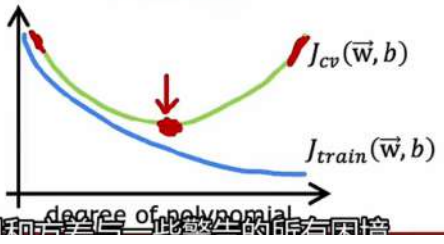
The bias variance tradeoff

$f_{\vec{w},b}(x) = w_1x + b$ — Simple model — High bias

$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$

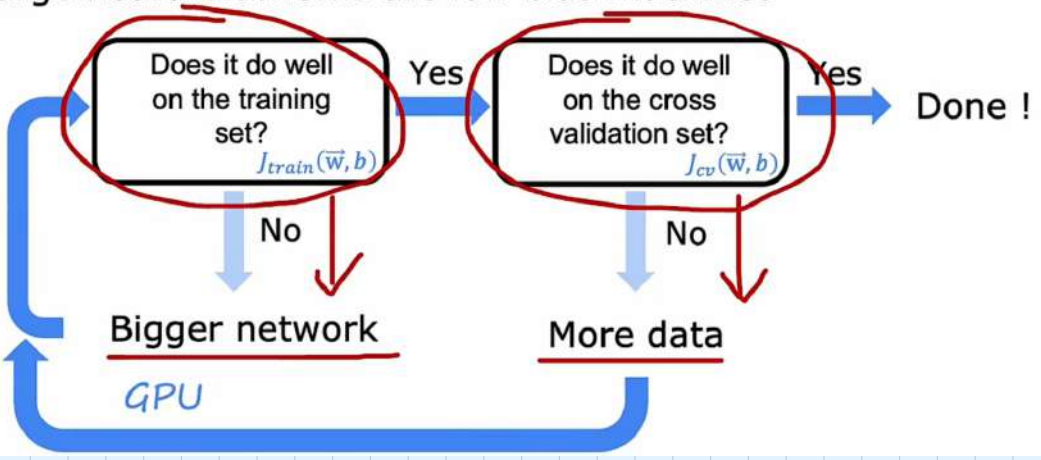
$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$ — Complex model — High variance

tradeoff

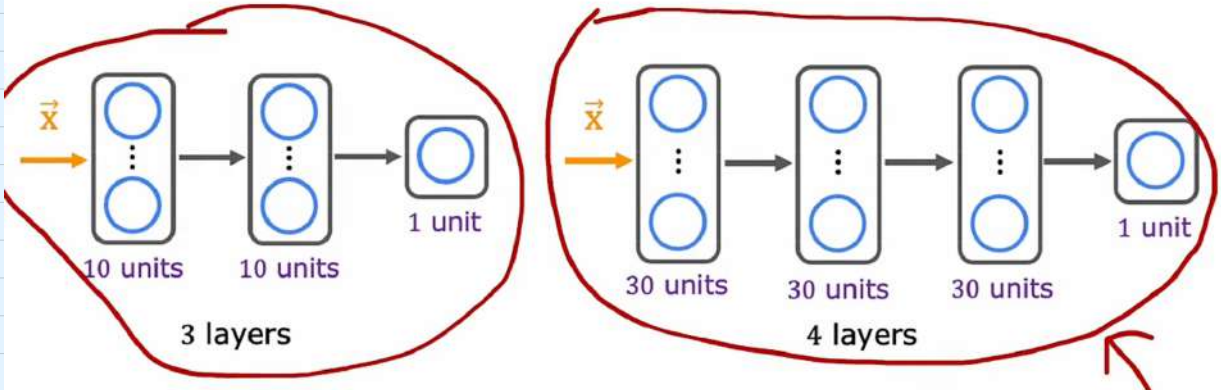


Neural networks and bias variance

Large neural networks are low bias machines



Neural networks and regularization



A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

Neural network regularization

$$J(\mathbf{W},\mathbf{B}) = \frac{1}{m}\sum_{i=1}^{m} L\left(f(\vec{x}^{(i)}), y^{(i)}\right) + \frac{\lambda}{2m}\sum_{\text{all weights } \mathbf{W}}(w^2)$$

b

Unregularized MNIST model

```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

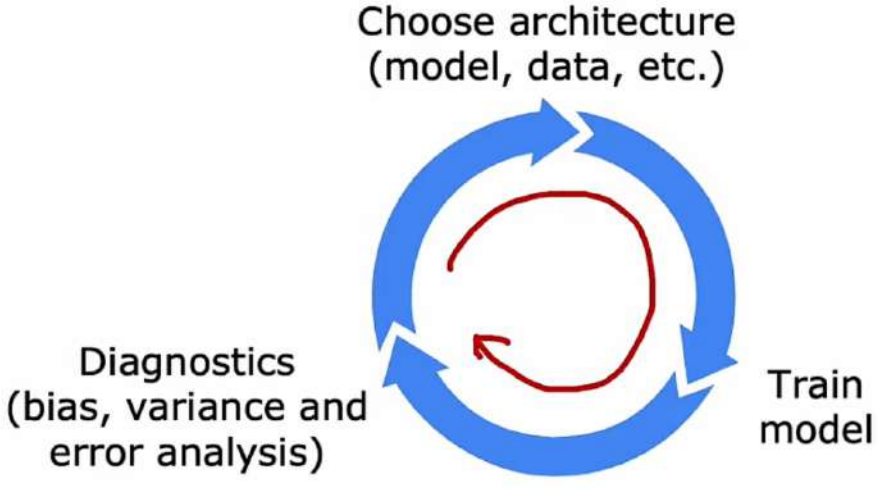
λ

Regularized MNIST model

```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

Iterative loop of ML development

Iterative loop of ML development



Spam classification example

From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.

From: Alfred Ng
To: Andrew Ng
Subject: Christmas dates?

Hey Andrew,
Was talking to Mom about plans
for Xmas. When do you get off
work. Meet Dec 22?
Alf

Building a spam classifier

Supervised learning: $\vec{x}$ = features of email
y = spam (1) or not spam (0)

Features: list the top 10,000 words to compute $x_1, x_2, \cdots, x_{10,000}$

$$\vec{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} a \\ andrew \\ buy \\ deal \\ discount \\ \vdots \end{matrix}$$

From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.

Building a spam classifier

How to try to reduce your spam classifier's error?

- Collect more data. E.g., "Honeypot" project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body. E.g., should "discounting" and "discount" be treated as the same word.
- Design algorithms to detect misspellings. E.g., w4tches, med1cine, m0rtgage.

不是最有成果的方向。
not be the most fruitful direction.

Error analysis for case above

Error analysis

m_{cv} = ~~500~~ 5000 examples in cross validation set.

Algorithm misclassifies ~~100~~ 1000 of them.

