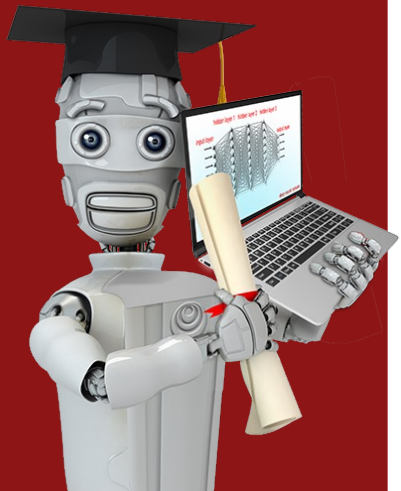
DeepLearning.AI

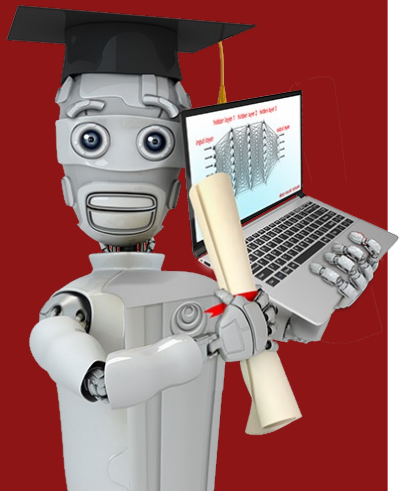
Stanford
ONLINE



Recommender Systems

DeepLearning.AI

Stanford ONLINE



Recommender System

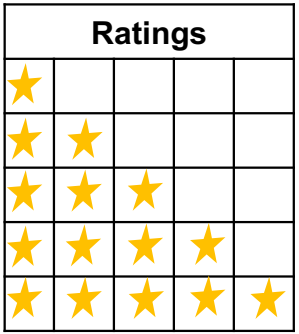
Making recommendations

Predicting movie ratings

User rates movies using one to five stars

zero

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



n_u = no. of users

n_m = no. of movies

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

$n_u = 4$ $r(1,1) = 1$

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

$y^{(i,j)}$ = rating given by user j to movie i (defined only if $r(i,j)=1$)

Collaborative Filtering

Notation:

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

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

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

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

$m^{(j)}$ = no. of movies rated by user j

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

This can be rewritten as $w^{(j)} \cdot x^{(i)}$ if there's $x_0^{(i)} = 1$.

just linear regression

For example, if x_1 is romance, x_2 is action. $x^{(1)} = \begin{pmatrix} 0.9 \\ 0 \end{pmatrix}$, $x^{(3)} = \begin{pmatrix} 0.99 \\ 0 \end{pmatrix}$

and for user 1: $w^{(1)} = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$, $b^{(1)} = 0$,

then predict rating for movie 3 as: $w^{(1)} \cdot x^{(3)} + b^{(1)} = 4.95$

The aim: to learn $w^{(j)}, b^{(j)}$

Cost function: for single user j, given $x^{(i)}$

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

number of features

Only sum over the movies i that user j has actually rated.

$m^{(j)}$: $m^{(j)}$ is eliminated because it's a constant.

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

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

Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	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 puppies of love	?	4	0	?	~~0.99~~ ?	~~0~~ ?
Nonstop car chases	0	0	5	4	~~0.1~~ ?	~~1.0~~ ?
Swords vs. karate	0	0	5	?	~~0~~ ?	~~0.9~~ ?

The cost function

$X^{(i)}$ is unknown

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

is built on the assumption that the feature vector for movie i, $x^{(i)}$, is known beforehand.

But, in reality, $x^{(i)}$ is unknown.

Collaborative Filtering Algorithm

Problem Motivation

Since $x^{(i)}$ is unknown, so we need to learn $x^{(i)}$. But how?

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

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

From these four equations, we can derive $x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Notice that this works only because we have parameters for four users. That's what allows us to try to guess $x^{(1)}$. This is why in a typical linear regression application if we had just a single user, we don't actually have enough information to figure out what should be the features, x_1 and x_2, which is why in the linear regression contexts that we saw in Course 1, we can't come up with features x_1 and x_2 from scratch.

But in collaborative filtering, we have ratings from multiple users of the same item with the same movie. That's what makes it possible to try to guess the features.

Cost function:

Given $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \ldots, w^{(n_u)}, b^{(n_u)}$ for all users, we want to learn $x^{(i)}$:

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

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

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

Now we are ready to put two cost functions together. Thus, the term collaborative filtering

Cost function to learn $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(n_u)}, b^{(n_u)}$:

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

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

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

Put them together:

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

Gradient Descent:

Step 1: Initialize $x^{(1)},x^{(2)},\dots,x^{(n_m)}, w^{(1)},w^{(2)},\dots,w^{(n_u)}, b^{(1)},b^{(2)},\dots,b^{(n_u)}$ to small random values.

Step 2:

Repeat {

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

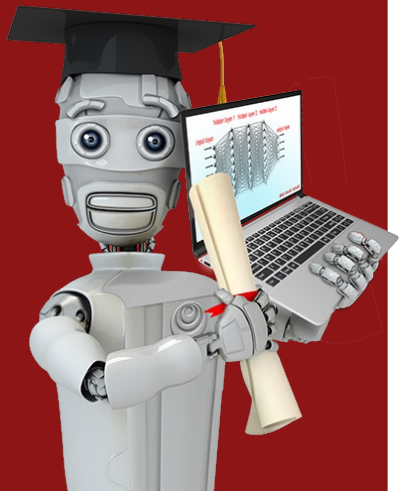
$$b^{(j)} = b^{(j)} - \alpha\frac{\partial}{\partial b^{(j)}}J(w,b,x)$$

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

}

DeepLearning.AI

Stanford
ONLINE



Collaborative Filtering

Binary labels:
favs,
likes and clicks

Binary labels

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

1 : hit like

0 : not hit like

? : not watch the movie

The binary labels can represent many features.
It depends on the questions.

Example applications

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

Meaning of ratings:

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

From regression to binary classification

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

where $g(z) = \frac{1}{1+e^{-z}}$

Cost function for binary application

Previous cost function:

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

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

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

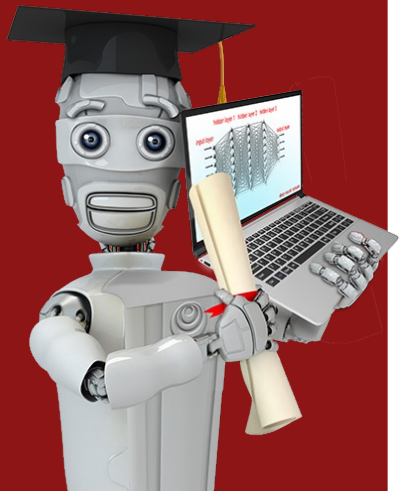
$$J(w,b,x)=\sum_{(i,j):r(i,j)=1}L\left(f_{(w,b,x)}(x),y^{(i,j)}\right)$$

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

cost for all examples

DeepLearning.AI

Stanford
ONLINE



Recommender Systems implementation

Mean normalization

In the first course, Andrew showed that future normalization can help the gradient descent run faster. It turns out the same idea can also help recommender systems performs better.

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	?	$r^{(1)}$
Romance forever	5	?	?	0	?	$r^{(2)}$
Cute puppies of love	?	4	0	?	?	$r^{(3)}$
Nonstop car chases	0	0	5	4	?	$r^{(4)}$
Swords vs. karate	0	0	5	?	?	$r^{(5)}$

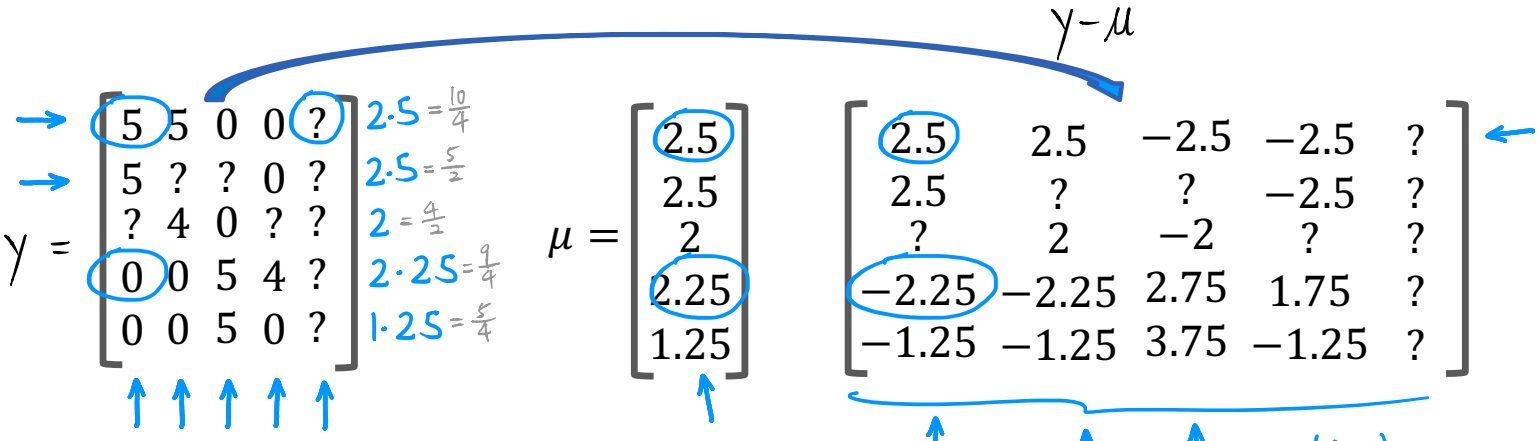
$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

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

For the new user Eve who hasn't had any rating, its parameters should be $w^{(5)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $b^{(5)} = 0$. Thus, the prediction $w^{(5)} \cdot x^{(i)} + b^{(5)} = 0$, which is not making too much sense.

