

## Week 2

# Recommender Systems

Second to last week, almost done. Congrats!

Commercial impact and number of use cases is vastly greater than attention in academia. Every online shopping site uses it. A large fraction of sales is driven by recommender systems. Shopping sites, restaurants, shopping articles to show, ads, posts... It's used so widely.

**\*\*You could try making a movie recommender system, considering you already have the data.**

## NOTATION

**Predicting movie ratings**

User rates movies using one to five stars

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

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

**Ratings**

★				
★	★			
★	★	★		
★	★	★	★	
★	★	★	★	★

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

Remember: Some users have not voted and that's the key. We'll find movies that they haven't rated, and predict a rating for that film, considering that person.

DeepLearning.AI   Stanford ONLINE
Andrew Ng

There will be two scenarios where we'll work on:

- 1) We have access to features about movies: romance, action...
- 2) We don't have these features.

## Using per-item features

**What if we have features of the movies?**

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

$n_u = 4$   
 $n_m = 5$   
 $n = 2$

For user 1: Predict rating for movie  $i$  as:  $w^{(1)} \cdot x^{(i)} + b^{(1)}$       just linear regression

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

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

DeepLearning.AI   Stanford ONLINE
Andrew Ng

## Cost Function

Idea: For every film user has rated, how wrong are your predictions? Can use MSE.

For user  $j$  and movie  $i$ , predict rating:  $w^{(j)} \cdot x^{(i)} + b^{(j)}$   
 $\rightarrow m^{(j)} = \text{no. of movies rated by user } j$   
 To learn  $w^{(j)}, b^{(j)}$

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

Remember to add the reg. term to prevent overfitting.

**\*\*In recommender systems, it'll be convenient to eliminate  $m_j$  term, which you divide the cost to. It's just a constant,  $w$  and  $b$  will converge to same values if they can.**

To get total cost for all users, just sum this cost function for all users from  $j=1$  to  $n_u$ .

**NEXT UP: What if we don't have those features?**

## Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$x_1$ (romance)	$x_2$ (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

Imagine you have ratings and parameters. Then you could learn the values of features, knowing that  $w \cdot x + b$  will give you approximately the ratings. This gives you equations to solve. (if you had only 1 user, you wouldn't have enough information to find values of  $x$ )

$$\left. \begin{aligned} w^{(1)} &= \begin{bmatrix} 5 \\ 0 \end{bmatrix}, w^{(2)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}, w^{(3)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}, w^{(4)} = \begin{bmatrix} 0 \\ 5 \end{bmatrix} \\ b^{(1)} &= 0, b^{(2)} = 0, b^{(3)} = 0, b^{(4)} = 0 \end{aligned} \right\} \text{ using } w^{(j)} \cdot x^{(i)} + b^{(j)}$$

$$\left. \begin{aligned} w^{(1)} \cdot x^{(1)} &\approx 5 \\ w^{(2)} \cdot x^{(1)} &\approx 5 \\ w^{(3)} \cdot x^{(1)} &\approx 0 \\ w^{(4)} \cdot x^{(1)} &\approx 0 \end{aligned} \right\} \rightarrow x^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

**Cost function for deriving feature values:**

For that film, loop over every people that watched that movie and compute MSE with regularization term. This time  $w$  and  $b$  are fixed,  $x$  changes. So regularization term includes  $x$ 's.

## Cost function

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

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

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

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

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

**Problem: Where do you get these parameters from?**

## COLLABORATIVE FILTERING ALGORITHM

We'll do these two actions at the same time. Realize that they are summing through same examples (every movie that has been rated) and computing MSE.

### Collaborative filtering

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

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

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

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

Put them together:

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

	j=1	j=2	j=3
	Alice	Bob	Carol
Movie1	5	5	?
Movie2	?	2	3

Now work with gradient descent to minimize  $w$ ,  $b$  and  $x$ .

Minimize  $x$  the same way you're minimizing  $w$  and  $b$ .

$$X_{k+1} = X_k - lr * dj\_dx\_i$$

### BINARY LABELS: favs, likes and clicks

Meanings of signs:

1: user engaged with that item

0: user didn't engage with that item

?: user hasn't seen it yet

Just like distinction between linear regression and logistic regression, we'll apply sigmoid the function's output to get probability as an answer.

**\*\*This changes cost function as well. Instead of MSE, we'll use cross binary entropy.**

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

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

Loss for single example

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

## Mean Normalization

Your algorithm will work better if you do normalization on movie ratings. It'll help make better predictions as well.

