Machine Learning Engineer Nanodegree

Capstone Project

Mahmoud Ahmed Ibrahim

October 26th, 2018

I. Definition

The Video games sector seems to be one of the fast-growing industries in the software market. With the advancement of gaming technology like virtual reality and the huge peaks in GPUs architectures, computers and consoles can now be used as a media center (stores). One of the emerging areas is also in using it for gaming, players may have different preferences in what genres of games they will be more interested in at different times. Furthermore, it can sometimes be difficult to purchase games with not cheap cost and have a big size. In this project, a personalized video game recommendation system helps the user and online games stores to present relevant games for players which takes into consideration user playing time history.

Project Overview

I present a video-games recommender system anticipated to be used by a community of gamers players or game centers. This system filters out or evaluate games through the opinions of other similar gamers using collaborative filtering technique in deep learning and suggest those to the intended user.

The system uses individual ratings given by the members of the community based on playing time, the longtime playing rating of the games that a particular gamer likes, in order to predict and recommend new games to that gamer. The aim is to recommend games that match the user preferences and then user-based collaborative filtering is applied to the individual ratings of the games for a particular gamer to find similarity between those gamers. it should be possible to integrate them into the game stores so that users automatically get personalized recommendations while exploring games.

The prediction is also tested on another algorithm (Matrix Factorization by SVD), in order to verify the prediction accuracy. The results of our collaborative filtering approach are also compared using Root Mean Square Error (RMSE) to check usability and recommendation quality.

The data used in our analysis comes from Steam offered from Kaggle
Steam is the world's most popular PC Gaming hub. With a massive collection that includes everything from AAA blockbusters to small indie titles.

Problem Statement

Video-game players generate huge amounts of data, as everything they do within a game is recorded. In particular, among all the stored actions and behaviors, there is information on a number of hours user played some games. Such information is of critical importance in the gaming world, where gamers tend to spend more time playing games they like most, so we want to recommend similar games to games he spends more time playing them.

Such systems can better achieve their goal by employing machine learning algorithms that are able to predict the rating of an item or product by a particular user like Matrix Factorization (using SVD) or using deep neural networks In this project we build video-games collaborative recommendation system : using deep neural network, which promising candidates for operational video-game recommender system. it should be possible to integrate them into the game stores so that users automatically get personalized recommendations while exploring games. The presented models are able to meet relevant predictions of the games that a particular player will find attractive and related to his taste.

Metrics

The performance of each model is evaluated using **Root-mean-square error (RMSE)** :

The root-mean-square-error is one of the most popular, frequently used, simple measures to find the accuracy of a model. In a general sense, it is the difference between the actual and predicted values. By definition, it is the square root of mean square error, as given by the following equation:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (x_{act} - x_{pred})^{2}}{n}}$$

Here, Xact refers to the observed values, and Xpred refers to the predicted values.

Recommendation systems generally can be evaluated by many metrics like :

- Root mean square error
- Mean absolute error
- Precision and Recall

Here we worked on root mean square error (RMSE) as our comparable model used it to validate its matrix factorization algorithm and we will compare our final RMSE score with that scores.

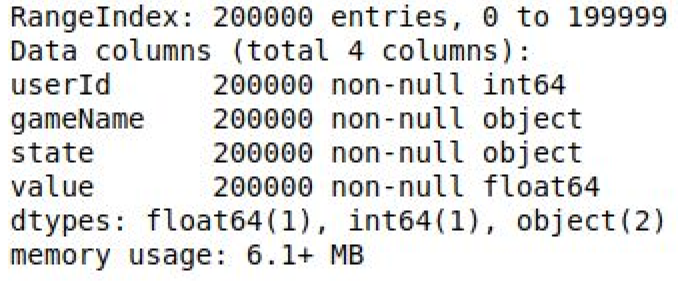
II. Analysis

Data Exploration

The data used in this project comes from Steam offered from Kaggle as.CSV file contains a list of user behaviors,(200K rating) with columns:

1. User-id (int)
2. Game-title (str)
3. Behavior-name (str) : included:-
    a. ‘Purchase'
    b. 'Play'
4. Value (int): The value indicates the degree to which the behavior was performed - in the case of 'purchase' the value is always 1.0, and in the case of 'play' the value represents the number of hours the user has played the game.