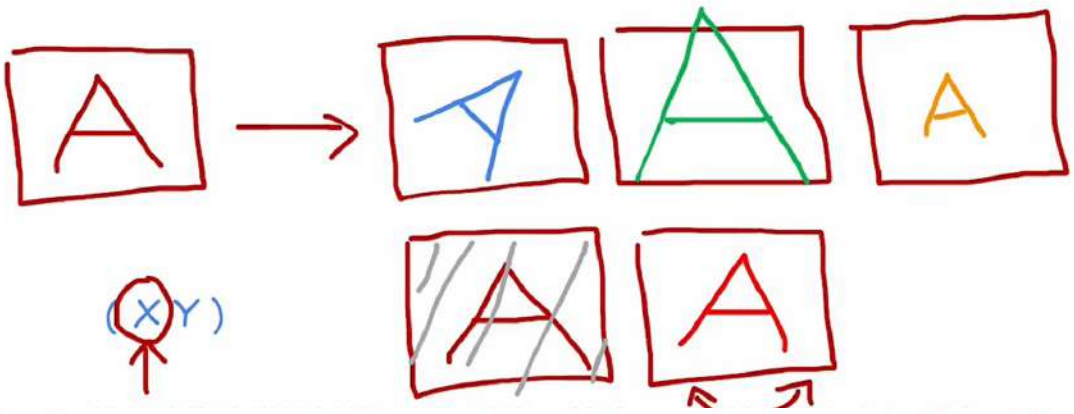
Manually examine 100 examples and categorize them based on common traits.

- Pharma: 21 → more data features
- Deliberate misspellings (w4tches, med1cine): 3
- Unusual email routing: 7
- Steal passwords (phishing): 18 → more data features
- Spam message in embedded image: 5

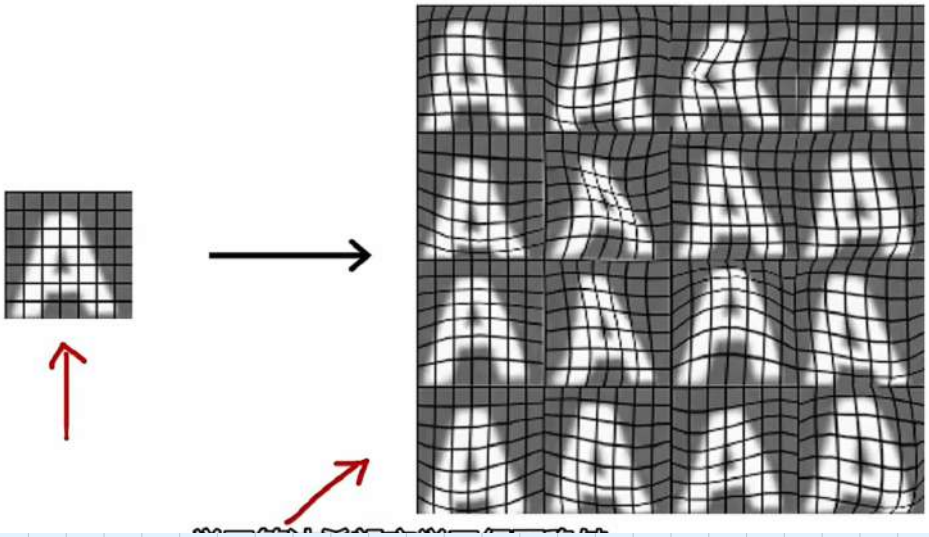
Adding data

Data augmentation

Augmentation: modifying an existing training example to create a new training example.



Data augmentation by introducing distortions



Data augmentation for speech

Speech recognition example

Original audio (voice search: "What is today's weather?")

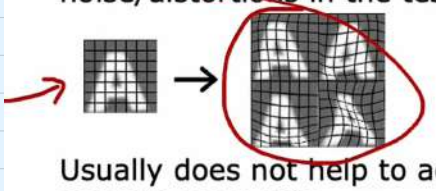
+ Noisy background: Crowd

+ Noisy background: Car

+ Audio on bad cellphone connection

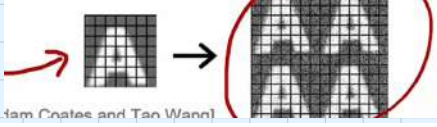
Data augmentation by introducing distortions

Distortion introduced should be representation of the type of noise/distortions in the test set.



Audio:
Background noise,
bad cellphone connection

Usually does not help to add purely random/meaningless noise to your data.



x_i=intensity (brightness) of pixel i
$x_i \leftarrow x_i$ + random noise

加数据
增强
鲁棒性

Data synthesis

Synthesis: using artificial data inputs to create a new training example.

Artificial data synthesis for photo OCR

Artificial data synthesis for photo OCR

Real data

Synthetic data

Engineering the data used by your system

Conventional model-centric approach:

AI = Code + Data
(algorithm/model)

Work on this

Data-centric approach:

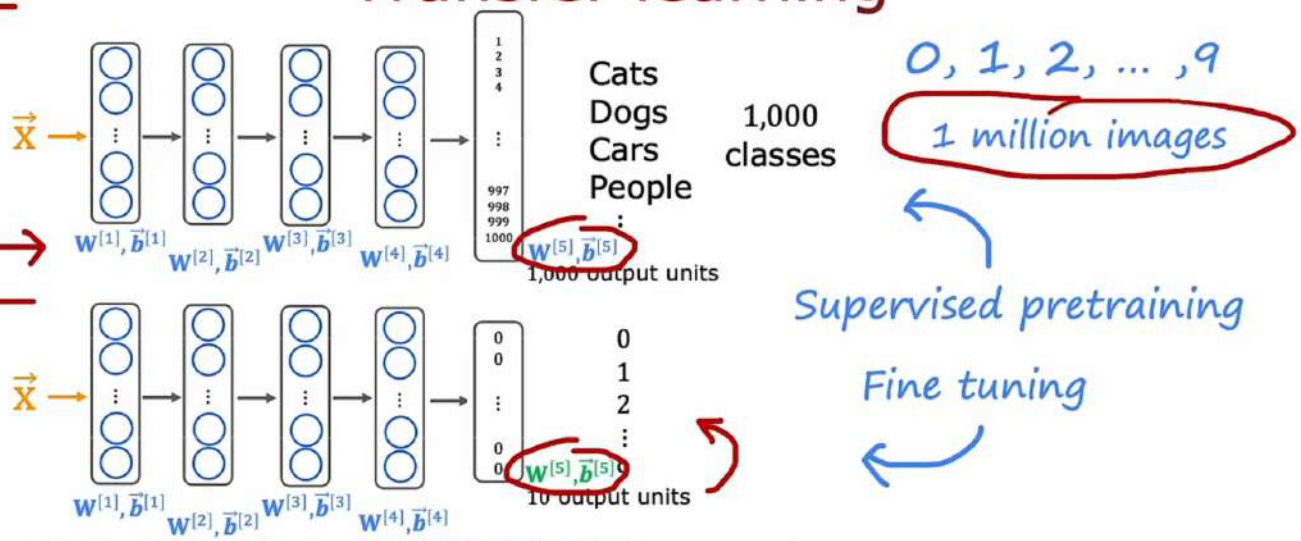
AI = Code + Data
(algorithm/model)

Work on this

Transfer Learning 迁移学习

与没那么多数据时可用

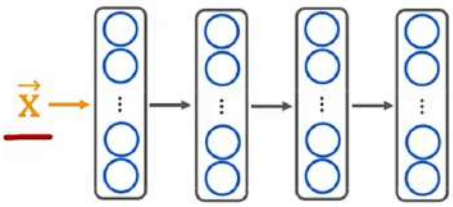
Transfer learning



Option 1: only train output layers parameters.

