```
# 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)
```

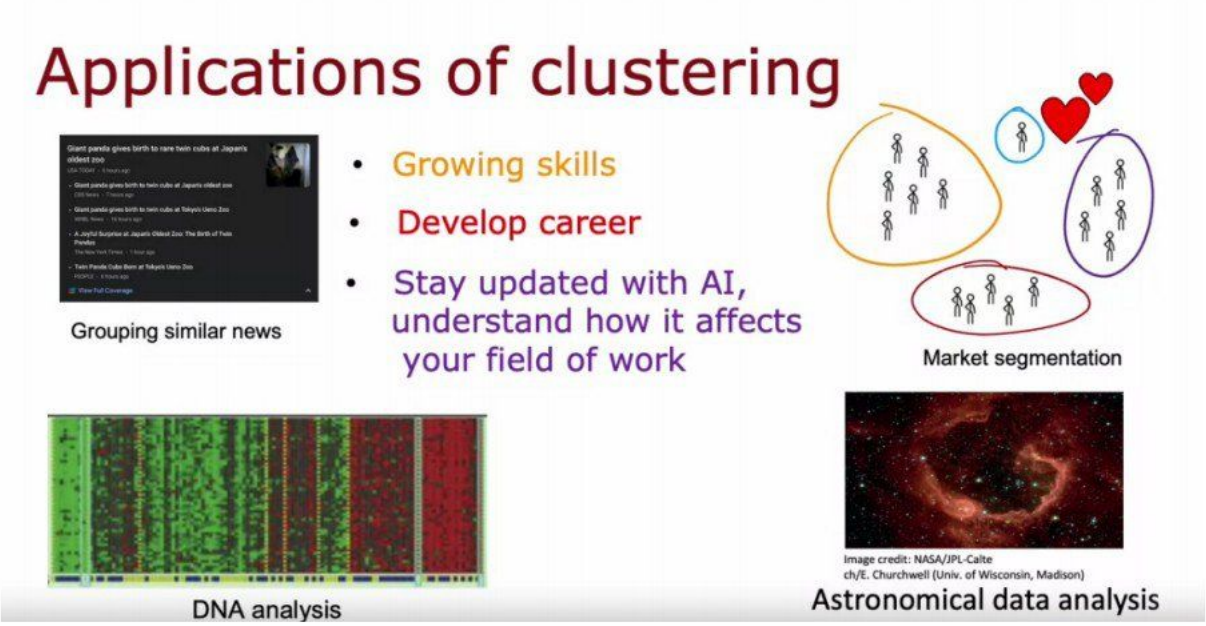
When to use Decision Trees

Decision Tree and Tree Ensembles	Neural Networks
Works well on tabular data	Works well on Tabular (Structured and Unstructured data)
Not recommended for Images, audio and text	Recommended for Image, audio, and text
fast	slower than DT
Small Decision tree may be human interpretable	works with transfer learning
We can train one decision tree at a time.	when building a system of multiple models working together, multiple NN can be stringed together easily. We can train them all together using gradient descent.

Unsupervised Learning, Recommenders, Reinforcement Learning - Course 3

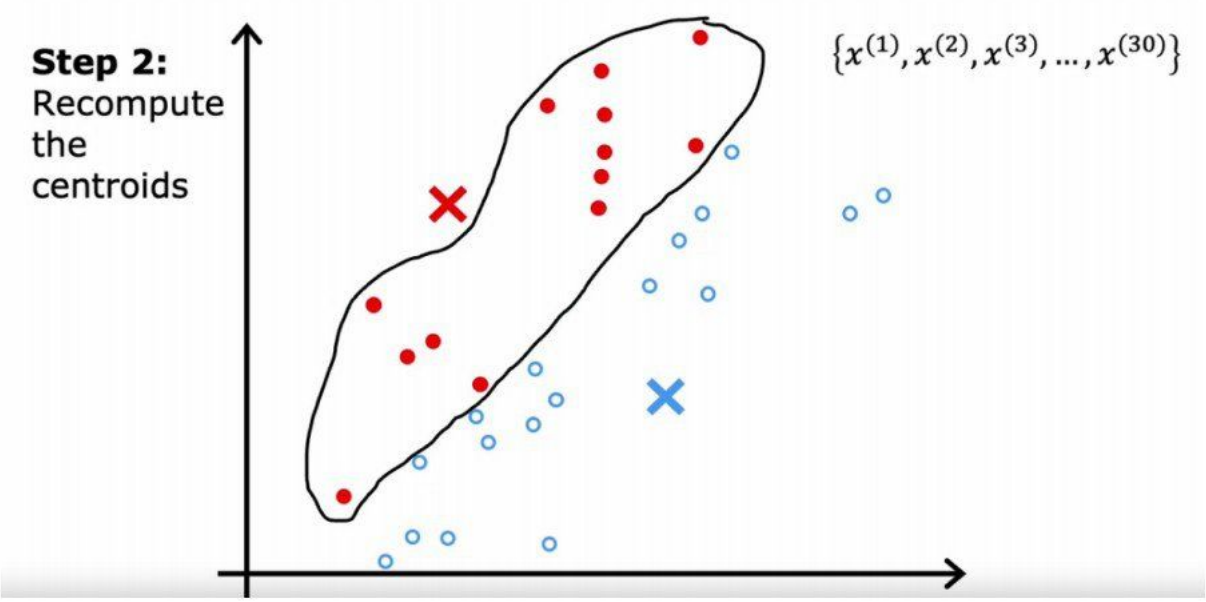
What is Clustering?

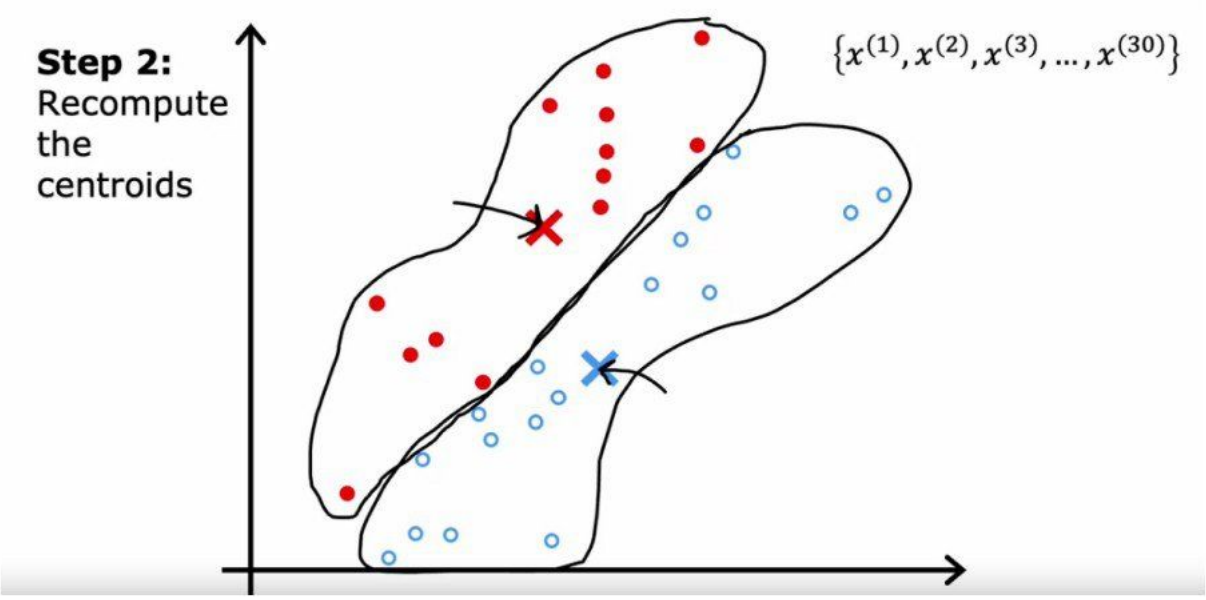
- Clustering is used to group unlabeled data
- There are various algorithms to perform the Clustering.

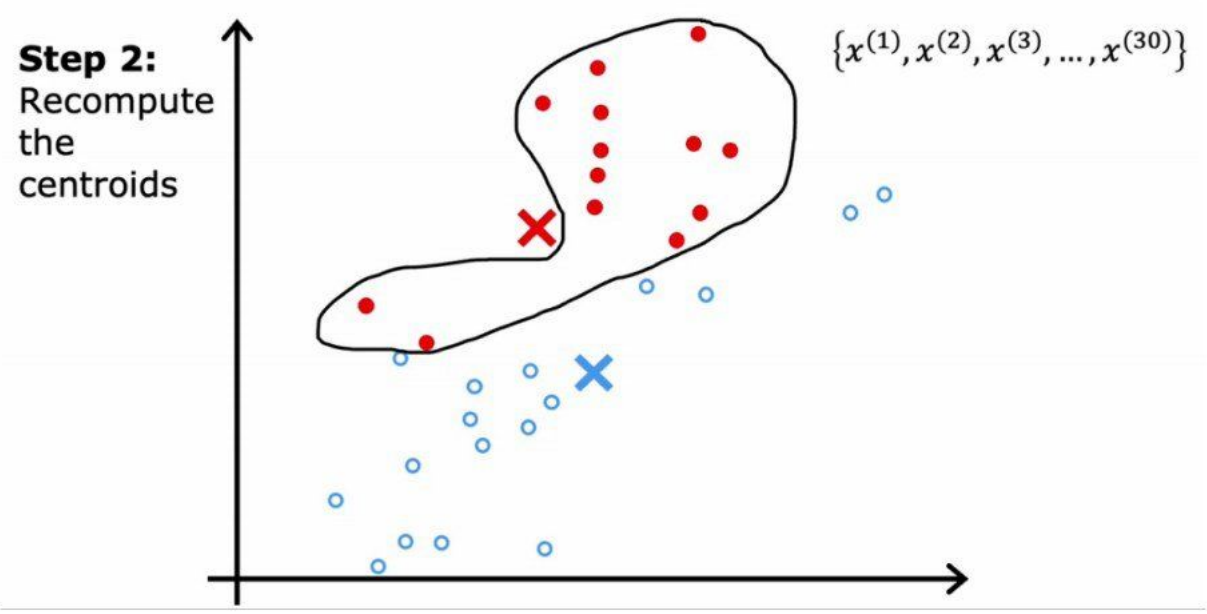


K- Means Intuition

- K is the decided number of clusters by us.
- Pick 2 random point
- These 2 point form center of cluster called **Cluster Centroids**
- After classifying like that it finds the average/mean
- Move all **Cluster Centroids** to that average/mean point
- Repeat the process for that **Cluster Centroids**
- After repeating it for a time period we finally get a **Cluster Centroids** where further repeating don't make any change.

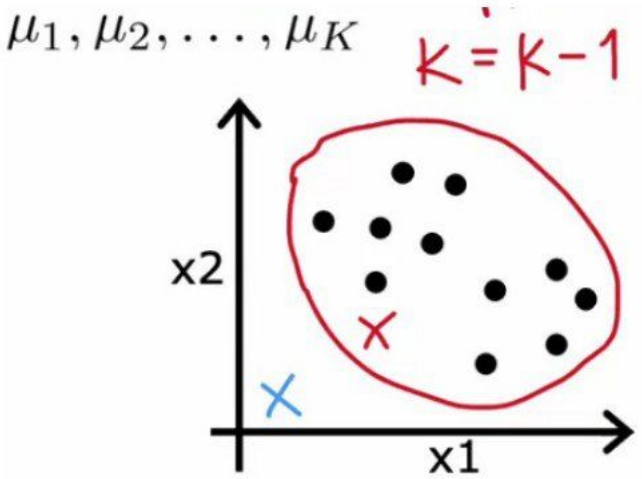

Step 2:
Recompute
the
centroids
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$


Step 2:
Recompute
the
centroids
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$

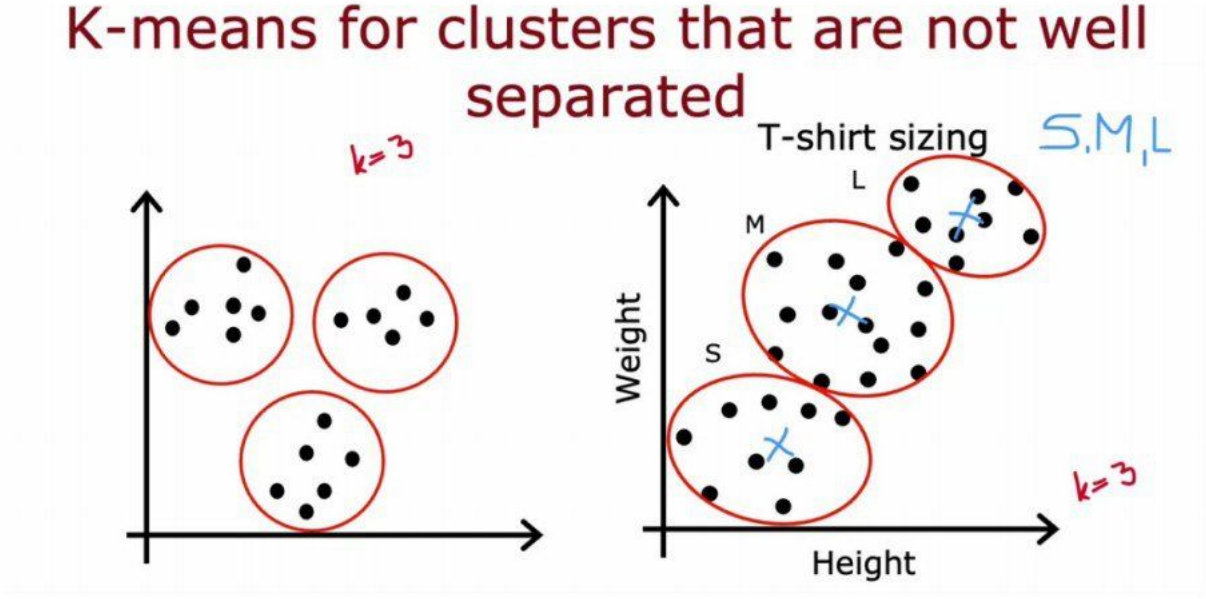


K-means Algorithm

- Find minimum value of Norm of each points with the Clustering Centroid
- Then find and mark the clusters
- Find average of cluster and move Clustering Centroid to that mean
- Repeat process
- If no sample assigned to a cluster. We can't move forward. Because we will be trying to find average of zero points.
- So we can eliminate that cluster, Or Reinitialize that K-means



K - Means can be also helpful for data that are not that much separated



Optimization objective of K-Means - Distortion Function

- Here also we try to minimize the a Cost Function called Distortion Function
- We calculate average of squared, difference of distance
- we want to minimize the cost function

K-means optimization objective

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

μ_k = cluster centroid k

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

$x^{(10)}$ $c^{(10)}$ $\mu_c^{(10)}$

Cost function

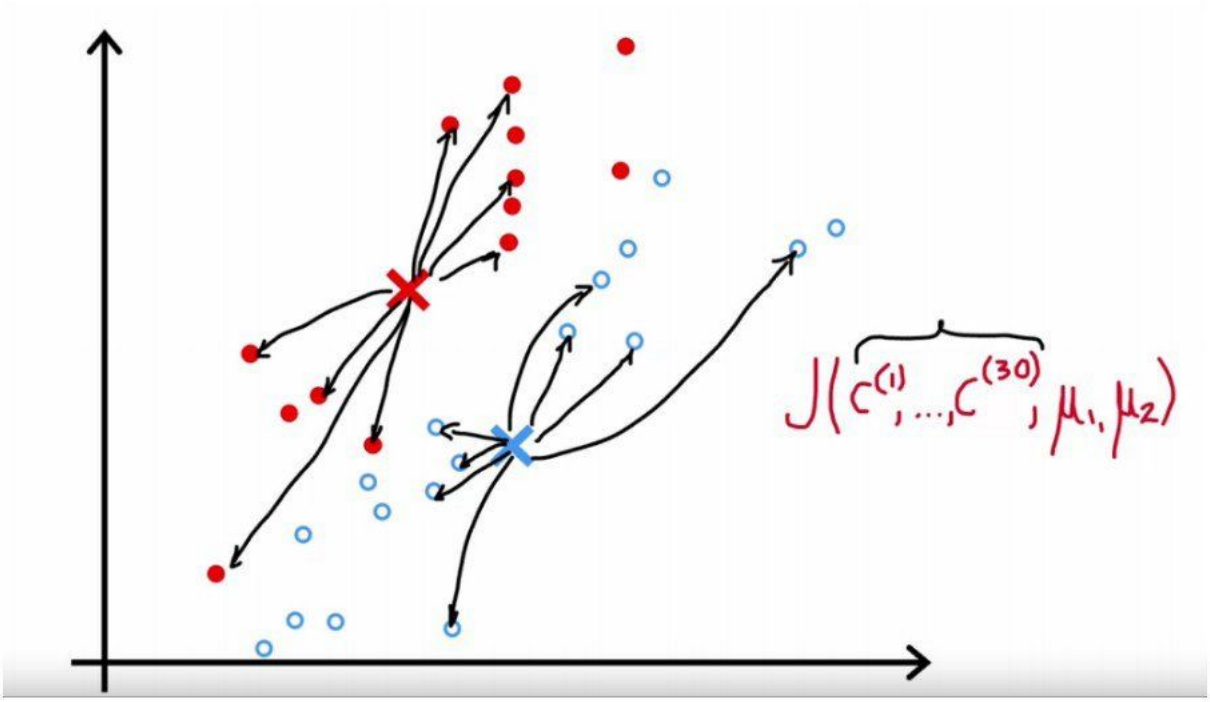
$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, .., \mu_K) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

