Practice lab: Collaborative Filtering Recommender Systems

In this exercise, you will implement collaborative filtering to build a recommender system for movies.

Outline

***NOTE:** To prevent errors from the autograder, you are not allowed to edit or delete non-graded cells in this lab. Please also refrain from adding any new cells. **Once you have passed this assignment** and want to experiment with any of the non-graded code, you may follow the instructions at the bottom of this notebook.*

Packages

We will use the now familiar NumPy and Tensorflow Packages.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from recsys_utils import *
```

1 - Notation

|General Notation | Description| Python (if any) |
|:-------------|:--|| | $r(i,j)$ | scalar; = 1 if user j rated movie i = 0 otherwise || | $y(i,j)$ | scalar; = rating given by user j on movie i (if r(i,j) = 1 is defined) || |$w^{(j)}$ | vector; parameters for user j || |$b^{(j)}$ | scalar; parameter for user j || | $x^{(i)}$ | vector; feature ratings for movie i ||
| n_u | number of users |num_users| | n_m | number of movies | num_movies | | n | number of

features | num_features | | X | matrix of vectors $x^{(i)}$ | X | | W | matrix of vectors $w^{(j)}$ | W | | b | vector of bias parameters $b^{(j)}$ | b | | R | matrix of elements $r(i,j)$ | R |

2 - Recommender Systems

In this lab, you will implement the collaborative filtering learning algorithm and apply it to a dataset of movie ratings. The goal of a collaborative filtering recommender system is to generate two vectors: For each user, a 'parameter vector' that embodies the movie tastes of a user. For each movie, a feature vector of the same size which embodies some description of the movie. The dot product of the two vectors plus the bias term should produce an estimate of the rating the user might give to that movie.

The diagram below details how these vectors are learned.

Existing ratings are provided in matrix form as shown. Y contains ratings; 0.5 to 5 inclusive in 0.5 steps. 0 if the movie has not been rated. R has a 1 where movies have been rated. Movies are in rows, users in columns. Each user has a parameter vector w^{user} and bias. Each movie has a feature vector x^{movie}. These vectors are simultaneously learned by using the existing user/movie ratings as training data. One training example is shown above: $w^{(1)} \cdot x^{(1)} + b^{(1)} = 4$. It is worth noting that the feature vector x^{movie} must satisfy all the users while the user vector w^{user} must satisfy all the movies. This is the source of the name of this approach - all the users collaborate to generate the rating set.

Once the feature vectors and parameters are learned, they can be used to predict how a user might rate an unrated movie. This is shown in the diagram above. The equation is an example of predicting a rating for user one on movie zero.

In this exercise, you will implement the function `cofiCostFunc` that computes the collaborative filtering objective function. After implementing the objective function, you will use a TensorFlow custom training loop to learn the parameters for collaborative filtering. The first step is to detail the data set and data structures that will be used in the lab.

3 - Movie ratings dataset

The data set is derived from the MovieLens "ml-latest-small" dataset.
[F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872]

The original dataset has 9000 movies rated by 600 users. The dataset has been reduced in size to focus on movies from the years since 2000. This dataset consists of ratings on a scale of 0.5 to 5 in 0.5 step increments. The reduced dataset has $n_u = 443$ users, and $n_m = 4778$ movies.

Below, you will load the movie dataset into the variables Y and R.

The matrix Y (a $n_m \times n_u$ matrix) stores the ratings $y^{(i,j)}$. The matrix R is an binary-valued indicator matrix, where $R(i,j)=1$ if user j gave a rating to movie i, and $R(i,j)=0$ otherwise.

Throughout this part of the exercise, you will also be working with the matrices, X, W and b:

$$X=\begin{bmatrix} --- \left(x^{(0)}\right)^T --- \\ --- \left(x^{(1)}\right)^T --- \\ \vdots \\ --- \left(x^{(n_m-1)}\right)^T --- \end{bmatrix}, W=\begin{bmatrix} --- \left(w^{(0)}\right)^T --- \\ --- \left(w^{(1)}\right)^T --- \\ \vdots \\ --- \left(w^{(n_u-1)}\right)^T --- \end{bmatrix}, b=\begin{bmatrix} b^{(0)} \\ b^{(1)} \\ \vdots \\ b^{(n_u-1)} \end{bmatrix}$$

The i-th row of X corresponds to the feature vector $x^{(i)}$ for the i-th movie, and the j-th row of W corresponds to one parameter vector $w^{(j)}$, for the j-th user. Both $x^{(i)}$ and $w^{(j)}$ are n-dimensional vectors. For the purposes of this exercise, you will use $n=10$, and therefore, $x^{(i)}$ and $w^{(j)}$ have 10 elements. Correspondingly, X is a $n_m \times 10$ matrix and W is a $n_u \times 10$ matrix.

We will start by loading the movie ratings dataset to understand the structure of the data. We will load Y and R with the movie dataset.
We'll also load X, W, and b with pre-computed values. These values will be learned later in the lab, but we'll use pre-computed values to develop the cost model.

