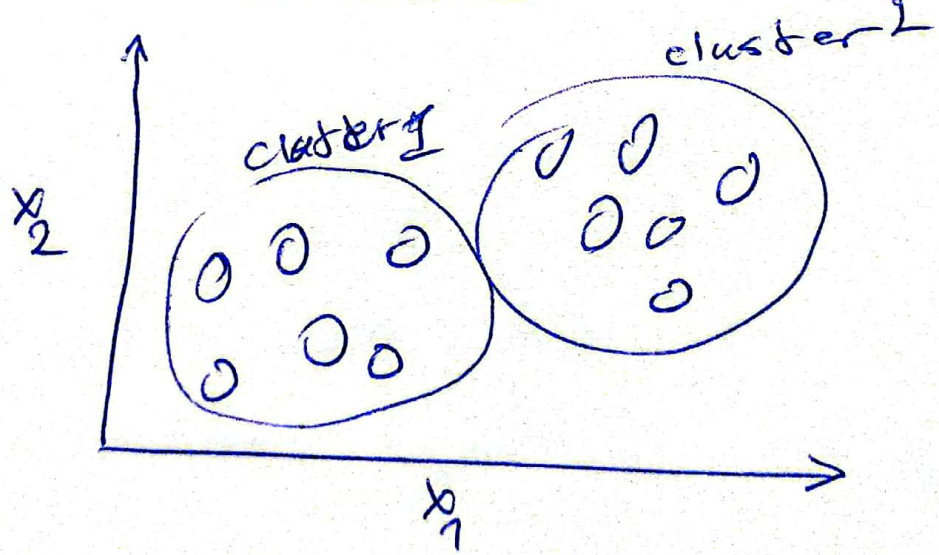
"unsupervised learning"

- unsupervised learning
  - clustering
  - Anatomy detection
- Recommender systems
- Reinforcement learning

clustering



Training set = $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

- there is no target (label) for features.

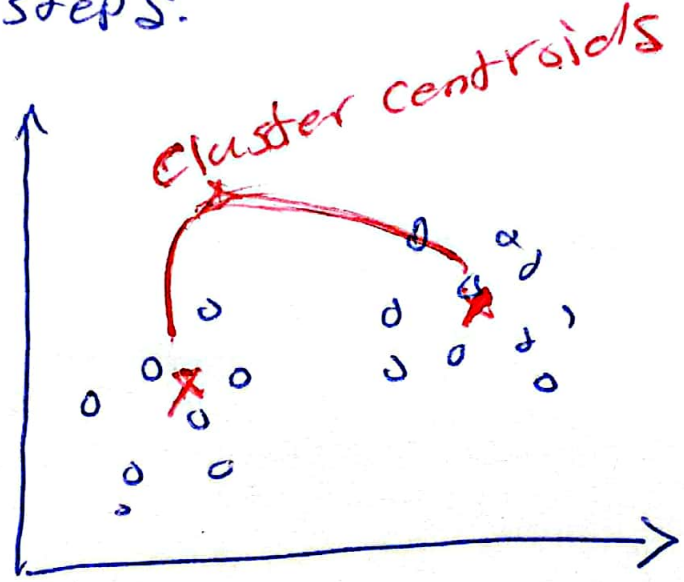
K-means algorithm (a clustering algorithm)

init : K-centroids (randomly choose)

Step 1: Assign each point to its closest centroid

Step 2: recompute the centroids.

repeat the steps.



steps in detail

1. Randomly initialize K cluster centroids

$\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_K$ (u: has the same dimensions like a training ~~set~~ each example)

2. Repeat {

   a. Assign points to cluster centroids

   for $i = 1$ to m

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

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

B. move cluster centroids

for $k = 1$ to K

$\mu_k :=$ average (mean) of points assigned to cluster k.

{

$\vec{\mu}_k = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)} \right]$

if during the process one cluster (centroid k, i) has no data point assign to it

one should eliminate that cluster centroid.

$K = K - 1$

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 (K-means) →

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

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

- K-means algorithm is trying to ~~find minimize~~ find assignments of points of ~~clusters~~ clusters centroid as well as find locations of clusters centroids that minimizes the squared distance.

- cost function of K-means called "Distortion function"

---

initializing K-means

K-means algorithm

step 0: randomly initialize K ~~means~~ clusters centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {
  step 1: Assign points to cluster centroids
  step 2: move cluster centroids
}

Step 0

Choose $K < m$ → number of training examples.

- randomly pick K training examples.
- set $\mu_1, \mu_2, \ldots, \mu_K$ equal to those K examples.

- ran the algorithm multiple times and each time with different set of randomly chosen cluster centroids is a good way to find a better local minimum for cost function J (Distortion function)

↓

for i = 1 to 100 { → (50 to 1000 times is common)

Randomly initialize K-means.

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

compute cost function (distortion)

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

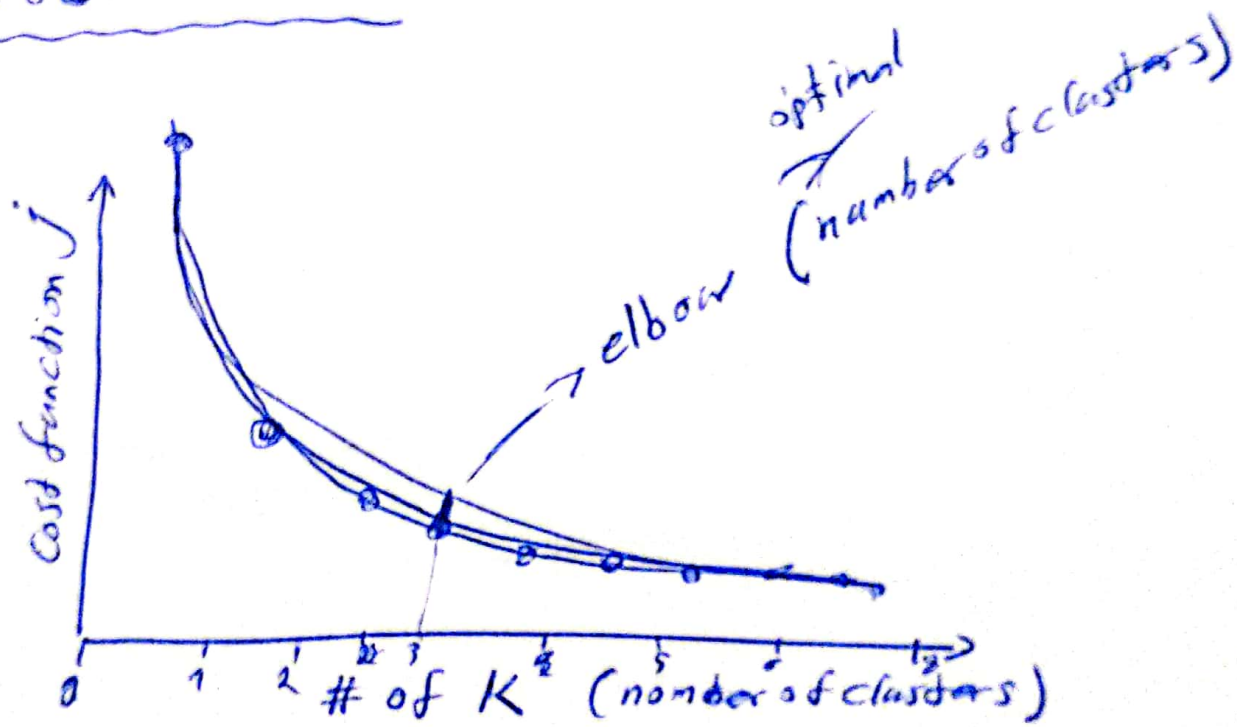
}

Pick set of clusters that gave lowest cost J

choosing the number of clusters (# K ?)

