Movie Recommendation

NAME: Arpita Chakraborty
B-NUMBER: B00709870

Problem Statement

- Given MovieLens dataset containing user info, movie info and rating
- Recommend movies for the user based on its past user ratings.
- Predict rating value for the unseen movies of the given user. Select top n movies as recommended movie for that user.

MovieLens Dataset

Dataset Link : https://grouplens.org/datasets/movielens/latest/

This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018.

The dataset contains the following info:

- User Id.
- Movie Information. (movieId,title,genres).
- Rating parameter given by the user. (userId,movieId,rating,timestamp)
- Movie Tag(userId,movieId,tag,timestamp)

Data Visualization - Dimensionality Reduction (PCA)

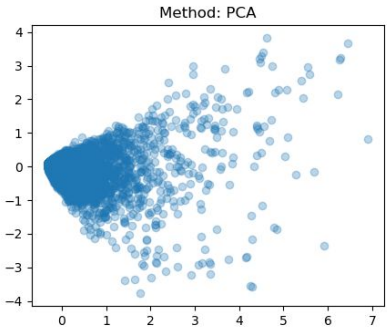
```
(   userId  movieId  rating   timestamp
 0       1       31     2.5  1260759144
 1       1     1029     3.0  1260759179
 2       1     1061     3.0  1260759182
 3       1     1129     2.0  1260759185
 4       1     1172     4.0  1260759205,
    movieId                               title  \
 0        1                    Toy Story (1995)
 1        2                      Jumanji (1995)
 2        3             Grumpier Old Men (1995)
 3        4            Waiting to Exhale (1995)
 4        5  Father of the Bride Part II (1995)

                                        genres
 0  Adventure|Animation|Children|Comedy|Fantasy
 1                   Adventure|Children|Fantasy
 2                               Comedy|Romance
 3                        Comedy|Drama|Romance
 4                                      Comedy  )
```

ratings.csv

movies.csv

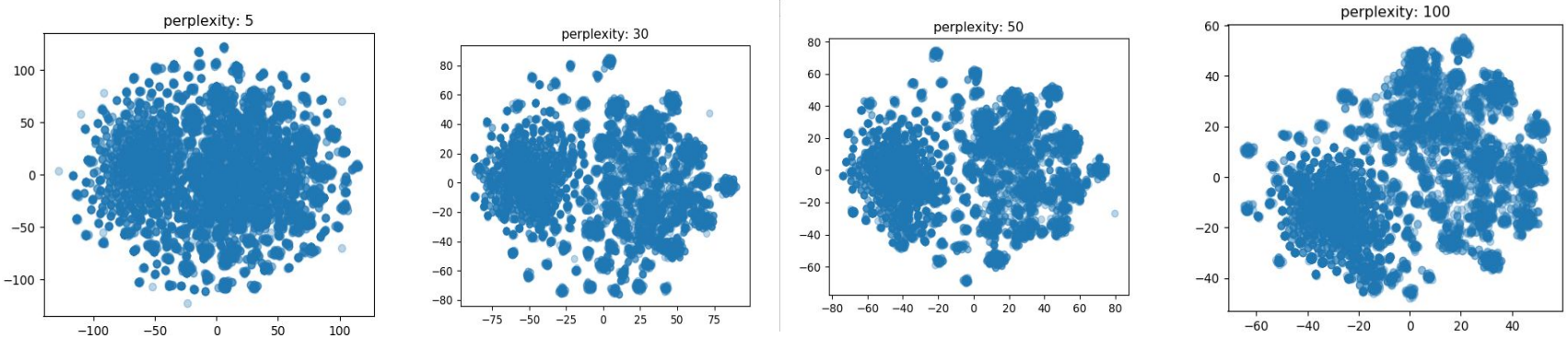
Feature Vector Size: [9724,(610*10)] =[9724,6100]
9724= no of unique users
610 = no of unique movies
10 = no of unique ratings



Reduced the dimensions to 2 using PCA:
But It does not reveal any cluster.

