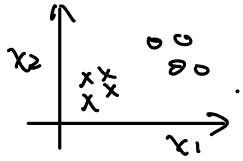
Unsupervised Learning. Recomender System & Reinforcement Learning.

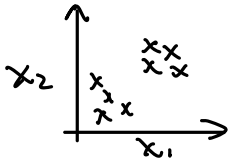
Clustering.

Supervised learning.



$(x^{(1)}, y)$. $(x^{(2)}, y)$.

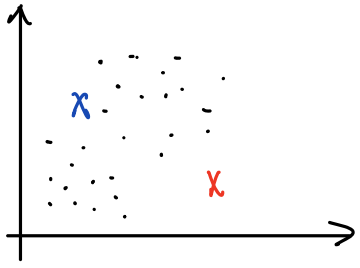
But now. we don't have the labels.



Training Set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$.

- Application of clustering.
  - Analyze DNA seq.
  - Astronomical data analysis.

K-means Intuition.



We have 30 points.

Random initial guess of two centers.

Decide. which centroid every point is closer to.

Look at all the points that belongs to one centroid.

Compute the mean to be the new centroid.

Now we repeat & compute.

K-means Algorithm.

- Randomly initialize K cluster centroid. $\mu_1, \mu_2, \cdots, \mu_K$.

  μ_i has the same dim as your training points.

- Repeat {
  - for i = 1 to m. # Assign points to cluster centroid.
    - $c^{(i)}$:= index (from 1 to k) of cluster centroid closest to $x^{(i)}$

↖ $\min_k \|x^{(i)} - \mu_k\|^2$.

- for k = 1 to K. # Move cluster to centroid.
  - μ_k := average (mean) of points assigned to cluster k.

}.

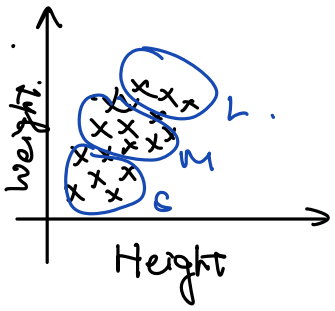
Notice. if sometimes no points assign to a centroid. we can.

① Directly eliminate the centroid.

② Re-initialize the centroid.

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

e.g.



T-shirt S.M.L. K>3.

K-means just work fine.

Optimization Objective.

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

μ_k = cluster centroid k. (Current actual).

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

↓ (Current Computed).

- Cost function.

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

$$\min_{\substack{c^{(1)}, \dots, c^{(m)} \\ \mu_1, \dots, \mu_K}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$ # Distortion cost function.

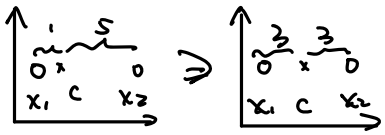
- Again. look at the alg.

In order to make $\|x^{(i)} - \mu_{c^{(i)}}\|^2$ smaller.

① Assign points $x^{(i)}$ to a closer centroid.

$1^2+2^2=26$ $2^2+3^2=18$.

③ Move centroid to the mean.



- Every step. the cost function should always go down. Never will go up.

Initializing K-means.

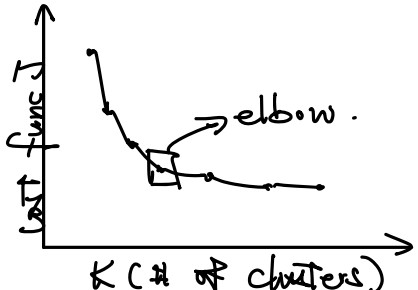
- Random Initialization.
- Choose $K < m$.
- Randomly pick K training examples. set $\mu_1, \cdots \mu_K$ equal to these examples

Notice. you might fell into local minimum. You may run k-mean multiple times. and compare the final cost func of them. Pick the smallest one.

- Algorithm.
  - For i = 1 to 100 {.
    - Randomly initialize K-means.
    - Run K means. Get $c^{(1)}, \cdots c^{(m)}$. $\mu_1 \cdots \mu_K$.
    - Compute cost function.
  - }
  - Pick set of clusters that gave lowest cost J.

Choosing the number of clusters.

- Elbow method.



Hardly ever use this. Because most of the time it decrease smoothly.

Notice: choose k minimize J is incorrect!

- Sometimes, you are running k-means to get clusters to use for some later / downstream purpose. Evaluate k-means based on a metric for how well it performs for that later purpose.

Anomaly Detection

Finding Unusual Events.

- Anomaly detection example:

  Aircraft engine features.