Here is dataset info to check data types and null values and total size in memory



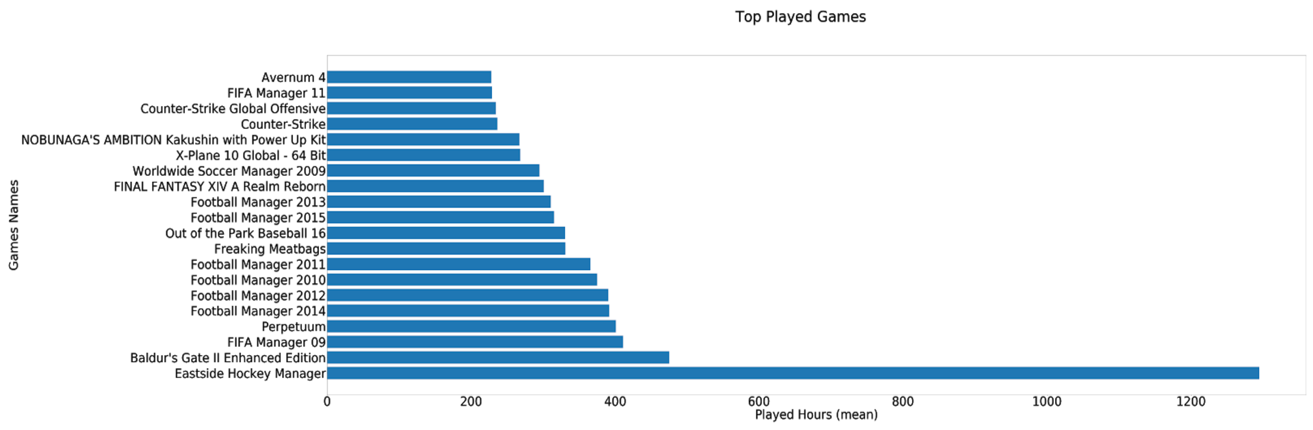
```
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 4 columns):
userId      200000 non-null int64
gameName    200000 non-null object
state       200000 non-null object
value       200000 non-null float64
dtypes: float64(1), int64(1), object(2)
memory usage: 6.1+ MB
```

Fig[1]

In this dataset will work on transforming game playing time to the normalized rating between [1,5] like stars rating system.

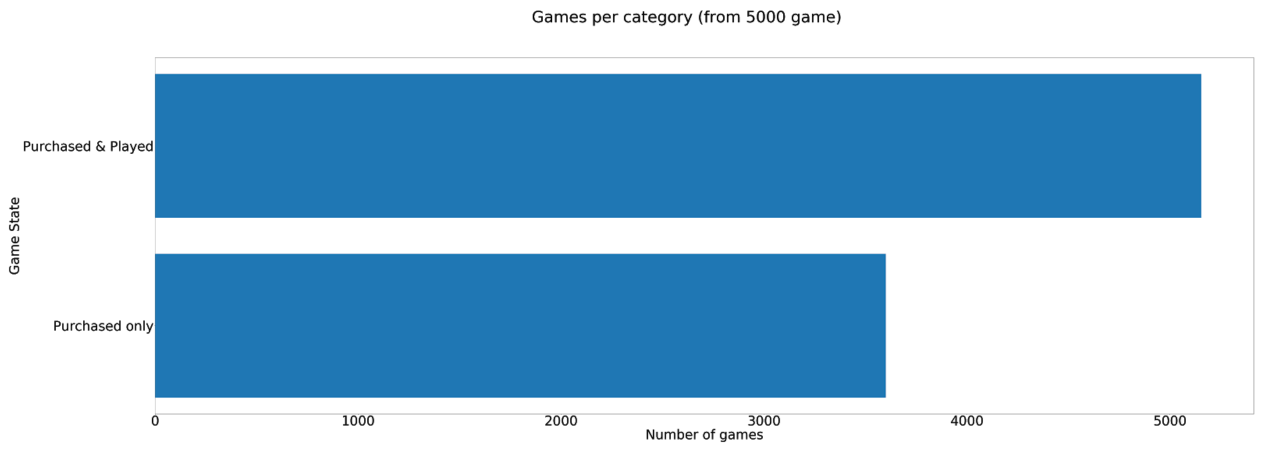
Exploratory Visualization

- Top played games :
  Is one of the problems in this dataset because some games have long playing time as default like open world games which enables playing unlimited time so we need to explore that games. (in the figure below bar chart represents top played time games)