choosing the value of K

Elbow method



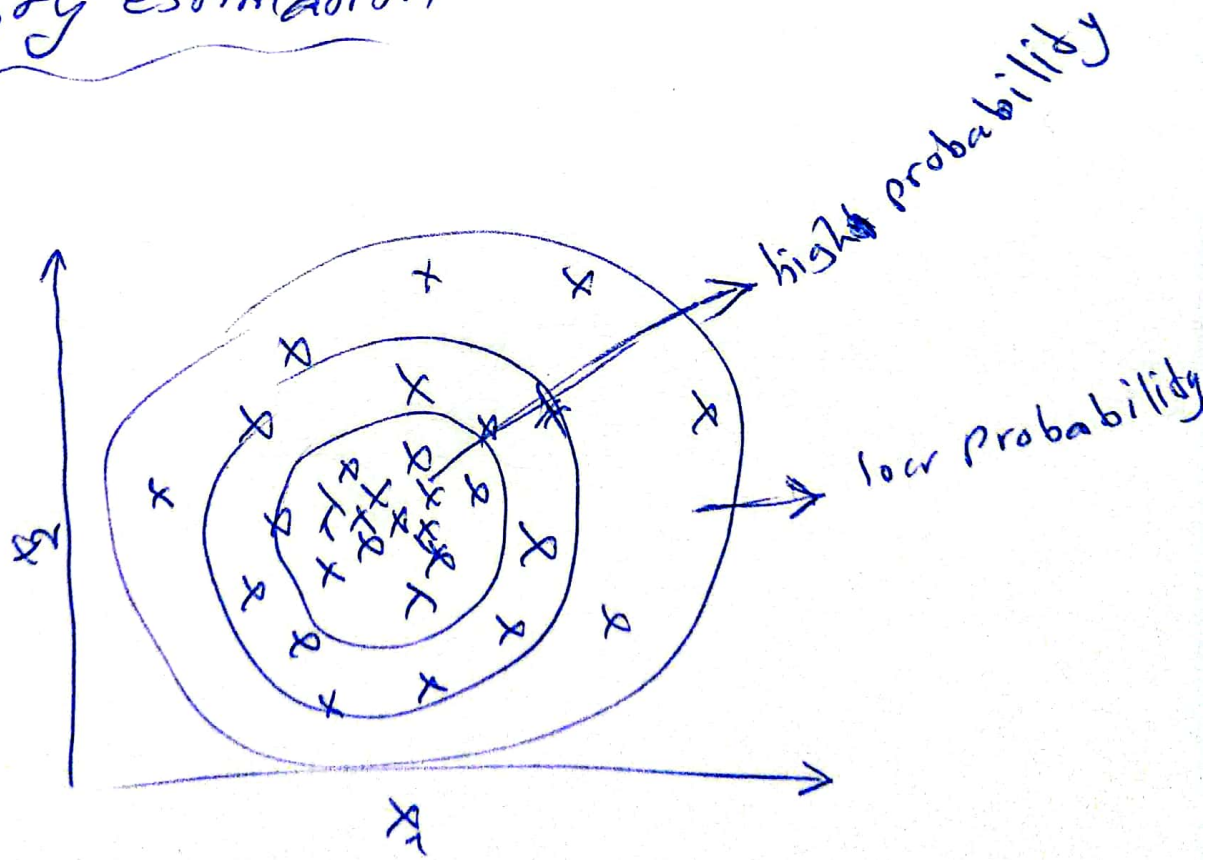
problem; more K (cluster centroids) leads to reduce cost function normally, so with elbow method in most cases number of clusters are more than necessary.

choosing the value of K

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

"Anomaly detection"

Density estimation



Dataset = $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

model $p(x)$ = probability of x being seen in dataset

epsilon

is x_{test} anomalous? yes if $p(x_{test}) < \varepsilon$

Anomaly detection example:

- fraud detection:

- manufacturing:

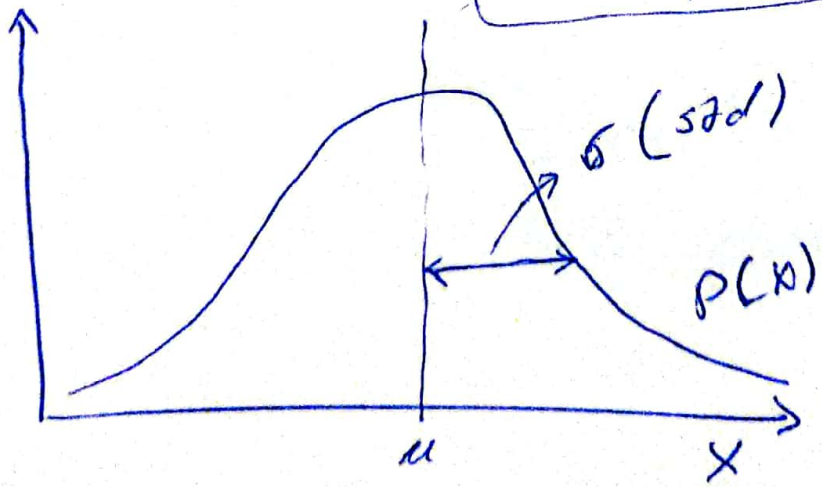
- monitoring computers in a data center.

Gaussian distribution

Say x is a number

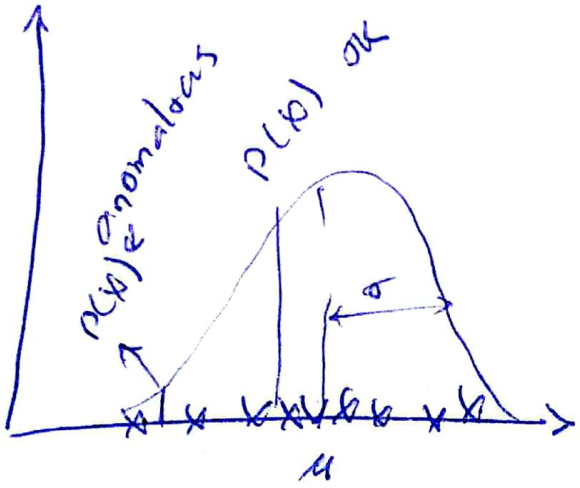
- probability of x is determined by a Gaussian with mean μ and variance σ^2.

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



Parameter estimation

Data set: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$



$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

maximum likelihood for μ and σ

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

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

note: here ~~x is yes~~ the data set has just one feature.

Anomaly Detection Algorithm.

Training set: $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(m)}\}$

(each example $\vec{x}^{(i)}$ has n features.)

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times p(x_3; \mu_3, \sigma_3^2) \quad (10)$$

$$\times \dots \times p(x_n; \mu_n, \sigma_n^2)$$

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

Algorithm

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

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

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

vectorized formula

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^{m} \vec{x}^{(i)}$$

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

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

Anomaly if $p(x) < \varepsilon$

"The importance of real-number evaluation"

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

Algorithm evaluation (anomaly detection)

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

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

- 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 detection

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

Large number of negative ($y=0$) examples.

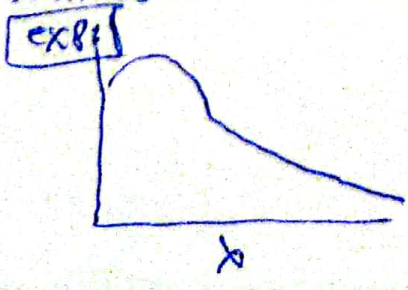
supervised learning

larger number of positive and negative examples.

choosing what features to use.