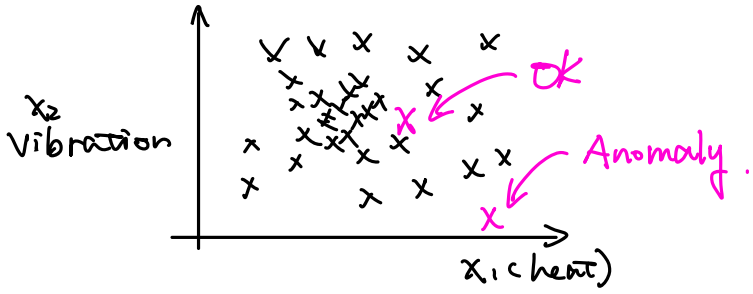
  x_1 = heat generated.

  x_2 = vibration intensity.

  ...

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

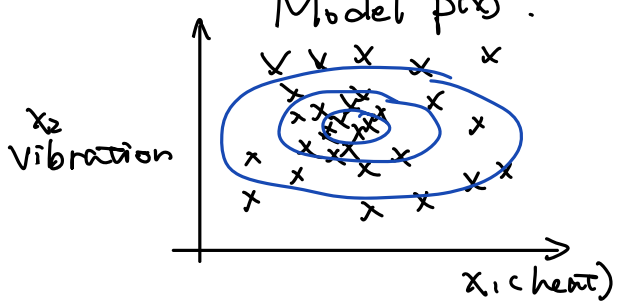
  New engine: x_{test}.



How we do this?

- Density estimation.

  Dataset: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$. Is x_{test} anomalous.



  Given some threshold ε.

  $p(x_{test}) < \varepsilon \rightarrow$ flag "anomaly".

  $p(x_{test}) \geq \varepsilon \rightarrow$ "OK".

- Fraud Detection. (example).

  $x^{(i)}$ = feature of user i's activity.

  (e.g. how often he/she access? Which web? ...).

Model $p(x)$ from data.

Identify unusual users by checking which have $p(x) < \varepsilon$.

- Monitoring computer in a data center.

  $x^{(i)}$ = feature of machine i.

  x_1: memory usage. x_2: number of disk accesses/sec.

  x_3: CPU load. x_4: CPU load/network traffic. ...

Gaussian (Normal) distribution.

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

- Parameter Estimation.

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

  $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$. $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2$

  Maximum likelihood estimation of normal distribution.

Anomaly detection algorithm.

- Density estimation.

  Training set: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$. each $x^{(i)}$ has n features.

  $p(x) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * \cdots * p(x_n; \mu_n, \sigma_n^2) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$

  In case to make this work. we need $x_1 \sim x_n$ to be mutually indep.

  But. even if they are not independent. we still get some results.

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

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

Vectorized:

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

- 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 $p(x) < \varepsilon$.

Developing and evaluating an anomaly detection system.

- The importance of real-number evaluation.
  - When developing a learning alg. (choosing features. etc.), making decisions is much easier if we have a way of evaluating our alg.
  - Assume we have some labeled data. of anomalous and non-anomalous examples. ($y=0$ if normal. $y=1$ if anomalous).
  - Training set: $x^{(1)}, x^{(2)}, \cdots, x^{(m)}$. (Assume normal examples)
  - Cross validation dataset. $(x_{cv}^{(1)}, y_{cv}^{(1)}), \cdots, (x_{cv}^{(m)}, y_{cv}^{(m)})$ } Include a few anomalous examples.
  - Test set. $(x_{test}^{(1)}, y_{test}^{(1)}), \cdots, (x_{test}^{(m)}, y_{test}^{(m)})$.

- Aircraft engines monitoring examples.

  {10000 good (normal) engines.
  20 flawed engines

  Training set: 6000 good engines. ← Train algorithm
  CV: 2000 good engines. 10 anomalous. ← Tune ε so that 10 anomalous are detect as "anomalous".
  Test: 2000 good engines. 10 anomalous ← Test to get the error rate.

- Alternative.

  Training set: 6000 good (normal) engines.
  CV: Rest. * We will use this if we only have really small amount

of anomalous data. E.g. if we only have 4 instead of 20.

- Algorithm evaluation.

  Fit model $p(x)$ on training set: $\{x^{(1)}, \dots, x^{(m)}\}$.

  On a cross validation/test example: predict:

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

- Possible evaluation metrics:
  - True pos, false pos, false neg, true neg.
  - Precision & Recall.
  - F_1-score.
- Can also use cross validation dataset to choose parameter ε.

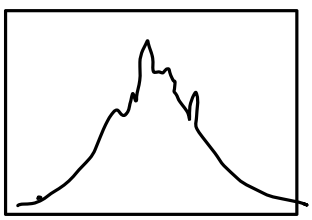
Anomaly detection vs. Supervised learning.

