Movie Recommendation System

Tiberiu Simion Voicu

Post graduate thesis project, ECS751P

Abstract

Recommender systems aim to provide users with personalized recommendations of items. This helps a lot in filtering irrelevant items from the big collection. It is usually achieved through collaborative filtering. Neural networks have proven to be highly effective in natural language processing, computer vision and speech recognition. Despite this, they haven't seen as much use in solving recommendation problems. This is partly because state of the art matrix factorization methods are already very effective for this problem. The system described in this paper will make use of multilayer perceptron networks to solve the recommendation problem using collaborative filtering. The design and implementation details of the recommendation engine will be discussed and the results compared with traditional matrix factorization methods. Furthermore, it will also go into detail about the choices that went into creating a scalable overall system and accompanying web application.

Contents

1. Introduction

Nowadays every customer is faced with a multitude of possible choices. Because of this, it is very important that the user doesn't have to waste time manually filtering through the items to find relevant ones. Recommender systems overcome this by providing personalized recommendations that suit a particular users' tastes. They are present in many different websites such as google, amazon, and netflix and help highly with user retention and satisfaction. I will focus on the problem of recommending movies and also building the underlying system and infrastructure required to serve these recommendations.

1.1. Dataset

The dataset used as the basis for the recommendation engine is the movie lens 20m dataset, put together by the Grouplens research group at the University of Minnesota Harper and Konstan [3]. This is an explicit feedback dataset, where users manually assigned ratings to movies. The characteristics of this dataset are the following.

- 20000263 ratings on a scale of 0.5-5 in 0.5 increments
- 27278 movies having been rated at least once
- 138493 users that have rated at least 20 movies
- 465564 tags about movies

The dataset is split into different files of which I am only using the ratings, movies and links files. They are all available in comma-separated value format. The ratings file comprises of values for userId, movieId, rating, and timestamp.1. Timestamp remains unused while the other values are used together to perform collaborative filtering. The movies file is made up of movieId, title, and genre. 2 It is used for incorporating genre data into the model. The links file contains movieId, imdbId and tmdbId. 3 This file provides imdbId and tmdbId used in web-scraping scripts to provide content for the demo web-app.

userId	movieId	rating	timestamp
57772	3861	4.0	982690041
122521	1411	2.0	1001957487
22759	46578	4.0	1216540196
92812	1206	2.5	1118792437

Table 1: Rating file

movieId	title	genres
106749	Mr. Morgan's Last Love (2013)	Comedy|Drama
2	Jumanji (1995)	Adventure|Children|Fantasy
91169	Easier with Practice (2009)	Drama|Romance
106879	Fright Night 2: New Blood (2013)	Horror

Table 2: Movie file

movieId	imdbId	tmdbId
6	0113277	949
206	0114805	77350
71466	1174730	28053
96834	2343601	111174

Table 3: Links file

2. Background & Related Work

The most common types of recommender systems can be split into collaborative filtering and content based. They can be further split into model-based, and memory-based.

2.1. Content Based

Content based recommendations require several features related to an item instead of historic user-item interactions. In the case of movie recommendations, these could be year, actors, genres, producers, writers, etc. This method relies on calculating the similarity between items using features such as the ones previously stated. The general idea is that if a user likes a given item he will also like items similar to it. One way of achieving this is by calculating term frequency (TF) and inverse document frequency (IDF) of the items. Then using the vector space model and a choice of similarity metrics such as cosine or pearson, we can compare different items Koren et al. [7]. Another way of achieving this is to represent different content items as dense vectors that can be easily fed into a neural network architecture.

2.2. Collaborative Filtering

Collaborative filtering (CF) algorithms aim to recommend items to a user by combining the item interactions of a given user with item interactions of all other users. CF can be split into two categories. User-based where the aim is to measure the similarity of a given user and all other users. Item-based

where we aim to measure the similarity between the items a given user has interacted with and other items. The most widely used method of achieving this is through factorization of the very sparse user-item interaction matrix.

