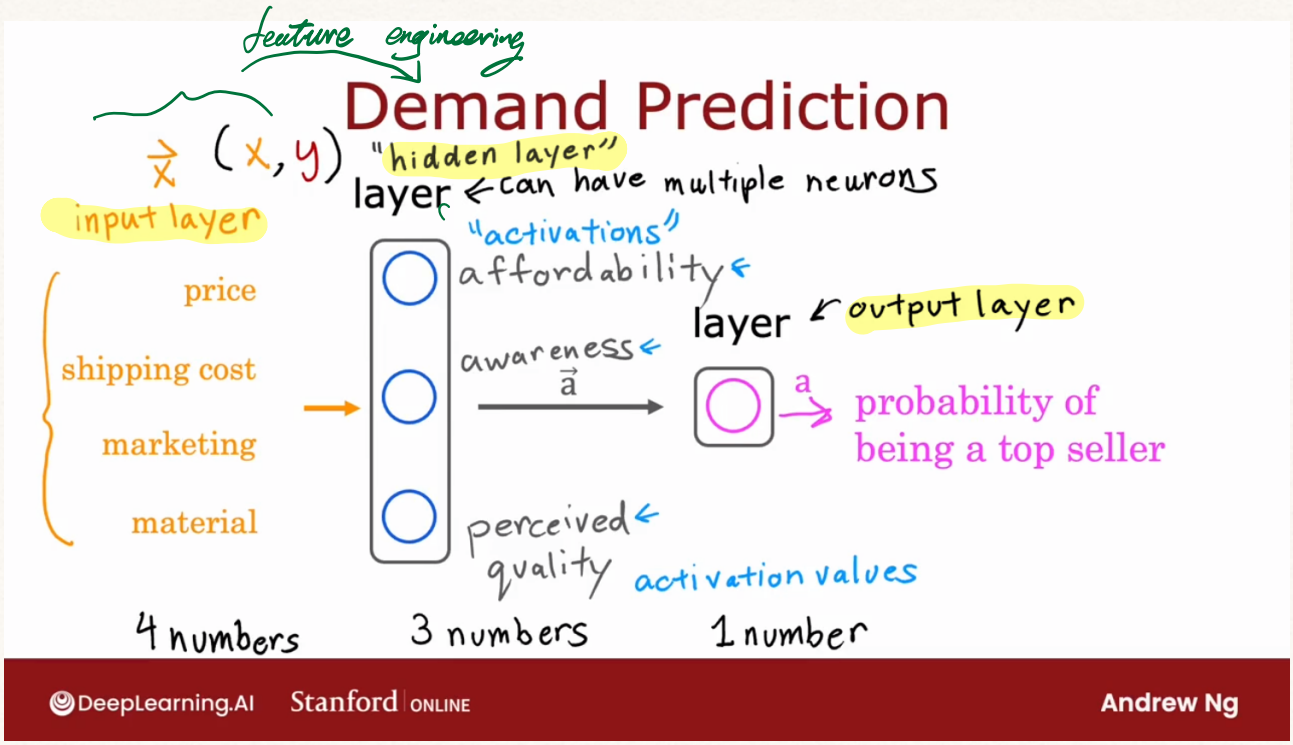
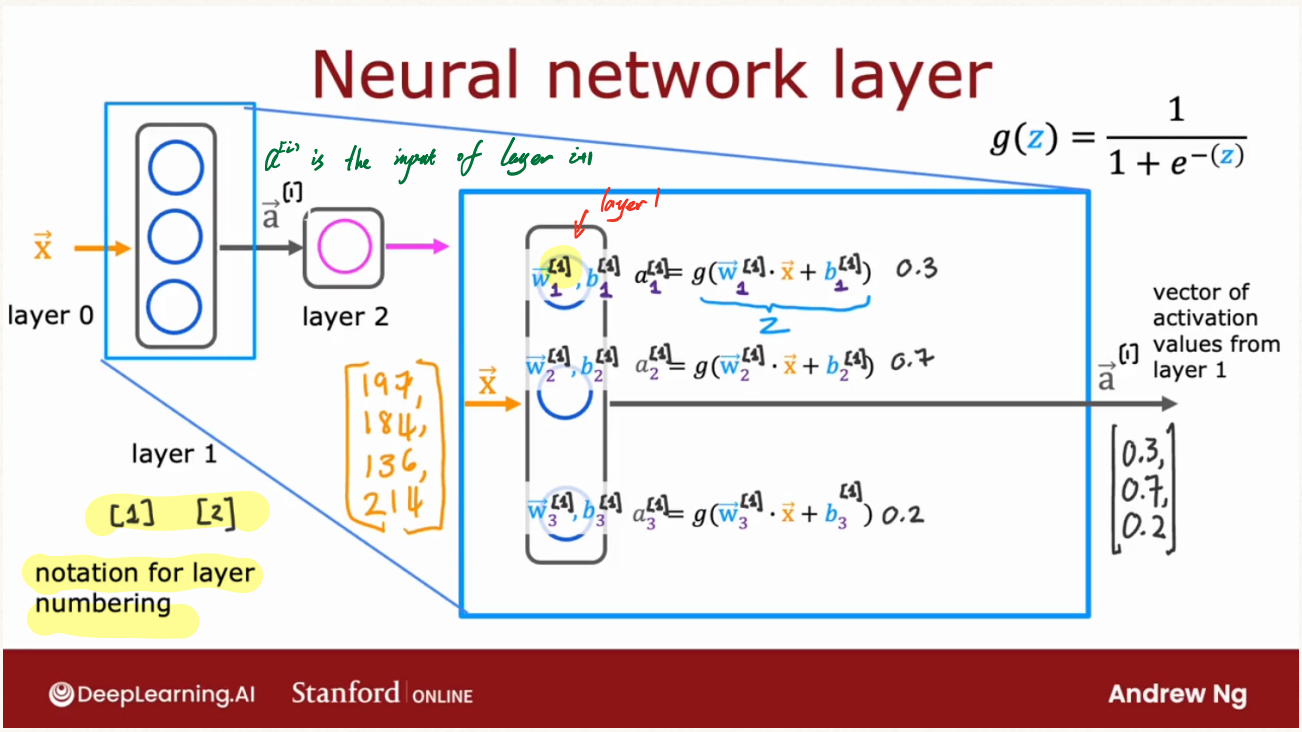
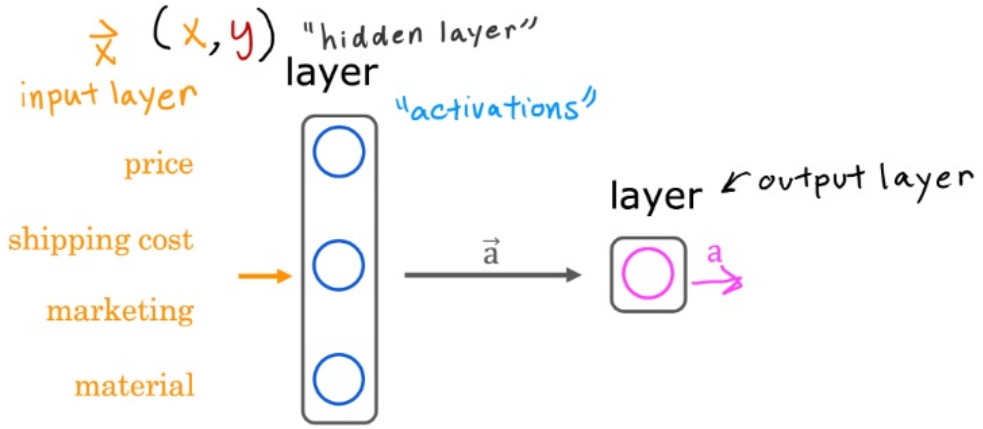

feature engineering
Demand Prediction
$\vec{x}$ (x, y)
"hidden layer"
layer can have multiple neurons
input layer
price
shipping cost
marketing
material
"activations"
affordability
awareness
perceived quality
$\vec{a}$
layer output layer
a
probability of being a top seller
activation values
4 numbers
3 numbers
1 number
DeepLearning.AI
Stanford ONLINE
Andrew Ng