```
#Load data
X, W, b, num_movies, num_features, num_users =
load_precalc_params_small()
Y, R = load_ratings_small()

print("Y", Y.shape, "R", R.shape)
print("X", X.shape)
print("W", W.shape)
print("b", b.shape)
print("num_features", num_features)
print("num_movies",   num_movies)
print("num_users",    num_users)
```

```
Y (4778, 443) R (4778, 443)
X (4778, 10)
W (443, 10)
b (1, 443)
num_features 10
num_movies 4778
num_users 443
```

```
#  From the matrix, we can compute statistics like average rating.
tsmean =  np.mean(Y[0, R[0, :].astype(bool)])
print(f"Average rating for movie 1 : {tsmean:0.3f} / 5" )
```

```
Average rating for movie 1 : 3.400 / 5
```

4 - Collaborative filtering learning algorithm

Now, you will begin implementing the collaborative filtering learning algorithm. You will start by implementing the objective function.

The collaborative filtering algorithm in the setting of movie recommendations considers a set of n-dimensional parameter vectors $x^{(0)}, \ldots, x^{(n_m-1)}$, $w^{(0)}, \ldots, w^{(n_u-1)}$ and $b^{(0)}, \ldots, b^{(n_u-1)}$, where the model predicts the rating for movie i by user j as $y^{(i,j)} = w^{(j)} \cdot x^{(i)} + b^{(j)}$. Given a dataset that consists of a set of ratings produced by some users on some movies, you wish to learn the parameter vectors $x^{(0)}, \ldots, x^{(n_m-1)}, w^{(0)}, \ldots, w^{(n_u-1)}$ and $b^{(0)}, \ldots, b^{(n_u-1)}$ that produce the best fit (minimizes the squared error).

You will complete the code in cofiCostFunc to compute the cost function for collaborative filtering.

4.1 Collaborative filtering cost function

The collaborative filtering cost function is given by

$$J\left(x^{(0)}, \ldots, x^{(n_m-1)}, w^{(0)}, b^{(0)}, \ldots, w^{(n_u-1)}, b^{(n_u-1)}\right) = \left[\frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2\right) + \left[\frac{\lambda}{2} \sum_{j=0}^{n_u-1} \sum_{k=0}^{n-1} \left(w_k^{(j)}\right)^2 + \frac{\lambda}{2} \sum_{i=0}^{n_m-1} \sum_{k=0}^{n-1} \left(x_k^{(i)}\right)^2\right)$$

The first summation in (1) is "for all i, j where $r(i,j)$ equals 1" and could be written:

$$¿\left[\frac{1}{2} \sum_{j=0}^{n_u-1} \sum_{i=0}^{n_m-1} r(i,j) * \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2\right) + \text{regularization}$$

You should now write cofiCostFunc (collaborative filtering cost function) to return this cost.

Exercise 1

For loop Implementation:
Start by implementing the cost function using for loops. Consider developing the cost function in two steps. First, develop the cost function without regularization. A test case that does not include regularization is provided below to test your implementation. Once that is working, add regularization and run the tests that include regularization. Note that you should be accumulating the cost for user j and movie i only if $R(i,j) = 1$.

