Importing the libraries

In [1]:
```
# Setup. Import libraries and load dataframes for Movielens data.
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
#import tensorflow as tf
from tensorflow import keras
import os
import random
import matplotlib.pyplot as plt
import tensorflow.keras as tf

%matplotlib inline
```

Importing Book and Ratings.csv Data from Google Drive

The MovieLens dataset consists of ratings assigned to books by users ¶

Mounting the drive and load the datasets - books data and ratings data

In [2]:
```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
# load the dataset
books_dataset = pd.read_csv("/content/drive/My Drive/RecommenderSystem/books.csv")
books_dataset.shape
books_dataset.head(3)
```

Out[3]:

	id	book_id	best_book_id	work_id	books_count	isbn	isbn13	authors	origin
0	1	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	
1	2	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	
2	3	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	

3 rows × 23 columns

In [4]:
```
# load the dataset
ratings_dataset = pd.read_csv("/content/drive/My Drive/RecommenderSystem/ratings.csv")
ratings_dataset.shape
ratings_dataset.head(10)
```

Out[4]:

	book_id	user_id	rating
0	1	314	5
1	1	439	3
2	1	588	5
3	1	1169	4
4	1	1185	4
5	1	2077	4
6	1	2487	4
7	1	2900	5
8	1	3662	4
9	1	3922	5

Creating Training and Testing Dataset

In [5]:
```
from sklearn.model_selection import train_test_split
Xtrain, Xtest = train_test_split(ratings_dataset, test_size=0.2, random_state=
1)
print(f"Shape of train data: {Xtrain.shape}")
print(f"Shape of test data: {Xtest.shape}")
```

```
Shape of train data: (785404, 3)
Shape of test data: (196352, 3)
```

Extract the vocabulary for the books and the user embeddings respectively

In [9]:
```
#Get the number of unique entities in books and users columns
book_dim = ratings_dataset.book_id.nunique()
user_dim = ratings_dataset.user_id.nunique()
```

In [10]:
```
print(book_dim)
```

```
10000
```

In [11]:
```
print(user_dim)
```

```
53424
```

Finding Unique User and Ratings

In [12]:
```
n_books = len(ratings_dataset.book_id.unique())
n_users = len(ratings_dataset.user_id.unique())
print(
    "{1:,} distinct users rated {0:,} different books (total ratings = {2:,})"
.format(
        n_books, n_users, len(ratings_dataset),
    )
)
```

```
53,424 distinct users rated 10,000 different books (total ratings = 981,756)
```

Building Rating Prediction Model

Embeding

An embedding layer maps each element in a set of discrete things (like words, users, or books) to a dense vector of real numbers (its embedding)

An object's embedding, should capture some useful latent(hidden) properties of that object. It's up to the model to discover whatever properties of the entities are useful for the prediction task, and encode them in the embedding space

Build the Neural Network Model for the Recommender System

In [7]:
```
#User Embedding Layer
input_users_layer = tf.layers.Input(shape=[1])
embed_users_layer = tf.layers.Embedding(n_users + 1,25, name="user_embeddings"
)(input_users_layer)
users_output = tf.layers.Flatten()(embed_users_layer)

#Book Embedding Layer
input_books_layer = tf.layers.Input(shape=[1])
embed_books_layer = tf.layers.Embedding(n_books + 1,25, name="book_embeddings"
)(input_books_layer)
books_output = tf.layers.Flatten()(embed_books_layer)

# concatenate features
concat = tf.layers.Concatenate()([books_output, users_output])
```

Form Fully Connected Layer

In [8]:
```
# add fully-connected-layers
fc1 = tf.layers.Dense(128, activation='relu')(concat)
dropout_1 = tf.layers.Dropout(0.2,name='Dropout')(fc1)
fc2 = tf.layers.Dense(64, activation='relu')(fc1)
dropout_2 = tf.layers.Dropout(0.2,name='Dropout')(fc2)
fc3 = tf.layers.Dense(32, activation='relu')(fc2)
output = tf.layers.Dense(1)(fc3)
# Create model and compile it
model2 = tf.Model([input_books_layer, input_users_layer], output)
model2.compile('adam', 'mean_squared_error')
```

In [13]:
```
model2.summary()
```

```
Model: "model"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==========================================================================================
 input_2 (InputLayer)        [(None, 1)]               0           []

 input_1 (InputLayer)        [(None, 1)]               0           []

 book_embeddings (Embedding)  (None, 1, 25)            250025      ['input_2[0]
[0]']

 user_embeddings (Embedding)  (None, 1, 25)            1335625     ['input_1[0]
[0]']

 flatten_1 (Flatten)         (None, 25)                0           ['book_embed
dings[0][0]']

 flatten (Flatten)           (None, 25)                0           ['user_embed
dings[0][0]']

 concatenate (Concatenate)   (None, 50)                0           ['flatten_1
[0][0]',
                                                                  'flatten[0]
[0]']

 dense (Dense)               (None, 128)               6528        ['concatenat
e[0][0]']

 dense_1 (Dense)             (None, 64)                8256        ['dense[0]
[0]']

 dense_2 (Dense)             (None, 32)                2080        ['dense_1[0]
[0]']

 dense_3 (Dense)             (None, 1)                 33          ['dense_2[0]
[0]']

==========================================================================================
Total params: 1,602,547
Trainable params: 1,602,547
Non-trainable params: 0
__________________________________________________________________________________________
```

