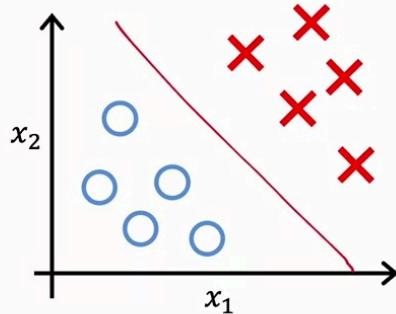


## Unsupervised Learning, Recommenders, Reinforcement Learning

### Clustering

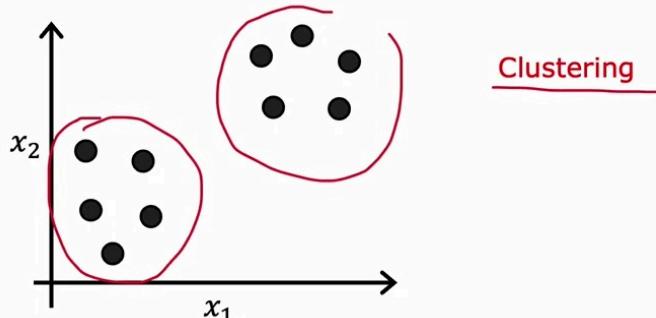
#### Week1

## Supervised learning



Training set:  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots, (x^{(m)}, y^{(m)})\}$  ?

## Unsupervised learning



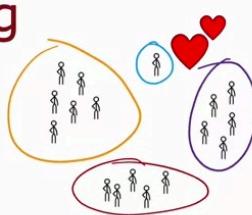
Training set:  $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$

## Applications of clustering

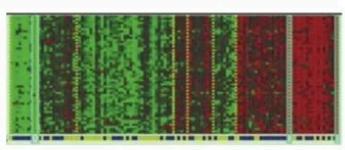


Grouping similar news

- Growing skills
- Develop career
- Stay updated with AI, understand how it affects your field of work



Market segmentation

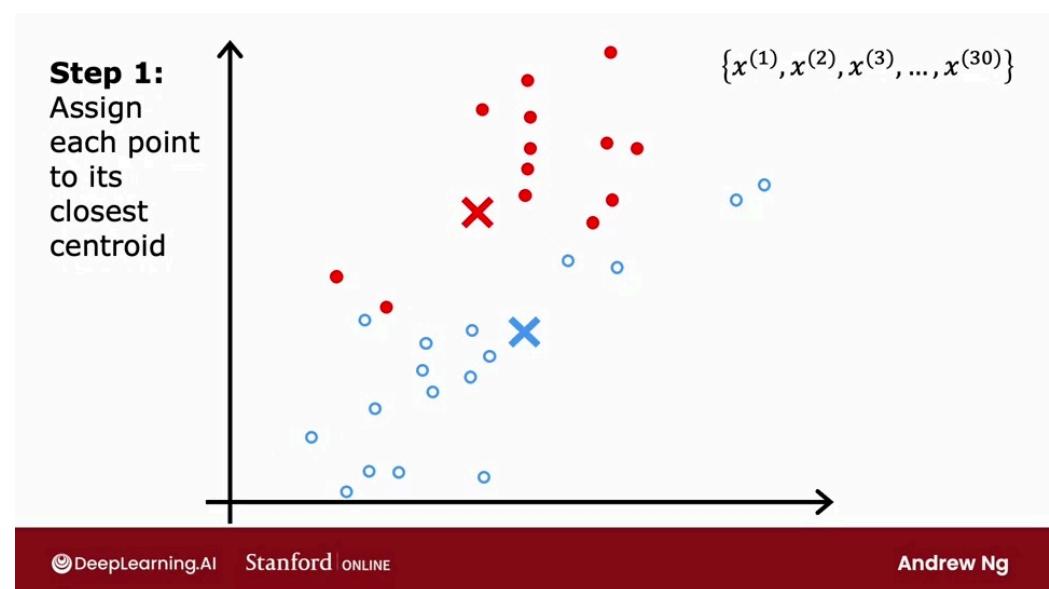
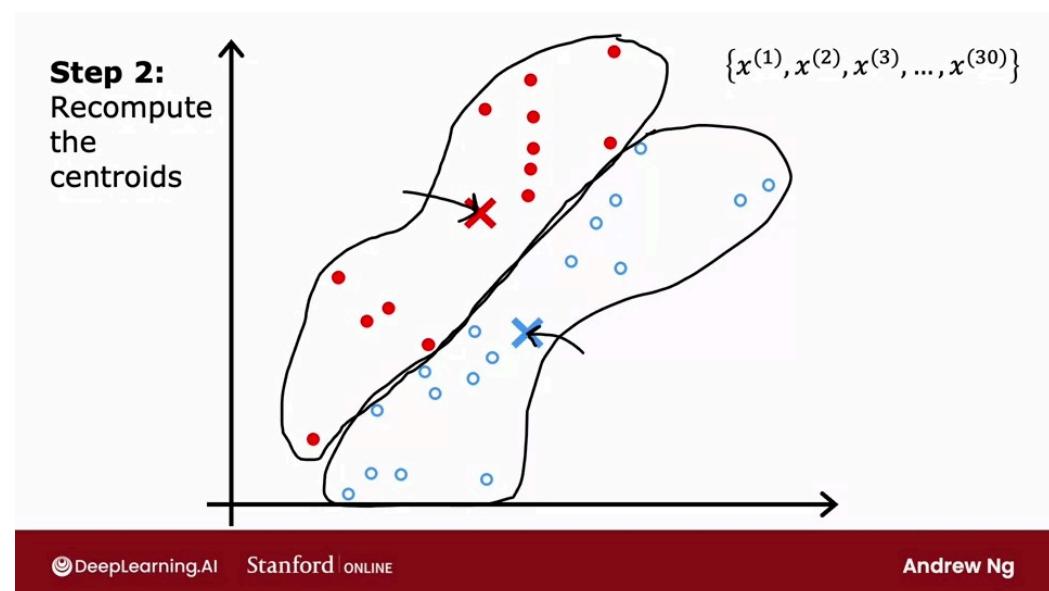
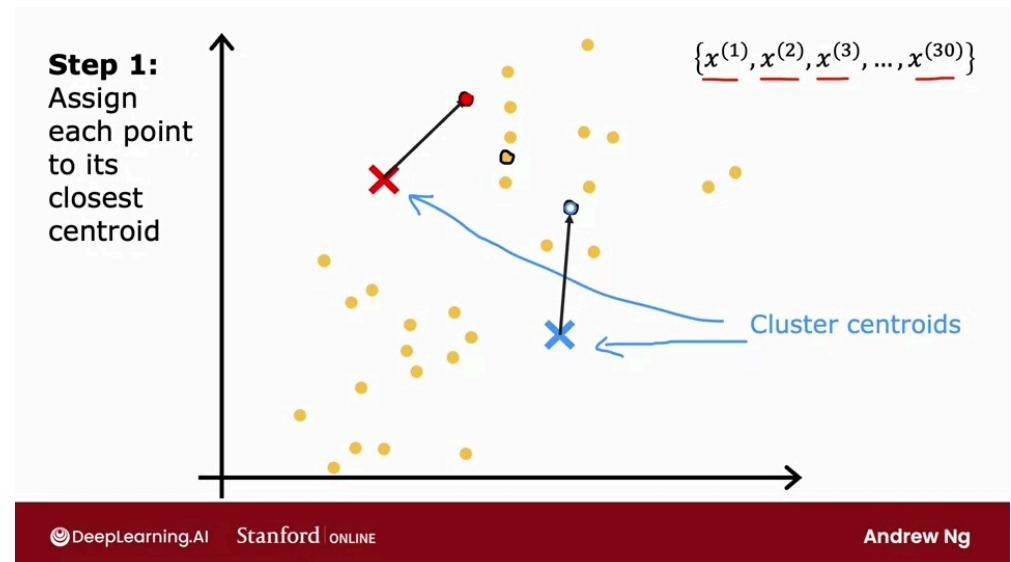


DNA analysis



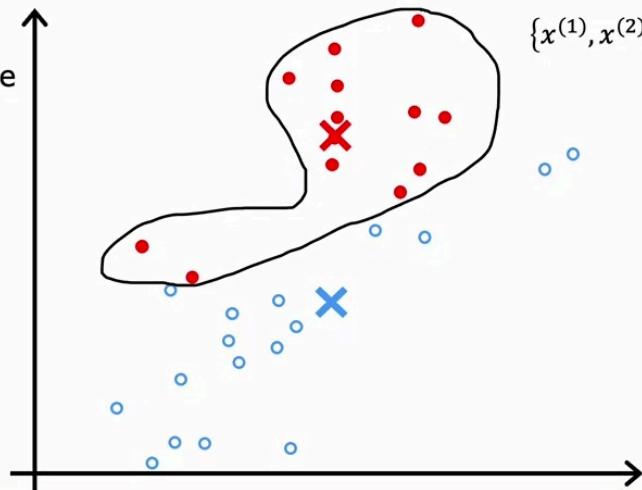
Astronomical data analysis

## K-means intuition



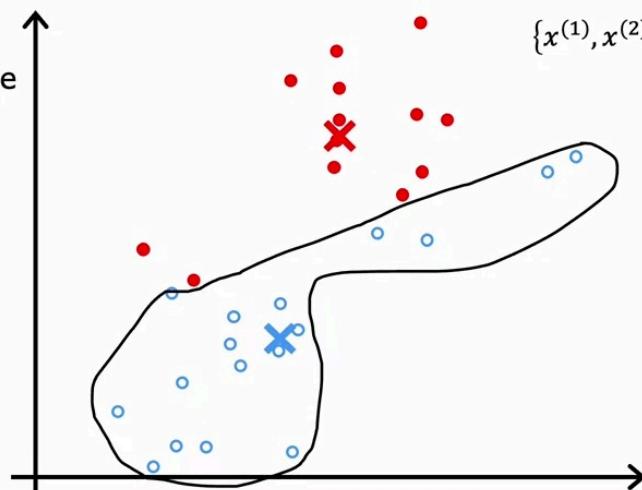
**Step 2:**  
Recompute  
the  
centroids