If you predict Eve's ratings on the right,  $w$  and  $b$  will be as shown. Because everything in  $r(i, j)$  is 0 and first term in cost function doesn't get affected. Cost only tries to minimize parameters and they become 0. **So if a user hasn't seen any movies, you predict 0 for every movie, for that person.** asd

$$MN = (a - \mu) / \text{std.dev}$$

Here we'll just subtract  $\mu$ .

**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	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

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

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

**REMEMBER:** Because you subtracted mean of every movie, when making a prediction you need to add that movie's average rating to  $w \cdot x + b$ .

This way, Eve is not predicted as 0, just average ratings.

Also optimization runs a bit faster.

You could, in some scenarios or problems, mean with columns. An example is a movie just coming out and noone has seen it yet.

## Mean Normalization

$$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \begin{matrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{matrix} \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \quad \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:

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

User 5 (Eve):

$$w^{(5)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad b^{(5)} = 0 \quad \underbrace{w^{(5)} \cdot x^{(i)} + b^{(5)}}_0 + \mu_i = 2.5$$

## Tensorflow For Coll. Filt.

After you choose cost function, tensorflow automatically figures out derivatives to do optimization. (sometimes partial derivative can be complicated)



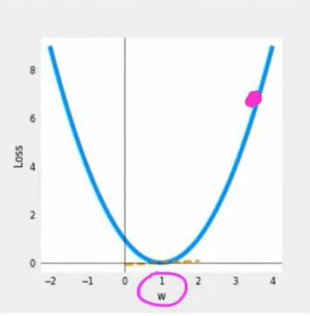
Some tensorflow code first: Pay attention to “tape” and “tape.gradient”. Also see how tf.Variable() can not be assigned directly and .assign\_add is used.

$J = (wx - 1)^2$

Gradient descent algorithm  
Repeat until convergence

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

Fix b = 0 for this example



## Custom Training Loop

```

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 GradientTape 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

DeepLearning.AI
Stanford ONLINE
Andrew Ng

And below is shown workflow of Adam optimizer.

## Implementation in TensorFlow

Gradient descent algorithm  
Repeat until convergence

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

```

# 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]))
        
```

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

DeepLearning.AI
Stanford ONLINE
Andrew Ng

\*\*\*This problem doesn't fit into a dense layer or a standard neural network. So we use a single optimizer.

## Finding Related Items

Collaborative filtering gives you a good way to do so.

\*When you use co.fi. to find features automatically, they are quite hard to interpret. They don't mean a lot by their own. But they collectively convey what that movie is like. So what you can do is to find similar items.

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

For a distance score, we'll use squared distance between two vectors. This value shows how similar they are and you can just pick 10 items with smallest distances.

**PROBLEMS:**

## 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:

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

- 1) Cold start problem: A new item or user has insufficient data.
- 2) You don't really benefit from side information on items or users but they're actually so important. Even knowing user's web browser tells a lot about their behaviour. THESE ARE IMPORTANT.

**NOTE:** Content based filtering addresses many of these problems and is state-of-the-art technique.

\*\*\*Lastly, code version of cost function is shown below.

```

def cofi_cost_func(X, W, b, Y, R, lambda_):
    """
    Returns the cost for the content-based filtering
    Args:
        X (ndarray (num_movies,num_features)): matrix of item features
        W (ndarray (num_users,num_features)) : matrix of user parameters
        b (ndarray (1, num_users)           : vector of user parameters
        Y (ndarray (num_movies,num_users)   : matrix of user ratings of movies
        R (ndarray (num_movies,num_users)   : matrix, where R(i, j) = 1 if the
        lambda_ (float): regularization parameter
    Returns:
        J (float) : Cost
    """
    nm, nu = Y.shape
    J = 0
    ### START CODE HERE ###

    for i in range(nu): #Every user has their own parameters
        w = W[i, :]
        for j in range(nm):
            J += R[j, i]*(np.dot(w, X[j]) + b[0, i] - Y[j, i])**2 / 2

    J += (lambda_/2) * (np.sum(np.square(W)) + np.sum(np.square(X)))