Neural network layer
$g(z) = \frac{1}{1+e^{-(z)}}$
$\vec{a}^{[1]}$ is the input of layer 1
layer 1
$\vec{x}$
layer 0
$\vec{a}^{[1]}$
layer 2
layer 1
[1] [2]
notation for layer numbering
197, 184, 136, 214
$\vec{w}_1^{[1]}, b_1^{[1]}$ $a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$ 0.3
z
$\vec{w}_2^{[1]}, b_2^{[1]}$ $a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$ 0.7
$\vec{w}_3^{[1]}, b_3^{[1]}$ $a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$ 0.2
$\vec{a}^{[1]}$
vector of activation values from layer 1
[0.3, 0.7, 0.2]
DeepLearning.AI
Stanford ONLINE
Andrew Ng

1.

1 / 1 分



Which of these are terms used to refer to components of an artificial neural network?

- [x] activation function

  正確
  Yes, an activation is the number calculated by a neuron (and an activation is a vector that is output by a layer that contains multiple neurons).

- [x] layers

  正確
  Yes, a layer is a grouping of neurons in a neural network

- [x] neurons

  正確
  Yes, a neuron is a part of a neural network

- [ ] axon

1.

1 / 1 分

For a neural network, here is the formula for calculating the activation of the third neuron in layer 2, given the activation vector from layer 1: $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[1]} + b_3^2)$. Which of the following are correct statements?

- [x] The activation of layer 2 is determined using the activations from the previous layer.

  正确
  Correct. The previous layer's activations can be considered as the input features into the current layer.

- [x] The activation of unit 3 (neuron 3) of layer 2 is calculated using a parameter vector $\vec{w}$ and b that are specific to unit 3 (neuron 3).

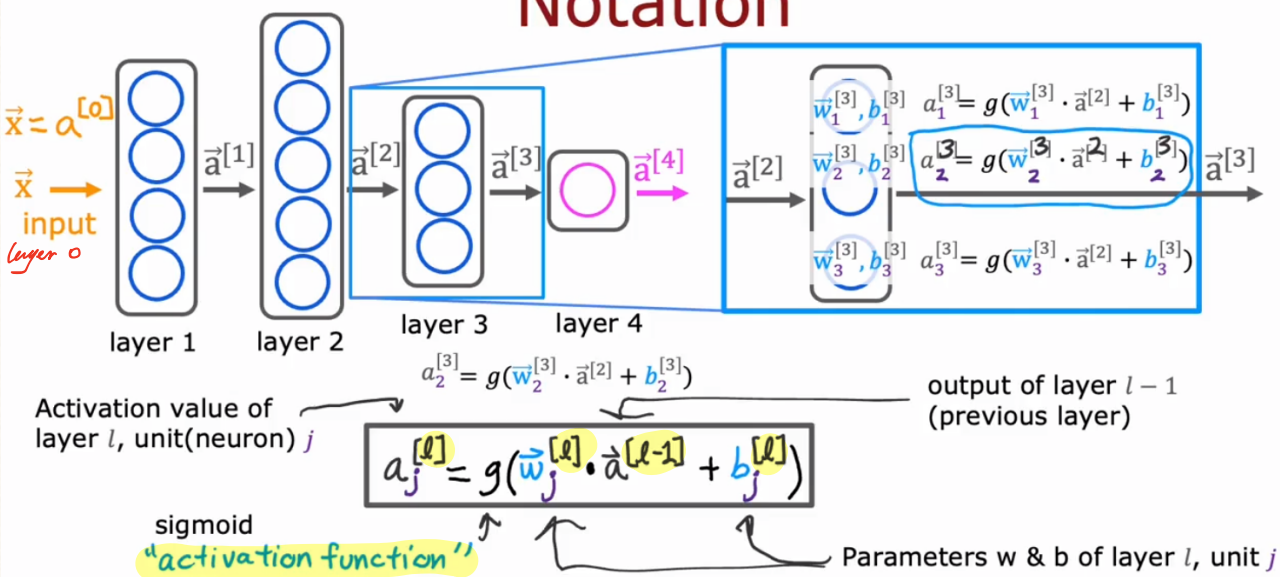
  正确
  Correct. Each neuron has its own set of parameters $\vec{w}$ and b.

- [x] Unit 3 (neuron 3) outputs a single number (a scalar).

  正确
  Correct, the calculation above for unit 3 (neuron 3) is a single number (and not a vector).

- [ ] If you are calculating the activation for layer 1, then the previous layer's activations would be denoted by $\vec{a}^{-1}$

Notation



$a_2^{[3]} = g(\vec{w}_2^{[3]} \cdot \vec{a}^{[2]} + b_2^{[3]})$

Activation value of layer l, unit(neuron) j

output of layer $l-1$ (previous layer)

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

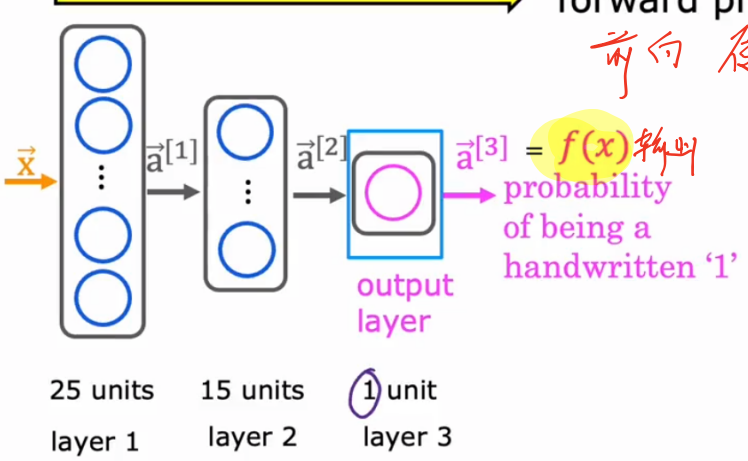
sigmoid "activation function"

Parameters w & b of layer l, unit j

Handwritten digit recognition

forward propagation

前向传播



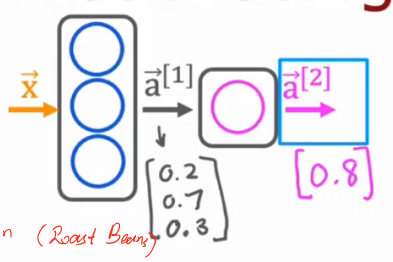
$$\vec{a}^{[3]} = \left[g\left(\vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]} \right) \right]$$

is $a_1^{[3]} \geq 0.5$?