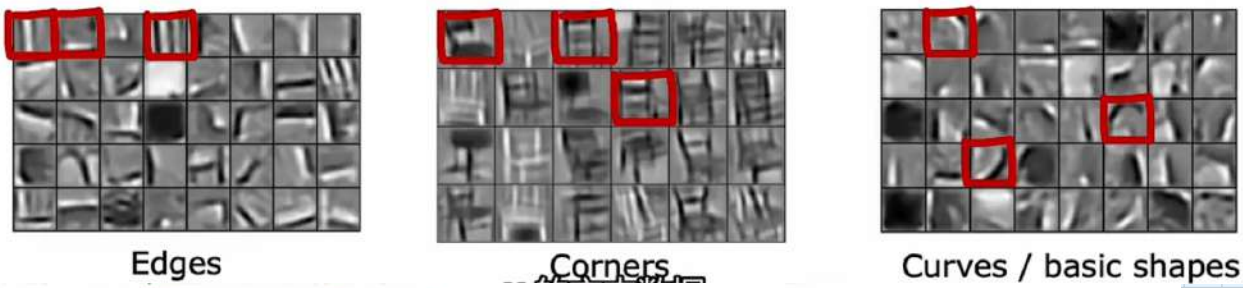
Option 2: train all parameters.

Why does transfer learning work?



use the same input type

detects edges | detects corners | detects curves/basic shapes



Edges | Corners | Curves / basic shapes

Transfer learning summary

1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own).

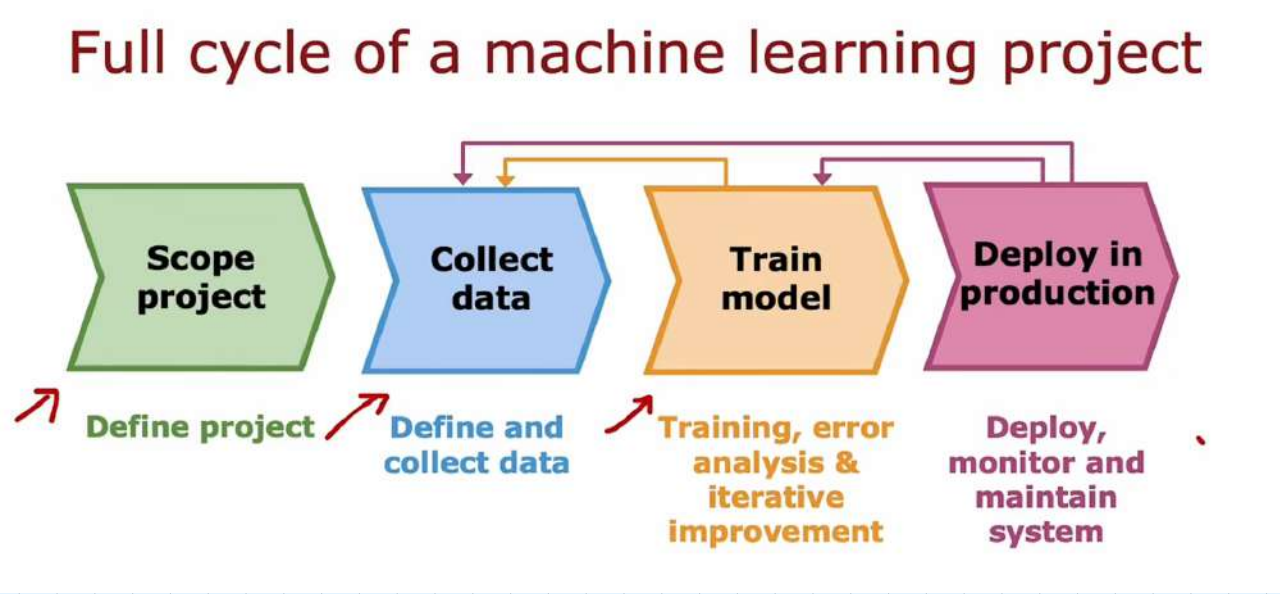
1M

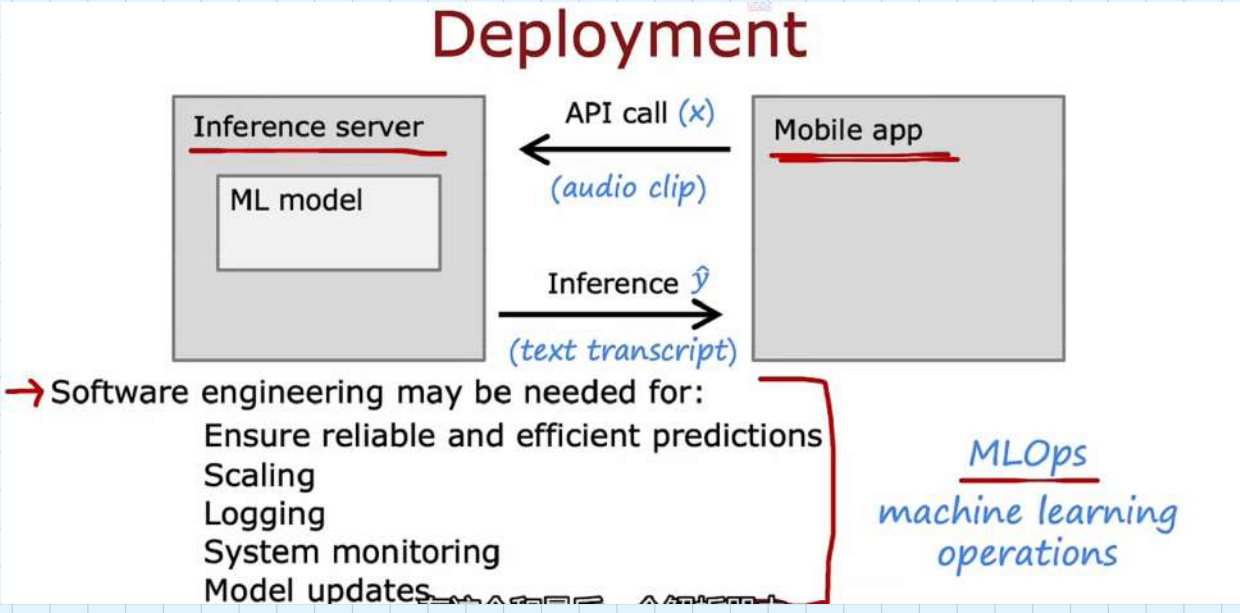
2. Further train (fine tune) the network on your own data.