    ### END CODE HERE ###

    return J

```

THOUGH VECTORIZED IMPLEMENTATION IS MUCH BETTER AS ALWAYS. It's in the lab.

## CONTENT-BASED FILTERING ALGORITHM

First, a comparison: It requires match in user's and item's features. But previous rating procedure stays the same (we have  $r$  and  $y$ ).

### → Collaborative filtering:

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

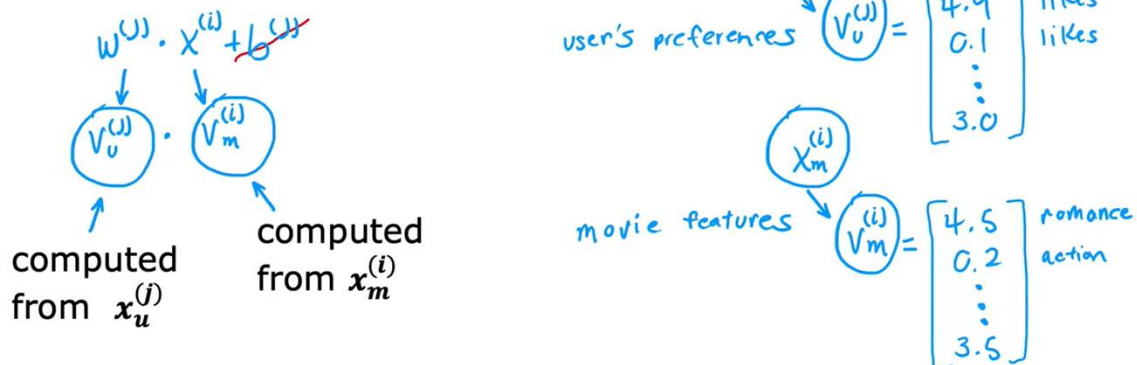
### Content-based filtering:

Recommend items to you based on features of user and item to find good match

You have  $i$ .th movie and  $j$ .th user and their features. With these features, your objective is to **predict whether that person will love that movie**. Keep in mind that these features could have very different sizes. A user can have 1500 features while a movie only has 50.

## Content-based filtering: Learning to match

Predict rating of user  $j$  on movie  $i$  as



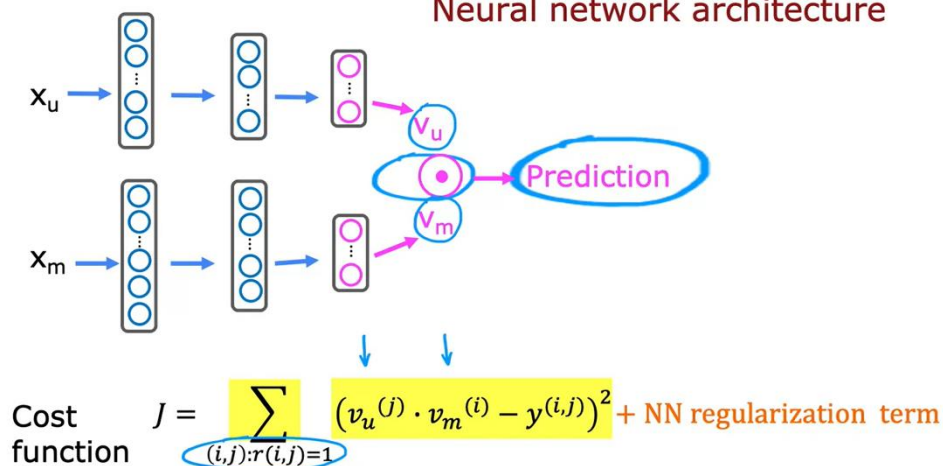
To summarize this image:

From features of the user and the item, you “obtain” two vectors. Dot product of these two vectors give you the prediction (if label is binary, just use sigmoid to estimate probability.). **But how do we obtain these vectors?** A note, two features can be of different sizes but dimensions of two vectors need to be the same.

\*You can develop content-based filtering with deep learning.

Just like how token embeddings are acquired in NLP, process of acquiring the vector from features involves a neural network. User network and movie network, in this case, can have different number of layers and neurons but their output will have the same dimensions. **But to do this (meaning train the parameters), you need to have an objective, just like “next word prediction” in token embeddings.** For that we’ll create a cost function. This cost function will measure how far our predictions are from known examples.

### Neural network architecture





Also, it turns out that by doing this you could find related items as well. Each vector describes a movie. You could get distance to other vectors, to measure how similar different movies are.

To find movies similar to movie  $i$ :  $\|v_m^{(k)} - v_m^{(i)}\|^2$  small

Note: This can be pre-computed. Meaning you could run the server at night to be ready to recommend 10 similar movies for every movie. This'll be important later when movie catalogue is large. Because when you have a lot of movies, this process becomes so computationally expensive.

## Recommending from a large catalogue

Today's systems recommend from tens of millions choices (like adds or songs). If every time a user shows up, you try to compute items, you'll see that it's not feasible (not in a pleasant way though). So:

- 1) Retrieval: Create a large set that user may like.

### Two steps: Retrieval & Ranking

Retrieval:

- • Generate large list of plausible item candidates ~100s  
e.g.
  - 1) For each of the last 10 movies watched by the user, find 10 most similar movies  
 $\|v_m^{(k)} - v_m^{(i)}\|^2$
  - 2) For most viewed 3 genres, find the top 10 movies
  - 3) Top 20 movies in the country
- Combine retrieved items into list, removing duplicates and items already watched/purchased

It's okay to contain some not-worth ones in this step. Your objective is to get a broad representation that includes good examples.

**1, 2 and 3 can all be pre-computed.**

- 2) Ranking: Take the list from "retrieval" step and rank using your model. Then just display ranked items. **Neural network that you use in this process is obtained from before.** Also movie vectors can be pre-computed. All these make this process faster.

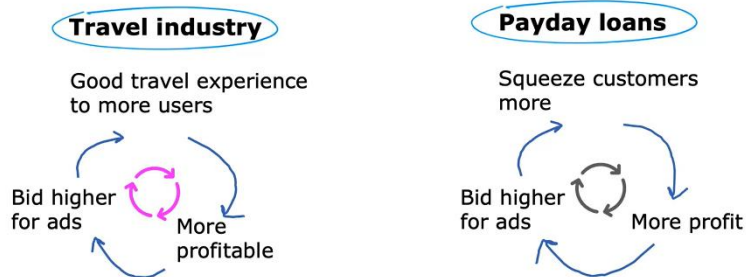
- • Retrieving more items results in better performance, but slower recommendations.
- • To analyse/optimize the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations (i.e.,  $p(y^{(i,j)}) = 1$  of items displayed to user are higher).

**With great power, comes great responsibility.**

What we carry out decides what a user will see.

Example: Recommend not what you're likely to buy but what makes most profit.

### Ethical considerations with recommender systems



Two possible cycles are shown in the image. They are both flattered by a recommender system but only one benefits people.

Ask yourself: Is it good?

### 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?

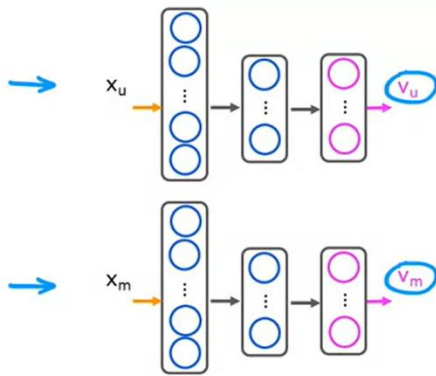
Amelioration : Be transparent with users

Some other problems...

**It's not just a moral aspect, it affects the society.**

### Implementing Co.Ba.Fi. in Tensorflow

- 1) Have two sequential models, for user and item.
- 2) Tell keras how to feed user and item features to neural networks.
- 3) Normalize.



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

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

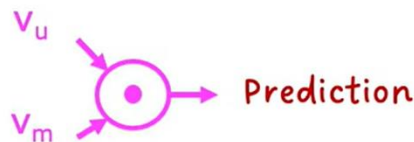
```
# create the user input and point to the base network
input_user = tf.keras.layers.Input(shape=(num_user_features))
vu = user_NN(input_user)
vu = tf.linalg.l2_normalize(vu, axis=1)
```