$$\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(30)}\}$$



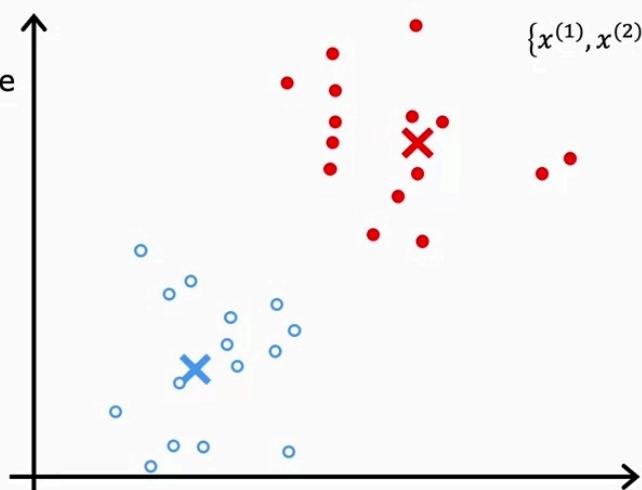
**Step 2:**  
Recompute  
the  
centroids

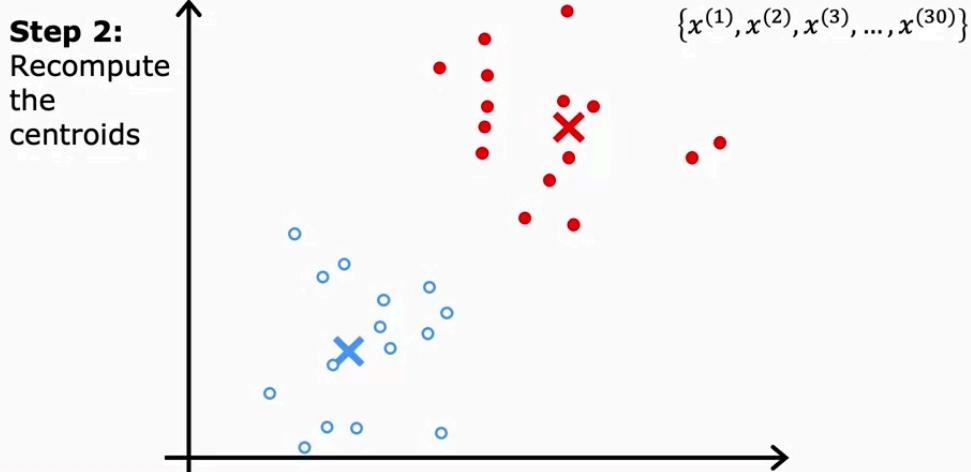
$$\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(30)}\}$$



**Step 2:**  
Recompute  
the  
centroids

$$\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(30)}\}$$





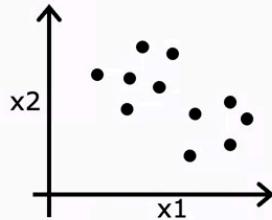
### K-means algorithm

#### K-means algorithm

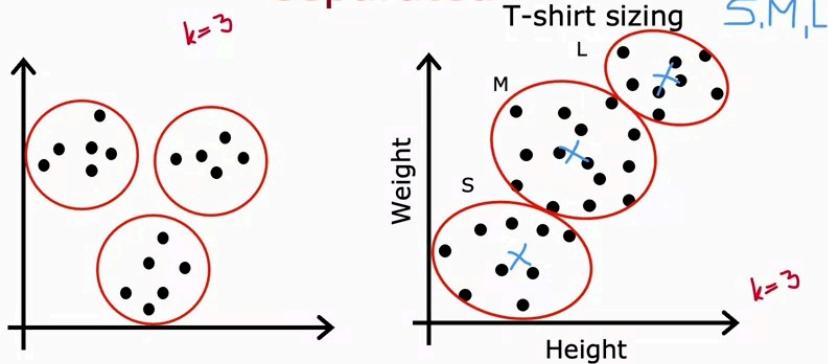
$$\mu_1, \mu_2 \quad x^{(1)}, x^{(2)}, \dots, x^{(30)} \\ n=2 \quad K=2$$

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K$

```
Repeat {
    # Assign points to cluster centroids
    for  $i = 1$  to  $m$ 
         $c^{(i)} :=$  index (from 1 to  $K$ ) of cluster
            centroid closest to  $x^{(i)}$ 
    # Move cluster centroids
    for  $k = 1$  to  $K$ 
         $\mu_k :=$  average (mean) of points assigned to cluster  $k$ 
}
```



### K-means for clusters that are not well separated



## Optimization objective

### K-means optimization objective

$c^{(i)}$  = index of cluster ( $1, 2, \dots, K$ ) to which example  $x^{(i)}$  is currently assigned

$\mu_k$  = cluster centroid  $k$

$\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned

#### Cost function

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

$$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

Distortion

### Cost function for K-means

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Repeat {

# Assign points to cluster centroids

for  $i = 1$  to  $m$

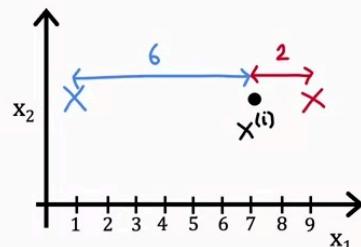
$c^{(i)}$  := index of cluster centroid closest to  $x^{(i)}$

# Move cluster centroids

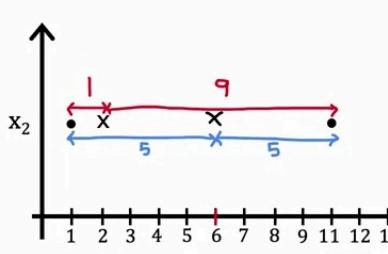
for  $k = 1$  to  $K$

$\mu_k$  := average of points in cluster  $k$

}



### Moving the centroid



$$\frac{1}{2}(1^2 + 7^2) = \frac{1}{2}(1+49) = 25$$

$$\frac{1}{2}(1 + 11) = 6$$

$$\frac{1}{2}(5^2 + 5^2) = 25$$

## Initializing K-means

### K-means algorithm

Step 0: Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_1, \dots, \mu_k$

Repeat {

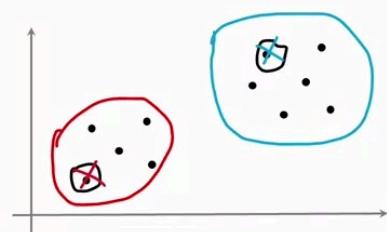
Step 1: Assign points to cluster centroids

Step 2: Move cluster centroids

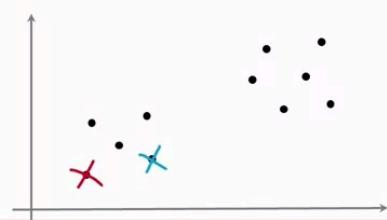
}

### Random initialization

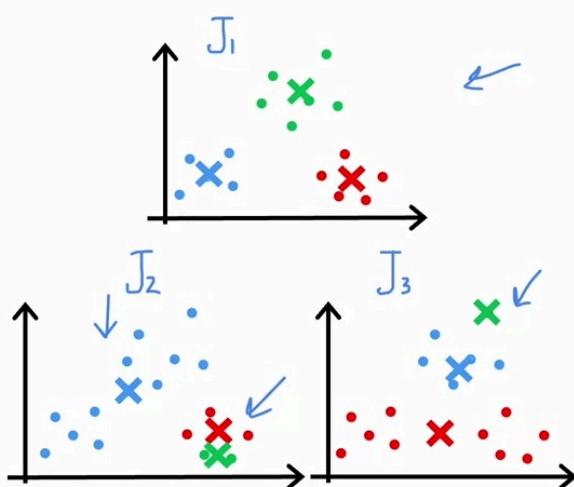
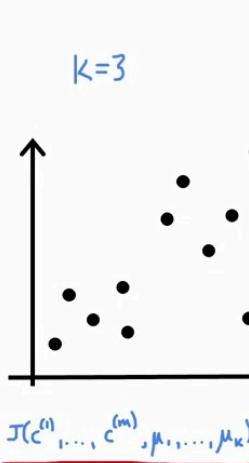
Choose  $K < m$

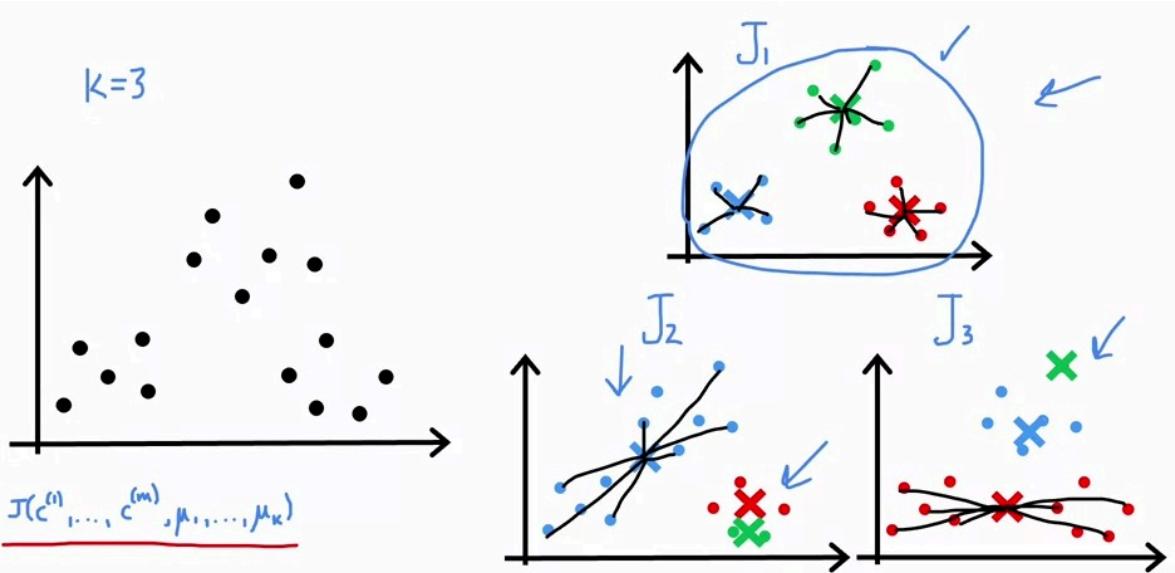


Randomly pick  $K$  training examples.