1000

50

Full cycle of machine Learning





Fairness, bias and ethics (伦理)

Bias

Hiring tool that discriminates against women.

Facial recognition system matching dark skinned individuals to criminal mugshots.

Biased bank loan approvals.

Toxic effect of reinforcing negative stereotypes.

Adverse use cases

Deepfakes

Spreading toxic/incendiary speech through optimizing for engagement.

Generating fake content for commercial or political purposes.

Using ML to build harmful products, commit fraud etc.

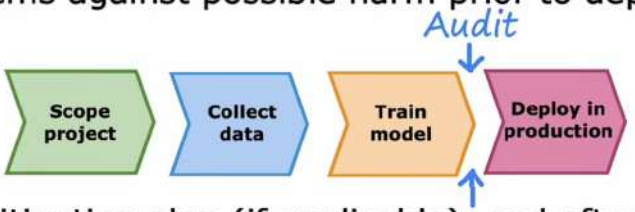
Spam vs anti-spam : fraud vs anti-fraud.

Guidelines

Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.

Carry out literature search on standards/guidelines for your industry.

Audit systems against possible harm prior to deployment.



Develop mitigation plan (if applicable), and after deployment, monitor for possible harm.

Error metrics for skewed datasets 偏斜数据集的误差度量

Rare disease classification example

Train classifier $f_{\vec{w},b}(\vec{x})$ ($y = 1$ if disease present, $y = 0$ otherwise)

Find that you've got 1% error on test set (99% correct diagnoses)

Only 0.5% of patients have the disease

```
print("y=0")
```

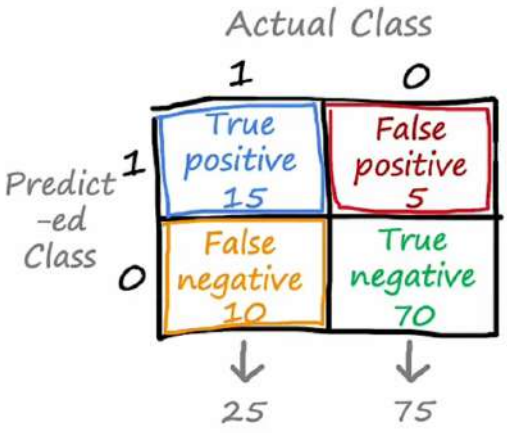
99.5% accuracy, 0.5% error ←

1%

1.2%

Precision/recall

$y = 1$ in presence of rare class we want to detect.



Precision:
(of all patients where we predicted $y = 1$, what fraction actually have the rare disease?)

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False pos}} = \frac{15}{15+5} = 0.75$$

Recall:
(of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

$$\frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15+10} = 0.6$$

Trading off precision and Recall

在曲线开始急剧下降之前选择一个点

Trading off precision and recall

Logistic regression: $0 < f_{\vec{w},b}(\vec{x}) < 1$

Predict 1 if $f_{\vec{w},b}(\vec{x}) \geq 0.5$ (0.7) (0.3)

Predict 0 if $f_{\vec{w},b}(\vec{x}) < 0.5$ (0.7) (0.3)

$$\text{precision} = \frac{\text{true positives}}{\text{total predicted positive}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{total actual positive}}$$

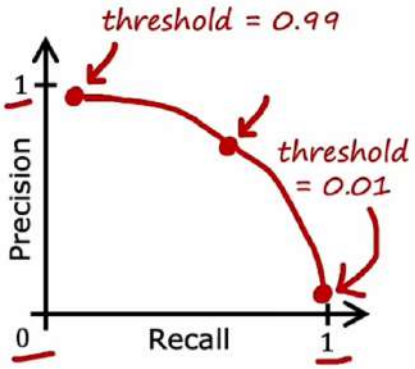
Suppose we want to predict $y = 1$ (rare disease) only if very confident.

→ higher precision, lower recall.

Suppose we want to avoid missing too many cases of rare disease (when in doubt predict $y = 1$)

→ lower precision, higher recall.

More generally predict 1 if: $f_{\vec{w},b}(\vec{x}) \geq$ threshold.



这边~并联哈哈哈

F1 score

越高越好

How to compare precision/recall numbers?

	Precision (P)	Recall (R)	Average	F_1 score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.501	0.0392

print("y=1")

Average $= \frac{P+R}{2}$

$$F1\ score = \frac{1}{\frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)} = 2\frac{PR}{P+R}$$

Fourth

Decision Tree Model

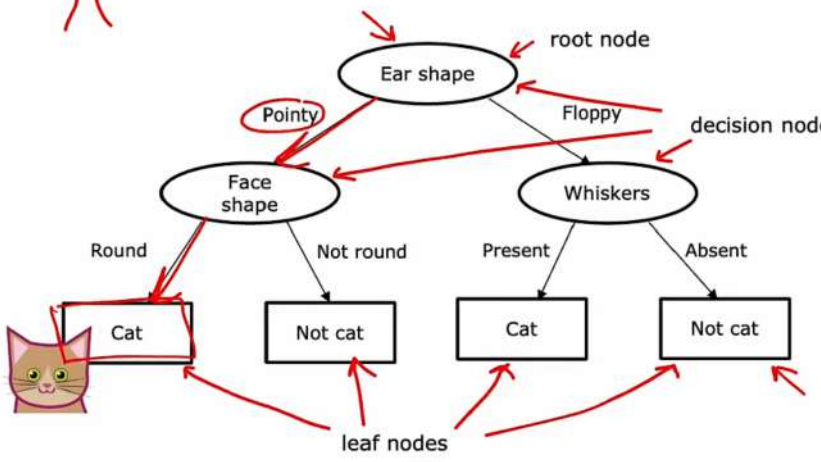
Cat classification example

	Ear shape (x_1)	Face shape (x_2)	Whiskers	Cat
	Pointy	Round	Present	1
	Floppy	Not round	Present	1
	Floppy	Round	Absent	0
	Pointy	Not round	Present	0
	Pointy	Round	Present	1
	Pointy	Round	Absent	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0

Categorical (discrete values)