```
# GRADED FUNCTION: cofi_cost_func
# UNQ_C1

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 i-th movies was rated by the j-th user
      lambda_ (float): regularization parameter
    Returns:
      J (float) : Cost
    """
    nm, nu = Y.shape
    J = 0
    ### START CODE HERE ###
    for j in range(nu):
        w = W[j,:]
        b_j = b[0,j]
        for i in range(nm):
            x = X[i,:]
            y = Y[i,j]
            r = R[i,j]
            J += r * np.square((np.dot(w,x) + b_j - y ))
    J += (lambda_) * (np.sum(np.square(W)) + np.sum(np.square(X)))
    J = J/2
    ### END CODE HERE ###

    return J
```

Click for hints You can structure the code in two for loops similar to the summation in (1).
Implement the code without regularization first.
Note that some of the elements in (1) are vectors. Use np.dot(). You can also use np.square(). Pay close attention to which elements are indexed by i and which are indexed by j. Don't forget to divide by two.

```
    ### START CODE HERE ###
    for j in range(nu):


        for i in range(nm):


    ### END CODE HERE ###

# Reduce the data set size so that this runs faster
num_users_r = 4
num_movies_r = 5
num_features_r = 3
```

```
X_r = X[:num_movies_r, :num_features_r]
W_r = W[:num_users_r,  :num_features_r]
b_r = b[0, :num_users_r].reshape(1,-1)
Y_r = Y[:num_movies_r, :num_users_r]
R_r = R[:num_movies_r, :num_users_r]

# Evaluate cost function
J = cofi_cost_func(X_r, W_r, b_r, Y_r, R_r, 0);
print(f"Cost: {J:0.2f}")
```

```
Cost: 13.67
```

Expected Output (lambda = 0):
13.67.

```
# Evaluate cost function with regularization
J = cofi_cost_func(X_r, W_r, b_r, Y_r, R_r, 1.5);
print(f"Cost (with regularization): {J:0.2f}")
```

```
Cost (with regularization): 28.09
```

Expected Output:

28.09

```
# Public tests
from public_tests import *
test_cofi_cost_func(cofi_cost_func)
```

```
All tests passed!
```

Vectorized Implementation

It is important to create a vectorized implementation to compute J, since it will later be called many times during optimization. The linear algebra utilized is not the focus of this series, so the implementation is provided. If you are an expert in linear algebra, feel free to create your version without referencing the code below.

Run the code below and verify that it produces the same results as the non-vectorized version.

```
def cofi_cost_func_v(X, W, b, Y, R, lambda_):
    """
    Returns the cost for the content-based filtering
    Vectorized for speed. Uses tensorflow operations to be compatible
with custom training loop.
    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 i-th movies was rated by the j-th user
      lambda_ (float): regularization parameter
    Returns:
      J (float) : Cost
    """
    j = (tf.linalg.matmul(X, tf.transpose(W)) + b - Y)*R
    J = 0.5 * tf.reduce_sum(j**2) + (lambda_/2) * (tf.reduce_sum(X**2)
+ tf.reduce_sum(W**2))
    return J

# Evaluate cost function
J = cofi_cost_func_v(X_r, W_r, b_r, Y_r, R_r, 0);
print(f"Cost: {J:0.2f}")

# Evaluate cost function with regularization
J = cofi_cost_func_v(X_r, W_r, b_r, Y_r, R_r, 1.5);
print(f"Cost (with regularization): {J:0.2f}")
```

```
Cost: 13.67
Cost (with regularization): 28.09
```

Expected Output:
Cost: 13.67
Cost (with regularization): 28.09

5 - Learning movie recommendations

After you have finished implementing the collaborative filtering cost function, you can start training your algorithm to make movie recommendations for yourself.

In the cell below, you can enter your own movie choices. The algorithm will then make recommendations for you! We have filled out some values according to our preferences, but after you have things working with our choices, you should change this to match your tastes. A list of all movies in the dataset is in the file movie list.