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

Mean Normalization



For user j, on movie i predict:

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

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

User 5 (Eve):

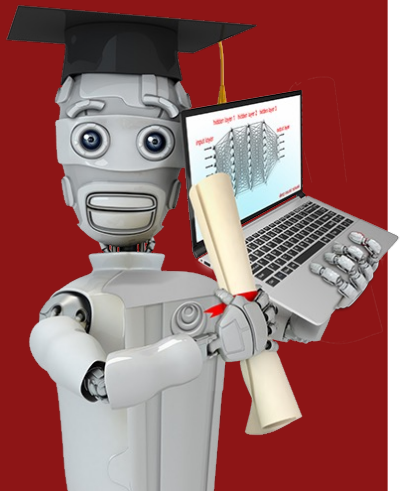
$$w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(5)} = 0$$

$$\underbrace{w^{(5)} \cdot x^{(1)} + b^{(5)}}_{0} + \mu_1 = 2.5$$

Now, the prediction is 2.5 instead of 0.

DeepLearning.AI

Stanford
ONLINE



Recommender Systems implementational detail

TensorFlow implementation

Derivatives in ML

Gradient descent algorithm

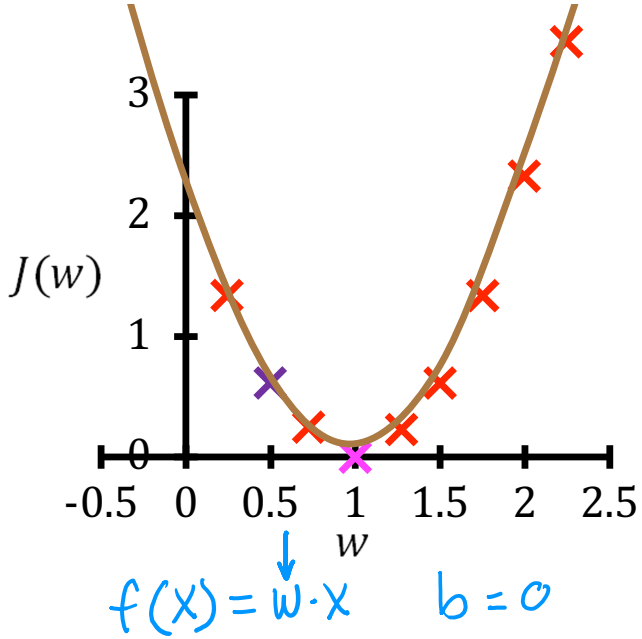
Repeat until convergence

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

$$b = b - \alpha \frac{d}{db} J(w,b) \leftarrow b = 0$$

Learning rate

Derivative



Custom Training Loop

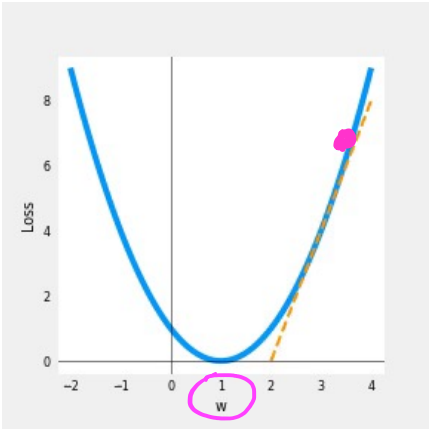
$$J = (wx - 1)^2$$

Gradient descent algorithm

Repeat until convergence

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

Fix b = 0 for this example




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

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

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

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

Tf.variables are the parameters we want to optimize

Auto Diff

Auto Grad

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

tf.variables require special function to modify

Implementation in TensorFlow

Gradient descent algorithm

Repeat until convergence

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

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

$$X = X - \alpha \frac{d}{dx} J(w,b,x)$$

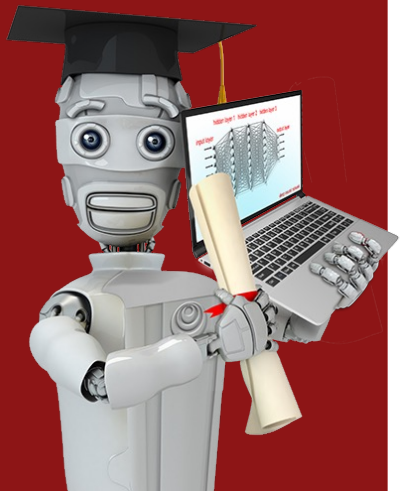
```
# Instantiate an optimizer.
optimizer = keras.optimizers.Adam(learning_rate=1e-1)

iterations = 200
for iter in range(iterations):
    # Use TensorFlow's GradientTape
    # to record the operations used to compute the cost
    with tf.GradientTape() as tape:

        # Compute the cost (forward pass is included in cost)
        cost_value = cofiCostFuncV(X, W, b, Ynorm, R,
            num_users, num_movies, lambda)

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

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

n_u n_m $y^{(i,j)}$ $r^{(i,j)}$

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

DeepLearning.AI

Stanford
ONLINE



After training the model, $x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}$ are known. But, we don't know what the meaning is for these features. E.g., we don't know if x_1 is romance or x_2 is action, etc. But we can still find similar items by calculating distances among features.

Collaborative Filtering

Finding related items

Finding related items

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

To find other items related to it,
find item k with $x^{(k)}$ similar to $x^{(i)}$

romance
action

x_1, x_2, x_3

n

i.e. with smallest distance

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

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

$x^{(k)}$ $x^{(i)}$

Limitations of Collaborative Filtering

Cold start problem. How to

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

Use side information about items or users:

Collaborative filtering cannot use side information about items or users.

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

Content-based Filtering

Collaborative filtering v.s. Content-based filtering

Collaborative filtering:

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

Content-based filtering:

Recommend items to you based on features of users and features of items to find good match.

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

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

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

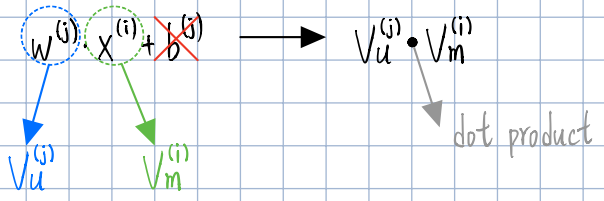
vector size could be different.