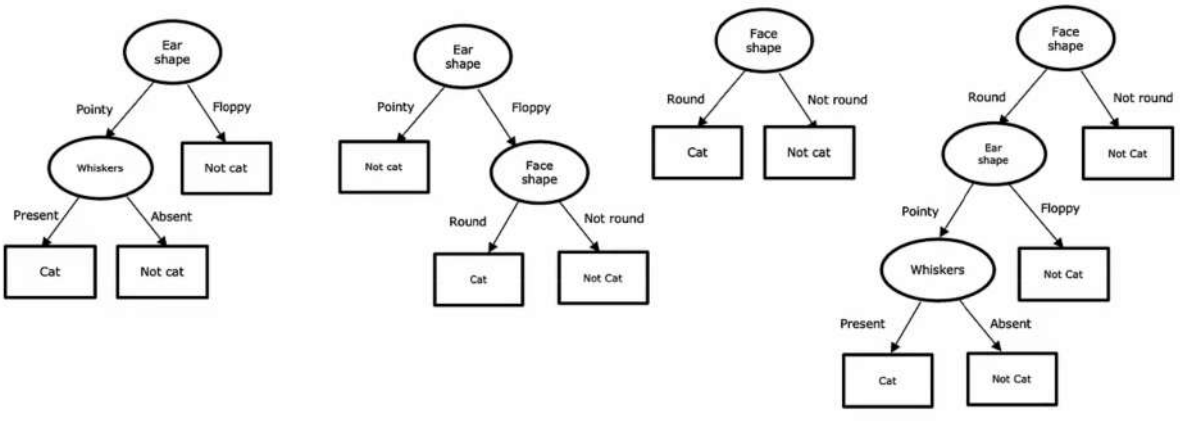
Decision Tree

New test example



Ear shape: Pointy
Face shape: Round
Whiskers: Present

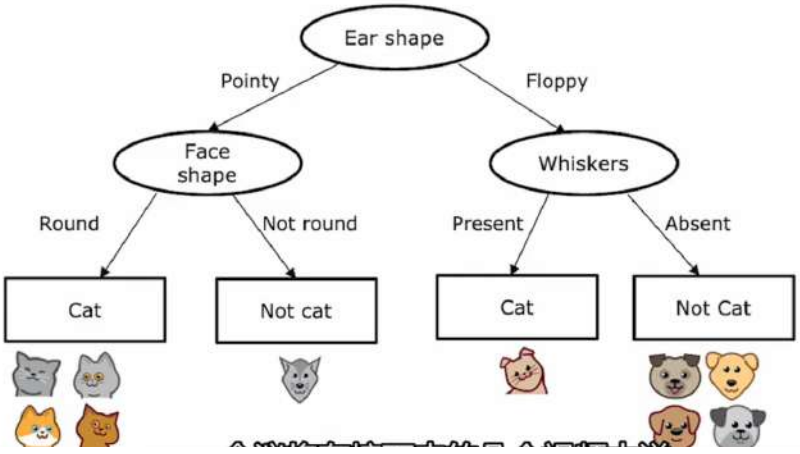
Decision Tree



在训练集上表现良好的人

Learning Process

Decision Tree Learning

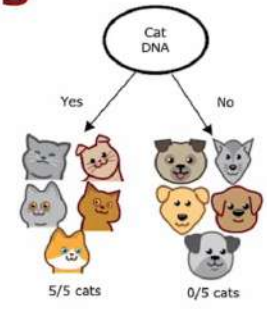


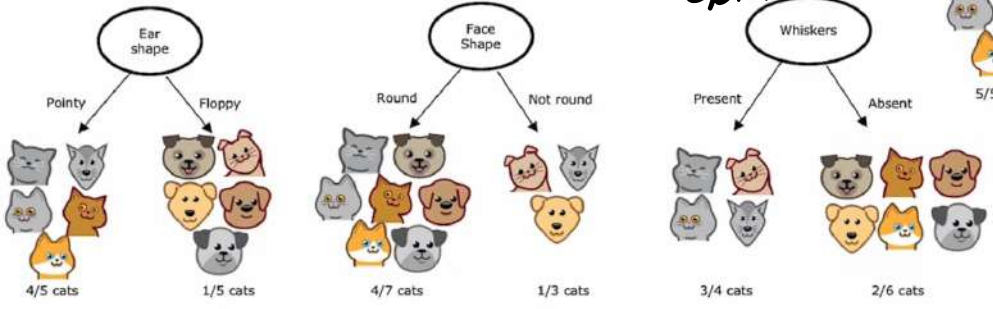
Decision Tree Learning

Decision 1: How to choose what feature to split on at each node?

Maximize purity (or minimize impurity)

→ calc Information Gain

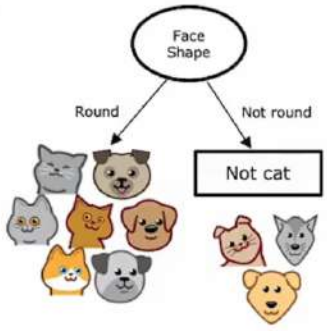




Decision Tree Learning

Decision 2: When do you stop splitting? → 剪枝

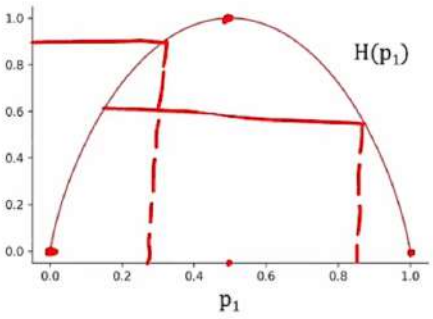
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold



Measuring purity

Entropy as a measure of impurity

p_1 = fraction of examples that are cats



$p_1 = 0 \quad H(p_1) = 0$

$p_1 = 2/6 \quad H(p_1) = 0.92$

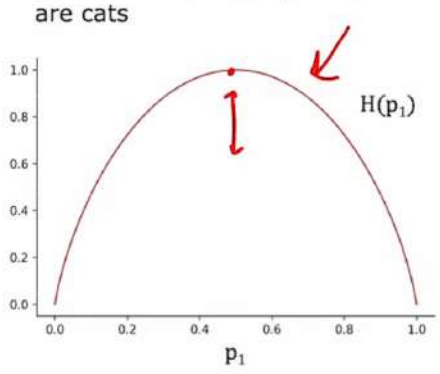
$p_1 = 3/6 \quad H(p_1) = 1$

$p_1 = 5/6 \quad H(p_1) = 0.65$

$p_1 = 6/6 \quad H(p_1) = 0$

Entropy as a measure of impurity

p_1 = fraction of examples that are cats



$$p_0 = 1 - p_1$$

$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

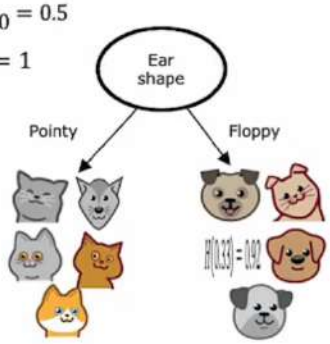
Note: "0 log(0)" = 0

Choosing a split: Information Gain

Choosing a split

$p_1 = 5/10 = 0.5$

$H(0.5) = 1$



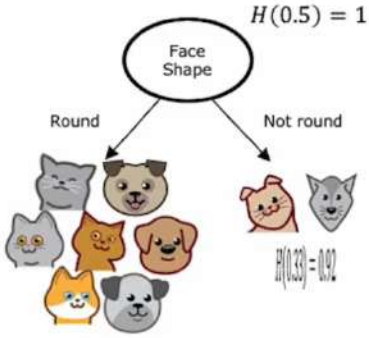
$p_1 = 4/5 = 0.8 \quad p_1 = 1/5 = 0.2$

$H(0.8) = 0.72 \quad H(0.2) = 0.72$

$$H(0.5) - \left(\frac{5}{10}H(0.8) + \frac{5}{10}H(0.2)\right)$$

$= 0.28$

$H(0.5) = 1$



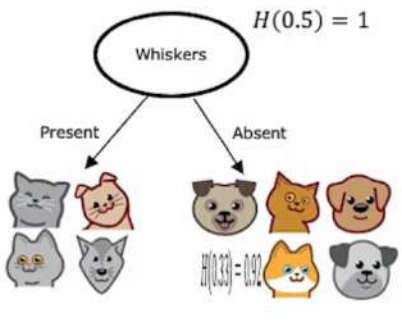
$p_1 = 4/7 = 0.57 \quad p_1 = 1/3 = 0.33$