```
# create the item input and point to the base network
input_item = tf.keras.layers.Input(shape=(num_item_features))
vm = item_NN(input_item)
vm = tf.linalg.l2_normalize(vm, axis=1)
```

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

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

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



## Practice Lab

"""

In the collaborative filtering lab, you generated two vectors, a user vector and an item/movie vector whose dot product would predict a rating. The vectors were derived solely from the ratings.

Content-based filtering also generates a user and movie feature vector but recognizes there may be other information available about the user and/or movie that may improve the prediction. The additional information is provided to a neural network which then generates the user and movie vector

"""

- Scikit learn → `MinMaxScaler()`: Scales values between -1 and 1.

To allow us to evaluate the results, we will split the data into training and test sets as was discussed in Course 2, Week 3. Here we will use [sklearn train\\_test\\_split](#) to split and shuffle the data. Note that setting the initial random state to the same value ensures item, user, and y are shuffled identically.

```
item_train, item_test = train_test_split(item_train, train_size=0.80, shuffle=True, random_state=1)
user_train, user_test = train_test_split(user_train, train_size=0.80, shuffle=True, random_state=1)
y_train, y_test = train_test_split(y_train, train_size=0.80, shuffle=True, random_state=1)
print(f"movie/item training data shape: {item_train.shape}")
print(f"movie/item test data shape: {item_test.shape}")
```

```
# scale training data
item_train_unscaled = item_train
user_train_unscaled = user_train
y_train_unscaled    = y_train

scalerItem = StandardScaler()
scalerItem.fit(item_train)
item_train = scalerItem.transform(item_train)

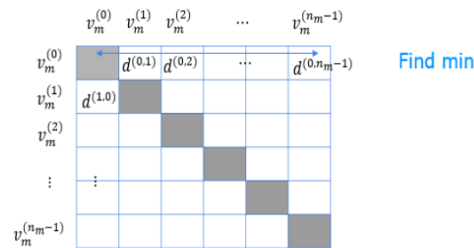
scalerUser = StandardScaler()
scalerUser.fit(user_train)
user_train = scalerUser.transform(user_train)

scalerTarget = MinMaxScaler((-1, 1))
scalerTarget.fit(y_train.reshape(-1, 1))
y_train = scalerTarget.transform(y_train.reshape(-1, 1))
#ynorm_test = scalerTarget.transform(y_test.reshape(-1, 1))

print(np.allclose(item_train_unscaled, scalerItem.inverse_transform(item_train)))
print(np.allclose(user_train_unscaled, scalerUser.inverse_transform(user_train)))
```

Here a way to scale data is shown.

Let's now compute a matrix of the squared distance between each movie feature vector and all other movie feature vectors:



We can then find the closest movie by finding the minimum along each row. We will make use of [numpy masked arrays](#) to avoid selecting the same movie. The masked values along the diagonal won't be included in the computation.

```
count = 50 # number of movies to display
dim = len(vms)
dist = np.zeros((dim,dim))

for i in range(dim):
    for j in range(dim):
        dist[i,j] = sq_dist(vms[i, :], vms[j, :])

m_dist = ma.masked_array(dist, mask=np.identity(dist.shape[0])) # mask the diagonal

disp = [["movie1", "genres", "movie2", "genres"]]
for i in range(count):
    min_idx = np.argmin(m_dist[i])
    movie1_id = int(item_vecs[i,0])
    movie2_id = int(item_vecs[min_idx,0])
    disp.append( [movie_dict[movie1_id]['title'], movie_dict[movie1_id]['genres'],
                  movie_dict[movie2_id]['title'], movie_dict[movie2_id]['genres']] )

