# RECOMMENDATION SYSTEM USING DEEP NEURAL NETWORK

Sarath R. K. Lella, Avinash Reddy T

School of Computing and Engineering: University of Missouri – Kansas City

2022 Spring CS-5542: Big Data Analytics and Applications

Prof. Syed Jawad Shah

# Contents

INTRODUCTION ..... 3

Goal and Objective ..... 3

Motivation ..... 3

Significance ..... 3

FEATURES AND DATA ..... 4

Features ..... 4

Data file Snapshot ..... 5

Item file snapshot ..... 5

METHODOLOGY ..... 7

Workflow ..... 7

Proposed model ..... 10

EXPERIMENTS ..... 11

PREDICTION ..... 13

SCREENSHOTS ..... 13

LEARNINGS FROM PROJECT ..... 16

CHALLENGES FACED ..... 17

CONCLUSION ..... 17

REFERENCES ..... 17

# INTRODUCTION

Video Explanation: https://youtu.be/K-pBzekq5NI

GitHub Repo Link: Lsarath25/Movie-Recommendation-System: Big Data Project for movie recommendation (github.com)

## Goal and Objective

The goal is to create a recommendation system for the Movie Lens dataset; however, the model may be applied to any dataset. Our Objective is to create a recommendation system that uses movie and user information to suggest movies to new users based on their likes and prior choices.

## Motivation

Recommender systems are designed to anticipate users' interests and propose products that are likely to be of interest to them. They are one of the most sophisticated machine learning algorithms used by internet businesses to boost sales.

## Significance

Companies that use recommender systems aim to increase sales by providing more tailored offers and a better customer experience. The consumer begins to feel recognized and understood, increasing their likelihood of purchasing further items or consuming more information. The firm receives a competitive edge by understanding what a user wants, and the risk of losing a customer to a rival is reduced.

It's enticing to provide that extra value to users by integrating suggestions in systems and goods. Furthermore, it enables businesses to stay ahead of their competition and, as a result, enhance their profits.

## FEATURES AND DATA

### Features

We utilize the MovieLens-100k dataset, which is made up of a set of user ratings for movies taken from the MovieLens website, which is a movie recommendation service. There are 943 users and 1664 movies on it. As features, we employ a variety of film genres. The GroupLens Research Project at the University of Minnesota gathered MovieLens data sets.

* 100,000 ratings (1-5) from 943 users on 1682 films make up this data collection.

* Each user has given a rating to at least 20 films.

* For the users, simple demographic information is provided (age, gender, occupation, zip)

During the seven-month period from September 19th, 1997 to April 22nd, 1998, data was collected via the MovieLens web site (movielens.umn.edu). Users with fewer than 20 ratings or who did not provide complete demographic information were deleted from the data set. At the end of this file, there are detailed descriptions of the data file.

## Data file Snapshot

| | user id | movie id | rating | timestamp |
|---|---|---|---|---|
| **0** | 196 | 242 | 3 | 881250949 |
| **1** | 186 | 302 | 3 | 891717742 |
| **2** | 22 | 377 | 1 | 878887116 |
| **3** | 244 | 51 | 2 | 880606923 |
| **4** | 166 | 346 | 1 | 886397596 |