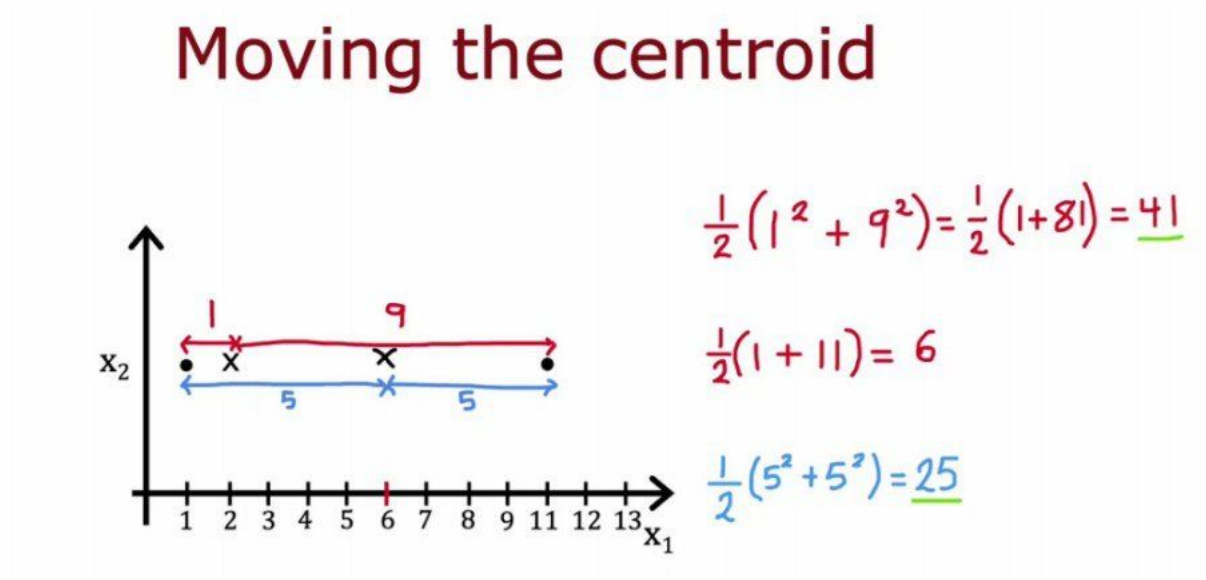
$$\min_{\substack{c^{(1)}, \ldots, c^{(m)} \\ \mu_1, \ldots, \mu_K}} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, .., \mu_K)$$

Distortion

Since we are moving cluster centroid to mean and calculating cost function really shows a decrease than previous one. So it is sure that distortion function, cost function goes down and

goes to convergence. So no need to run the K-means if distortion change is less than a small threshold. It means it reached convergence.

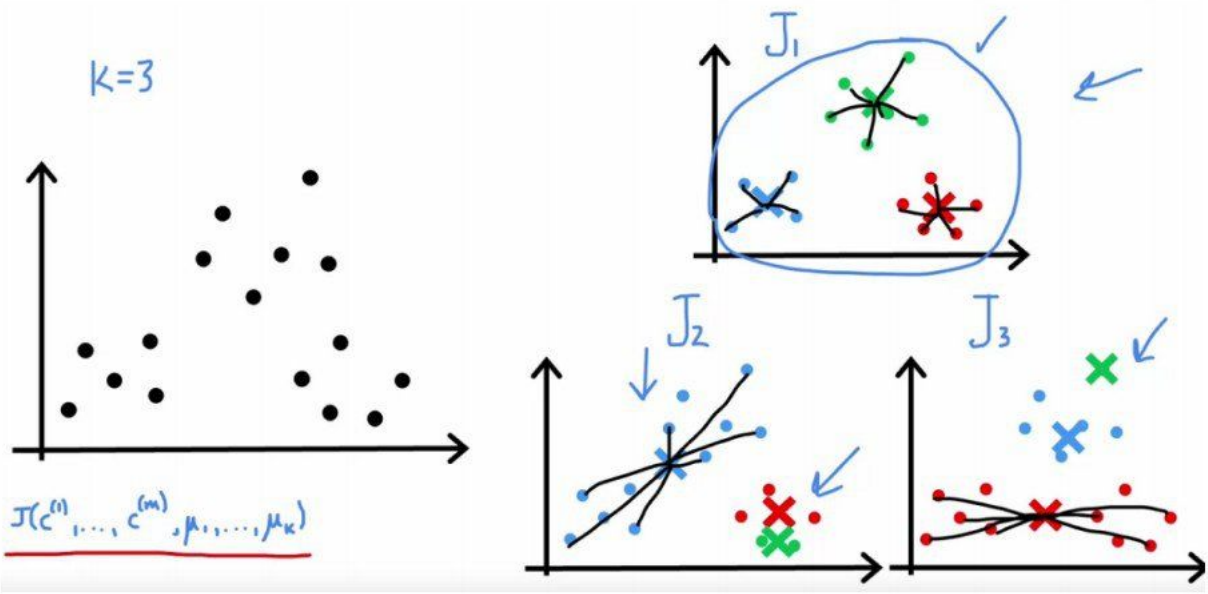


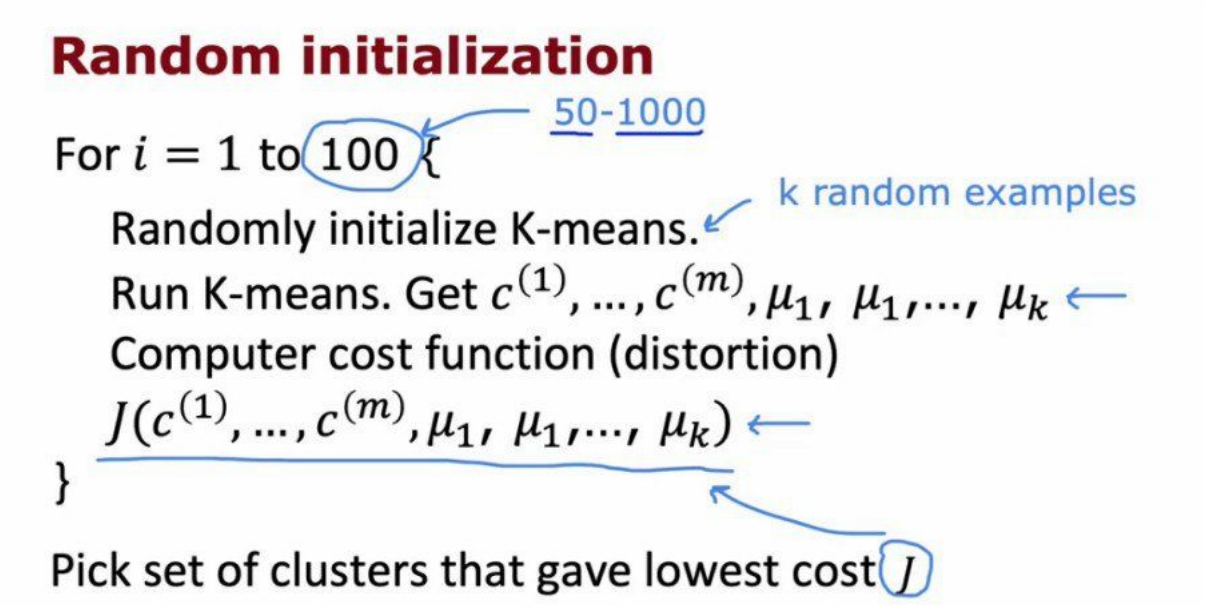


Initializing K-means

- First we need to pic a random points as cluster centroids
- Choose number of Cluster Centroid < Number of sample, K < m
- So pick K examples from the data

- Different Initialization give different result, we need to pic the one with Min Cost Function (Distortion)





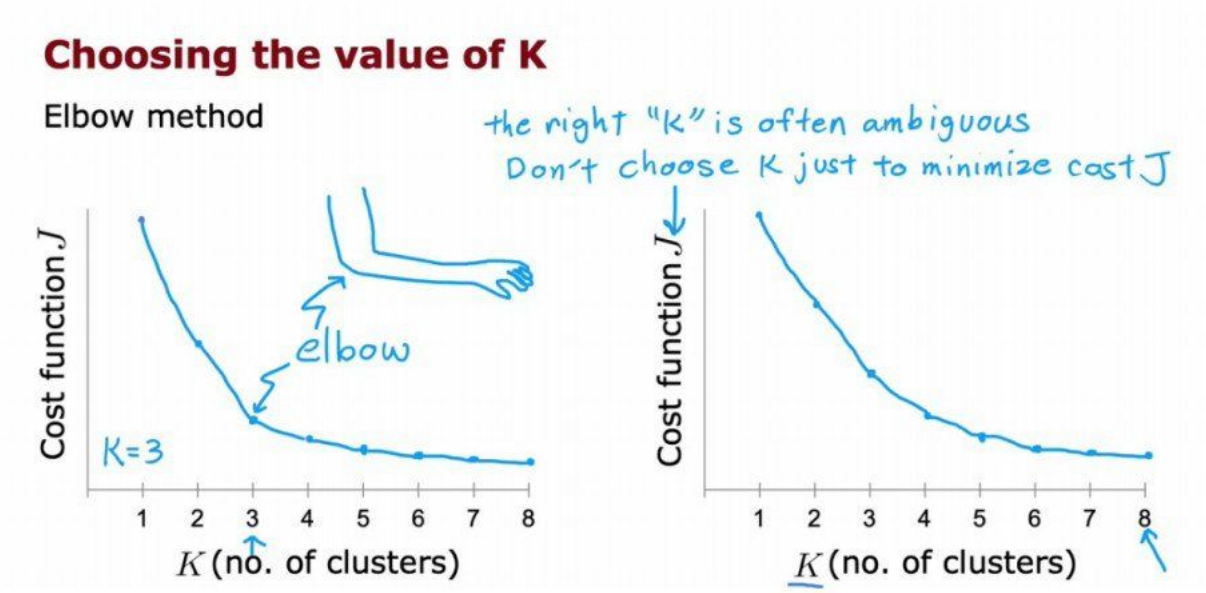
Choosing the Number of Clusters

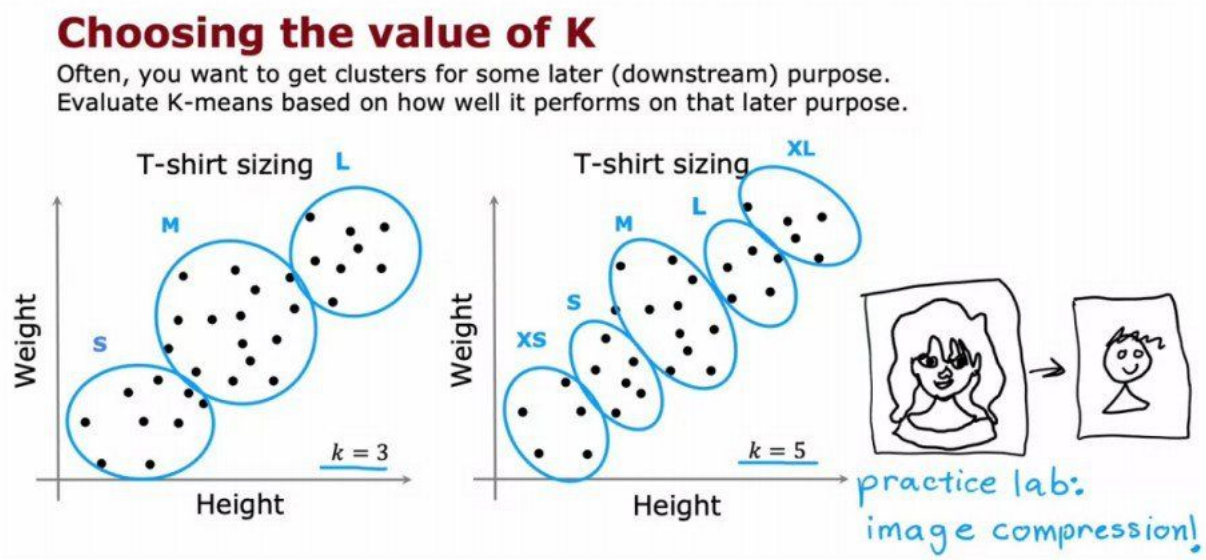
- Number of Clusters really ambiguous
- Our ultimate aim is not to pick really small Cost Function. In that case we can directly use large K to solve issue
- We really want to pick K that really make sense
- We can decide manually what k to pick based on Image Compression we want.

Types of method to choose value of K

Elbow Method

1. We plot the Cost Function for different K
2. It will look like Elbow

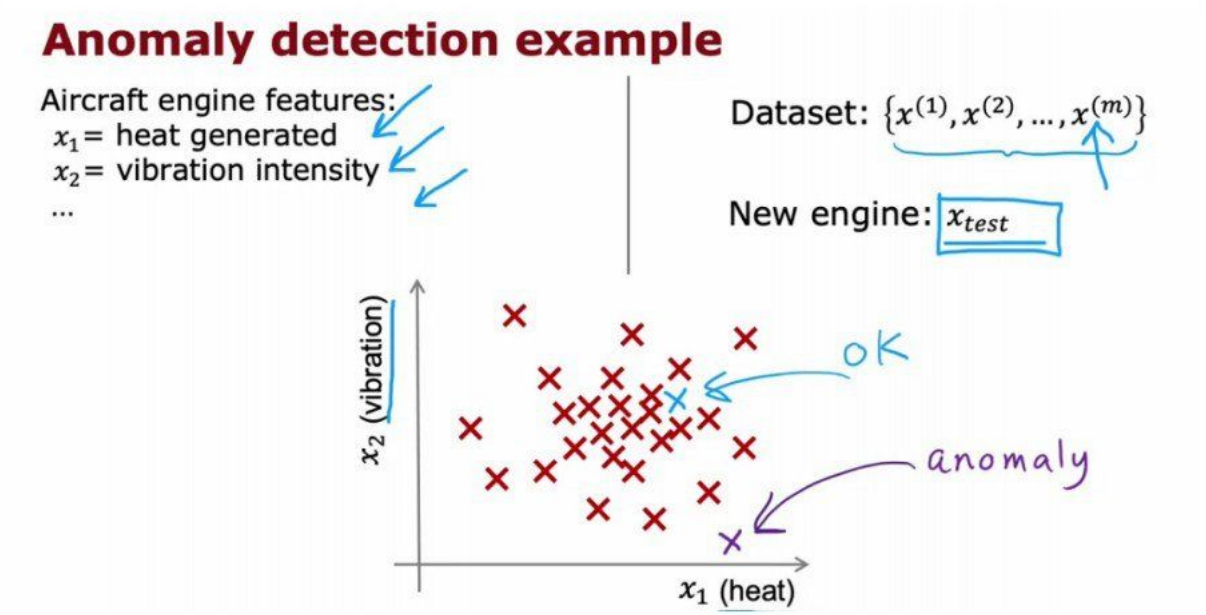


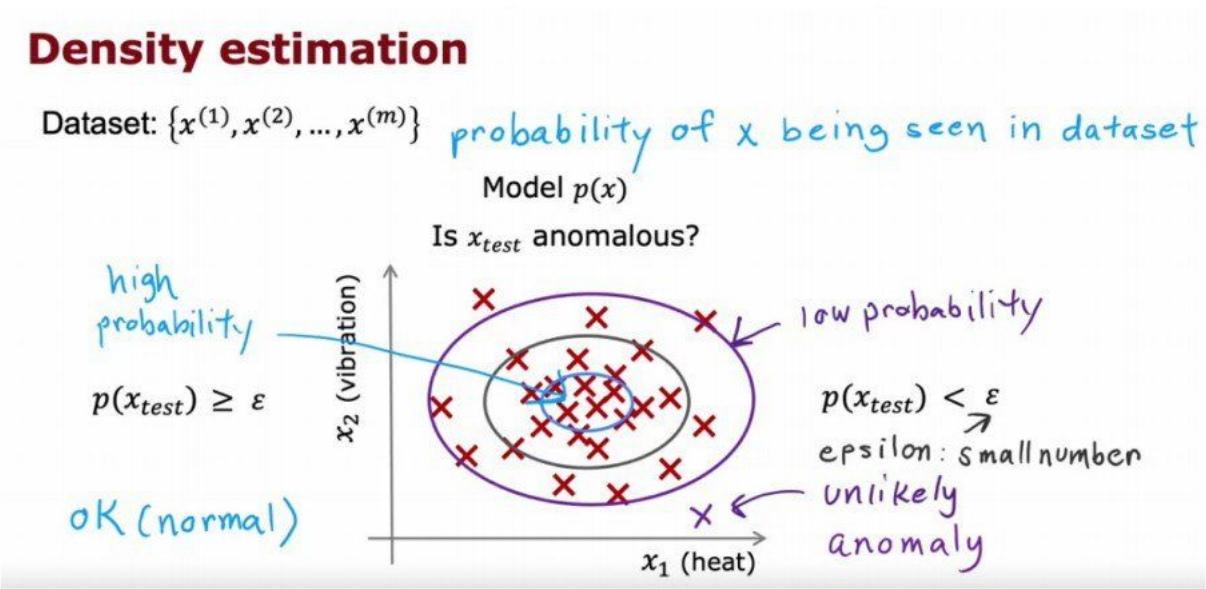


Anomaly Detection

- It is basically finding unusual things in data.
- We can perform Anomaly Detection using **Density Estimation**

