Subject: / /

Recommender systems:

n_u : No. of users

n_m : No. of movies

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

$y^{(i,j)}$: Rating of user j to movie i

$w^{(j)}, b^{(j)}$: parametres for user j

$x^{(i)}$: Feature vector for movie i

$m^{(j)}$: No. of movies rated by user j

$f(x) = w^{(1)} x^{(i)} + b^{(1)}$

Movie	Rating	x_1 (Romance)	x_2 (action)
1	5	0.9	0
2	5	1	0.01
3	?	0.99	0

Cost function:

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

Collaborative filtering algorithm:

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

- Reverse engineer features from w & b.
- Given $w^{(1)}, b^{(1)}, \ldots, w^{(n_u)}, b^{(n_u)}$ to learn $x^{(1)}, \ldots, x^{(n_m)}$:

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

Subject: / /

Add ① + ②:

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

Gradient descent:

Repeat {

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

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

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

}

Binary labels:

1 - Engaged after being shown

0 - Did not engage after being shown

? - Item not yet shown

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

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

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

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

Subject: / /

Mean normalization:

Normalise rows ↑

$\vec{x} = \begin{matrix} \text{movie 1} \\ \text{movie 2} \end{matrix}\begin{bmatrix} & \end{bmatrix}$ $\quad \mu = \begin{bmatrix} 2.5 \\ 1.25 \end{bmatrix}$ $\quad \vec{x} - \begin{bmatrix} \vec{x} - \mu \\ \vec{x} - \mu \end{bmatrix}$

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

Tensorflow implementation:

Auto diff/ Grad in Tensorflow:

```
# 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]) )
```

Auto-diff/ Grad custom:

```
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)
```

Subject: / /

Finding related items:

- Find item k with x^k similar to $x^{(i)}$

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

Limits:

- How to rank new items.
- Show something reasonable to new users

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

User feature:

- Age, Gender, country, Movies watched, Avg rating per genre $\} x_u^{(j)}$

Movie features:

- Year, Genre/Genres, Reviews, Average rating $\} x_m^{(i)}$

user's preferences ↑

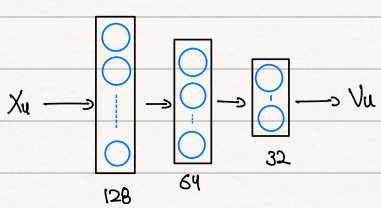
$v_u^{(j)} \cdot v_m^{(j)}$ → movie features

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

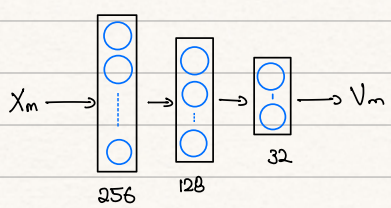
$x_u \rightarrow v_u$ $x_m \rightarrow v_m$

Deep learning for CBF:

User network:



Movie network:



Subject: / /

Prediction: $V_u \cdot V_m$

$g(V_u \cdot V_m)$ → for binary predictions

Cost function:

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

→ $V_u^{(j)}$ 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: $\| V_m^{(k)} - V_m^{(i)} \|^2 \Rightarrow$ small

Recommending from large catalog:

Retrieval:

- Generate large list of plausible item candidates

  eg:

  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 top 10 movies

  3) Top 20 movies in the country.

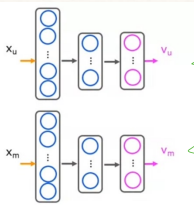
- Combine the retrieval & remove duplicates | watched

Ranking:

- Take list retrieved & rank using learned model
- Display list

Subject: / /

Tensor flow implementation:



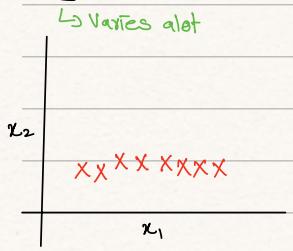

```
user_NN = tf.keras.models.Sequential([
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(32)
])

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 )
])

# 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) -> have length one of vector

# 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) -> have length one of vector

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

# specify the inputs and output of the model
model = Model([input_user, input_item], output) -> Tells keras what are the inputs & outputs

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

Principal component analysis:

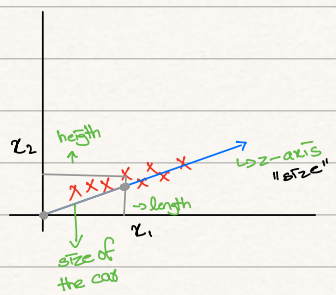
Find new coordinates and axis

Reducing the number of features:

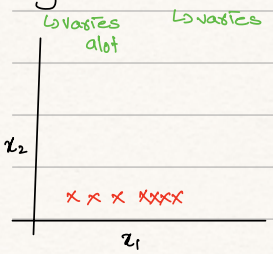
x_1 length x_2 width → varies not at lot
↳ Varies alot



x_1 length x_2 height

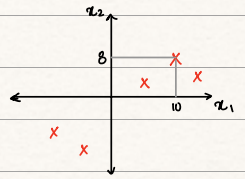


x_1 length x_2 Diametre
↳varies alot ↳varies less



Subject: / /

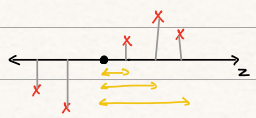
PCA Algorithm:



Pre-process features:

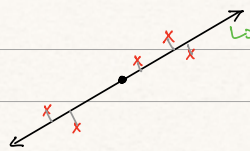
- Normalize to have zero mean.
- Be on same scale — feature scaling

Option 1:



"project" to z-axis

option 3:



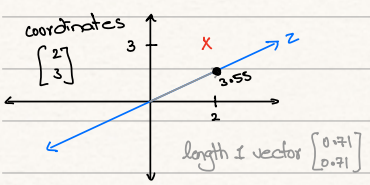
1 principal component
1 on the axis, we have max variance

- Better than last 2
- Easily see how much points vary

Option 2:

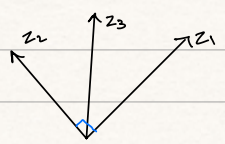
- Data is squished
- Bad choice

Coordinate on new axis:



NOTE:

The 2nd axis will be at 90° to the PCA.
The 3rd will be at 90° to both previous axis



Dot product $\begin{bmatrix}2\\3\end{bmatrix} \cdot \begin{bmatrix}0.71\\0.71\end{bmatrix} = 3.55$

PCA is NOT Linear Regression.

Subject: / /

Given $z = 3.55$

find original (x_1, x_2) — approx

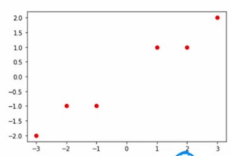
"Reconstruction"

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

PCA in scikit-learn:

- Perform feature scaling
- fit data to 2 (or 3) new axes
- examine how much variance explained by each component
- Transform (project) the data on new axes

Example



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

```
pca_1 = PCA(n_components=1)   # no. of components
pca_1.fit(X)
pca_1.explained_variance_ratio_   0.992
X_trans_1 = pca_1.transform(X)
X_reduced_1 = pca.inverse_transform(X_trans_1)
```

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