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   user id    100000 non-null  int64
 1   movie id   100000 non-null  int64
 2   rating     100000 non-null  int64
 3   timestamp  100000 non-null  int64
dtypes: int64(4)
memory usage: 3.1 MB
```

## Item file snapshot

| | movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children | ... | Fantasy | Film-Noir | Horror | Mu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Toy Story (1995) | 01-Jan-1995 | NaN | http://us.imdb.com/M/title-exact?Toy%20Story%2... | 0 | 0 | 0 | 1 | 1 | ... | 0 | 0 | 0 | |
| **1** | 2 | GoldenEye (1995) | 01-Jan-1995 | NaN | http://us.imdb.com/M/title-exact?GoldenEye%20(... | 0 | 1 | 1 | 0 | 0 | ... | 0 | 0 | 0 | |
| **2** | 3 | Four Rooms (1995) | 01-Jan-1995 | NaN | http://us.imdb.com/M/title-exact?Four%20Rooms%... | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **3** | 4 | Get Shorty (1995) | 01-Jan-1995 | NaN | http://us.imdb.com/M/title-exact?Get%20Shorty%... | 0 | 1 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |
| **4** | 5 | Copycat (1995) | 01-Jan-1995 | NaN | http://us.imdb.com/M/title-exact?Copycat%20(1995) | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | |

5 rows × 24 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1682 entries, 0 to 1681
Data columns (total 24 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   movie id            1682 non-null   int64  
 1   movie title         1682 non-null   object 
 2   release date        1681 non-null   object 
 3   video release date  0 non-null      float64
 4   IMDb URL            1679 non-null   object 
 5   unknown             1682 non-null   int64  
 6   Action              1682 non-null   int64  
 7   Adventure           1682 non-null   int64  
 8   Animation           1682 non-null   int64  
 9   Children            1682 non-null   int64  
 10  Comedy              1682 non-null   int64  
 11  Crime               1682 non-null   int64  
 12  Documentary         1682 non-null   int64  
 13  Drama               1682 non-null   int64  
 14  Fantasy             1682 non-null   int64  
 15  Film-Noir           1682 non-null   int64  
 16  Horror              1682 non-null   int64  
 17  Musical             1682 non-null   int64  
 18  Mystery             1682 non-null   int64  
 19  Romance             1682 non-null   int64  
 20  Sci-Fi              1682 non-null   int64  
 21  Thriller            1682 non-null   int64  
 22  War                 1682 non-null   int64  
 23  Western             1682 non-null   int64  
dtypes: float64(1), int64(20), object(3)
memory usage: 315.5+ KB
```

The movie genre is represented by the following characteristic. The value is 1 if the film belongs to that genre, 0 if it does not. The genre considered are:

1) Action
2) Adventure
3) Animation
4) Children
5) Comedy
6) Crime
7) Documentary
8) Drama
9) Fantasy
10) Film-Noir
11) Horror
12) Musical
13) Mystery
14) Romance
15) Sci-Fi

16) Thriller
17) War
18) Western

Here, We divide dataset into (Using the 90-10-10 split) -

- Training data
- Validation data
- Testing data

The hyper parameters are chosen based on the validation data, and the final accuracy for the best hyper parameters is reported.

## METHODOLOGY

### Workflow

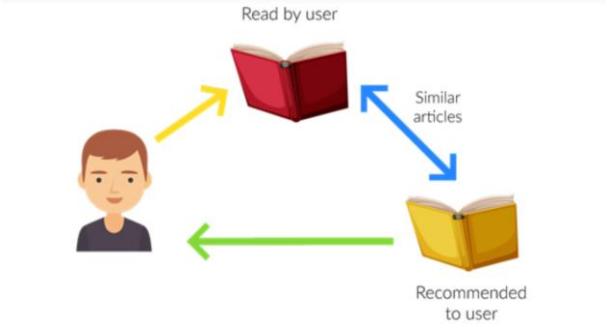
Recommender systems use two types of data to function:

1. *Item and user characteristics* - This is information on items (keywords, categories, and preferences, profiles, etc.).

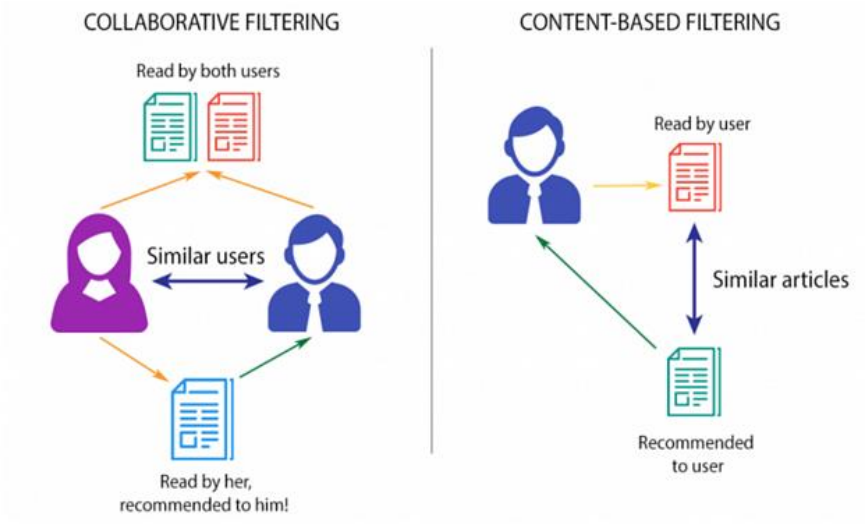
2. *User-item interactions* - This includes data such as ratings, purchases, likes, and so on.