- We can find the probability of the testing data and if it is less than the threshold epsilon, It is unusual, If it is greater than the epsilon it is normal





Fraud Detection and in Manufacturing Checking

Anomaly detection example

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

Fraud detection:

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

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

- It is also called bell shaped curve
- When width of sigma (standard deviation) increases then the values is spread out
- When width of sigma decreases then the values are close together

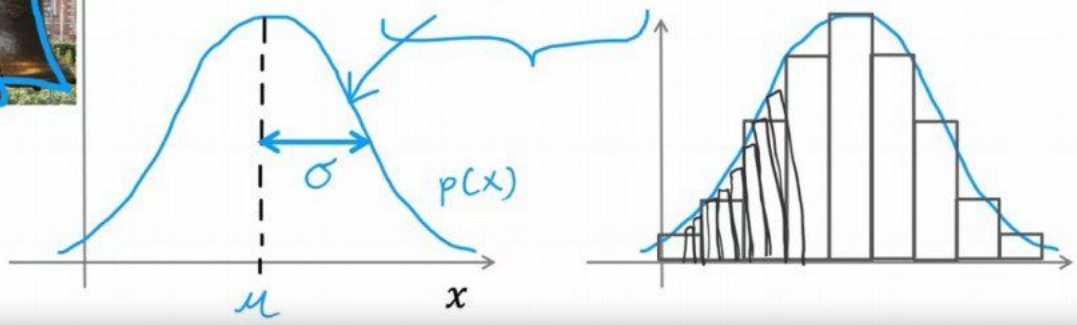
Gaussian (Normal) distribution

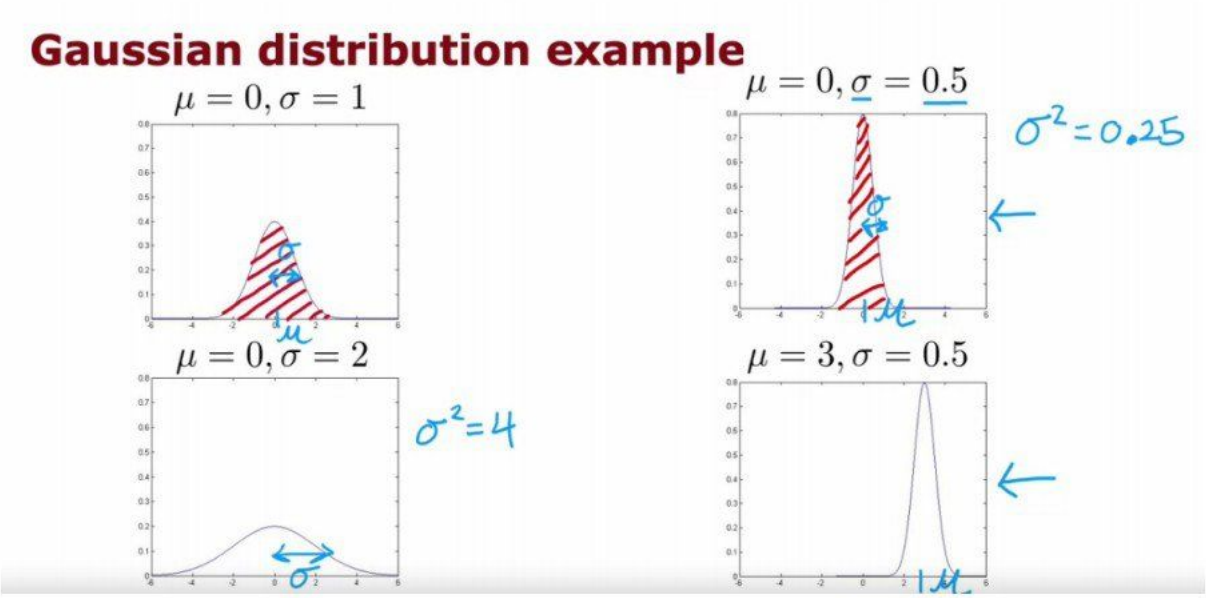
σ standard deviation
σ^2 variance

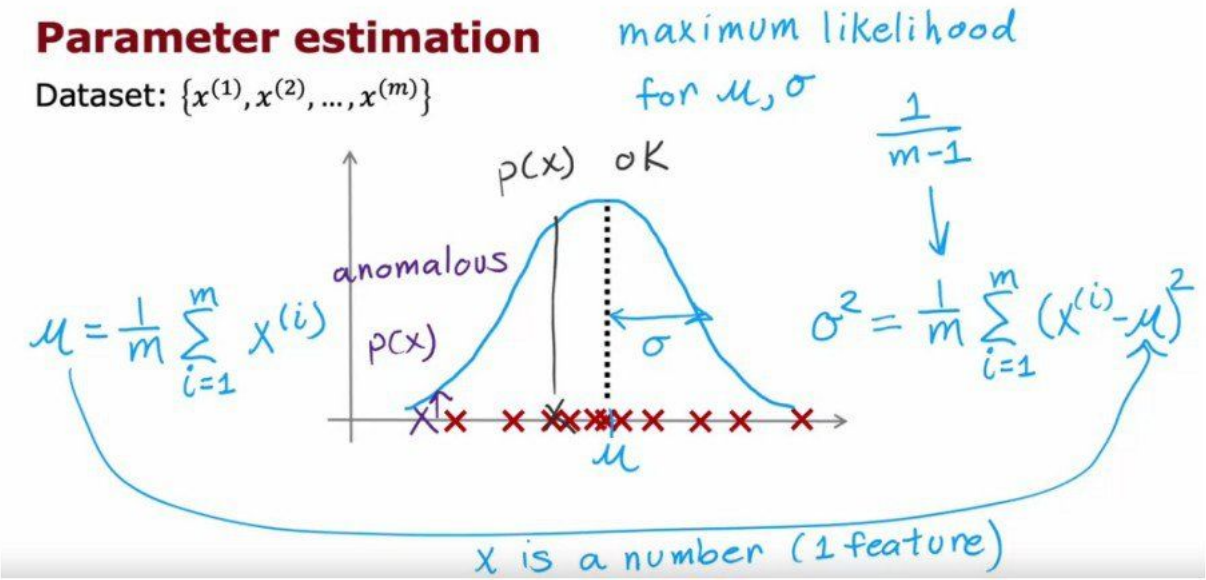
Say x is a number.
Probability of x is determined by a Gaussian with mean μ, variance σ^2.

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$\pi = 3.14$



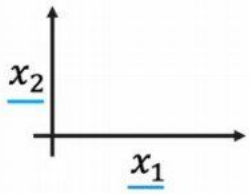




Anomaly Detection Algorithm

Density estimation

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



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

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

$$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

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

Anomaly detection algorithm

1. Choose n features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

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

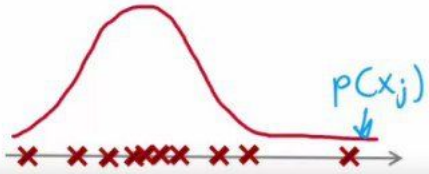
Vectorized formula

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

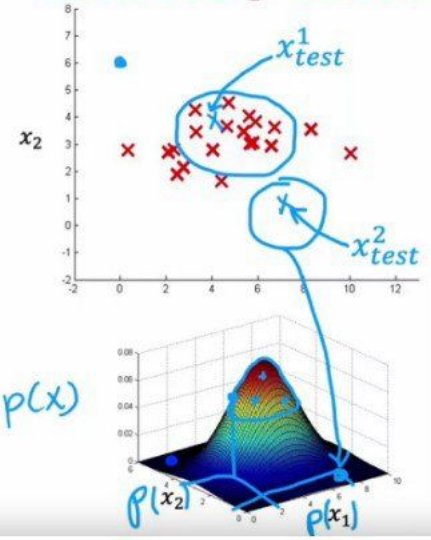
3. Given new example x, compute $p(x)$:

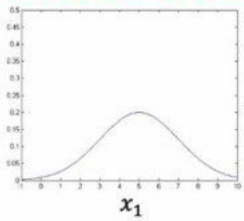
$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2})$$

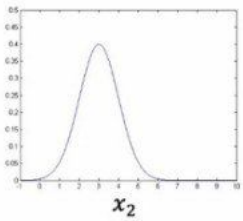
Anomaly if $p(x) < \varepsilon$



Anomaly detection example







$\mu_1 = 5,\ \sigma_1 = 2$ $\quad$ $\mu_2 = 3,\ \sigma_2 = 1$

$p(x_1; \mu_1, \sigma_1^2)$ $\quad$ $p(x_2; \mu_2, \sigma_2^2)$

$\varepsilon = 0.02$

$p\left(x_{test}^{(1)}\right) = 0.0426 \longrightarrow$ "ok"

$p\left(x_{test}^{(2)}\right) = 0.0021 \longrightarrow$ anomaly

Developing and evaluating an anomaly detection system

- Even if we a labelled data by assumption, anomaly can be applied to find the real anomaly out of wrongly labelled data

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

$y=0$ for all training examples

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

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

include a few anomalous examples $y=1$

mostly normal exan

Aircraft engines monitoring example

10000 good (normal) engines
20 flawed engines (anomalous)

Training set: 6000 good engines ($y = 0$)

CV: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Alternative: No test set

Training set: 6000 good engines ($y = 0$)

CV: 4000 good engines ($y = 0$), 20 anomalous ($y = 1$)

Algorithm evaluation

Fit model $p(x)$ on training set $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

On a cross validation/test example x, predict

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

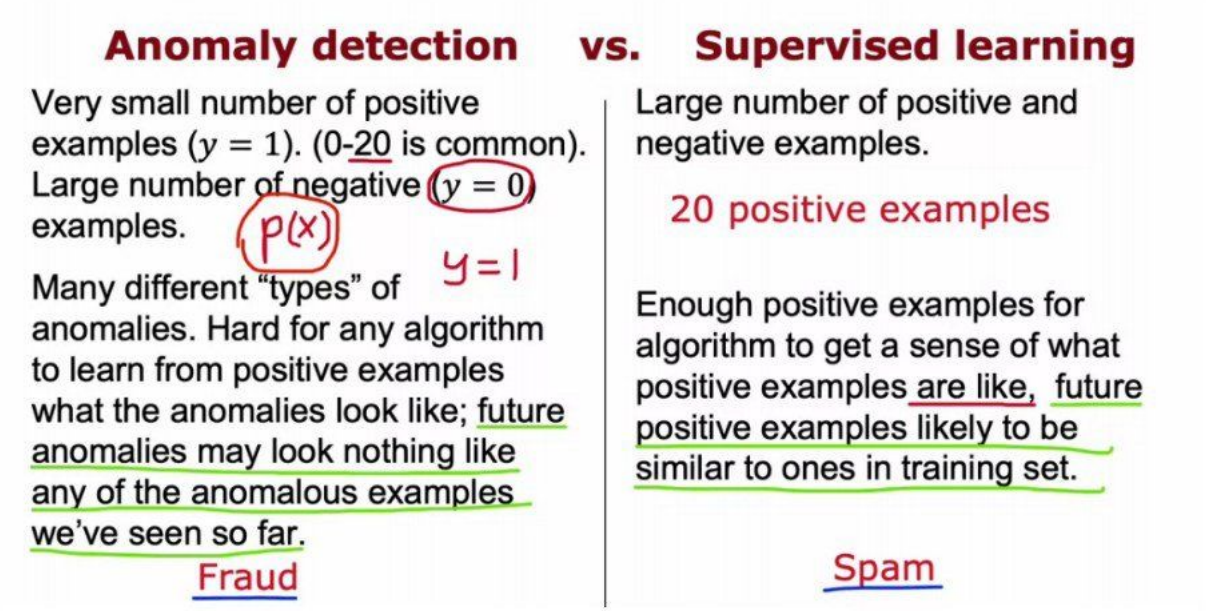
Possible evaluation metrics:

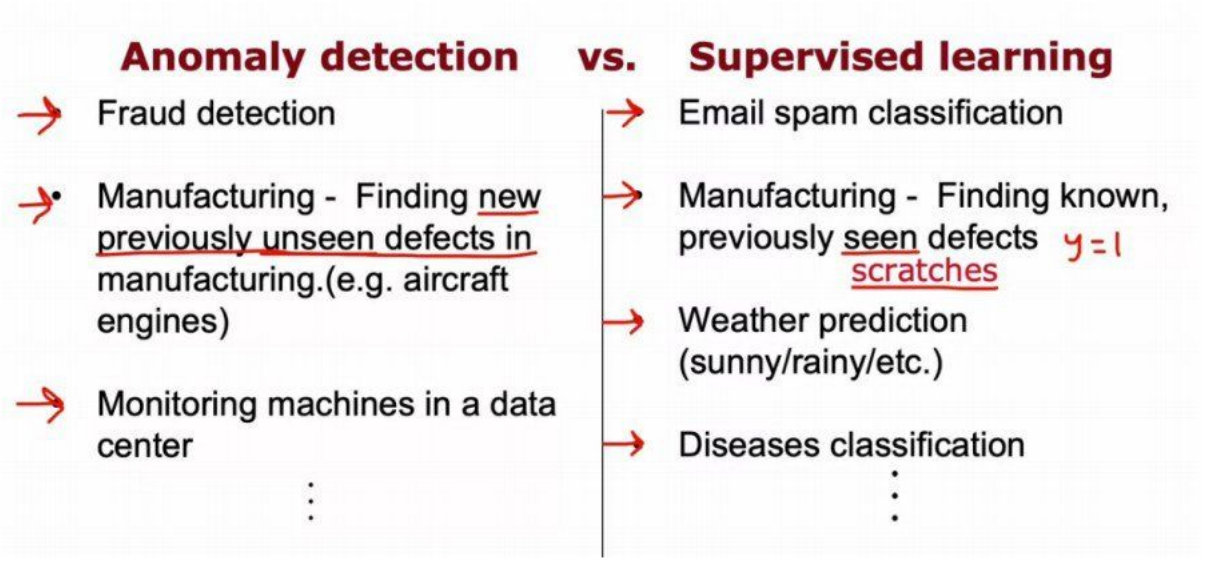
- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1-score

Use cross validation set to choose parameter ε

Anomaly detection vs. supervised learning

Anomaly is flexible than SL, because in SL we are learning from the data that is available. If something happened out of the box, SL cannot catch that but Anomaly can.





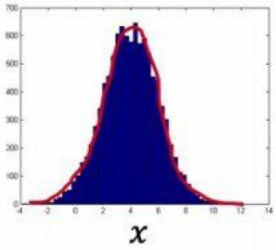
Choosing What Features to Use

- Anomaly like features with Gaussian Distribution
- To make features gaussian we can make use of log and other polynomial transformation.

```
# Feature transformations to make it gaussian
plt.hist(np.log(x+1), bins=50)
plt.hist(x**0.5, bins=50)
plt.hist(x**0.25, bins=50)
plt.hist(x**2, bins=50)

# In code bins is changed to make histogram less box type.
# Width of histograms decreases as bins increases.
```

Non-gaussian features



x

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

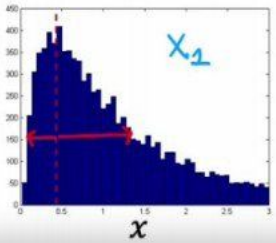
`plt.hist(x)`

$x_1 \leftarrow \log(x_1)$

$x_2 \leftarrow \log(x_2 + 1)$ log $(x_2 + c)$

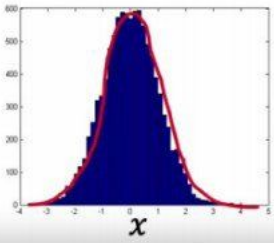
$x_3 \leftarrow \sqrt{x_3} = x_3^{1/2}$

$x_4 \leftarrow x_4^{1/3}$



x

`np.log(x)`

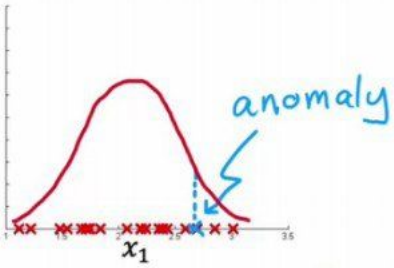


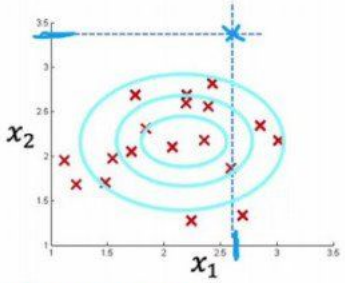
x

Error analysis for anomaly detection

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

Most common problem:
$p(x)$ is comparable for normal and anomalous examples.
($p(x)$ is large for both)





x_1 num transactions x_2 typing speed

Monitoring computers in a data center

Choose features that might take on unusually large or small values in the event of an anomaly.

x_1 = memory use of computer
x_2 = number of disk accesses/sec
x_3 = CPU load (high / low)
x_4 = network traffic

← not unusual

$$x_5 = \frac{\text{CPU load}}{\text{network traffic}} \qquad x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$

Deciding feature choice based on $p(x)$
Large for normal examples;
Becomes small for anomaly in the cross validation set

Making Recommendations

- Amazon/Netflix every company now uses recommender systems
- Here we try to find the movies they have not rated, then we predict the rating for that movie and recommend it to the user if the predicted rating is 5 star.