Set  $\mu_1, \mu_1, \dots, \mu_k$  equal to these  $K$  examples.





## Random initialization

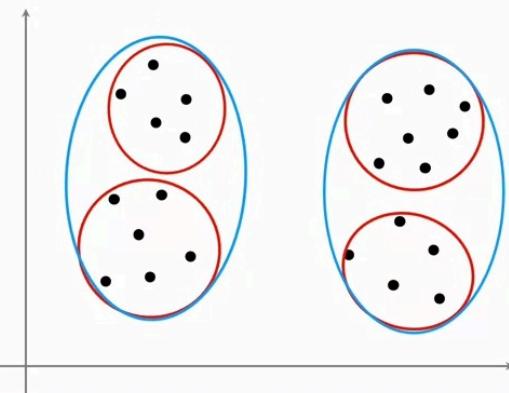
```

For i = 1 to 100 { 50-1000
    Randomly initialize K-means. k random examples
    Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k$  ←
    Computer cost function (distortion)
     $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k)$  ←
}
} ← Pick set of clusters that gave lowest cost J

```

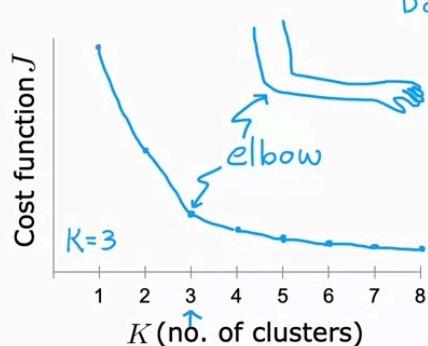
## Choosing the number of clusters

What is the right value of K?

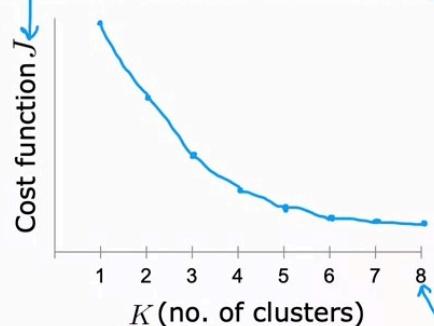


## Choosing the value of K

Elbow method

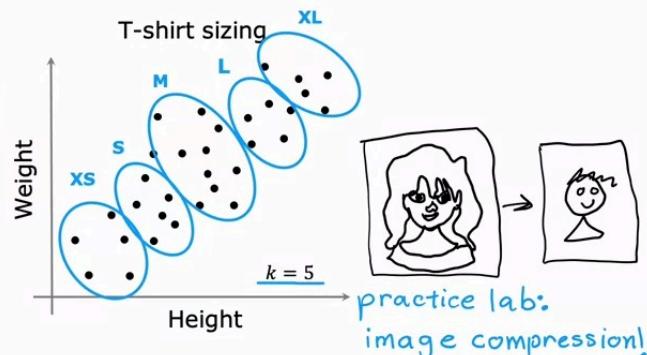
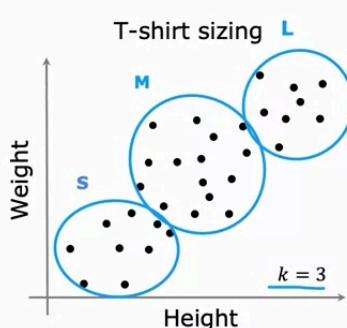


the right "K" is often ambiguous  
Don't choose K just to minimize cost J



## Choosing the value of K

Often, you want to get clusters for some later (downstream) purpose.  
Evaluate K-means based on how well it performs on that later purpose.



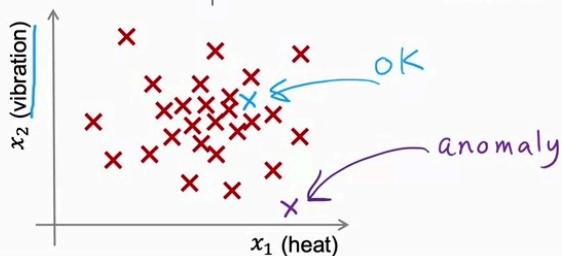
## Anomaly detection

### Anomaly detection example

Aircraft engine features:  
 $x_1$  = heat generated  
 $x_2$  = vibration intensity  
 $\dots$

Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

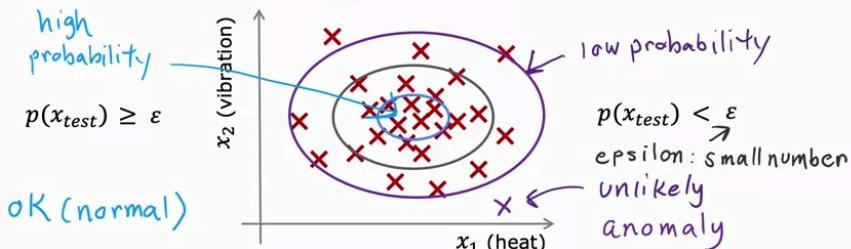
New engine:  $x_{test}$



### Density estimation

Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  probability of  $x$  being seen in dataset  
 Model  $p(x)$

Is  $x_{test}$  anomalous?



### Anomaly detection example

Fraud detection:

- $x^{(i)}$  = features of user  $i$ 's activities
- Model  $p(x)$  from data.
- Identify unusual users by checking which have  $p(x) < \epsilon$

how often log in?  
 how many web pages visited?  
 transactions?  
 posts? typing speed?

perform additional checks to identify real fraud vs. false alarms

Manufacturing:

$x^{(i)}$  = features of product  $i$

airplane engine  
 circuit board  
 smartphone

ratios

Monitoring computers in a data center:

- $x^{(i)}$  = features of machine  $i$
- $x_1$  = memory use,
- $x_2$  = number of disk accesses/sec,
- $x_3$  = CPU load,
- $x_4$  = CPU load/network traffic.

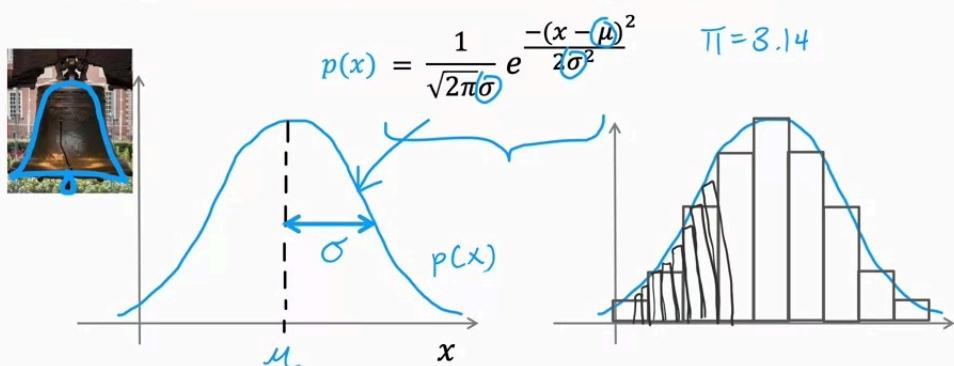
## Gaussian (normal) distribution

### Gaussian (Normal) distribution

Say  $x$  is a number.

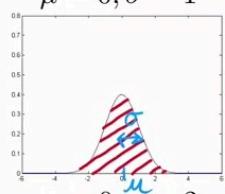
Probability of  $x$  is determined by a Gaussian with mean  $\mu$ , variance  $\sigma^2$ .