There are two types of recommender systems:

1. *Content-based* - This strategy recommends items based on the content of a specific item.




2. *Collaborative Filtering* - This forecasts a user's rating for a certain item based on that user's previous ratings and the preferences of other users. It captures the user-item interaction matrix in a nutshell.

When two different types of entities are multiplied, matrix factorization is used to generate latent features. Matrix factorization is used in collaborative filtering to determine the link between item and user entities. We'd like to forecast how customers would evaluate shop items based on the input of user ratings so that consumers can receive recommendations based on the prediction. Matrix Factorization will be utilized in our example to learn latent features from user, item, and rating data.

Create a list of users (U), items (D), and the R size of |U| and |D|. The matrix |U|*|D| contains all of the user ratings. The objective is to find K latent features. It would construct the product result R given the input of two matrices, P (|U|*k) and Q (|D|*k).

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

The association between a user and the features is represented by matrix P, whereas the association between an object and the features is represented by matrix Q. Calculating the dot product of the two vectors corresponding to u_i and d_j yields the prediction of an item's rating.

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

- Recommender systems are intended to predict users' preferences and recommend things that are likely to appeal to them. They're one of the most advanced machine learning algorithms utilized by online retailers to increase sales.

- We're working on a recommendation system. The model was tested on the Movie Lens dataset, but it can be applied to a variety of other datasets as well.

**Proposed model**

We use a deep neural network to conduct matrix factorization in the suggested model. We begin by employing a label encoder for both the user and the movies. This encoding is used as an embedding input.

