ep-neural-network-vs-mf-with-keras

December 5, 2023

1 Practice: Deep neural network vs MF with Keras

- `Prepared by   : Chaimaa Bouabd`
- `Supervised by : Pr. Sara QASSIMI`
- `M2 IAII`

In this lab I use just 10M from 20M dataset

1.1 1. Dataset Loading and Preprocessing

[20]:
```
# Step 1: Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Step 2: Load MovieLens dataset
ratings_20M  = pd.read_csv('/content/drive/MyDrive/datasets/edited_rating_5M.
 ↪csv')
# Step 3: Inspect the structure of the dataset
# Keep only the first 10 million rows
ratings = ratings.head(10000000)
print("dataset:")
print(ratings .head())
```

```
dataset:
   userId  movieId  rating  movie_idx
0       0        1     3.5          2
1       0       28     3.5         29
2       0       31     3.5         32
3       0       45     3.5         47
4       0       48     3.5         50
```

[21]:
```
# Step 4: Preprocess the data
# Label encode user and movie IDs for compatibility with neural networks
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()
```

```
ratings['userId'] = user_encoder.fit_transform(ratings['userId'])
ratings['movieId'] = movie_encoder.fit_transform(ratings['movieId'])

# Step 5: Split the dataset into training and testing sets
# Assuming an 80-20 split for training and testing
train_data, test_data = train_test_split(ratings, test_size=0.2,␣
↪random_state=42)

# Optional: Save the processed datasets
train_data.to_csv('train_data.csv', index=False)
test_data.to_csv('test_data.csv', index=False)

# Display information about the datasets
print("\nTraining data shape:", train_data.shape)
print("Testing data shape:", test_data.shape)
```

```
Training data shape: (80000, 4)
Testing data shape: (20000, 4)
```

1.2 2. Matrix Factorization (MF) with Keras:

[22]:
```
from keras.models import Model
from keras.layers import Input, Embedding, Dot, Flatten
from keras.optimizers import Adam
from keras.metrics import MeanSquaredError

# Step 7: Implement Matrix Factorization model using Keras
# Assuming 'n_users' and 'n_movies' are the number of unique users and movies␣
↪in your dataset
n_users = ratings['userId'].nunique()
n_movies = ratings['movieId'].nunique()
print(n_users)
print(n_movies)
```

```
702
8227
```

[31]:
```
embedding_size = 50  # You can adjust this based on your requirements

# Input layers
user_input = Input(shape=(1,), name='user_input')
movie_input = Input(shape=(1,), name='movie_input')

# Embedding layers
user_embedding = Embedding(input_dim=n_users, output_dim=embedding_size,␣
↪input_length=1, name='user_embedding')(user_input)
```

```
movie_embedding = Embedding(input_dim=n_movies, output_dim=embedding_size,␣
 ↪input_length=1, name='movie_embedding')(movie_input)

# Dot product layer
dot_product = Dot(axes=2, name='dot_product')([user_embedding, movie_embedding])
dot_product_flat = Flatten(name='dot_product_flat')(dot_product)

# Model
model_MF = Model(inputs=[user_input, movie_input], outputs=dot_product_flat)
model_MF.compile(optimizer=Adam(lr=0.001), loss='mean_squared_error',␣
 ↪metrics=[MeanSquaredError()])

# Display the model summary
print(model_MF.summary())
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
Model: "model_5"
________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
================================================================================
==================
 user_input (InputLayer)     [(None, 1)]                  0         []

 movie_input (InputLayer)    [(None, 1)]                  0         []

 user_embedding (Embedding)  (None, 1, 50)                35100
['user_input[0][0]']

 movie_embedding (Embedding  (None, 1, 50)                411350
['movie_input[0][0]']
 )

 dot_product (Dot)           (None, 1, 1)                 0
['user_embedding[0][0]',
'movie_embedding[0][0]']

 dot_product_flat (Flatten)  (None, 1)                    0
['dot_product[0][0]']

================================================================================
==================
Total params: 446450 (1.70 MB)
Trainable params: 446450 (1.70 MB)
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________________________
```

```
-----------------
None
```