1. choose features which are normaly distributed
   if they are not normaly distributed try to transform them. ~~to transform~~ or $x = \frac{1}{x}$

ex1:

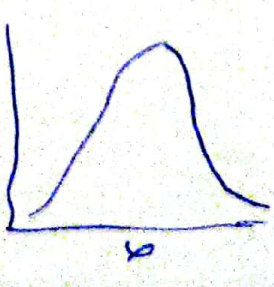


np.sqrt(x)

np.log(x)

or

np.log(x+1)



Error analysis for anomaly detection

want $\begin{cases} p(x) \geq \varepsilon \text{ for normal examples } x. \\ p(x) < \varepsilon \text{ for anomalous examples.} \end{cases}$

most common problem:

$p(x) \geq \varepsilon$ for both normal and anomalous examples.

"Recommender Systems"

an example: movie rating

$r(i,j) = 1$ if user j has rated movie i (0 otherwise)

$y(i,j)$ = rating given by user j on movie i (if defined)

$w^{(j)}, b^{(j)}$ = parameters for user j

$x^{(i)}$ = feature vector for movie i

for user j and movie i, predict rating: $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$m^{(j)}$ = number of movies rated by user j

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

cost function

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

cost function

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

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

(n_u: number of users)

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

"collaborative filtering algorithm"

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

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

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

→ minimizing the cost function as a function of $f(x^{(i)})$

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

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

→ $j(x^{(1)}, x^{(2)}, \ldots, x^{(n_m)})$

in collaborative filtering algorithm we can learn the features for a given ~~data~~ data set.

(in contrast to most machine learning algorithms, ~~which~~ in which the features have to be externally given)

collaborative filtering Algorithm

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

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

② cost function to learn $x^{(1)}, \ldots, x^{(n_m)}$:

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

put them together (① and ②):

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

overall cost function for learning Params and features in collaborative filtering algorithm.

Gradient descent for collaborative filtering

Linear regression (Collaborative filtering)

repeat {

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

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

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

}

Parameters: w, b, x

"Binary labels" (collaborative filtering)

From regression to binary classification:

regression: predict $y^{(i,j)}$ as $w^{(j)} \cdot x^{(i)} + b^{(j)}$

For binary labels:

predict the probability of $y^{(i,j)} = 1$

is given by $g(w^{(j)} \cdot x^{(i)} + b^{(j)})$ where $g(z) = \frac{1}{1+e^{-z}}$

in collaborative filtering algorithm we can learn the features for a given ~~data~~ data set.

(in contrast to most machine learning algorithms, ~~which~~ in which the features have to be externally given)

collaborative filtering Algorithm

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

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

② cost function to learn $x^{(1)}, \ldots, x^{(n_m)}$:

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

put them together (① and ②):

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

overall cost function for learning Params and features in collaborative filtering algorithm.

Gradient descent for collaborative filtering

Linear regression (Collaborative filtering)

repeat {

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

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

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

}

Parameters : w, b, x

"Binary labels" (collaborative filtering)

From regression to binary classification:

regression: predict $y^{(i,j)}$ as $w^{(j)} \cdot x^{(i)} + b^{(j)}$

For binary labels:

Predict 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 (collaborative filtering)

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

cost function:

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

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

---

mean normalization

movies

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & 2 & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 0 & 3 \end{bmatrix} \begin{matrix} 2.5 \\ 2.5 \\ 2 \\ 1.25 \end{matrix} \ominus \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \end{bmatrix} = \begin{bmatrix} 2.5 & 2.5 & -2.5 & \_ \\ \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ \end{bmatrix}$$

user

$y^{(i,j)}$

Tensorflow implementation of collaborative filtering

Custom training loop

$$J = (\underbrace{wx}_{f(x)} - \underbrace{1}_{y})^2$$

Gradient descent algorithm

$$w = w - \alpha \frac{d}{dw} J(w,b)$$

fix $b = 0$ for this example, initialize w with 3

using tensorflow

```
w = tf.Variable(3.0)
x = 1.0
y = 1.0 # target value

alpha = 0.07

iterations = 30

for iter in range(iterations):
    # Use Tensorflow's gradient tape to record the steps.
    # used to compute the cost j, to enable auto differentiation.
    with tf.GradientTape() as tape:

        fwb = w * x

        costj = (fwb - y)**2
    # use the Gradient tape to calculate the gradients of the
    # cost with respect to the Parameter w.
    [djdw] = tape.gradient(costj, [w])
```

```
# run one step of GD by updating the value of w to
# to reduce the costs
w.assign_add(-alpha * dJdw)
```

Implementation in Tensorflow (collaborative filtering)

```
optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iters = 200
for iter in range(iters):
    with tf.GradientTape() as tape:
        cost_value = cofiCostFunc(X, W, b, Ynorm, R, num_users,
                                  num_movies, lambda)

    grads = tape.gradient(cost_value, [X, W, b])

    optimizer.apply_gradients(zip(grads, [X, W, b]))
```

limitations of Collaborative Filtering

- Cold start problem: How do
  - rank new items that few users have rated?
  - show something reasonable to new users who have rated few items?
- use side information about items or users:
  - items: Genre, movie stars, studio, ...

- users Demographics (age, gender, location), expressed preferences, ...

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
- movies watched
- Average rating per genre

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

movie features:
- Year
- Genre/Genres
- Reviews
- Average rating

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

Predict rating of user j on movie i as

~~in content~~

collaborative filtering

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

$v_u^{(j)} \cdot v_m^{(i)}$ ← content based filtering

computed from $x_u^{(j)}$ — computed from $x_m^{(i)}$

exp:

user's preferences $v_u^{(j)} = \begin{bmatrix} 0.9 \\ 0.7 \\ \vdots \\ 0 \end{bmatrix}$ Genres — how much user j likes each genre

both have to have the same dimention

movie features $v_m^{(i)} = \begin{bmatrix} 0.5 \\ 0.2 \\ \vdots \\ 3.5 \end{bmatrix}$ romance, action, ... — how much movie i is action or...

---

deep learning for content-based filtering

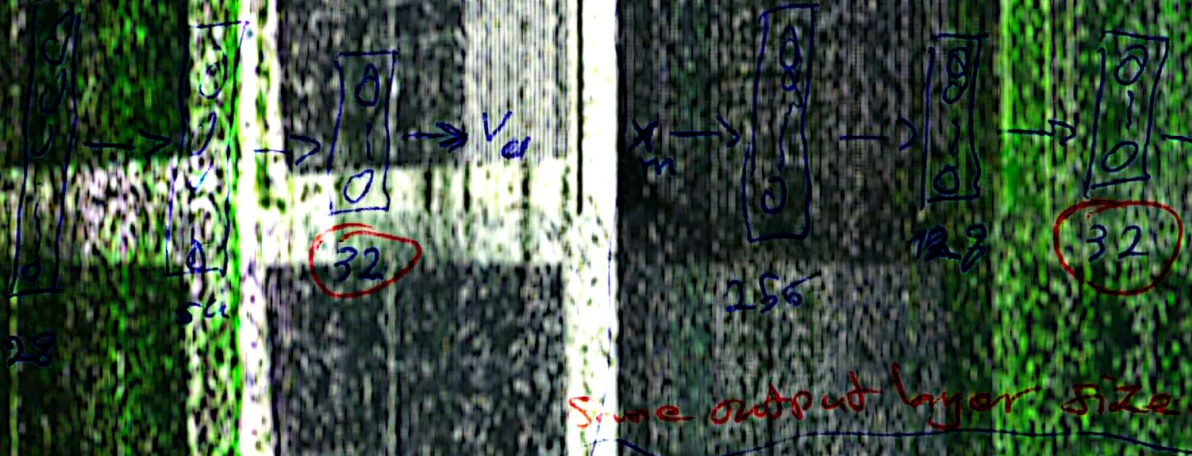
neural network architecture

$x_u \rightarrow v_u$ user network

$x_m \rightarrow v_m$ movie network

user network
$(x_u \rightarrow v_u)$

movie network
$(x_m \rightarrow v_m)$



Same output layer size.

