**Predicting Movie Recommendations by Leveraging Deep Learning and MovieLens Data**

by Annie Phan, Drew Solomon, Yuyang Li, and Roma Coffin

**Project Overview:**

Recommendation systems use consumer data to develop personalized preferences to customers. Tech companies have honed in on this strategy as it has proven to be successful in enhancing user-experience, popular examples include restaurant suggestions on Grubhub, playlist suggestions on Spotify, product suggestions on Amazon, or movie suggestions on Netflix. Recommendation systems typically use clustering, nearest neighbor, or matrix factorization techniques. Deep learning models have recently increased in popularity though to overcome limitations of these methods and increase prediction accuracy.

We are accessing the MovieLens dataset which consists of 100k ratings on 3,900 movies from 6,040 MovieLens users and leveraging deep learning. Our goals include finding new applications and to build better movie recommendation systems that more accurately provide personalized content for the modern consumers.

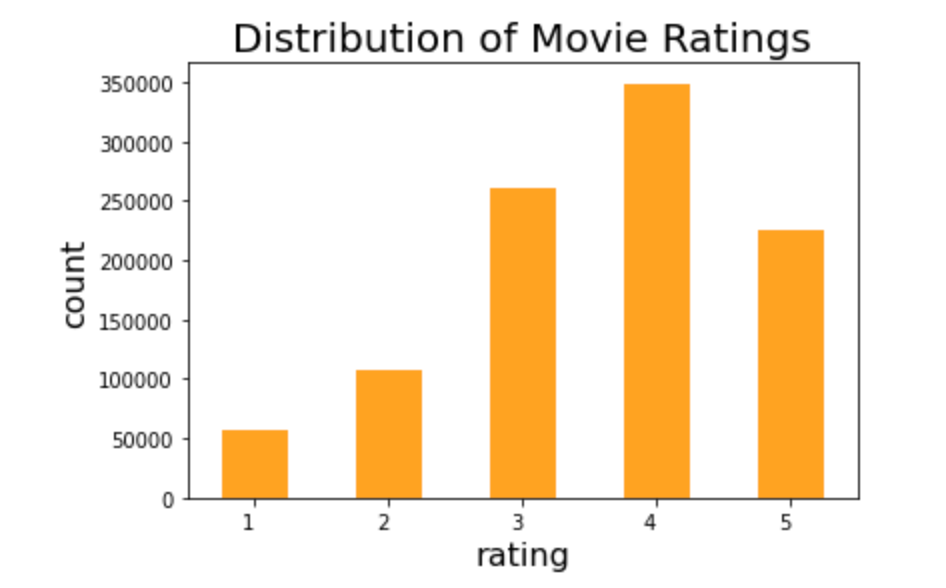
While our main objective is to predict movie recommendations using MovieLens data, our aim is to replicate a previously created model and improve upon it and also find new applications. This first blog post will shed light on the data collection processes, exploratory data analysis, model methodology, and next steps for this project. Please do take note that although changes were made, information was used from the original research project site to assist with our analysis.

Our study is based off of James Le’s research, “[The 4 Recommendation Engines That Can Predict Your Movie Tastes](https://le-james94.medium.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223)”. [James Le](https://jameskle.com/) is a graduate researcher who is focused on deep meta-learning techniques that qualify recommendation systems to attain high performance accuracy, applicable when data is limited, and macro-scale appropriately. We were interested in Jame’s Le research over other research due to many reasons. His approach is clear, transparent, applicable, and achieves quality results. Additionally, his knowledge with recommendation systems is well accredited so we believe his analysis is “state of the art” in terms of what is publicly available.

Our interest in exploring recommendation systems using MovieLens data stems from our desire to learn more about recommendation systems. Recommendation systems are so ubiquitous, we many times do not even notice them. We wanted to know more about the effectiveness of these algorithms, as they not only enhance content and services for a customer but also attempts to draw the most value out of a customer as well. Incorporating deep learning models into recommendation systems has become quite popular as of late, as it can defeat limitations of other algorithms. Although we have never directly learned about recommendation systems yet(as is the last topic to be covered in our class), we wanted to challenge ourselves and utilize the deep learning models we have been taught, and apply those to improve our results.