Content-based filtering: Learning to match

Previously, in collaborative filtering, Predict rating of user j on movie i as

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

But now, in content-based filtering,



$v_u^{(j)}$: computed from $x_u^{(j)}$

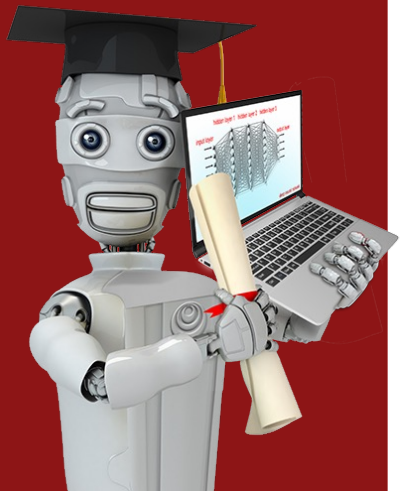
$v_m^{(i)}$: computed from $x_m^{(i)}$

Note that $x_u^{(j)}$, $x_m^{(i)}$ could be different size but $v_u^{(j)}$ and $v_m^{(i)}$ must be the same size.

The question is how to calculate $v_u^{(j)}$ and $v_m^{(i)}$. Ans: Deep Learning

DeepLearning.AI

Stanford
ONLINE

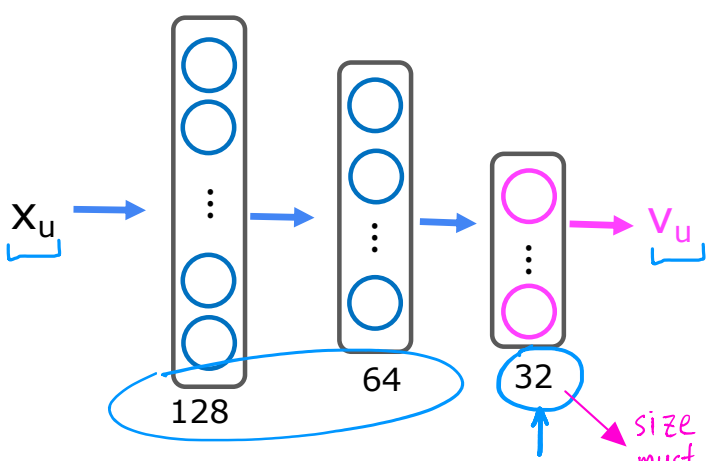


Content-based Filtering

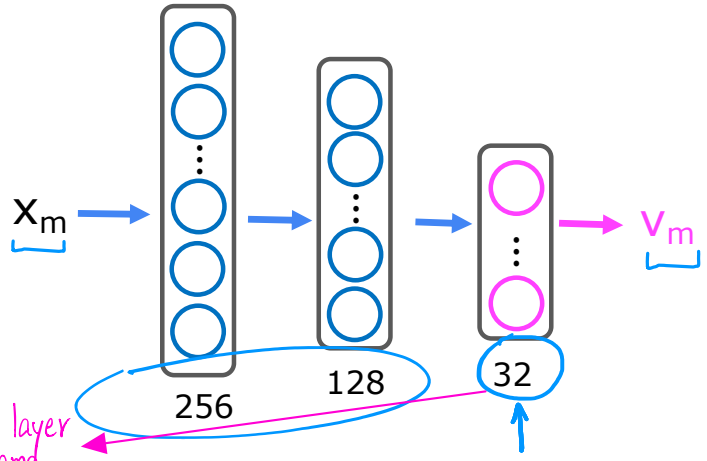
Deep learning for content-based filtering

Neural network architecture

$x_u \rightarrow v_u$ User network



$x_m \rightarrow v_m$ Movie network

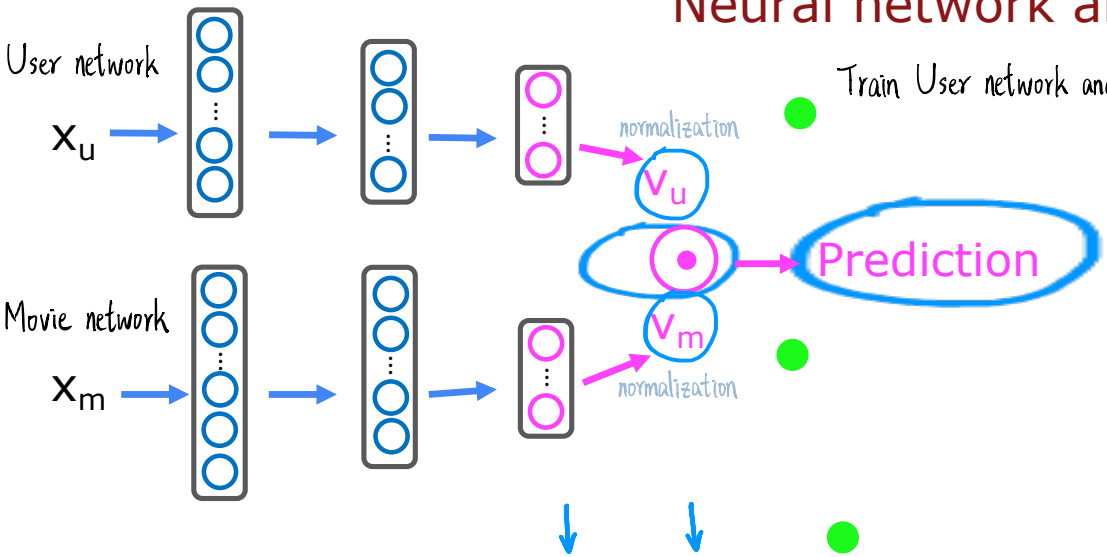


size of output layer must be the same.

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

sigmoid function $g(v_u^{(j)} \cdot v_m^{(i)})$ ***to predict the probability that*** $y^{(i,j)}$ ***is*** 1 (Only do it if output label is binary.)

Neural network architecture



Train User network and Movie network altogether.

Cost function

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

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:

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

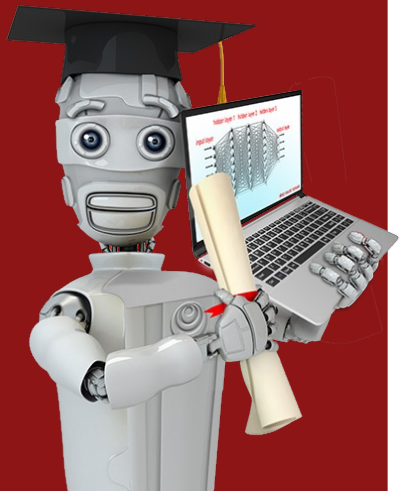
For each movie i, similar K movies can be figured out when features vectors $v_m^{(i)}$ are learned.

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

- Note: This can be pre-computed ahead of time

DeepLearning.AI

Stanford
ONLINE



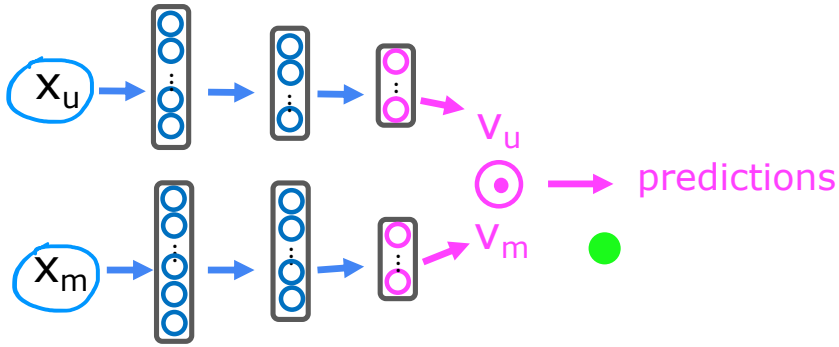
Advanced implementation

Recommending from a large catalogue

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

The problem is that when a user Xu visits my website and I have to run thousands or millions times of inferences to predict the best item, This is super inefficient.

- Movies 1000+
- Ads 1m+
- Songs 10m+
- Products 10m+



Two steps: Retrieval & Ranking

Retrieval:

- Generate large list of plausible item candidates

  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

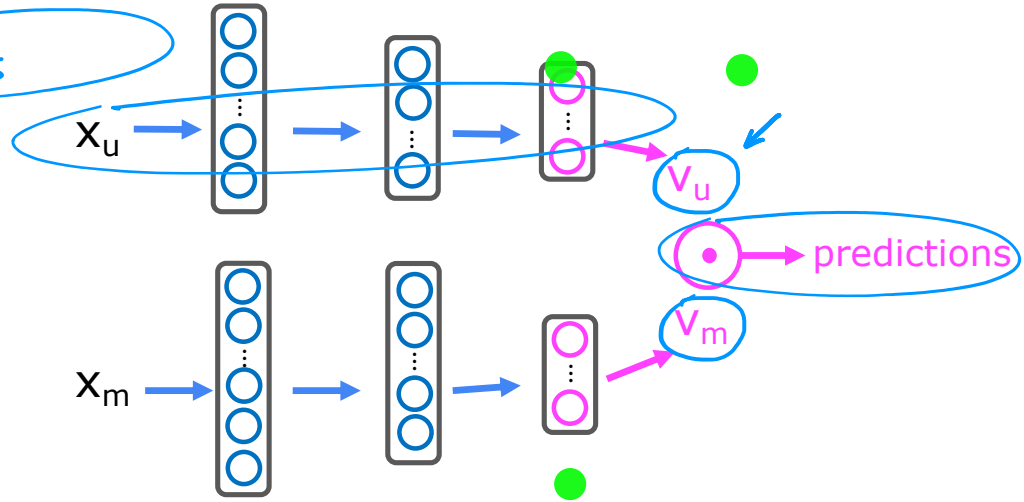
70 x 10
~100s

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

Two steps: Retrieval & ranking

Ranking:

- Take list retrieved and rank using learned model



- Display ranked items to user

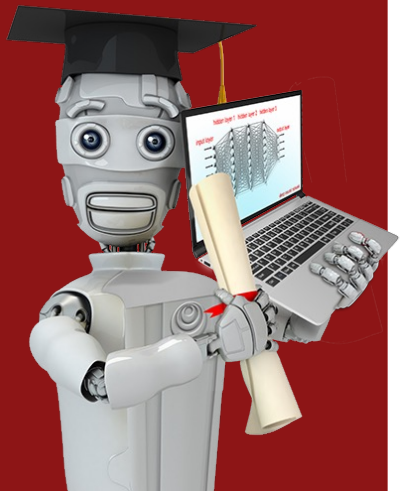
Retrieval step

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

100 500

DeepLearning.AI

Stanford
ONLINE



Advanced implementation

Ethical use of recommender systems

What is the goal of the recommender system?

Recommend:

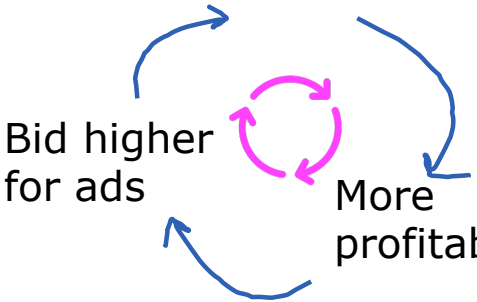
- Movies most likely to be rated 5 stars by user
- Products most likely to be purchased
- Ads most likely to be clicked on +high bid
- Products generating the largest profit
- Video leading to maximum watch time

這就是我為什麼這麼喜歡 Andrew 的原因

Ethical considerations with recommender systems

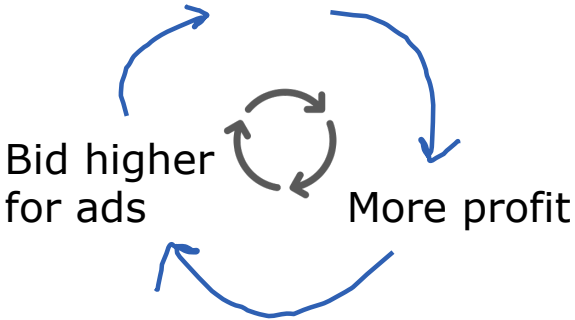
Travel industry

Good travel experience to more users



Payday loans

Squeeze customers more



Amelioration: Do not accept ads from exploitative businesses

Other problematic cases:

- Maximizing user engagement (e.g. watch time) has led to large social media/video sharing sites to amplify conspiracy theories and hate/toxicity

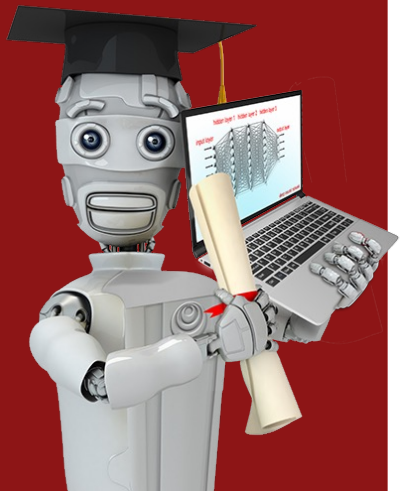
  → Amelioration : Filter out problematic content such as hate speech, fraud, scams and violent content

- Can a ranking system maximize your profit rather than users' welfare be presented in a transparent way?

  → Amelioration : Be transparent with users

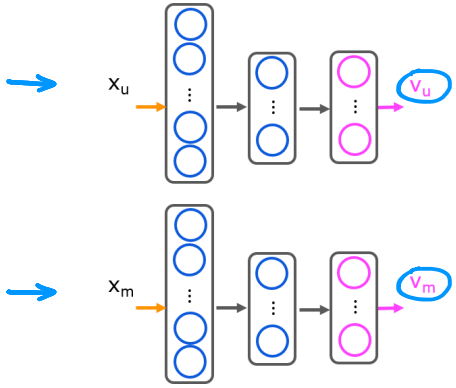
DeepLearning.AI

Stanford
ONLINE



Content-based Filtering

TensorFlow Implementation



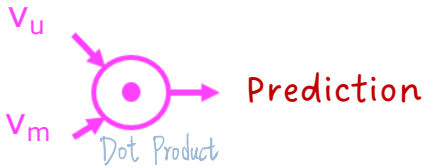