yes → $\hat{y} = 1$ image is digit 1

no → $\hat{y} = 0$ image isn't digit 1

Build the model using TensorFlow



$is\ a_1^{[2]} \geq 0.5?$ (threshold)

yes → $\hat{y} = 1$ ✗

no → $\hat{y} = 0$ ○

```
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
```


Note about numpy arrays

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$ 2 rows, 3 columns — 2 x 3 matrix

$$\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix}$$ 4 rows, 2 columns — 4 x 2 matrix

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

```
[[1, 2, 3],
 [4, 5, 6]]
```

```
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,]]
```

2D array

2 x 3

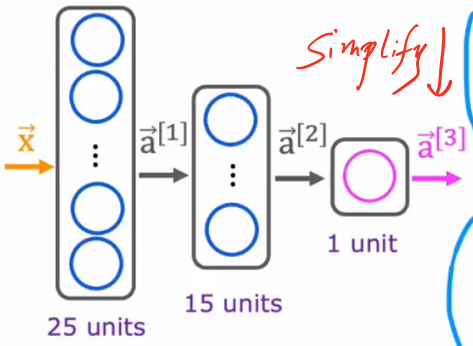
4 x 2

1 x 2

2 x 1

数据在 tensorflow 中以 Tensor（或叫 Martix）存储

Digit classification model



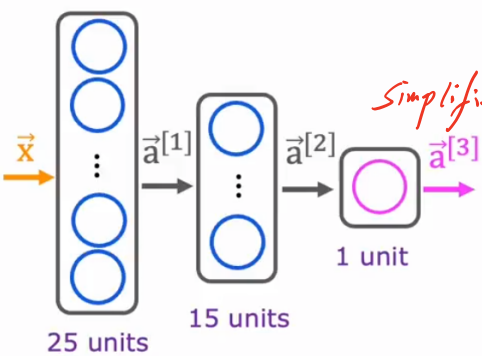
Simplify

```
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)
```

to create a neural network by these layers

← more about this next week!

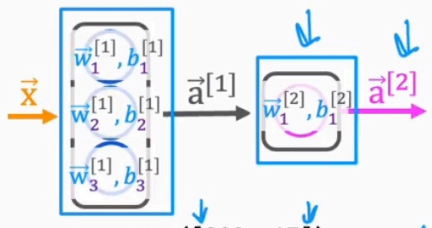
Digit classification model



Simplified

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

model.compile(...)

x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])

model.fit(x,y)

model.predict(x_new)
```

forward prop (coffee roasting model)



Complicated

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

```
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)
```

$w_1^{[2]} \rightarrow$ w2_1

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

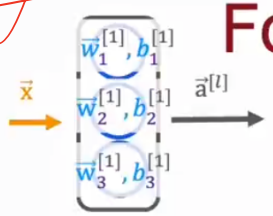
1D arrays

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

```
w1_1 = np.array([1, 2])
b1_1 = np.array([-1])
z1_1 = np.dot(w1_1,x)+ b
a1_1 = sigmoid(z1_1)

w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b
a1_2 = sigmoid(z1_2)

w1_3 = np.array([5, -6])
b1_3 = np.array([2])
z1_3 = np.dot(w1_3,x)+b
a1_3 = sigmoid(z1_3)

a1 = np.array([a1_1, a1_2, a1_3])
```


Forward prop in NumPy

Simplified



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

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

2 by 3

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

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

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

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

定义层

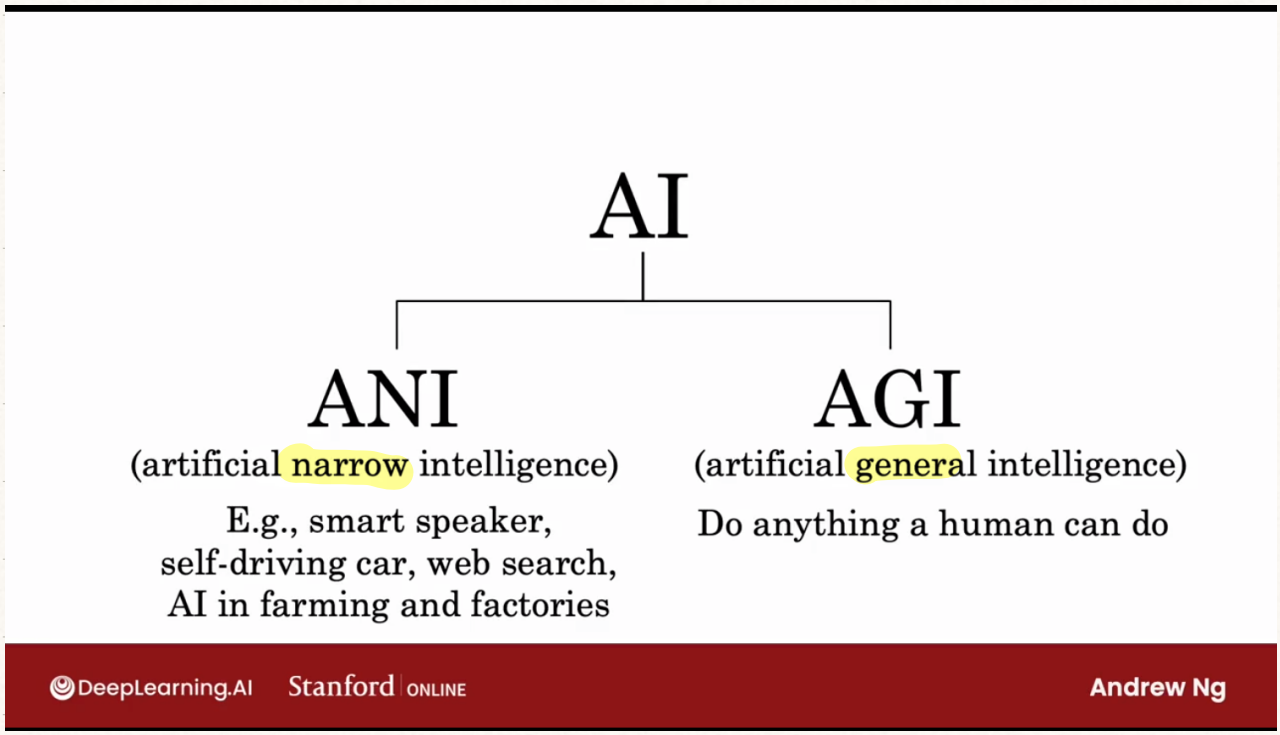
```
def dense(a_in,W,b, g):
    units = W.shape[1]
    a_out = np.zeros(units)
    for j in range(units):
        w = W[:,j]
        z = np.dot(w,a_in) + b[j]
        a_out[j] = g(z)
    return a_out
```

[0,0,0] 0,1,2 sigmoid

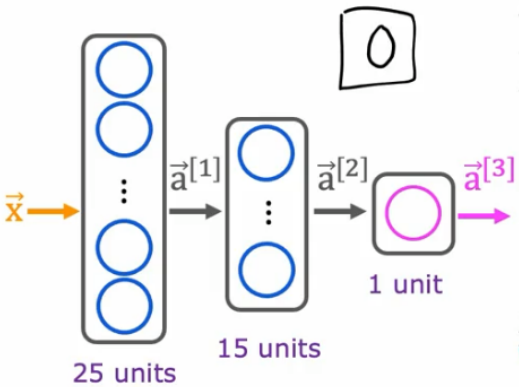
定义网络