```
movieList, movieList_df = load_Movie_List_pd()

my_ratings = np.zeros(num_movies)          #  Initialize my ratings

# Check the file small_movie_list.csv for id of each movie in our
dataset
# For example, Toy Story 3 (2010) has ID 2700, so to rate it "5", you
can set
my_ratings[2700] = 5

#Or suppose you did not enjoy Persuasion (2007), you can set
```

```
my_ratings[2609] = 2;

# We have selected a few movies we liked / did not like and the
ratings we
# gave are as follows:
my_ratings[929]  = 5   # Lord of the Rings: The Return of the King,
The
my_ratings[246]  = 5   # Shrek (2001)
my_ratings[2716] = 3   # Inception
my_ratings[1150] = 5   # Incredibles, The (2004)
my_ratings[382]  = 2   # Amelie (Fabuleux destin d'Amélie Poulain, Le)
my_ratings[366]  = 5   # Harry Potter and the Sorcerer's Stone (a.k.a.
Harry Potter and the Philosopher's Stone) (2001)
my_ratings[622]  = 5   # Harry Potter and the Chamber of Secrets
(2002)
my_ratings[988]  = 3   # Eternal Sunshine of the Spotless Mind (2004)
my_ratings[2925] = 1   # Louis Theroux: Law & Disorder (2008)
my_ratings[2937] = 1   # Nothing to Declare (Rien à déclarer)
my_ratings[793]  = 5   # Pirates of the Caribbean: The Curse of the
Black Pearl (2003)
my_rated = [i for i in range(len(my_ratings)) if my_ratings[i] > 0]

print('\nNew user ratings:\n')
for i in range(len(my_ratings)):
    if my_ratings[i] > 0 :
        print(f'Rated {my_ratings[i]} for
{movieList_df.loc[i,"title"]}');
```

```
New user ratings:

Rated 5.0 for  Shrek (2001)
Rated 5.0 for  Harry Potter and the Sorcerer's Stone (a.k.a. Harry
Potter and the Philosopher's Stone) (2001)
Rated 2.0 for  Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)
Rated 5.0 for  Harry Potter and the Chamber of Secrets (2002)
Rated 5.0 for  Pirates of the Caribbean: The Curse of the Black Pearl
(2003)
Rated 5.0 for  Lord of the Rings: The Return of the King, The (2003)
Rated 3.0 for  Eternal Sunshine of the Spotless Mind (2004)
Rated 5.0 for  Incredibles, The (2004)
Rated 2.0 for  Persuasion (2007)
Rated 5.0 for  Toy Story 3 (2010)
Rated 3.0 for  Inception (2010)
Rated 1.0 for  Louis Theroux: Law & Disorder (2008)
Rated 1.0 for  Nothing to Declare (Rien à déclarer) (2010)
```

Now, let's add these reviews to Y and R and normalize the ratings.

```
# Reload ratings
Y, R = load_ratings_small()

# Add new user ratings to Y
Y = np.c_[my_ratings, Y]

# Add new user indicator matrix to R
R = np.c_[(my_ratings != 0).astype(int), R]

# Normalize the Dataset
Ynorm, Ymean = normalizeRatings(Y, R)
```

Let's prepare to train the model. Initialize the parameters and select the Adam optimizer.

```
#  Useful Values
num_movies, num_users = Y.shape
num_features = 100

# Set Initial Parameters (W, X), use tf.Variable to track these 
variables
tf.random.set_seed(1234) # for consistent results
W = tf.Variable(tf.random.normal((num_users, 
num_features),dtype=tf.float64),  name='W')
X = tf.Variable(tf.random.normal((num_movies, 
num_features),dtype=tf.float64),  name='X')
b = tf.Variable(tf.random.normal((1,          num_users),  
dtype=tf.float64),  name='b')

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

Let's now train the collaborative filtering model. This will learn the parameters X, W, and b.