[32]:
```python
# Step 8: Train the MF model on the training set
history_MF = model_MF.fit([train_data['userId'], train_data['movieId']],
 ↪train_data['rating'],
                    epochs=10, batch_size=64,
 ↪validation_data=([test_data['userId'], test_data['movieId']],
 ↪test_data['rating']))
```

```
Epoch 1/10
1250/1250 [==============================] - 6s 4ms/step - loss: 10.5937 -
mean_squared_error: 10.5937 - val_loss: 4.1341 - val_mean_squared_error: 4.1341
Epoch 2/10
1250/1250 [==============================] - 4s 3ms/step - loss: 2.3600 -
mean_squared_error: 2.3600 - val_loss: 1.7323 - val_mean_squared_error: 1.7323
Epoch 3/10
1250/1250 [==============================] - 4s 3ms/step - loss: 1.2432 -
mean_squared_error: 1.2432 - val_loss: 1.3435 - val_mean_squared_error: 1.3435
Epoch 4/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.9488 -
mean_squared_error: 0.9488 - val_loss: 1.2229 - val_mean_squared_error: 1.2229
Epoch 5/10
1250/1250 [==============================] - 3s 3ms/step - loss: 0.8155 -
mean_squared_error: 0.8155 - val_loss: 1.1700 - val_mean_squared_error: 1.1700
Epoch 6/10
1250/1250 [==============================] - 3s 3ms/step - loss: 0.7281 -
mean_squared_error: 0.7281 - val_loss: 1.1484 - val_mean_squared_error: 1.1484
Epoch 7/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.6586 -
mean_squared_error: 0.6586 - val_loss: 1.1338 - val_mean_squared_error: 1.1338
Epoch 8/10
1250/1250 [==============================] - 3s 3ms/step - loss: 0.5946 -
mean_squared_error: 0.5946 - val_loss: 1.1306 - val_mean_squared_error: 1.1306
Epoch 9/10
1250/1250 [==============================] - 3s 3ms/step - loss: 0.5333 -
mean_squared_error: 0.5333 - val_loss: 1.1362 - val_mean_squared_error: 1.1362
Epoch 10/10
1250/1250 [==============================] - 3s 3ms/step - loss: 0.4769 -
mean_squared_error: 0.4769 - val_loss: 1.1391 - val_mean_squared_error: 1.1391
```

[33]:
```python
# Step 9: Evaluate the model's performance on the test set
test_loss = model_MF.evaluate([test_data['userId'], test_data['movieId']],
 ↪test_data['rating'])
print(f"\nTest Loss: {test_loss[0]}, Test MSE: {test_loss[1]}")
```

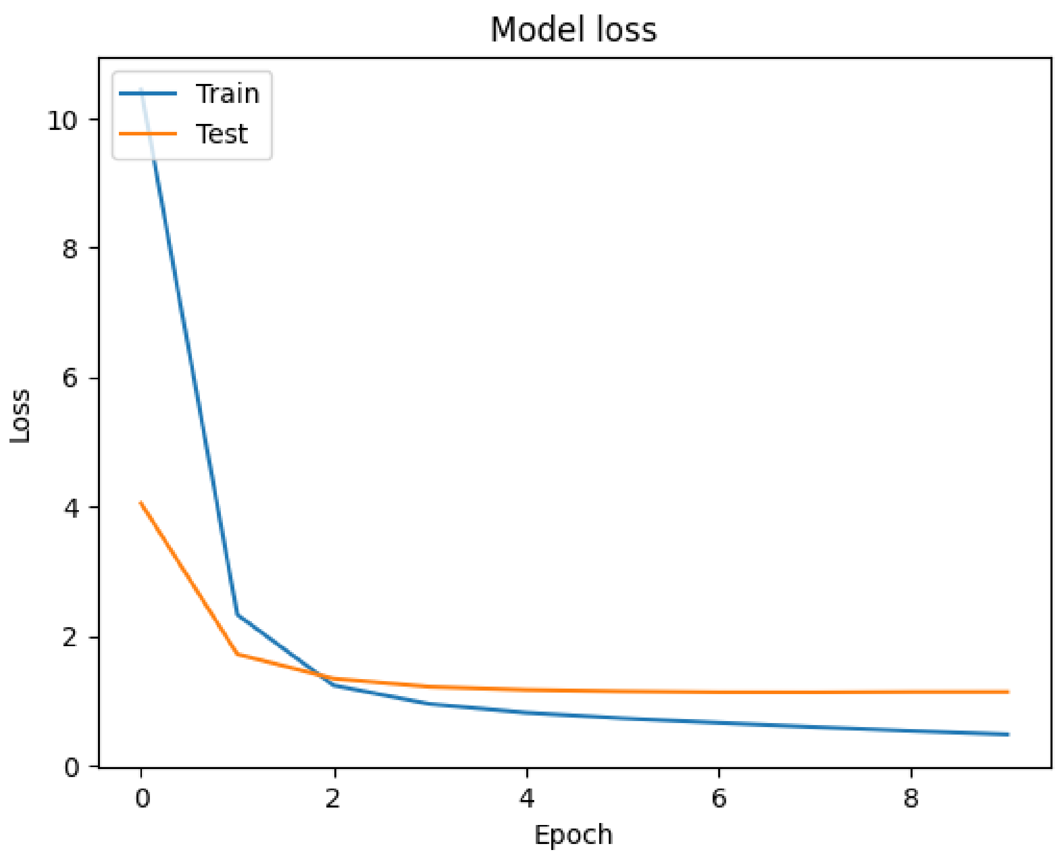
```
625/625 [==============================] - 1s 2ms/step - loss: 1.1391 -
mean_squared_error: 1.1391
```