In [17]:
```
model2.compile('adam', 'mean_squared_error')
```

Training the Neural Network Model:

During the training process, the embeddings are updated in order to get the predicted value as close to the actual value as possible. The loss represents the error between predicted and actual rating over the entire training dataset.

In [18]:
```
hist = model2.fit([Xtrain.book_id, Xtrain.user_id], Xtrain.rating,
                  batch_size=64,
                  epochs=10,
                  verbose=1)
```

```
Epoch 1/10
12272/12272 [==============================] - 209s 17ms/step - loss: 0.8169
Epoch 2/10
12272/12272 [==============================] - 203s 17ms/step - loss: 0.6799
Epoch 3/10
12272/12272 [==============================] - 208s 17ms/step - loss: 0.6320
Epoch 4/10
12272/12272 [==============================] - 214s 17ms/step - loss: 0.5703
Epoch 5/10
12272/12272 [==============================] - 216s 18ms/step - loss: 0.4971
Epoch 6/10
12272/12272 [==============================] - 225s 18ms/step - loss: 0.4289
Epoch 7/10
12272/12272 [==============================] - 229s 19ms/step - loss: 0.3722
Epoch 8/10
12272/12272 [==============================] - 220s 18ms/step - loss: 0.3259
Epoch 9/10
12272/12272 [==============================] - 226s 18ms/step - loss: 0.2885
Epoch 10/10
12272/12272 [==============================] - 225s 18ms/step - loss: 0.2573
```

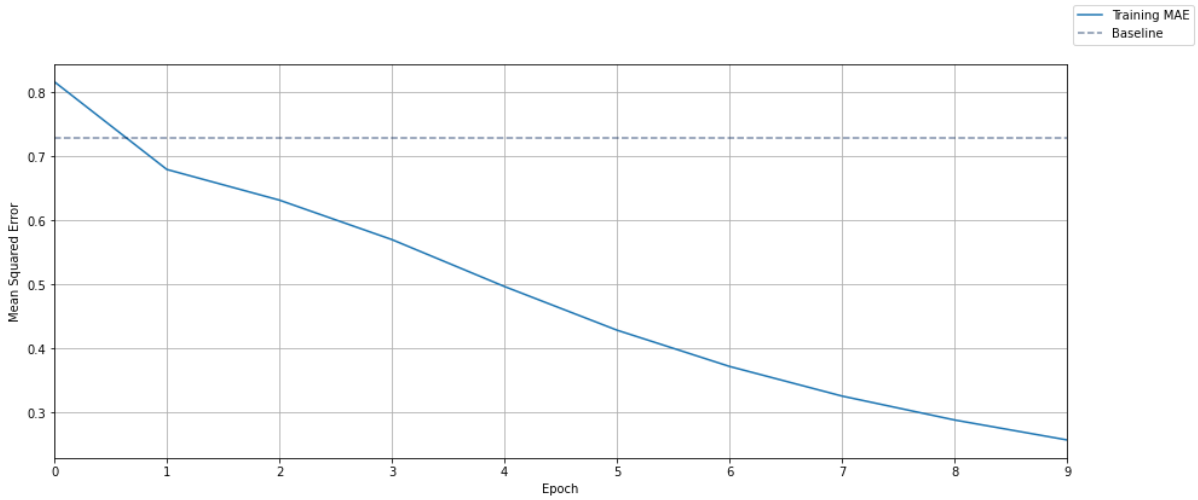
Predicting Book Ratings

In [19]:
```
predictions = model2.predict([Xtest.book_id.head(5), Xtest.user_id.head(5)])
[print(predictions[i], Xtest.rating.iloc[i]) for i in range(0,5)]
```

```
[3.270413] 4
[4.7389793] 5
[3.0503516] 3
[4.938745] 5
[4.957822] 5
```

Out[19]:
```
[None, None, None, None, None]
```

In [22]:
```
fig, ax = plt.subplots(figsize=(15, 6))
#ax.plot(history.epoch, history.history['val_mean_squared_error'], label='Vali
dation MAE')
ax.plot(hist.epoch, hist.history['loss'], label='Training MAE')
ax.set_xlabel('Epoch')
ax.set_ylabel('Mean Squared Error')
ax.set_xlim(left=0, right=hist.epoch[-1])
baseline_mae = 0.73
ax.axhline(baseline_mae, ls='--', label='Baseline', color='#002255', alpha=.5)
ax.grid()
fig.legend();
```