disp.append( [movie_dict[movie1_id]['title'], movie_dict[movie1_id]['genres'],
              movie_dict[movie2_id]['title'], movie_dict[movie2_id]['genres']] )
table = tabulate.tabulate(disp, tablefmt='html', headers="firstrow")
table
```

movie1	genres	movie2	genres
Save the Last Dance (2001)	Drama Romance	Mona Lisa Smile (2003)	Drama Romance
Wedding Planner, The (2001)	Comedy Romance	Sweetest Thing, The (2002)	Comedy Romance
Hannibal (2001)	Horror Thriller	Final Destination 2 (2003)	Horror Thriller
Saving Silverman (Evil Woman) (2001)	Comedy Romance	Wedding Planner, The (2001)	Comedy Romance
Down to Earth (2001)	Comedy Fantasy Romance	Bewitched (2005)	Comedy Fantasy Romance
Mexican, The (2001)	Action Comedy	Rush Hour 2 (2001)	Action Comedy
15 Minutes (2001)	Thriller	Panic Room (2002)	Thriller
Enemy at the Gates (2001)	Drama	8 Mile (2002)	Drama

- Notice that recommended movies have the same genres. This shows you've done what you were aiming.

## PCA: Principal Component Analysis

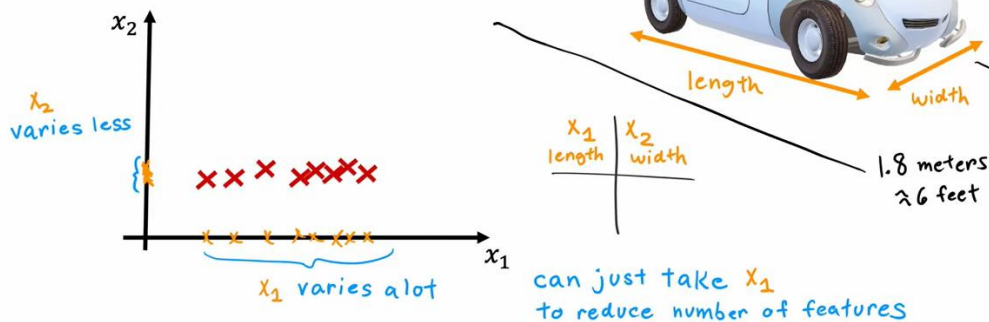
Like TSN-E, it can be used to reduce dimensions. How?

It's actually an unsupervised learning algorithm.

- Problem: Our features have thousands of dimensions. You can not possibly plot a 1000 dimensional plot.
- Solution: Use unsup. lear. to reduce dimensions to two or three.



## Car measurements



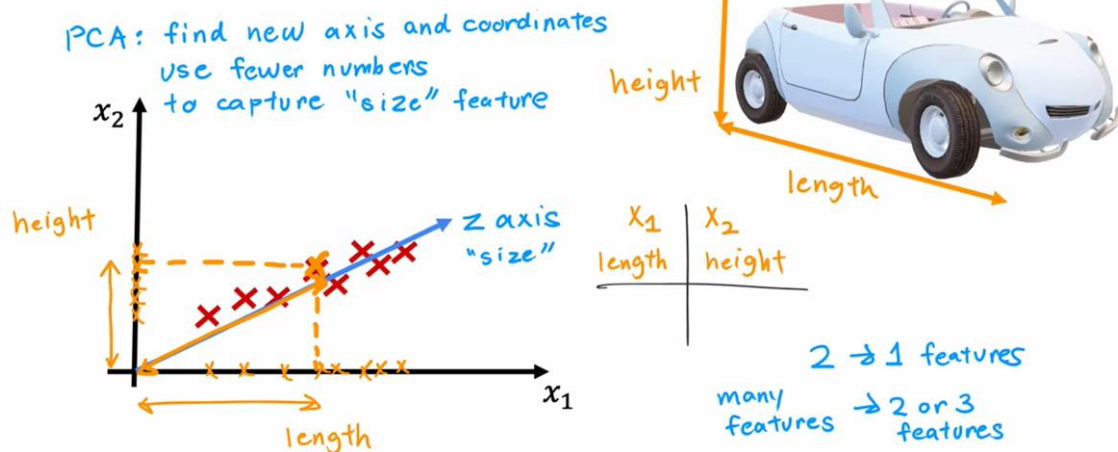
### Example 1:

When applied to this dataset, PCA can see how variables behave and choose to take just  $x_1$  in this scenario.

Here it's simple because only 1 variable has a meaningful variation.

But it can do more than that.