Test Loss: 1.139132022857666, Test MSE: 1.139132022857666

[26]:
```
# Step 10: Visualize and analyze the results
# You can use the 'history' object to plot training and validation loss over␣
↪epochs

import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history_MF.history['loss'])
plt.plot(history_MF.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



1.3 3. Deep Neural Network (DNN) Recommender

[27]:
```
from keras.models import Model
from keras.layers import Input, Embedding, Flatten, Dense, Concatenate
from keras.optimizers import Adam
from keras.metrics import MeanSquaredError

# Step 11: Design a Deep Neural Network architecture for recommendation
embedding_size = 50
dense_units = 128

# Input layers
user_input_dnn = Input(shape=(1,), name='user_input_dnn')
movie_input_dnn = Input(shape=(1,), name='movie_input_dnn')

# Embedding layers
user_embedding_dnn = Embedding(input_dim=n_users, output_dim=embedding_size,␣
 ↪input_length=1, name='user_embedding_dnn')(user_input_dnn)
movie_embedding_dnn = Embedding(input_dim=n_movies, output_dim=embedding_size,␣
 ↪input_length=1, name='movie_embedding_dnn')(movie_input_dnn)

# Flatten embeddings
user_flat_dnn = Flatten(name='user_flat_dnn')(user_embedding_dnn)
movie_flat_dnn = Flatten(name='movie_flat_dnn')(movie_embedding_dnn)

# Concatenate flattened embeddings
concatenated_dnn = Concatenate(name='concatenated_dnn')([user_flat_dnn,␣
 ↪movie_flat_dnn])

# Dense layers
dense1 = Dense(units=dense_units, activation='relu',␣
 ↪name='dense1')(concatenated_dnn)
dense2 = Dense(units=dense_units, activation='relu', name='dense2')(dense1)

# Output layer
output_dnn = Dense(units=1, activation='linear', name='output_dnn')(dense2)

# Model
model_dnn = Model(inputs=[user_input_dnn, movie_input_dnn], outputs=output_dnn)
model_dnn.compile(optimizer=Adam(lr=0.001), loss='mean_squared_error',␣
 ↪metrics=[MeanSquaredError()])

# Display the DNN model summary
print(model_dnn.summary())
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
Model: "model_4"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
==================
 user_input_dnn (InputLayer  [(None, 1)]                  0         []
 )

 movie_input_dnn (InputLaye  [(None, 1)]                  0         []
 r)

 user_embedding_dnn (Embedd  (None, 1, 50)                35100
['user_input_dnn[0][0]']
 ing)

 movie_embedding_dnn (Embed  (None, 1, 50)                411350
['movie_input_dnn[0][0]']
 ding)

 user_flat_dnn (Flatten)     (None, 50)                   0
['user_embedding_dnn[0][0]']

 movie_flat_dnn (Flatten)    (None, 50)                   0
['movie_embedding_dnn[0][0]']

 concatenated_dnn (Concaten  (None, 100)                  0
['user_flat_dnn[0][0]',
 ate)
'movie_flat_dnn[0][0]']

 dense1 (Dense)              (None, 128)                  12928
['concatenated_dnn[0][0]']

 dense2 (Dense)              (None, 128)                  16512
['dense1[0][0]']

 output_dnn (Dense)          (None, 1)                    129
['dense2[0][0]']