```
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
```

capital W refers to a matrix


AI
ANI
(artificial narrow intelligence)
E.g., smart speaker,
self-driving car, web search,
AI in farming and factories
AGI
(artificial general intelligence)
Do anything a human can do
DeepLearning.AI
Stanford ONLINE
Andrew Ng

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

Tensor Flow

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)
```

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

DeepLearning.AI Stanford ONLINE Andrew Ng

但它最终会变成逻辑回归模型。

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

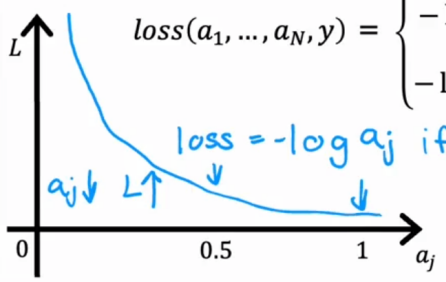
$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$

$\vdots$

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

Crossentropy loss

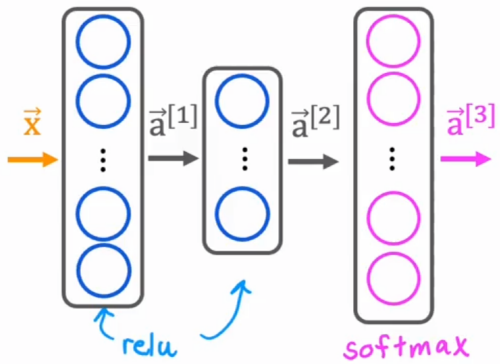
$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \\ -\log a_N & \text{if } y=N \end{cases}$$



DeepLearning.AI Stanford ONLINE Andrew Ng

例如，如果y = 2，

Neural Network with Softmax output



25 units 15 units 10 units

10 classes

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

a_i：概率

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

logistic regression

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

softmax

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

MNIST with softmax

1. specify the model

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

2. specify loss and cost

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

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.

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)

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

```
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) )
```