$H(0.57) = 0.99 \quad H(0.33) = 0.92$

$$H(0.5) - \left(\frac{7}{10}H(0.57) + \frac{3}{10}H(0.33)\right)$$

$= 0.03$

$H(0.5) = 1$



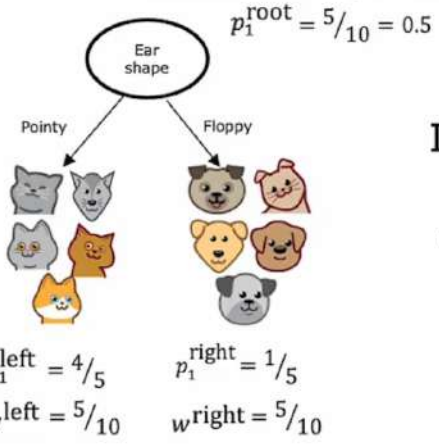
$p_1 = 3/4 = 0.75 \quad p_1 = 2/6 = 0.33$

$H(0.75) = 0.81 \quad H(0.33) = 0.92$

$$H(0.5) - \left(\frac{4}{10}H(0.75) + \frac{6}{10}H(0.33)\right)$$

$= 0.12$

Information Gain



$p_1^{root} = 5/10 = 0.5$

$p_1^{left} = 4/5$ $\quad p_1^{right} = 1/5$

$w^{left} = 5/10$ $\quad w^{right} = 5/10$

Information gain

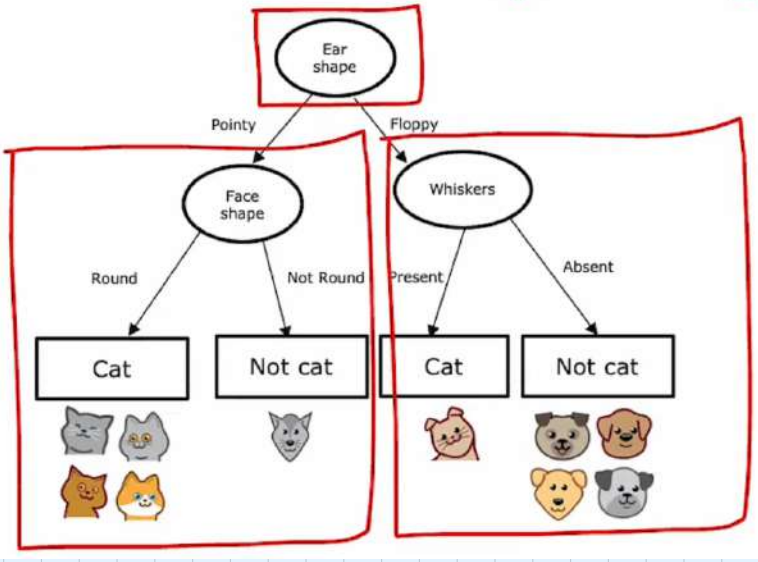
$$= H(p_1^{root}) - \left(w^{left} H\left(p_1^{left}\right) + w^{right} H\left(p_1^{right}\right)\right)$$

Putting it together

Decision Tree Learning

- Start with all examples at the root node
- Calculate information gain for all possible features, and pick the one with the highest information gain
- Split dataset according to selected feature, and create left and right branches of the tree
- Keep repeating splitting process until stopping criteria is met:
  - When a node is 100% one class
  - When splitting a node will result in the tree exceeding a maximum depth
  - Information gain from additional splits is less than threshold
  - When number of examples in a node is below a threshold

Recursive splitting

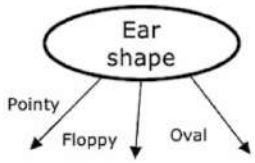


Recursive algorithm

Using one-hot encoding of categorical feature

Features with three possible values

Ear shape (x_1)	Face shape (x_2)	Whiskers (x_3)	Cat (y)
Pointy	Round	Present	1
Oval	Not round	Present	1
Oval	Round	Absent	0
Pointy	Not round	Present	0
Oval	Round	Present	1
Pointy	Round	Absent	1
Floppy	Not round	Absent	0
Oval	Round	Absent	1
Floppy	Round	Absent	0
Floppy	Round	Absent	0



3 possible values

~~Ear shape~~	Pointy ears	Floppy ears	Oval ears	Face shape	Whiskers	Cat
~~Pointy~~	1	0	0	Round	Present	1
~~Oval~~	0	0	1	Not round	Present	1
~~Oval~~	0	0	1	Round	Absent	0
~~Pointy~~	1	0	0	Not round	Present	0
~~Oval~~	0	0	1	Round	Present	1
~~Pointy~~	1	0	0	Round	Absent	1
~~Floppy~~	0	1	0	Not round	Absent	0
~~Oval~~	0	0	1	Round	Absent	1
~~Floppy~~	0	1	0	Round	Absent	0
~~Floppy~~	0	1	0	Round	Absent	0

If a categorical feature can take on k values, create k binary features (0 or 1 valued).

One hot encoding and neural networks

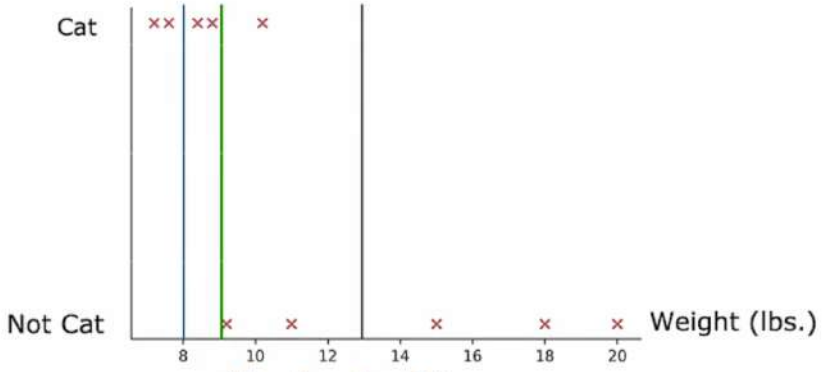
Pointy ears	Floppy ears	Round ears	Face shape	Whiskers	Cat
1	0	0	~~Round~~ 1	~~Present~~ 1	1
0	0	1	~~Not round~~ 0	~~Present~~ 1	1
0	0	1	~~Round~~ 1	~~Absent~~ 0	0
1	0	0	~~Not round~~ 0	~~Present~~ 1	0
0	0	1	~~Round~~ 1	~~Present~~ 1	1
1	0	0	~~Round~~ 1	~~Absent~~ 0	1
0	1	0	~~Not round~~ 0	~~Absent~~ 0	1
0	0	1	~~Round~~ 1	~~Absent~~ 0	1
0	1	0	~~Round~~ 1	~~Absent~~ 0	1
0	1	0	~~Round~~ 1	~~Absent~~ 0	1