$\sigma$  standard deviation  
 $\sigma^2$  variance

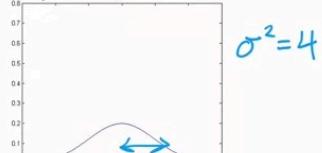


### Gaussian distribution example

$$\mu = 0, \sigma = 1$$

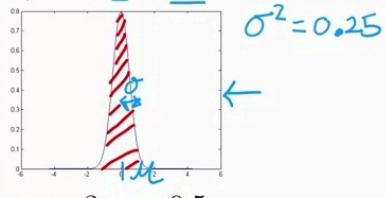


$$\mu = 0, \sigma = 2$$



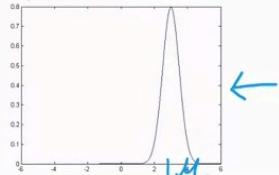
$$\sigma^2 = 4$$

$$\mu = 0, \sigma = 0.5$$



$$\sigma^2 = 0.25$$

$$\mu = 3, \sigma = 0.5$$



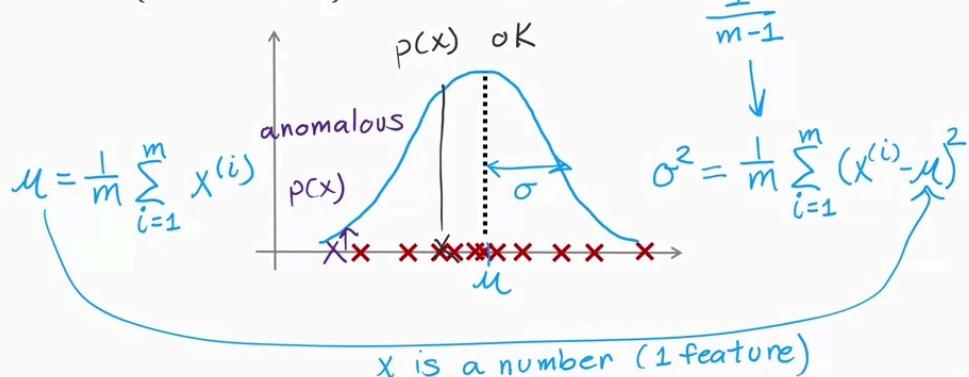
### Parameter estimation

Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

maximum likelihood

for  $\mu, \sigma$

$$\frac{1}{m-1}$$



$x$  is a number (1 feature)

## Anomaly detection algorithm

### Density estimation

Training set:  $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$   
Each example  $\vec{x}^{(i)}$  has  $n$  features

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \dots * p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) \quad \sum \text{"add"} \quad \prod \text{"multiply"}$$

$$\begin{aligned} p(x_1 = \text{high temp}) &= 1/10 \\ p(x_2 = \text{high vibra}) &= 1/20 \\ p(x_1, x_2) &= p(x_1) * p(x_2) \\ &= \frac{1}{10} * \frac{1}{20} = \frac{1}{200} \end{aligned}$$

## Anomaly detection algorithm

- Choose  $n$  features  $x_i$  that you think might be indicative of anomalous examples.

- Fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

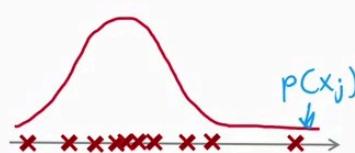
Vectorized formula

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$$

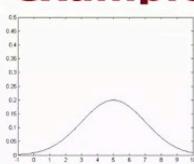
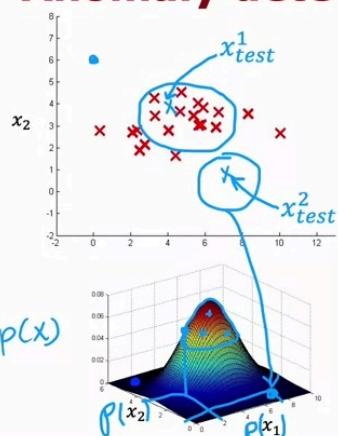
- Given new example  $x$ , compute  $p(x)$ :

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if  $p(x) < \varepsilon$



## Anomaly detection example



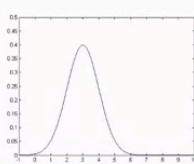
$$\mu_1 = 5, \sigma_1 = 2$$

$$p(x_1; \mu_1, \sigma_1^2)$$

$$\varepsilon = 0.02$$

$$p(x_1^{(1)}) = 0.0426 \rightarrow \text{"ok"}$$

$$p(x_1^{(2)}) = 0.0021 \rightarrow \text{anomaly}$$



$$\mu_2 = 3, \sigma_2 = 1$$

$$p(x_2; \mu_2, \sigma_2^2)$$

## Developing and evaluating an anomaly detection system

### The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

Assume we have some labeled data, of anomalous and non-anomalous examples.

$$y=1 \quad y=0$$

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal examples/not anomalous)

$y=0$  for all training examples

Cross validation set:  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

Test set:  $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

} include a few anomalous examples  
 $y=1$

mostly normal examples  
 $y=0$

### Aircraft engines monitoring example

10000 good (normal) engines  
20 flawed engines (anomalous)  
2  $y=0$   $y=1$

Training set: 6000 good engines train algorithm on training set

CV: 2000 good engines ( $y = 0$ ) 10 anomalous ( $y = 1$ )  
use cross validation set tune  $\epsilon$  tune  $x_j$   
Test: 2000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

Alternative: No test set use if very few labeled anomalous examples

Training set: 6000 good engines 2 higher risk of overfitting

CV: 4000 good engines ( $y = 0$ ), 20 anomalous ( $y = 1$ )  
tune  $\epsilon$  tune  $x_j$

### Algorithm evaluation

course 2 week 3  
skewed datasets

Fit model  $p(x)$  on training set  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$   
On a cross validation/test example  $x$ , predict

$$y = \begin{cases} 1 & \text{if } p(x) < \epsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \epsilon \text{ (normal)} \end{cases}$$

10  
2000

Possible evaluation metrics:

- True positive, false positive, false negative, true negative
- Precision/Recall
- $F_1$ -score

Use cross validation set

## Anomaly detection vs. supervised learning

### Anomaly detection    vs.    Supervised learning

Very small number of positive examples ( $y = 1$ ). (0-20 is common).  
Large number of negative ( $y = 0$ ) examples.

$p(x)$

$y=1$

Many different “types” of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.

Fraud

Large number of positive and negative examples.

20 positive examples

Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set.

Spam

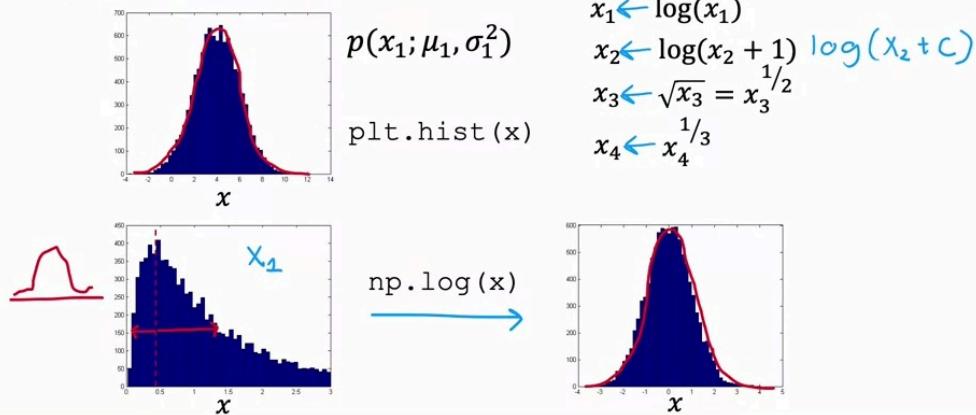
### Anomaly detection    vs.    Supervised learning

- Fraud detection
- Manufacturing - Finding new previously unseen defects in manufacturing.(e.g. aircraft engines)
- Monitoring machines in a data center
- ⋮

- Email spam classification
- Manufacturing - Finding known, previously seen defects  $y=1$  scratches
- Weather prediction (sunny/rainy/etc.)
- Diseases classification
- ⋮

## Choosing what features to use

### Non-gaussian features

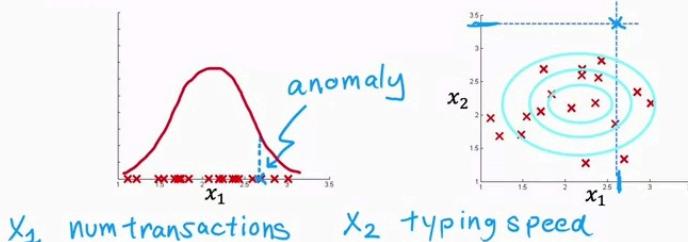


### Error analysis for anomaly detection

Want  $p(x) \geq \epsilon$  large for normal examples  $x$ .  
 $p(x) < \epsilon$  small for anomalous examples  $x$ .

Most common problem:

$p(x)$  is comparable for normal and anomalous examples.  
 $(p(x)$  is large for both)

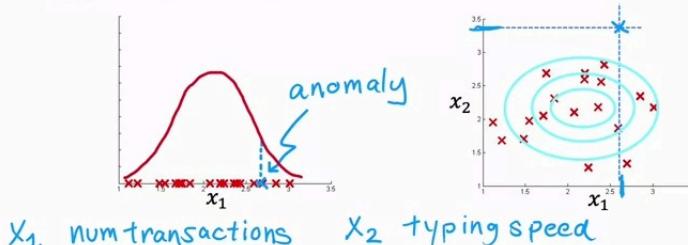


### Error analysis for anomaly detection

Want  $p(x) \geq \epsilon$  large for normal examples  $x$ .  
 $p(x) < \epsilon$  small for anomalous examples  $x$ .