Fig[2]

- Game State :
  More games are purchased but not played that may represent it not liked game comparing with purchased and played for a long time so we need to explore them
  And this graph represents the difference in the number of games based on those categories



Fig[3]

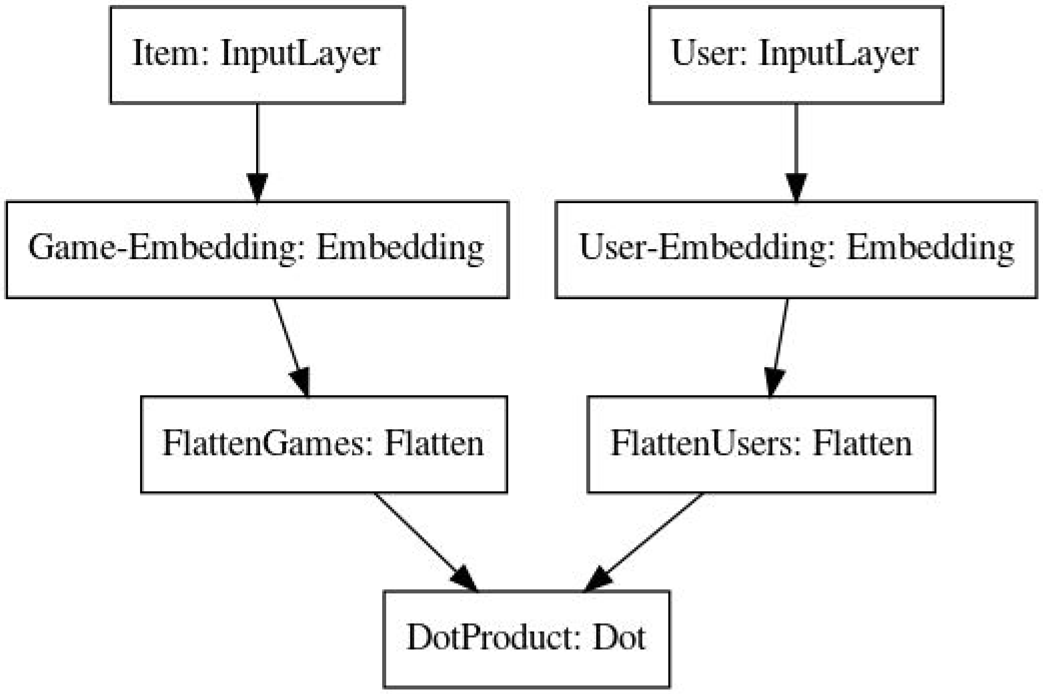
Algorithms and Techniques

we will use a Deep Learning / Neural Network approach that is up and coming with recent development in machine learning and AI technologies.

The idea of using deep learning is similar to that of Model-Based Matrix Factorization. In matrix factorization, we decompose our original sparse matrix into a product of 2 low-rank orthogonal matrices. For deep learning implementation, we don't need them to be orthogonal, we want our model to learn the values of the embedding matrix itself. The user latent features and game latent features based on it will consider nearest neighbors means more latent factors means more training time that is looked up from the embedding matrices for specific game-user combination. These are the input values for further linear and non-linear layers.

Here are the main components of my neural network:

- A left embedding layer that creates a Users by Latent Factors matrix.
- A right embedding layer that creates a Games by Latent Factors matrix.
- When the input to these layers are (i) a **user id** and (ii) a **game id**, they'll return the latent factor vectors for the user and the game, respectively based on **rating**.
- A merge layer that takes the dot product of these two latent vectors to return the predicted rating.