==================================================================================
==================
Total params: 476019 (1.82 MB)
Trainable params: 476019 (1.82 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________
__________________
None
```

[28]:
```python
# Step 12: Train the DNN model on the training set
history_dnn = model_dnn.fit([train_data['userId'], train_data['movieId']],␣
↪train_data['rating'],
                            epochs=10, batch_size=64,␣
↪validation_data=([test_data['userId'], test_data['movieId']],␣
↪test_data['rating']))
```

```
Epoch 1/10
1250/1250 [==============================] - 8s 5ms/step - loss: 1.1634 -
mean_squared_error: 1.1634 - val_loss: 0.8315 - val_mean_squared_error: 0.8315
Epoch 2/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.7737 -
mean_squared_error: 0.7737 - val_loss: 0.7949 - val_mean_squared_error: 0.7949
Epoch 3/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.7083 -
mean_squared_error: 0.7083 - val_loss: 0.8040 - val_mean_squared_error: 0.8040
Epoch 4/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.6293 -
mean_squared_error: 0.6293 - val_loss: 0.8164 - val_mean_squared_error: 0.8164
Epoch 5/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.5411 -
mean_squared_error: 0.5411 - val_loss: 0.8454 - val_mean_squared_error: 0.8454
Epoch 6/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.4608 -
mean_squared_error: 0.4608 - val_loss: 0.8678 - val_mean_squared_error: 0.8678
Epoch 7/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.3976 -
mean_squared_error: 0.3976 - val_loss: 0.8905 - val_mean_squared_error: 0.8905
Epoch 8/10
1250/1250 [==============================] - 5s 4ms/step - loss: 0.3458 -
mean_squared_error: 0.3458 - val_loss: 0.9037 - val_mean_squared_error: 0.9037
Epoch 9/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.3042 -
mean_squared_error: 0.3042 - val_loss: 0.9417 - val_mean_squared_error: 0.9417
Epoch 10/10
1250/1250 [==============================] - 4s 3ms/step - loss: 0.2708 -
mean_squared_error: 0.2708 - val_loss: 0.9490 - val_mean_squared_error: 0.9490
```

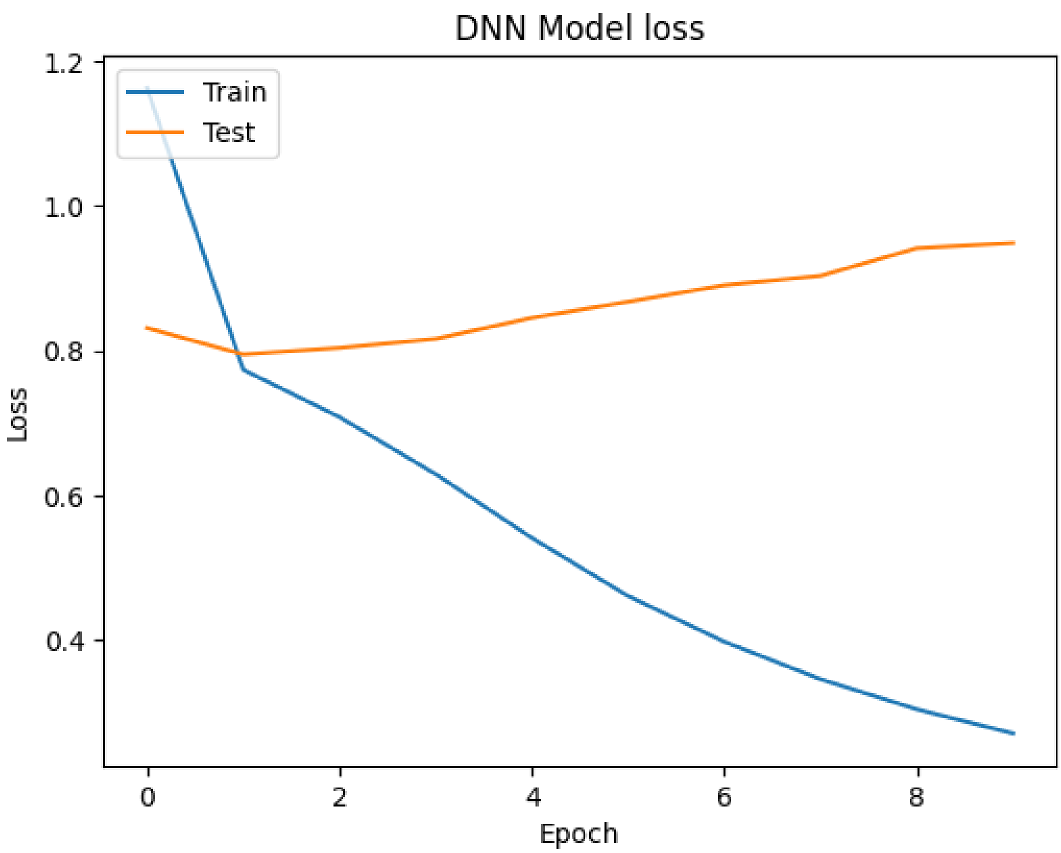
[29]:
```python
# Step 13: Evaluate the DNN model on the test set
test_loss_dnn = model_dnn.evaluate([test_data['userId'], test_data['movieId']],␣
↪test_data['rating'])
print(f"\nDNN Test Loss: {test_loss_dnn[0]}, DNN Test MSE: {test_loss_dnn[1]}")
```