Most common problem:

$p(x)$  is comparable for normal and anomalous examples.  
 $(p(x)$  is large for both)



## Week 2

### Collaborative filtering

#### Making recommendations

**Predicting movie ratings**

User rates movies using one to five stars

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$n_u = 4$        $r(1,1) = 1$   
 $n_m = 5$        $r(3,1) = 0$        $y^{(3,2)} = 4$



$n_u = \text{no. of users}$   
 $n_m = \text{no. of movies}$   
 $r(i,j) = 1$  if user  $j$  has rated movie  $i$   
 $y^{(i,j)}$  = rating given by user  $j$  to movie  $i$  (defined only if  $r(i,j)=1$ )

DeepLearning.AI Stanford ONLINE

Andrew Ng

#### Using per-item features

## Cost function

Notation:

- $r(i,j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)
  - $y^{(i,j)}$  = rating given by user  $j$  on movie  $i$  (if defined)
  - $w^{(j)}, b^{(j)}$  = parameters for user  $j$
  - $x^{(i)}$  = feature vector for movie  $i$
- For user  $j$  and movie  $i$ , predict rating:  $w^{(j)} \cdot x^{(i)} + b^{(j)}$
- $m^{(j)}$  = no. of movies rated by user  $j$
- To learn  $w^{(j)}, b^{(j)}$
- $$\min_{w^{(j)}, b^{(j)}} J(w^{(j)}, b^{(j)}) = \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^n (w_k^{(j)})^2$$
- number of features

DeepLearning.AI Stanford ONLINE

Andrew Ng

## Cost function

To learn parameters  $w^{(j)}, b^{(j)}$  for user  $j$ :

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (w_k^{(j)})^2$$

To learn parameters  $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$  for all users :

$$J\left(\begin{matrix} w^{(1)}, & \dots, & w^{(n_u)} \\ b^{(1)}, & \dots, & b^{(n_u)} \end{matrix}\right) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

DeepLearning.AI Stanford ONLINE

Andrew Ng

## Collaborative filtering algorithm

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$x_1$ (romance)	$x_2$ (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	$x^{(2)}$
Cute puppies of love	?	4	0	?	?	$x^{(3)}$
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ ,  $w^{(2)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ ,  $w^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$ ,  $w^{(4)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$  }  
 $b^{(1)} = 0$ ,  $b^{(2)} = 0$ ,  $b^{(3)} = 0$ ,  $b^{(4)} = 0$  ←

using  $w^{(j)} \cdot x^{(i)} + b^{(j)}$  →  $x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$   
 $\left. \begin{array}{l} w^{(1)} \cdot x^{(1)} \approx 5 \\ w^{(2)} \cdot x^{(1)} \approx 5 \\ w^{(3)} \cdot x^{(1)} \approx 0 \\ w^{(4)} \cdot x^{(1)} \approx 0 \end{array} \right\}$  ↑

## Cost function

Given  $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$

to learn  $x^{(i)}$ :

$$J(x^{(i)}) = \frac{1}{2} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^{(i)})^2$$

→ To learn  $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ :

$$J(x^{(1)}, x^{(2)}, \dots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

DeepLearning.AI Stanford ONLINE Andrew Ng

## Collaborative filtering

Cost function to learn  $w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}$ :

$$\min_{w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}} \frac{1}{2} \sum_{i=1}^{n_u} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

Cost function to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

Put them together:

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

DeepLearning.AI Stanford ONLINE Andrew Ng

## Binary labels: favs, likes and clicks

### Binary labels

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)
Love at last	1	1	0	0
Romance forever	1	? ←	? ←	0
Cute puppies of love	?	1 ←	0	? ←
Nonstop car chases	0	0	1	1
Swords vs. karate	0	0	1	? ←
	1 0 ?			

### Example applications

- 1. Did user  $j$  purchase an item after being shown? 1, 0, ?
- 2. Did user  $j$  fav/like an item? 1, 0, ?
- 3. Did user  $j$  spend at least 30sec with an item? 1, 0, ?
- 4. Did user  $j$  click on an item? 1, 0, ?

Meaning of ratings:

- 1 - engaged after being shown item
- 0 - did not engage after being shown item
- ? - item not yet shown

### From regression to binary classification

- Previously:
- Predict  $y^{(i,j)}$  as  $w^{(j)} \cdot x^{(i)} + b^{(j)}$
- For binary labels:  
 Predict that the probability of  $y^{(i,j)} = 1$   
 is given by  $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$   
 where  $g(z) = \frac{1}{1+e^{-z}}$

## Cost function for binary application

Previous cost function:

$$\frac{1}{2} \sum_{(i,j):r(i,j)=1} (f(x) - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2$$

Loss for binary labels  $y^{(i,j)}: f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1 - y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

Loss for single example

$$J(w, b, x) = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

cost for all examples

## Recommender systems implementation detail

### Users who have not rated any movies

Movie	Alice(1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)	
Love at last	5	5	0	0	?	5 5 0 0 ?
Romance forever	5	?	?	0	?	5 ? ? 0 ?
Cute puppies of love	?	4	0	?	?	? 4 0 ? ?
Nonstop car chases	0	0	5	4	?	0 0 5 4 ?
Swords vs. karate	0	0	5	?	?	0 0 5 0 ?

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} \frac{1}{2} \sum_{(i,j):r(i,j)=1} (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

$$w^{(s)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(s)} = 0 \quad w^{(s)} \cdot x^{(s)} + b^{(s)}$$

### Mean Normalization

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \xrightarrow{\text{mean normalization}} \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:  $w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$

$$w^{(s)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(s)} = 0 \quad \underbrace{w^{(s)} \cdot x^{(s)} + b^{(s)}}_0 + \mu_1 = 2.5$$

User 5 (Eve):

## TensorFlow implementation of collaborative filtering

$J = (wx - 1)^2$

Gradient descent algorithm  
Repeat until convergence  
 $w = w - \alpha \frac{\partial J(w,b)}{\partial w}$

Fix  $b = 0$  for this example

```

Custom Training Loop
w = tf.Variable(3.0)
x = 1.0
y = 1.0 # target value
alpha = 0.01

iterations = 30
for iter in range(iterations):
    # Use TensorFlow's Gradient tape to record the steps
    # used to compute the cost J, to enable auto differentiation.
    with tf.GradientTape() as tape:
        fwb = w*x
        costJ = (fwb - y)**2

    # Use the gradient tape to calculate the gradients
    # of the cost with respect to the parameter w.
    dJdw = tape.gradient(costJ, [w])

    # Run one step of gradient descent by updating
    # the value of w to reduce the cost.
    w.assign_add(-alpha * dJdw)

```

Tf.variables are the parameters we want to optimize

Auto Diff      Auto Grad

$\frac{\partial}{\partial w} J(w)$

tf.variables require special function to modify

## Implementation in TensorFlow

Gradient descent algorithm

Repeat until convergence

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

```

# Instantiate an optimizer.
optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iterations = 200
for iter in range(iterations):
    # Use TensorFlow's GradientTape
    # to record the operations used to compute the cost
    with tf.GradientTape() as tape:
        # Compute the cost (forward pass is included in cost)
        cost_value = cofiCostFunc(X, W, b, Ynorm, R,
                                   num_users, num_movies, lambda)
        n_u          n_m

    # Use the gradient tape to automatically retrieve
    # the gradients of the trainable variables with respect to
    # the loss
    grads = tape.gradient(cost_value, [X, W, b])

    # Run one step of gradient descent by updating
    # the value of the variables to minimize the loss.
    optimizer.apply_gradients(zip(grads, [X, W, b]))

```

Dataset credit: Harper and Konstan. 2015. The MovieLens Datasets: History and Context

## Finding related items

# Finding related items

The features  $x^{(i)}$  of item  $i$  are quite hard to interpret.

To find other items related to it,

find item  $k$  with  $x^{(k)}$  similar to  $x^{(i)}$

i.e. with smallest distance

$$\sum_{l=1}^n (x_l^{(k)} - x_l^{(i)})^2$$
$$\|x^{(k)} - x^{(i)}\|^2$$



## Limitations of Collaborative Filtering

→ **Cold start problem.** How to

- rank new items that few users have rated?
- show something reasonable to new users who have rated few items?

→ Use side information about items or users:

- Item: Genre, movie stars, studio, ....
- User: Demographics (age, gender, location), expressed preferences, ...

## Collaborative filtering vs Content-based filtering

### Collaborative filtering vs Content-based filtering

#### → Collaborative filtering:

Recommend items to you based on ratings of users who gave similar ratings as you

#### → Content-based filtering:

Recommend items to you based on features of user and item to find good match

### Collaborative filtering vs Content-based filtering

#### → Collaborative filtering:

Recommend items to you based on ratings of users who gave similar ratings as you

#### → Content-based filtering:

Recommend items to you based on features of user and item to find good match

$r(i,j) = 1$  if user  $j$  has rated item  $i$

$y^{(i,j)}$  rating given by user  $j$  on item  $i$  (if defined)

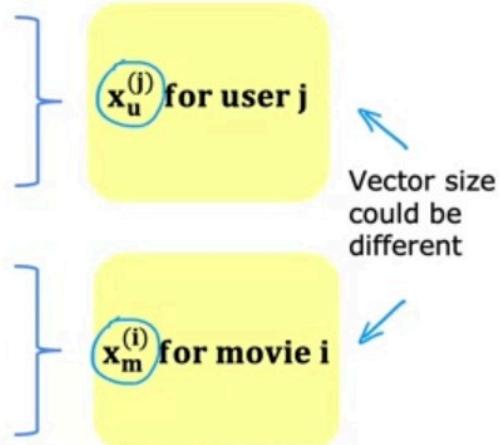
## Examples of user and item features

### User features:

- • Age
- • Gender (1 hot)
- • Country (1 hot, 200)
- • Movies watched (1000)
- • Average rating per genre
- ...

### Movie features:

- • Year
- • Genre/Genres
- • Reviews
- • Average rating
- ...