Fig[4]
The figure represents neural network architecture

Benchmark

One of contributors of the dataset used Matrix Factorization approach by:

- Using basic SVD:
- Using SVD via Gradient Descent

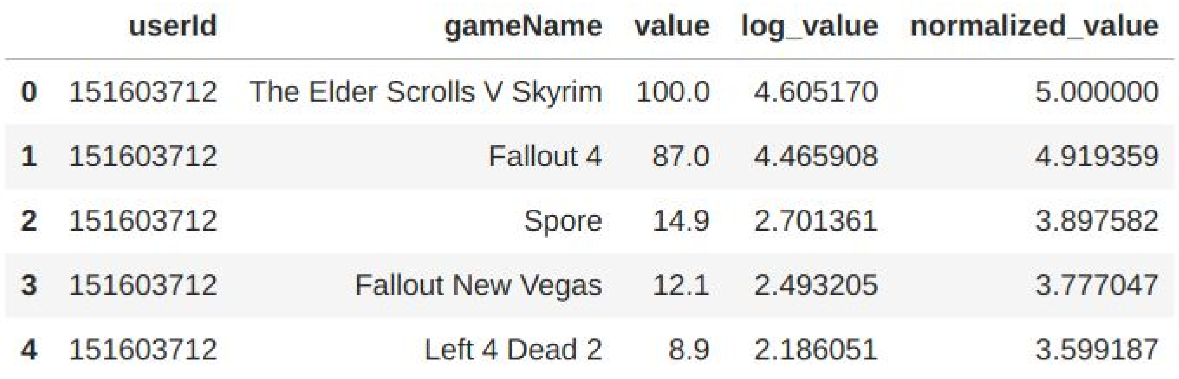
On the same data set and get 2.933 RMSE score for basic SVD, and 1.369 RMSE score for Gradient Descent SVD.

III. Methodology

Data Preprocessing

The preprocessing done in the “data-preprocessing” notebook consists of the following steps:

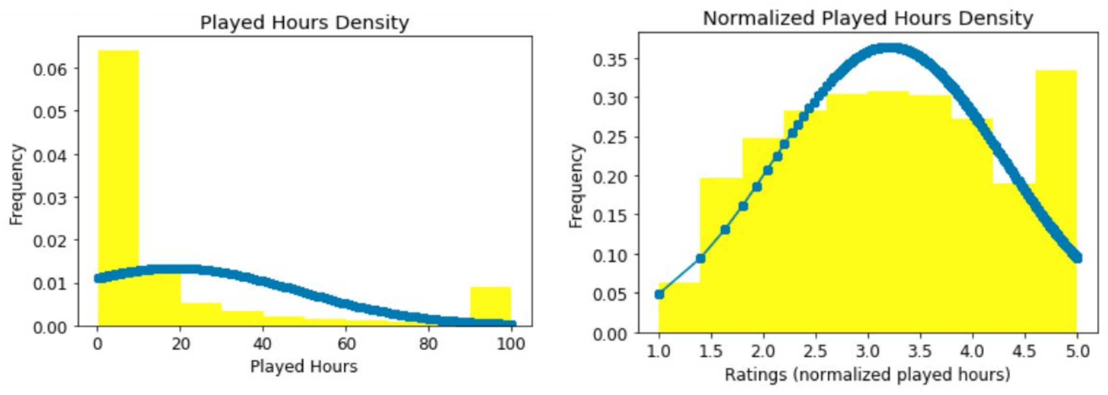
1. Delete unnecessary column because it contains zeros only.
2. Our recommendations are based on the number of playing hours and we found some records have much more played hours than other records (as you can see in Fig[2]) so we take a threshold of 100 if any record has more than 100 played hours we make it 100 played hours.
3. Take the log of the played hours to make it normal distribution function.
4. Normalize played hours values using min-max scaler in range 1 to 5.



	userId	gameName	value	log_value	normalized_value
0	151603712	The Elder Scrolls V Skyrim	100.0	4.605170	5.000000
1	151603712	Fallout 4	87.0	4.465908	4.919359
2	151603712	Spore	14.9	2.701361	3.897582
3	151603712	Fallout New Vegas	12.1	2.493205	3.777047
4	151603712	Left 4 Dead 2	8.9	2.186051	3.599187

Fig[5]

