Unsupervised Learning, Recommenders, Reinforcement Learning

K-means algorithm

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

Repeat {

 # Assign points to cluster centroids

for i = 1 to m

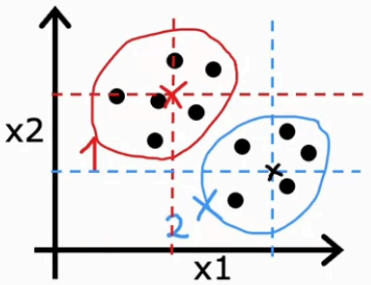
$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

 # Move cluster centroids

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

}



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

Cost function

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

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

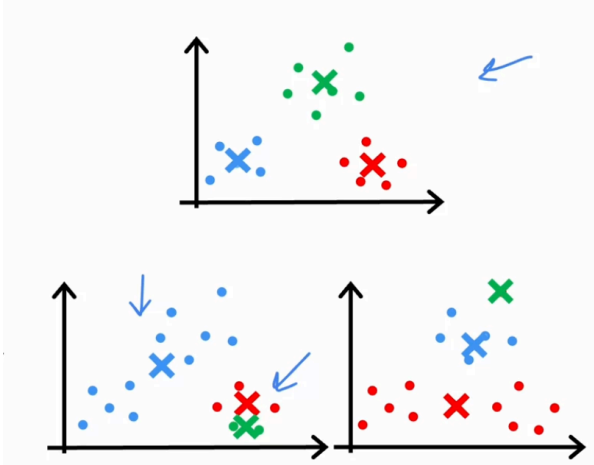
Distortion

Random initialization

Choose $K < m$

Randomly pick K training examples.

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



With random initialization using some of the data points themselves, depending on which points you choose, we can end up with different clusters. Local minima of the cost function may be found. We can run it with multiple random initializations and then pick the one with lowest final cost function.

Random initialization

50-1000

For $i = 1$ to 100 {

k random examples

Randomly initialize K-means.

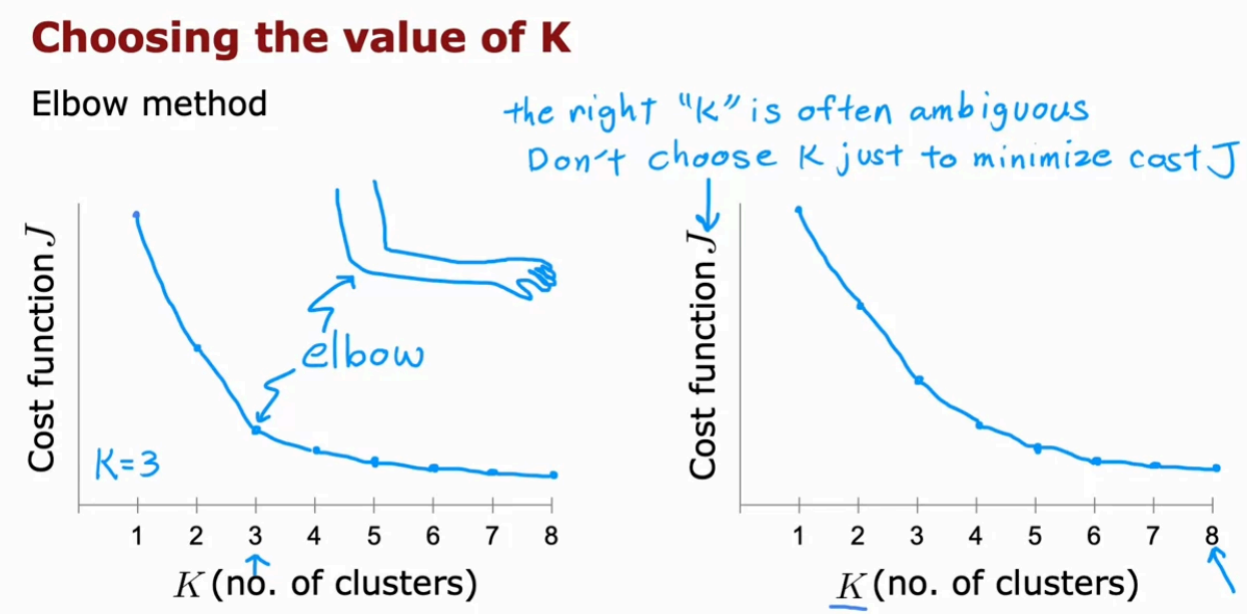
Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \mu_1, \ldots, \mu_k$

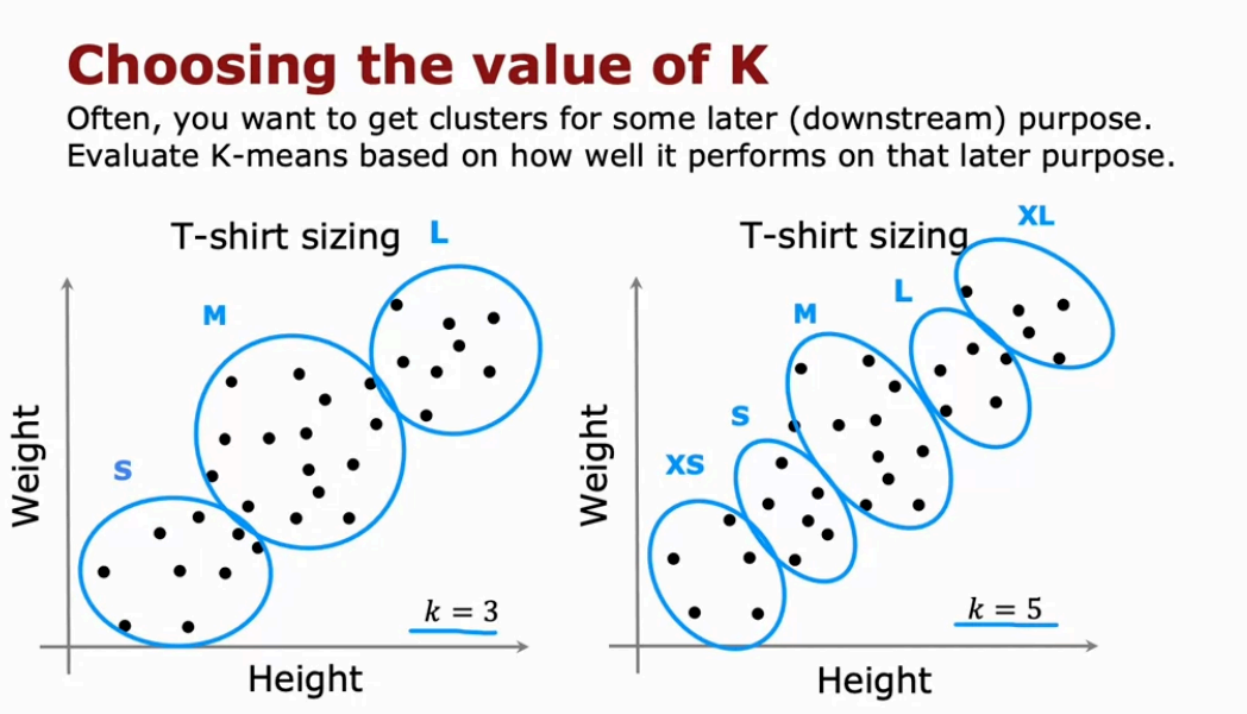
Computer cost function (distortion)