## Content-based filtering: Learning to match

Predict rating of user  $j$  on movie  $i$  as

$$w^{(j)} \cdot x^{(i)} + b^{(j)}$$

computed from  $x_u^{(j)}$

computed from  $x_m^{(i)}$

user's preferences

$$V_u^{(j)} = \begin{bmatrix} 4.9 \\ 0.1 \\ \vdots \\ 3.0 \end{bmatrix}$$

likes

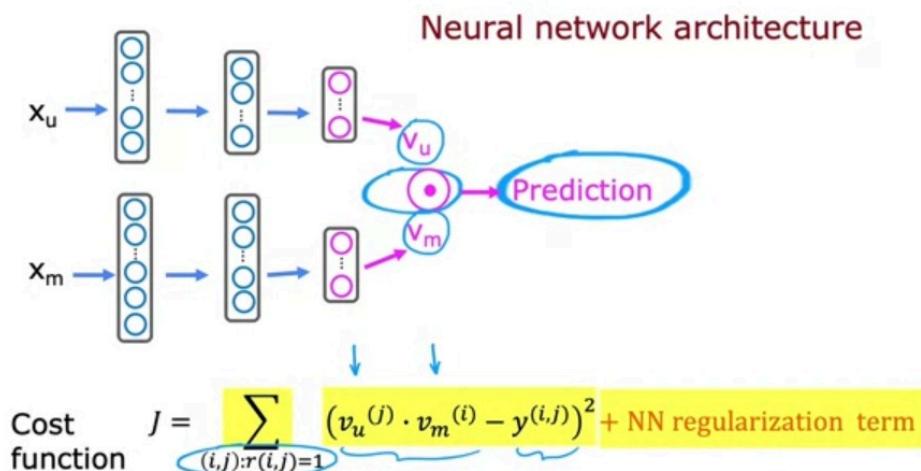
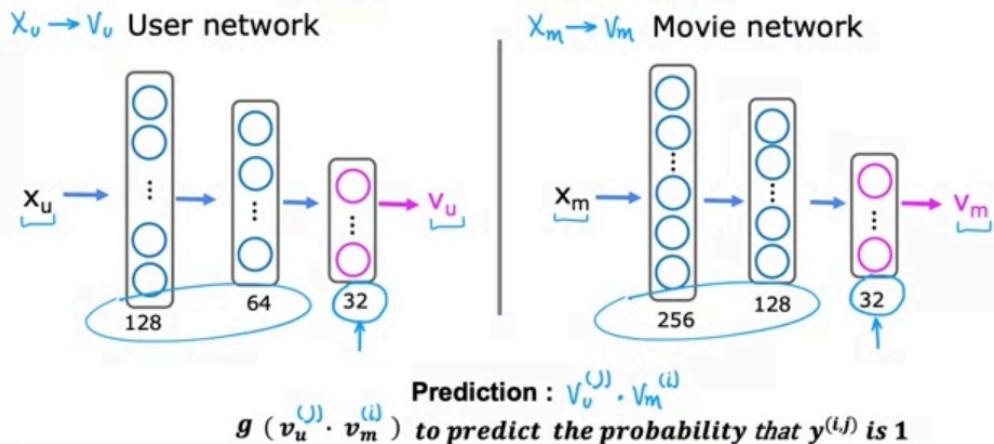
movie features

$$V_m^{(i)} = \begin{bmatrix} 4.5 \\ 0.2 \\ \vdots \\ 3.5 \end{bmatrix}$$

romance action

## Deep learning for content-based filtering

### Neural network architecture



### Learned user and item vectors:

- $v_u^{(j)}$  is a vector of length 32 that describes user j with features  $x_u^{(j)}$
- $v_m^{(i)}$  is a vector of length 32 that describes movie i with features  $x_m^{(i)}$

To find movies similar to movie i:

$$\| (V_m^{(k)} - V_m^{(i)}) \|_2^2 \text{ small}$$

$$\| X^{(k)} - X^{(i)} \|_2^2$$

Note: This can be pre-computed ahead of time

## Recommending from a large catalogue

### Two steps: Retrieval & Ranking

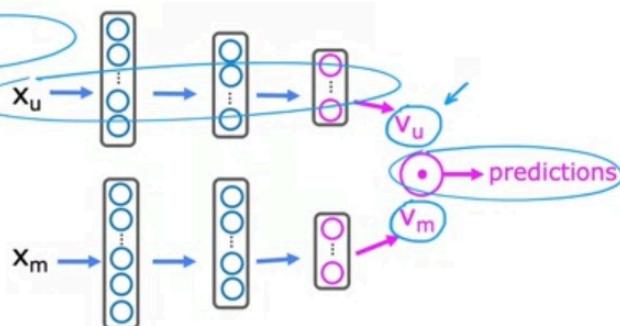
#### Retrieval:

- • Generate large list of plausible item candidates  $\sim 100s$
- e.g.
  - 1) For each of the last 10 movies watched by the user, find 10 most similar movies  
$$\| v_m^{(k)} - v_m^{(i)} \|^2$$
  - 2) For most viewed 3 genres, find the top 10 movies
  - 3) Top 20 movies in the country
- • Combine retrieved items into list, removing duplicates and items already watched/purchased

### Two steps: Retrieval & ranking

#### Ranking:

- Take list retrieved and rank using learned model



- Display ranked items to user

### Retrieval step

- • Retrieving more items results in better performance, but slower recommendations.
- • To analyse/optimize the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations (i.e.,  $p(y^{(i,j)}) = 1$  of items displayed to user are higher).

100    500

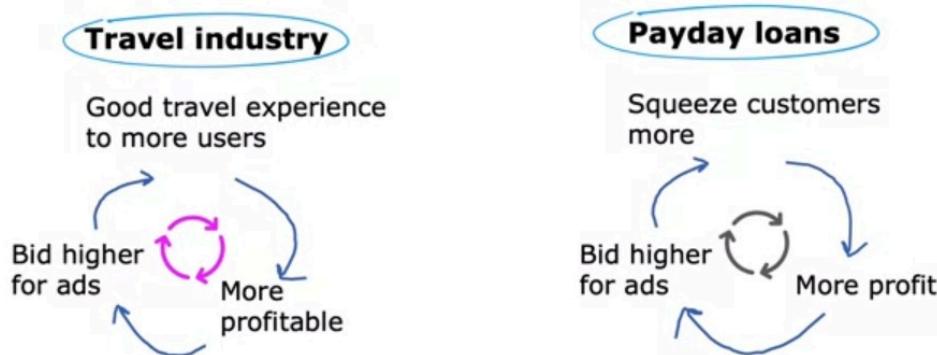
## Ethical use of recommender systems

### What is the goal of the recommender system?

Recommend:

- • Movies most likely to be rated 5 stars by user
- • Products most likely to be purchased
- • Ads most likely to be clicked on *+high bid*
- • Products generating the largest profit
- • Video leading to maximum watch time

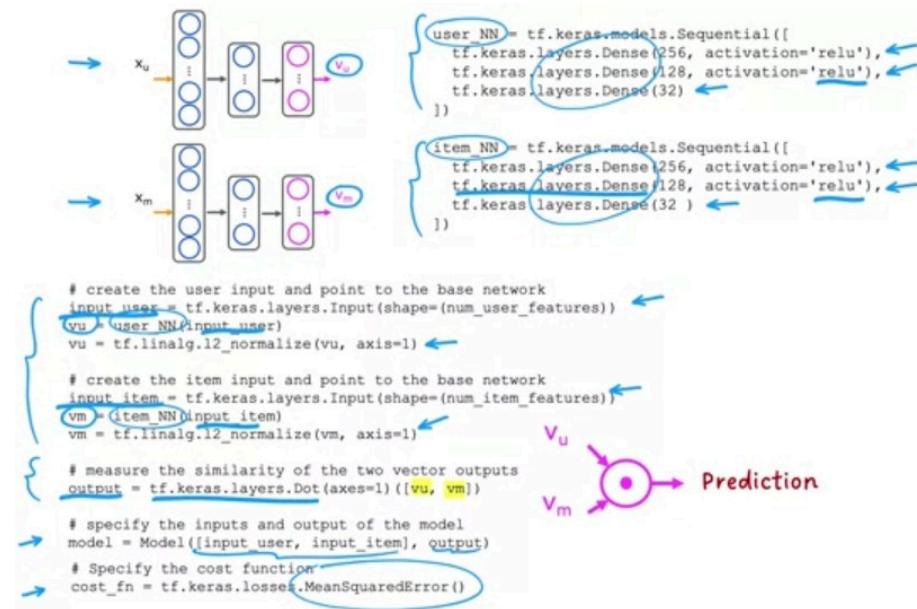
### Ethical considerations with recommender systems



### Other problematic cases:

- • Maximizing user engagement (e.g. watch time) has led to large social media/video sharing sites to amplify conspiracy theories and hate/toxicity
- Amelioration : Filter out problematic content such as hate speech, fraud, scams and violent content
- • Can a ranking system maximize your profit rather than users' welfare be presented in a transparent way?
- Amelioration : Be transparent with users

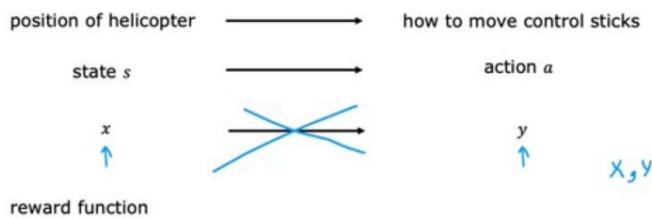
## TensorFlow implementation of content-based filtering