Played hours value, played hours log value and played hours normalized value

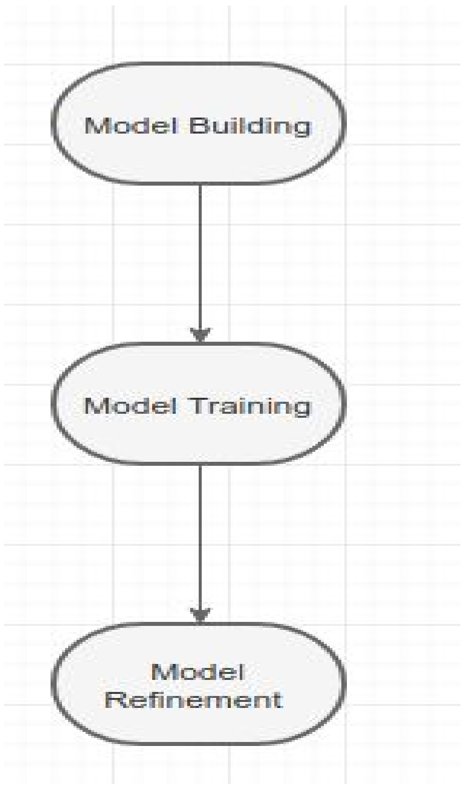


Fig[6]

Normal distribution for played hours values before and after taking log and normalization of the values

5. Split the dataset to train and test datasets (at the beginning of 'model_building' notebook).

Implementation



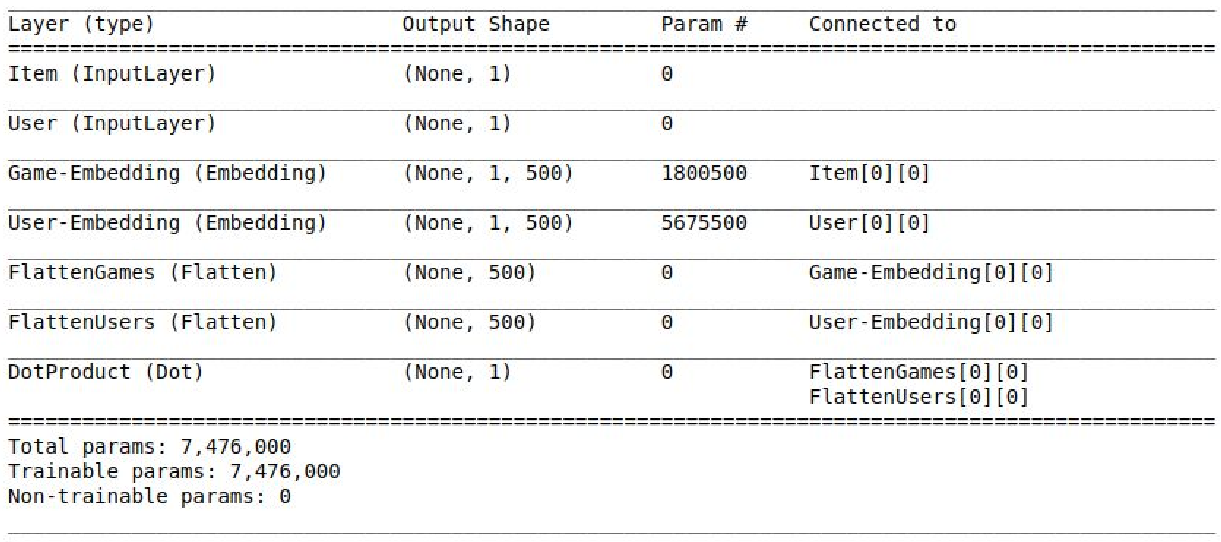
The implementation stage can be described by this flow :

- **Model Building**

  Model building contains some steps :

  1. Setting the number of users and number of games.
  2. Setting the number of latent factors that represents the maximum number of contributed features in similarity score.
  3. Building Network architecture, and here we used buildNN() function and pass to it the above parameters