PCA (essentially) maps data onto a plane of a lower dimension, preserving the distances between the datapoints. The orientation of the lower dimensional plane is chosen to reflect the structure of the higher dimensional data.

Data Visualization - Dimensionality Reduction (t-SNE)

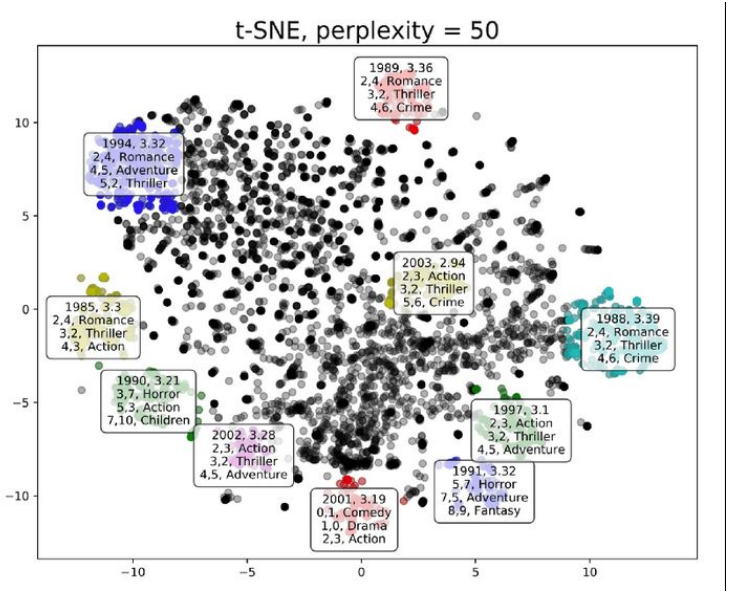


Higher perplexities seem to more clearly show clusters.

t-SNE is a non-linear algorithm which considers the similarity of different objects, and uses this to decide on their distance in the 2D (or 3D) plane. A probability distribution (where similar objects have a high chance of being picked) is used to gauge the similarity of objects, and then plots the points on a lower dimension so that the probability distribution is similar to in the higher plane. The recommended values of perplexity are between 5–50; since a perplexity of 50 yielded well defined clusters, I ran the t-SNE algorithm at a perplexity of 50 for 5000 iterations to ensure it converged.

Analyze the Dataset

- **Average_release_Year** = 1994.786507752835
- **Average_rating** = 3.2624482748109633
- **Average_genre_distribution** = [('Drama', 4349), ('Comedy', 3753), ('Thriller', 1889), ('Action', 1828), ('Romance', 1591), ('Adventure', 1262), ('Crime', 1196), ('Sci-Fi', 980), ('Horror', 977), ('Fantasy', 778), ('Children', 664), ('Animation', 610), ('Mystery', 573), ('Documentary', 438), ('War', 381), ('Musical', 333), ('Western', 167), ('IMAX', 158), ('Film-Noir', 85), ('(no genres listed)', 34)]



Each annotation has the average year of release, the average rating, and the first 3 genres in the 'difference array'. Across the entire dataset, the average release year of a movie is 1992, and the average rating is 3.29.

Collaborative Filtering Model

The collaborative filtering algorithm uses “User Behavior” for recommending items. This is one of the most commonly used algorithms in the industry as it is not dependent on any additional inform

User-User collaborative filtering: This algorithm first finds the similarity score between users. Based on this similarity score, it then picks out the most similar users and recommends products which these similar users have liked or bought previously.The prediction of an item for a user *u* is calculated by computing the weighted sum of the user ratings given by other users to an item *i*.