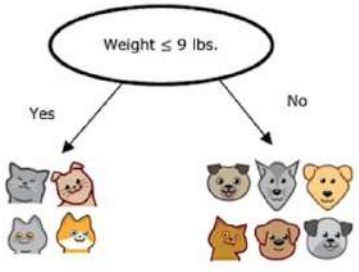
Continuous valued feature

Continuous features

Ear shape	Face shape	Whiskers	Weight (lbs.)	Cat
Pointy	Round	Present	7.2	1
Floppy	Not round	Present	8.8	1
Floppy	Round	Absent	15	0
Pointy	Not round	Present	9.2	0
Pointy	Round	Present	8.4	1
Pointy	Round	Absent	7.6	1
Floppy	Not round	Absent	11	0
Pointy	Round	Absent	10.2	1
Floppy	Round	Absent	18	0
Floppy	Round	Absent	20	0

Splitting on a continuous variable





$$H(0.5) - \left(\frac{2}{10}H\left(\frac{2}{2}\right) + \frac{8}{10}H\left(\frac{3}{8}\right)\right) = 0.24$$

$$H(0.5) - \left(\frac{4}{10}H\left(\frac{4}{4}\right) + \frac{6}{10}H\left(\frac{1}{6}\right)\right) = 0.61$$

$$H(0.5) - \left(\frac{7}{10}H\left(\frac{5}{7}\right) + \frac{3}{10}H\left(\frac{0}{3}\right)\right) = 0.40$$

用不等式解决连续决策

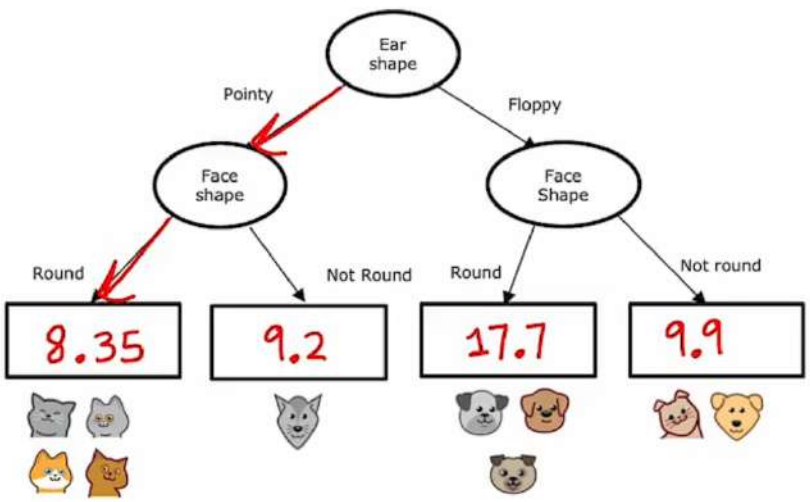
Regression Trees

Regression with Decision Trees: Predicting a number

Ear shape	Face shape	Whiskers	Weight (lbs.)
Pointy	Round	Present	7.2
Floppy	Not round	Present	8.8
Floppy	Round	Absent	15
Pointy	Not round	Present	9.2
Pointy	Round	Present	8.4
Pointy	Round	Absent	7.6
Floppy	Not round	Absent	11
Pointy	Round	Absent	10.2
Floppy	Round	Absent	18
Floppy	Round	Absent	20

X y

Regression with Decision Trees



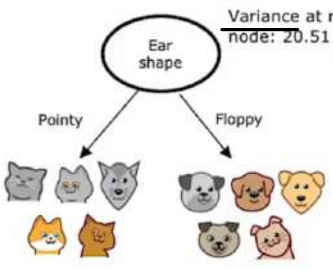
Weights(lbs.): 7.2, 8.4, 7.6, 10.2

Weights(lbs.): 9.2

Weights (lbs.): 15, 18, 20

Weights(lbs.): 8.8,11

Choosing a split



Weights: 7.2, 9.2, 8.4,7.6, 10.2

Variance: 1.47

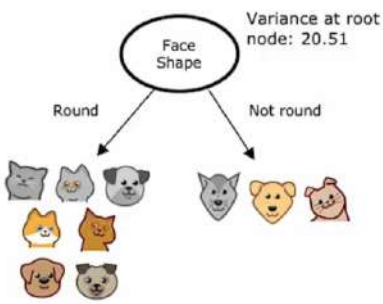
Weights: 8.8, 15, 11, 18, 20

Variance: 21.87

$w^{\text{left}} = 5/10$ $w^{\text{right}} = 5/10$

$$20.51 - \left(\frac{5}{10} * 1.47 + \frac{5}{10} * 21.87\right)$$

$$= 8.84$$



Weights: 7.2, 15, 8.4, 7.6,10.2, 18, 20

Variance: 27.80

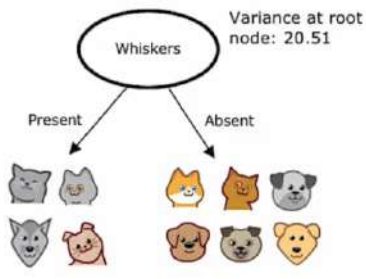
Weights: 8.8,9.2,11

Variance: 1.37

$w^{\text{left}} = 7/10$ $w^{\text{right}} = 3/10$

$$20.51 - \left(\frac{7}{10} * 27.80 + \frac{3}{10} * 1.37\right)$$

$$= 0.64$$



Weights: 7.2, 8.8, 9.2, 8.4

Variance: 0.75

Weights: 15, 7.6, 11, 10.2, 18, 20

Variance: 23.32

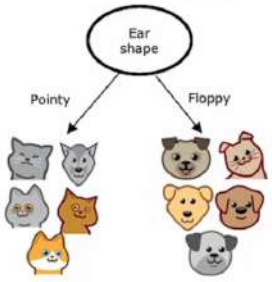
$w^{\text{left}} = 4/10$ $w^{\text{right}} = 6/10$