$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \mu_1, \ldots, \mu_k)$

}

Pick set of clusters that gave lowest cost J

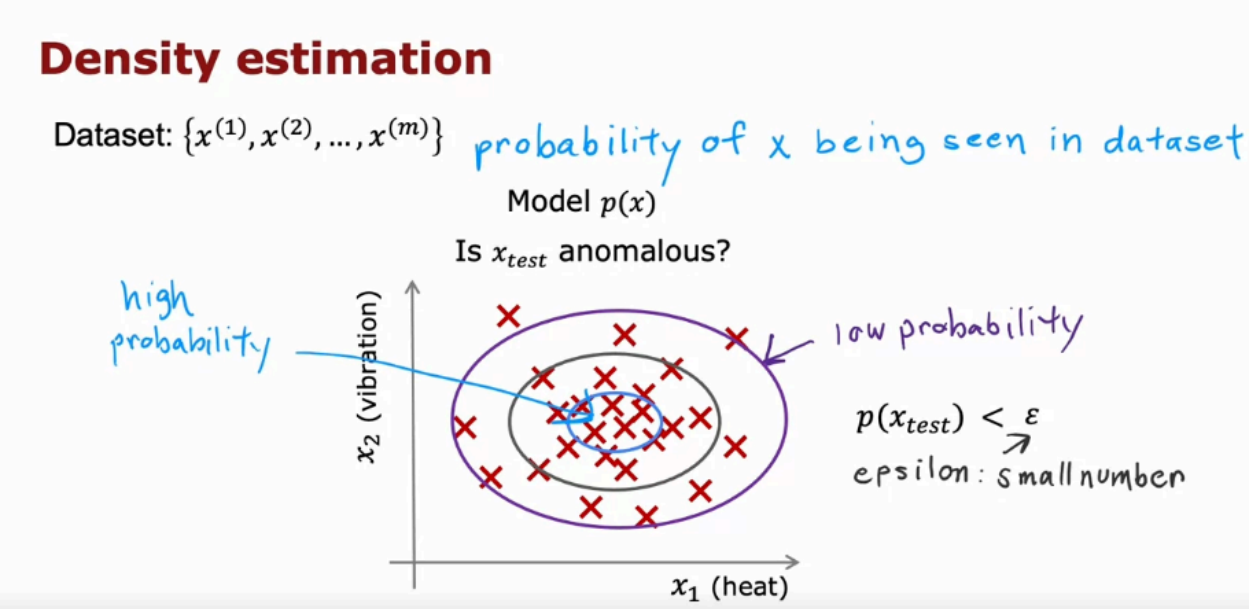


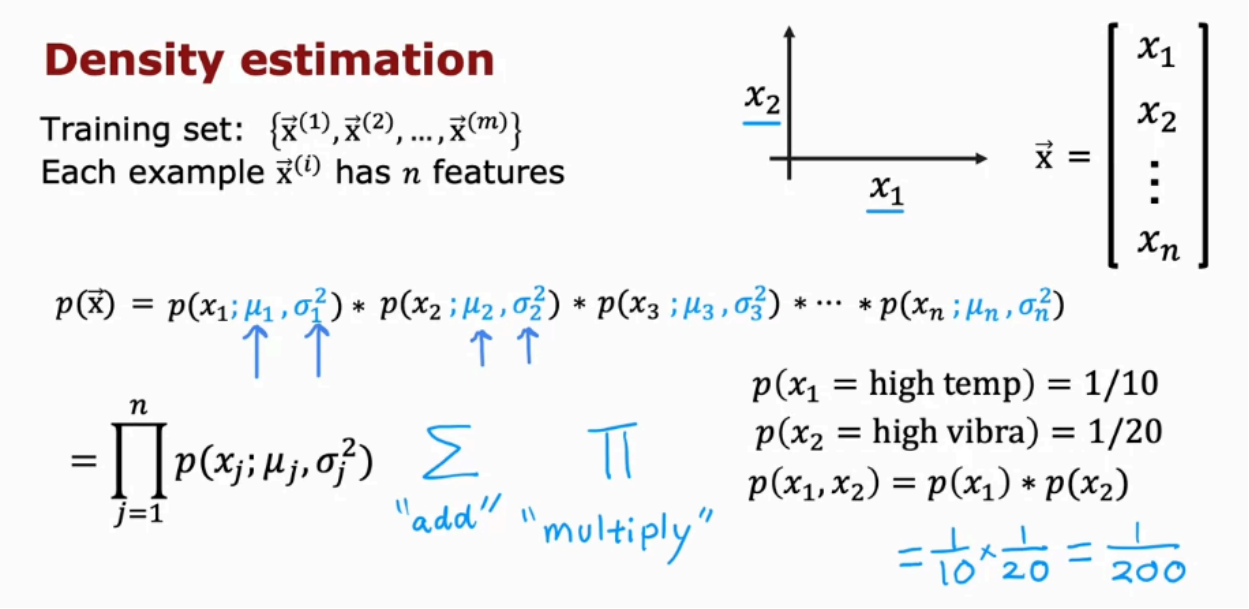


Anomaly Detection

Majority of training examples are normal. Does the test example look like them?

Uses: fraud (e.g. financial), manufacturing defects, computer monitoring in data centers





We fit a Gaussian distribution for each feature that could be indicative of anomaly. The intuition is that is any feature is too small or large, then it's likely an anomaly...

Anomaly detection algorithm

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

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

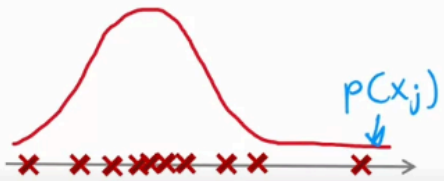
Vectorized formula

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

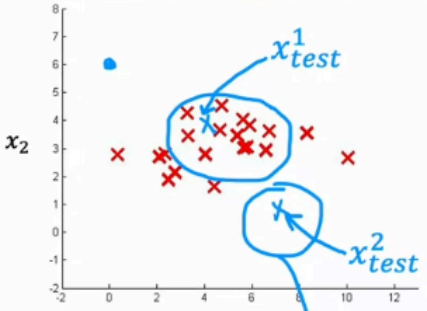
3. Given new example x, compute $p(x)$:

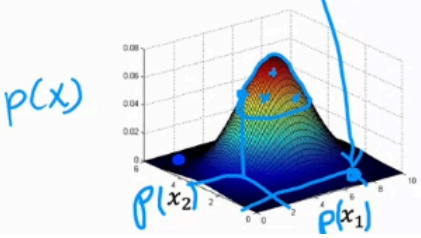
$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2})$$

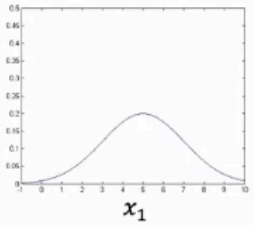
Anomaly if $p(x) < \varepsilon$

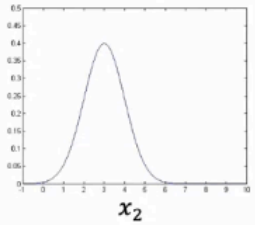


Anomaly detection example









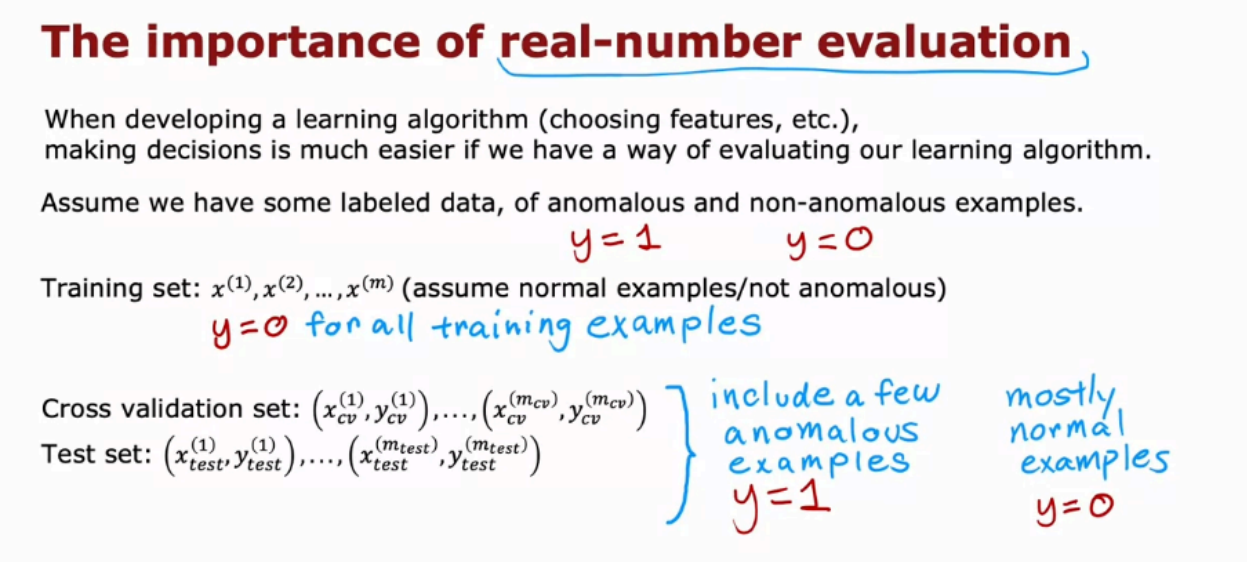
$\mu_1 = 5,\ \sigma_1 = 2$ $\qquad$ $\mu_2 = 3,\ \sigma_2 = 1$

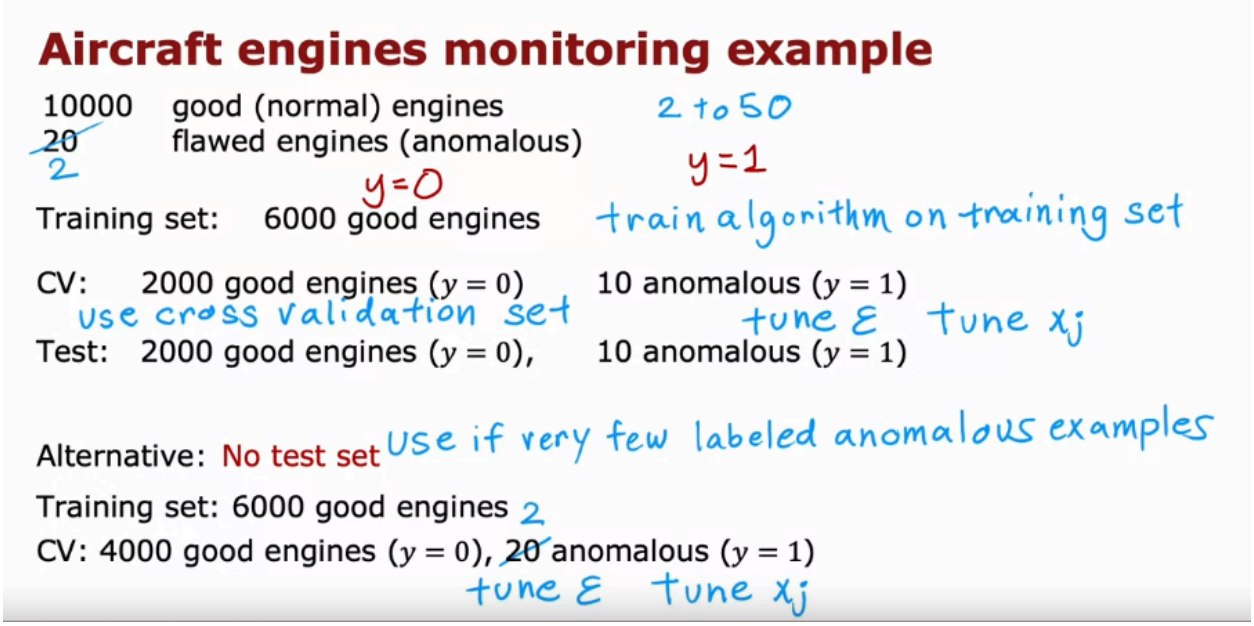
$p(x_1; \mu_1, \sigma_1^2)$ $\qquad$ $p(x_2; \mu_2, \sigma_2^2)$

$\varepsilon = 0.02$

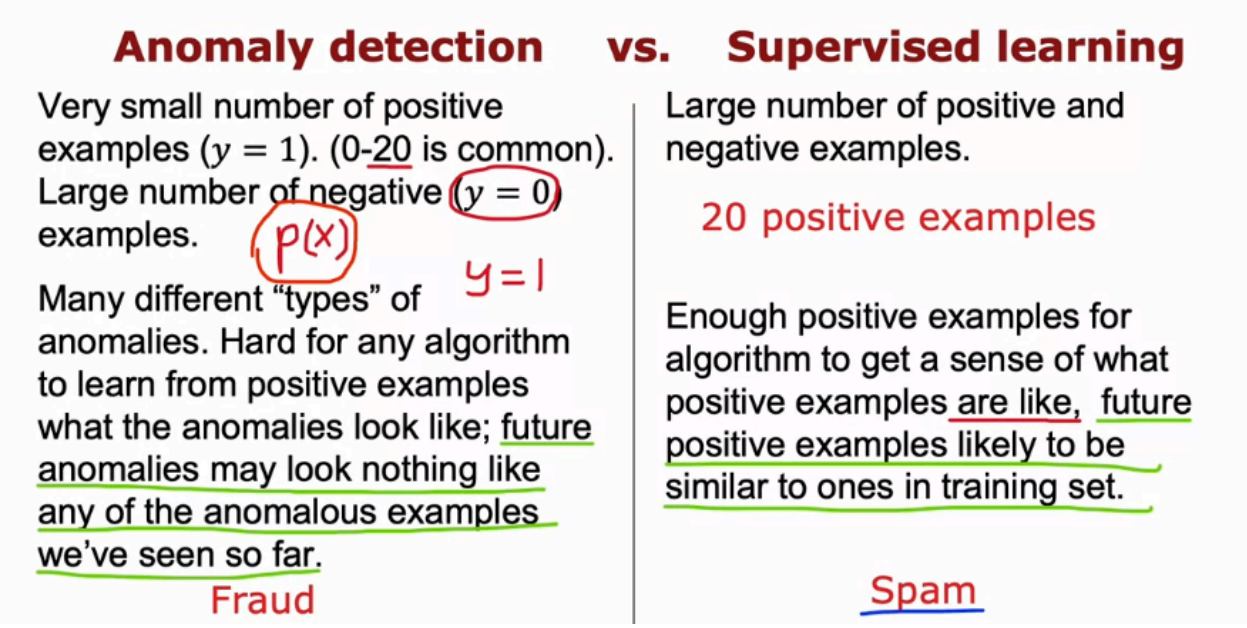
$p\left(x_{test}^{(1)}\right) = 0.0426 \longrightarrow$ "ok"