## Week 3

### Reinforcement learning introduction

## Reinforcement Learning

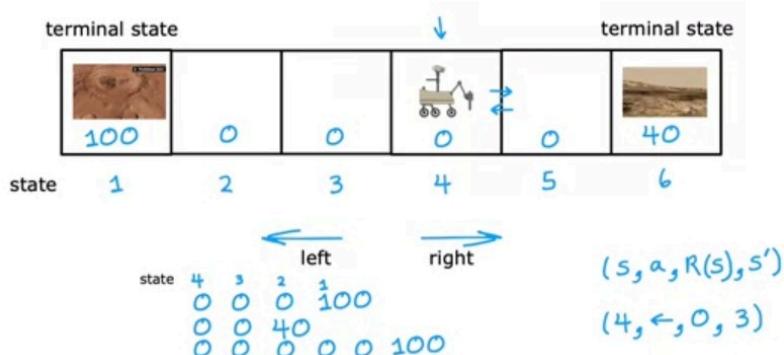


DeepLearning.AI Stanford ONLINE

Andrew Ng

### Mars rover example

## Mars Rover Example



[Credit: Jagriti Agrawal, Emma Brunskill]

DeepLearning.AI Stanford ONLINE

Andrew Ng

## The Return in reinforcement learning

### Return

	0	0		0	40
state 1	2	3	4	5	6

$$\text{Return} = 0 + (0.9)0 + (0.9)^20 + (0.9)^3100 = 0.729 \times 100 = 72.9$$

$$\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \quad (\text{until terminal state})$$

$$\text{Discount Factor } \gamma = 0.9 \quad 0.99 \quad 0.999$$

$$\gamma = 0.5$$

$$\text{Return} = 0 + (0.5)0 + (0.5)^20 + (0.5)^3100 = 12.5$$

### Example of Return

100	50	25	12.5	6.25	40
100	0	0	0	0	40
1	2	3	4	5	6

← return       $\gamma = 0.5$   
← reward

The return depends on the actions you take.

100	2.5	5	10	20	40
100	0	0	0	0	40
1	2	3	4	5	6

$0 + (0.5)0 + (0.5)^240 = 10$

100	50	25	12.5	20	40
100	0	0	0	0	40
1	2	3	4	5	6

$0 + (0.5)40 = 20$

## Policies in reinforcement learning

### Policy

state s	$\pi_a$	policy	action	100	←	←	→	→	40
				100	←	←	←	←	40
				100	→	→	→	→	40
				100	←	←	←	→	40

$\pi(s) -$   
 $\pi(2) = \leftarrow$   
 $\pi(3) = \leftarrow$   
 $\pi(4) = \leftarrow$   
 $\pi(5) = \rightarrow$

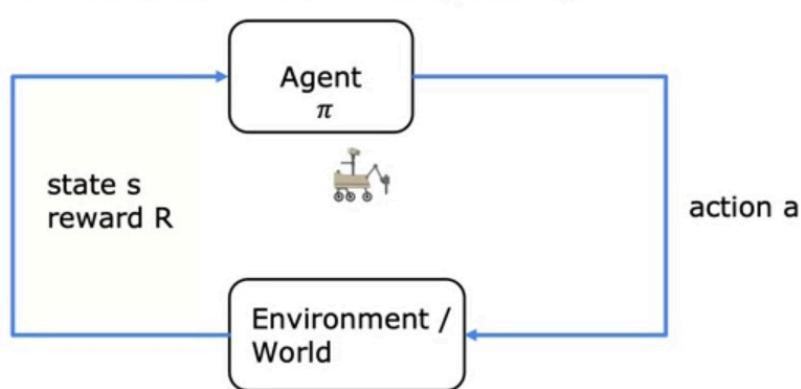
# The goal of reinforcement learning



Find a policy  $\pi$  that tells you what action ( $a = \pi(s)$ ) to take in every state ( $s$ ) so as to maximize the return.

		Mars rover	Helicopter	Chess
↳ states	6 states		position of helicopter	pieces on board
↳ actions	$\leftarrow \rightarrow$		how to move control stick	possible move
↳ rewards	$100, 0, 40$		$+1, -1000$	$+1, 0, -1$
↳ discount factor $\gamma$	0.5		0.99	0.995
↳ return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$		$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$
↳ policy $\pi$			Find $\pi(s) = a$	Find $\pi(s) = a$

## Markov Decision Process (MDP)



## State-action value function definition

# State action value function (Q-function)

$Q(s, a) =$  Return if you  
 • start in state  $s$ .  
 • take action  $a$  (once).  
 • then behave optimally after that.

$Q(s, a)$

100	50	25	12.5	20	40
100	0	0	0	0	40

$Q(2, \leftarrow)$ $Q(2, \rightarrow)$					
100	50	25	12.5	20	40
100	0	0	0	0	40

← return  
← action  
← reward

$$Q(2, \rightarrow) = 12.5 \\ 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$$

$$Q(2, \leftarrow) = 50 \\ 0 + (0.5)100$$

$$Q(4, \leftarrow) = 12.5 \\ 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$$

## Picking actions

→	100	50	25	12.5	20	40
100	0	0	0	0	0	40
100	100	50	12.5	25	6.25	12.5

← return  
← action  
← reward

$$\max_a Q(s, a)$$

$$\pi(s) = a$$

$Q(s, a) =$  Return if you  
 • start in state  $s$ .  
 • take action  $a$  (once).  
 • then behave optimally after that.

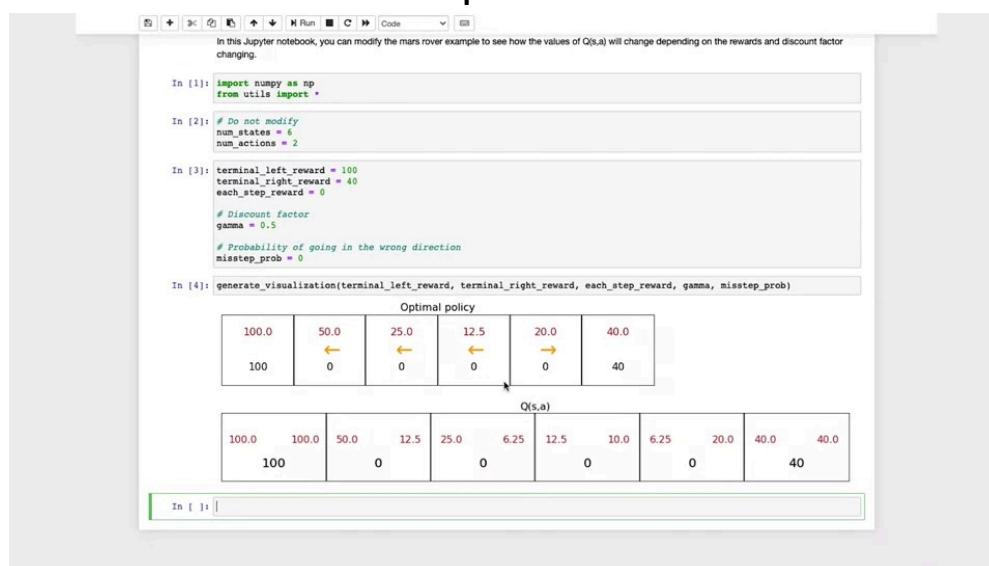
The best possible return from state  $s$  is  $\max_a Q(s, a)$ .

$Q^*$

The best possible action in state  $s$  is the action  $a$  that gives  $\max_a Q(s, a)$ .

Optimal  $Q$  function

## State-action value function example



In this Jupyter notebook, you can modify the mars rover example to see how the values of  $Q(s, a)$  will change depending on the rewards and discount factor changing.

```

In [1]: import numpy as np
from utils import *

In [2]: # Do not modify
num_states = 6
num_actions = 2

In [3]: terminal_left_reward = 100
terminal_right_reward = 40
each_step_reward = 0

# Discount factor
gamma = 0.5

# Probability of going in the wrong direction
misstep_prob = 0

```

In [4]: generate\_visualization(terminal\_left\_reward, terminal\_right\_reward, each\_step\_reward, gamma, misstep\_prob)

Optimal policy

100.0	50.0	25.0	12.5	20.0	40.0
100	0	0	0	0	40

$Q(s, a)$

100.0	100.0	50.0	0	12.5	25.0	6.25	12.5	10.0	6.25	20.0	40.0	40.0
100			0		0		0		0		0	40

**State Action Value Function Example**

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

```
In [1]: import numpy as np
from utils import *

In [2]: # Do not modify
num_states = 6
num_actions = 2

In [5]: terminal_left_reward = 100
terminal_right_reward = 10
each_step_reward = 0

# Discount factor
gamma = 0.5

# Probability of going in the wrong direction
misstep_prob = 0

In [6]: generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	50.0	25.0	12.5	6.25	10.0
100	0	0	0	0	10

Q(s,a)

100.0	100.0	50.0	12.5	25.0	6.25	12.5	3.12	6.25	5.0	10.0	10.0
100		0		0		0		0		10	

**State Action Value Function Example**

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

```
In [1]: import numpy as np
from utils import *

In [2]: # Do not modify
num_states = 6
num_actions = 2

In [7]: terminal_left_reward = 100
terminal_right_reward = 40
each_step_reward = 0

# Discount factor
gamma = 0.9

# Probability of going in the wrong direction
misstep_prob = 0

In [8]: generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	90.0	81.0	72.9	65.61	40.0
100	0	0	0	0	40

Q(s,a)

100.0	100.0	90.0	72.9	81.0	65.61	72.9	59.05	65.61	36.0	40.0	40.0
100		0		0		0		0		40	