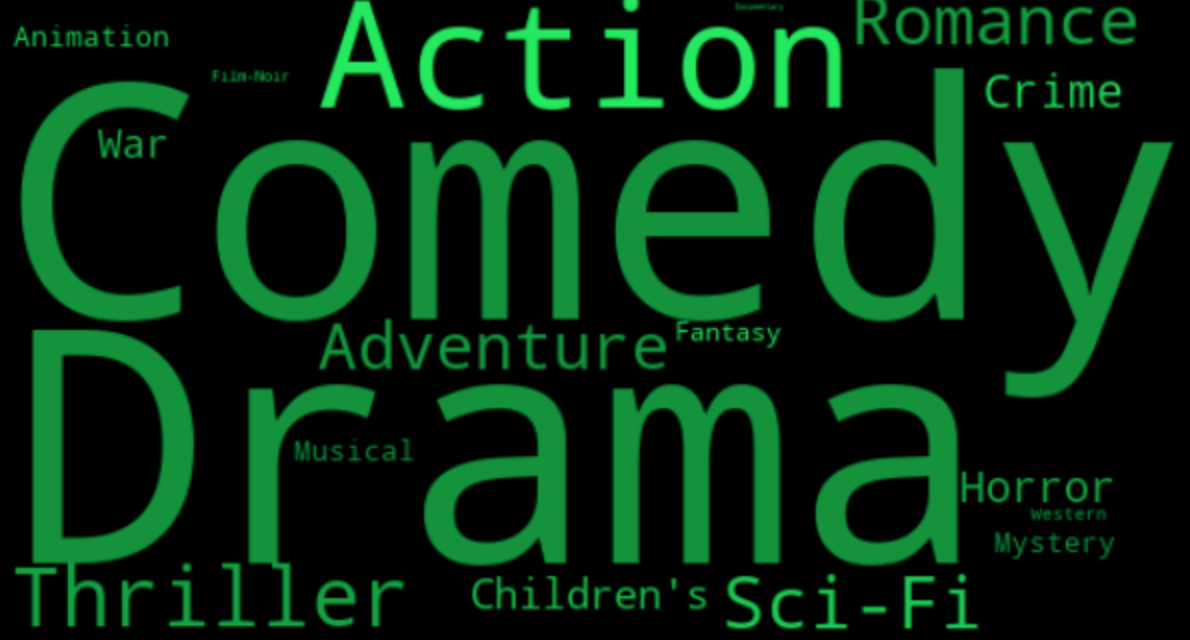
**Exploratory Data Analysis:**

For us to better understand the features in the MovieLens dataset, exploratory data analysis (EDA) was performed. EDA helped to familiarize us with the three datasets used (Movies, Ratings, and Users). It also helped us with the three datasets when they were merged, by capturing common data patterns and to help us with data visualization. Additionally, our EDA process will help us discover if the deep learning approaches being applied are applicable.

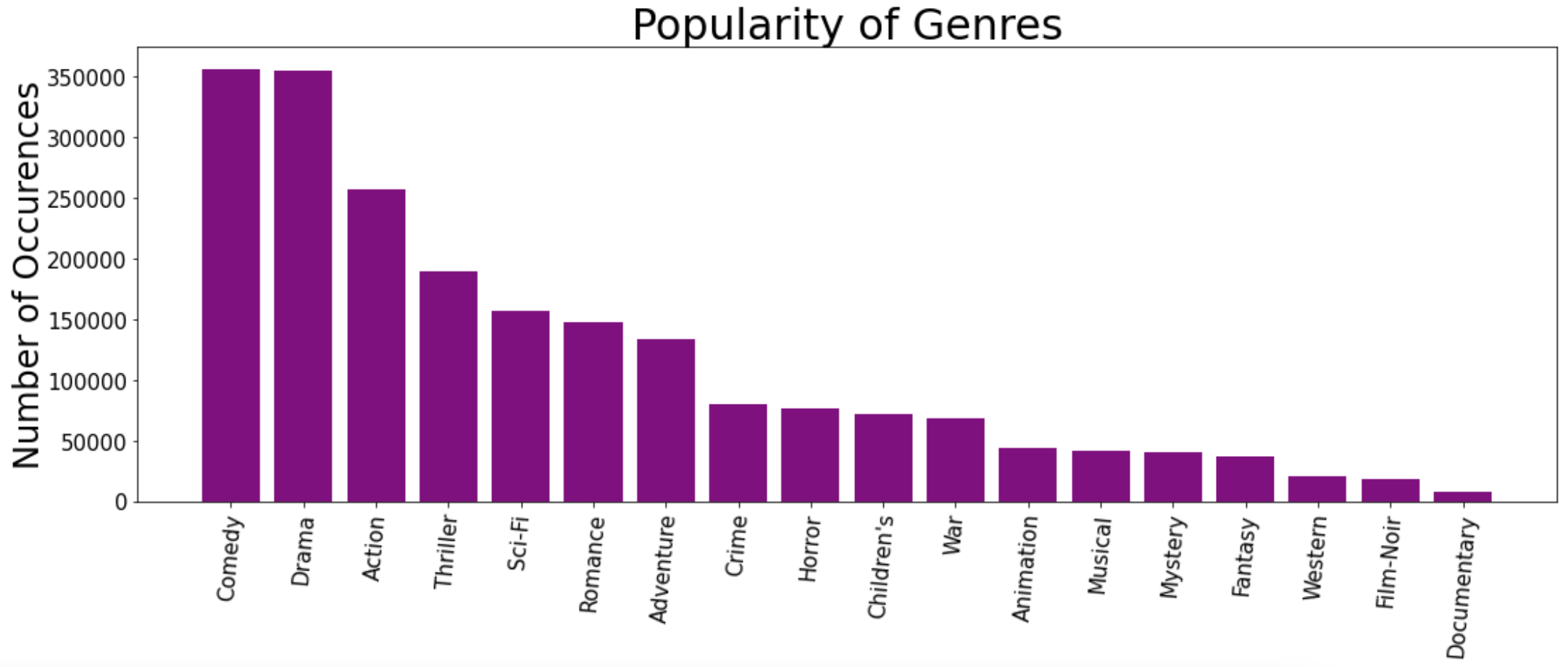
We started by checking to see if any of the three datasets had missing or confusing information, as we would desire to remove missing values or fix any faulty or corrupted aspects of the data. Luckily, no unusable items were found, but we did notice that the genres feature had 301 different types which we will discuss in more detail later on in this blog post. Next, we wanted to get a sense of the distribution of ratings in the dataset. This bar chart reveals the dataset is skewed to the left, slightly imbalanced, and the rating 4 was the most dominant. This is important to know so in the future we can potentially stratify (e.g. using stratified K fold, may try this later for improving the performance) to ensure all of the ratings are adequately represented across training, validation, and testing datasets.