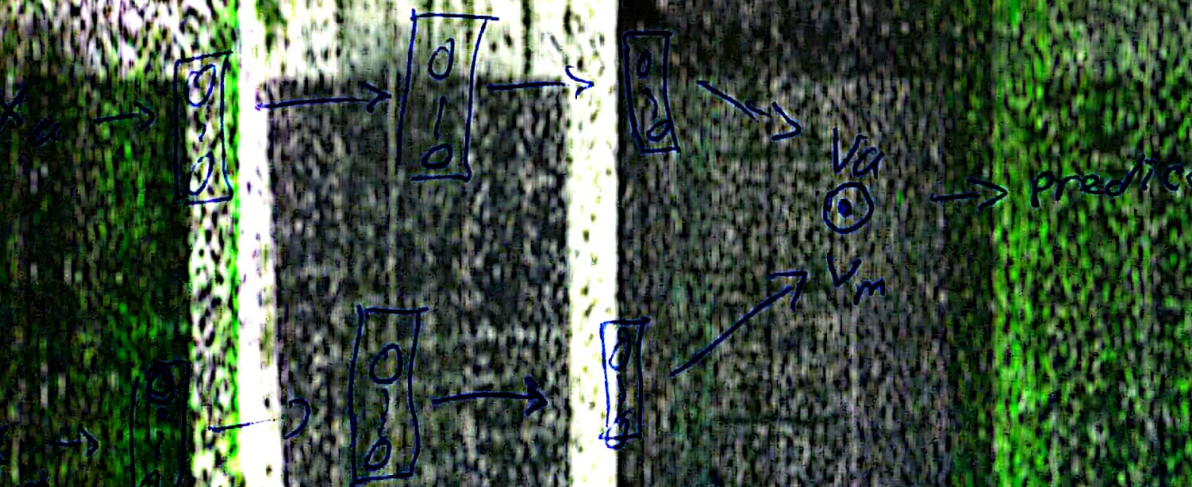
predict: $v_u^{(j)} \cdot v_m^{(j)}$

dot product

or

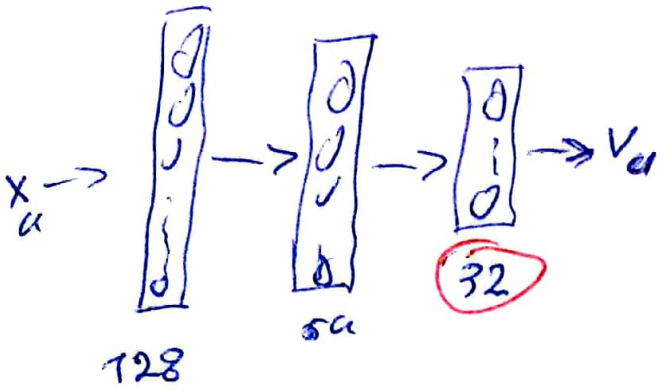
$g(v_u^{(j)} \cdot v_m^{(j)})$ to predict the probability that $y^{(ij)}$ is 1

neural network architecture



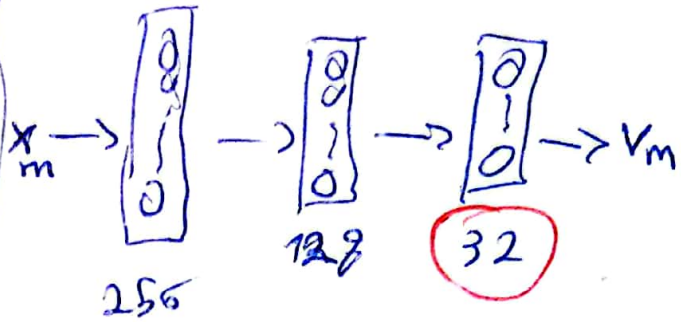
user network

$(x_u \rightarrow v_u)$



movie network

$(x_m \rightarrow v_m)$



Same output layer size.

Predict: $v_u^{(i)} \cdot v_m^{(j)}$

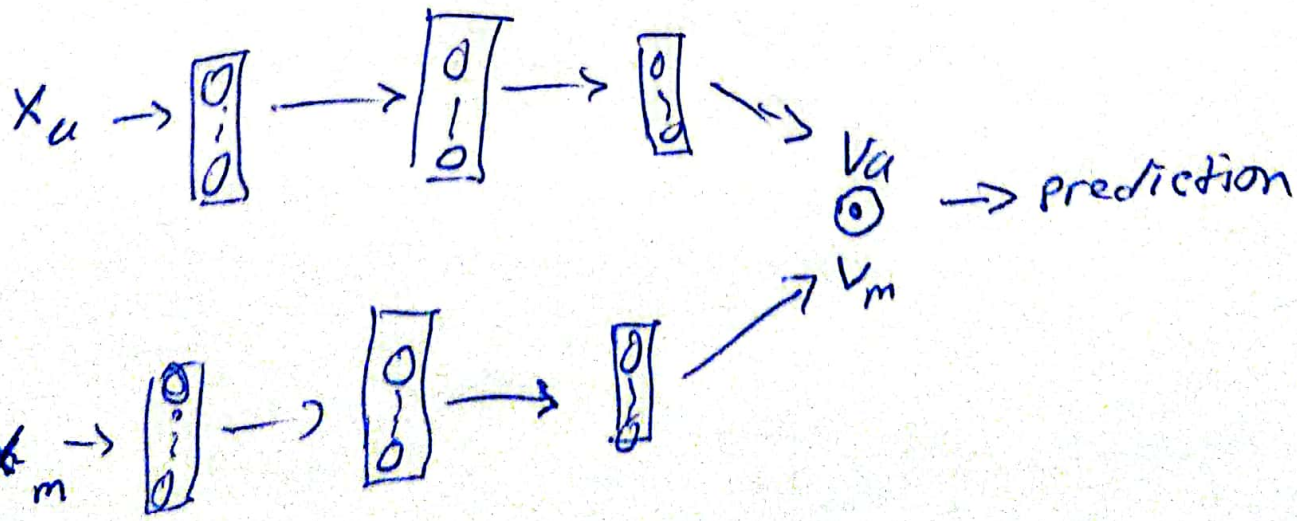
↓

dot product

or

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

neural network architecture



cost function for content-based filtering

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

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:

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

→ compute distance

Note: this can be pre-computed ahead of time.

How to ~~effective~~ efficiently find recommendation from a large set of items?

Two steps: Retrieval & Ranking

Retrieval:

- Generate large list of plausible item candidates

e.g. ~ 1000s

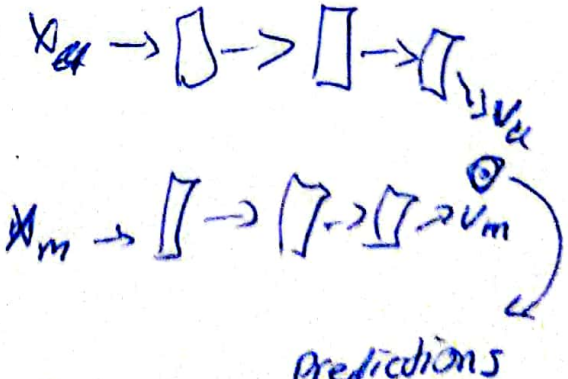
retrieved items:

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.