```
user_NN = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(32)
])

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 )
])
```

The output layers' size must be the same.

```
# 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()
```

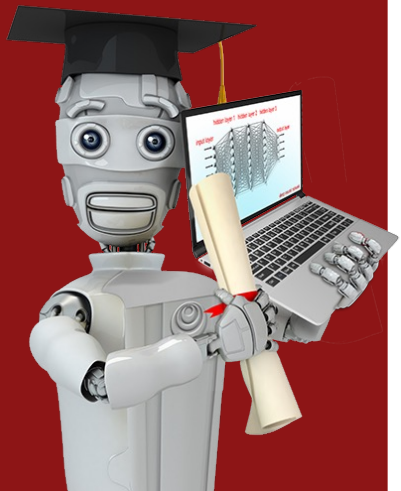
normalization

normalization



DeepLearning.AI

Stanford ONLINE

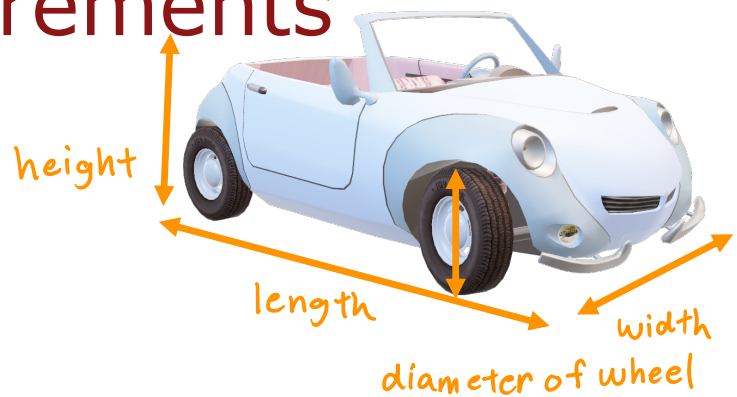


Principal Component Analysis (Optional)

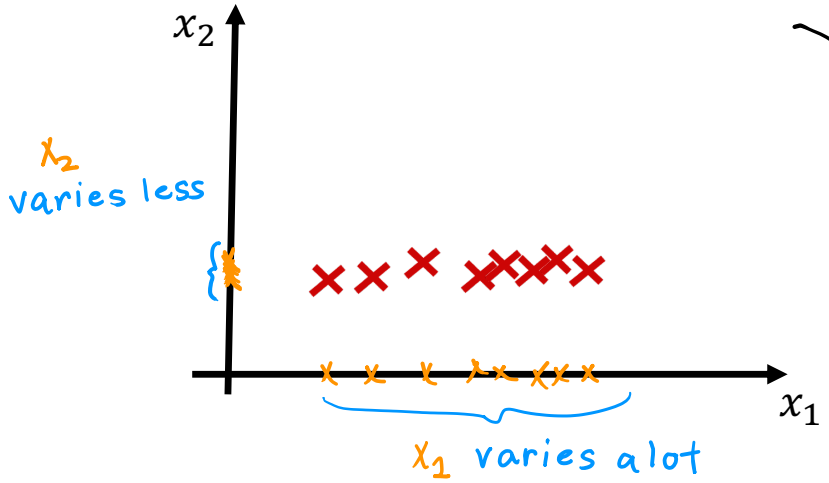
Reducing the number of features

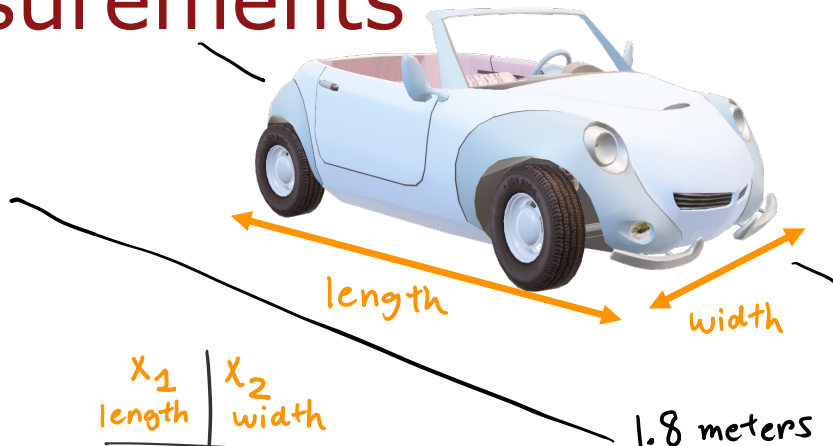
Car measurements

Given cars' features as shown in the right image



Car measurements

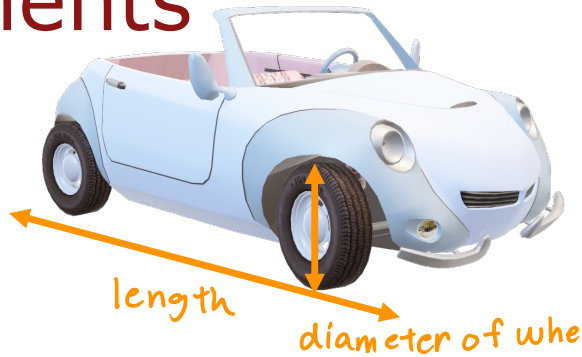


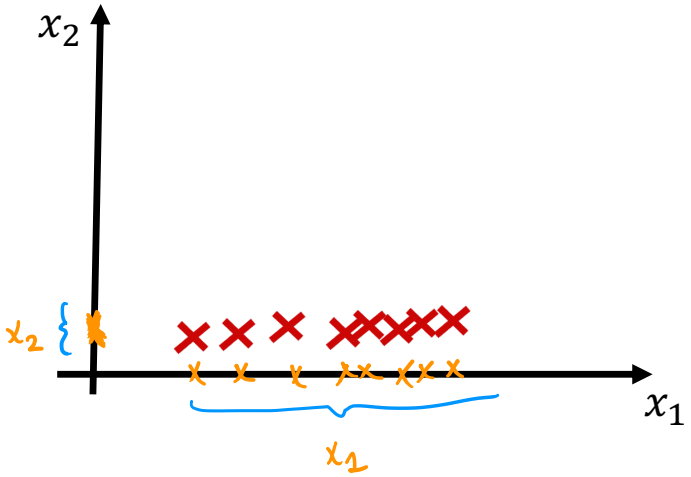


x_1 length	x_2 width

can just take x_1
to reduce number of features

Car measurements



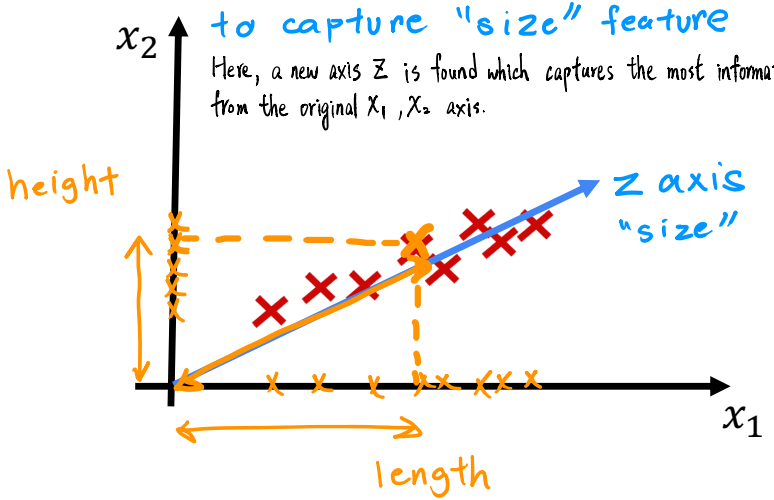


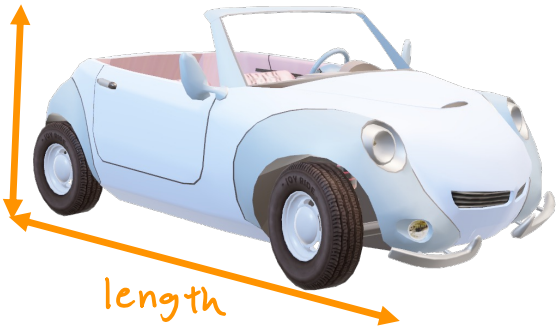
x_1	x_2
length	diameter

can just take x_1
to reduce number of features

Size

PCA: find new axis and coordinates
use fewer numbers
to capture "size" feature



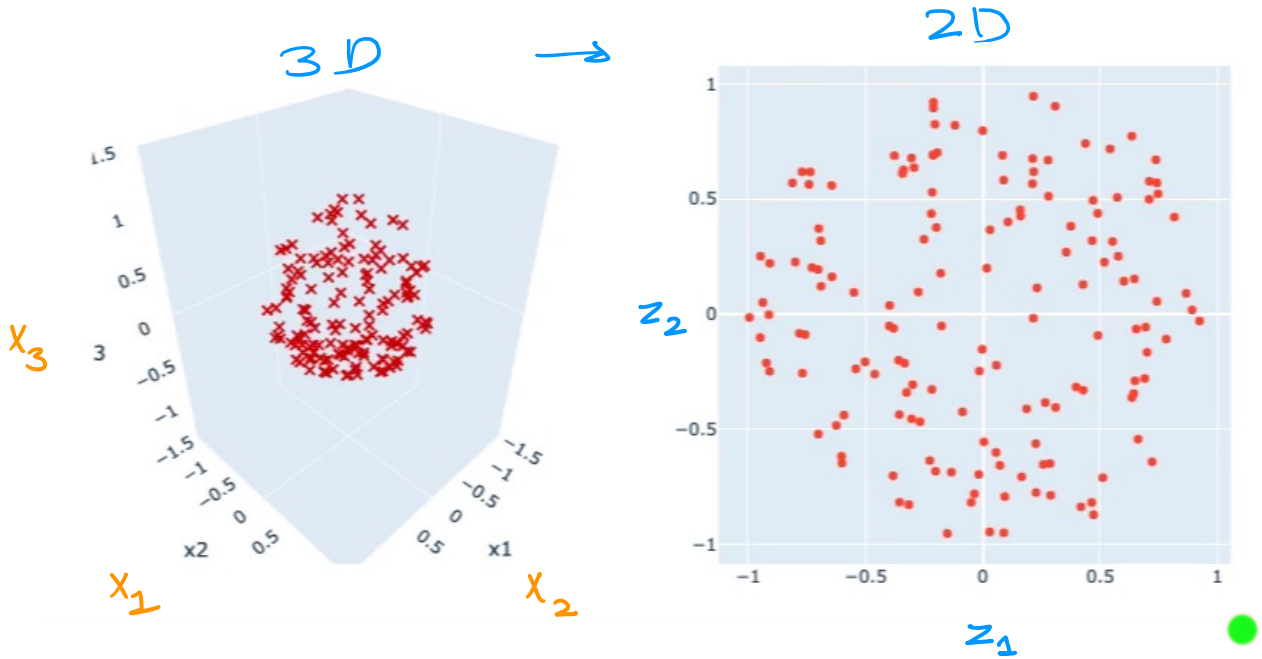


X_1	X_2
length	height

2 → 1 features

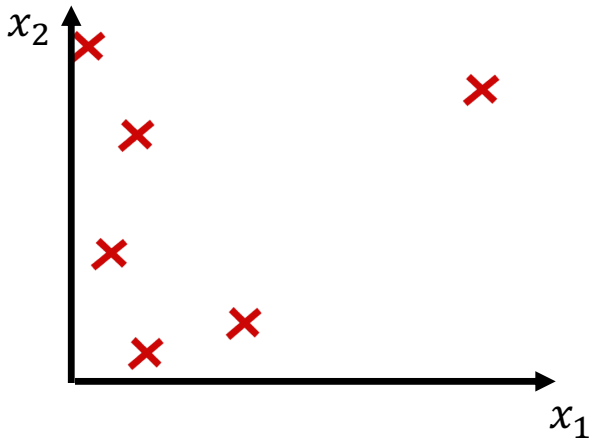
many features → 2 or 3 features

From 3D to 2D

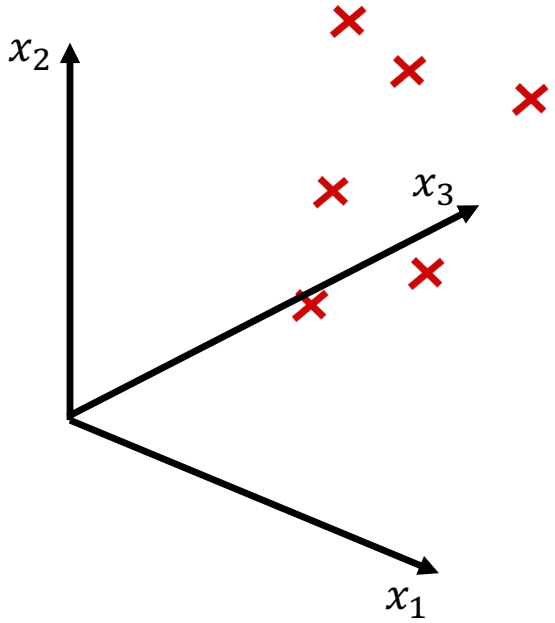


Country	GDP (trillions of US$)
Canada	1.577
China	5.878
India	1.632
Russia	1.48
Singapore	0.223
USA	14.527
…	…

Country	GDP (trillions of US$)	Per capita GDP (thousands of intl. $)
Canada	1.577	39.17
China	5.878	7.54
India	1.632	3.41
Russia	1.48	19.84
Singapore	0.223	56.69
USA	14.527	46.86



Country	GDP (trillions of US$)	Per capita GDP (thousands of intl. $)	Human Development Index