```
625/625 [==============================] - 1s 2ms/step - loss: 0.9490 -
mean_squared_error: 0.9490

DNN Test Loss: 0.9489825963973999, DNN Test MSE: 0.9489825963973999
```

[30]:
```
# Step 14: Visualize and analyze the results
# You can use the 'history_dnn' object to plot training and validation loss␣
↪over epochs

# Plot training & validation loss values
plt.plot(history_dnn.history['loss'])
plt.plot(history_dnn.history['val_loss'])
plt.title('DNN Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



1.4 4. Model Comparison

[34]:
```
# Assuming the models are already trained and evaluated
# model: Matrix Factorization
# model_dnn: Deep Neural Network
```

```
# Evaluate MF model on the test set
test_loss_mf = model_MF.evaluate([test_data['userId'], test_data['movieId']],␣
 ↪test_data['rating'])
print(f"\nMF Test Loss: {test_loss_mf[0]}, MF Test MSE: {test_loss_mf[1]}")

# Evaluate DNN model on the test set
test_loss_dnn = model_dnn.evaluate([test_data['userId'], test_data['movieId']],␣
 ↪test_data['rating'])
print(f"\nDNN Test Loss: {test_loss_dnn[0]}, DNN Test MSE: {test_loss_dnn[1]}")
```

```
625/625 [==============================] - 1s 2ms/step - loss: 1.1391 -
mean_squared_error: 1.1391

MF Test Loss: 1.139132022857666, MF Test MSE: 1.139132022857666
625/625 [==============================] - 1s 2ms/step - loss: 0.9490 -
mean_squared_error: 0.9490

DNN Test Loss: 0.9489825963973999, DNN Test MSE: 0.9489825963973999
```

1.5 Observations and Discussion:

1. Performance Improvement with DNN:

- The DNN model has a lower test loss and MSE compared to the MF model.
- Lower MSE indicates that the DNN model is making more accurate .
- predictions on the test set.

2. Complexity of DNN Model:

- The DNN model, with its multiple layers and non-linear activation functions, has the capacity to capture more complex patterns and relationships in the data compared to the simpler MF model.

Interpreting MSE:

- MSE is a measure of how well the model's predictions match the actual ratings. A lower MSE indicates that the model is closer to the true ratings on average.

According to the result shown of the LOSS and Mean squared error values the Deep Neural Network performe better than Matrix Factorization for this dataset