Predicting movie ratings

User rates movies using ~~one~~ zero 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	?

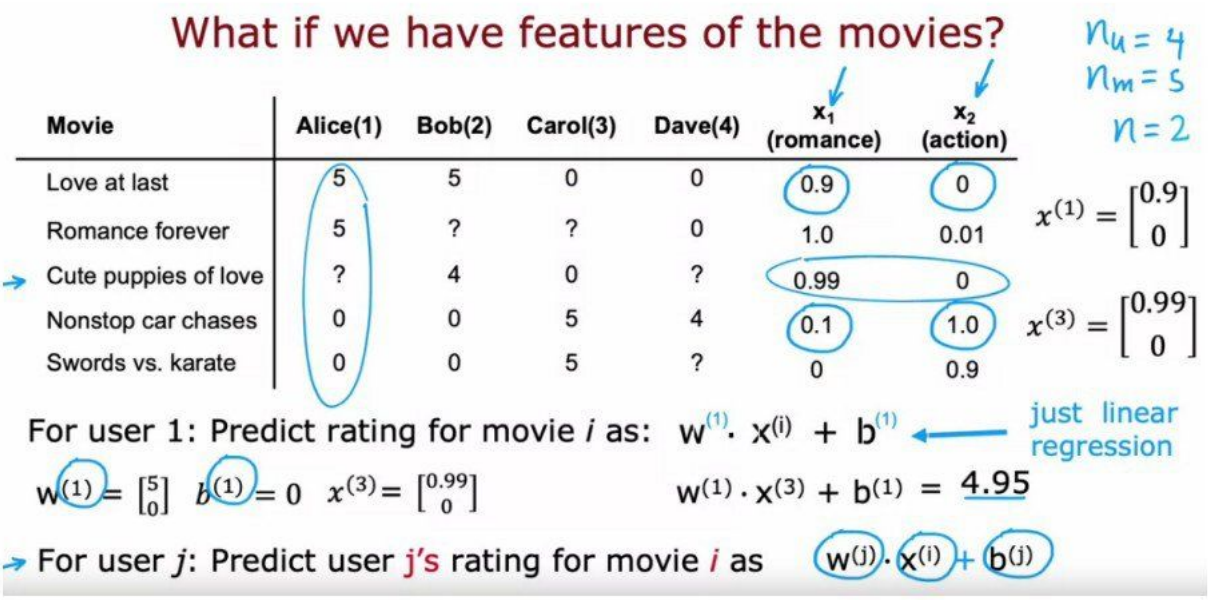
n_u = no. of users
n_m = no. of movies
$r(i,j)=1$ if user j has rated movie i
$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j)=1$)

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

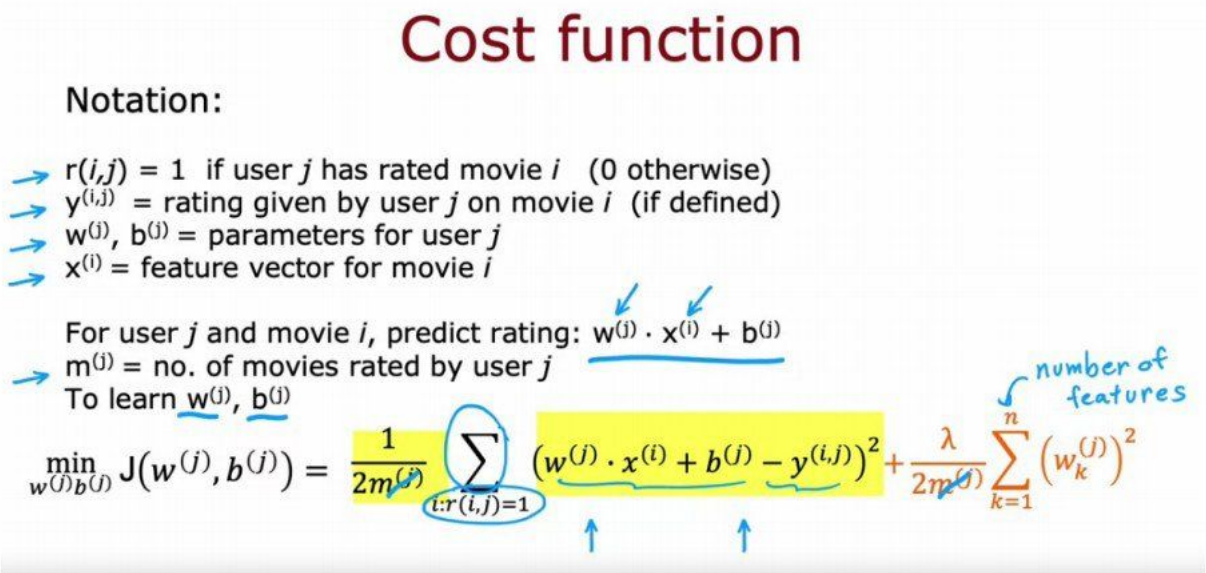
Using per-item features

- It is more like Linear Regression. But here we usually add Regularization along with that.

- Here also we calculate the cost function and try to minimize the Cost Function
- Suppose that we have features of movies, for that vector we need to find the w and be for a user to predict ratings of any movies
- So we need to find the parameters w and b



- To find the W and b, we find and try to minimize the cost function
- For cost function we use MSE plus the L2 regularization



- But we need to learn the parameters for all users
- So the cost function need a little modification.

Cost function

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

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

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

$$J\begin{pmatrix} w^{(1)}, \ldots, w^{(n_u)} \\ b^{(1)}, \ldots, b^{(n_u)} \end{pmatrix} = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \underbrace{\left(w^{(j)} \cdot x^{(i)} + b^{(j)}\right)}_{f(x)} - y^{(i,j)}\Big)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

- What to do if we don't have the features data
- Suppose we already learned the parameters W and b, then comparing with the ratings of movie by users we can find the features.
- Now by using that features we can predict the ratings of un rated movies by users

Problem motivation

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	? ←	?	?

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

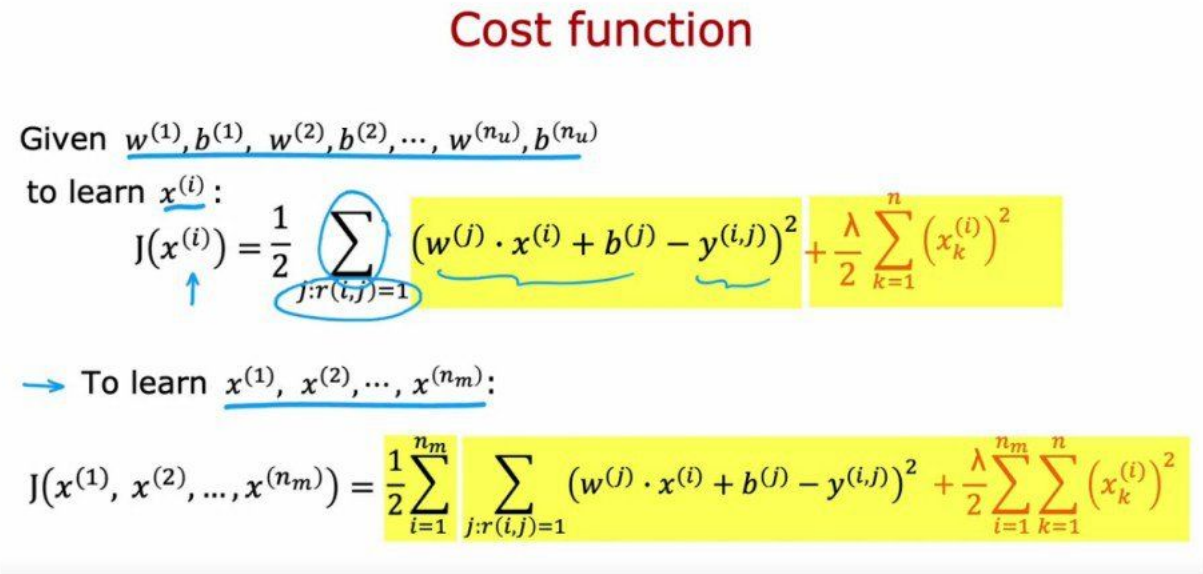
$b^{(1)} = 0$, $b^{(2)} = 0$, $b^{(3)} = 0$, $b^{(4)} = 0$ ←

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

$w^{(1)} \cdot x^{(1)} \approx 5$
$w^{(2)} \cdot x^{(1)} \approx 5$
$w^{(3)} \cdot x^{(1)} \approx 0$
$w^{(4)} \cdot x^{(1)} \approx 0$

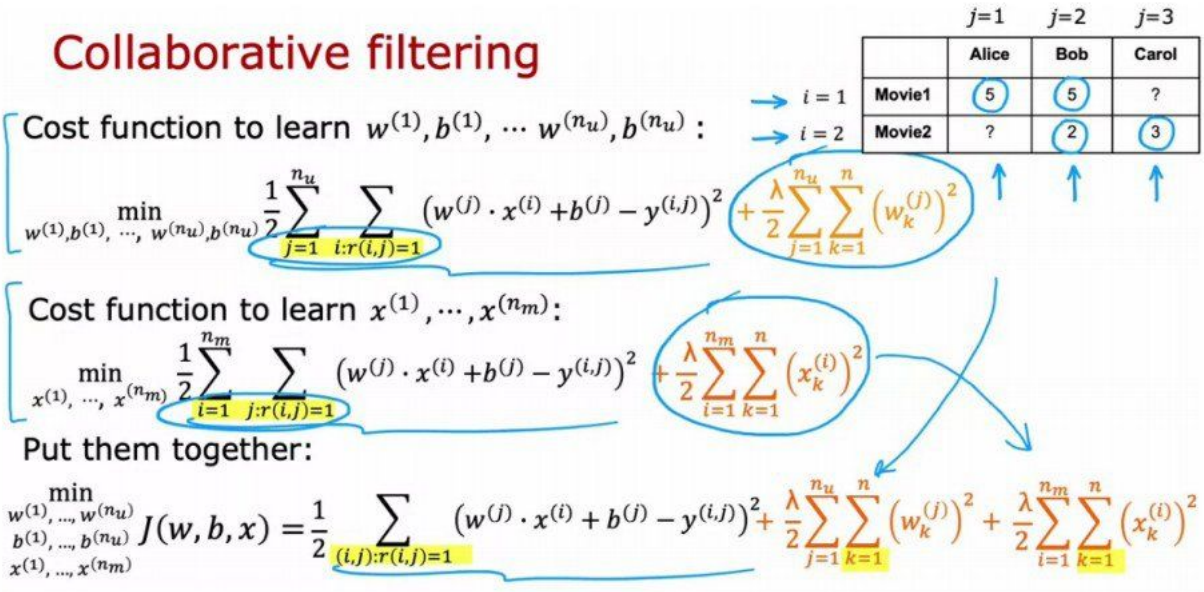
$\rightarrow \quad x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

- To learn the features, the cost function is given below
- In order to generalize the cost function, we try to learn for all users

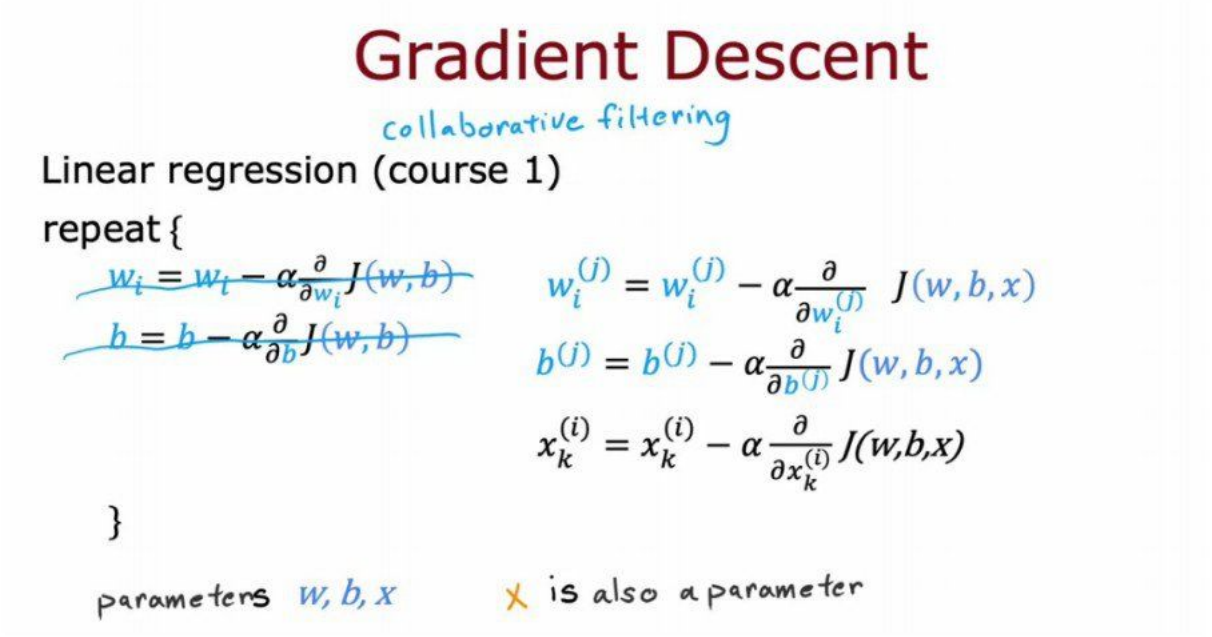


Collaborative filtering algorithm

Collaborative filtering refers to the sense that because multiple users have rated the same movie collaboratively, given you a sense of what this movie maybe like, that allows you to guess what are appropriate features for that movie, and this in turn allows you to predict how other users that haven't yet rated that same movie may decide to rate it in the future.



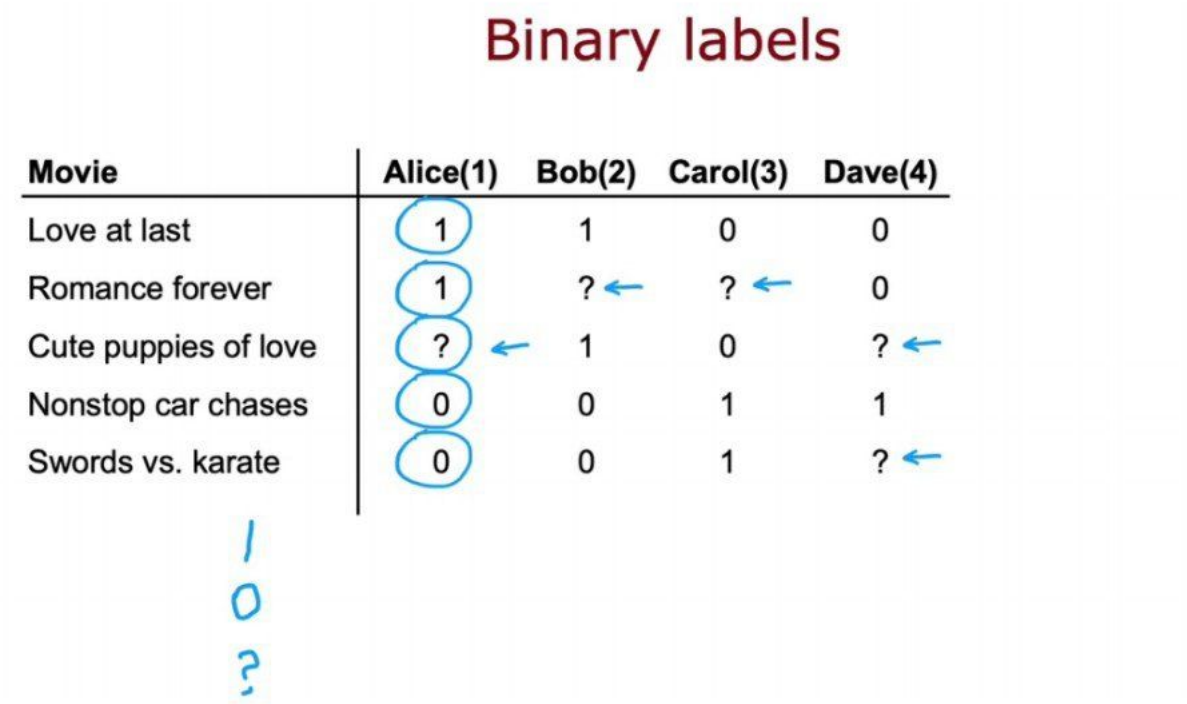
- Now the cost function is a function of W, b and x
- So while applying gradient descent we need to use it in x also.