The prediction *Pu,i* is given by $$P_{u,i} = \frac{\sum_v (r_{v,i} * s_{u,v})}{\sum_v s_{u,v}}$$

- *Pu,i* is the prediction of an item
- *Rv,i* is the rating given by a user *v* to a movie *i*
- *Su,v* is the similarity between users

Collaborative Filtering Model- Contd

Item-Item collaborative filtering

In this algorithm, the similarity is computed between each pair of items.The similarity is found between each movie pair and based on that the movies are recommended based on the likes of users in the past.This algorithm works similar to user-user collaborative filtering with just a little change – instead of taking the weighted sum of ratings of “user-neighbors”,weighted sum of ratings of “item-neighbors” have been considered. The prediction is given by:

$$P_{u,i} = \frac{\sum_N (s_{i,N} * R_{u,N})}{\sum_N (|s_{i,N}|)}$$

$$sim(i,j) = cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{||\vec{i}||_2 * ||\vec{j}||_2}$$

Collaborative Filtering Model- Contd

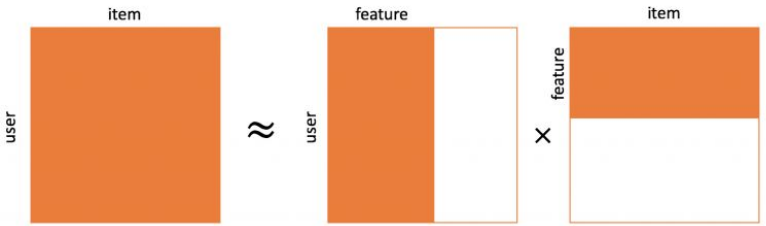
1. Recommend movies based on the user-user similarity and item-item similarity.
2. Create the user Item matrix which can be used to calculate the similarity between users and items.
3. Compute the similarity (cosine similarity) which generates item-item and user-user similarity in an array form.
4. Make predictions based on user similarity and item similarity.
5. Train the item similarity model and make prediction.
6. The example below shows prediction for user_id 1 to 5.

user_id	movie_id	score	rank
1	423	0.9850328512319172	1
1	202	0.9431346880115625	2
1	655	0.7932585664377868	3
1	403	0.765623665037956	4
1	568	0.7633005562629408	5
2	50	1.1256258487701416	1
2	181	1.0651773168490484	2
2	7	0.9754774272441864	3
2	121	0.94162796323116	4
2	9	0.8319899130326O5	5
3	313	0.6353766620159149	1
3	328	0.6032880300825293	2
3	315	0.5422587123784152	3
3	331	0.5355071858926252	4
3	332	0.5316696112806146	5
4	50	1.1311477082116264	1
4	288	1.0487151145935059	2
4	181	0.9505999386310577	3
4	7	0.9417778807027	4
4	302	0.9139021464756557	5
5	195	1.0158377741322373	1
5	202	0.9353599468866984	2
5	56	0.8498673283692563	3
5	82	0.7769144300258521	4
5	96	0.7397452755407854	5

[25 rows x 4 columns]

Matrix Factorization

- Popular recommender systems approach.
- A low-dimensional representation (embedding) of user and movie is learnt. The user and the movie embeddings can be combined to estimate the ratings on unseen movies.
- This approach can also be viewed as: given a matrix (A [M X N]) containing users and movies, estimate low dimensional matrices (W [M X k] and H [M X k]), such that: A≈W.HT (HT is transposed of H)
- The key thing is to learn an embedding for movies and users, and then combine them using the dot product.The model would optimise the embeddings such that the MSE on ratings gets minimized from the train set.



Assume, K latent feature. Divide the rating matrix R(MxN) into P(MxK) and Q(NxK) such that P x QT (here QT is the transpose of Q matrix) approximates the R matrix:

$$R=P\Sigma Q^T$$

Matrix Factorization - Model Visualization

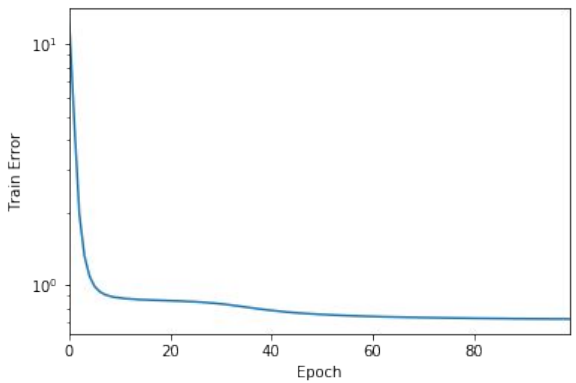
```
Using TensorFlow backend.
```

Layer (type)	Output Shape	Param #	Connected to
Item (InputLayer)	(None, 1)	0	
User (InputLayer)	(None, 1)	0	
NonNegMovie-Embedding (Embeddin	(None, 1, 3)	5049	Item[0][0]
NonNegUser-Embedding (Embedding	(None, 1, 3)	2832	User[0][0]
FlattenMovies (Flatten)	(None, 3)	0	NonNegMovie-Embedding[0][0]
FlattenUsers (Flatten)	(None, 3)	0	NonNegUser-Embedding[0][0]
DotProduct (Merge)	(None, 1)	0	FlattenMovies[0][0] FlattenUsers[0][0]

```
Total params: 7,881
Trainable params: 7,881
Non-trainable params: 0
```

Matrix Factorization - Results

Train error v/s epoch number



User Embeddings

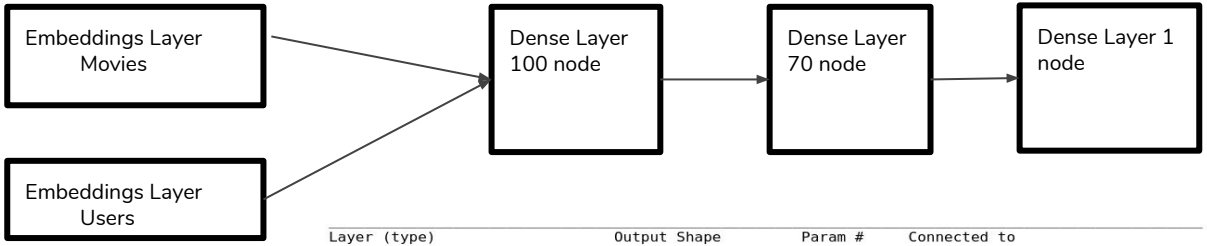
	0	1	2
count	944.000000	944.000000	944.000000
mean	-1.126231	1.171609	1.109131
std	0.517478	0.409016	0.548384
min	-2.883226	-0.500010	-0.415373
25%	-1.458197	0.903574	0.735729
50%	-1.159480	1.199517	1.084089
75%	-0.836746	1.456610	1.468611
max	0.899436	2.605330	2.826109

Movie Embeddings

	0	1	2
count	1683.000000	1683.000000	1683.000000
mean	-0.935420	0.857862	0.954169
std	0.517458	0.447439	0.458095
min	-2.524487	-0.459752	-0.989537
25%	-1.323431	0.546364	0.642444
50%	-0.949188	0.851243	0.993619
75%	-0.550862	1.159588	1.283555
max	0.500618	2.140607	2.683658