Ranking:

- take ~~list~~ retrieved list and rank using learned model



- display ranked items to user.

Tensor flow implementation of content-based filtering.

x_u → input

```
user_NN = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256, activation="relu"),
    "    "    "    "  (128, activation="relu"),
    "    "    "    "  (32)
```

]) → v_u → output

x_m → input

```
item_NN = tf.keras.models.Sequential([

    tf.keras.layers.Dense(256, activation="relu"),
    "    "    "    "  (128, activation="relu"),
    "    "    "    "  (32)
```

]) → v_m → output

```
# 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(axis=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()
```

Reducing the number of features (PCA)

Principal component analysis (an unsupervised learning algorithm to reduce the number of features)

PCA: finds new axis and coordinates

for example: a car dataset with two feature

x_1 = length of car
x_2 = height of car

] can be combined and make a feature (size). ← new

PAC: takes a data set with too many features (for example: 50) and compress them into just 2 features which one can visualize in a 2 dimensional graph plot and gains information about all the data set (features) in just a visualization.

PCA Algorithm

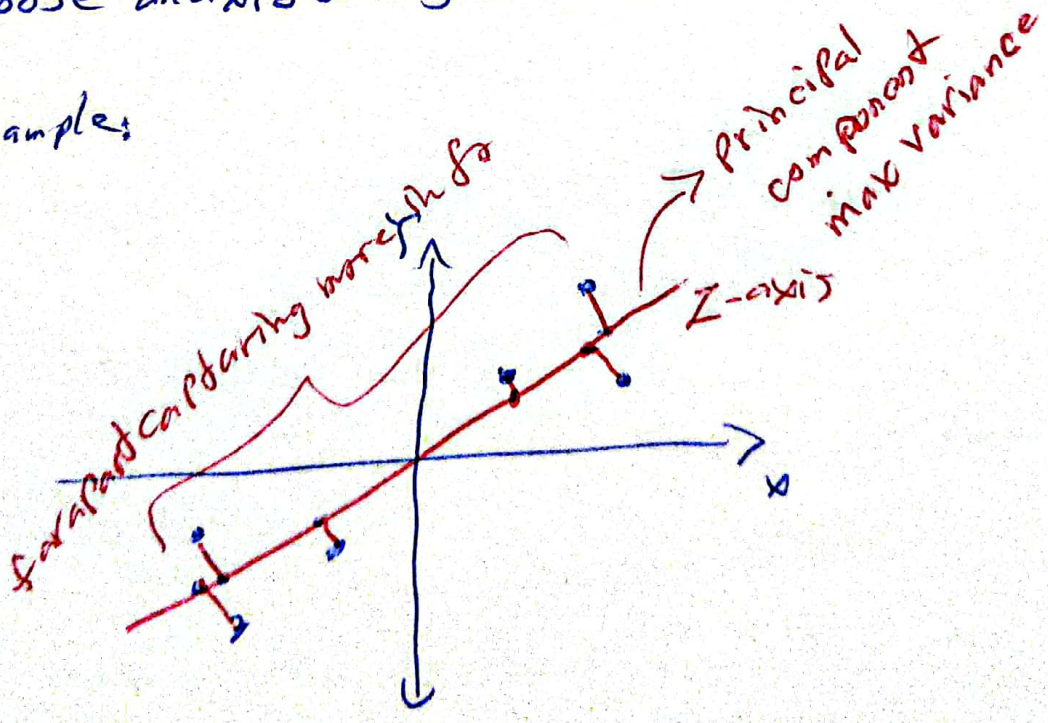
feature scaling

1. preprocessing: normalized the features to have the same mean (0) → Z-score normalization.

$\rightarrow (Z)$

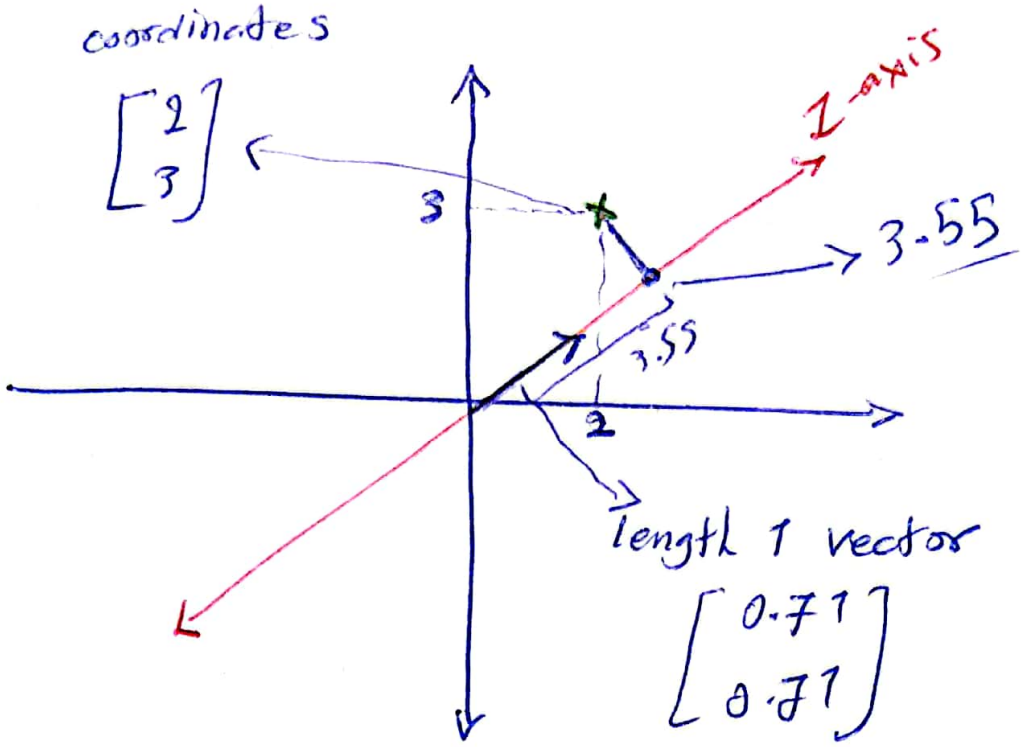
2. Choose an axis to project all points (values).

example:



coordinates

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



How to project a point X (with two dimensions in one dimension-Z)