The final architecture summary is described below with total parameters number



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
Item (InputLayer)                (None, 1)             0
____________________________________________________________________________________________________
User (InputLayer)                (None, 1)             0
____________________________________________________________________________________________________
Game-Embedding (Embedding)       (None, 1, 500)        1800500     Item[0][0]
____________________________________________________________________________________________________
User-Embedding (Embedding)       (None, 1, 500)        5675500     User[0][0]
____________________________________________________________________________________________________
FlattenGames (Flatten)           (None, 500)           0           Game-Embedding[0][0]
____________________________________________________________________________________________________
FlattenUsers (Flatten)           (None, 500)           0           User-Embedding[0][0]
____________________________________________________________________________________________________
DotProduct (Dot)                 (None, 1)             0           FlattenGames[0][0]
                                                                   FlattenUsers[0][0]
====================================================================================================
Total params: 7,476,000
Trainable params: 7,476,000
Non-trainable params: 0
____________________________________________________________________________________________________
```

- **Model Training** :

  It follows some steps :

  1. Define the training parameters like the number of epochs and so on.
  2. Define the loss function and accuracy.
  3. Train the network, logging the validation/training loss and the validation accuracy.
  4. Plot the logged values.
  5. If the accuracy is not high enough, return to step 1.
  6. Save and freeze the trained weights.

- Some of the problems faced me while working on this project are

- selecting the optimal number of latent factors
- Selecting the maximum played hours threshold
- How to makes data in normal distribution by calculating the log of played hours

But on other project parts, the flow was smooth and more interest.

Refinement

We will prepare the data by splitting feature and target columns. and perform data cleaning. To check if the model we created is any good, we will split the data into training and test sets to check the accuracy of the best model. We will split the given training data in two, 80% of which will be used to train our models and 20% we will hold back as a test set and while training we will take 20% of the training set as a validation set.