MSE: 0.69

Neural Network Model

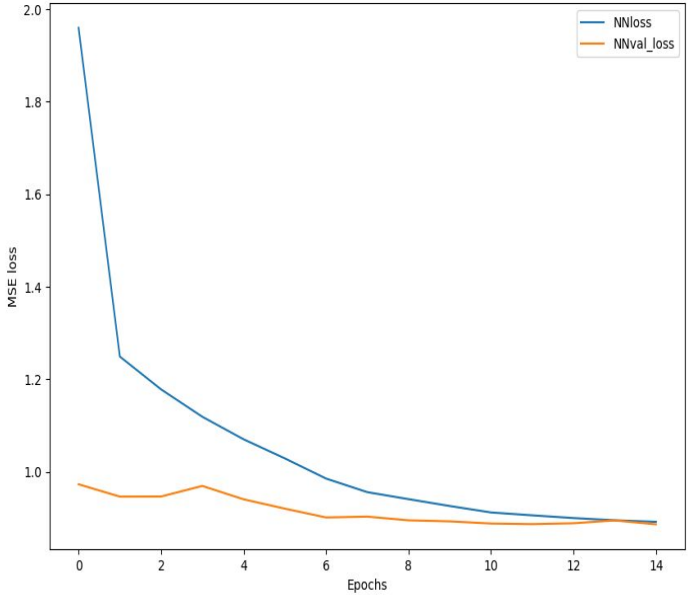


Layer (type)	Output Shape	Param #	Connected to
user_input (InputLayer)	(None, 1)	0	
movie_input (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 50)	47150	user_input[0][0]
embedding_2 (Embedding)	(None, 1, 50)	84100	movie_input[0][0]
merge_1 (Merge)	(None, 1, 100)	0	embedding_1[0][0] embedding_2[0][0]
flatten_1 (Flatten)	(None, 100)	0	merge_1[0][0]
dropout_1 (Dropout)	(None, 100)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 100)	10100	dropout_1[0][0]
dropout_2 (Dropout)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 70)	7070	dropout_2[0][0]
dropout_3 (Dropout)	(None, 70)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	71	dropout_3[0][0]

4 layer -
a> Input Layer: user and movie embedding layer.
b>2 Fully connected layer
c>Output Layer(Rating)

Optimizer: Adam

Neural Network Model - Result

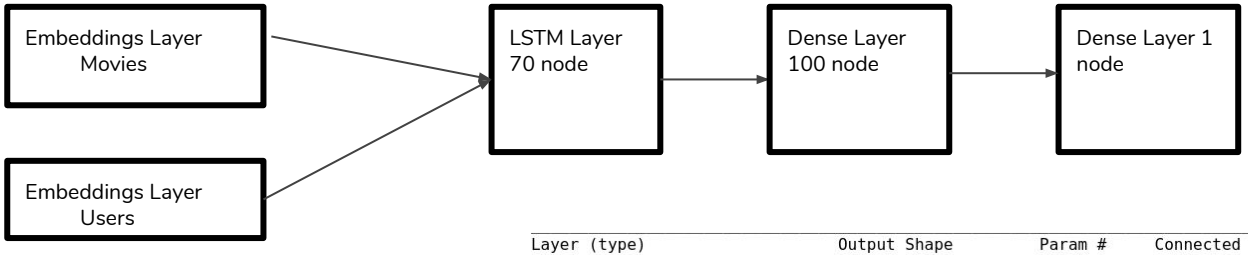


MSE loss Vs Epochs

NNloss- Training loss
NNVal_loss - Validation loss

MSE: 0.72

Recurrent Neural Network(LSTM) Model

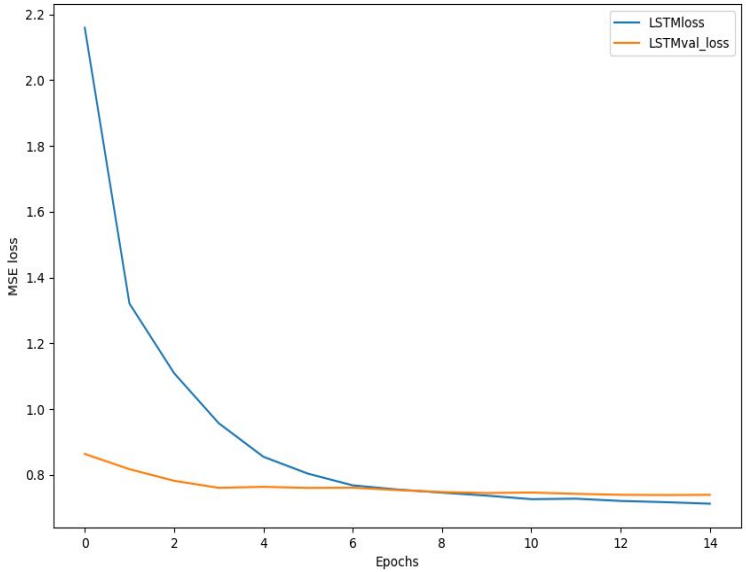


4 layer:
a>Input layer- Movie and user embeddings layer
b>LSTM layer - 70 nodes
c> Dense Layer - 100 nodes.
c>Output layer(Rating)