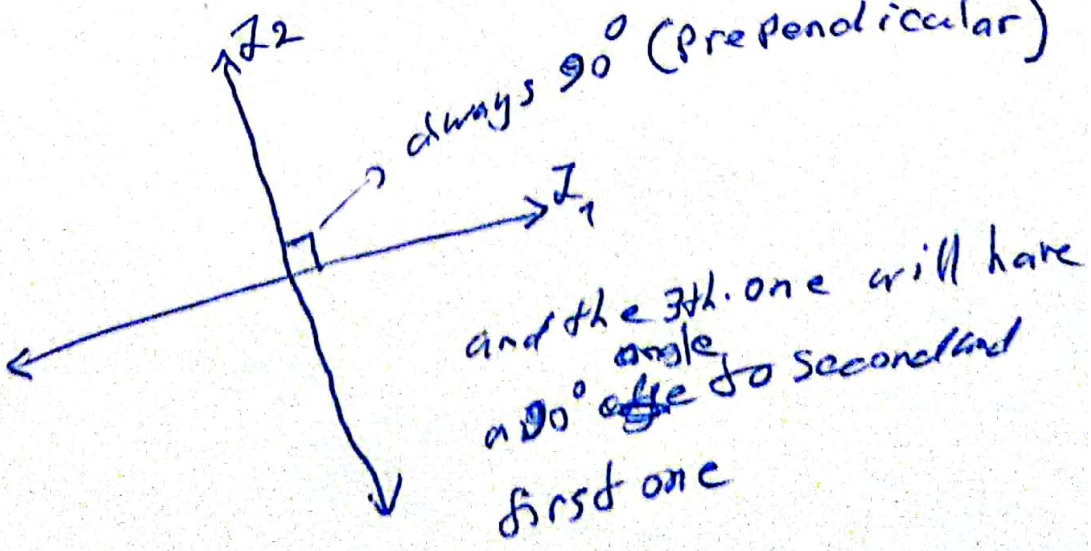
dot product

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix} = 2 \times 0.71 + 3 \times 0.71 = 3.55$$

projected point

---

more principal components



approximation to the original data

given Z_1: 3.55 —> a1

find original $(X_1, X_2) = ?$

"reconstruction" —> approximation

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

↙

approximate the original point

(PCA in ~~code~~ ~~sklearn~~) → scikit-learn

1. scale data: (feature scaling) Z-normalization

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

   ↳ performs

   ↳ fit contains mean normalization automatically, so on

   no need to normalize data beofr "fit"

3. optionally examine how much variance is explained by each principal component.

4. transform (project) the data onto the new axes. (30)

code

1. (optional) normalization:

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

2. "fit"

PCA_1 = PCA(n_components=1)

PCA_1.fit(X)

3. variance explained:

PCA_1.explained_variance_ratio_ (0.992)

↙

in this example

4. projection (transform):

X_trans_1 = PCA_1.transform(X)

X_reduced_1 = PCA_1.inverse_transform(X_trans_1)

(array([
[1, 3],
[2, 2],
[3, 6],
[-1, 32],
[2, 2],
[-3, 5]]))

Applications of PCA

- appropriate for visualization-plotting (reduce data to 2 or 3 Ds)

- less frequently used for:
  - Data compression (to reduce storage or transmission costs)
  - speeding up training of a supervised learning model.

Reinforcement learning

eg:

Position of helicopter ⟶ how to move control sticks

state (S) ⟶ action (a)

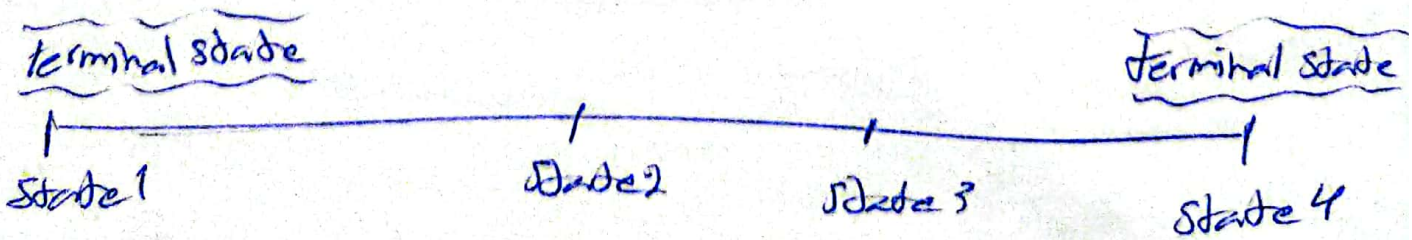
↑

reward function

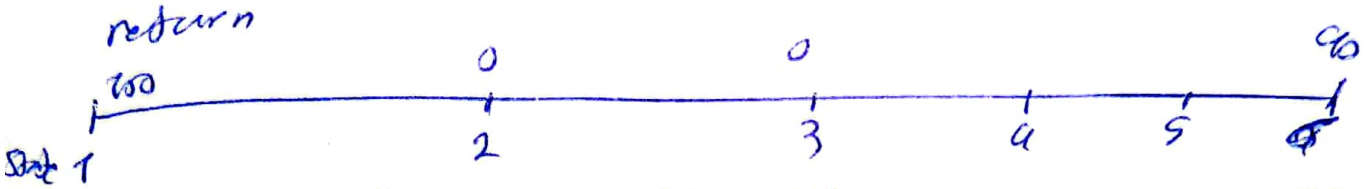
- Positive reward: Helicopter flying well : +1 ↑
- negative reward: helicopter flying poorly : −1 ↓

Applications of reinforcement algorithms

- controlling robots
- factory optimization
- financial (stock) trading
- playing games (including video games)



return in reinforcement ~~algorithms~~ learning



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

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

discount factor $\gamma = 0.9$ → in this example

a common choices of γ : 0.9 , 0.99 , 0.999 , ...

a number near to 1

exp;

$\gamma = 0.5$

return = $0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 = 12.5$

making decision in reinforcement learning

State → action

$S \xrightarrow{\pi} a$

π ↓ policy

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

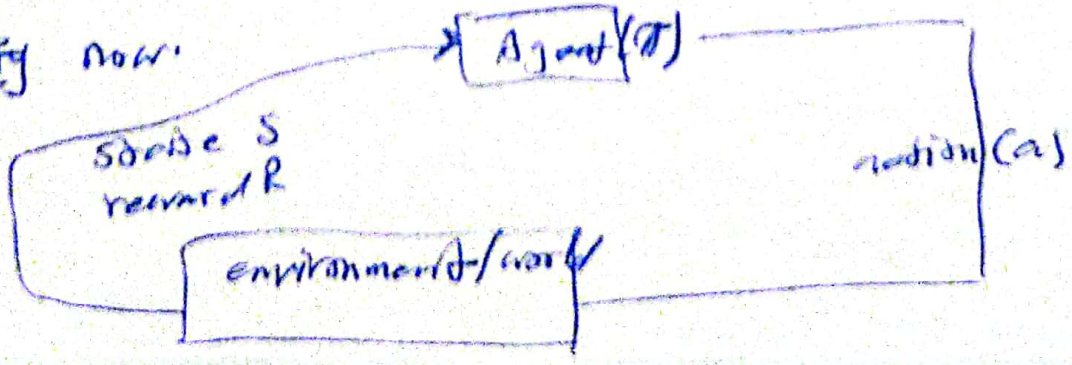
key concepts in reinforcement learning

example (mars rover)

state	6 states
actions	← →
rewards	100, 0.0, 0, 0, 40
discount factor γ	0.5
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$
Polity	

Markov Decision Process (MDP)

- in a markov decision process the current state depends on the last step. in other words the feature step depends on where you are now.