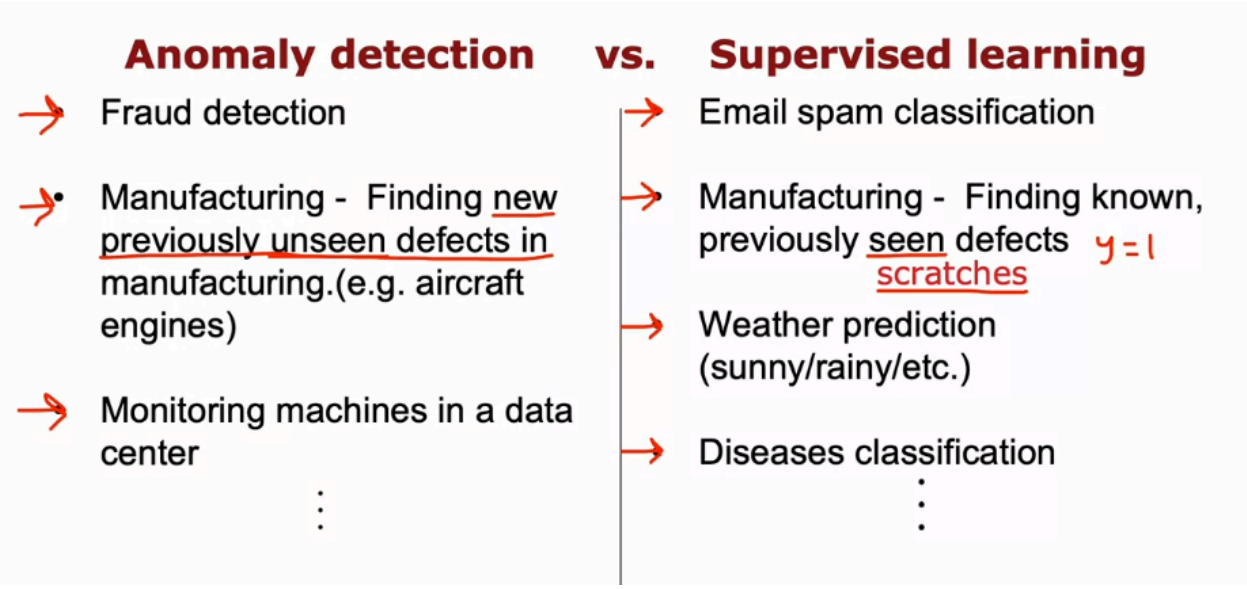
$p\left(x_{test}^{(2)}\right) = 0.0021 \longrightarrow$ anomaly





We use the cross validation to tune parameters of the model and the test set to see how well it's doing. If we have very few anomalous examples, one alternative is to only have a cross validation set, although the chances of overfitting are higher. The algorithm is still unsupervised since the training examples are unlabeled data. We use labels only on the cross validation to evaluate the model that was already fit. We can then get metrics on the cross validation and test sets. We can calculate TP, FP, FN, TN, precision, recall, f1 score, specially if the dataset is really skewed.





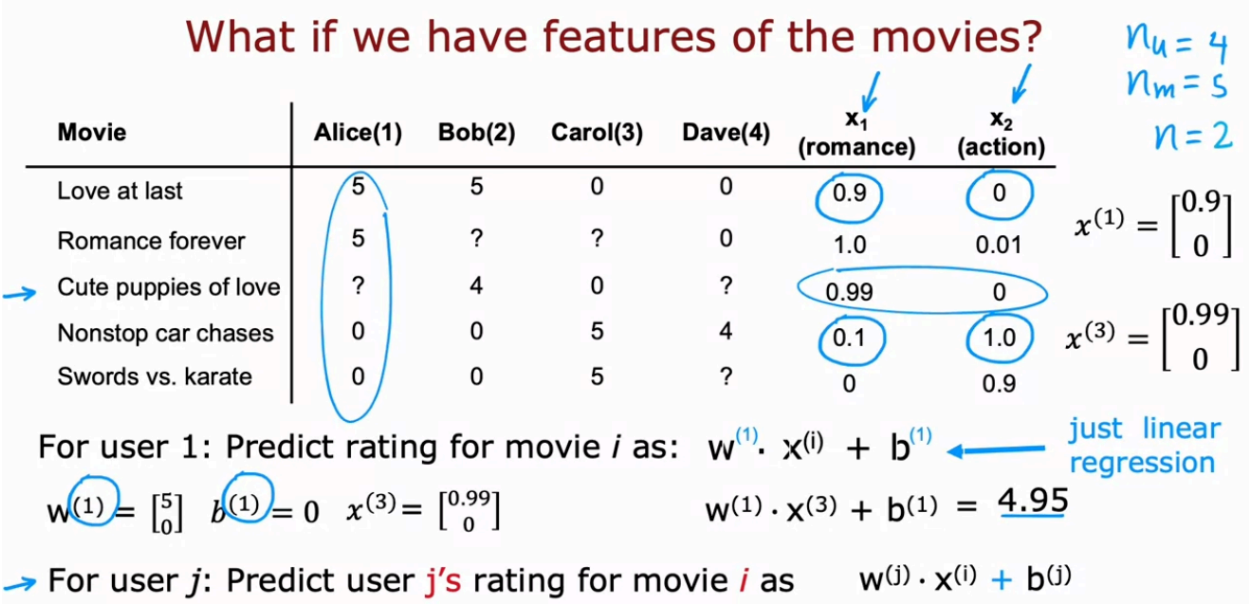
Carefully choosing the right features is more important for anomaly detection than supervised learning.

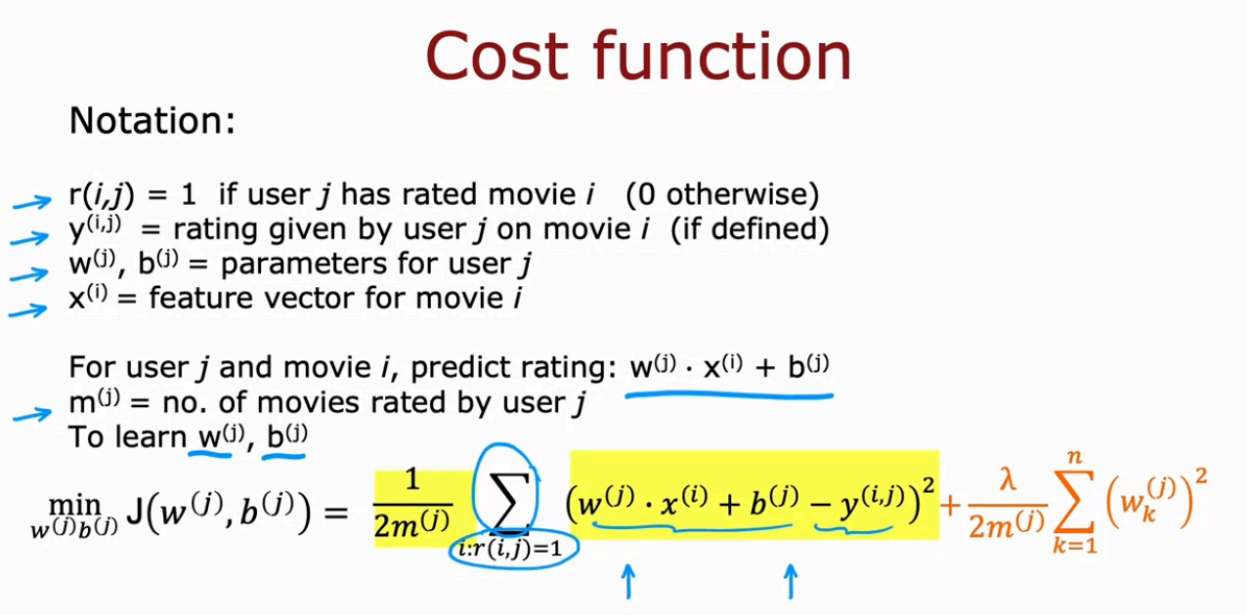
- Pick Gaussian features or adjust non-gaussian features to gaussian (e.g. take the log or the sqrt)
- Error analysis (which anomalies is the algorithm not detecting?) to try and identify a new feature that would help it label the anomaly correctly

Recommender Systems

e.g. Given set of movies that a user has rated, what movies could we recommend next from the ones the user hasn't watched yet?

If we get access to features about the movies, we can fit a linear regression model. We can use regularization to prevent overfitting.





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

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

If we don't have the features, we'd have to learn them. We're now given a set of parameters of the model and minimize a cost function to get the features.