logit: z

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() )
```

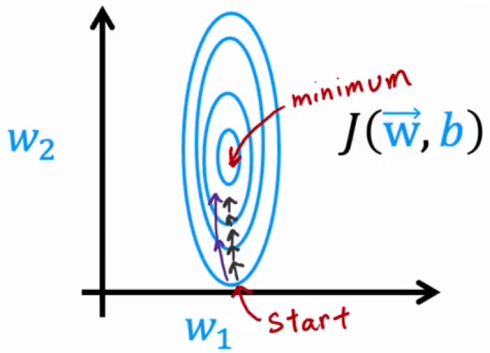
'linear'

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


除了这个推荐的版本，在数字上更准确，

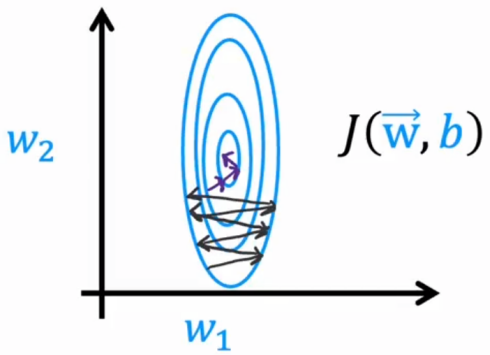
except that the version that is recommended is more numerically accurate,

Adam Algorithm Intuition



If w_j (or b) keeps moving in same direction, increase α_j.

adjust α automatically

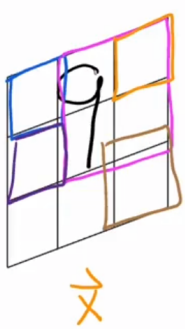


If w_j (or b) keeps oscillating, reduce α_j.

Adam算法具体是如何做到这一点的有点复杂，超出了本课程的范围，

Convolutional Layer

卷积层

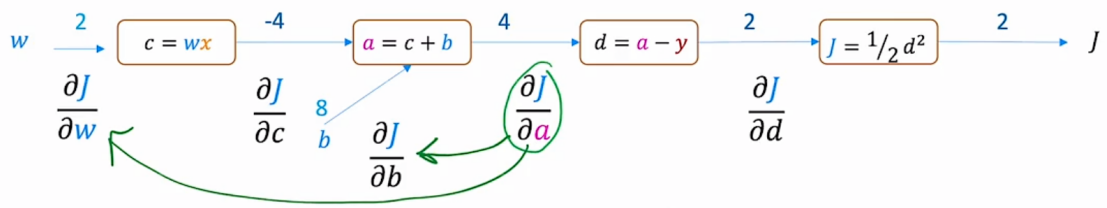


Each Neuron only looks at part of the previous layer's inputs.

Why?

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

Backprop is an efficient way to compute derivatives



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

If N nodes and P parameters, compute derivatives

in roughly $N + P$ steps rather than $N \times P$ steps.

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

the computation graph gives you a very efficient way to compute all the derivatives, and that's

Neural Network Example

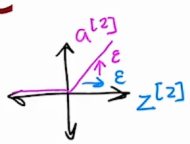
$x = 1 \quad y = 5$

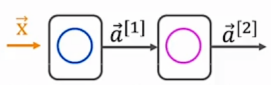
$w^{[1]} = 2, b^{[1]} = 0$

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

ReLU activation

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

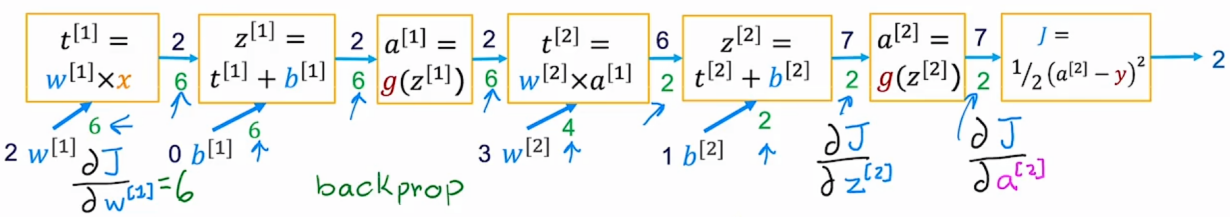




$a^{[1]} = g(w^{[1]} x + b^{[1]}) = \overbrace{w^{[1]} x + b^{[1]}}^{z^{[1]}} = 2 \times 1 + 0 = 2$

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

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



if $w^{[1]} \uparrow \varepsilon$, then $J \uparrow 6 \times \varepsilon$ let's verify this!

阶数

确定模型(d) 确定参数(w, b) 测试误差

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. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$

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

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

⋮

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

$w^{<1>}, b^{<1>}$ → $J_{cv}(w^{<1>}, b^{<1>})$

$J_{cv}(w^{<2>}, b^{<2>})$

⋮

$J_{cv}(w^{<10>}, b^{<10>})$

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

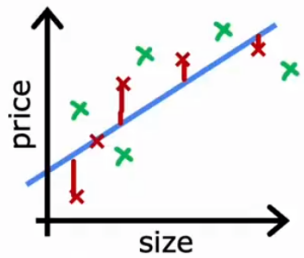
Estimate generalization error using test the set: $J_{test}(w^{<4>}, b^{<4>})$

DeepLearning.AI Stanford ONLINE Andrew Ng

所以到目前为止，你还没有把任何参数 w b d 放到测试集中

and so up until this point, you've not fit any parameters,either w or b or d to the test set

Bias/variance

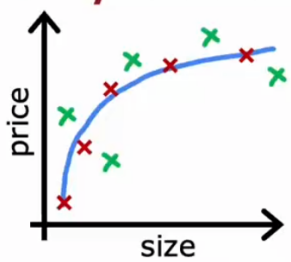