Optimizer: Adam

Layer (type)	Output Shape	Param #	Connected to
user_input (InputLayer)	(None, 1)	0	
movie_input (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 50)	30500	user_input[0][0]
embedding_2 (Embedding)	(None, 1, 50)	486200	movie_input[0][0]
merge_1 (Merge)	(None, 1, 100)	0	embedding_1[0][0] embedding_2[0][0]
dropout_1 (Dropout)	(None, 1, 100)	0	merge_1[0][0]
lstm_1 (LSTM)	(None, 70)	47880	dropout_1[0][0]
dropout_2 (Dropout)	(None, 70)	0	lstm_1[0][0]
dense_1 (Dense)	(None, 100)	7100	dropout_2[0][0]
dropout_3 (Dropout)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 1)	101	dropout_3[0][0]

Recurrent Neural Network(LSTM) Result



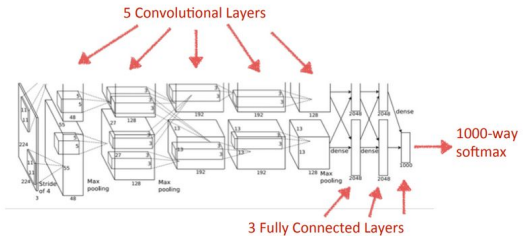
MSE loss Vs Epochs

LSTMloss- Training loss
LSTMVal_loss - Validation loss

MSE: 0.66

Convolutional Neural Network - Alexnet

- AlexNet is one of the most important & influential neural network architectures that demonstrate the power of convolutional layers in machine vision.
- AlexNet consist of 5 convolutional layers and 3 dense layers.
- The input data is 3-dimensional and has been flattened before passing it into the dense layer.
- Used adam for optimiser and accuracy for performance metrics.
- Input Images are Movie Poster Images (224*224*3) and Output layer Dense layer with 1 node=>the movie rating




```
Using TensorFlow backend.
```

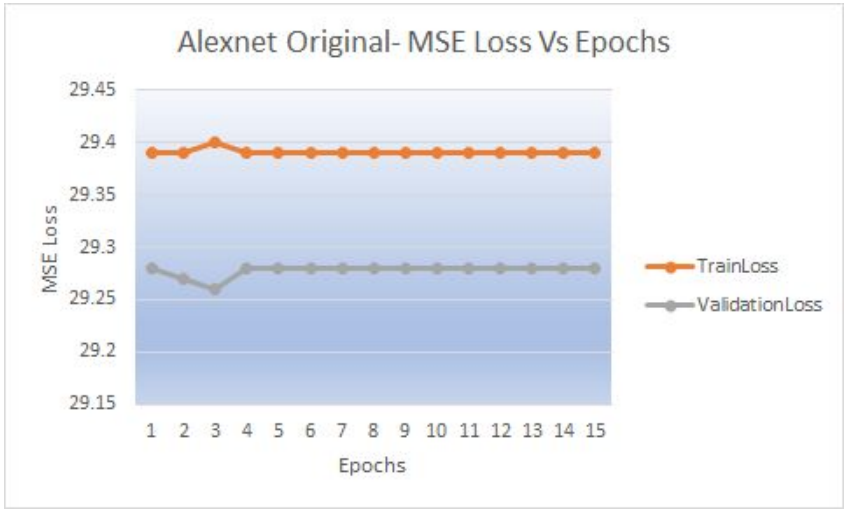
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 54, 54, 96)	34944
activation_1 (Activation)	(None, 54, 54, 96)	0
max_pooling2d_1 (MaxPooling2	(None, 27, 27, 96)	0
conv2d_2 (Conv2D)	(None, 17, 17, 256)	2973952
activation_2 (Activation)	(None, 17, 17, 256)	0
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 256)	0
conv2d_3 (Conv2D)	(None, 6, 6, 384)	885120
activation_3 (Activation)	(None, 6, 6, 384)	0
conv2d_4 (Conv2D)	(None, 4, 4, 384)	1327488
activation_4 (Activation)	(None, 4, 4, 384)	0
conv2d_5 (Conv2D)	(None, 2, 2, 256)	884992
activation_5 (Activation)	(None, 2, 2, 256)	0
max_pooling2d_3 (MaxPooling2	(None, 1, 1, 256)	0
flatten_1 (Flatten)	(None, 256)	0
dense_1 (Dense)	(None, 4096)	1052672
activation_6 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
activation_7 (Activation)	(None, 4096)	0
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 1000)	4097000
activation_8 (Activation)	(None, 1000)	0
dropout_3 (Dropout)	(None, 1000)	0
dense_4 (Dense)	(None, 1)	1001
activation_9 (Activation)	(None, 1)	0