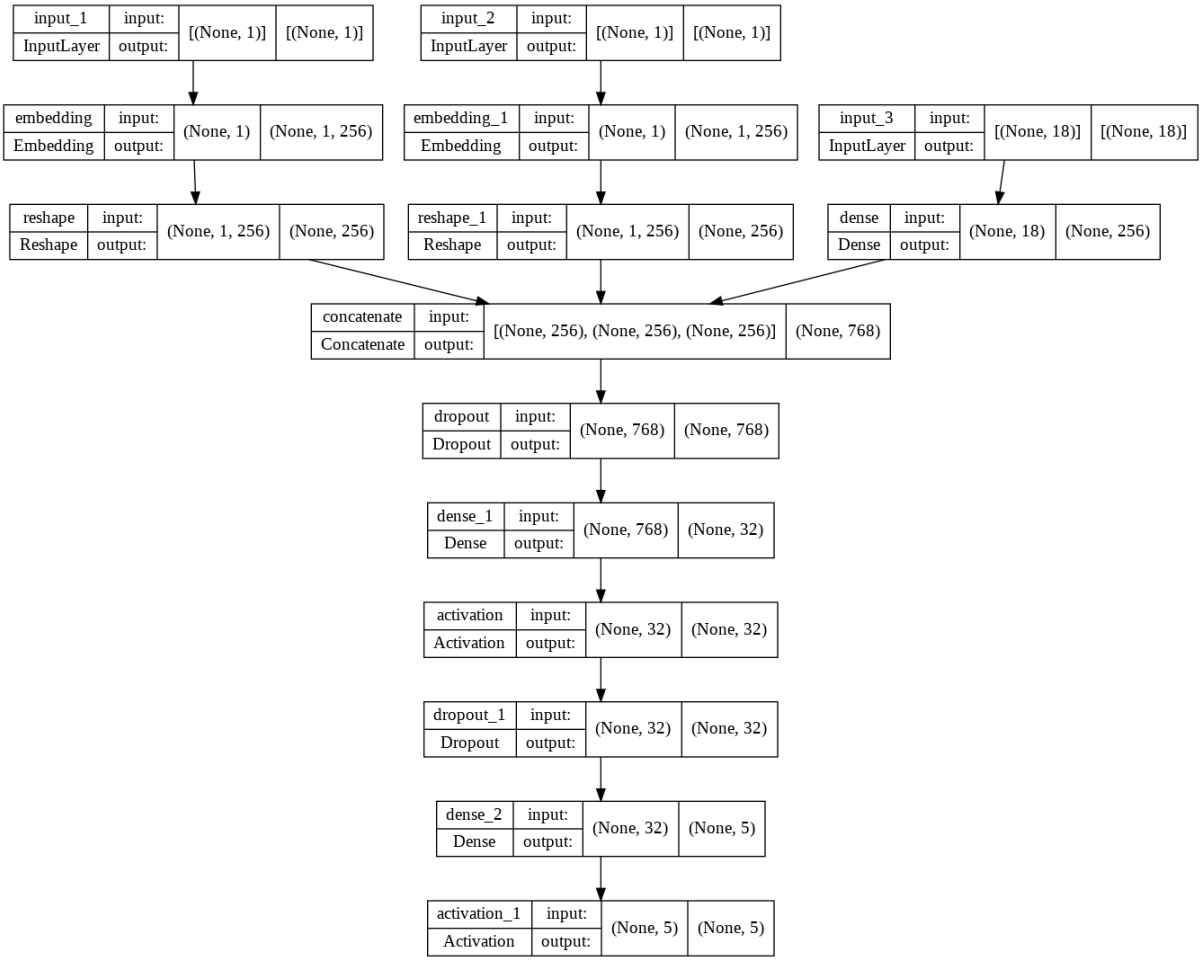
The embedding's listed below are what we learn-

- user embedding
- movie embedding
- embedding for feature space

The size of the embedding is a hyperparameter, and we fine-tune the hyperparameters using the validation dataset.

- Later, these latent representations of movies, movie features, and users are combined. The concatenated embedding method is used to train the model based on a user's rating of a movie.
- The Dense layer is next applied, which is started with He Normal initialization, which gets samples from a truncated normal distribution centered on 0.
- To do so, we employ the Sparse Cross Entropy Loss and the SGD optimizer. For better generalization performance, we implemented dropout and L2 regularization in our implementation. We also normalize the input to speed up the optimization process.
- The final output layer consists of a dense layer with a size equal to the number of ratings (5 in our case) and a sigmoid activation function that produces an output between 0 and 1, representing normalized rating.
- We utilize the user-id of the person for whom movies must be forecasted and the amount of movies that must be recommended as input. We employ a label encoder to retrieve a user's

encoding representation for prediction. Then we assign a rating to each film. The movies that are suggested are those that have received high ratings.