Cost function

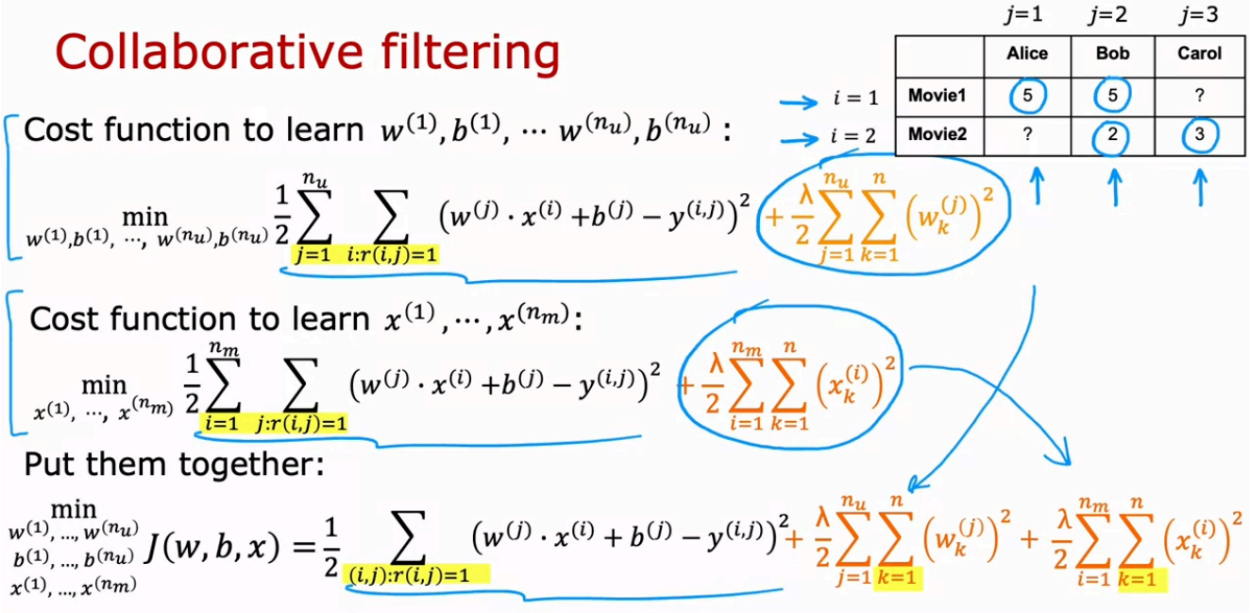
Given $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \cdots, w^{(n_u)}, b^{(n_u)}$

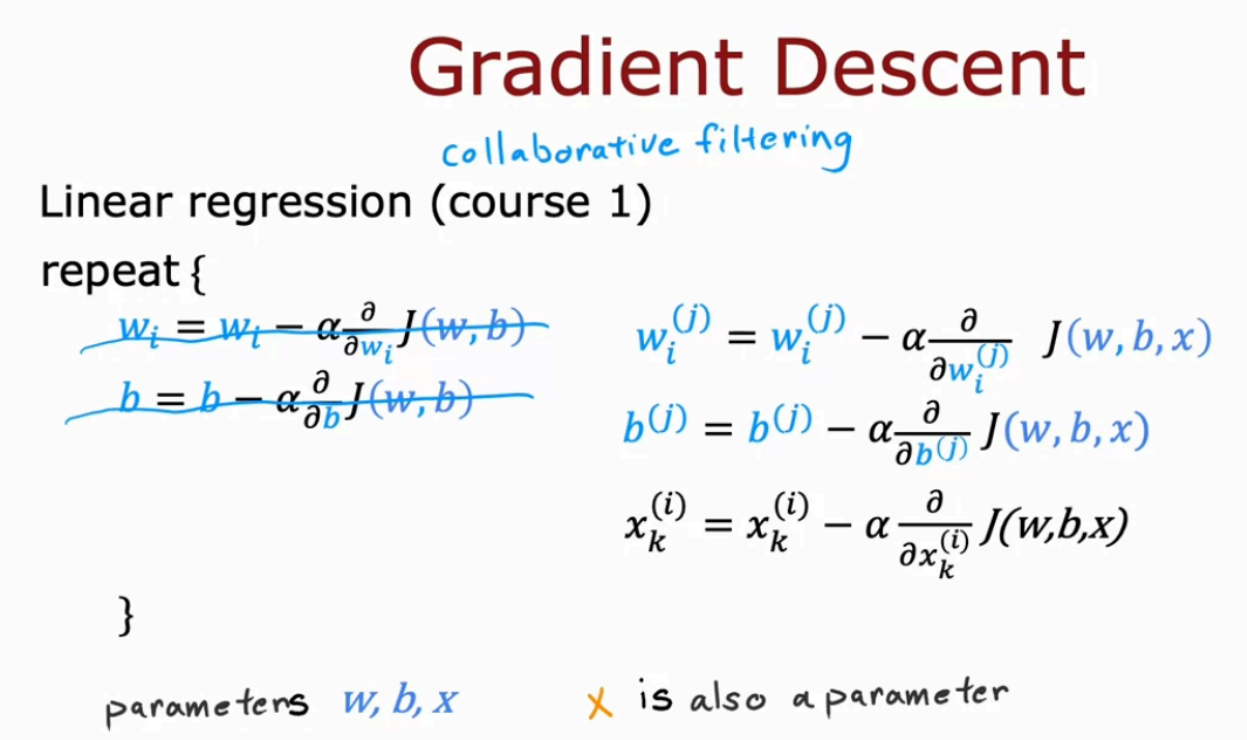
to learn $x^{(i)}$:

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

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

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





Collaborative filtering naming is in the sense that multiple users have helped come up with good values for the features of the model and then we can use these to predict rating for other users in the future.

For binary labels, (e.g. likes, favs, clicks) we can use logistic regression and the corresponding loss function we've seen before.

Cost function for binary application

Previous cost function:

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

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

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

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

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

cost for all examples

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	?	$r^{(i,j)}$
Romance forever	5	?	?	0	?	$r^{(i,j)}$
Cute puppies of love	?	4	0	?	?	$r^{(i,j)}$
Nonstop car chases	0	0	5	4	?	$r^{(i,j)}$
Swords vs. karate	0	0	5	?	?	$r^{(i,j)}$

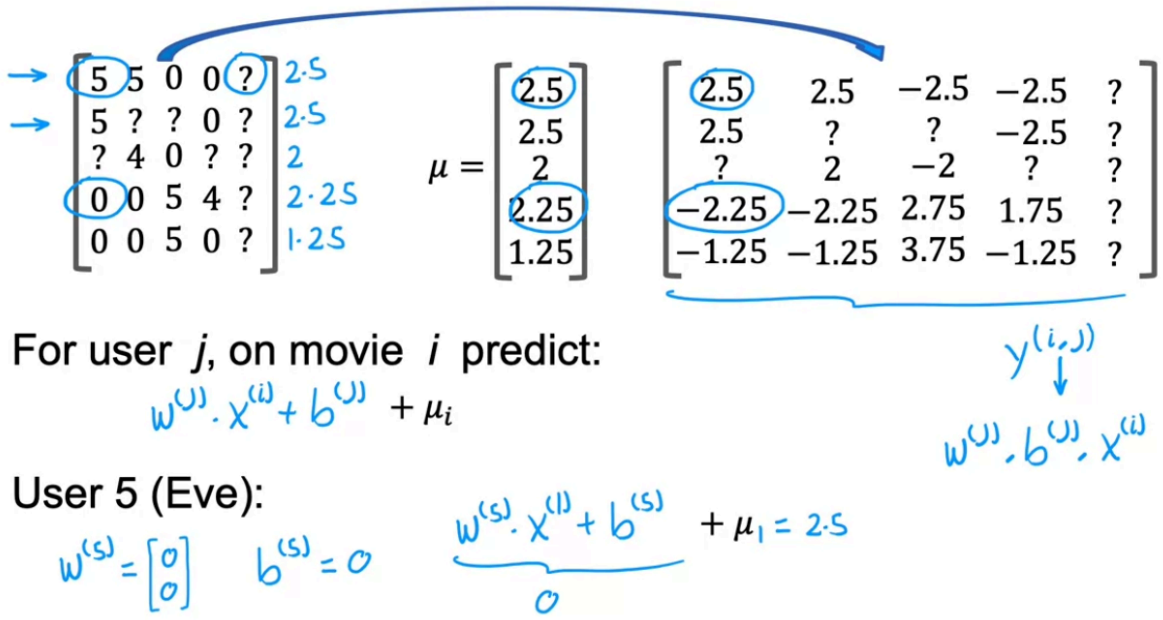
$$\begin{bmatrix}5&5&0&0&?\\5&?&?&0&?\\?&4&0&?&?\\0&0&5&4&?\\0&0&5&0&?\end{bmatrix}$$

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

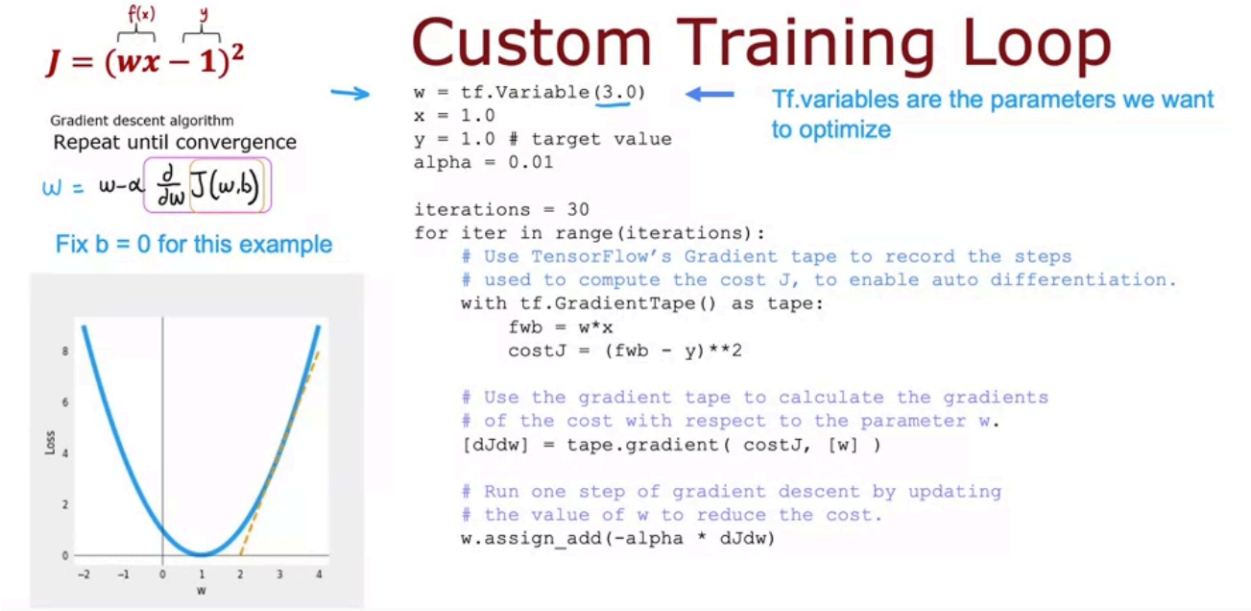
$w^{(5)}=\begin{bmatrix}0\\0\end{bmatrix}$ $b^{(5)}=0$ $w^{(5)}\cdot x^{(i)}+b^{(5)}$