Canada	1.577	39.17	0.908
China	5.878	7.54	0.687
India	1.632	3.41	0.547
Russia	1.48	19.84	0.755
Singapore	0.223	56.69	0.866
USA	14.527	46.86	0.91
…	…	…	…



Country	GDP (trillions of US$)	Per capita GDP (thousands of intl. $)	Human Development Index	Life expectancy
Canada	1.577	39.17	0.908	80.7
China	5.878	7.54	0.687	73
India	1.632	3.41	0.547	64.7
Russia	1.48	19.84	0.755	65.5
Singapore	0.223	56.69	0.866	80
USA	14.527	46.86	0.91	78.3
…	…	…	…	…

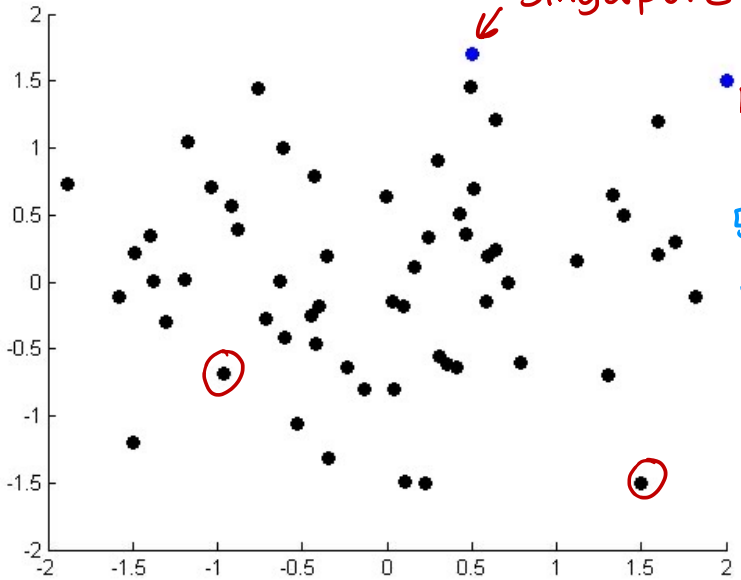
what if 50 features?

50 features

Country	x_1 GDP (trillions of US$)	x_2 Per capita GDP (thousands of intl. $)	x_3 Human Development Index	x_4 Life expectancy	... Poverty Index (Gini as percentage)	Mean household income (thousands of US$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...

Country	z_1	z_2
Canada	1.6	1.2
China	1.7	0.3
India	1.6	0.2
Russia	1.4	0.5

Data visualization



Per Person GDP

z_2

z_1

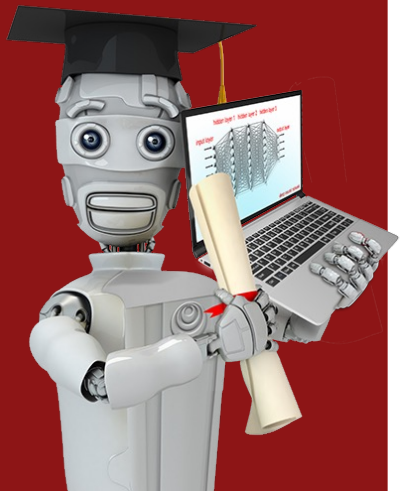
50 features → 2 features

reduce 50 "dimensional" data to 2D data

country's GDP

DeepLearning.AI

Stanford
ONLINE



Principal Component Analysis

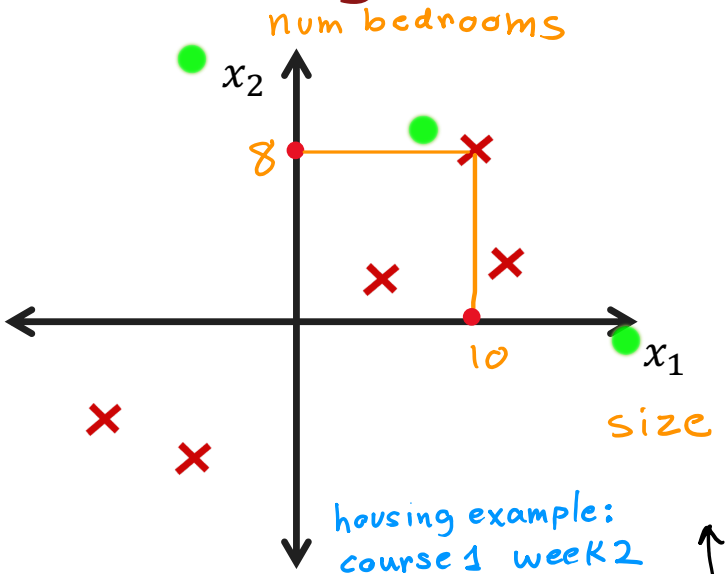
PCA Algorithm

PCA algorithm

coordinates

$x_1 = 10 \quad x_2 = 8$

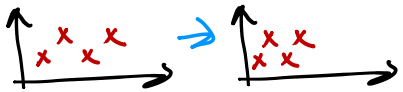
Can we choose a different axis?



Preprocess features

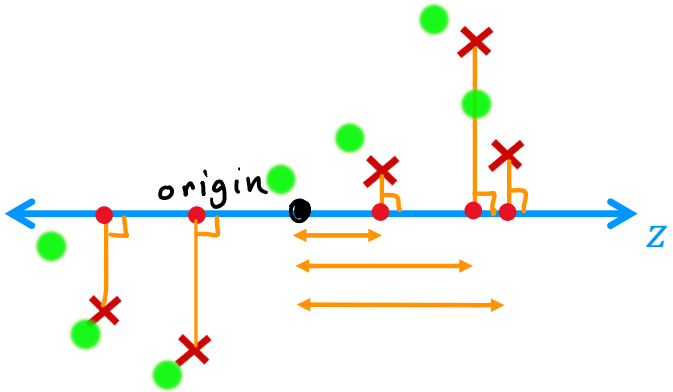
Normalized to have zero mean

feature scaling



Choose an axis

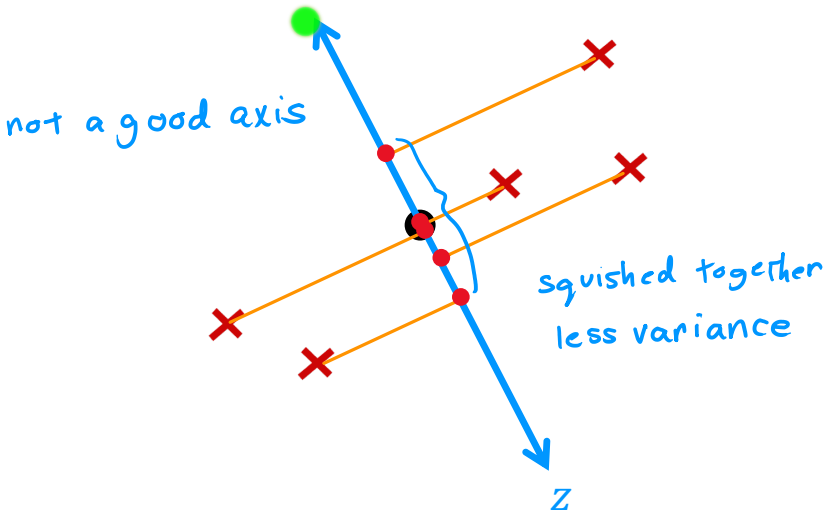
"project" examples onto the axis



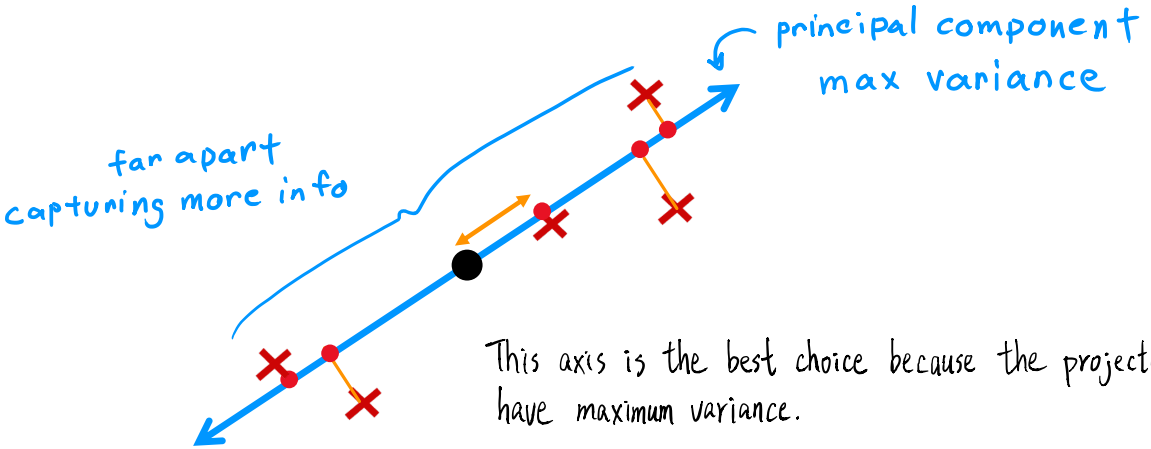
variance is large

capturing info of original data

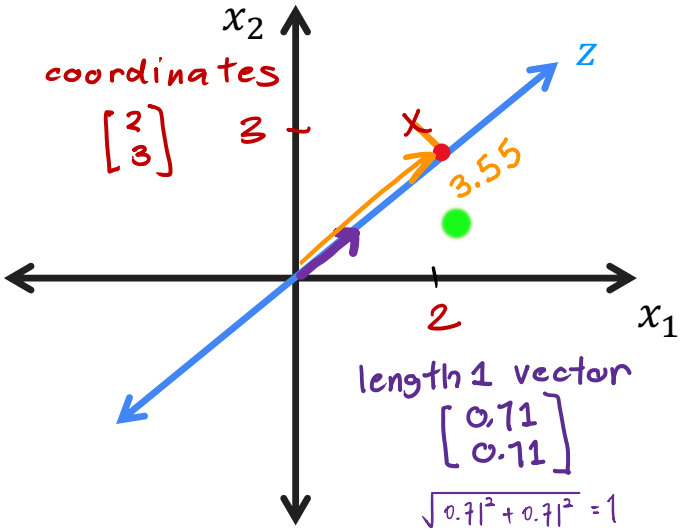