2.3. Matrix Factorization

The idea behind matrix factorization (MF) is to decompose the matrix R containing user-item interactions, into the product of two lower dimensional matrices P of size nXk and Q of size kXm. Matrix P is the user matrix where n is the number of users, k the number of latent factors and p_u is the latent vector of user u. The other matrix is the movie matrix with m number of movies, the same k latent factors and q_i is the latent vector of item i. The predicted rating $\hat{y}_{ui}$ can then be calculated by taking the dot product of those two vectors Koren et al. [7].

$$\hat{y}_{ui} = f(u, i|P_u Q_i) = P_u^T Q_i = \sum_{K=0}^{k} P_{uk} Q_{ik} \tag{1}$$

Singular value decomposition (SVD) and non-negative matrix factorization (NMF) are two techniques successfully applied in literature to achieve the decomposition. SVD is the general case for principal component analysis
The resulting matrices are dense and have much lower dimension than the initial matrix R. By choosing a different number of latent factors we can include more or less abstract information about the initial matrix R. MF poses the recommendation problem as a regression optimization one. Two common metrics used for this are root mean squared error (RMSE) and

mean absolute error (MAE). The RMSE and MAE can be calculated as follows given that, e_i is the difference between the actual and predicted value of rating i.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2} \tag{2}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid e_i \mid \tag{3}$$

Stochastic gradient descent (SGD) and alternating least square (ALS) are two optimization algorithm generally employed to learn a good approximation.

- Set item factor matrix constant and adjust user factor matrix by taking the cost function derivative.
- Set user factor matrix constant and adjust item factor matrix.
- Repeat until convergence.

2.4. Neural Networks & Deep Learning

Neural networks are universal approximators, typically organized in layers of neurons connected through weights and put through an activation function. The number of layers and neurons at each layer also called the depth and width of the network are variable and many different configurations seem to work in practice. The simplest neural network is made up of 3 layers, input, hidden and output.

They can be used for both supervised and unsupervised learning and can be applied to classification and regression problems just as effectively. Deep

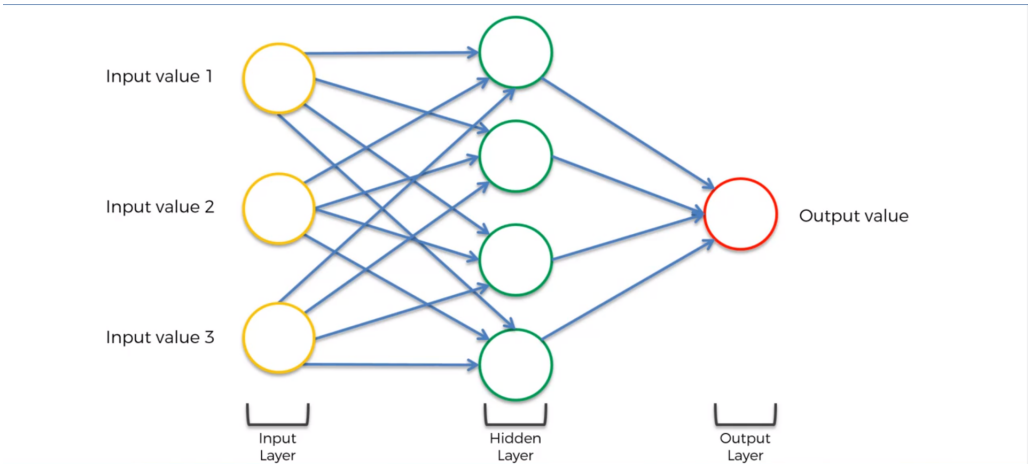


Figure 1: 3 Layer Neural Network

neural networks (DNN) are NN of more than 3 layers, although in practice and state of the art implementations these are many layers deep and very wide as well. The main benefit of using NNs algorithms is that they don't require manual feature engineering. A big disadvantage is the lack of interpretability for the predictions, due to this NN are generally viewed as black boxes. This means that we are not aware of the 'why' and 'how' did the network product a certain output given some inputs. This could be especially detrimental in the context of recommendation systems because the users might wish to know the reasons behind their recommendations. Other memory and similarity based methods provide much more transparency in this regard.