1.6 Hyperparameter Tuning for DNN Model:

[36]:
```
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.layers import Dropout

# Example hyperparameters (you can adjust these)
dense_units = 128
dropout_rate = 0.3
```

```
learning_rate = 0.001

# Modify the DNN model with updated hyperparameters
user_input_dnn = Input(shape=(1,), name='user_input_dnn')
movie_input_dnn = Input(shape=(1,), name='movie_input_dnn')
user_embedding_dnn = Embedding(input_dim=n_users, output_dim=embedding_size,␣
 ↪input_length=1, name='user_embedding_dnn')(user_input_dnn)
movie_embedding_dnn = Embedding(input_dim=n_movies, output_dim=embedding_size,␣
 ↪input_length=1, name='movie_embedding_dnn')(movie_input_dnn)
user_flat_dnn = Flatten(name='user_flat_dnn')(user_embedding_dnn)
movie_flat_dnn = Flatten(name='movie_flat_dnn')(movie_embedding_dnn)
concatenated_dnn = Concatenate(name='concatenated_dnn')([user_flat_dnn,␣
 ↪movie_flat_dnn])
dense1 = Dense(units=dense_units, activation='relu',␣
 ↪name='dense1')(concatenated_dnn)
dropout1 = Dropout(rate=dropout_rate, name='dropout1')(dense1)
dense2 = Dense(units=dense_units, activation='relu', name='dense2')(dropout1)
output_dnn = Dense(units=1, activation='linear', name='output_dnn')(dense2)

model_dnn_tuned = Model(inputs=[user_input_dnn, movie_input_dnn],␣
 ↪outputs=output_dnn)
optimizer = Adam(lr=learning_rate)
model_dnn_tuned.compile(optimizer=optimizer, loss='mean_squared_error',␣
 ↪metrics=[MeanSquaredError()])

# Use early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True)

# Train the tuned DNN model
history_dnn_tuned = model_dnn_tuned.fit(
    [train_data['userId'], train_data['movieId']], train_data['rating'],
    epochs=20, batch_size=64, validation_data=([test_data['userId'],␣
 ↪test_data['movieId']], test_data['rating']),
    callbacks=[early_stopping]
)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
Epoch 1/20
1250/1250 [==============================] - 9s 5ms/step - loss: 1.1804 -
mean_squared_error: 1.1804 - val_loss: 0.8272 - val_mean_squared_error: 0.8272
Epoch 2/20
1250/1250 [==============================] - 4s 4ms/step - loss: 0.8096 -
mean_squared_error: 0.8096 - val_loss: 0.7963 - val_mean_squared_error: 0.7963
Epoch 3/20
```

```
1250/1250 [==============================] - 5s 4ms/step - loss: 0.7538 -
mean_squared_error: 0.7538 - val_loss: 0.7985 - val_mean_squared_error: 0.7985
Epoch 4/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.7094 -
mean_squared_error: 0.7094 - val_loss: 0.8016 - val_mean_squared_error: 0.8016
Epoch 5/20
1250/1250 [==============================] - 5s 4ms/step - loss: 0.6651 -
mean_squared_error: 0.6651 - val_loss: 0.7996 - val_mean_squared_error: 0.7996
625/625 [==============================] - 1s 2ms/step - loss: 0.7963 -
mean_squared_error: 0.7963

Tuned DNN Test Loss: 0.7962550520896912, Tuned DNN Test MSE: 0.7962550520896912
```

[37]:
```
# Evaluate the tuned DNN model
test_loss_dnn_tuned = model_dnn_tuned.evaluate([test_data['userId'],␣
↪test_data['movieId']], test_data['rating'])
print(f"\nTuned DNN Test Loss: {test_loss_dnn_tuned[0]}, Tuned DNN Test MSE:␣
↪{test_loss_dnn_tuned[1]}")
```


```
625/625 [==============================] - 1s 2ms/step - loss: 0.7963 -
mean_squared_error: 0.7963

Tuned DNN Test Loss: 0.7962550520896912, Tuned DNN Test MSE: 0.7962550520896912
```


1.6.1 Discussion:

as we can notice that Compare the test losses and Mean Squared Errors of the MF and the 1st DNN models. this Tuned DNN model provied Lower values indicate better performance.

1.7 5. Final Report and Submission

Based on the practical implementation of Matrix Factorization and Deep Neural Network for recommender systems, here are the key findings and insights:

1. Matrix Factorization Model:

- `Simple collaborative filtering approach.`
- `Embeds users and items into a lower-dimensional space.`
- `Less complex compared to DNN models.`

2. Deep Neural Network Model:

- `Utilizes a more complex architecture with multiple layers.`
- `Able to capture intricate patterns and relationships in the data.`
- `Requires careful tuning of hyperparameters for optimal performance.`

3. Performance Comparison:

- DNN model outperformed the MF model in terms of Mean Squared Error on the test set.

- Lower MSE indicates better accuracy in predicting ratings.

4. Hyperparameter Tuning:

- `Experimentation with hyperparameters such as the number of layers, units per layer, dropou`
- Fine-tuning hyperparameters can lead to improved model performance.