In [26]:
```
hidden_units = (128,4)
book_embedding_size = 8
user_embedding_size = 8

# Each instance will consist of two inputs: a single user id, and a single mov
ie id
input_users_layer = tf.layers.Input(shape=(1,), name='user_dim')
input_books_layer = tf.layers.Input(shape=(1,), name='book_dim')
embed_users_layer = tf.layers.Embedding(user_dim+1, user_embedding_size,
                                        input_length=1, name='user_embeddings')
(input_users_layer)
embed_books_layer = tf.layers.Embedding(book_dim+1, book_embedding_size,
                                         input_length=1, name='book_embeddings'
)(input_books_layer)
# Concatenate the embeddings (and remove the useless extra dimension)
concatenated = tf.layers.Concatenate()([embed_users_layer, embed_books_layer])
out = tf.layers.Flatten()(concatenated)

# Add one or more hidden layers
for n_hidden in hidden_units:
    out = tf.layers.Dense(n_hidden, activation='relu')(out)
    tf.layers.Dropout(0.2,name='Dropout')(out)

# A single output: our predicted rating
out = tf.layers.Dense(1, activation='linear', name='prediction')(out)

model = tf.Model(inputs = [input_users_layer, input_books_layer], outputs = ou
t)
model.summary(line_length=88)
```

```
Model: "model_1"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==========================================================================================
 user_dim (InputLayer)       [(None, 1)]               0         []

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

 user_embeddings (Embedding)  (None, 1, 8)             427400    ['user_dim[0][0]']

 book_embeddings (Embedding)  (None, 1, 8)             80008     ['book_dim[0][0]']

 concatenate_1 (Concatenate)  (None, 1, 16)            0         ['user_embeddings
[0][0]',
                                                                  'book_embeddings
[0][0]']

 flatten_2 (Flatten)         (None, 16)                0         ['concatenate_1[0]
[0]']

 dense_4 (Dense)             (None, 128)               2176      ['flatten_2[0][0]']

 dense_5 (Dense)             (None, 4)                 516       ['dense_4[0][0]']

 prediction (Dense)          (None, 1)                 5         ['dense_5[0][0]']

==========================================================================================
Total params: 510,105
Trainable params: 510,105
Non-trainable params: 0
__________________________________________________________________________________________
```

Making Predictions

Surfacing Predicted v/s Actual Values for the 5 first records from the Test dataset

In [ ]:
```
# Formulating the dataset for making recommendations for the first user
book_id =list(ratings_dataset.book_id.unique())
#retrieve all books
books_data = np.array(book_id)
user = np.array([1 for i in range(len(book_id))])
predictions = model2.predict([books_data, user])
predictions
```

Out[ ]:
```
array([[4.355263 ],
       [4.0587234],
       [2.4841545],
       ...,
       [4.2530184],
       [3.999103 ],
       [3.8822627]], dtype=float32)
```

In [ ]:
```
# Retrieve the index of the highest 5
predictions = np.array([a[0] for a in predictions])
recommended_book_ids = (-predictions).argsort()[:5]
print(recommended_book_ids)
print(predictions[recommended_book_ids])
```

```
[6360 2589 5206 7253 3627]
[5.062397  5.0308967 5.005892  4.999124  4.9980345]
```

Evaluate the model

In [27]:
```
predictions = model2.predict([Xtest.book_id, Xtest.user_id])
```

In [28]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
import statsmodels.formula.api as smf
import math
from sklearn.metrics import r2_score
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: Futu
reWarning: pandas.util.testing is deprecated. Use the functions in the public
API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [29]:
```
print('Mean absolute error ', mean_absolute_error(Xtest.rating, predictions))
print('Mean suared error ', mean_squared_error(Xtest.rating, predictions))
print('Suare Root of Mean suqared error ',math.sqrt(mean_squared_error(Xtest.r
ating, predictions)))
print('Mean value or Output test variable ', np.mean(Xtest.rating))
print('Mean value or Predicted Output variable ',np.mean(predictions))
print('Rating model accuracy with 2 Hidden layer ',r2_score(Xtest.rating,predi
ctions))
```

```
Mean absolute error  0.7219467108864022
Mean suared error  0.9245619275342032
Suare Root of Mean suqared error  0.9615414330824248
Mean value or Output test variable  3.8581476124511083
Mean value or Predicted Output variable  3.8807728
Rating model accuracy with 2 Hidden layer  0.04126874771529332
```

Summarize the findings

To sum up, #### In most cases below predicted values are close to actual values of rating

Below are 5 sample predicted values and actual values of Rating

[3.927527] 4

[4.9457016] 5

[4.2614436] 3

[4.9071617] 5

[4.7594547] 5

Mean Test Value of Rating 3.8581476124511083

Mean Predicted Value of Rating 3.776287