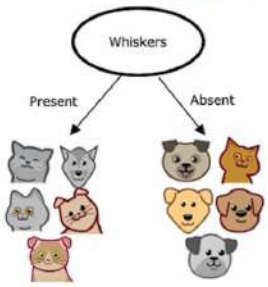
$$20.51 - \left(\frac{4}{10} * 0.75 + \frac{6}{10} * 23.32\right)$$

$$= 6.22$$

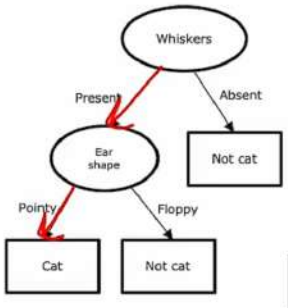
Using multiple decision trees

Trees are highly sensitive to small changes of the data

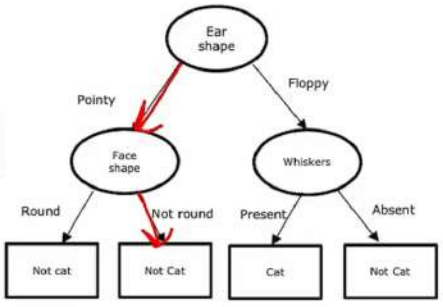




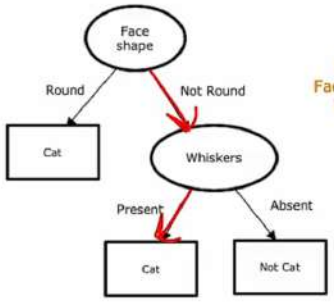
Tree ensemble



Prediction: Cat



Prediction: Not cat



Prediction: Cat

New test example

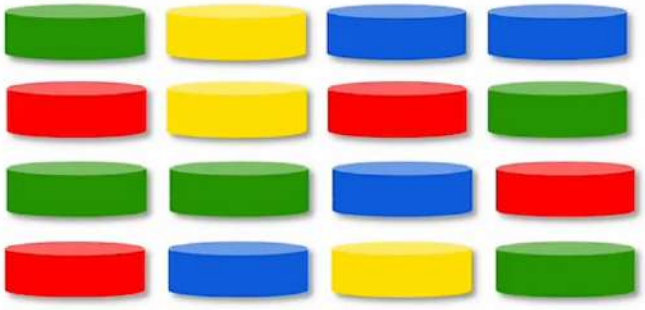
Ear shape: Pointy
Face shape: Not Round
Whiskers: Present

Final prediction: Cat

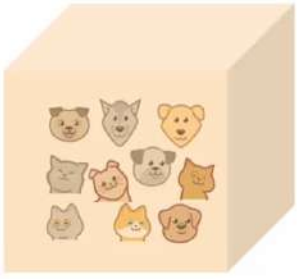
Sampling with replacement 有放回抽样

Sampling with replacement

Sampling with replacement:



Sampling with replacement



Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	1
Floppy	Not round	Absent	0
Pointy	Round	Absent	1
Pointy	Not round	Present	0
Floppy	Not round	Absent	0
Pointy	Round	Absent	1
Pointy	Round	Present	1
Floppy	Not round	Present	1
Floppy	Round	Absent	0
Pointy	Round	Absent	1

Random forest algorithm 随机森林

Generating a tree sample

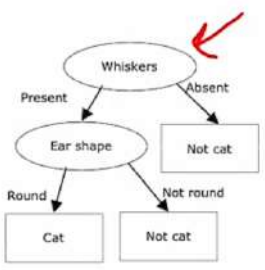
Given training set of size m

For $b = 1$ to B:

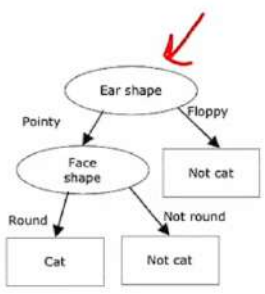
Use sampling with replacement to create a new training set of size m

Train a decision tree on the new dataset

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Pointy	Round	Absent	Yes
Floppy	Not Round	Absent	No
Floppy	Not Round	Absent	No
Pointy	Round	Absent	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Round	Present	Yes



...

Bagged decision tree

Randomizing the feature choice

At each node, when choosing a feature to use to split, if n features are available, pick a random subset of $k < n$ features and allow the algorithm to only choose from that subset of features.

$k = \sqrt{n}$

建议100个树

Random forest algorithm

XGBoost

Boosted trees intuition

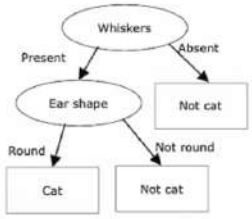
Given training set of size m

For $b = 1$ to B:

Use sampling with replacement to create a new training set of size m
But instead of picking from all examples with equal (1/m) probability, make it more likely to pick examples that the previously trained trees misclassify

Train a decision tree on the new dataset

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Boosted trees intuition

Given training set of size m

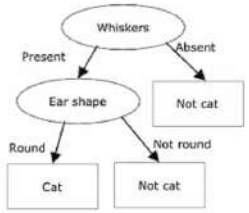
For $b = 1$ to B:

Use sampling with replacement to create a new training set of size m
But instead of picking from all examples with equal (1/m) probability, make it more likely to pick examples that the previously trained trees misclassify

Train a decision tree on the new dataset

1, 2, ..., b-1 ↑ b

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Ear shape	Face shape	Whiskers	Prediction
Pointy	Round	Present	Cat ✓
Floppy	Not Round	Present	Not cat ✗ ←
Floppy	Round	Absent	Not cat ✓
Pointy	Not Round	Present	Not cat ✓
Pointy	Round	Present	Cat ✓
Pointy	Round	Absent	Not cat ✗ ←
Floppy	Not Round	Absent	Not cat ✓
Pointy	Round	Absent	Not cat ✗ ←
Floppy	Round	Absent	Not cat ✓
Floppy	Round	Absent	Not cat ✓

XGBoost (eXtreme Gradient Boosting)

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)

Using XGBoost

Classification

```
from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Regression

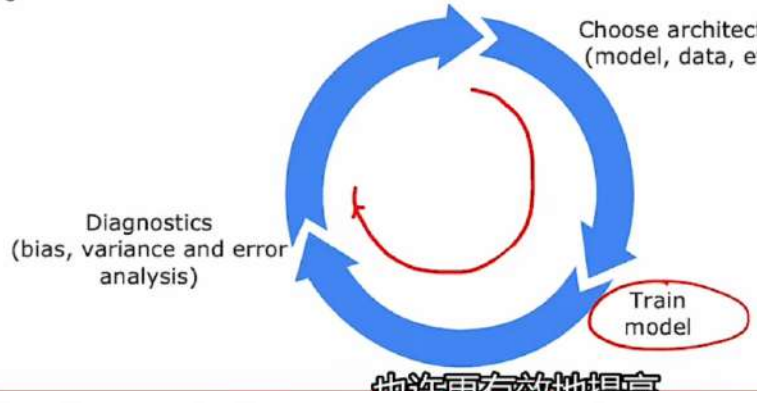
```
from xgboost import XGBRegressor

model = XGBRegressor()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

When to use decision trees decision tree Vs Neural Network

Decision Trees vs Neural Networks

Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast



Decision Trees vs Neural Networks

Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees may be human interpretable

Neural Networks

- Works well on all types of data, including tabular (structured) and unstructured data
- May be slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural networks