Choose an axis



Choose an axis



Coordinate on the new axis

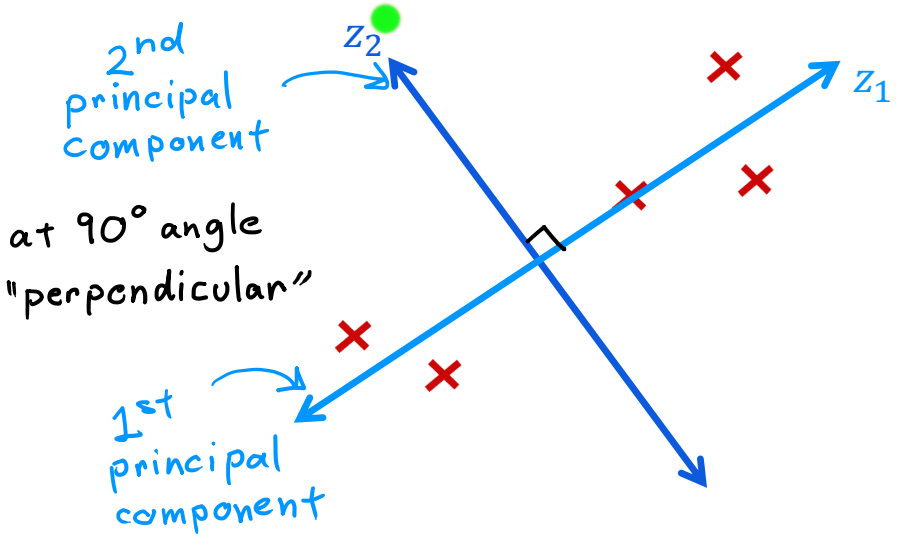


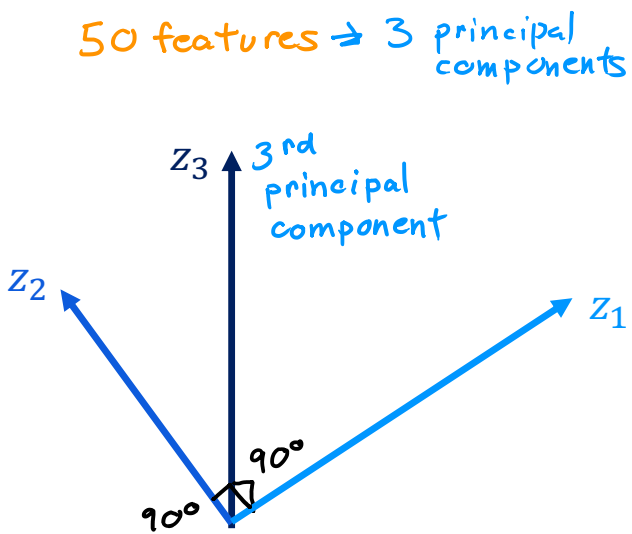
dot product

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

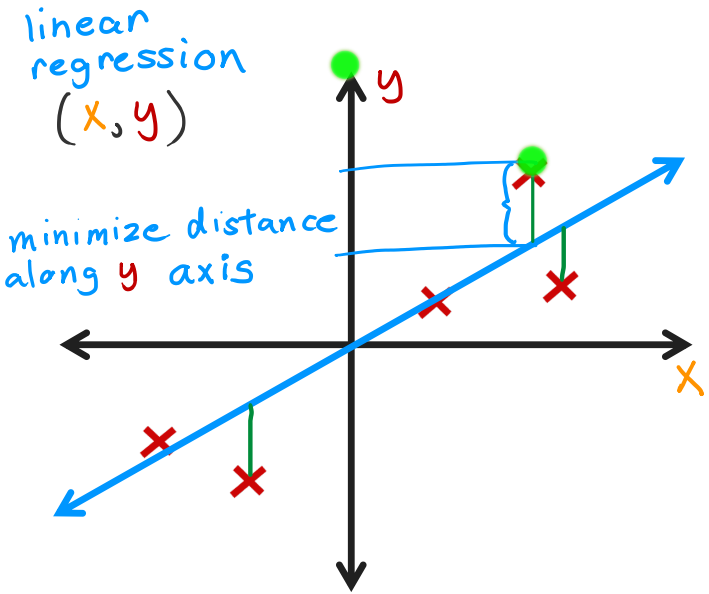
$2 \times 0.71 + 3 \times 0.71 = 3.55$

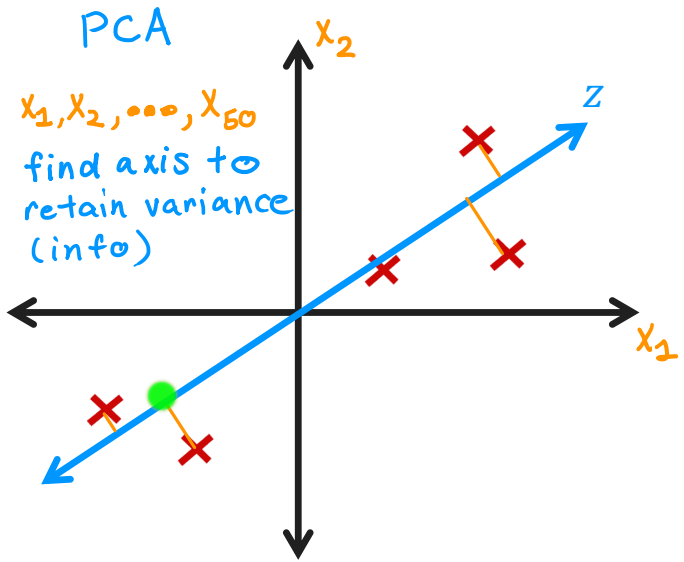
More principal components



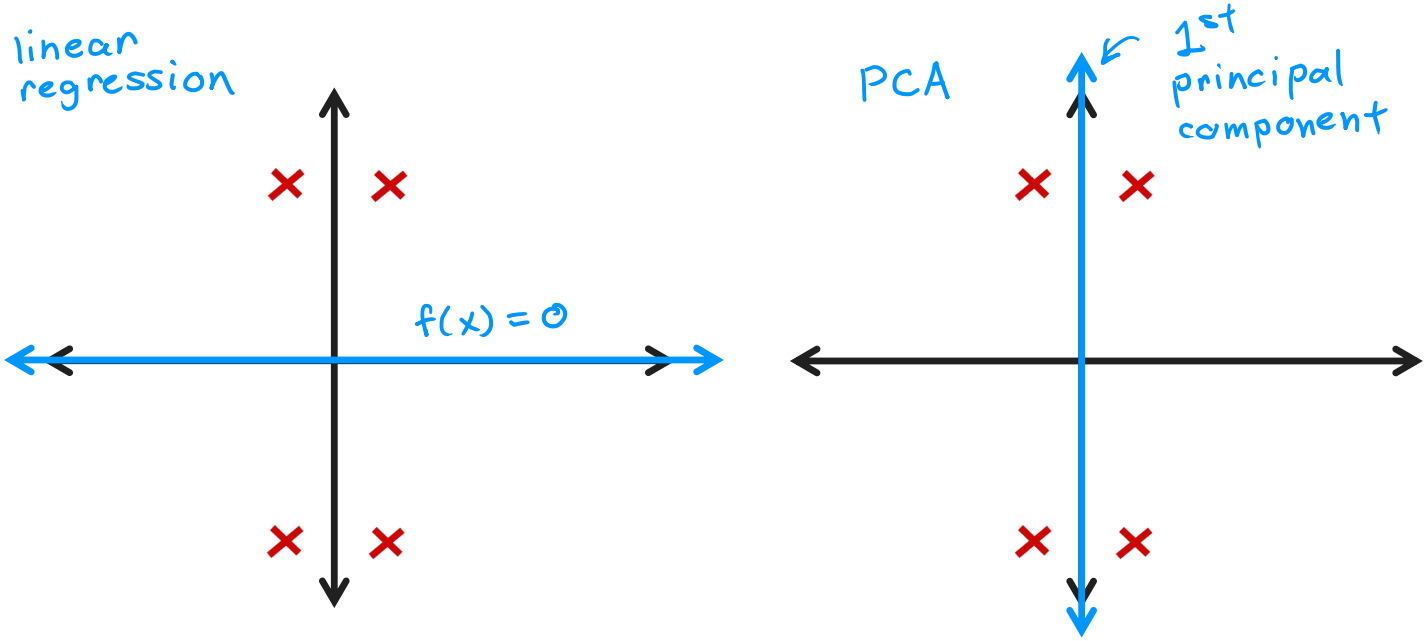


PCA is not linear regression

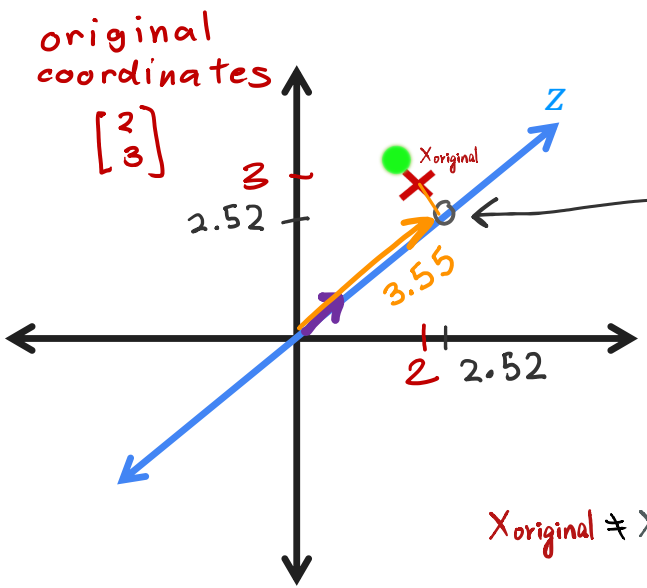




PCA is not linear regression



Approximation to the original data



given $z = 3.55$,

find original (x_1, x_2) (approximately)

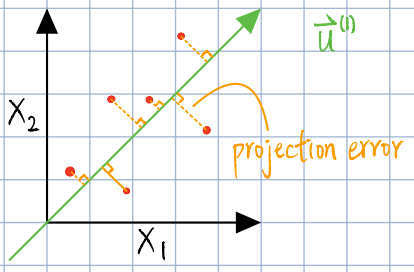
"reconstruction"

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

$x_{approximate}$

$x_{original} \neq x_{approximate}$

PCA Algorithm



Given training set $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$, $x^{(i)} \in \mathbb{R}^n$

The basic idea is to find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Then, we can find k principal components $u^{(1)}, u^{(2)}, \ldots, u^{(k)} \in \mathbb{R}^n$

Data Preprocessing:

Calculate mean: $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$, $j = 1, \ldots, n$

Move toward mean: replace each $x_j^{(i)}$ with $x_j^{(i)} - \mu_j$

Rescale features: E.g. X_1 = size of house, X_2 = number of bedrooms. We can rescale X_1 and X_2 to have comparable range of values.

Compute "covariance matrix". $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)T}$. $\Sigma \in \mathbb{R}^{n \times n}$

Compute "eigen vectors" of Σ: $u, s, v = \text{svd}(\Sigma)$