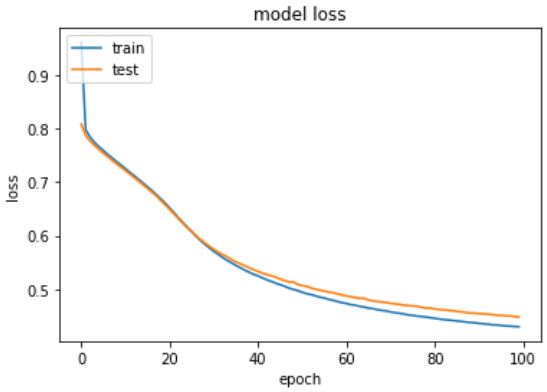



## EXPERIMENTS

**Test 1:**

Epochs: 100, Embed_dim = 128, Dropout = 0.05, L2 reg = 1e-3

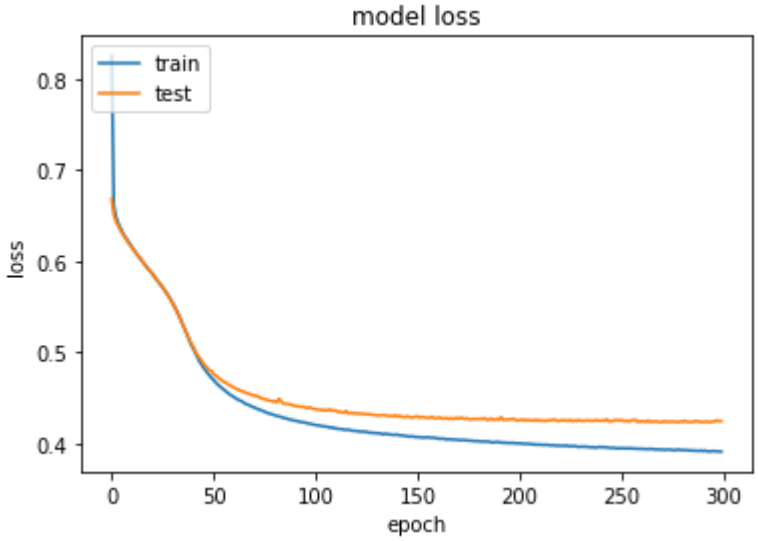
Val_accuracy: 0.1283




**Test 2:**

Epochs: 300, Embed_dim = 64, Dropout = 0.05, L2 reg = 1e-3

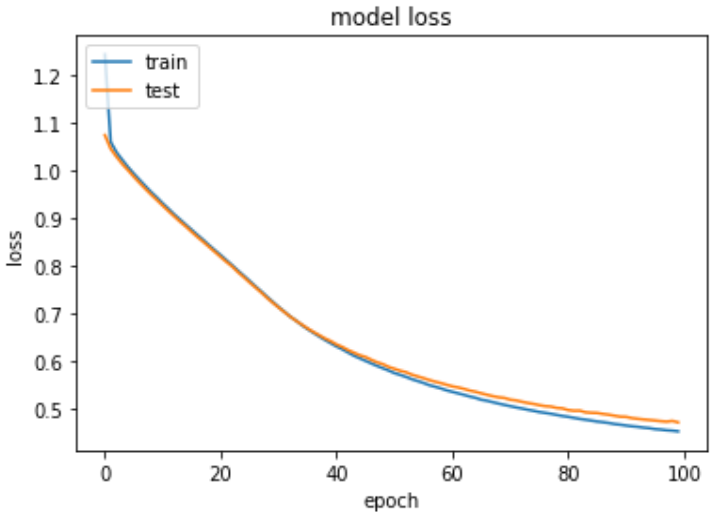
Val_accuracy: 0.1372




**Test 3:**

Epochs: 300, Embed_dim = 256, Dropout = 0.05, L2 reg = 1e-3

Val_accuracy: 0.1198




## PREDICTION

We ask the user to provide a list of recommended movies as well as the quantity of movies that should be forecasted. Using the trained model, the model predicts the top k predictions. The output screen appears as follows –

```
----------------------------------------------------------------------------------
Top 10 Movie recommendations for the User 10 are:
['Fargo (1996)',
 'M*A*S*H (1970)',
 "Devil's Own, The (1997)",
 'Getting Even with Dad (1994)',
 'Lawnmower Man 2: Beyond Cyberspace (1996)',
 'Air Bud (1997)',
 'Anne Frank Remembered (1995)',
 "Monty Python's Life of Brian (1979)",
 'Rear Window (1954)',
 'Air Force One (1997)']
```

Test Accuracy of our model on test data is 0.1302

## SCREENSHOTS

- We have combined the given datasets into single dataframe into a final dataset. We have used this dataset to perform all the required operations.

```
final_dataset = combined_dataset_df
final_dataset.head() #display first 5 rows of the final dataset
```

| | user id | movie id | rating | timestamp | movie title | Action | Adventure | Animation | Children | Comedy | ... | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 196 | 242 | 3 | 881250949 | Kolya (1996) | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 63 | 242 | 3 | 875747190 | Kolya (1996) | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 226 | 242 | 5 | 883888671 | Kolya (1996) | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 154 | 242 | 3 | 879138235 | Kolya (1996) | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 306 | 242 | 5 | 876503793 | Kolya (1996) | 0 | 0 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 23 columns