Different classes of NN have emerged in literature including convolutional networks (CNN), autoencoders (AE), recurrent networks (RNN), generative adversarial networks (GAN) besides of simple fully connected multilayer perceptrons (MLP).

2.5. Related Work

Strub et al. [8] propose a hybrid recommender system based on auto-encoders He et al. [5] propose a neural collaborative filtering system using implicit training data for training.

3. System Architecture

3.1. Technologies, tools and frameworks

3.1.1. React

React is a fast, declarative and efficient javascript library for creating web interfaces. It works around the concept of components. They are self-contained and composable blocks of code that encapsulates a part of user interface and its functionality. By putting together multiple small components it's possible to build complex user interfaces (UI). Components can be stateful or stateless. The library provides a virtual-dom similar to the browsers document object model (DOM). They are both node trees that list elements together with attributes and content as objects and properties. Updating the dom is rather slow which is why the virtual-dom is useful for efficiency. It allows react to optimize DOM updates under the hood to only happen when it's necessary.

3.1.2. Docker & Containers

Containers are self-contained pieces of code that can be run on any computer and operating system (OS). They contain the code and all of its necessary parts such as libraries, tools ,and frameworks. They are similar to virtual machines but the main difference is in efficiency and application size. Containers are more efficient and smaller because they run on the same underlying kernel as the operating system as opposed to virtual machines which runs an entirely different OS.

Docker is tool that allows creating, running and managing containers. Docker

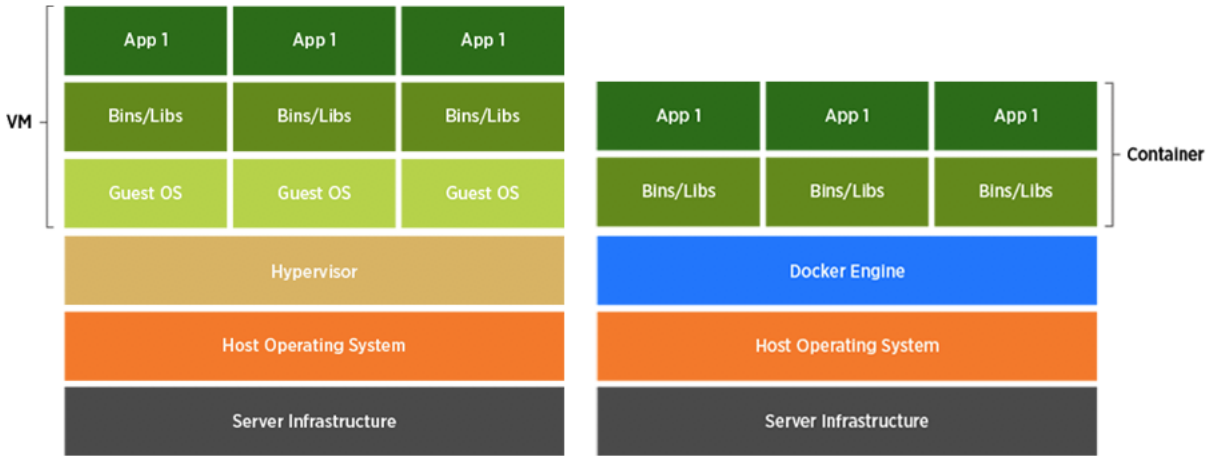


Figure 2: Containers and virtual-machine comparison diagram

3.1.3. Kubernetes

Kubernetes is a container orchestration platform created by Google and open-sourced in 2014. It allows automation of deployment, scaling, and management of containerized applications. It groups the containers that make up a multi micro-service application into logical units for easy discovery and management. It's built with scalability in mind