Anomaly detection.	vs.	Supervised Learning.
• Very small number of pos e.g. (0-20 is common) & large number of neg e.g.		• Large number of pos & neg examples.
• Many diff. "types" of anomalies Hard for any alg. to learn from pos e.g. what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.		• Enough pos e.g. for alg to get a sense of what pos eg are like, future pos e.g likely to be similar to ones in the training set.
* We build distribution, detect things deviate from normal.		* Try to learn how anomaly looks like and detect that pattern.

Anomaly detection.	vs.	Supervised Learning.
• Fraud detection • Manufacturing - Finding new previously unseen defects in manufacturing (e.g. aircraft engine) • Monitoring machines in a data center. ⋮		• Email spam detection. • Manufacturing - Finding known, previous seen defects. • Weather prediction. • Diseases classification. ⋮

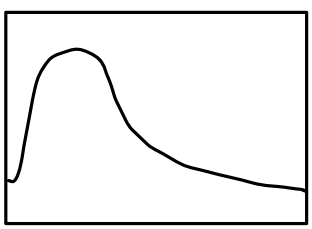
Choosing what features to use.

- Non-gaussian feature. Use plt.hist.



looks like gaussian.

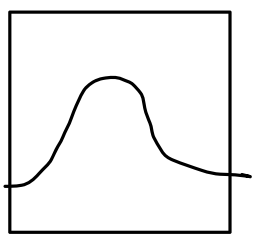
x



x

Maybe you want to transform this to make it "gaussian".

$\log(x)$ →



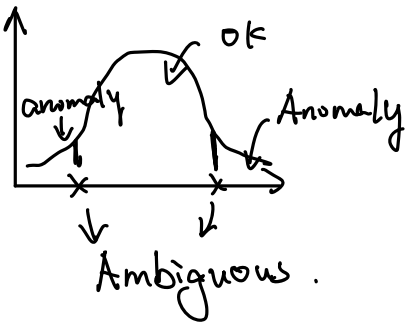
Examples of transformation:

$\log(x)$. $\log(x+1)$ $\log(x+c)$. $x^{\frac{1}{2}}$. $x^{\frac{1}{3}}$.

- Error Analysis for anomaly detection.
- What $p(x) \geq \varepsilon$ for normal examples x.

  $p(x) < \varepsilon$ small for anomalous examples x.
- Most common problem.

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



Maybe include more features to check if it is anomalous or not.

- Monitoring computers in a dataset center.

  x_1 = memory usage.

  x_2 = number of disk access/sec

  x_3 = CPU load.

  x_4 = network traffic.

We can further create features like:

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

Recommender System

Making recommendations

- Predicting movie ratings.

  User gave rating range from 0 to 5 stars.

Movie.	Alice	Bob	Carol.	Dave.
Love at last.	5	5	0	0
Romance forever	5	?	?	0
Cute puppie of love	?	4	0	?
Nonstop car chase	0	0	5	4
Swords vs. karate.	0	0	5	?

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

- Notations:

n_u = no. of users. n_m = number 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 $r(i,j)=1$).

Using per-item features.

- Same example. What if we have features of the movies?

Movie	Alice	Bob	Carol	Dave	x_1 Romance	x_2 Action
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppie of love	?	4	0	?	0.99	0
Nonstop car chase	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

- We use $n=2$ to denote the number of features.

Just linear reg.

- For user 1. Alice: Predicting rating for movie i as $w \cdot x^{(i)} + b$.

$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$. $b^{(1)} = 0$. $x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$. $w^{(1)} \cdot x^{(3)} + b^{(1)} = 4.95$.

For user j. Predict user j's rating for movie i as. $w^{(j)} x^{(i)} + b^{(j)}$.

- Cost function. New notation. $m^{(j)}$ = no. of movies rated by user j.

Objective: To learn $w^{(j)}, b^{(j)}$.

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

- To learn params: $w^{(1)}, b^{(1)}, \dots, w^{(n_u)}, b^{(n_u)}$ for all users.

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

Collaborative filtering algorithm.

Movie	Alice	Bob	Carol	Dave	x_1 Romance	x_2 Action
Love at last	5	5	0	0		
Romance forever	5	?	?	0		
Cute puppie of love	?	4	0	?	?	
Nonstop car chase	0	0	5	4		
Swords vs. karate	0	0	5	?		

- Say we already learned some params $w^{(1)} = w^{(2)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$. $w^{(3)} = w^{(4)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$.
  $b^{(1)} = b^{(2)} = b^{(3)} = b^{(4)} = 0$.

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

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

  $\Rightarrow x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ So we can approximate this feature vec.

  And we can do this for all other $x^{(i)}$.

- Cost function.

- Given $w^{(1)}, b^{(1)}, \cdots, w^{(n_u)}, b^{(n_u)}$, to learn $x^{(i)}$.

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

- To learn $x^{(1)}, \cdots, x^{(n_m)}$.