**State Action Value Function Example**

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

```
In [1]: import numpy as np
from utils import *

In [2]: # Do not modify
num_states = 6
num_actions = 2

In [9]: terminal_left_reward = 100
terminal_right_reward = 40
each_step_reward = 0

# Discount factor
gamma = 0.3

# Probability of going in the wrong direction
misstep_prob = 0

In [8]: generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	90.0	81.0	72.9	65.61	40.0
100	0	0	0	0	40

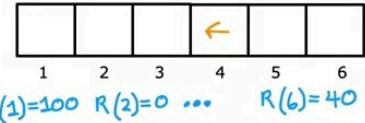
Q(s,a)

100.0	100.0	90.0	72.9	81.0	65.61	72.9	59.05	65.61	36.0	40.0	40.0
100		0		0		0		0		40	

## Bellman Equation

# Bellman Equation

$Q(s, a) =$  Return if you  
 • start in state  $s$ .  
 • take action  $a$  (once).  
 • then behave optimally after that.



$s$  : current state       $R(s)$  = reward of current state  
 $a$  : current action

$s'$  : state you get to after taking action  $a$   
 $a'$  : action that you take in state  $s'$

$$Q(s, a) = R(s) + \gamma \max_{a'} Q(s', a')$$

# Bellman Equation

$$Q(s, a) = R(s) + \gamma \max_{a'} Q(s', a')$$

$$Q(s, a) = R(s)$$

100	100	50	12.5	25	6.25	12.5	10	6.25	20	40	40
100	0	0	0	0	0	0	0	0	40	40	

$$\begin{aligned} s &= 2 \\ a &= \rightarrow \\ s' &= 3 \end{aligned}$$

$$\begin{aligned} Q(2, \rightarrow) &= R(2) + 0.5 \max_{a'} Q(3, a') \\ &= 0 + (0.5)25 = 12.5 \end{aligned}$$

$$\begin{aligned} Q(4, \leftarrow) &= R(4) + 0.5 \max_{a'} Q(3, a') \\ &= 0 + (0.5)25 = 12.5 \end{aligned}$$

$$\begin{aligned} s &= 4 \\ a &= \leftarrow \\ s' &= 3 \end{aligned}$$

# Explanation of Bellman Equation

$\left\{ \begin{array}{l} Q(s, a) = \text{Return if you} \\ \cdot \text{ start in state } s. \\ \cdot \text{ take action } a \text{ (once).} \\ \cdot \text{ then behave optimally after that.} \end{array} \right.$

$s \rightarrow s'$

→ The best possible return from state  $s$  is  $\max_{a'} Q(s', a')$

$$Q(s, a) = R(s) + \gamma \max_{a'} Q(s', a')$$

Reward you get right away      Return from behaving optimally starting from state  $s'$ .

$$\begin{aligned} Q(s, a) &= R_1 + \gamma [R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots] \\ &\quad \uparrow \quad \uparrow \quad \uparrow \end{aligned}$$

# Explanation of Bellman Equation

$$Q(s, a) = R(s) + \gamma \max_{a'} Q(s', a')$$

100	100	50	12.5	25	6.25	12.5	10	6.25	20	40	40
100	0	0	0	0	0	0	0	0	0	40	40
1	2	3	4	5	6						

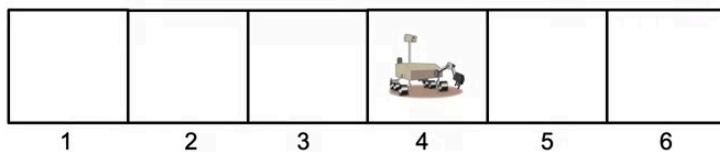
Handwritten annotations:

$$\begin{aligned} & Q(4, \leftarrow) \\ & = 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 \\ & = R(4) + (0.5)(0 + (0.5)0 + (0.5)^2 100) \\ & = R(4) + (0.5) \max_{a'} Q(3, a') \end{aligned}$$

continuous state space applications

## Discrete vs Continuous State

Discrete State:



Continuous State:



$$s = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

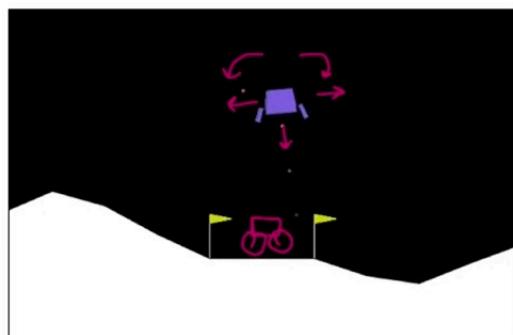
## Autonomous Helicopter



$$s = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \omega \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix}$$

## Lunar lander

# Lunar Lander



actions:

- do nothing
- left thruster
- main thruster
- right thruster

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

0 or 1 {

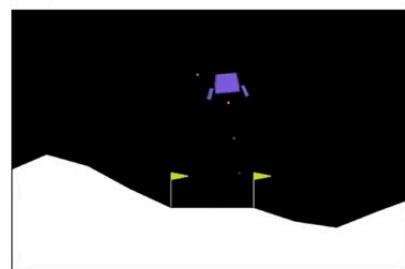
Andrew Ng

DeepLearning.AI Stanford ONLINE

## Lunar Lander Problem

Learn a policy  $\pi$  that, given

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$



picks action  $a = \pi(s)$  so as to maximize the return.

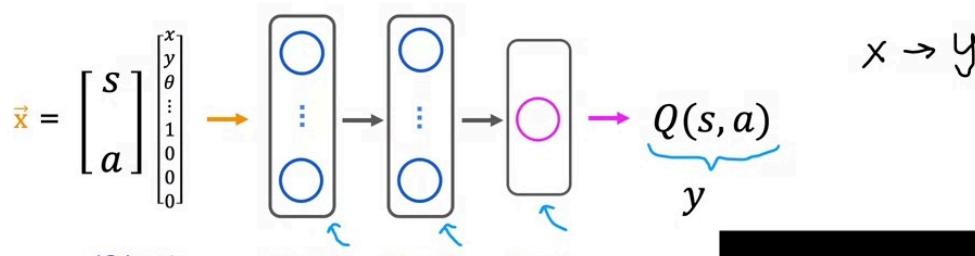
$$\gamma = 0.985$$

Andrew Ng

DeepLearning.AI Stanford ONLINE

## Learning the state-value function

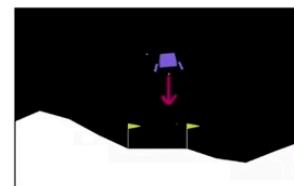
### Deep Reinforcement Learning



In a state  $s$ , use neural network to compute

$Q(s, \text{nothing}), Q(s, \text{left}), Q(s, \text{main})$  Q(s, right)

Pick the action  $a$  that maximizes  $Q(s, a)$



Andrew Ng

DeepLearning.AI Stanford ONLINE

## Bellman Equation

$$Q(s, a) = R(s) + \gamma \max_{a'} Q(s', a')$$

$f_{w,B}(x) \approx y$

$$(s, a, R(s), s')$$

$$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)}) \leftarrow$$

$$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)}) \leftarrow$$

$$(s^{(3)}, a^{(3)}, R(s^{(3)}), s'^{(3)}) \leftarrow$$

## Learning Algorithm

Initialize neural network randomly as guess of  $Q(s, a)$ .

Repeat {

    Take actions in the lunar lander. Get  $(s, a, R(s), s')$ .

    Store 10,000 most recent  $(s, a, R(s), s')$  tuples.



Replay Buffer

Train neural network:

    Create training set of 10,000 examples using

$$x = (s, a) \text{ and } y = R(s) + \gamma \max_{a'} Q(s', a')$$

    Train  $Q_{new}$  such that  $Q_{new}(s, a) \approx y$ .

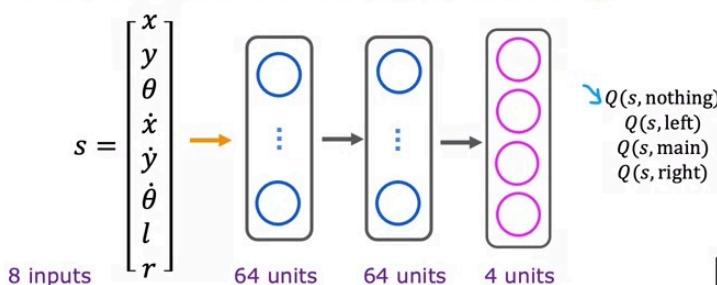
    Set  $Q = Q_{new}$ .

$$f_{w,B}(x) \approx y$$

$$\begin{aligned} &x, y & &x^{(1)}, y^{(1)} \\ &\vdots & &\vdots \\ &x^{(10000)}, y^{(10000)} & & \end{aligned}$$

### Algorithm refinement: Improved neural network architecture

## Deep Reinforcement Learning



In a state  $s$ , input  $s$  to neural network.

Pick the action  $a$  that maximizes  $Q(s, a)$ .



## Algorithm refinement: $\epsilon$ -greedy policy

# How to choose actions while still learning?

In some state  $s$

Option 1:

Pick the action  $a$  that maximizes  $Q(s, a)$ .

Option 2:

- With probability 0.95, pick the action  $a$  that maximizes  $Q(s, a)$ . Greedy, "Exploitation"
- With probability 0.05, pick an action  $a$  randomly. "Exploration"

$\epsilon$ -greedy policy ( $\epsilon = 0.05$ )  
0.95

$Q(s, \text{main})$  is low



Start  $\epsilon$  high  
 $1.0 \rightarrow 0.01$   
Gradually decrease

state of reinforcement learning

## Limitations of Reinforcement Learning

- Much easier to get to work in a simulation than a real robot!
- Far fewer applications than supervised and unsupervised learning.
- But ... exciting research direction with potential for future applications.