Binary labels: favs, likes and clicks

So far, our problem formulation has used movie ratings from 1- 5 stars or from 0- 5 stars. A very common use case of recommended systems is when you have binary labels such as that the user favors, or like, or interact with an item. A generalization of the model that you've seen so far to binary labels.

- So we can convert linear regression to logistic regression model. So MSE cost function changes to Log Loss



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:

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

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

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

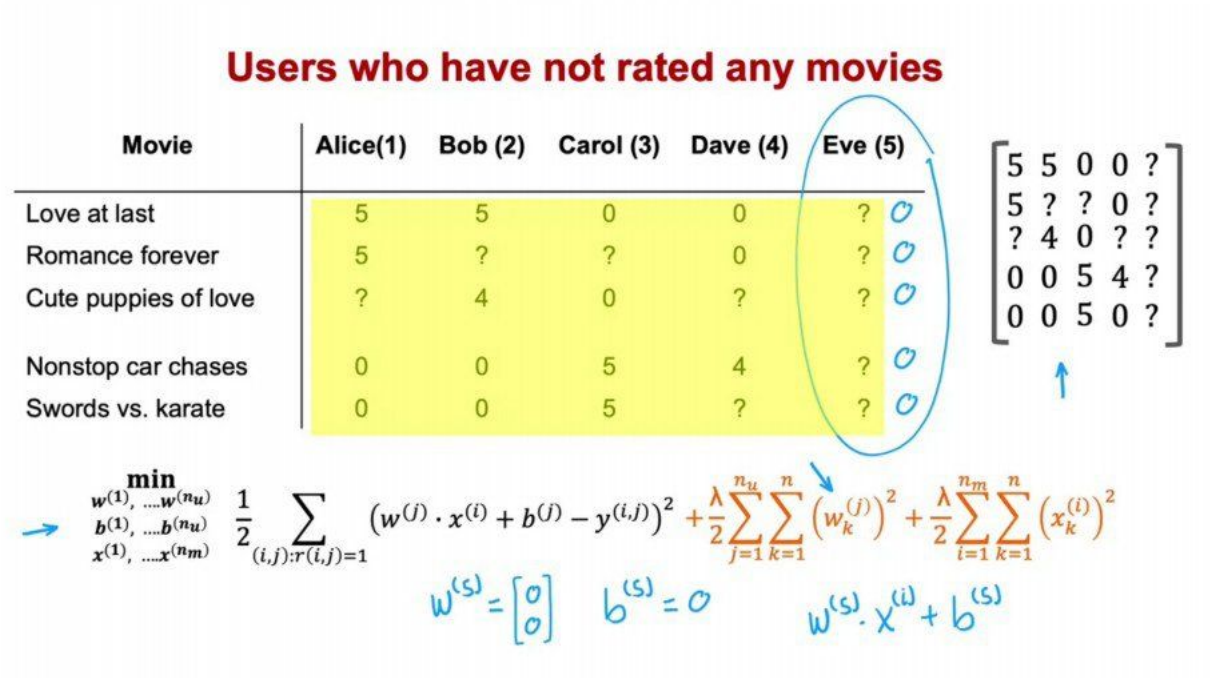
Loss for single example

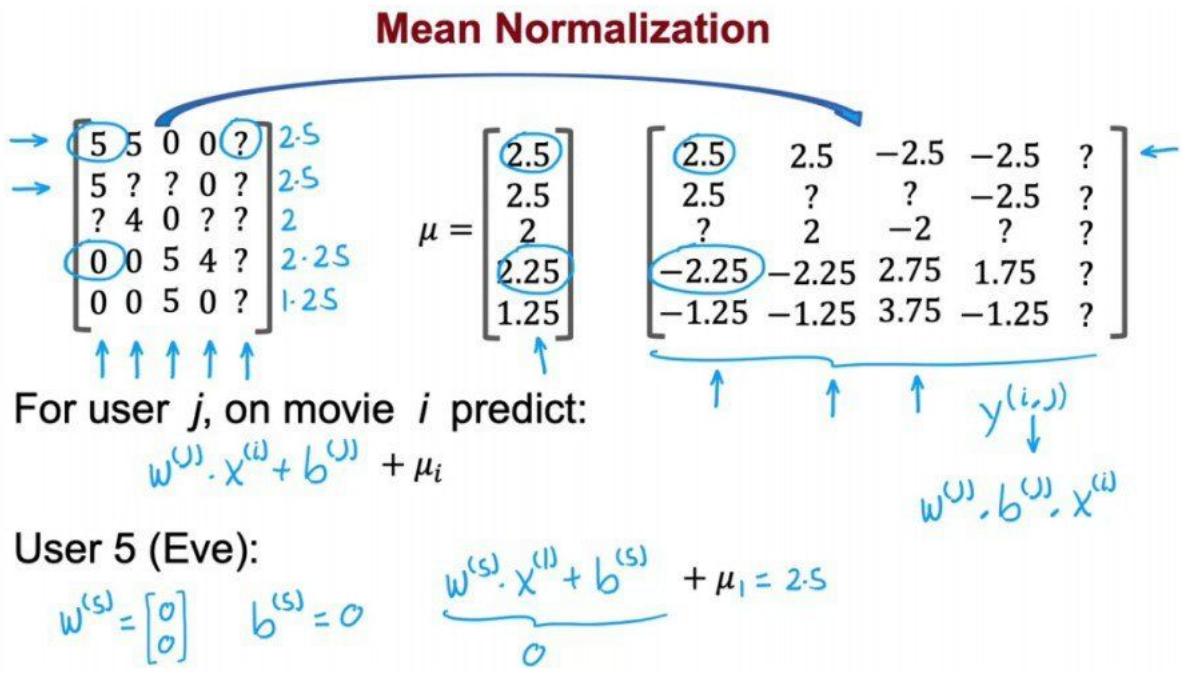
$$J(w,b,x) = \sum_{(i,j):r(i,j)=1} L\left(\underbrace{f_{(w,b,x)}(x)}_{g(w^{(j)} \cdot x^{(i)} + b^{(j)})}, y^{(i,j)}\right)$$

cost for all examples

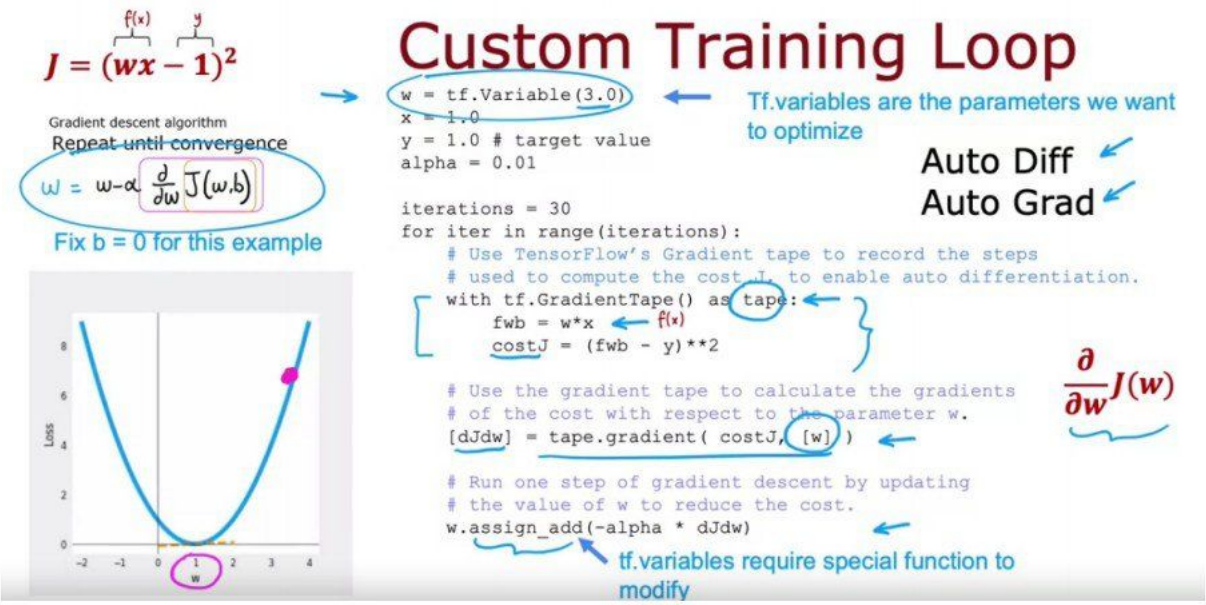
Mean Normalization

- Subtract the mean (average rating) from the rows, which is each user rating.
- So the movies not rated by user is replaced by average rating. This is Normalization.





TensorFlow Implementation of Collaborative Filtering



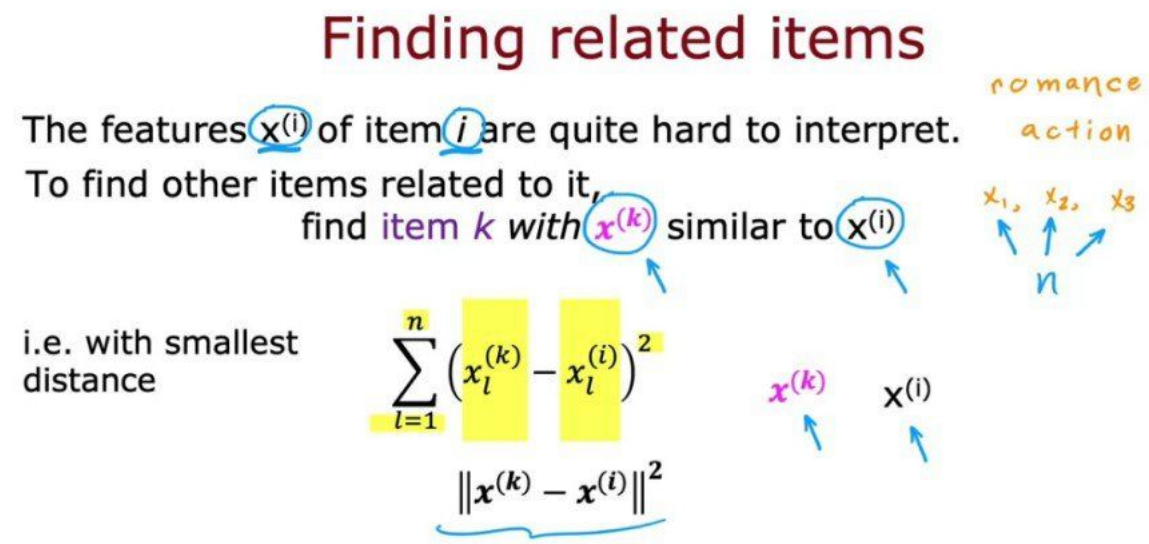

```
# Instantiate an optimizer.
optimizer = keras.optimizers.Adam (learning_rate=le-1)

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 = cofiCostFuncV (X, W, b, Ynorm, R, num_users, num movies, lambda)

    # 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]) )
```

Finding Related Items

- In all cases for recommending something, we need to find the relation between the items.
- Collaborative Filtering can find relation, using MSE between items. But has so many limitation.



Limitations of Collaborative Filtering

1. Cold start problem How to
   - **rank new items that few users have rated?**
2. 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 - Recommend items to you based on ratings of users who gave similar ratings as you.
- Content-based - Recommend items based on features of user and item to find good match.

Examples of user and item features

The features User and Movie can be clubbed together to form a vector.

User Features

- Age
- Gender

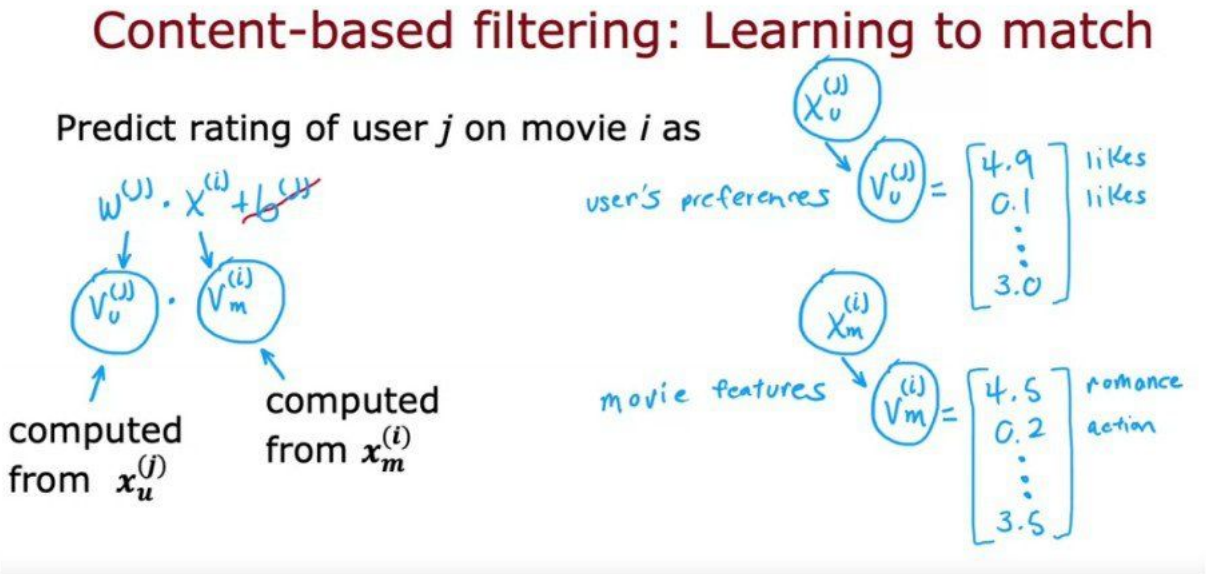
- Country
- Movie Watched
- Average rating per genre

Movie Feature

- Year
- Genre/Genres
- Reviews
- Average Rating

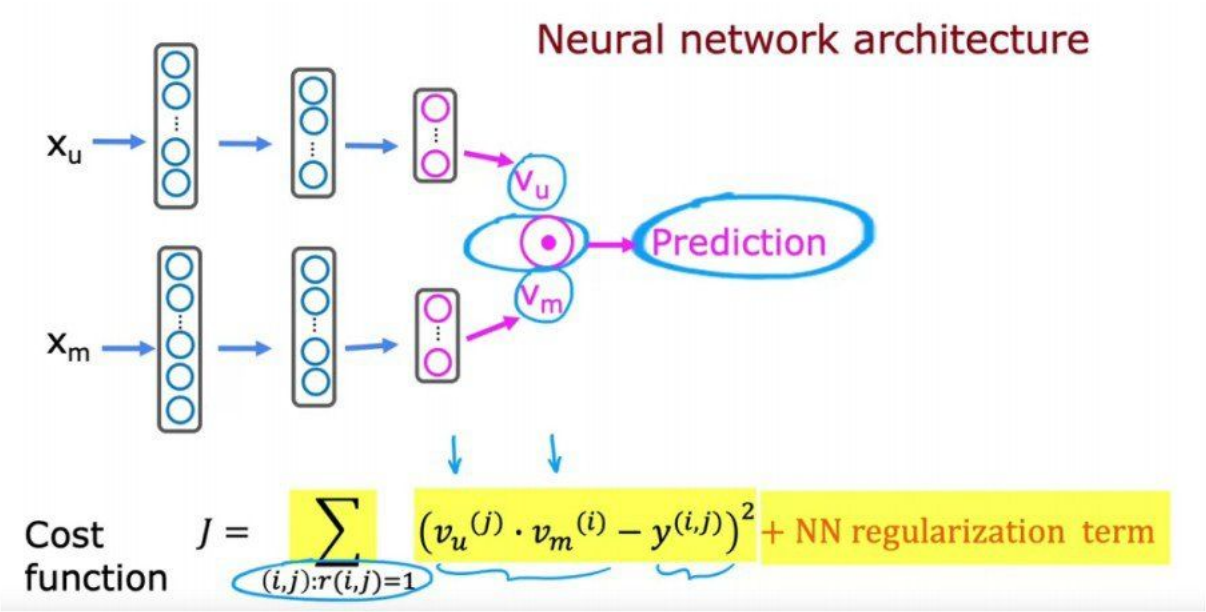
To predict the movie rating we can use a linear regression were w can be heavily depend on user feature vector and x can depend on movie feature vector.