## Size

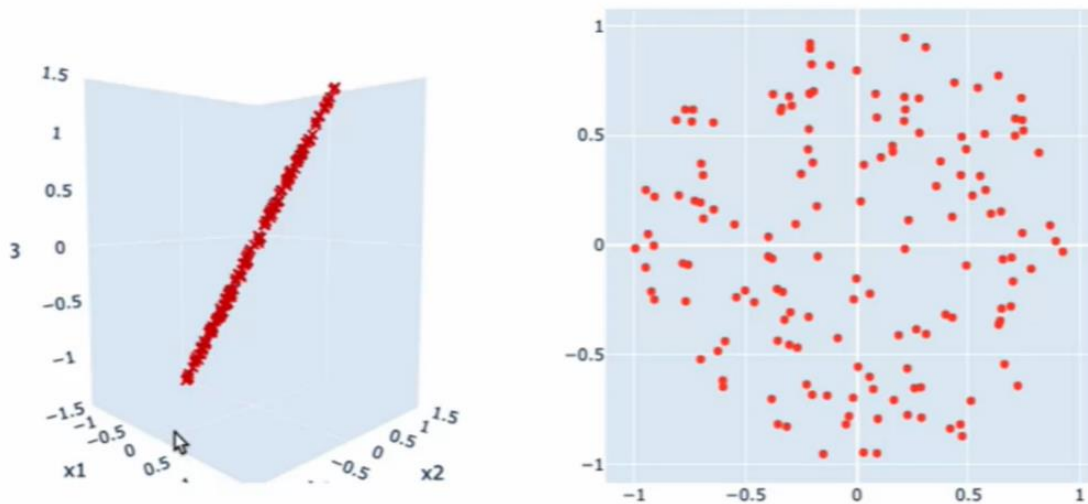


### Example 2:

A car has height and length and these two features both occupy an axis. PCA examines the dataset to come up with a new axis and new coordinates, while using fewer features. **It's like feature engineering in a way.**

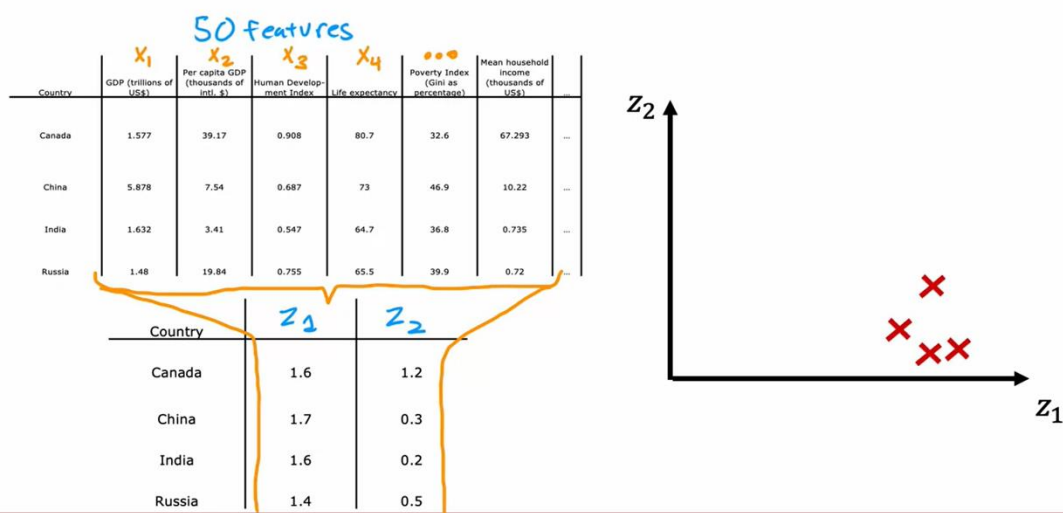
**Example 3:** If you had a 3D data, structured like a surface (Imagine an egg just broken into a pan, it's round but it's height is small. So its points are lined up in a thin surface when you look at it in a certain angle.) it can remove that thin layer dimension and give you a circle-like 2D data.

# From 3D to 2D



You had  $x_1$ ,  $x_2$  and  $x_3$  but now you have  $z_1$  and  $z_2$ .

## Example 4:



**New features have new meanings.**

Andrew: "When I do sth, I like to look at the data. What are countries like, what do cars look like, I like to see. Sometimes you catch unexpected things and patterns."

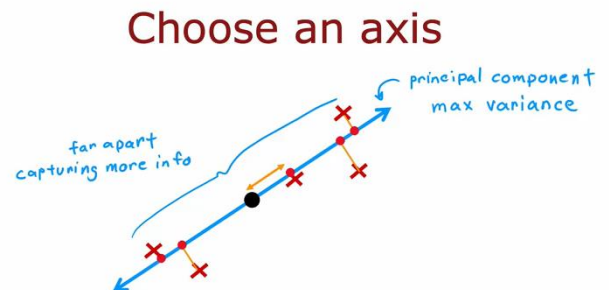
## PCA ALGORITHM

Decision can be thought as: How do you choose a new axis to capture properties of the data?

**NOTE:** Steps require pre-processing. You should normalize the data to have zero mean. **Also apply feature scaling if necessary.**

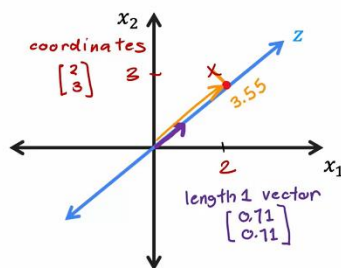
**Principal Component:** It's the axis in which if you select it you'll get maximum amount of variance in data points' projections. We aim to have high variance in these projections, because it means we're capturing information about the data.

If you decided a line perpendicular to one on right, projections would be squished and you'd lose information about data.