$f_{\vec{w},b}(x) = w_1 x + b$

High bias (underfit)

J_{train} is high

J_{cv} is high

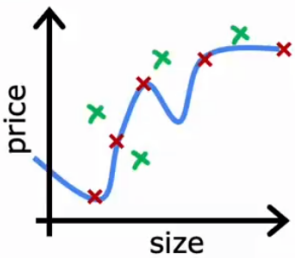


$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$

"Just right"

J_{train} is low

J_{cv} is low



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

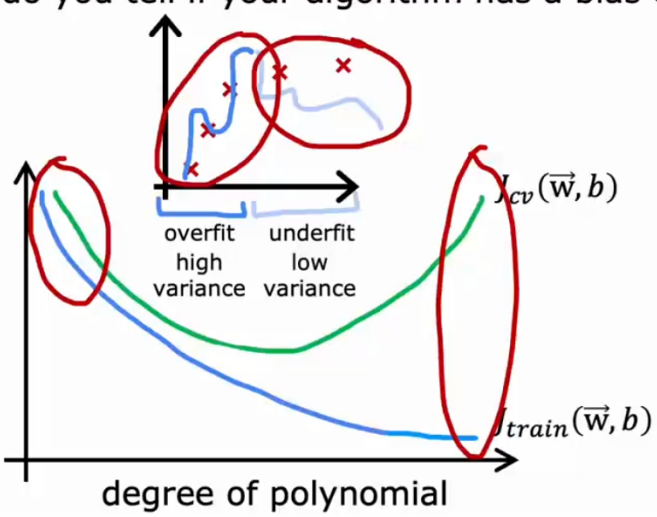
High variance (overfit)

J_{train} is low

J_{cv} is high

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

High Bias : not even well in Training Set

High Variance : not good in CVS, but may good in TS

全连接层 (dense laver）类型，这一层中的每个神经元都从前一层得到所有的激活。

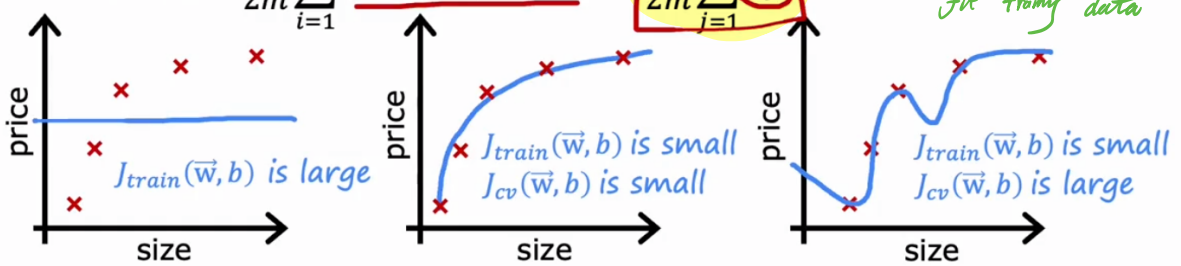
Linear regression with regularization

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

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

正则项,

λ: keep w_j small & fit training data



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

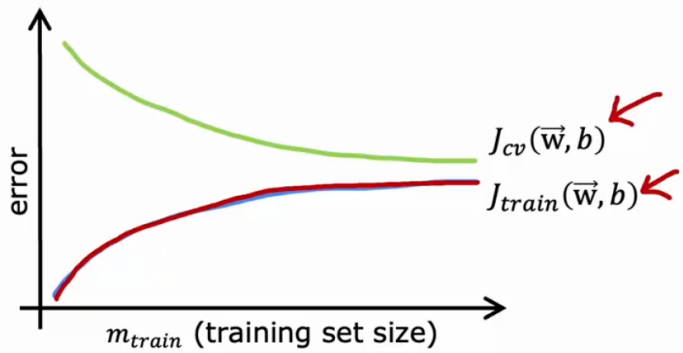
λ

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

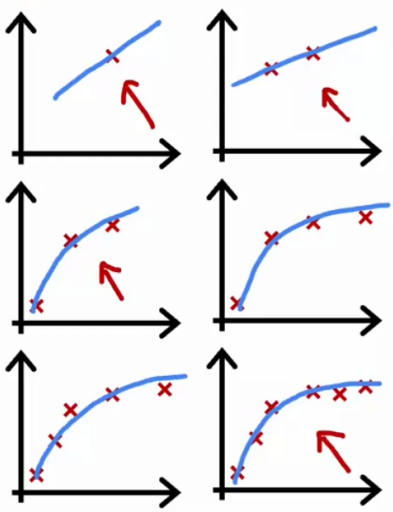
Learning curves

J_{train} = training error

J_{cv} = cross validation error



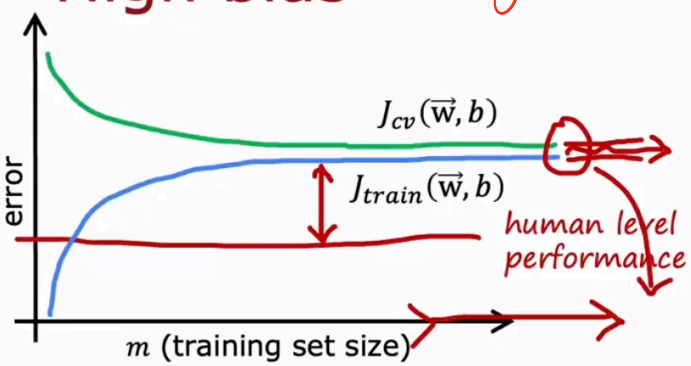
$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$



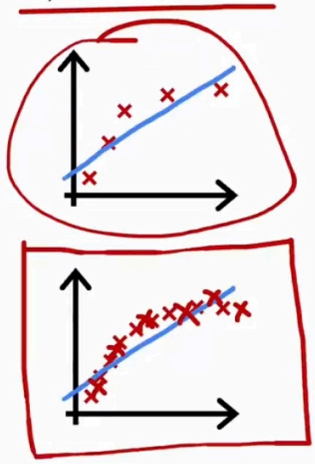
DeepLearning.AI Stanford ONLINE Andrew Ng

High bias

underfit



$f_{\vec{w},b}(x) = w_1 x + b$



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

DeepLearning.AI Stanford ONLINE Andrew Ng

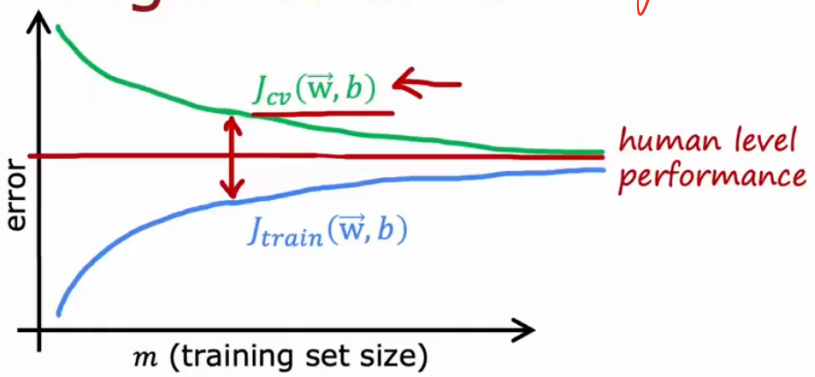
如果一个学习算法有很高的偏差，获得更多的训练数据本身不会有多大帮助。

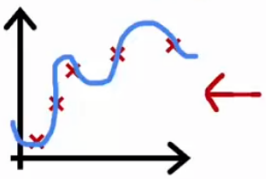
that if a learning algorithm has high bias, getting more training data will not by itself help that much.

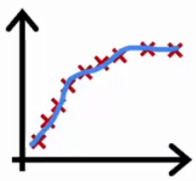
High variance

overfit

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







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

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

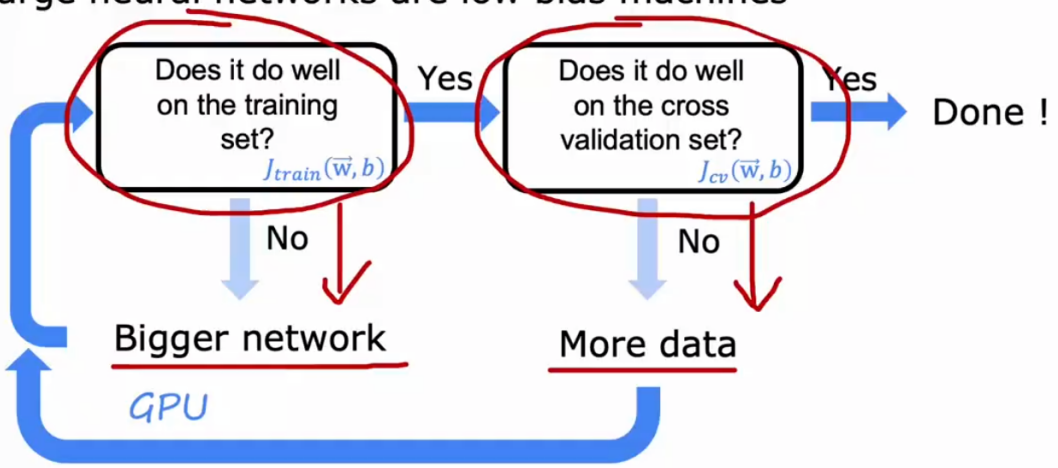
$$J(\vec{w}, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

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

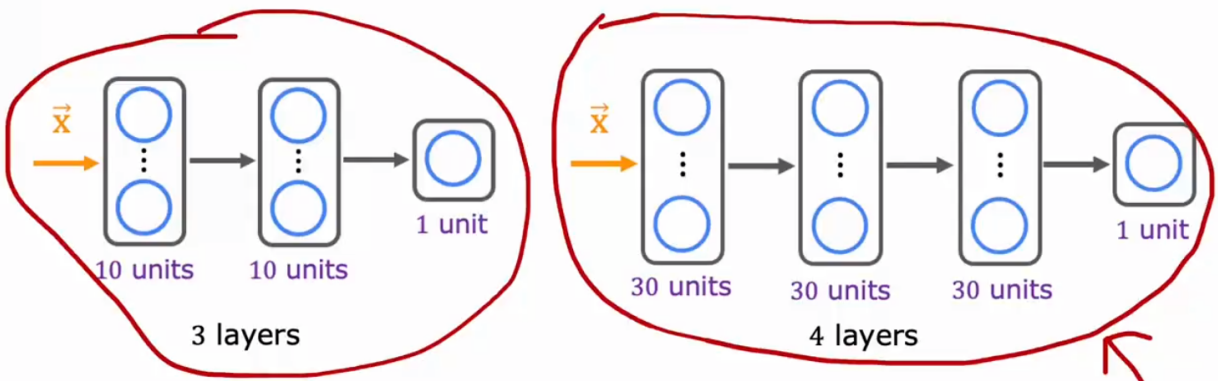
→ Get more training examples	fixes high variance
→ Try smaller sets of features $x, x^2, x^3, x^4, x^5 \ldots$	fixes high variance
→ Try getting additional features	fixes high bias
→ Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$	fixes high bias
→ Try decreasing λ	fixes high bias
→ Try increasing λ	fixes high variance

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.

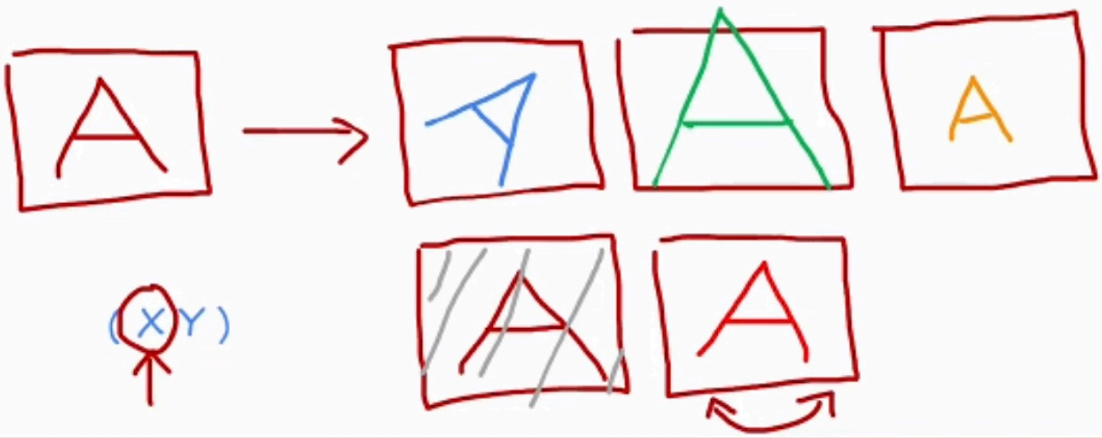
数据 增强

Data augmentation

To modify examples