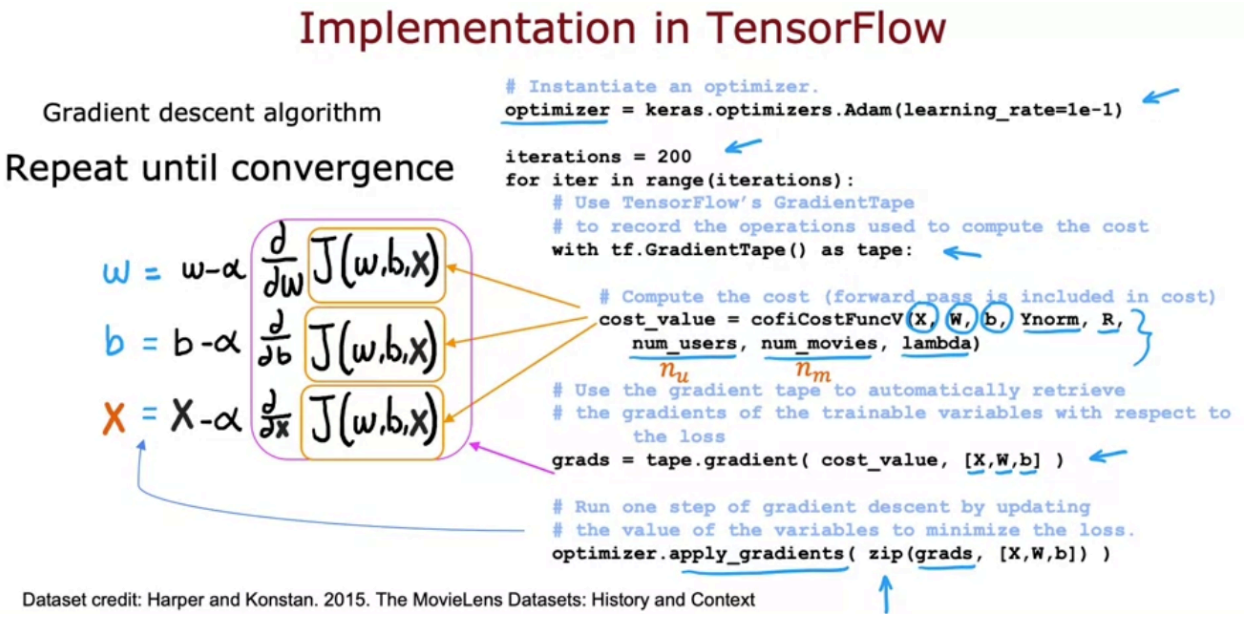
Initial guess for a new user is the average of the ratings for all other users. This makes the algorithm run faster and predictions be more reasonable.

Mean Normalization

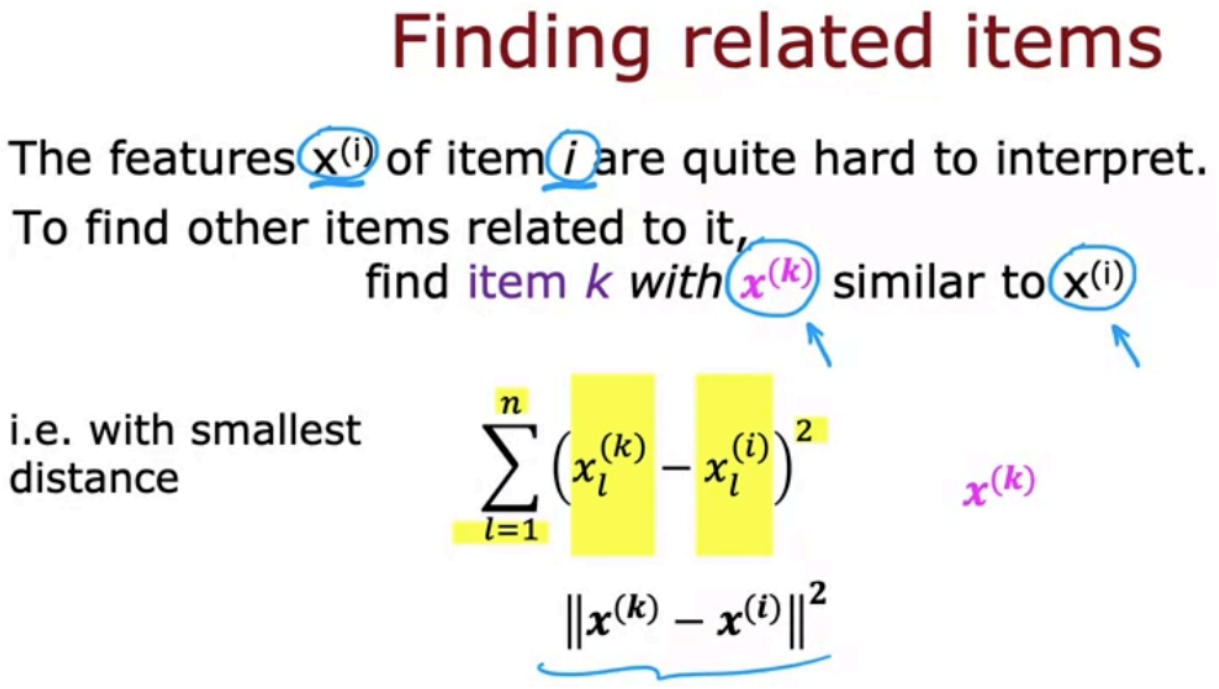


With tensorflow, we only need to tell it how to compute the function and the related cost function. It computes the gradients for us (”auto diff”).





The learned features might be difficult to interpret but nevertheless, we can find similar items if we get the items with smallest distances to any given item.



Collaborative filtering has a cold start problem:

- rank new items that few users have rated?

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

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

Examples of user and item features

User features:

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

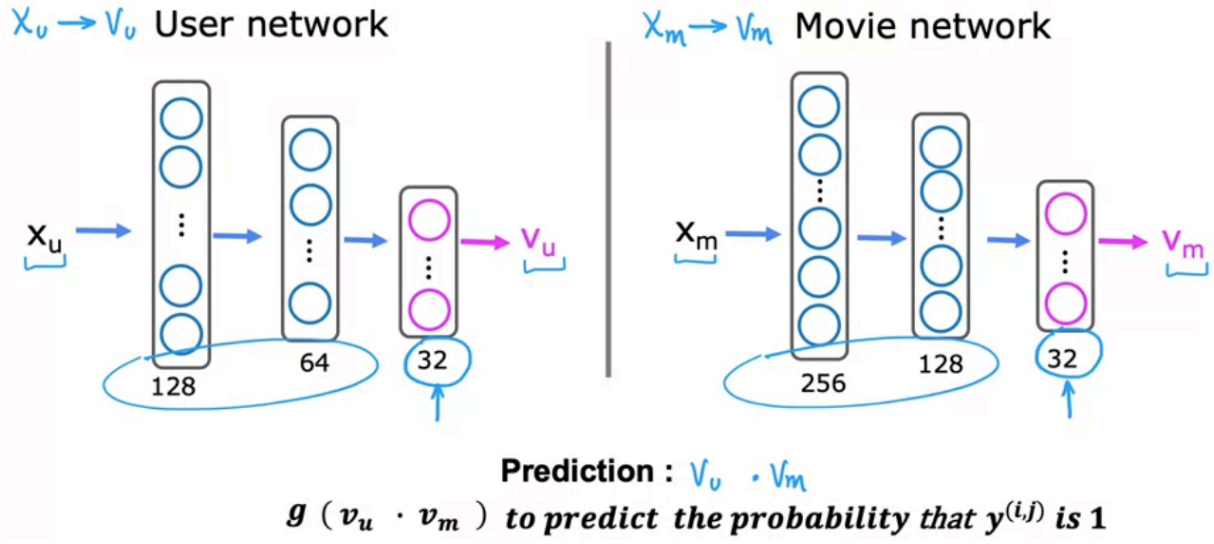
$\mathbf{x_u^{(j)}}$ **for user j**

Movie features:

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

$\mathbf{x_m^{(i)}}$ **for movie i**

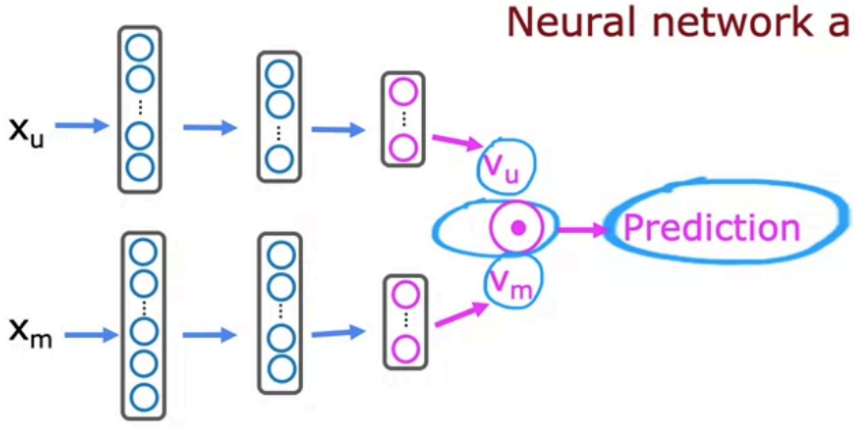
Neural network architecture



Prediction : $v_u \cdot v_m$

$g(v_u \cdot v_m)$ *to predict the probability that* $y^{(i,j)}$ *is* 1

Neural network architecture



Cost function $J = \sum_{(i,j):r(i,j)=1} \left(v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)}\right)^2$ + NN regularization term

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: $\| v_m^{(k)} - v_m^{(i)} \|^2$ small

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

Note: This can be pre-computed ahead of time

Two steps: Retrieval & Ranking

Retrieval:

- Generate large list of plausible item candidates ~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

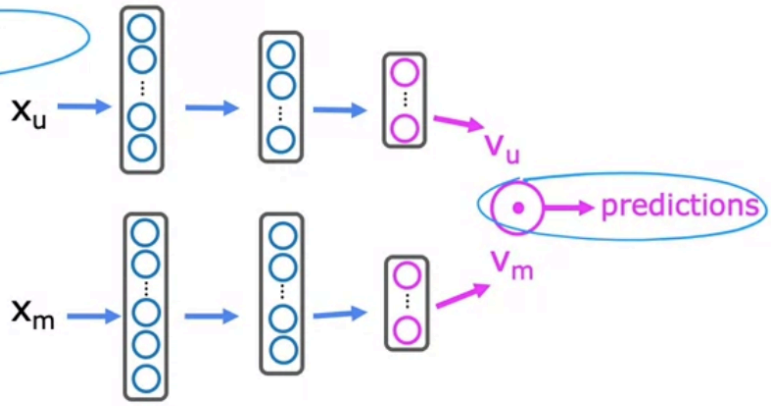
- Combine retrieved items into list, removing duplicates and items already watched/purchased

Two steps: Retrieval & ranking

Ranking:

~100s

- 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).