## HOW TO CONVERT TO THE NEW AXIS?

### Coordinate on the new axis



dot product

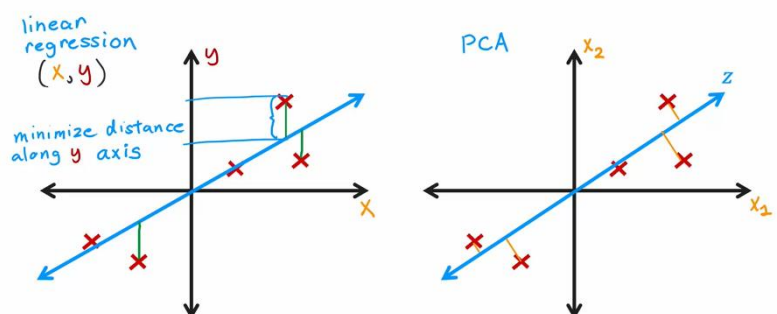
$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 0.71 \\ 0.71 \end{bmatrix}$$
$$2 \times 0.71 + 3 \times 0.71 = 3.55$$

**Note:** Second and third principal components will be perpendicular to previous ones.

PCA IS NOT LINEAR REGRESSION. IT'S A TOTALLY DIFFERENT ALGORITHM. There is not a label, just features. You're also not trying to fit a line.

In 2 features, they might look similar but not in high dimensions. They give totally different answers.

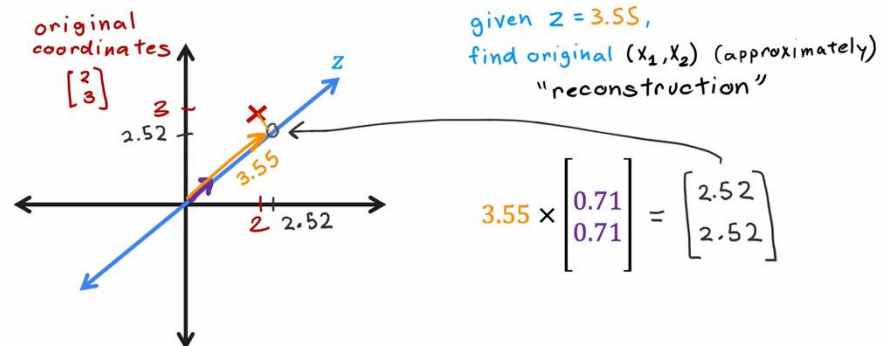
### PCA is not linear regression



**NOTE:** While maximizing variance in projections, you are actually trying to minimize a cost function. It's distances of points to the new axis.

There is one more thing PCA does and it's **reconstruction**. It means trying to approximate the original point from it's projection.

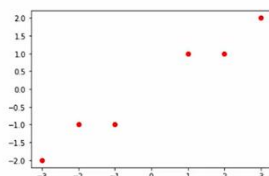
## Approximation to the original data



## PCA in code

We'll use scikit-learn. Steps:

- Optional pre-processing: Perform feature scaling.
- "fit" the data to obtain 2 or 3 principal components (new axes).
  - **Fit function includes mean normalization.**
- Optional examination of how much variance is gotten from pc's.
  - This is done with **explained\_variance\_ratio**.
- Transform the data.
  - With **transform**.

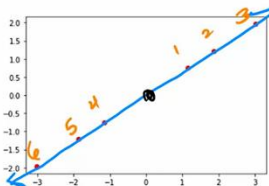


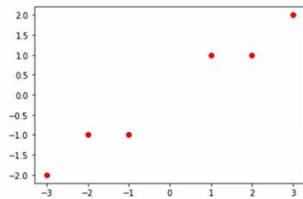
## Example

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

