Deep Learning Assignment - 2

Neelampalle Nikhil Kumar- 18bec031

IMPORTING LIBRARIES

In [13]:

```
import pandas as pd
import numpy as np
from zipfile import ZipFile
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from pathlib import Path
import matplotlib.pyplot as plt
```

LOADING DATASET AND PREPROCESSING

In [14]:

```
movielens_url = (
    "http://files.grouplens.org/datasets/movielens/ml-latest-small.zip"
)

zip_file = keras.utils.get_file(
    "ml-latest-small.zip", movielens_url, extract=False
)

datasets_path = Path(zip_file).parents[0]
movielens = datasets_path / "ml-latest-small"

if not movielens.exists():
    with ZipFile(zip_file, "r") as zip:

        print("Extracting.........")
        zip.extractall(path=datasets_path)
        print("Done!!!")

ratings = movielens / "ratings.csv"
tags = movielens / "tags.csv"
movies = movielens / "movies.csv"
```

In [15]:

```

df = pd.read_csv(ratings)
tags = pd.read_csv(tags)
movies = pd.read_csv(movies)
```

Exploratory Data Analysis

In [25]:

```
df.head()
```

Out[25]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [26]:

```
tags.head()
```

Out[26]:

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200

In [27]:

```
movies.head()
```

Out[27]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

In [17]:

```
df.describe()
```

Out[17]:

	userId	movieId	rating	timestamp
count	100836.000000	100836.000000	100836.000000	1.008360e+05
mean	326.127564	19435.295718	3.501557	1.205946e+09
std	182.618491	35530.987199	1.042529	2.162610e+08
min	1.000000	1.000000	0.500000	8.281246e+08
25%	177.000000	1199.000000	3.000000	1.019124e+09
50%	325.000000	2991.000000	3.500000	1.186087e+09
75%	477.000000	8122.000000	4.000000	1.435994e+09
max	610.000000	193609.000000	5.000000	1.537799e+09

In [18]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

In [21]:

```
user_id = df["userId"].unique().tolist()
user2user_encoded = {x: i for i, x in enumerate(user_id)}
userencoded2user = {i: x for i, x in enumerate(user_id)}


movie_id = df["movieId"].unique().tolist()
movie2movie_encoded = {x: i for i, x in enumerate(movie_id)}
movie_encoded2movie = {i: x for i, x in enumerate(movie_id)}

df["user"] = df["userId"].map(user2user_encoded)
df["movie"] = df["movieId"].map(movie2movie_encoded)

num_users = len(user2user_encoded)
num_movies = len(movie_encoded2movie)
df['rating'] = df['rating'].values.astype(np.float32)

minimumrating = min(df["rating"])
maximumrating = max(df["rating"])

print(f"Number of users: {num_users}, Number of Movies: {num_movies}, Min Rating: {minimumr
```

```
Number of users: 610, Number of Movies: 9724, Min Rating: 0.5, Max Rating:
5.0
```

In [22]:

```
df = df.sample(frac=1, random_state=42)
x = df[["user", "movie"]].values

y = df["rating"].apply(lambda x: (x - min_rating) / (max_rating - min_rating)).values

train_indices = int(0.9 * df.shape[0])
x_train, x_val, y_train, y_val = (
    x[:train_indices],
    x[train_indices:],
    y[:train_indices],
    y[train_indices:],
)
```

CREATING MODEL

In [23]:

```
EMBEDDING_SIZE = 50

class RecommenderNet(keras.Model):
    def __init__(self, num_users, num_movies, embedding_size, **kwargs):
        super(RecommenderNet, self).__init__(**kwargs)
        self.num_users = num_users
        self.num_movies = num_movies
        self.embedding_size = embedding_size
        self.user_embedding = layers.Embedding(
            num_users,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6),
        )
        self.user_bias = layers.Embedding(num_users, 1)
        self.movie_embedding = layers.Embedding(
            num_movies,
            embedding_size,
            embeddings_initializer="he_normal",
            embeddings_regularizer=keras.regularizers.l2(1e-6)
        )
        self.movie_bias = layers.Embedding(num_movies, 1)

    def call(self, inputs):
        user_vector = self.user_embedding(inputs[:, 0])
        user_bias = self.user_bias(inputs[:, 0])
        movie_vector = self.movie_embedding(inputs[:, 1])
        movie_bias = self.movie_bias(inputs[:, 1])
        dot_user_movie = tf.tensordot(user_vector, movie_vector, 2)
        # Add all the components (including bias)
        x = dot_user_movie + user_bias + movie_bias
        # The sigmoid activation forces the rating to be between 0 and 11
        return tf.nn.sigmoid(x)

model = RecommenderNet(num_users, num_movies, EMBEDDING_SIZE)
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(), optimizer=keras.optimizers.Adam(lr=0.001)
)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:105: U
serWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

TRAINING AND TESTING

In [24]:

```
history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=64,
    epochs=5,
#     verbose=1,
    validation_data=(x_val, y_val)
)
```

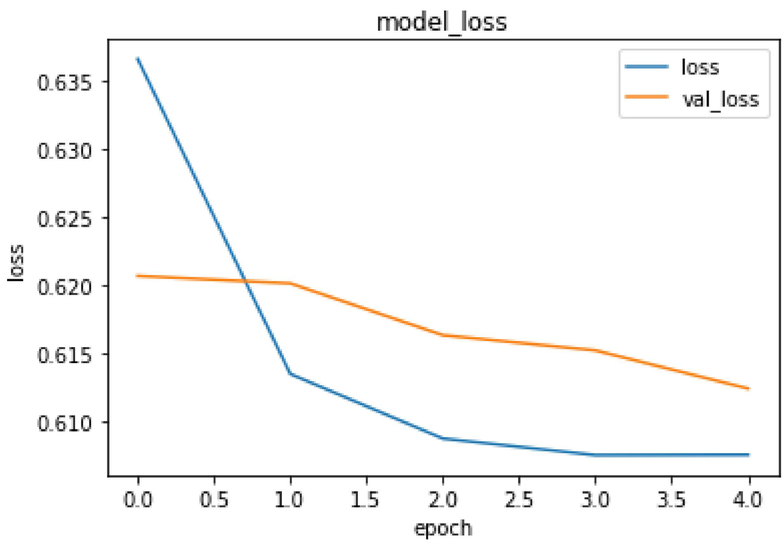
```
Epoch 1/5
1418/1418 [==============================] - 16s 9ms/step - loss: 0.6365 - v
al_loss: 0.6208
Epoch 2/5
1418/1418 [==============================] - 14s 10ms/step - loss: 0.6129 -
val_loss: 0.6195
Epoch 3/5
1418/1418 [==============================] - 14s 10ms/step - loss: 0.6088 -
val_loss: 0.6146
Epoch 4/5
1418/1418 [==============================] - 13s 9ms/step - loss: 0.6074 - v
al_loss: 0.6146
Epoch 5/5
1418/1418 [==============================] - 14s 10ms/step - loss: 0.6065 -
val_loss: 0.6167
```

In [14]:

```
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('model_loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
```

Out[14]:

```
<matplotlib.legend.Legend at 0x1e7de818610>
```



Showing top 10 movie recommendations to a user

In [15]:

```
movie_df = pd.read_csv(movielens_dir / 'movies.csv')

user_id = df.userId.sample(1).iloc[0]
movies_watched_by_user = df[df.userId == user_id]
movies_not_watched = movie_df[~movie_df['movieId'].isin(movies_watched_by_user.movieId.valu

movies_not_watched = list(set(movies_not_watched).intersection(set(movie2movie_encoded.keys

movies_not_watched = [[movie2movie_encoded.get(x)] for x in movies_not_watched]

user_encoder = user2user_encoded.get(user_id)

user_movie_array = np.hstack(
    ([[user_encoder]] * len(movies_not_watched), movies_not_watched)
)

ratings = model.predict(user_movie_array).flatten()
top_ratings_indices = ratings.argsort()[-10:][::-1]
recommended_movie_ids = [
    movie_encoded2movie.get(movies_not_watched[x][0]) for x in top_ratings_indices
]
```

In [16]:

```
print("Showing recommendations for user: {}".format(user_id))
print("====" * 9)
print("Movies with high ratings from user")
print("----" * 8)
top_movies_user = (
    movies_watched_by_user.sort_values(by="rating", ascending=False)
    .head(5)
    .movieId.values
)
movie_df_rows = movie_df[movie_df["movieId"].isin(top_movies_user)]
for row in movie_df_rows.itertuples():
    print(row.title, ":", row.genres)

print("----" * 8)
print("Top 10 movie recommendations")
print("----" * 8)
recommended_movies = movie_df[movie_df["movieId"].isin(recommended_movie_ids)]
for row in recommended_movies.itertuples():
    print(row.title, ":", row.genres)
```

```
Showing recommendations for user: 496
====================================
Movies with high ratings from user
--------------------------------
Godfather, The (1972) : Crime|Drama
Rear Window (1954) : Mystery|Thriller
Casablanca (1942) : Drama|Romance
Dark Knight, The (2008) : Action|Crime|Drama|IMAX
Her (2013) : Drama|Romance|Sci-Fi
--------------------------------
Top 10 movie recommendations
--------------------------------
Shawshank Redemption, The (1994) : Crime|Drama
Star Wars: Episode VI - Return of the Jedi (1983) : Action|Adventure|Sci-Fi
Third Man, The (1949) : Film-Noir|Mystery|Thriller
Goodfellas (1990) : Crime|Drama
Alien (1979) : Horror|Sci-Fi
Psycho (1960) : Crime|Horror
Full Metal Jacket (1987) : Drama|War
Amadeus (1984) : Drama
Boot, Das (Boat, The) (1981) : Action|Drama|War
Glory (1989) : Drama|War
```