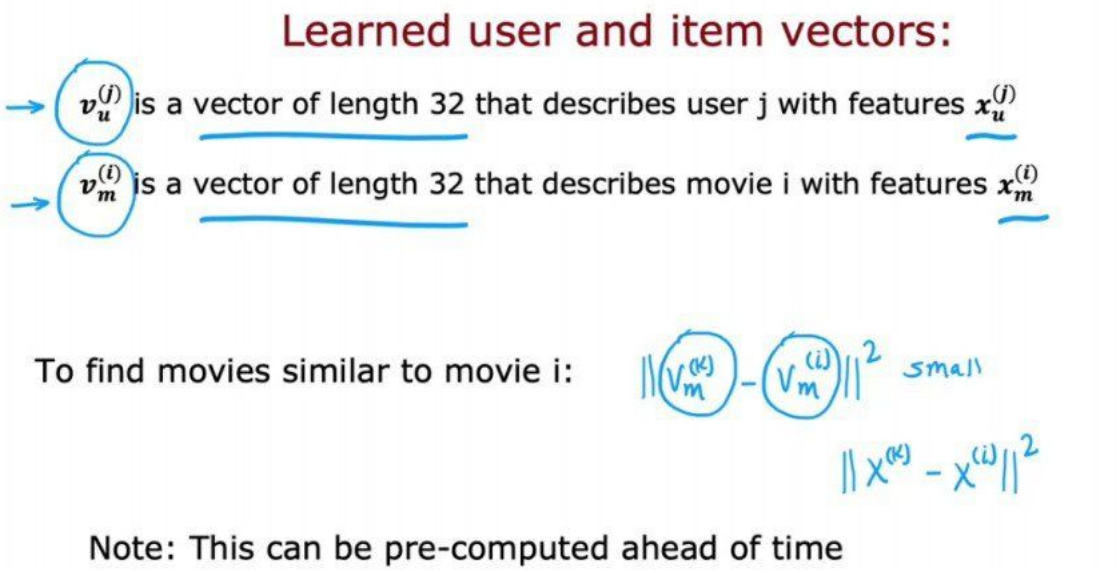
For this we need vectors for User and Movies



Deep learning for content-based filtering

- For User and Movie features we need to create vectors
- First NN will be User Network, Output Vu, which is a Vector of User
- Second NN will be Movie Network, Output Vm, which is a Vector of Movie
- Both output Vu and Vm have same dimension. Using that we create the cost function
- Since various NN can be linked together easily, we can take product of both user and movie vector.





Recommending from a large catalogue

We always have large set of items when it comes to Movies. Songs, Ads, Products. So having to run NN instance for millions of times whenever user log in to system is difficult.

So there are two steps

Retrieval

1. Generate large list of plausible (probable) item of candidate
    a. For last 10 movies watched by user, find 10 most similar movies
    b. For most viewed genre find top 10 movies

    c. Top 20 movies in the country

2. Combined Retrieved items to list, removing duplicates, items already watched, purchased etc

Ranking

1. Take list retrieved and rank them based on learned model
2. Display ranked items to user

Retrieving more items results in better performance, but slower recommendations

To analyze that run it offline and check if recommendation, that is p(y) is higher for increased retrieval.

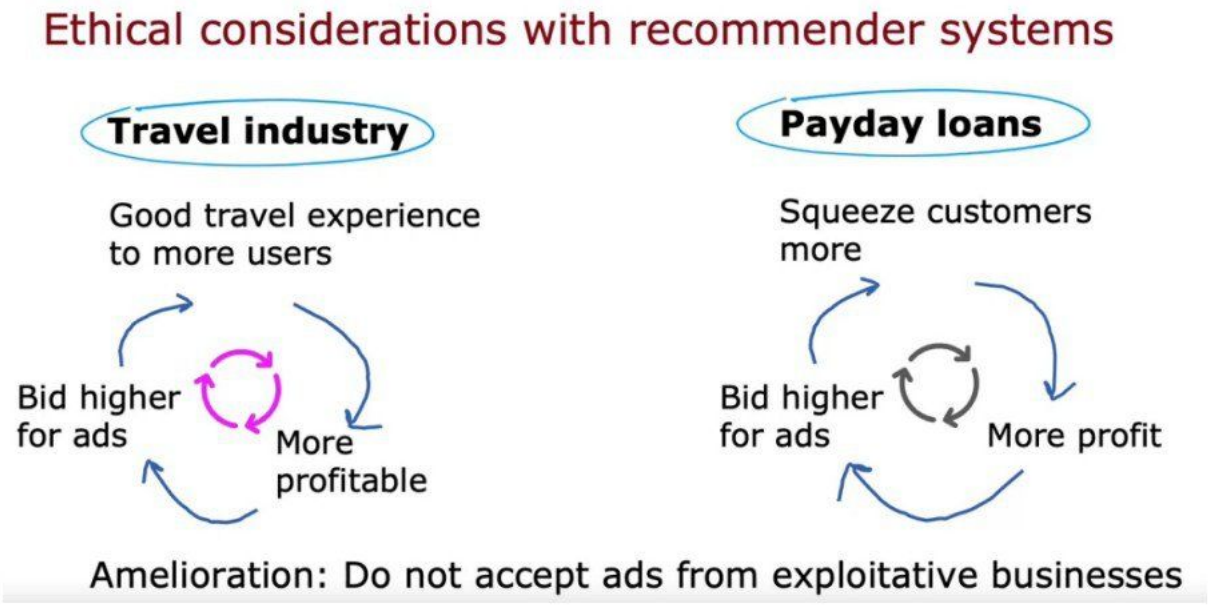
Ethical use of recommender systems

What is the goal of the recommender system?

- Movies most likely to be rated 5 stars by user
- Products most likely to be purchased

Illegal Things

- Ads most likely to be clicked on
- Products generating the largest profit
- Video leading to maximum watch time



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

First create the 2 NN of user and movie/item

```
user_NN = tf.keras.models. Sequential ([
                            tf.keras.layers.Dense (256, activation='relu'),
                            tf.keras.layers.Dense (128, activation=' relu'),
                            tf.keras.layers.Dense (32)
                          ])
item NN = tf.keras.models. Sequential ([
                            tf.keras.layers.Dense (256, activation= 'relu'),
                            tf.keras.layers.Dense (128, activation= 'relu'),
                            tf.keras.layers.Dense (32)
                          ])
```

- Now we need to combine both Vm and Vu
- So we take user_NN and item_NN as input layer of combined model
- Then take their product to make output
- Prepare the model using this NN
- Finally evaluate model by MSE cost function to train the model

```
# create the user input and point to the base network
input_user = tf.keras.layers. Input (shape=(num_user_features))
vu = user_NN (input_user)
vu = tf.linalg.12_normalize (vu, axis=1)

# create the item input and point to the base network
input_item = tf.keras.layers. Input (shape=(num_item_features))
vm = item_NN (input_item)
vm = tf.linalg.12_normalize (vm, axis=1)

# measure the similarity of the two vector outputs
output = tf.keras.layers. Dot (axes=1) ([vu, vm])

# specify the inputs and output of the model
model = Model ([input_user, input_item], output)

# Specify the cost function
cost_fn= tf.keras.losses. Mean SquaredError ()
```

What is Reinforcement Learning?

- Here we try to perform a action from it’s given state
- We can use the linear relation too, but it is not possible to get the dataset of the input x and output y
- So supervised learning won’t work here.
- Here we tell the model what to do. But not how to do
- We can make the mark the data as 1 if it performs well and -1 if it fails
- Algorithm automatically figure out what to do. We just need to say to the algorithm, if it is doing well or not using data input.

Applications

- To Fly a Helicopter
- Control Robots

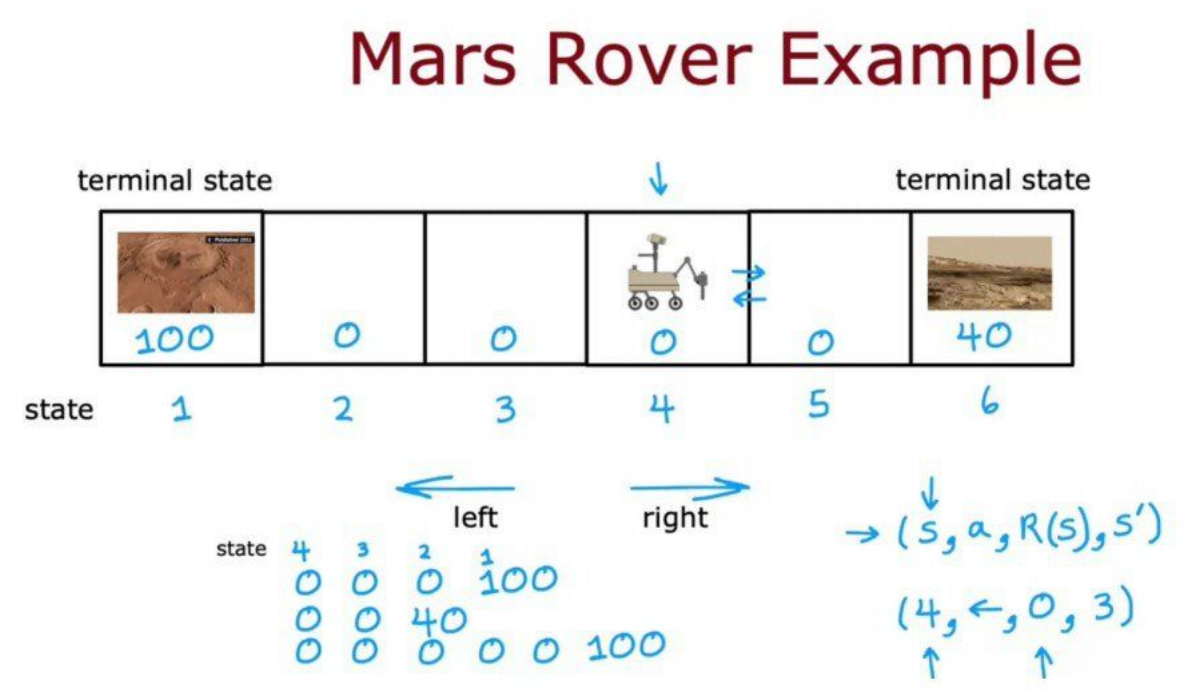
- Factory Optimization
- Stock Trading
- Play Video Games

Mars Rover example

For a robot in mars trying to find water/rock/surface. The position of rover or robot is State. The reward we provide to state decide the target for the rover. It can go to left or right. The final state after which nothing happens is called Terminal State. But it will learn from mistakes.

It will have 4 values

1. The current state it is having
2. The action it took
3. The reward it got for action
4. The new state



The Return in Reinforcement Learning

- We can connect Reward and Discount Factor (gamma) for reaching at final state.

- The reward value decreases as Length/Distance increases, since it takes more time to run the model.
- Discount Factor can be between 0 and 1
- **Discount Factor close to 1 - Then it is really Patient to run long for reward**
- **Discount Factor close to 0 - Then it is really Impatient to run long for reward. It need reward fast.**

Return

Return $= 0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100 = 0.729 \times 100 = 72.9$

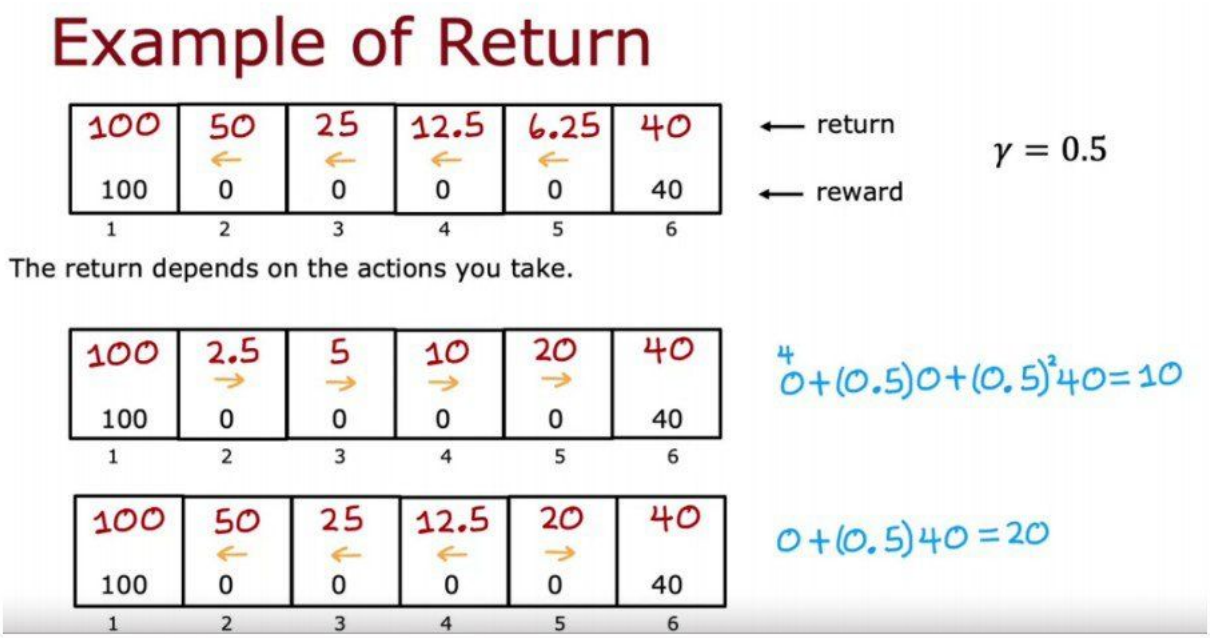
Return $= R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$ (until terminal state)

Discount Factor $\gamma = 0.9 \quad 0.99 \quad 0.999$

$\gamma = 0.5$

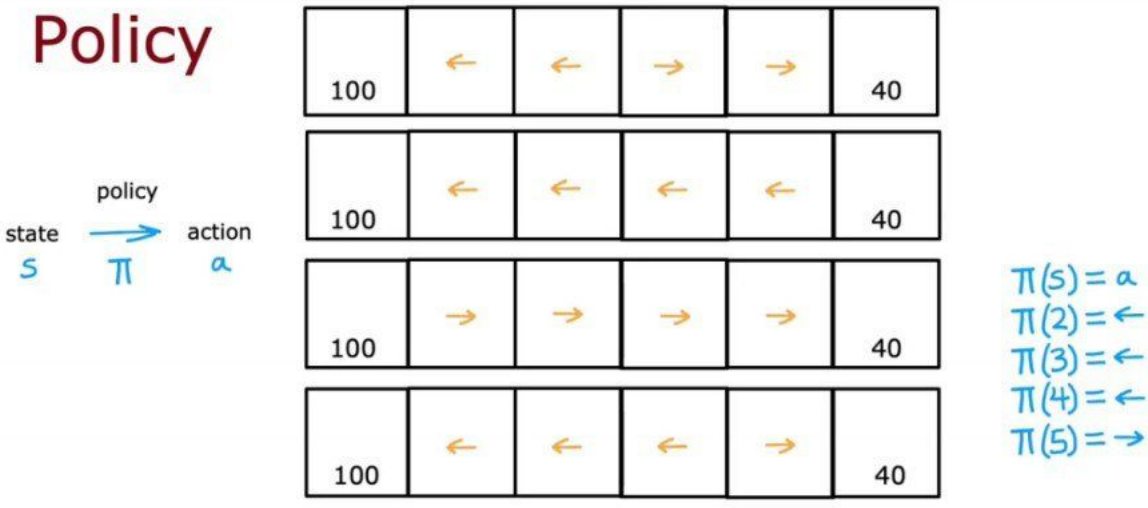
Return $= 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

- The Return decide the Reward and Return is based on the action we take
- Return is the sum of Rewards which is weighted by the discount factor.



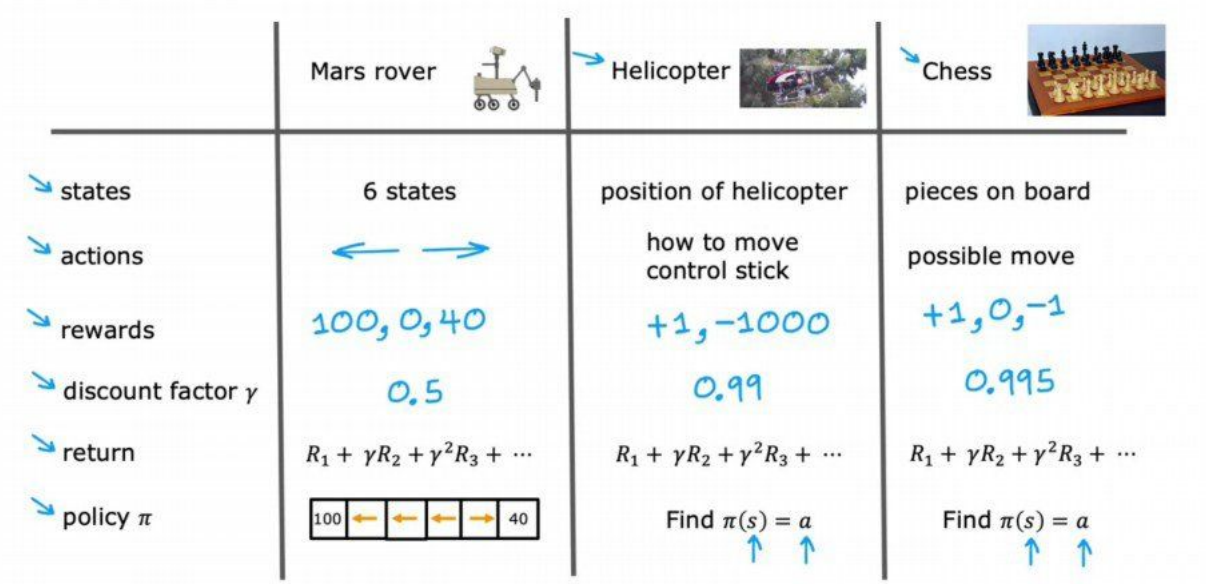
Making Decisions: Policies in Reinforcement Learning

- It is basically a function, which tells you to take what action to take In order to maximize the return.