$$u = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad u_{reduce} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times k}, \quad \underset{k \times 1}{z^{(i)}} = \underset{k \times n}{u_{reduce}^T} \underset{n \times 1}{x^{(i)}} \in \mathbb{R}^{k \times 1}$$

Reconstruction: $x_{approximate}^{(i)} = u_{reduce} \cdot z^{(i)}$. $x_{approximate}^{(i)} \in \mathbb{R}^n$

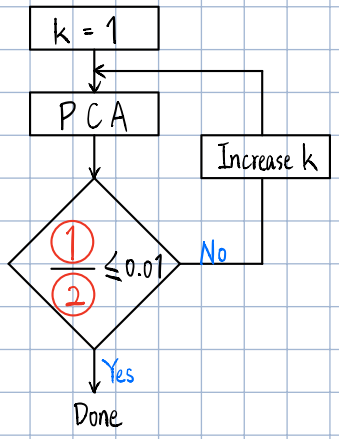
Now, the question is how to determine k? I.e., how many principal components are enough to capture most information?

Calculate average squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \| x^{(i)} - x_{approximate}^{(i)} \|^2$ — ①

Calculate total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)T}$ — ②

$$S = \begin{bmatrix} S_{11} & & 0 \\ & S_{22} & \\ 0 & & \ddots \\ & & & S_{ii} \end{bmatrix}$$

Run:



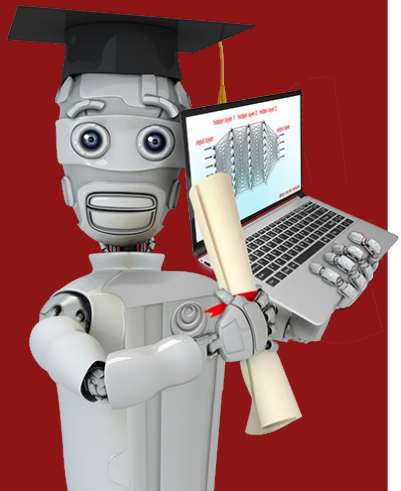
$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

$$\Rightarrow \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

99% of variance is retained.

DeepLearning.AI

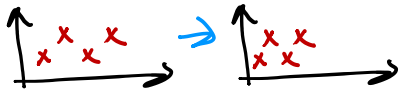
Stanford
ONLINE



PCA

PCA in Code

PCA in scikit-learn

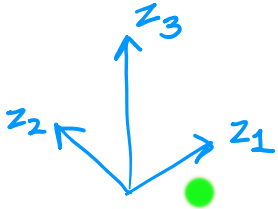


Optional pre-processing: Perform feature scaling

for visualization

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

   `fit` includes mean normalization



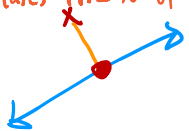
info

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

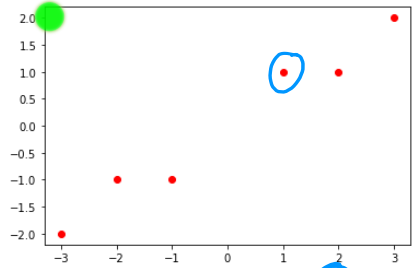
   `explained_variance_ratio` E.g., if it's 0.992, it means the 1st principal component captures 99.2% of information.

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]])
```