- For creating a model, we have created a RecommenderSystem class, using which we created object and methods to initialize the object as a model and used a prediction function for the prediction. We have used a constructor for RecommenderSystem class to initialize the class variables.

```
class RecommenderSystem():
  def __init__(self, embed_dim, n_users, n_movies, num_features): #constructor for embedded dimenstions, users, movies, num_features
    self.embed_dim = embed_dim
    self.n_users = n_users
    self.n_movies = n_movies
    self.num_features = num_features
```

- *Defining the Model*: After creating the RecommenderSystem object, we will call the set_method which is used to create a model, as shown in the below figure.

```
def set_model(self): #model definition
  user = tf.keras.layers.Input(shape = (1,)) #input layer #1 user
  movie = tf.keras.layers.Input(shape = (1,))  #input layer #2 movie
  movie_features = tf.keras.layers.Input(shape = (num_features,)) #input layer #3 movie features

  embed_user = keras.layers.embeddings.Embedding(n_users, embed_dim, embeddings_regularizer = tf.keras.regularizers.l2(1e-3))(user)
  embed_user = tf.keras.layers.Reshape((embed_dim ,))(embed_user) #embedding for users

  embed_movie = keras.layers.embeddings.Embedding(n_movies, embed_dim, embeddings_regularizer=tf.keras.regularizers.l2(1e-3))(movie)
  embed_movie = tf.keras.layers.Reshape((embed_dim,))(embed_movie) #embedding for movie

  embed_movie_features = tf.keras.layers.Dense(embed_dim)(movie_features) #embedding for movie features

  x = tf.keras.layers.Concatenate()([embed_user,embed_movie,embed_movie_features]) #combining the embedding layers
  x = tf.keras.layers.Dropout(0.05)(x)  #adding dropout layer 5%

  x = tf.keras.layers.Dense(32, kernel_initializer='he_normal')(x) #adding dense layer
  x = tf.keras.layers.Activation(activation='relu')(x) #using relu activation function
  x = tf.keras.layers.Dropout(0.05)(x) #adding dropoutlayer 5%

  x = tf.keras.layers.Dense(5)(x)
  x = tf.keras.layers.Activation(activation='softmax')(x) #softmax activation for output layer

  self.model = tf.keras.models.Model(inputs=[user,movie, movie_features], outputs=x)

  self.model.compile(optimizer='sgd', loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy']) #model compile
  print(self.model.summary()) #printing model summary
  return self
```

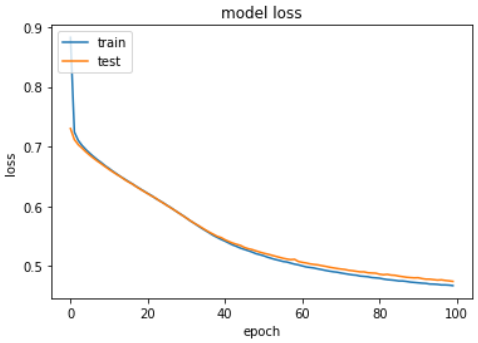
- Model Summary:

```
Model: "model_8"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_25 (InputLayer)          [(None, 1)]          0           []

 input_26 (InputLayer)          [(None, 1)]          0           []

 embedding_16 (Embedding)       (None, 1, 256)       241408      ['input_25[0][0]']

 embedding_17 (Embedding)       (None, 1, 256)       2560        ['input_26[0][0]']

 input_27 (InputLayer)          [(None, 18)]         0           []

 reshape_16 (Reshape)           (None, 256)          0           ['embedding_16[0][0]']

 reshape_17 (Reshape)           (None, 256)          0           ['embedding_17[0][0]']

 dense_24 (Dense)               (None, 256)          4864        ['input_27[0][0]']

 concatenate_8 (Concatenate)    (None, 768)          0           ['reshape_16[0][0]',
                                                                  'reshape_17[0][0]',
                                                                  'dense_24[0][0]']

 dropout_16 (Dropout)           (None, 768)          0           ['concatenate_8[0][0]']

 dense_25 (Dense)               (None, 32)           24608       ['dropout_16[0][0]']

 activation_16 (Activation)     (None, 32)           0           ['dense_25[0][0]']

 dropout_17 (Dropout)           (None, 32)           0           ['activation_16[0][0]']

 dense_26 (Dense)               (None, 5)            165         ['dropout_17[0][0]']

 activation_17 (Activation)     (None, 5)            0           ['dense_26[0][0]']

==================================================================================================
Total params: 273,605
Trainable params: 273,605
Non-trainable params: 0
__________________________________________________________________________________________________
```