What is the goal of the recommender system?

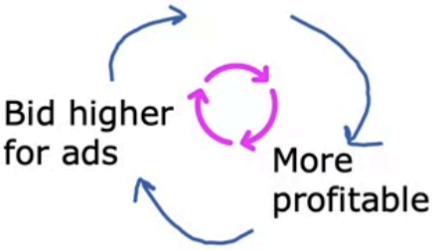
Recommend:

- 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

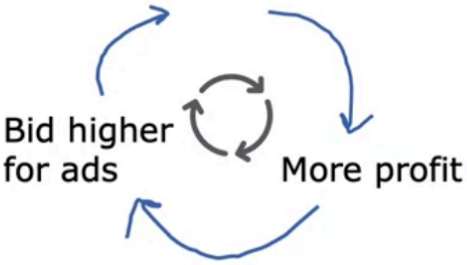
Travel industry

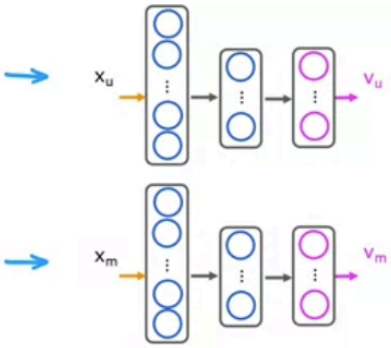
Good travel experience to more users



Payday loans

Squeeze customers more






```
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 )
])
```

```
# 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.l2_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.l2_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.MeanSquaredError()
```

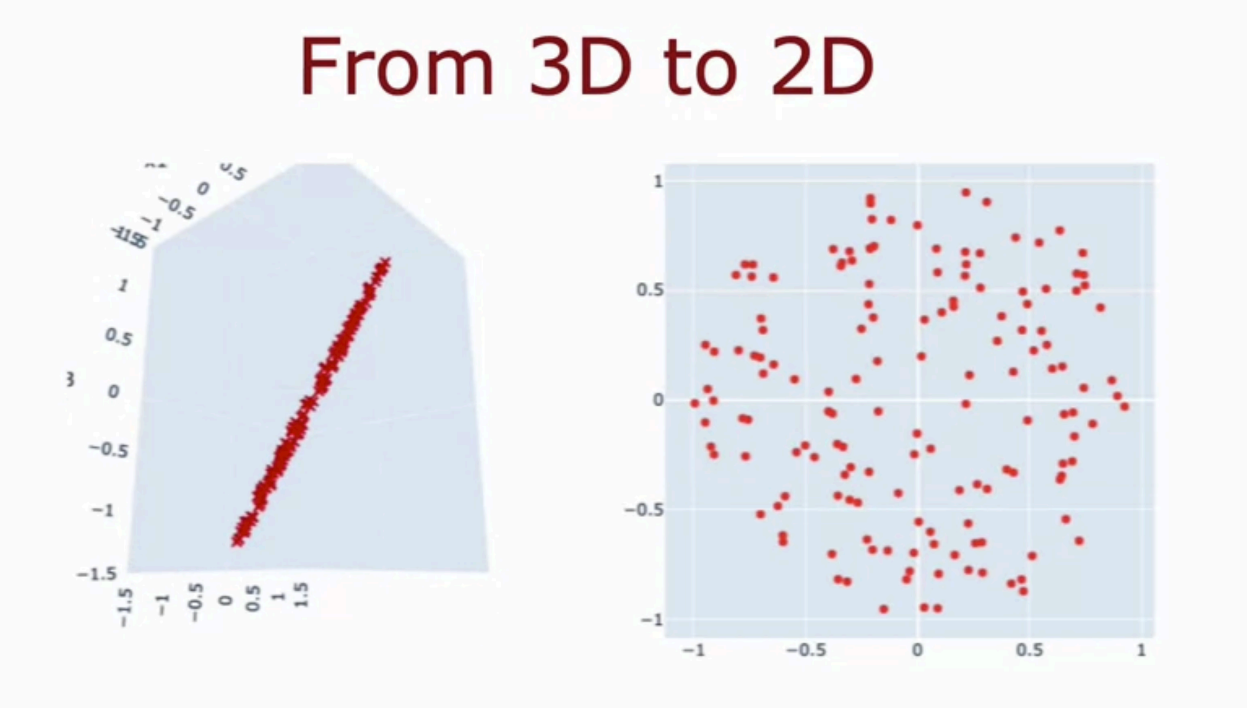
Principal Component Analysis

Used for: **visualization**, data compression (reduce storage or network), speed up training of supervised learning model

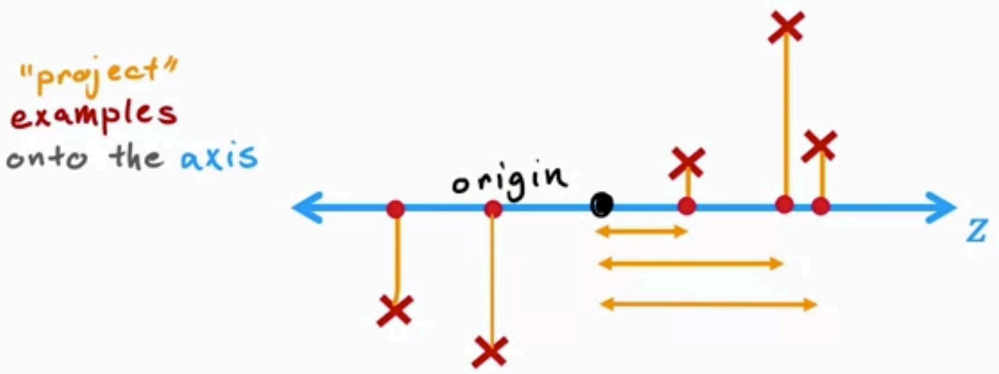
Reduces number of features by finding new axis and coordinates to keep useful information but reduce the number of features to describe that information

e.g. below all the points lie on a plane in 3D, so we can find new coordinates to represent them in 2D

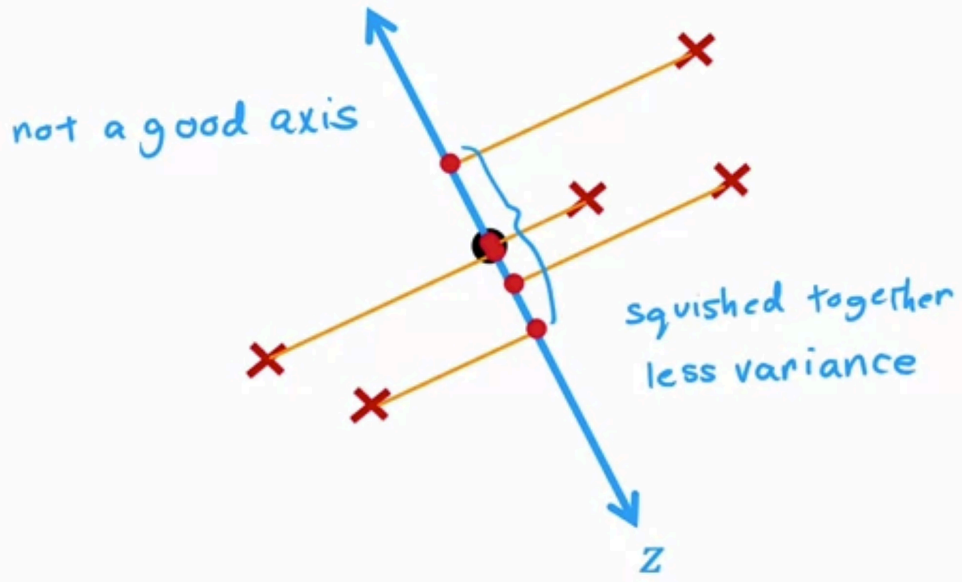
The principal component is the axis that would give more variance to the data points, preserving the most amount of information of the original data.