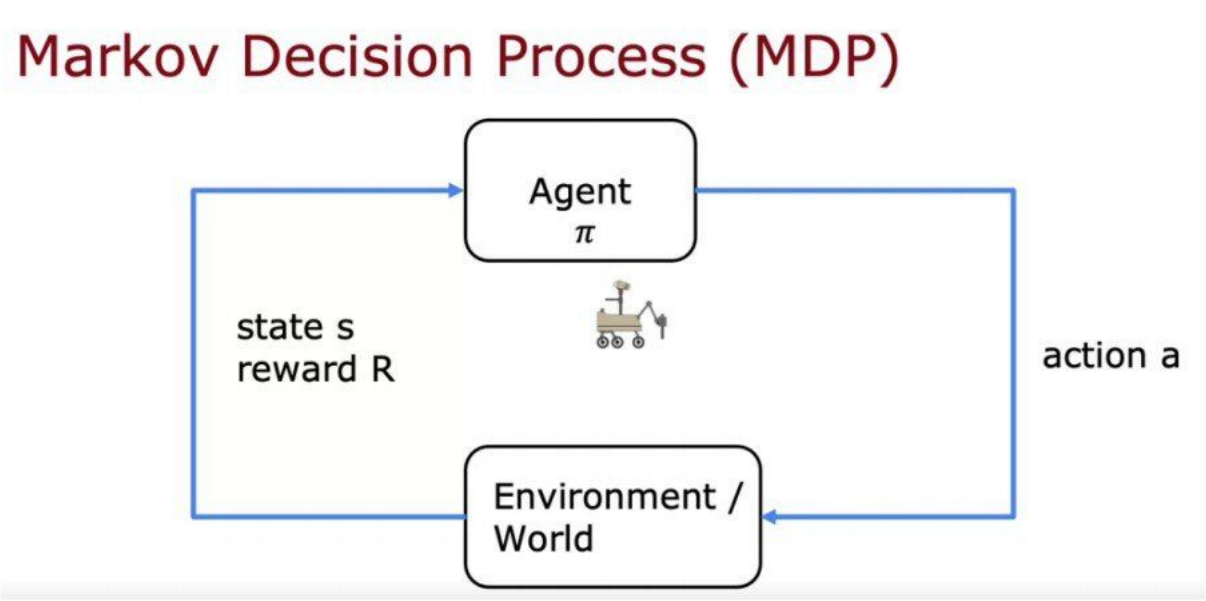


The Goal of Reinforcement Learning is to find a policy pie that tells you what actions (a=pie(s)) to take in every state (s) so as to maximize the return.

Review of Key Concepts



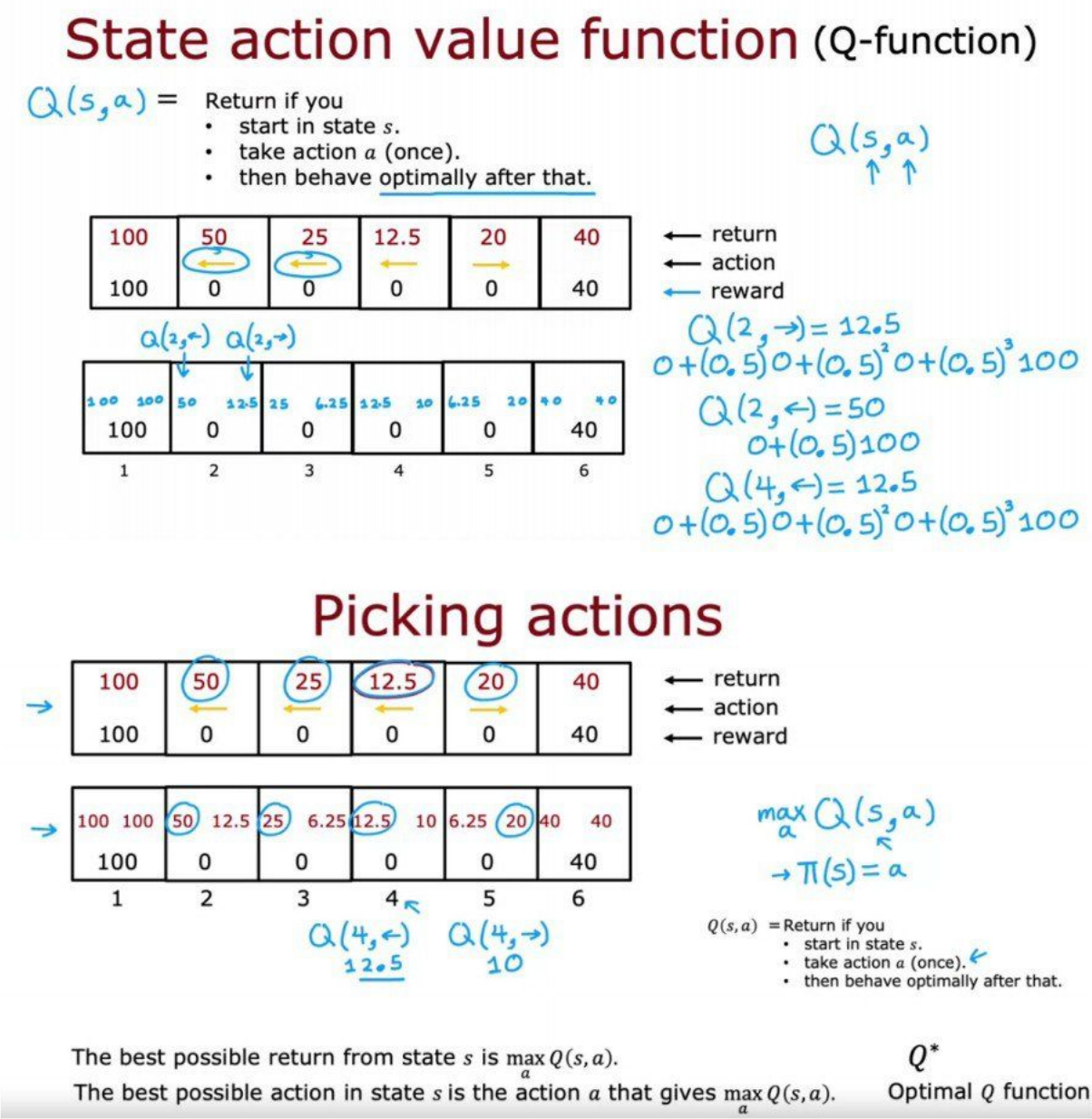
It is a process which defines, future depends on where you are now, not on how you got here.



State-Action Value Function - Q Function

- It is a function denoted by Q, which depends on State and Action. Which helps us to calculate the Return
- Policy and Q is almost similar use function, with minor change
- Best Return will be Maximum of Q Function