- Finally, for the prediction function, we have used the predict_recommendation function, which takes the userid and the number of movie recommendations required. We have initialized the required model_input and used the model.predict() to find the predicted values. Later, we sorted the values and return required data.

```
def predict_recommendation(self, user_id, n_movies):
  encoded_user_id = user_enc.transform([user_id]) #input transform
  seen_movies = list(final_dataset[final_dataset['user id'] == user_id]['movie']) #list of seen movies
  unseen_movies = [i for i in range(min(final_dataset['movie']), max(final_dataset['movie'])+1) if i not in seen_movies] #list of unseen moces
  unseen_movies_features = np.empty((0, num_features))
  feature_names=['Action','Adventure','Animation','Children','Comedy','Crime','Documentary','Drama','Fantasy','Film-Noir','Horror','Musical','Mystery','Romance','Sci-Fi','Thriller'
  for i in unseen_movies:
    unseen_movies_features = np.append(unseen_movies_features, np.array(final_dataset.loc[final_dataset['movie'] == i ,  feature_names].iloc[0]).reshape(1, num_features), axis=0)
  model_input = [np.asarray(list(encoded_user_id)*len(unseen_movies)), np.asarray(unseen_movies),np.asarray(unseen_movies_features)] #preparing the model input
  predicted_ratings = self.model.predict(model_input) #model predict
  predicted_ratings = np.max(predicted_ratings, axis=1) #predicted ratings
  sorted_index = np.argsort(predicted_ratings)[::-1] #sorting the index of the predicted ratings
  recommended_movies = item_enc.inverse_transform(sorted_index) #reccommended movies in the sorting order
  print("----------------------------------------------------------------------------")
  print("Top "+str(n_movies)+" Movie recommendations for the User "+str(user_id)+ " are:")
  pprint(list(recommended_movies[:n_movies]))
```

- Training Loss vs Validation Loss Plot:




## LEARNINGS FROM PROJECT

We learned how to create a recommendation system from the ground up for a new user. A basic recommendation system may be built in a variety of ways. The representation of latent characteristics is the fundamental challenge of recommendation systems. This hidden representation can be acquired in a variety of ways. This is done with the help of a neural network.

Because a neural network is a universal approximator, it learns a better representation and so performs better in generalization.

## CHALLENGES FACED

- SVD can also be used for matrix factorization. However, because neural networks are more generic, we decided to use them for our recommendation system.
- The addition of movie functionality, as well as user embedding and movie embedding, added to the difficulty.
- We have also had a slight difficulty in changing the input shapes and creating the model inputs with similar dimensions but have accomplished the task after putting some time and effort into it.

## CONCLUSION

For the Movie Lens dataset, we built an AI-based recommendation engine that recommends movies to a user. To do this, we used movie and user features. On the validation dataset, we experimented with various hyperparameter values before settling on the ideal option for test data. A test accuracy of 0.1302 is reported.

## REFERENCES


- https://d2l.ai/chapter_recommender-systems/recsys-intro.html
- https://towardsdatascience.com/recommendation-system-series-part-6-the-6-variants-of-autoencoders-for-collaborative-filtering-bd7b9eae2ec7

- https://colab.research.google.com/github/keras-team/keras-io/blob/master/guides/ipynb/intro_to_keras_for_engineers.ipynb