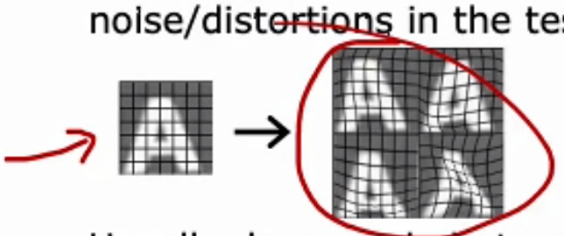
Augmentation: modifying an existing training example to create a new training example.

Increase dataset manually



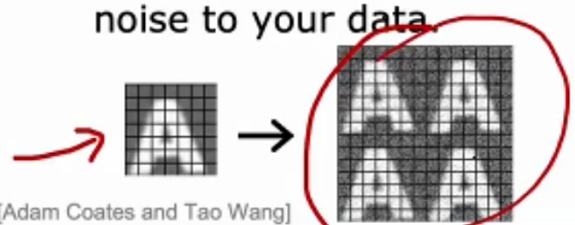
Data augmentation by introducing distortions

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.



[Adam Coates and Tao Wang]

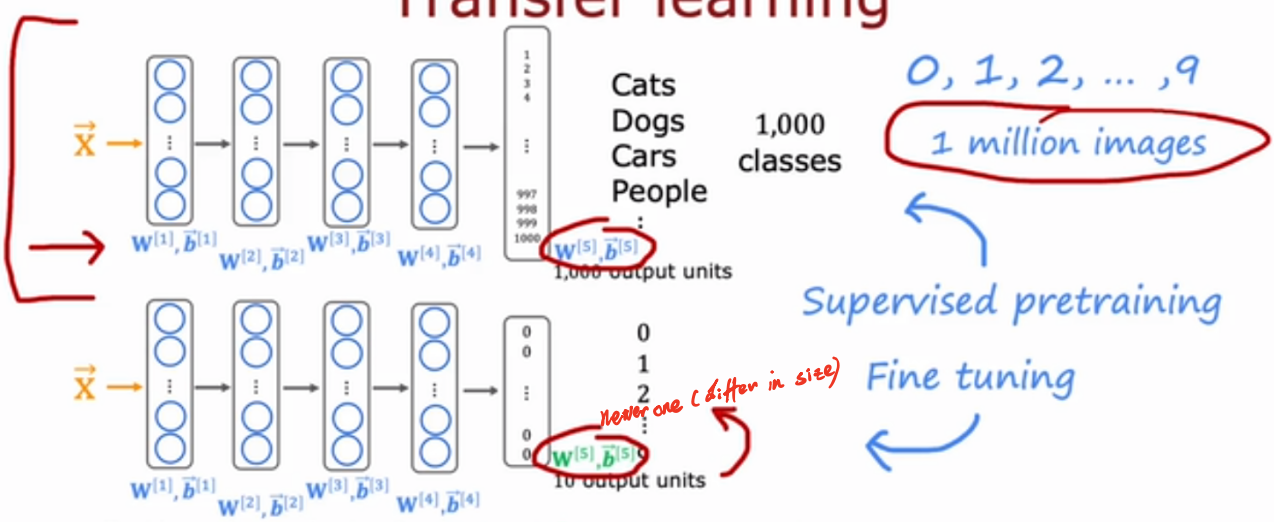
x_i=intensity (brightness) of pixel i
$x_i \leftarrow x_i$ + random noise

数据合成 To create examples

Data synthesis

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

Transfer learning



Option 1: only train output layers parameters.

Option 2: train all parameters. but start with existing W_i

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.

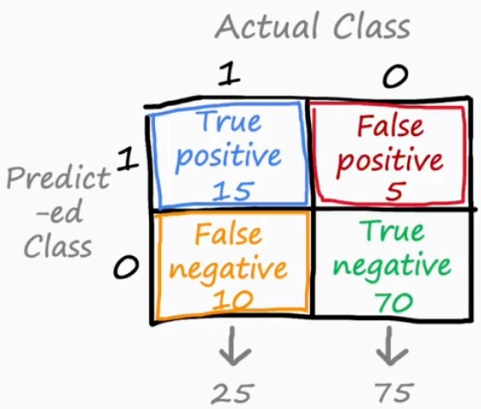
微调参数

1000
50

Pre-training

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?)

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

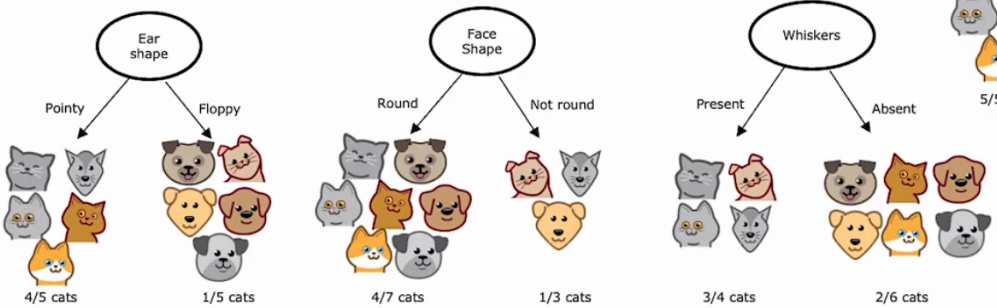
Recall:
(of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

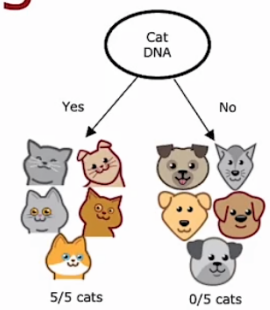
$$\frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15+10} = 0.6$$

Decision Tree Learning

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

Maximize purity (or minimize impurity)

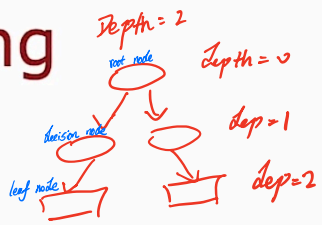


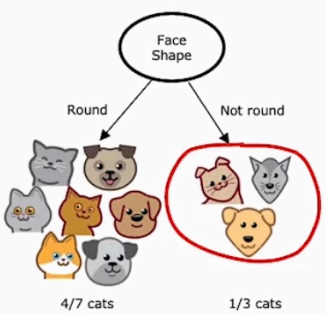


Decision Tree Learning

Decision 2: When do you stop splitting?

- 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

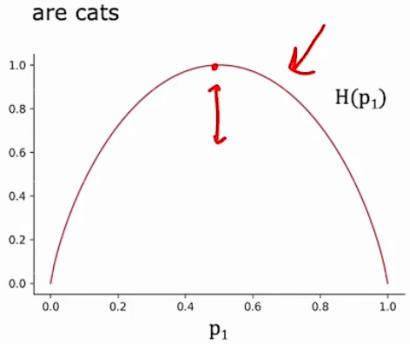




impurity ↑ → Entropy ↑

Entropy as a measure of impurity

p_1 = fraction of examples that are cats



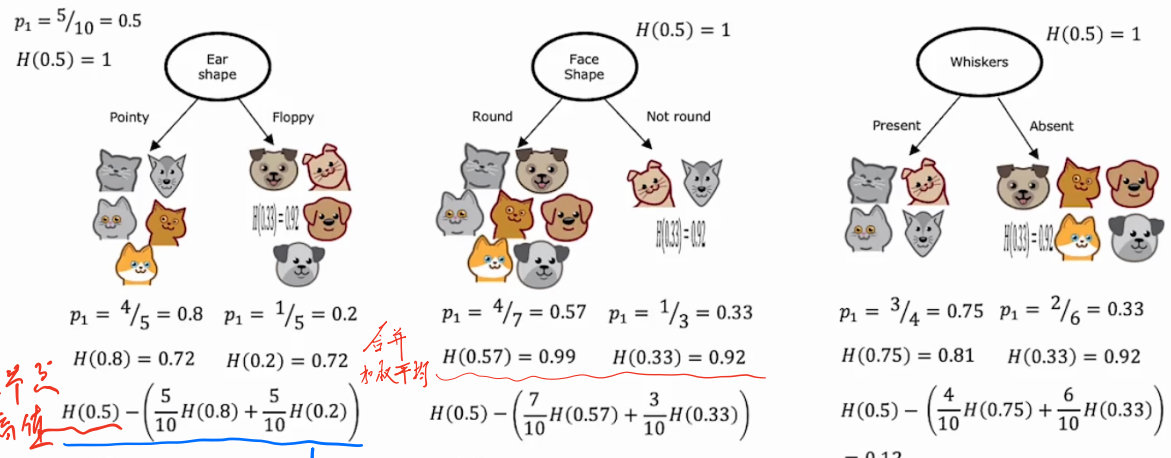
$$p_0 = 1 - p_1$$

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

Note: "0 log(0)" = 0

Goal: Reduce entropy → Raise purity

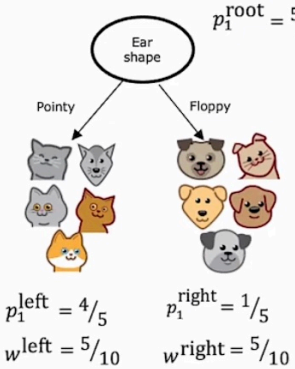
Choosing a split



Information gain: 信息增益

Information Gain

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



$p_1^{left} = 4/5$
$w^{left} = 5/10$

$p_1^{right} = 1/5$
$w^{right} = 5/10$

Information gain Maximize

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

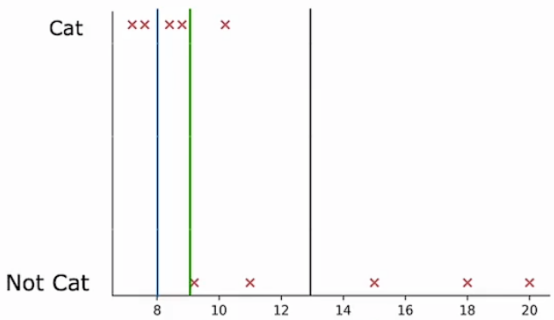
One hot encoding

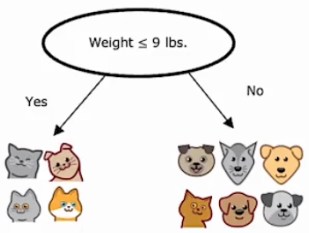
Ear shape	Pointy ears	Floppy ears	Oval ears	Face shape	Whiskers	Cat
Pointy	1	0	0	Round	Present	1
Oval	0	0	1	Not round	Present	1
Oval	0	0	1	Round	Absent	0
Pointy	1	0	0	Not round	Present	0
Oval	0	0	1	Round	Present	1
Pointy	1	0	0	Round	Absent	1
Floppy	0	1	0	Not round	Absent	0
Oval	0	0	1	Round	Absent	1
Floppy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