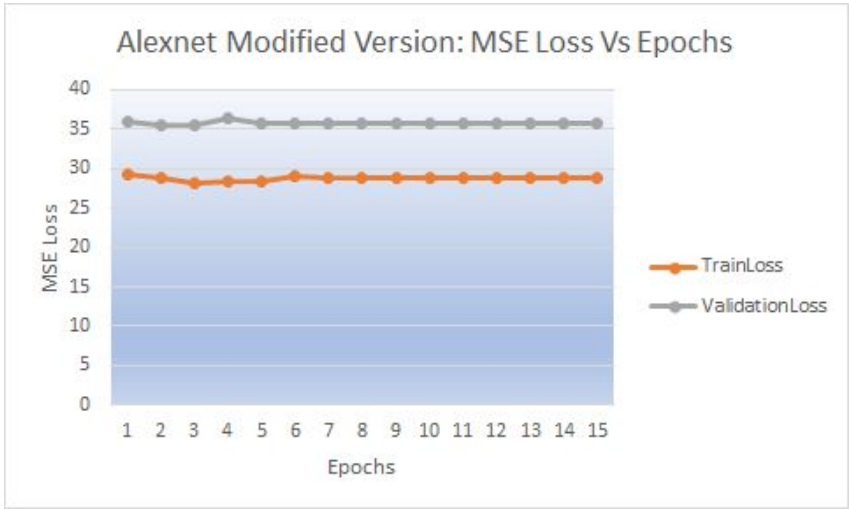
Convolutional Neural Network - Alexnet -Modified Version

1. Original Alexnet architecture
2. Input:
    a. k*k subimage of the original image(k=160)
    b. (k+1) row vector
    c. (k+1) column vector
    d. User embeddings
    e. Movie Embeddings

3. Output: Movie Rating.

Convolutional Neural Network - Alexnet





MSE : 26.73

MSE: 29.23

Conclusion

1. Here are the following models being used for recommending movies for users:
    a. Collaborative Filtering Model
    b. Matrix Factorization
    c. Neural Network
    d. Recurrent Neural Network.
    e. Convolutional Neural Network- AlexNet
    f. Convolutional Neural Network- AlexNet Modified

2. Neural Network and Recurrent Neural Network generates significant less MSE and performed well compared to other models.

References

- http://cs229.stanford.edu/proj2013/Bystrom-MovieRecommendationsFromUserRatings.pdf
- http://cs229.stanford.edu/proj2014/Christopher%20Aberger,%20Recommender.pdf
- https://cs.stanford.edu/people/nipunb/CS345a.pdf
- https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8
- https://towardsdatascience.com/creating-a-movie-recommender-using-convolutional-neural-networks-be93e66464a7
- https://becominghuman.ai/transfer-learning-retraining-inception-v3-for-custom-image-classification-2820f653c557
- https://medium.com/@franky07724_57962/using-keras-pre-trained-models-for-feature-extraction-in-image-clustering-a142c6cdf5b1
- https://www.kaggle.com/sinkie/keras-data-augmentation-with-multiple-inputs
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- https://en.wikipedia.org/wiki/AlexNet