```
pca_1 = PCA(n_components=1)
pca_1.fit(X)
pca_1.explained_variance_ratio_ 0.992
X_trans_1 = pca_1.transform(X)
X_reduced_1 = pca_1.inverse_transform(X_trans_1)
```

```
array([
 1 [ 1.38340578],
 2 [ 2.22189802],
 3 [ 3.6053038 ],
 4 [-1.38340578],
 5 [-2.22189802],
 6 [-3.6053038 ]])
```





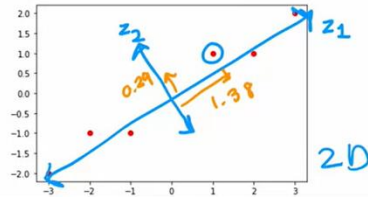
## Example

```
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)
```

$z_1$  0.992  $z_2$  0.008



```
→ 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]])
```

Here we tried to go from 2 dimensions to 2 dimensions. Data points stayed the same, only the axes and therefore their numerical values changed. But their alignment stayed exactly the same.

\*\*PCA is used for visualization. In past it was popularly used for data compression. Today with modern methods it's not that used. Same for speeding up training supervised learning. This used to make a difference in, for example, support vector machines. But today with modern deep learning, it's not used.

```
X_trans_2 = pca_2.transform(X)
X_trans_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]])
```

Think of column 1 as the coordinate along the first principal component (the first new axis) and column 2 as the coordinate along the second principal component (the second new axis).

You can probably just choose the first principal component since it retains 99% of the information (explained variance).

```
pca_1 = PCA(n_components=1)
pca_1

PCA(n_components=1)

pca_1.fit(X)
pca_1.explained_variance_ratio_

array([0.99244289])
```

```
X_trans_1 = pca_1.transform(X)
X_trans_1

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

Notice how this column is just the first column of `X_trans_2`.

This image is from PCA lab. Notice that:



- First rows of 1D and 2D forms are the same. It's because PCA computes first optimal principal component, than 2nd and maybe 3rd axes are perpendicular to first one (to gain more info).