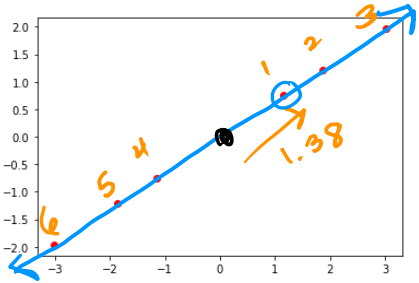
2D

```
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)
```

k=1 Only 1 principle components

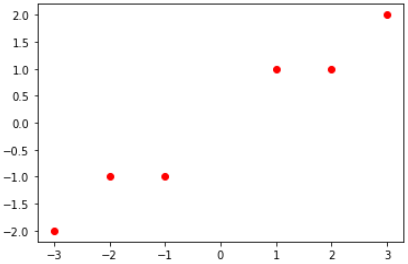
0.992

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



1D

Example



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

2D

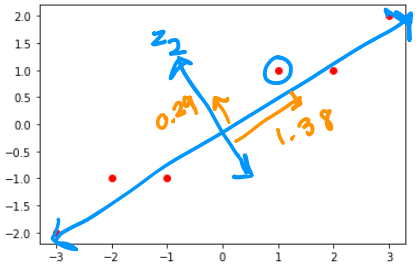
```
pca_2 = PCA(n_components=2)
pca_2.fit(X)
pca_2.explained_variance_ratio_
X_trans_2 = pca.transform(X)
X_reduced_2 = pca.inverse_transform(X_trans_2)
```

k = 2 Only 2 principle components

z_1 z_2
0.992 0.008

z_1 z_2

```
array([
[ 1.38340578,  0.2935787 ],
[ 2.22189802, -0.25133484],
[ 3.6053038 ,  0.04224385],
[-1.38340578, -0.2935787 ],
[-2.22189802,  0.25133484],
[-3.6053038 , -0.04224385]])
```



2D

0.992 + 0.002 = 1, which means z_1 and z_2 together explain 100% of data, which makes sense because X is only 2-D.

Applications of PCA

Note: Don't use PCA to prevent overfitting. Valuable information in data may be deprived due to PCA.

- Visualization reduce to 2 or 3 features

Less frequently used for:

- Data compression Andrew says it's not really often used nowadays because hardware improves a lot.
  (to reduce storage or transmission costs) 50 → 10
- Speeding up training of a supervised learning model

$n = 1000 \rightarrow 100$

The idea is by reducing feature's dimension, the dataset becomes smaller, which may make older generation of algorithms, such as Support Vector Machine, run faster. But SVM is not as good as neural networks, and neural networks work well even with lots of features. Thus, it's not necessary to reduce dimension of features.