The operations involved in learning w, b, and x simultaneously do not fall into the typical 'layers' offered in the TensorFlow neural network package. Consequently, the flow used in Course 2: Model, Compile(), Fit(), Predict(), are not directly applicable. Instead, we can use a custom training loop.

Recall from earlier labs the steps of gradient descent.

- repeat until convergence:
  - compute forward pass
  - compute the derivatives of the loss relative to parameters
  - update the parameters using the learning rate and the computed derivatives

TensorFlow has the marvelous capability of calculating the derivatives for you. This is shown below. Within the `tf.GradientTape()` section, operations on Tensorflow Variables are tracked. When `tape.gradient()` is later called, it will return the gradient of the loss relative to the tracked variables. The gradients can then be applied to the parameters using an optimizer. This is a very brief introduction to a useful feature of TensorFlow and other machine learning frameworks. Further information can be found by investigating "custom training loops" within the framework of interest.

```
iterations = 200
lambda_ = 1
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 included in cost)
        cost_value = cofi_cost_func_v(X, W, b, Ynorm, R, 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]) )

    # Log periodically.
    if iter % 20 == 0:
        print(f"Training loss at iteration {iter}: {cost_value:0.1f}")
```

```
Training loss at iteration 0: 2321191.3
Training loss at iteration 20: 136168.7
Training loss at iteration 40: 51863.3
Training loss at iteration 60: 24598.8
Training loss at iteration 80: 13630.4
Training loss at iteration 100: 8487.6
Training loss at iteration 120: 5807.7
Training loss at iteration 140: 4311.6
Training loss at iteration 160: 3435.2
Training loss at iteration 180: 2902.1
```

6 - Recommendations

Below, we compute the ratings for all the movies and users and display the movies that are recommended. These are based on the movies and ratings entered as `my_ratings[]` above. To predict the rating of movie i for user j, you compute $w^{(j)} \cdot x^{(i)} + b^{(j)}$. This can be computed for all ratings using matrix multiplication.

```
# Make a prediction using trained weights and biases
p = np.matmul(X.numpy(), np.transpose(W.numpy())) + b.numpy()
```

```python
#restore the mean
pm = p + Ymean

my_predictions = pm[:,0]

# sort predictions
ix = tf.argsort(my_predictions, direction='DESCENDING')

for i in range(17):
    j = ix[i]
    if j not in my_rated:
        print(f'Predicting rating {my_predictions[j]:0.2f} for movie 
{movieList[j]}')

print('\n\nOriginal vs Predicted ratings:\n')
for i in range(len(my_ratings)):
    if my_ratings[i] > 0:
        print(f'Original {my_ratings[i]}, Predicted 
{my_predictions[i]:0.2f} for {movieList[i]}')
```

```
Predicting rating 4.49 for movie My Sassy Girl (Yeopgijeogin geunyeo)
(2001)
Predicting rating 4.48 for movie Martin Lawrence Live: Runteldat
(2002)
Predicting rating 4.48 for movie Memento (2000)
Predicting rating 4.47 for movie Delirium (2014)
Predicting rating 4.47 for movie Laggies (2014)
Predicting rating 4.47 for movie One I Love, The (2014)
Predicting rating 4.46 for movie Particle Fever (2013)
Predicting rating 4.45 for movie Eichmann (2007)
Predicting rating 4.45 for movie Battle Royale 2: Requiem (Batoru
rowaiaru II: Chinkonka) (2003)
Predicting rating 4.45 for movie Into the Abyss (2011)


Original vs Predicted ratings:

Original 5.0, Predicted 4.90 for Shrek (2001)
Original 5.0, Predicted 4.84 for Harry Potter and the Sorcerer's Stone
(a.k.a. Harry Potter and the Philosopher's Stone) (2001)
Original 2.0, Predicted 2.13 for Amelie (Fabuleux destin d'Amélie
Poulain, Le) (2001)
Original 5.0, Predicted 4.88 for Harry Potter and the Chamber of
Secrets (2002)
Original 5.0, Predicted 4.87 for Pirates of the Caribbean: The Curse
of the Black Pearl (2003)
Original 5.0, Predicted 4.89 for Lord of the Rings: The Return of the
King, The (2003)
Original 3.0, Predicted 3.00 for Eternal Sunshine of the Spotless Mind
(2004)
```

```
Original 5.0, Predicted 4.90 for Incredibles, The (2004)
Original 2.0, Predicted 2.11 for Persuasion (2007)
Original 5.0, Predicted 4.80 for Toy Story 3 (2010)
Original 3.0, Predicted 3.00 for Inception (2010)
Original 1.0, Predicted 1.41 for Louis Theroux: Law & Disorder (2008)
Original 1.0, Predicted 1.26 for Nothing to Declare (Rien à déclarer)
(2010)
```