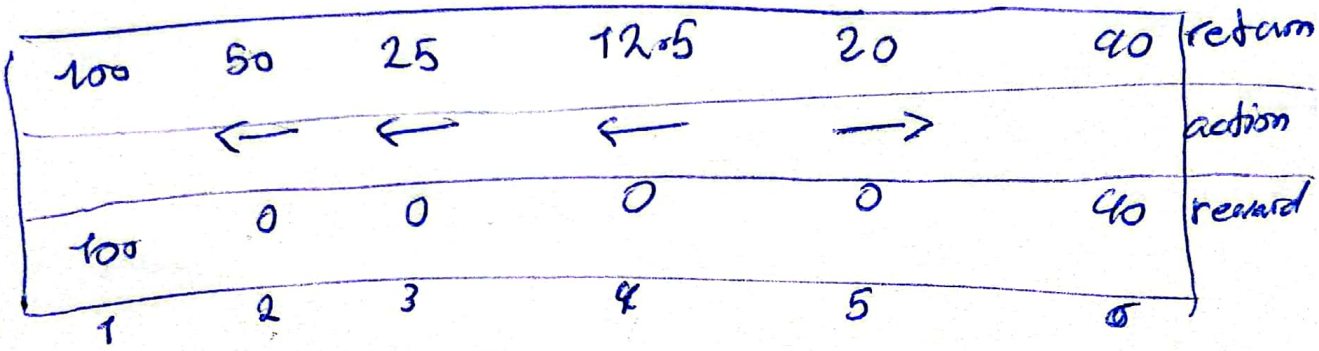


State-action value function

$Q(S, a)$ = Return if you

- start in state S.
- take action a (once)
- then behave optimally after that.

exp:



$Q(2, \rightarrow) = 12.5$

$\hookrightarrow 0 + (0.5)\,0 + (0.5)^2\,0 + (0.5)^3\,100$

start in state 2

take action a (once) (go to state 3)

behave optimally (go again to state 2)

finally go to state 1

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

Bellman Equation

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

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

a: current action

s': state you get to after taking action a.

a': action that you take in state s'

Bellman equation

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

$R(s)$: reward you get right away

$\max_{a'} Q(s',a')$: return from behaving optimally starting from state s'.

exp:

100	100	50	12.5	25	6.25	12.5	10	6.25	20	40	40	$s=2$
	100		0		0		0		0		40	$a=\rightarrow$
												$s'=3$
1		2		3		4		5		6		

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

$$= 0 + (0.5 \times 25) = 12.5$$

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

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

$= R_1 + \gamma [R_2 + \gamma R_3 + \gamma^2 R_4 + \cdots]$

$Q(s,a)$

both have the same result.

Random (Stochastic) environment

stochastic Reinforcement ~~to~~ Algorithm

expected

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

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

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

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

Continuous State Spaces

exp,1:

continuous state: (a car)

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

- x → x-Position
- y → y-Position
- θ → orientation
- $\dot{x}$ → velocity in x direction
- $\dot{y}$ → velocity in y direction
- $\dot{\theta}$ → ~~velocity of change in θ~~ How quickly the angle of car changes.

exp 2: Autonomous Helicopter.

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

→ angles in 3D.

continuous state space

("Lunar Lander")

actions:
- do nothing
- left thruster
- main thruster
- right thruster

$s =$ [x, y, $\dot{x}$, $\dot{y}$, θ, $\dot{\theta}$, l_g, r_g]

$x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}$ → left and right

l_g, r_g → binary values

Reward function

- Getting to landing path: 100 - 140
- Additional reward for moving toward/away from path
- crash: -100
- soft landing: +100
- leg grounded: +10
- fire main engine: -0.3
- fire side thruster: -0.03

Lunar lander problem

- Learn a policy π that, given

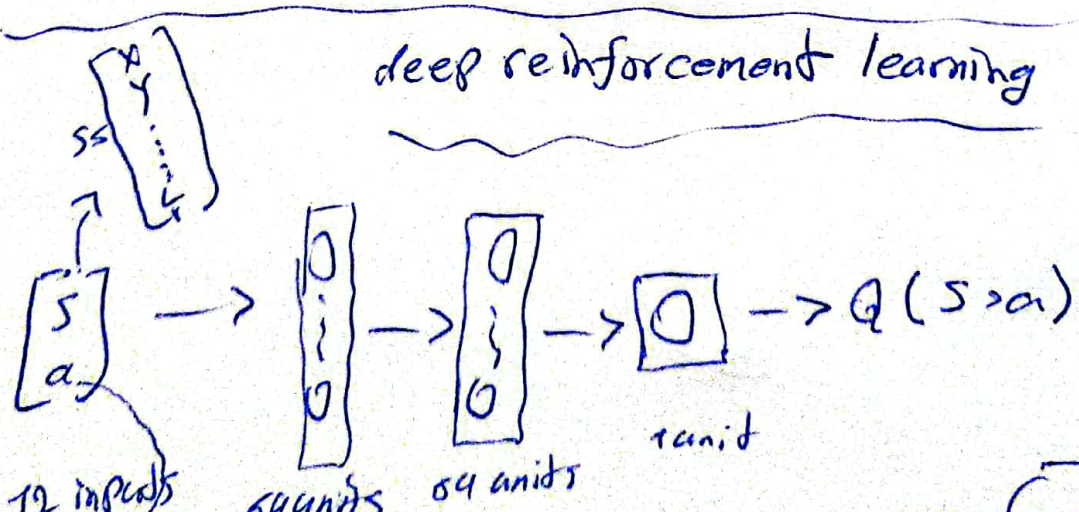
$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ l \\ r \end{bmatrix}$$

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

~~$\gamma = 0$~~.