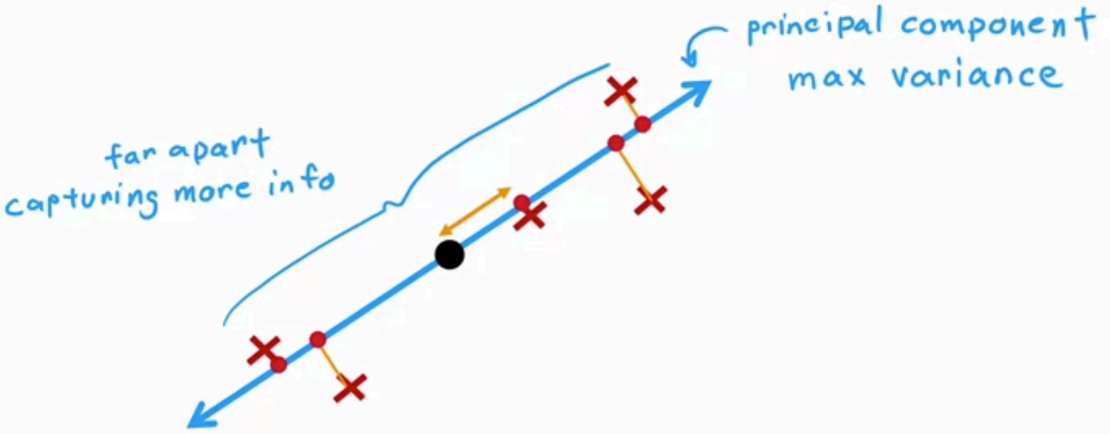
Choose an axis



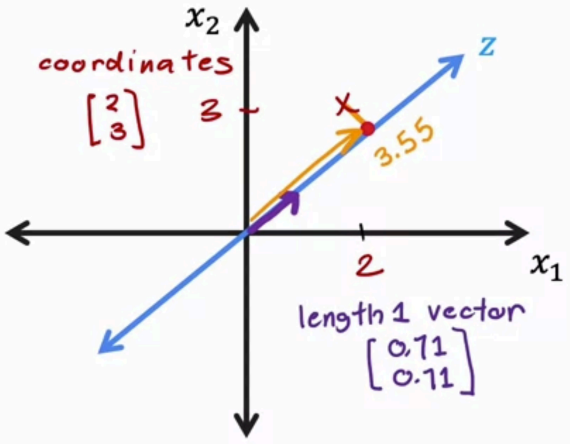
Choose an axis



Choose an axis



Coordinate on the new axis

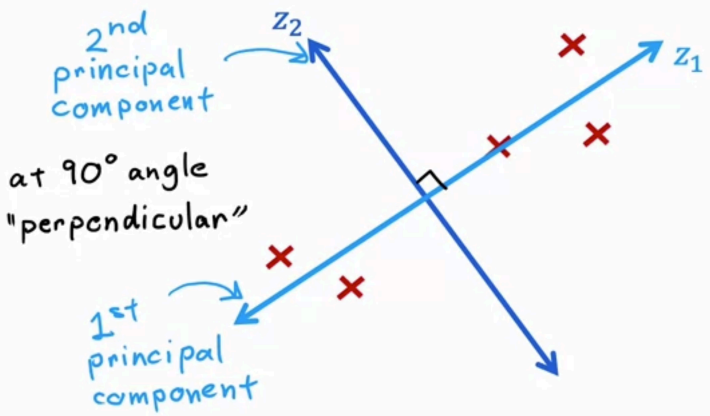


dot product

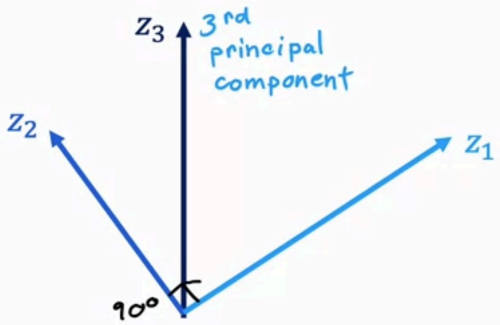
$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix}$$

$2 \times 0.71 + 3 \times 0.71 = 3.55$

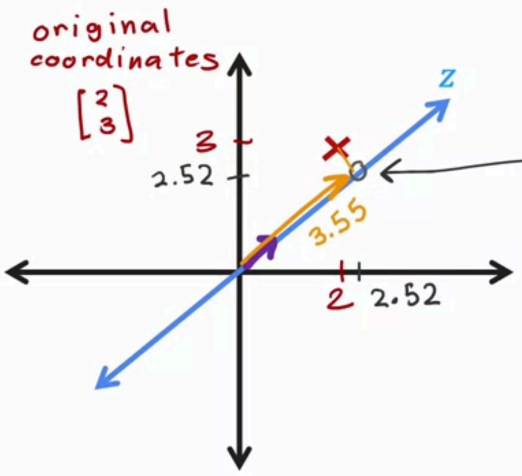
More principal components



50 features → 3 principal components



Approximation to the original data



given $z = 3.55$,
find original (x_1, x_2) (approximately)
"reconstruction"

$$3.55 \times \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix} = \begin{bmatrix} 2.52 \\ 2.52 \end{bmatrix}$$

PCA in scikit-learn

Optional pre-processing: Perform feature scaling

for visualization

1. "fit" the data to obtain 2 (or 3) new axes (principal components)

   `fit` includes mean normalization

info

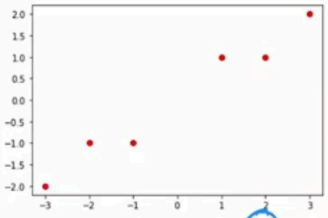
2. Optionally examine how much variance is explained by each principal component.

   `explained_variance_ratio`

3. Transform (project) the data onto the new axes

   `transform`

Example

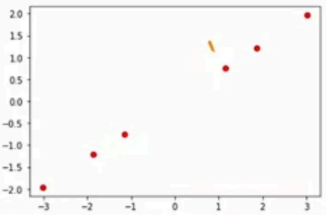


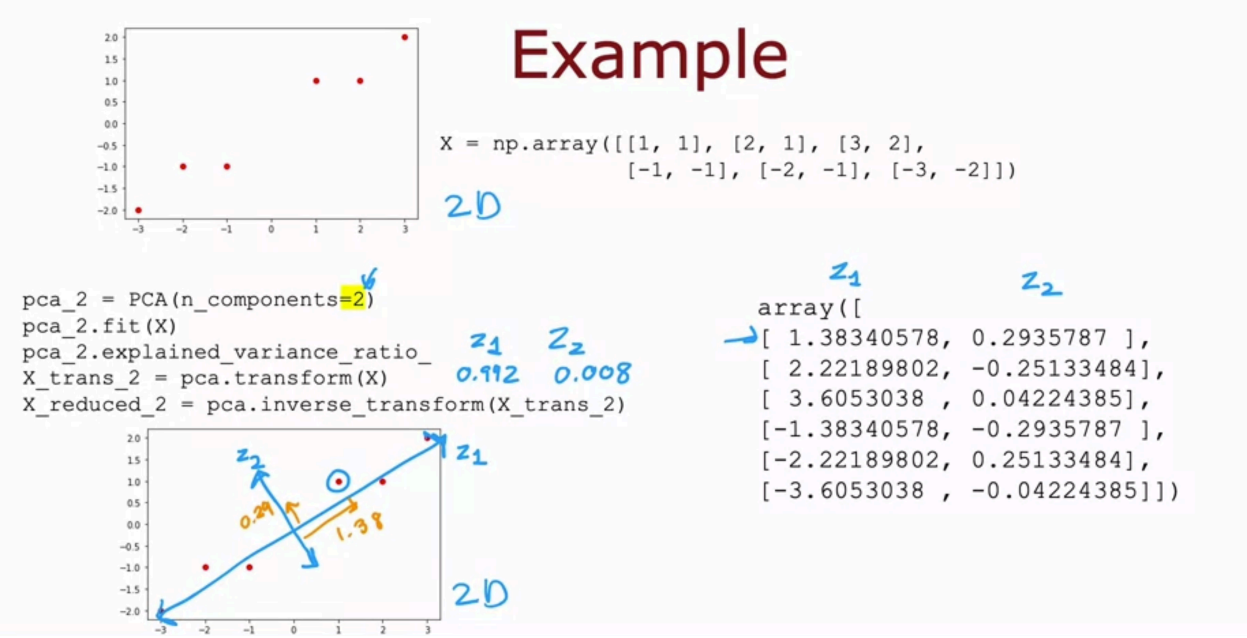
```
X = np.array([[1, 1], [2, 1], [3, 2],
              [-1, -1], [-2, -1], [-3, -2]])
```

```
pca_1 = PCA(n_components=1)
pca_1.fit(X)
pca_1.explained_variance_ratio_
X_trans_1 = pca_1.transform(X)
X_reduced_1 = pca.inverse_transform(X_trans_1)
```

0.992

```
array([
[ 1.38340578],
[ 2.22189802],
[ 3.6053038 ],
[-1.38340578],
[-2.22189802],
[-3.6053038 ]])
```

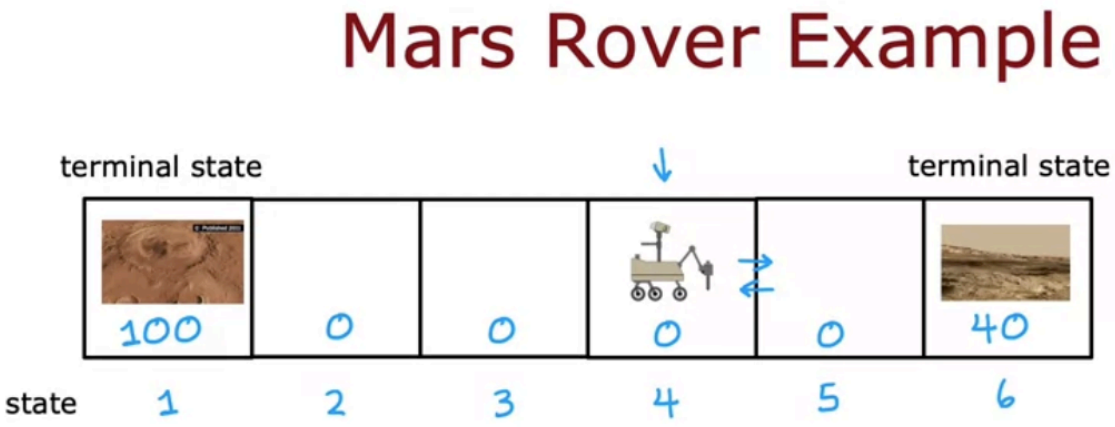




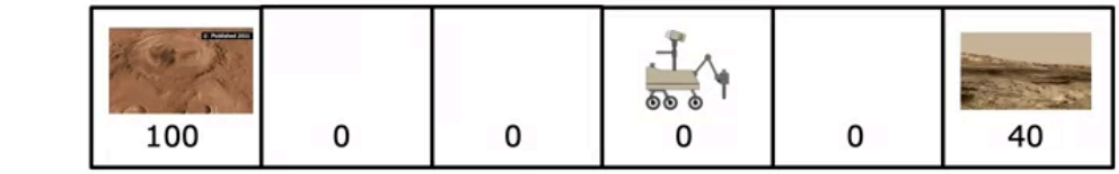
Reinforcement Learning

state s → action a given a reward function

We don't tell the action to take but the reward function and the algorithm decides by itself which action results in better rewards.



Return

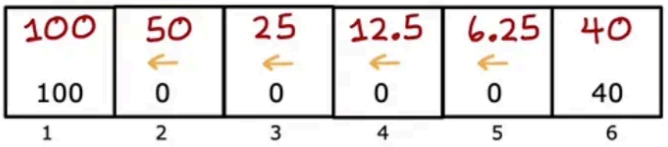


Return $= 0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100 = 0.729 \times 100 = 72.9$

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

Discount Factor $\gamma = 0.9$

Example of Return

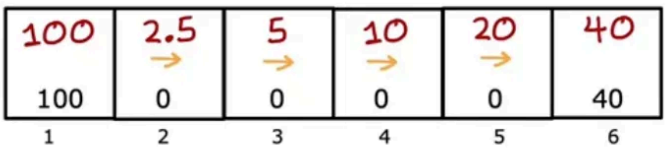


← return

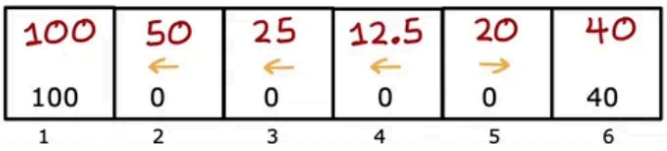
← reward

$\gamma = 0.5$

The return depends on the actions you take.