Node Pod Service Gateway Ingress

3.1.4. Mongodb & Mongoose

3.2. Micro-services

Authentication

Gateway

User

Search

Engine

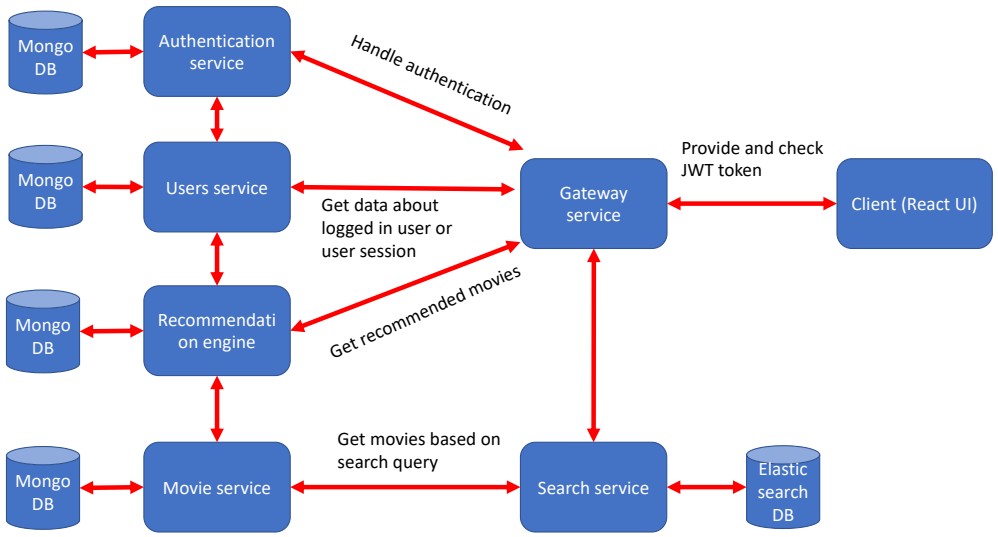


Figure 3: Architecture diagram

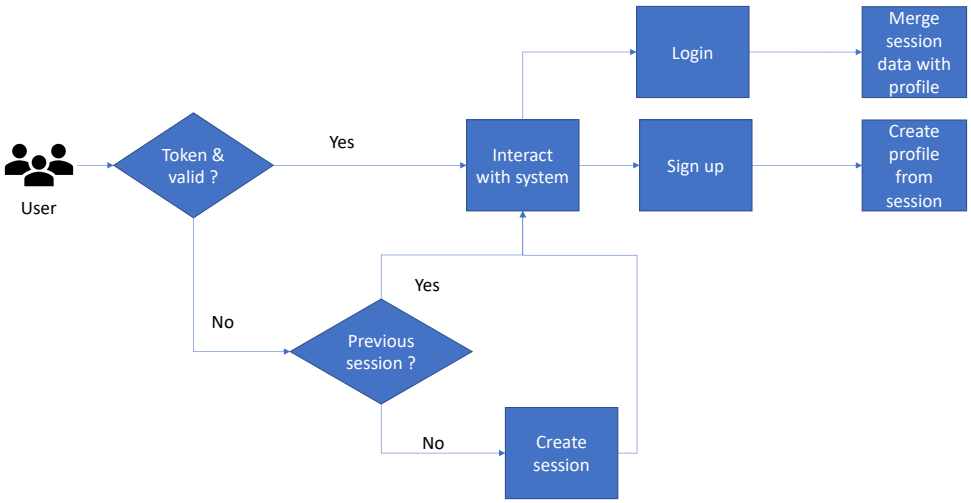


Figure 4: User flow diagram

3.3. Web application

4. Recommendation Engine

4.1. Embeddings

Neural networks do not deal well with categorical variables. One hot encoding is a way to handle this. It allows us to represent categorical data as sparse vectors of zeroes and a single 1 representing the specific category. This method has two main drawbacks. Firstly, the dimensionality of the vector representation size grows with the corpus. It can become unmanageable very quick. Secondly, each vector is equidistant from every other vector. This means 'similar' categories are not represented close together in the vector space.