We performed hyper tuning of parameters of our deep learning network tuned were shown in below table:

Parameter	**Description**	**Value Tested**	**Best Value**
n_latent_factors	the maximum number of contributed features in similarity score	(100, 500)	500
epochs	Number of steps in training a neural network	(10, 30, 50)	30

IV. Results

Model Evaluation and Validation

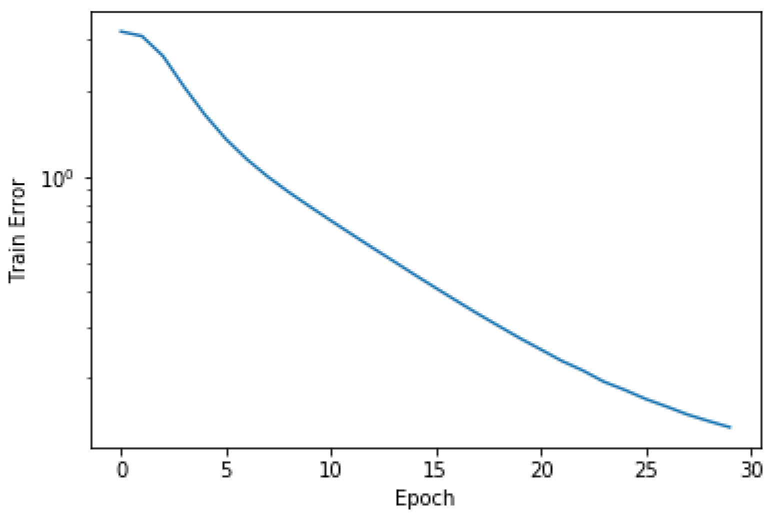
During development, a validation set was used to evaluate the model.

The final architecture and hyperparameters were chosen because they performed the best among the tried combinations.

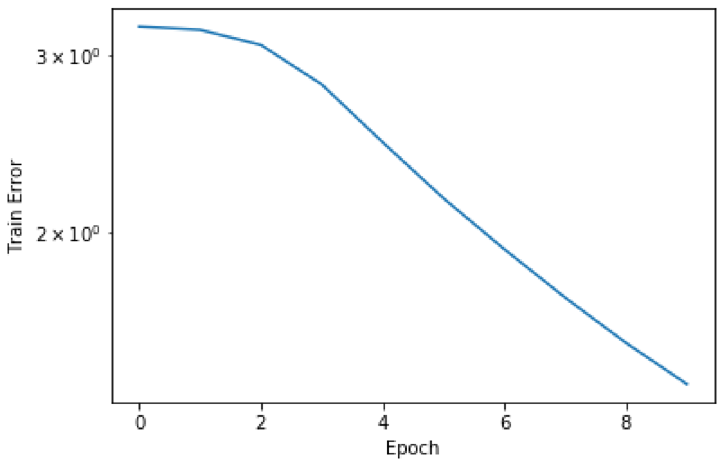
For a complete description of the final model and the training process along with the following list:

- Setting latent factors with **500**
- Setting number of epochs with **30** that's enough to prevent consuming more resources with no observed results.

In the graph below training error rmse score per each epoch and we can observe it saturated after ~27 epochs and get 0.13 rmse training error.



But in the below graph model is trained by 100 latent factor and 10 epochs to get 1.41 RMSE training error.



- **Robustness check :**
    - Normal model parameters:
        - Latent Factors = 100
        - Epochs number = 10
    - Optimized model tuned parameters:
        - Latent Factors = 500
        - Epochs number = 30

	Normal Model	**Optimized Model**	Difference
RMSE score on Training set	1.6911	**1.2839**	0.407
RMSE score on Validation set	1.864	**1.433**	0.431
RMSE score on Testing set	2.12	**1.71**	0.41

Therefore, we can see that even though the tuned parameters have a big effect on the difference between the tuned model and untuned model.

diffrence

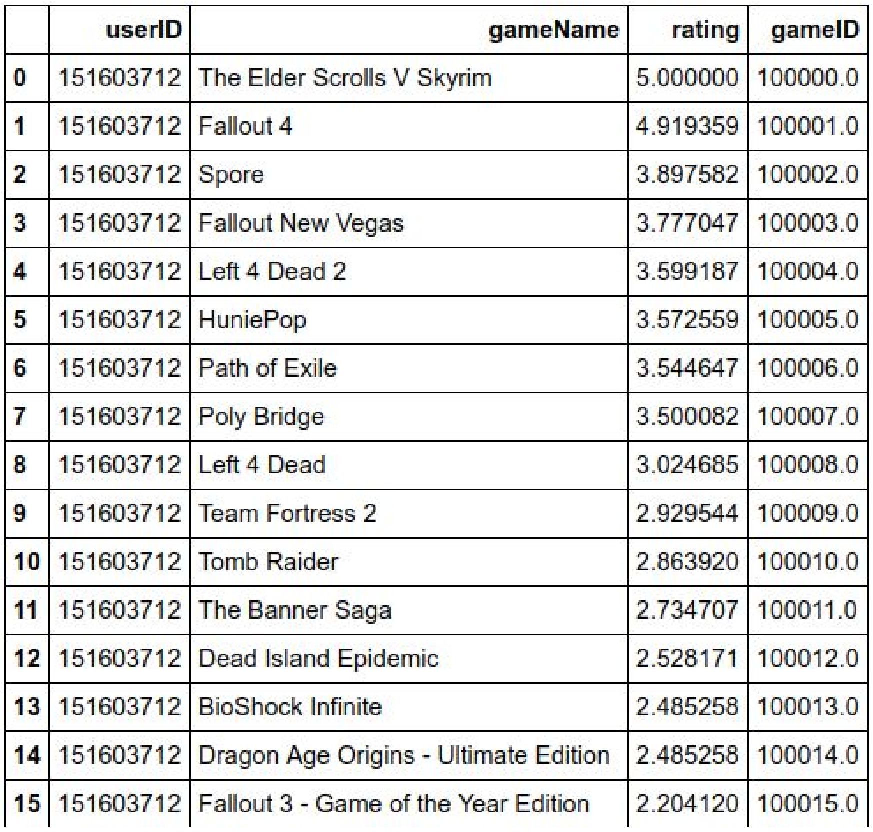
Justification

There is room for improvement on the final results, the tuned final model made no significant improvement over the untuned model. There could be more ways we could improve the score, particularly by selecting a good number of latent factors and number of epochs.

	Deep Learning Model	Basic SVD	SVD with Gradient Descent
RMSE score	**1.71**	2.9	1.36

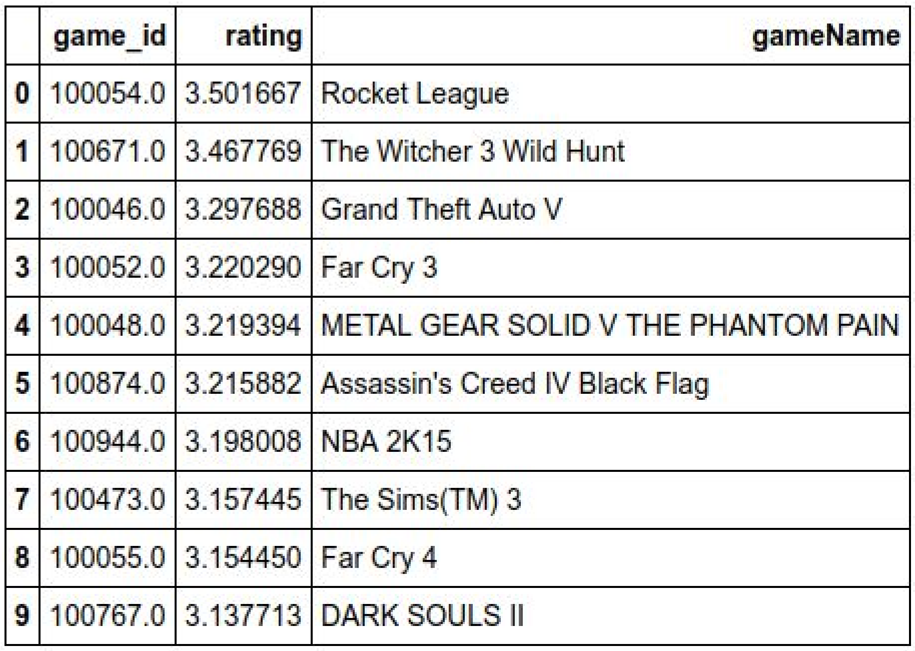
V. Conclusion

Free-Form Visualization



	userID	gameName	rating	gameID
0	151603712	The Elder Scrolls V Skyrim	5.000000	100000.0
1	151603712	Fallout 4	4.919359	100001.0
2	151603712	Spore	3.897582	100002.0