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

- Collaborative filtering. We combine two cost func together!

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

- Gradient Descent.

  repeat {

$$w_i^{(j)} = w_i^{(j)} - \alpha \cdot \frac{\partial}{\partial w_i^{(j)}} J(w.b.x).$$

$$b^{(j)} = b^{(j)} - \alpha \cdot \frac{\partial}{\partial b^{(j)}} J(w.b.x).$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w.b.x).$$

}.

Parameters: w.b.x.

Binary labels: favs, likes and clicks.

Binary labels.

Movie.	Alice	Bob	Carol.	Dave
Love at last.	1	1	0	0
Romance forever	1	?	?	0
Cute puppie of love	?	1	0	?
Nonstop car chase	0	0	1	1
Swords vs. karate.	0	0	1	?

1: engaged after being shown item.

0: did not engage after being shown item.

?: item not yet shown

- Example of applications.
  - Did user j purchase an item after being shown?
  - Did user j fav/like an item?
  - Did user j spend at least 30 sec. with an item?
  - Did user j clicked on a item?

- From reg. to binary classification.
- Previously.
  - Predict $y^{(i,j)}$ as $w^{(j)}x^{(i)} + b^{(j)}$.
- for binary labels:

- Predict that the probability of $y^{(i,j)}=1$. is given by $g(w^{(j)}x^{(i)}+b^{(j)})$ where $g(z)=\frac{1}{1+e^{-z}}$

- Cost function for binary application.
- Previously.

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

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

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

← loss of single eg

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

← Cost of all examples

Recommender System Implementation.

Mean normalization.

Movie.	Alice	Bob	Carol.	Dave	Eve
Love at last.	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppie of love	?	4	0	?	?
Nonstop car chase	0	0	5	4	?
Swords vs. karate.	0	0	5	?	?

$w^{(5)}=\begin{bmatrix}0\\0\end{bmatrix}$ So the system will set the rating to 0. Since she haven't

$b^{(5)}=0$. rate any movie.

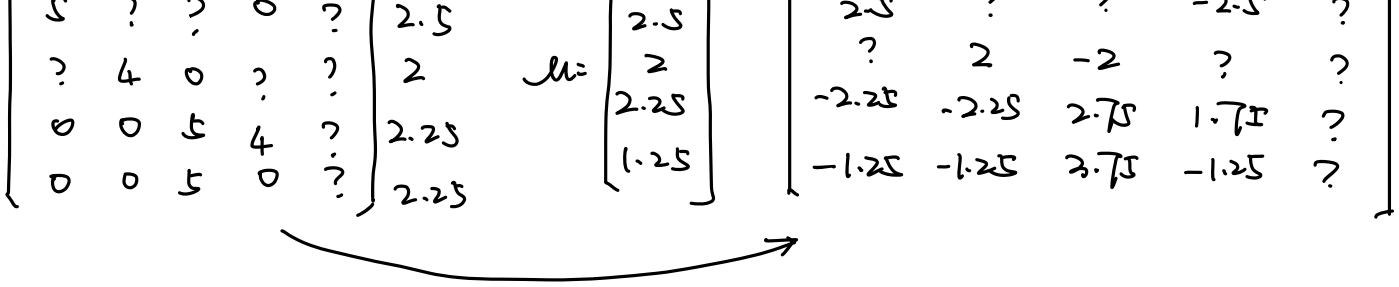
- Mean normalization.

$$\begin{bmatrix}5&5&0&0&?\\5&?&?&0&?\\?&4&0&?&?\\0&0&5&4&?\\0&0&5&?&?\end{bmatrix}\begin{matrix}2.5\\2.5\\2\\2\\2.5\end{matrix}$$

Minus

$$\begin{bmatrix}2.5\\2.5\\2\\2\\2.5\end{bmatrix}$$

$$\begin{bmatrix}2.5&2.5&-2.5&-2.5&?\\2.5&?&?&-2.5&?\\?&2&-2&?&?\\-2&-2&3&2&?\\-2.5&-2.5&2.5&?&?\end{bmatrix}$$



- For user j on movie i predict:

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

- User 5 (Eve).

$w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. $b^{(5)} = 0$. $w^{(5)} x^{(i)} + b^{(5)} + \mu_i = 2.5$. This is more reasonable. Compare to all rate 0.

- Mean normalization make your runtime faster and more reasonable.

Finding related items.

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

To find other items related to it.

i.e. with smallest dist. find item k with $x^{(k)}$ similar to $x^{(i)}$.

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

- Limitations of Collaborative filtering.
- Cold start problem. How to.
  - → rank new items 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 preference

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.
  - → Country
  - → Movie watched.
  - → Average rating per genre.
  - → ...