A better way of handling categorical data is through embeddings. When using embeddings we can project categories in a low dimensional latent space and represent them as dense continuous vectors. They are learned parameters and due to this, similar items are projected close together in the latent space. Therefore it is useful in the context of recommendations as we are trying to model user-item similarities. They are also well studied and highly applied in literature for natural language processing to represent words. Embeddings can be pre-trained and adapted to be used in a model or learned end-to-end with the other parameters.

The main inputs to the model consist of user, item and genre embeddings learned iteratively with the rest of the model parameters.

Genres are handled slightly different than the other two because a movie can have more than one genre. The basis of the embedding is multi hot encoding, meaning the vector has a value of one for each category that describes it.

4.2. Activation function

In a neural network activation functions are mathematical equations, applied to each neuron, that determine if it should activate or not based on its inputs. This function must be computationally efficient to calculate, differentiable and will generally be non-linear. The last part is very important because without non-linearities a NN would just behave like a single-layer perceptron and would not be able to model complex functions. One exception to this is the output layer for a regression NN which will have a linear activation to

allow the prediction of any real value.

Early neural networks were using tanh and sigmoid activation functions. Sigmoid also known as logistic function is S-shaped and bounded by 1 and 0. 4 Traditional sigmoid 4, tanh 5

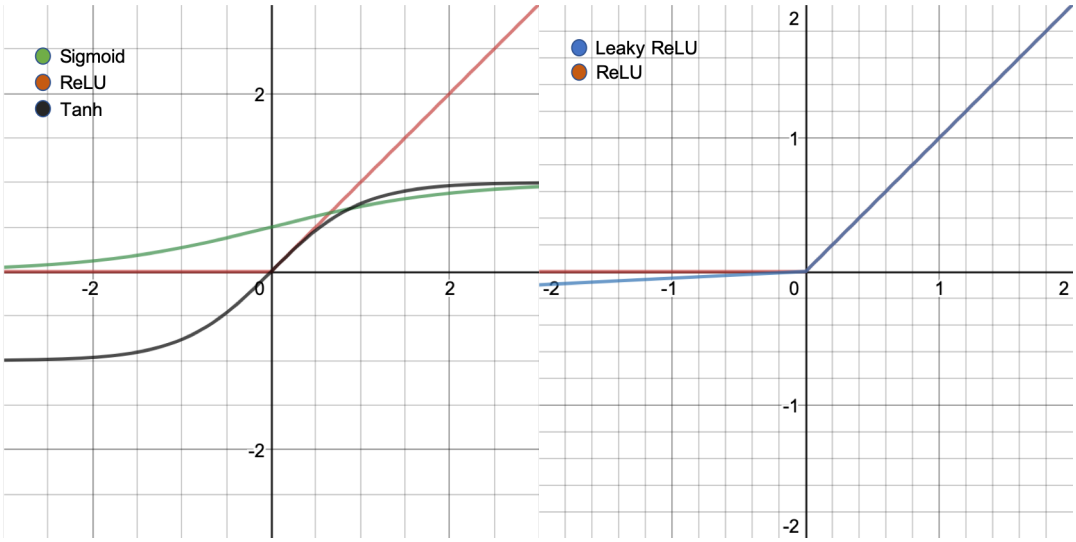
Rectified Linear Unit ReLU 6 A major drawback of using ReLU activations is the "dying ReLU" problem. leaky ReLU 7 comparison with traditional ones comparison between relu and leaky relu

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= f(x)(1-f(x)) \end{aligned} \tag{4}$$

$$\begin{aligned} f(x) &= \frac{2}{1+e^{-2x}} - 1 \\ f'(x) &= 1 - f(x)^2 \end{aligned} \tag{5}$$

$$\begin{aligned} f(x) &= \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \\ f'(x) &= \begin{cases} 1, & \text{if } x \geq 1 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{6}$$