3	151603712	Fallout New Vegas	3.777047	100003.0
4	151603712	Left 4 Dead 2	3.599187	100004.0
5	151603712	HuniePop	3.572559	100005.0
6	151603712	Path of Exile	3.544647	100006.0
7	151603712	Poly Bridge	3.500082	100007.0
8	151603712	Left 4 Dead	3.024685	100008.0
9	151603712	Team Fortress 2	2.929544	100009.0
10	151603712	Tomb Raider	2.863920	100010.0
11	151603712	The Banner Saga	2.734707	100011.0
12	151603712	Dead Island Epidemic	2.528171	100012.0
13	151603712	BioShock Infinite	2.485258	100013.0
14	151603712	Dragon Age Origins - Ultimate Edition	2.485258	100014.0
15	151603712	Fallout 3 - Game of the Year Edition	2.204120	100015.0

Fig[7]



	game_id	rating	gameName
0	100054.0	3.501667	Rocket League
1	100671.0	3.467769	The Witcher 3 Wild Hunt
2	100046.0	3.297688	Grand Theft Auto V
3	100052.0	3.220290	Far Cry 3
4	100048.0	3.219394	METAL GEAR SOLID V THE PHANTOM PAIN
5	100874.0	3.215882	Assassin's Creed IV Black Flag
6	100944.0	3.198008	NBA 2K15
7	100473.0	3.157445	The Sims(TM) 3
8	100055.0	3.154450	Far Cry 4
9	100767.0	3.137713	DARK SOULS II

Fig[8]

- In Fig[7] the games played by some user and we can observe the mind of the user which occurs in open battle, action and open world games.
- In Fig[8] the predicted gamed by this model which appears the same taste of the user but we can observe “NBA 2K15” that not near for the games user played but it may be relevant to users those similar to this user, and for that we need to drop users who haven't unique taste from training data that be a noise for other users.

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public datasets were found
2. The data was downloaded and preprocessed (transform playing time into rating)
3. The neural network architecture is built and model complied with required params.
4. The model was trained using the data (until the error is minimized as possible)
5. The model weights are saved to use it for later recommendations.

I found the hardest part is data pre-processing because of the possible scenarios is a lot but after data pre-processing the project works smoothly.

Improvement

- One of the improvements we need to get more data (not just quantity but more tastes) with clustered tastes because some users that played high difference games cause more noise for other users that appear in prediction performance of the model.
- Another aspect can be explored is game genres that will be a big factor in recommendation filters by determining the user taste based on the type of played games not just play time then recommend games by users who share the same taste.