  } $x_u^{(j)}$ for user j.

  ↓

  Vector size can be diff.

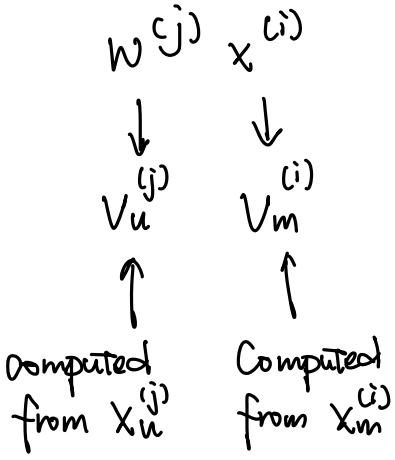
  ↑

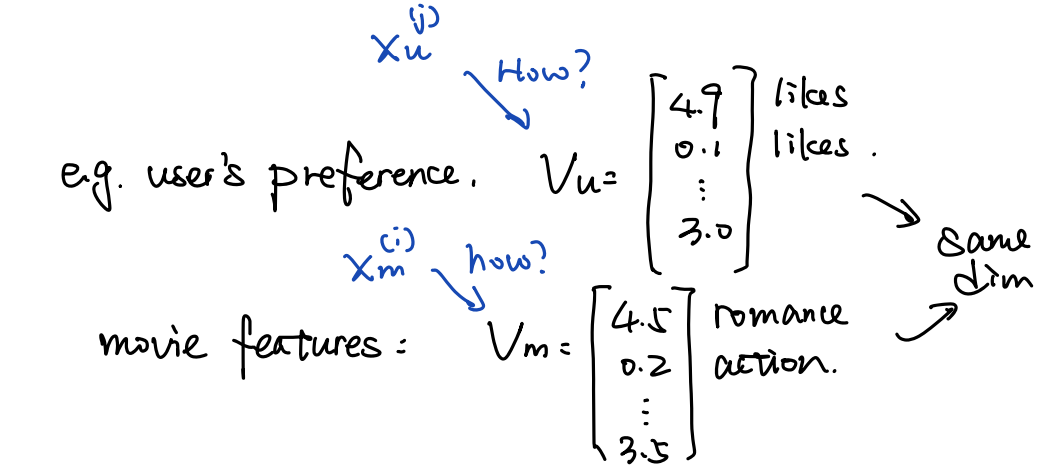
- Movie features.
  - → Year
  - → Genre
  - → Reviews.
  - → Average rating.
  - → ...

  } $x_m^{(i)}$ for movie i.

- Learn to match.

  Predict rating of user j on movie i as:



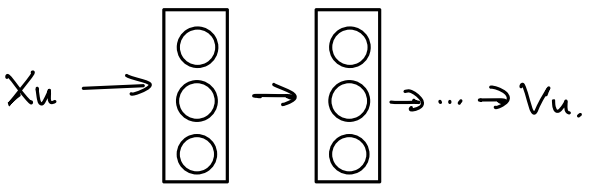


And the dot product give us the "matching".

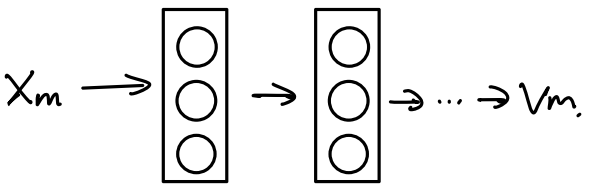
Deep learning for Content-based filtering.

- Neural network architecture.

$x_u \to v_u$. User net.



$x_m \to v_m$. Movie net.



Prediction: $v_u \cdot v_m$

We can futher use sigmoid($v_u \cdot v_m$) to predict the prob $y^{(i,j)} = 1$.

- Cost function.

$$J = \sum_{(i,j):r(i,j)=1} (v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2 + NN \text{ regularization items.}$$

- 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.

Notice: This can be pre-computed ahead of time.

Recommending from a large catalogue.

- Today, we have millions of movies, songs, Ads to recommend to user. Directly feed them into the network is computationally costly.
- Two Steps: Retrieval & Ranking.
- Retrieval:
  - Generate large list of plausible item candidate.
    - e.g.
      ① For each of the last 10 movies watched by the user, find 10 most similar movies.
      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.