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{otherwise} \end{cases}$$
$$f'(x) = \begin{cases} 1, & \text{if } x \geq 1 \\ 0.01, & \text{otherwise} \end{cases} \tag{7}$$



(a) Activation functions (b) ReLU vs Leaky ReLU

4.3. Regularization

One of the most common problems in training neural networks is over-fitting. A model is said to be over-fitting when it performs well on training data but poorly on validation data. Intuitively, it means the model is unable to generalize well and only memorizes the training examples. This generally occurs if the model is too complex or the size of the training dataset is too small. Reg-

ularization is a method of reducing over-fitting through small modifications of the learning algorithm. The most common types of regularization are l1 and l2. L1 Kernel Regularization Activity Regularization Dropout general value 0.5 gives higher RMSE than 0.2 Batch normalization

4.4. Optimizer

Optimizers are a very important parameter in NN configuration. Nowadays there is a vast choice of good optimizers. At its core, an optimizer is an iterative method of optimizing for a cost function. At each optimization step, the weights in the NN will be updated based on the negative of the gradient of the cost function. Stochastic gradient descent (SGD) is a variation of gradient descent in that the optimization happens after each training example. In practice, this usually implies mini-batches of between 32 and 1024 data points (Bengio [1]). Batch gradient descent involves updating the weights based on the gradient over the whole dataset. SGD optimizes based on an approximation of the gradient, unlike batch gradient descent. This turns out to be useful as it introduces noise in the network which leads to better generalization. It is also more scalable as the whole dataset does not have to be kept in memory. (Bengio [1])

$$\theta = \theta - \alpha \Delta_\theta J(\theta; x^{(i)} y^{(i)}) \tag{8}$$

More advanced optimizers are built on top of SGD and include things such as adaptive learning rates and momentum to increase convergence speed and

overall stability.

One such algorithm is adaptive moment estimation (Adam). Its widely used in literature and converges much faster than SGD. Adam can be seen as a combination of RMSProp and momentum (Kingma and Ba [6]).

Nesterov adaptive moment estimation (NAdam) combines adam with nesterov momentum which improve convergence (cite)

4.5. Training

Weight initialization plays an important role in training a neural network. Ideally, we wish the initial weights to be random but not too small and too big, otherwise, it will lead to problems of vanishing or exploding gradients. Xavier normal initialization is one technique that constricts the weights to these characteristics. It works by drawing from a truncated random distribution centered on 0 and with a standard deviation of $\frac{2}{n_{in}+n_{out}}$, where n_{in} and n_out represent the number of inputs and outputs (Glorot and Bengio [2]). This is the default initializer used in keras. It is best suited for use in NN that employ tanh activation functions. He normal initialization works better for relu activation. Its similar to xavier, but the standard deviation is $\frac{2}{n_{in}}$ (He et al. [4]).

```
xavier_w = np.random.rand((n_in, n_out)) * np.sqrt(2 / (n_in + n_out))
he_w = np.random.rand((n_in, n_out)) * np.sqrt(2 / n_in)
```

Too speed up the training I have implemented a special generator class extending keras *sequence*. This enables multiprocessing execution of the batch

generation algorithm and more importantly it ensures safety and single use of each training example per epoch. After creating the generator its possible to use the keras *fit_generator* function with 12 workers and a maximum queue size of 200. These parameters are required in order to enable multi-gpu training and to ensure that the gpus are not idle waiting for data.

4.6. Architecture

The NN architecture follows a common tower pattern, where layers near the top are widest and progressively decrease in width. The activation functions employed at the hidden layers are leakyRelu. Each hidden layer is followed by a dropout layer and a batch normalization layer.