- So best Action of State S will be action a that maximize the Q-Function



The Final Reward Value, Discount Factor (gamma) are the one depend upon the

Optimal Policy and 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 [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 [10]:
```
generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	30.0	9.0	3.6	12.0	40.0
100	← 0	← 0	↔ 0	→ 0	40

Q(s,a)

100.0	100.0	30.0	2.7	9.0	1.08	2.7	3.6	1.08	12.0	40.0	40.0
100		0		0		0		0		40	

Bellman Equations

- It helps us to calculate the state action value function (Q-Function)

Bellman Equation

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

$R(1)=100$ $R(2)=0$ ... $R(6)=40$

s : current state
a : current action

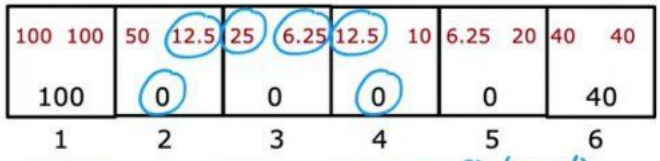
$R(s)$ = reward of current state

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

Bellman Equation

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

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



$s = 2$
$a = \rightarrow$
$s' = 3$

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

$s = 4$
$a = \leftarrow$
$s' = 3$

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

Explanation of Bellman Equation

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

$s \rightarrow s'$

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

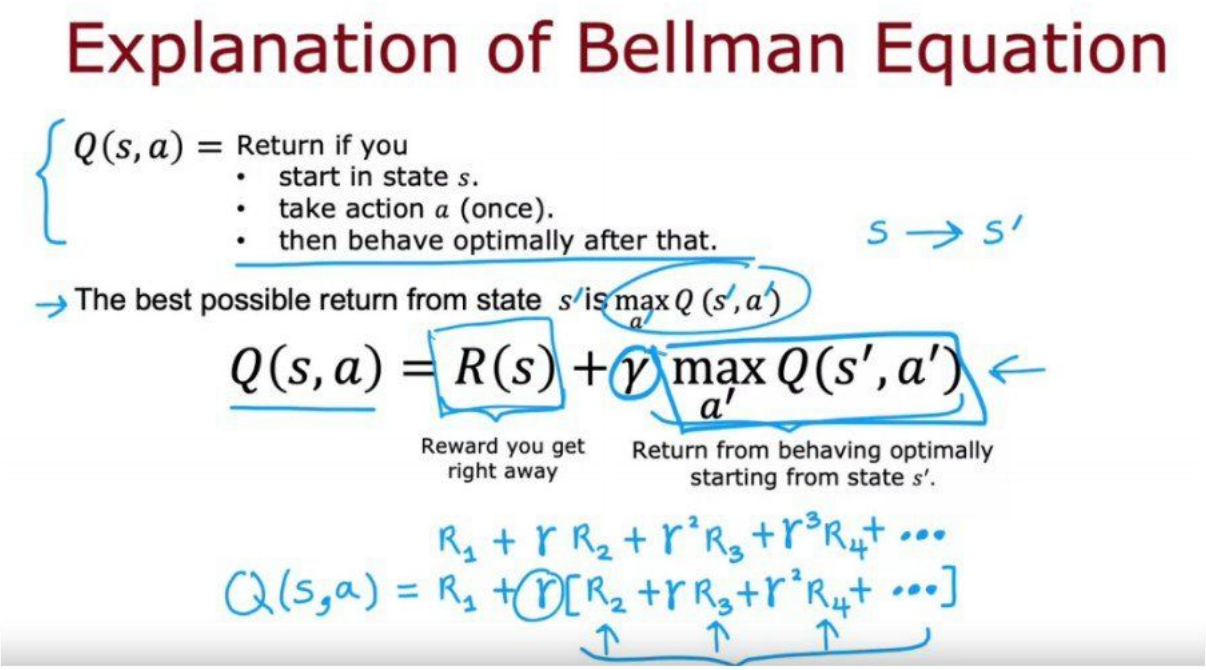
$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

Reward you get right away

Return from behaving optimally starting from state s'.

$R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$

$Q(s,a) = R_1 + \gamma [R_2 + \gamma R_3 + \gamma^2 R_4 + \cdots]$



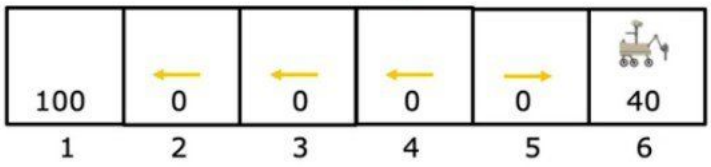
Total Return has two parts

- The reward you get right away
- The reward you get from next state due to our action

Random (Stochastic) Environment

- Suppose that the robot or problem have random output nature. A 0.1 or 10% chance to do the commands in wrong way.
- In stochastic environment there will be different rewards
- So our aim is to Maximize the Average of Sum of Discount of Rewards.
- Maximize the Expected Return
- So Bellman Equation changes with expected maximum of Q-Function

Expected Return



0 0 0 100 0 100
0 0 0 0
0 0 40

Expected Return = Average($R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$)

$= \mathrm{E}[R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots]$

Expected Return

Goal of Reinforcement Learning:

Choose a policy $\pi(s) = a$ that will tell us what action a to take in state s so as to maximize the expected return.

Bellman Equation:

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

3 ←

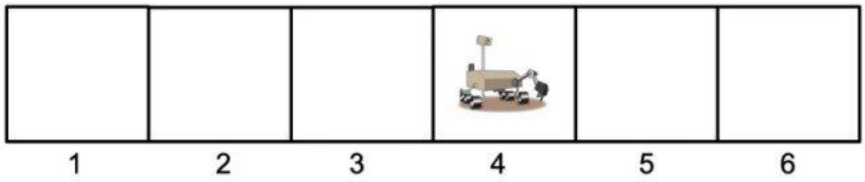
2 or 4

- There will be a mis-step variable in such cases
- If value of misstep is high, there will be decrease in Q-Value
- If value of misstep is low, Q-Value won't have that much impact.

Example of continuous state space applications

- In our robot example, we have fixed state to travel. It is Discrete
- If our goal is to run a truck from 0 to 6 km, it is Continuos
- For a truck since it won't fly. We can have 2 axis X and Y along with theta it turn, while moving in that plane.

Discrete State:



Continuous State:

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

- If we are controlling a helicopter, We have X, Y, Z axis and 3 angle between XY, YZ, XZ axis

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

Lunar Lander

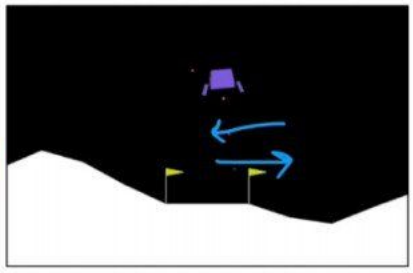
It mainly has 4 actions to do

- Do nothing
- Left Thruster
- Right Thruster
- Main Thruster

We can represent them as a vector of X, Y and tilt Theta along with the binary value l and r which represent left or right leg in ground

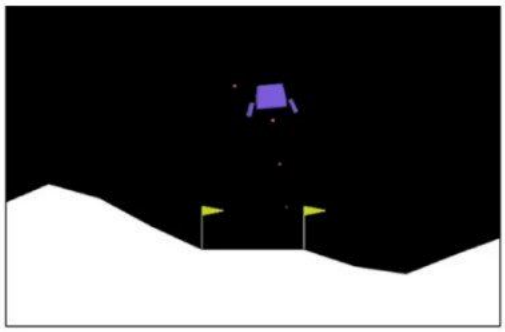
Reward Function

- Getting to landing pad: 100 – 140
- Additional reward for moving toward/away from pad.
- Crash: -100
- Soft landing: +100
- Leg grounded: +10
- Fire main engine: -0.3
- Fire side thruster: -0.03



Learn a policy π that, given

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

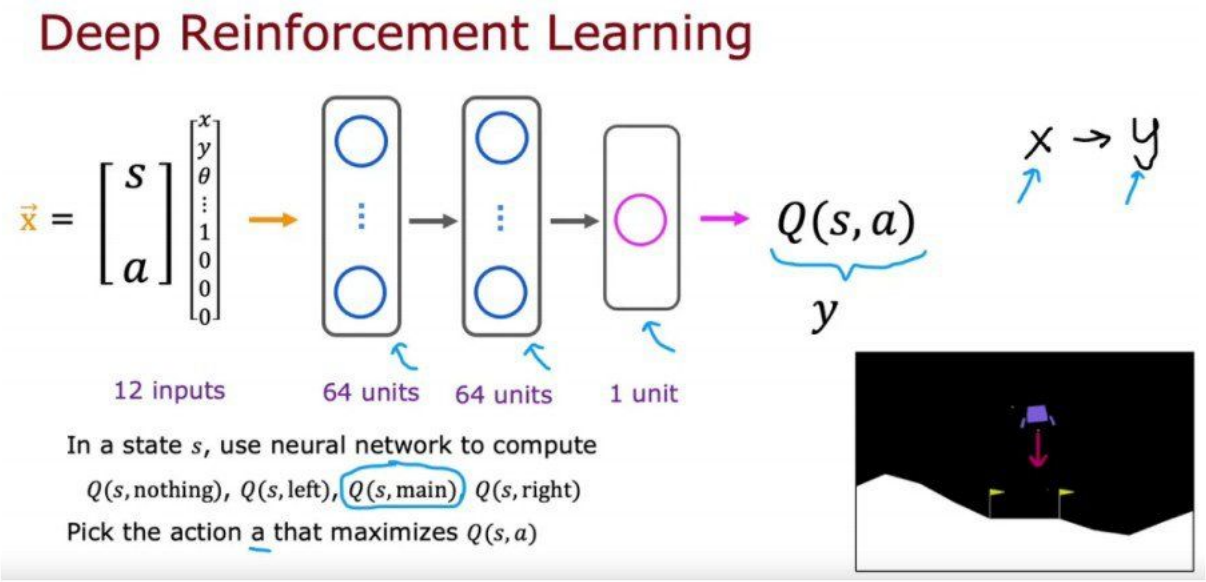


picks action $a = \pi(s)$ so as to maximize the return.

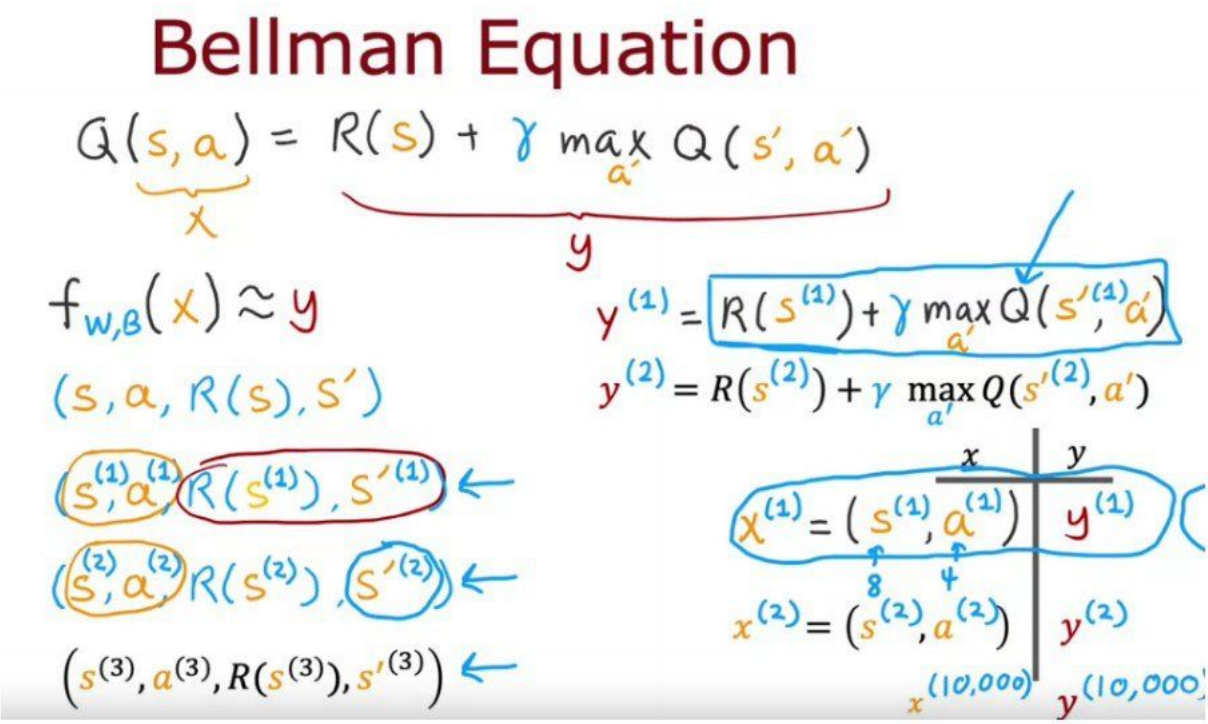
$$\gamma = 0.985$$

Learning the State-Value Function

- 12 inputs where 6 from the X, Y and theta before and after case, 2 for left l and right r.
- 4 for four actions represented as 1, 0, 0, 0 or 0, 1, 0, 0 like that.

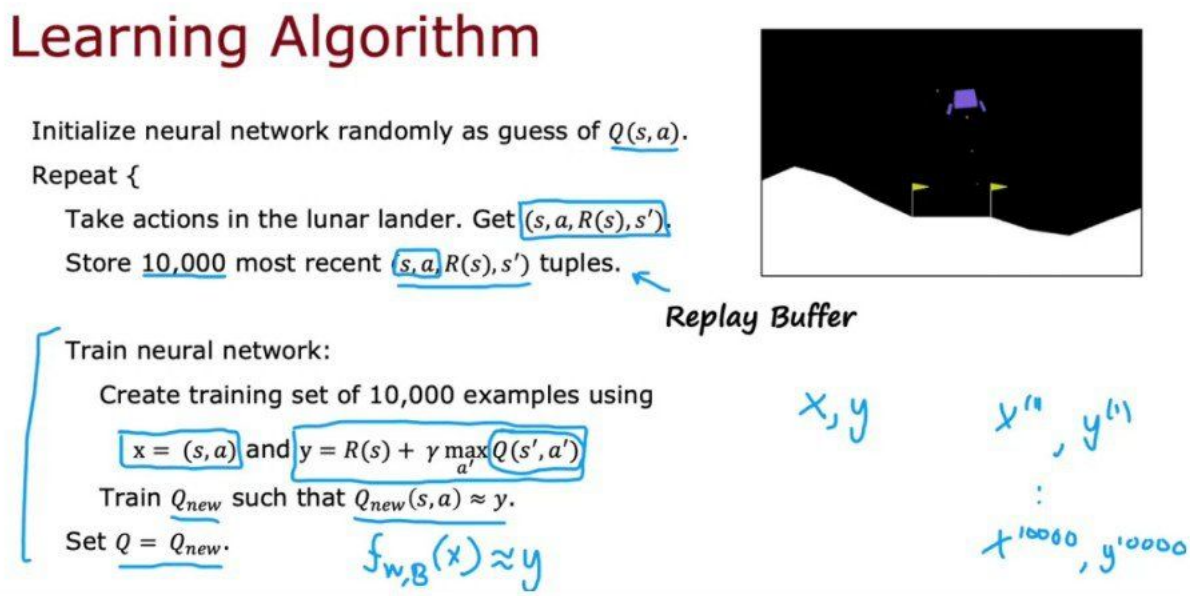


So here we apply similar to linear regression where we predict y based on a input x function



Learning Algorithm

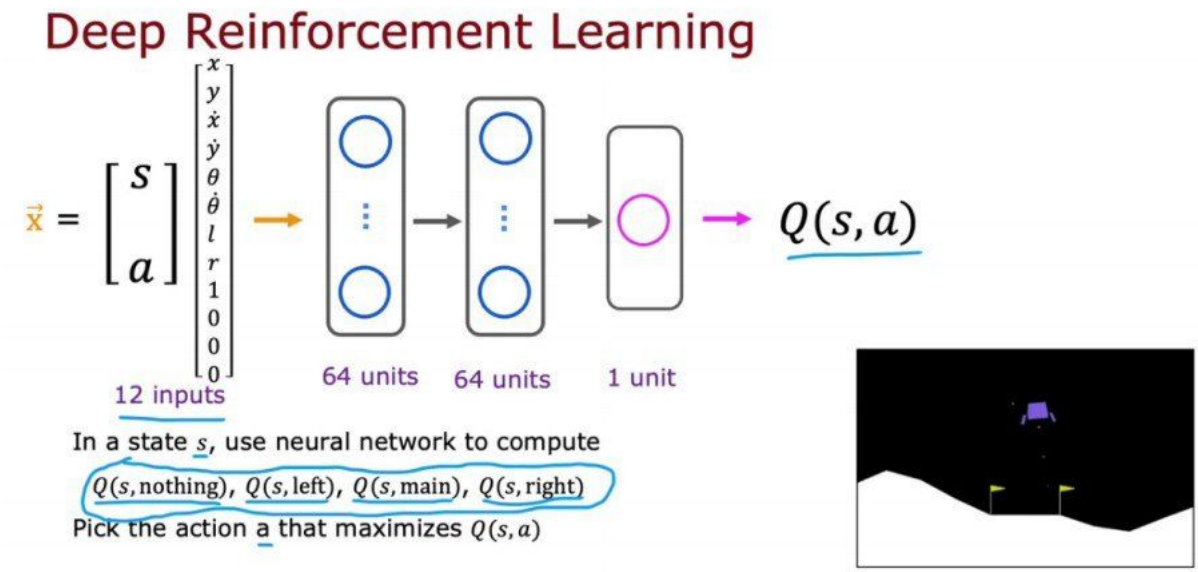
- Initialize the NN randomly as guess of Q(s, a)
- Repeat actions and store 10000 most recent one. It is called Replay Buffer
- Train NN using the 10000 training set
- Then calculate new Q-Function



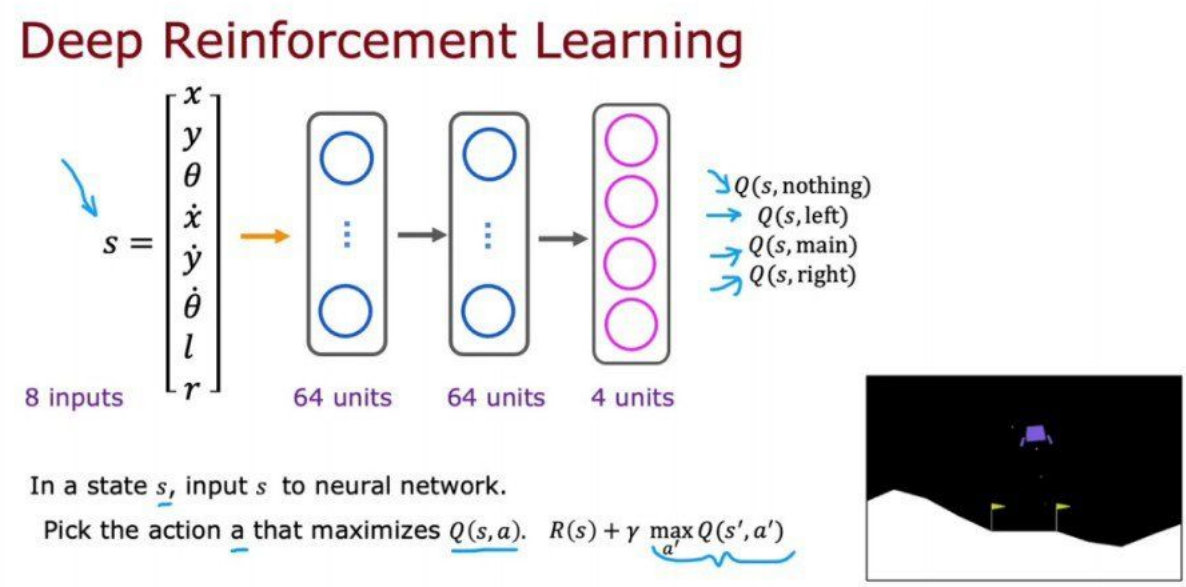
This algorithm is called DQN - Deep Q Network, we use deep learning and NN, to learn the Q-Function

Algorithm refinement: Improved neural network architecture

The current algorithm is inefficient, because we have to carry 4 inference of action from each state.

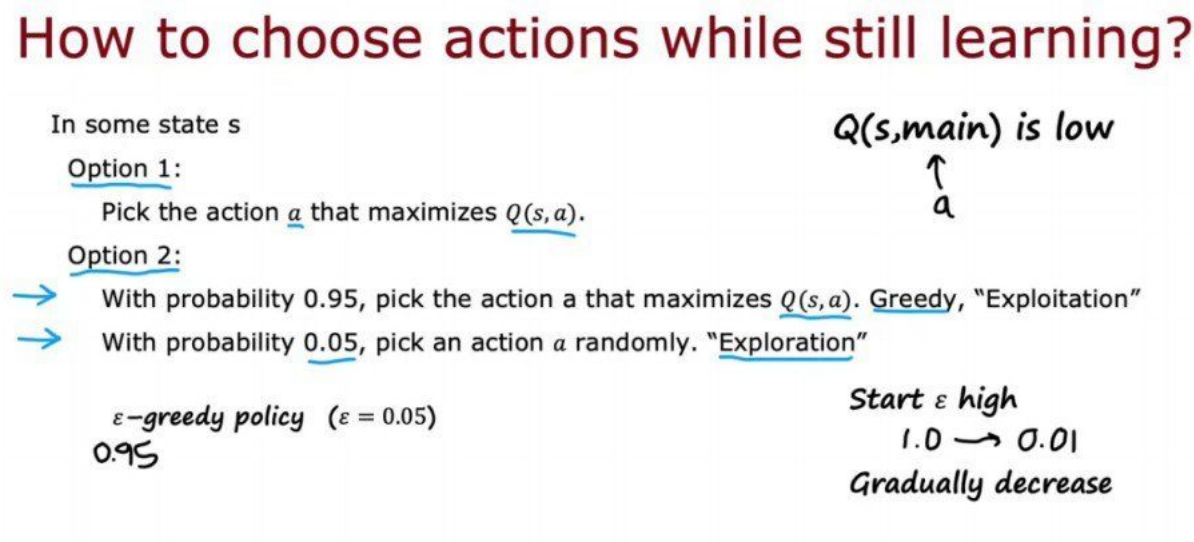


But if we change the NN by 8 input it become much more efficient.



Algorithm refinement: ϵ-greedy policy

- In order to learn things even it's a bad idea, random actions helps. Otherwise it will end up doing things which Maximize the Q Function and won't be prepared for the worse.
- This idea of picking randomly is called Exploration
- 1 - Exploration is also called a Greedy Action/Exploitation Step
- It have epsilon greedy policy, which give percentage of random picking



Start at high epsilon to get more random actions, slowly decrease it to learn the correct one. Learning will take more time in Reinforcement Learning, when parameters are not set in a correct way.

Algorithm Refinement: Mini-Batch and Soft Updates

- If data is huge, In Gradient Descent we need to find average and derivative in each time.
- So GD becomes slow
- We can use Mini-Batch to solve this issue
- We will pick subset of the complete data to run the GD. So it becomes fast.

It works on Supervised Learning and Reinforcement Learning

How to choose actions while still learning?

x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870

100,000,000

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

m = 100,000,000

m' = 1,000

repeat {

$$w = w - \alpha\frac{\partial}{\partial w}\frac{1}{2m'}\sum_{i=1}^{m'}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$b = b - \alpha\frac{\partial}{\partial b}\frac{1}{2m'}\sum_{i=1}^{m'}\left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)^2$$

}

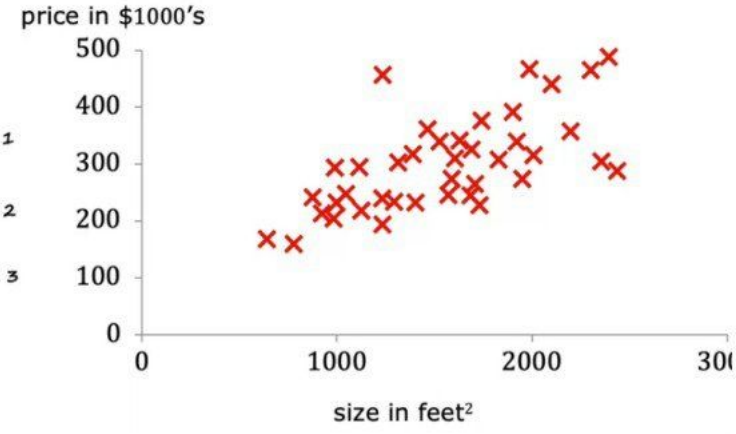
Mini-batch

x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870

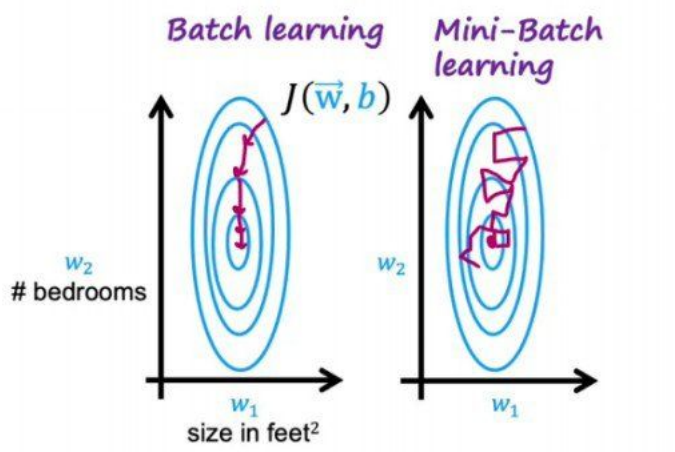
mini-batch 1

mini-batch 2

mini-batch 3

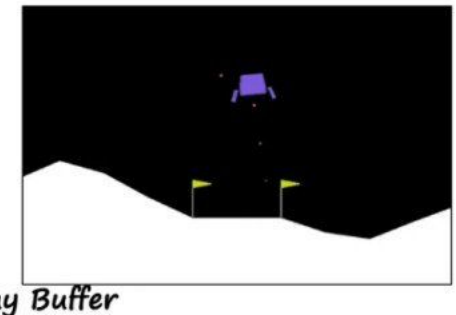


x	y
2104	400
1416	232
1534	315
852	178
...	...
3210	870



In case of training if 10000 examples are available, we only use a subset 1000 each time to become much faster, and little noisy.

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 model:

1,000

Create training set of ~~10,000~~ examples using

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

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

Set $Q = Q_{new}$.

$x^{(1)}, y^{(1)}$

$\vdots$

$x^{(1000)}, y^{(1000)}$

We only choose a subset with little different from the old one. We take time to learn the old things. Soft Update take care of this. We change gradually.

The State of Reinforcement Learning

Limitation

- Work easily on Game, Not in real Robot.
- Few application using RL than Supervised and Un-Supervised Learning.
- But existing research going on.

Overview

Courses

- **Supervised Machine Learning: Regression and Classification**
  - Linear regression
  - logistic regression
  - gradient descent
- **Advanced Learning Algorithms**
  - Neural networks
  - decision trees
  - advice for ML
- **Unsupervised Learning, Recommenders, Reinforcement Learning**
  - Clustering

- anomaly detection
- collaborative filtering
- content based filtering
- reinforcement learning

Thanks for reading this far…..

Made By Arjunan K