Notice: Retrival step can be pre-computed. So this step can be quick

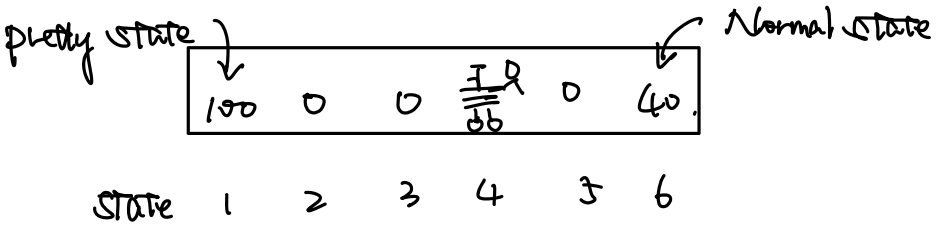
- Ranking.
  - Take list retrieved and rank using learned model.
  - Display ranked items to user.
- Retrival Step.
- Retrieving more items results in better performance, but slower recommed.
- To analyze / optimize the trade-off, carry-out offline experiments to see if retrieving additional items results in more relavent recommendations (i.e. $p(y^{(i,j)})=1$ of items displayed to user are higher)

Reinforcement learning introduction.

What is reinforcement learning.

- Example: Autonomous helicopter.
  - position of the helicopter. ⟶ how to move control sticks.
  - State s ⟶ Action a.
- Reward function.
  - Pos reward: helicopter flying well +1.
  - Neg reward: helicopter flying bad. / Even crash.
- Applications:
  - Controlling Robots.
  - Factory optimization.
  - Financial (stock) trading.
  - Playing games (including video games).

Mars rover example.



- Reward of 1 is 100. 6 is 40. Others are 0. How will robot move?

State: 4 3 2 1 4 5 6

0 0 0 100. 0 0 40.

But the robot might change mind. 4 → 4 → 4. Reward 0.

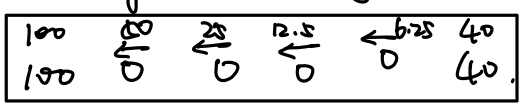
- (s, a, R(s), s'): State s. Take action a. reward at s. new state s'.

The return in reinforcement learning.

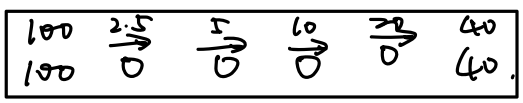
- keep going left: reward 0. 0. 0. 100.
- Return is defined if: return = $0 + 0.9 \cdot 0 + (0.9)^2 \cdot 0 + (0.9)^3 \cdot 100 = 72.9$.

  where 0.9 is called the discount factor γ. Often $\gamma = 0.9, 0.99, 0.999$.

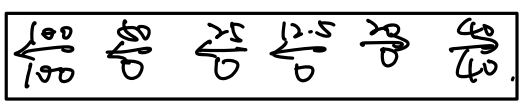
- At any state we go left.



- At any state we go right.

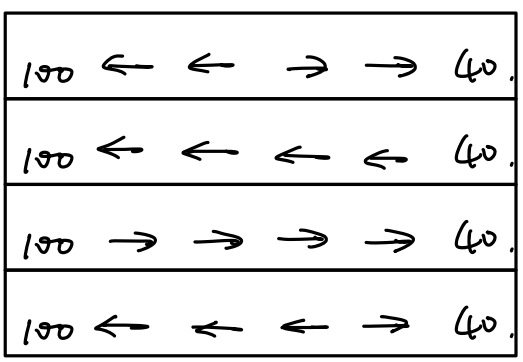


- We go left. at 1.2.3.4. Go right at 5.



Making decisions: Policies in reinforcement learning.

- Policy.



Policy

state → action

s π a.

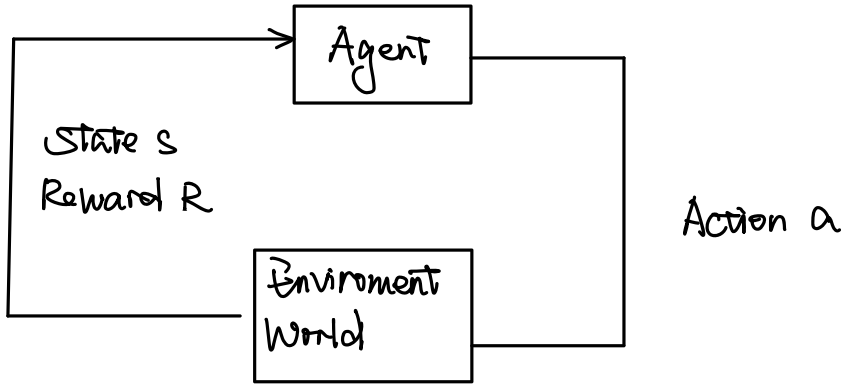
- A policy is a function $\pi(s) = a$ mapping from states to actions. that tells you what action a to take in a given state s.

- 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 maximum the return.

Review of Concepts.

	Mars rover.	Helicopter.	Chess.
States	6 states	pos of helicopter	Pieces on board
actions	← →	how to move control stick.	Possible moves
rewards	100. 0. 40.	+1. -1000	+1. 0. -1
discount factory 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.	100 ← ← ← → 40	Find $\pi(s) = a$.	Find $\pi(s) = a$.