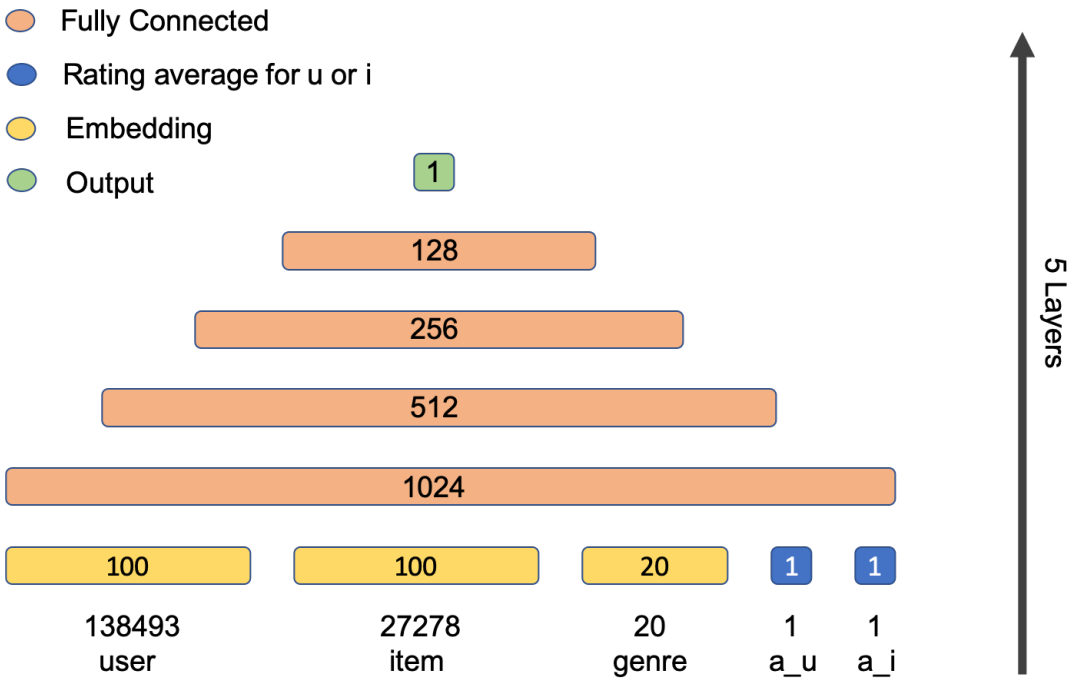


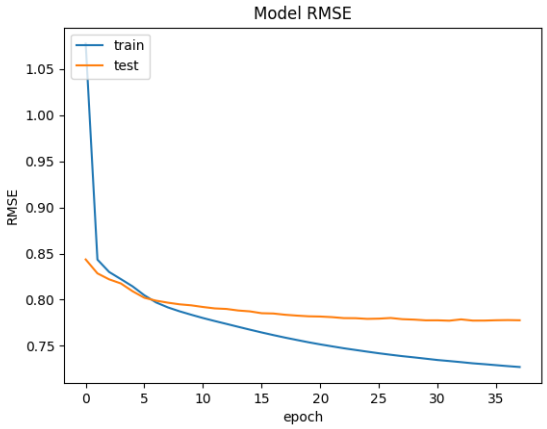
Figure 5: Network architecture

4.7. Serving recommendation & Api

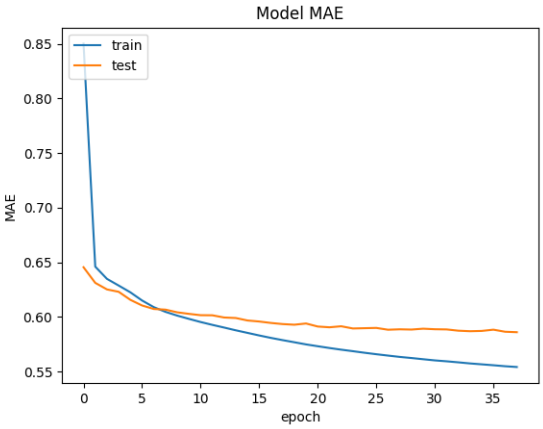
Structure of Api. Create pod that just retrains the model on new ratings. Once finished redeploy pods