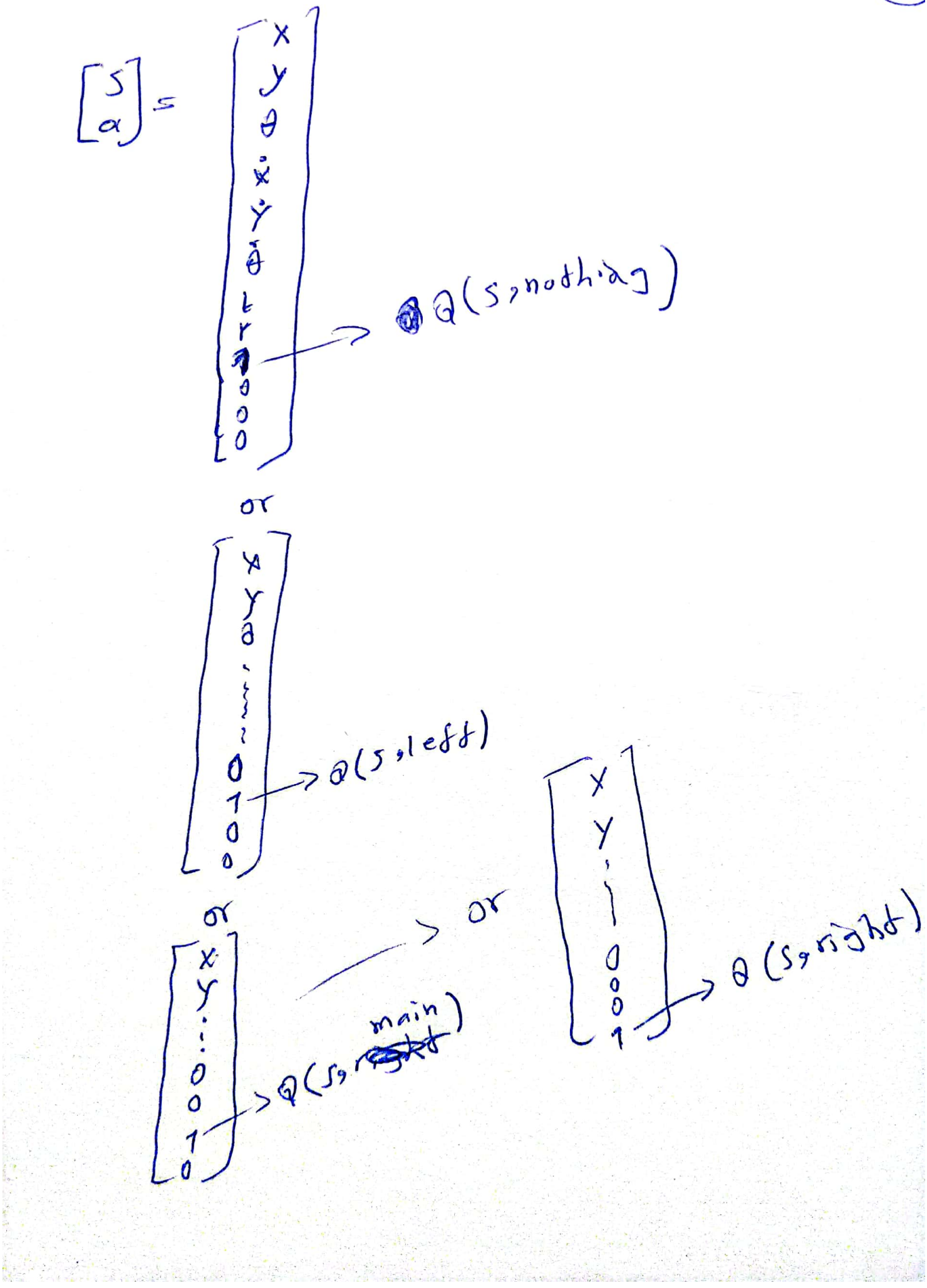
- discount factor $\gamma = 0.985$

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 that maximizes $Q(s, a)$.


[S a] =
x
y
θ
ẋ
ẏ
θ̇
L
R
0
0
0
0
Q(S, nothing)
or
x
y
0
0
1
0
0
Q(S, left)
or
x
y
0
0
1
0
Q(S, right)
or
x
y
1
0
0
0
1
Q(S, right)

$Q(s,a)$ ——> output of ~~neural~~ deep reinforcement network

y or ~~target~~

Bellman equation

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

x

y

create some labled training examples, ~~for~~,

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

$(s^{(2)}, a^{(1)}, R(s^{(2)}), s'^{(2)})$

⋮

X	Y
$x^{(1)} = (s^{(1)}, a^{(1)})$	$y^{(1)}$

$$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$$

Learning algorithm (reinforcement)
↳ (Deep Q network)

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

- Repeat {

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

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

Train neural network:

- create training set of 10000 examples using

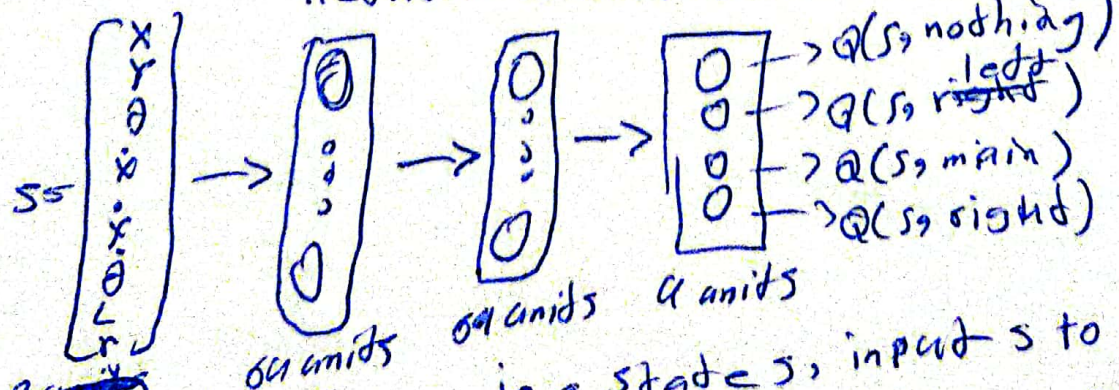
$x = (s,a)$ and $y = R(s) + \max_{a'} Q(s',a')$

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

set $Q = Q_{new}$

}

Algorithm (DQN) refinement: improved neural network architecture:



In a state s, input s to neural network. Pick the action a that maximizes $Q(s,a)$.

Algorithm reinforcement: ε-greedy policy

How to choose actions while still learning ?

- in some state s

option 1:
- pick the action a that maximizes $Q(s,a)$. → "greedy action" → or "exploitation"

option 2:
- with probability 0.95, pick the action a that maximizes $Q(s,a)$.
- with probability 0.05, pick an action a randomly. → "exploration step"

→ the better option

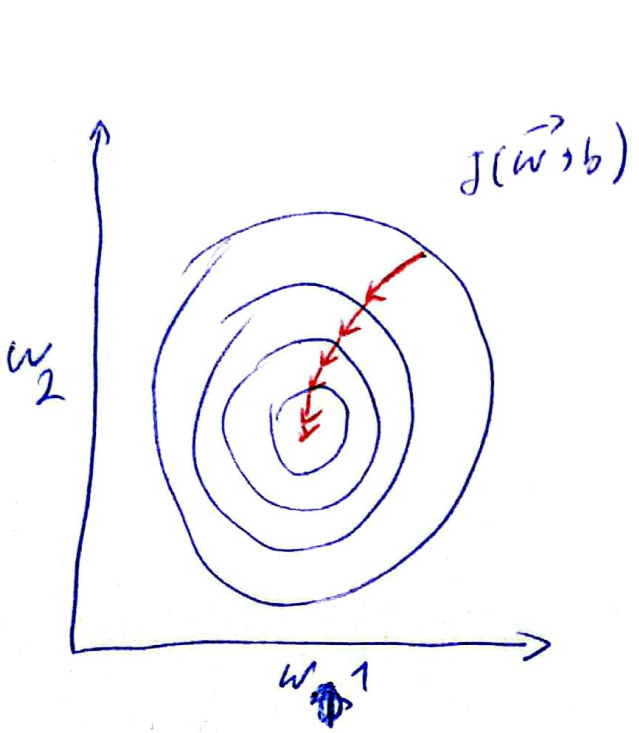
→ ε-greedy policy ($\varepsilon = 0.05$)

Start ε high
Gradually decrease

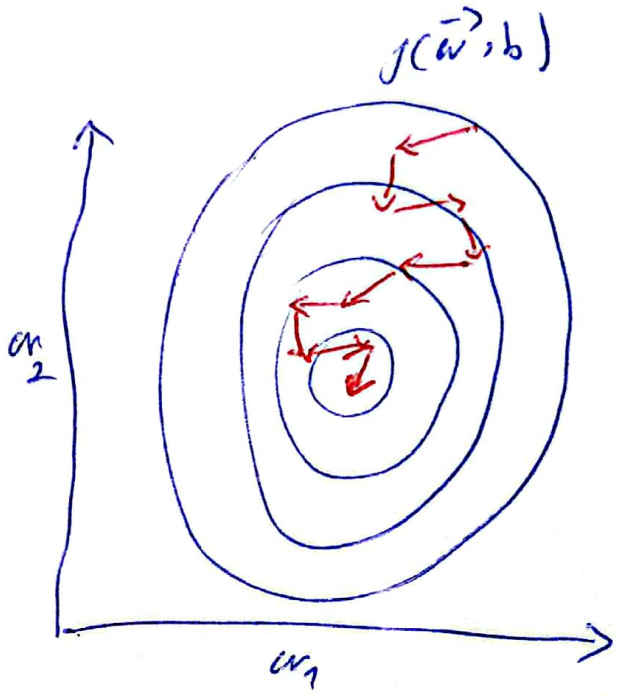
→ exp: start, $\varepsilon = 1$

finally: $\varepsilon = 0.01$

Algorithm refinement: mini-batch and soft updates.



batch learning



mini-batch learning

faster algorithm

soft update

Set $Q = Q_{new}$

w, B $\quad$ w_{new}, B_{new}

$$w = 0.01\, w_{new} + 0.99\, w$$

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

soft update.

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.

The end of the specialization machine-learning ~~specia~~

2024.2.27

:)

(Sanni