- Markov decision process. (MDP). The future only depend on current state



State-action value function

State-action value function definition. Q-function.

- $Q(s, a)$ = Return if you.
  - Start in state s.
  - Take action a (once).
  - the behave optimal after that.

100	50	25	12.5	20
100	←	←	←	→ 40

This is the optimal result with $\gamma = 0.5$.

- e.g. $Q(2, \rightarrow)$. We know in state 3. the optimal action is ←. And again in state 2. the optimal is ←.

  So $Q(2, \rightarrow) = 0 + (0.5) \cdot 0 + (0.5)^2 \cdot 0 + (0.5)^3 \cdot 100 = 12.5$.
- e.g. $Q(2, \leftarrow) = 0 + 0.5 \cdot 100 = 50$.
- e.g. $Q(4, \leftarrow) = 0 + 0.5 \cdot 0 + 0.5^2 \cdot 0 + 0.5^3 \cdot 100 = 12.5$.

In the end. left is $Q(i, \leftarrow)$. Right is $Q(i, \rightarrow)$.

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

Value of $Q(s, a)$ for all s and a.

- Notice: for each state. the largest possible return is the largest $Q(s, a)$ for state s and all actions.
- The best possible action in state s is the action a gives $\max_a Q(s, a)$.

  Sometimes we use Q^* for $\max_a Q(s, a)$.
- Value change.

  When gamma ↑. More patient. Looking forward to faraway rewards.

  ↓ less patient. Want just recent closer reward.

Bellman equation.

Recall the definition: $Q(s,a)$ = Return if you.

- Start in state s.
- Take action a (once).
- the behave optimal after that.

s: Current state. $R(s)$: reward of current state.

a: Current action.

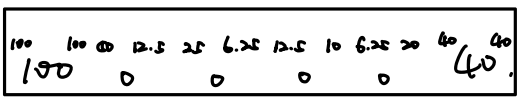
s': State you got to after taking action a.

a': action that you take in state s'.

- Bellman equation.

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

e.g.



If you are in terminal state. (e.g. 1-6)

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

- $Q(2,\rightarrow) = R(2) + 0.5 \cdot \max_{a'} Q(3,a')$.

$= 0 + 0.5 \cdot 25 = 12.5$.

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

$= 0 + 0.5 \cdot 25 = 12.5$.

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

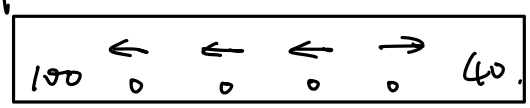
$$Q(s,a) = \underbrace{R(s)}_{\text{Reward you get right away.}} + \gamma \underbrace{\max_{a'} Q(s',a')}_{\text{Return from behaving optimally starting from } s'.}$$

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

$$= R_1 + \gamma \underbrace{[R_2 + \gamma R_3 + \gamma^2 R_4 + \cdots]}.$$

Random (Stochastic) environment.

- Say you rover at state 4. You command it to go left.
  Now it has 0.9 chance to go left and 0.1 chance to go right.
- Expected Return.



Its possible that you command it to go left from 4. but it go right.
And keep going left. 0 0 0 0 0 100.
Now our goal is not the maximum return.

- Expected Return = Average($R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$)
  $= E[R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots]$
- The goal of reinforcement learning.
  Same thing but maximize the expected return.
- Bellman Equation. $Q(s,a) = R(s) + \gamma E[\max_{a'} Q(s', a')]$

Continuous State Spaces.

Examples of Continuous state applications.

- Discrete vs. Continuous States.

The state might not contain only one number. But e.g.

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

x, y, θ = The position and angle.

$\dot{x}, \dot{y}, \dot{\theta}$: The speed at x, y. direction. and angle speed.

eg. of helicopter.

$$s = [x, y, z, \phi, \theta, \omega, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\omega}]^T.$$

Lunar Lander.

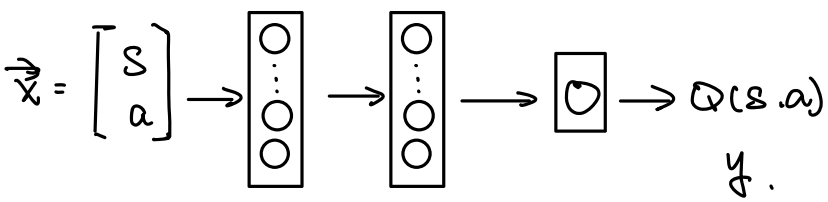
$$s = [x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}, l, r].$$

l, r: whether l/r leg is sitting on ground.