5. Evaluation

The evaluation section with results



(a) RMSE plot



(b) MAE plot

Model	Test RMSE	Test MAE
SVD	0.8152	0.6190
NMF	0.8246	0.62876
NNCF	0.7763	0.5861
NNCF+genres	0.7763	0.5861
NNCF+genres+avg	0.7763	0.5861

Table 4: Results Comparison (RMSE & MAE)

6. Conclusion

6.1. Future Work

Possible improvements to the system include the following.

- Incorporate more content information into the system such as movie title, summary, and cast. This could be achieved using pre-trained word embeddings such as word2Vec or glove.
- Add recurrent layers such as GRU or LSTM.
- Train a convolutional network on movie posters. Then incorporate said network into the CF model.

References

[1] Bengio, Y. [2012], 'Practical recommendations for gradient-based training of deep architectures', *CoRR* **abs/1206.5533**.
URL: *http://arxiv.org/abs/1206.5533*

[2] Glorot, X. and Bengio, Y. [2010], Understanding the difficulty of training deep feedforward neural networks, *in* Y. W. Teh and M. Titterington, eds, 'Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics', Vol. 9 of *Proceedings of Machine Learning Research*, PMLR, Chia Laguna Resort, Sardinia, Italy, pp. 249–256.
URL: *http://proceedings.mlr.press/v9/glorot10a.html*

[3] Harper, F. M. and Konstan, J. A. [2015], 'The movielens datasets: History and context', *ACM Trans. Interact. Intell. Syst.* **5**(4), 19:1–19:19.
URL: *http://doi.acm.org/10.1145/2827872*

[4] He, K., Zhang, X., Ren, S. and Sun, J. [2015], 'Delving deep into rectifiers: Surpassing human-level performance on imagenet classification', *CoRR* **abs/1502.01852**.
URL: *http://arxiv.org/abs/1502.01852*

[5] He, X., Liao, L., Zhang, H., Nie, L., Hu, X. and Chua, T.-S. [2017], 'Neural collaborative filtering', *ArXiv* **abs/1708.05031**.

[6] Kingma, D. P. and Ba, J. [2014], 'Adam: A method for stochastic optimization'. cite arxiv:1412.6980Comment: Published as a conference

paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

URL: *http://arxiv.org/abs/1412.6980*

[7] Koren, Y., Bell, R. and Volinsky, C. [2009], 'Matrix factorization techniques for recommender systems', *Computer* **42**(8), 30–37.

[8] Strub, F., Mary, J. and Gaudel, R. [2016], 'Hybrid recommender system based on autoencoders', *CoRR* **abs/1606.07659**.

URL: *http://arxiv.org/abs/1606.07659*

```
class Generator(Sequence):
    def __init__(self, data, user_avg_ratings, movie_avg_ratings, b
        self.data = data
        self.user_avg_ratings = user_avg_ratings
        self.movie_avg_ratings = movie_avg_ratings
        self.batch_size = batch_size

    def __len__(self):
        return int(np.floor(len(self.data) / float(self.batch_size)

    def __getitem__(self, idx):
        batch = self.data.take(np.arange(idx * self.batch_size, (idx
        userIds = batch.loc[:]['userEmbeddingId'].values
        movieIds = batch.loc[:]['movieEmbeddingId'].values
        genreEmbeddings = np.array(list(map(lambda x: np.array(x),
        ratings = batch.loc[:]['rating'].values

        userAvgRatings = np.array([self.user_avg_ratings.get(i) for
        movieAvgRatings = np.array([self.movie_avg_ratings.get(i) fo

        # return [[userIds, movieIds, genreEmbeddings], ratings]
        return [[userIds, userAvgRatings, movieIds, movieAvgRatings
```