In practice, additional information can be utilized to enhance our predictions. Above, the predicted ratings for the first few hundred movies lie in a small range. We can augment the above by selecting from those top movies, movies that have high average ratings and movies with more than 20 ratings. This section uses a Pandas data frame which has many handy sorting features.

```
filter=(movieList_df["number of ratings"] > 20)
movieList_df["pred"] = my_predictions
movieList_df = movieList_df.reindex(columns=["pred", "mean rating",
"number of ratings", "title"])
movieList_df.loc[ix[:300]].loc[filter].sort_values("mean rating",
ascending=False)
```

```
          pred  mean rating  number of ratings  \
1743  4.030965     4.252336                107
2112  3.985287     4.238255                149
211   4.477792     4.122642                159
929   4.887053     4.118919                185
2700  4.796530     4.109091                 55
653   4.357304     4.021277                188
1122  4.004469     4.006494                 77
1841  3.980647     4.000000                 61
3083  4.084633     3.993421                 76
2804  4.434171     3.989362                 47
773   4.289679     3.960993                141
1771  4.344993     3.944444                 81
2649  4.133482     3.943396                 53
2455  4.175746     3.887931                 58
361   4.135291     3.871212                132
3014  3.967901     3.869565                 69
246   4.897137     3.867647                170
151   3.971888     3.836364                110
1150  4.898892     3.836000                125
793   4.874935     3.778523                149
366   4.843375     3.761682                107
754   4.021774     3.723684                 76
79    4.242984     3.699248                133
622   4.878342     3.598039                102

                                            title
1743                                 Departed, The (2006)
```

```
2112                           Dark Knight, The (2008)
211                                     Memento (2000)
929   Lord of the Rings: The Return of the King, The...
2700                               Toy Story 3 (2010)
653       Lord of the Rings: The Two Towers, The (2002)
1122                          Shaun of the Dead (2004)
1841                                  Hot Fuzz (2007)
3083                      Dark Knight Rises, The (2012)
2804  Harry Potter and the Deathly Hallows: Part 1 (...
773                                Finding Nemo (2003)
1771                               Casino Royale (2006)
2649                    How to Train Your Dragon (2010)
2455      Harry Potter and the Half-Blood Prince (2009)
361                              Monsters, Inc. (2001)
3014                               Avengers, The (2012)
246                                        Shrek (2001)
151   Crouching Tiger, Hidden Dragon (Wo hu cang lon...
1150                            Incredibles, The (2004)
793   Pirates of the Caribbean: The Curse of the Bla...
366   Harry Potter and the Sorcerer's Stone (a.k.a. ...
754                            X2: X-Men United (2003)
79                                        X-Men (2000)
622     Harry Potter and the Chamber of Secrets (2002)
```

7 - Congratulations!

You have implemented a useful recommender system!