- Reward function:
  - Getting to landing pad. 100 ~ 140.
  - Additional reward for moving toward/away from landing.
  - Crash: -100.
  - Soft landing: +100.
  - leg grounded: +10.
  - Fire main engine: -0.3. Fire side thruster: -0.03.
- Learn a policy π that, given $s = \cdots$, picks action $a = \pi(s)$ so as to maximum the return. $\gamma = 0.985$.

Learning the State-value function.

- Deep reinforcement learning.



12 input is $\begin{bmatrix} s \\ a \end{bmatrix} = [x, y, \theta, \cdots, 1, 0, 0]$

Using 0 and 1 to encode the action.

- 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 $Q(s, a)$.

- Bellman equation: (how to get a training set for the network?).

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

Train the network to learn $f_{w,b}(x) \approx y$.

Using the lunar lander, just trying out a series of things.

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

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

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

$\vdots$

$(s^{(10000)}, a^{(10000)}, R(s^{(10000)}), s'^{(10000)})$.

- Notice. $s^{(i)}, a^{(i)}$ will be used for $x = [s, a]$.

$R(s^{(i)}), s'^{(i)}$ will be used for y.

But we may don't know how to compute $\max_{a'} Q(s'^{(i)}, a')$

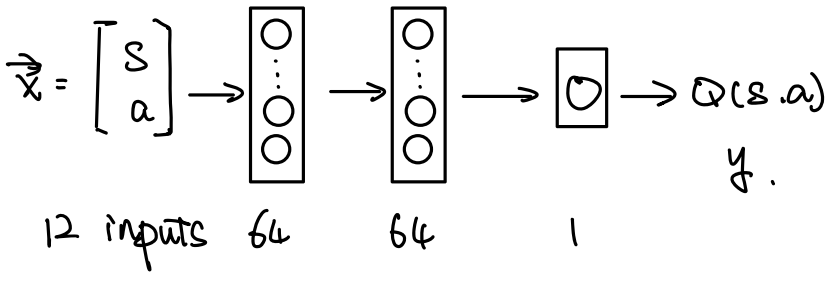
We can take a random guess of the Q function at start.

- Learning Algorithm.
  - Initialize NN 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:
      - Creating training set of 10,000 examples using.

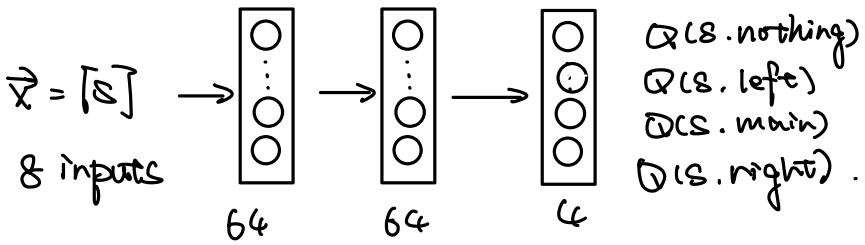
$x = (s, a)$ and $y = R(s) + \gamma \max_{a'} Q(s', a')$

- Train Q_{new} s.t. $Q_{new}(s, a) \approx y$.
- Set $Q = Q_{new}$.

Alg. refinement. Improved NN Architecture.



- Inefficient because we need to train for 4 diff. actions separately.
- New architecture:



- In a state s, input s to NN.
  - Pick the action a that maximizes $Q(s, a)$. $R(s) + \gamma \max_{a'} Q(s', a')$.

Alg. refinement: ε-greedy policy.

- Recall the Learning Algorithm.
  - Initialize NN 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:
  - Creating training set of 10,000 examples using.

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

  - Train Q_{new} s.t. $Q_{new}(s, a) \approx y$.
  - Set $Q = Q_{new}$.

- How to choose actions while still learning?
- In some state s.
  - Option 1:

    Pick the action a that maximize $Q(s, a)$.
  - Option 2: (What commonly done). $\varepsilon = 0.95$. greedy policy — Exploitation

    With probability 0.95: pick the action a that max $Q(s, a)$.

    With probability 0.05: pick an action a randomly. ← Exploration

By random initialization, it might have some pre conceptions that certain actions is really bad. So it will never try that.

- Common practice. Start with ε high, even $\varepsilon = 1$. Decrease gradually.

Alg. refinement. Mini-batch & Soft updates.

- In previous learning alg. we take training set 10,000. Mini-batch this.
- Soft updates: Recall "Set $Q = Q_{new}$". You might be unlucky. overwrite the Q value with an even worse value.

Set Q = Q_{new}

W, B ↑ W_{new}, B_{new}

Less likely to diverge

We set $W = 0.01 W_{new} + 0.99 W$.

$B = 0.01 B_{new} + 0.99 B$.

} Set how aggressively you move from Q_{new} to Q. Gradual change!