And because one of these features will always take on the value 1 that's

因为这些特征中总有一个取1，也就是hot（热）特征

Splitting on a continuous variable





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

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

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

尝试不同阈值
使信息增益最大

↑

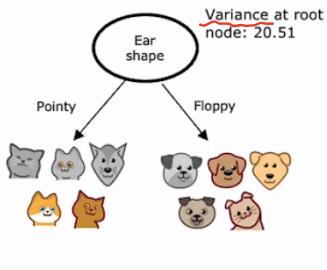
decision tree for classification

for regression

↓

Choosing a split

predict the weight

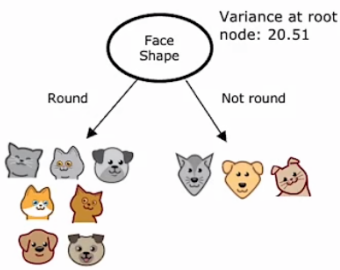


Weights: 7.2, 9.2, 8.4,7.6, 10.2
Variance: 1.47

Weights: 8.8, 15, 11, 18, 20
Variance: 21.87

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

$$20.51 - \left(\frac{5}{10} * 1.47 + \frac{5}{10} * 21.87\right)$$
$$= 8.84$$

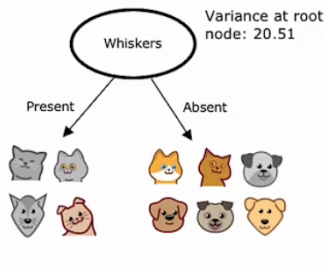


Weights: 7.2, 15, 8.4, 7.6,10.2, 18, 20
Variance: 27.80

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

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

$$20.51 - \left(\frac{4}{10} * 0.75 + \frac{6}{10} * 23.32\right)$$
$$= 6.22$$

DeepLearning.AI Stanford ONLINE Andrew Ng

Tree Ensemble

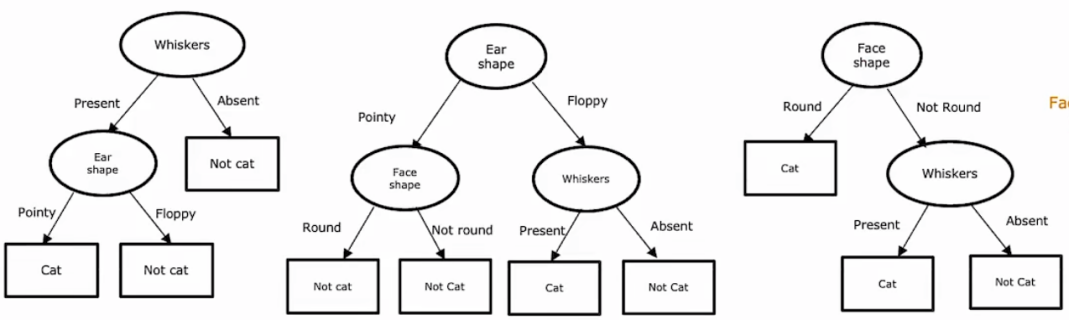
Trees are highly sensitive to small changes of the data

DeepLearning.AI Stanford ONLINE Andrew Ng

我们仅改变了单个训练样本，具有最高信息增益的节点特征就变成了胡须，而不是耳朵形状

With just changing a single training example, the highest information gain feature at the split on becomes the whiskers feature instead of the ear shape feature.

Tree ensemble



New test example

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

DeepLearning.AI Stanford ONLINE Andrew Ng

这就是我们所说的集成树，意思是多棵树的集合

And this is what we call a tree ensemble, which just means a collection of multiple trees.

如果你想给一个新的测试样本分类，那么你要做的就是在新样本上运行这三棵树，并让它们投票做出最终预测

If you had a new test example that you wanted to classify, then what you would do is run all three of these trees on your new example and get them to vote on what is the final prediction.

For Building ensemble tree ↓

有放回抽样

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

DeepLearning.AI Stanford ONLINE Andrew Ng

其中一些结果是重复的，并且你还注意到此训练集并不包含所有的原始训练样本

some of which are repeats and you notice also that this training set does not contain all ten of the original training examples.

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.

$k = \sqrt{n}$

Random forest algorithm

Boosted trees intuition

Given training set of size m

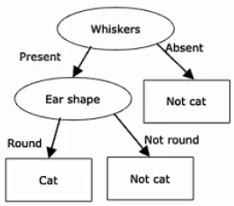
For b = 1 to B:

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



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)
```


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