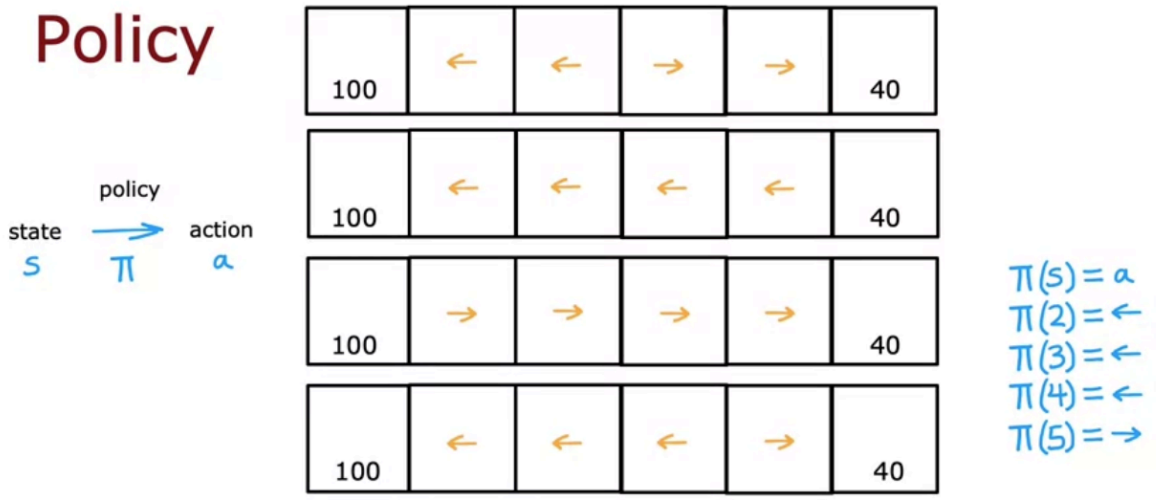


$0 + (0.5)0 + (0.5)^2 40 = 10$



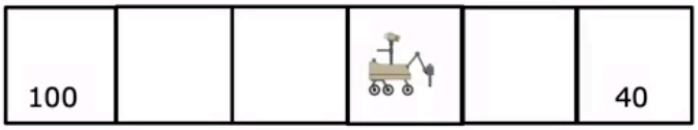
$0 + (0.5)40 = 20$

Policy



A policy is a function $\pi(s) = a$ mapping from states to actions, that tells you what action *a* to take in a given state *s*.

The goal of reinforcement learning



Find a policy π 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	← →	how to move control stick	possible move
rewards	100, 0, 40	+1, −1000	+1, 0, −1
discount factor γ	0.5	0.99	0.995
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$
policy π		Find $\pi(s) = a$	Find $\pi(s) = a$

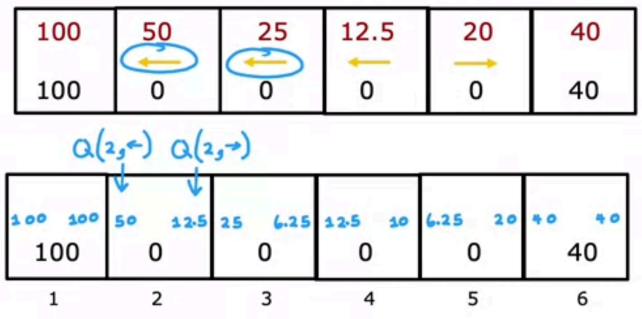
"Markov Decision Process"

"Markov" - the future depends only on the current state, not how you got to the current state

State action value function

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

$Q(s, a)$

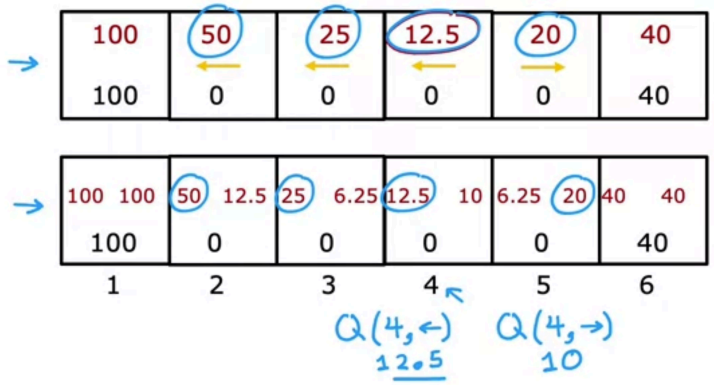


$Q(2, \rightarrow) = 12.5$
$0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

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

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

Picking actions



$\max_a Q(s,a)$

$\pi(s) = a$

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

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

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

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 $\max_{a'} Q(s',a')$

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

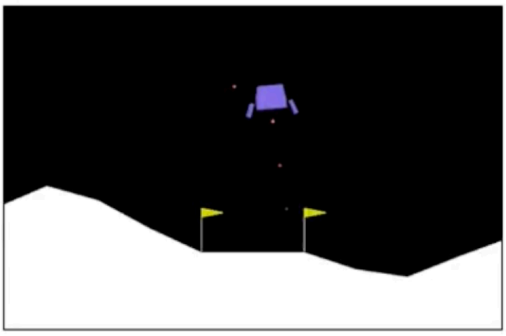
Reward you get right away

Return from behaving optimally starting from state s'.

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

Lunar Lander Problem

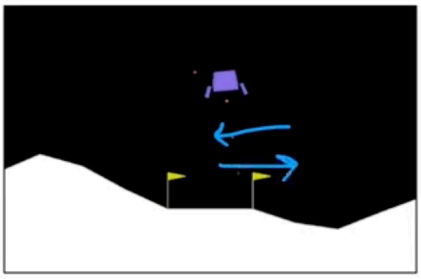
Learn a policy π that, given $s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$



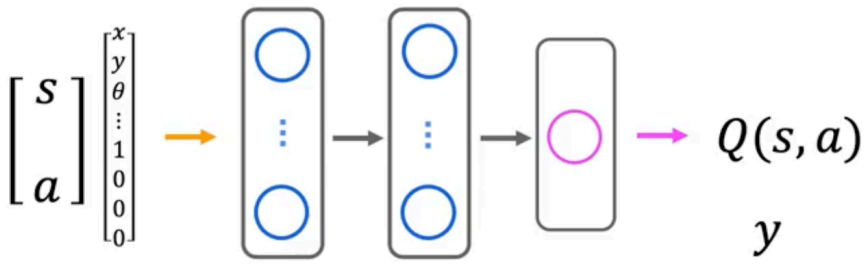
picks action $a = \pi(s)$ so as to maximize the return.

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



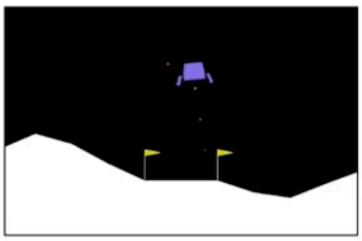
Deep Reinforcement Learning



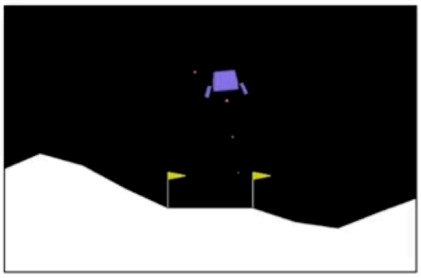
In a state s, use neural network to compute

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

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



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.

Train model:

Create training set of 10,000 examples using

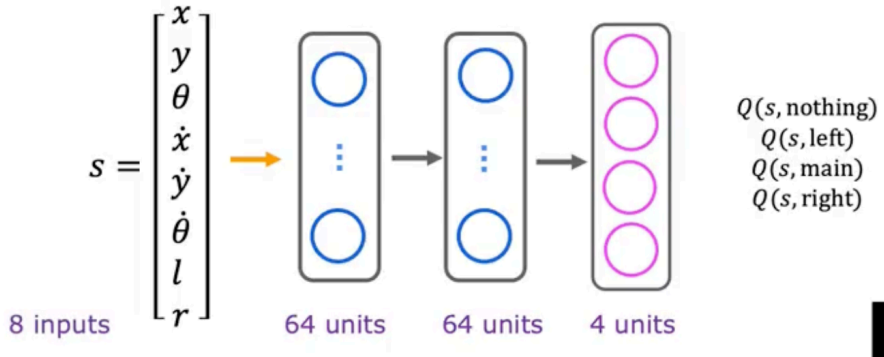
$x = (s,a)$ and $y = R(s) + \gamma \max_{a'} Q(s',a')$.

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

Set $Q = Q_{new}$.

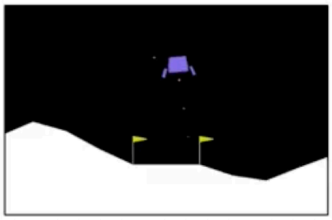
We can train the NN to output the 4 predictions of all 4 possible actions, instead of making one prediction per action.

Deep Reinforcement Learning



In a state s, input s to neural network.

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



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"

ε–greedy policy ($\varepsilon = 0.05$)

0.95

Q(s,main) is low

↑

a

Start ε high

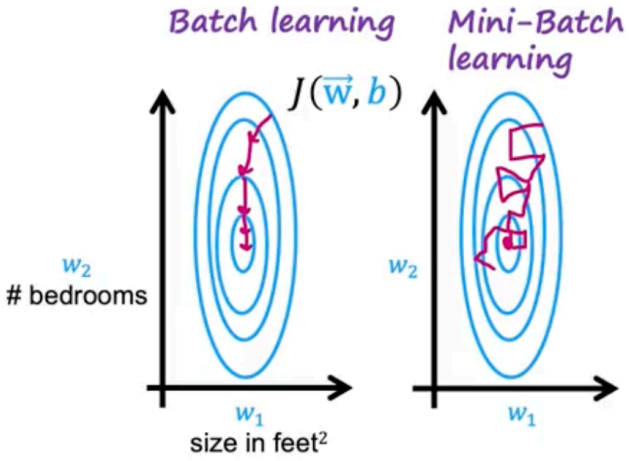
Gradually decrease

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.

Mini-batch

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



Every iteration runs much faster but may run not as smoothly towards the global minimum. If the dataset is super large, mini batch still runs faster overall.

Soft Update

Set $Q = Q_{new}$.

W, B $\quad$ W_{new}, B_{new}

$$W = 0.01\, W_{new} + 0.99\, W$$
$$B = 0.01\, B_{new} + 0.99\, B$$

Convergence is more reliable with soft updates.