*Bar chart of the count for each rating*

After checking the ratings, we decided to build a Wordcloud visualization and bar chart to see the number of occurrences for each genre. Genre output was edited because there were 301 different categories. Since, some movies had multiple genres (ex. romantic comedy would count as romance and comedy) they were counted as each genre separately. We can see that comedy and drama are the most present genres.This gives us a better idea of what potential bias we might have in the training set. Therefore, we can try to eliminate the bias during our model constructing stage.



*Word Cloud visualization of the most popular genre*



*Bar chart of the number of occurrences for each genre*

**Baseline Model Methodology & Results:**

Our baseline model is based off of James Le’s deep learning model. When reviewing his preprocessing methods we see that his overall approach was limited as he uses individualized approaches in his 4 implemented models. Because we will be focusing on his deep learning approach, we see that he creates a training and validation set by shuffling randomly the values from the original ratings dataset. This is a pretty simple method and it might behoove us to look into more complicated preprocessing methods that can assist with imbalances in the data or maximize on the dataset we have. In terms of EDA, James EDA is quite robust but he could have looked at a few more distributions to visualize more features’ data patterns

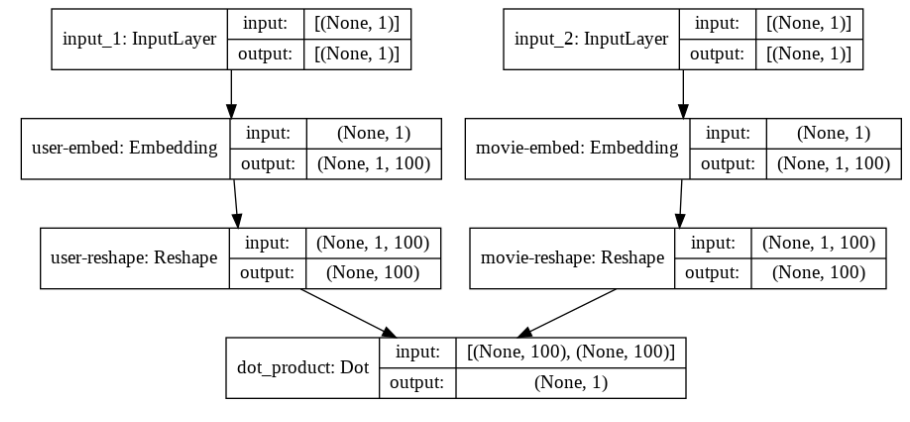
James analysis consisted of 3 different approaches besides deep learning:

* Content-based Filtering depends on alike content or items that are being recommended, so if you are interested in something then you will also be interested in something similar. The more a user takes actions on the recommendations, the more accurate it will be.
* Collaborative Filtering relies mostly on previous actions from a user and is not focused as much on the context, it looks at how alike one user is to other users and then makes recommendations.
* Matrix Factorization recommends content or items to users depending on their search history or previous actions on its past search/activity.

Unfortunately, although each approach has been thoroughly outlined and researched by James, due to time constraints and our focus on deep learning, we will only be improving upon his deep learning approach. By focusing on his deep learning approach, we hope that we are able to better understand this approach and make significant improvements on it.

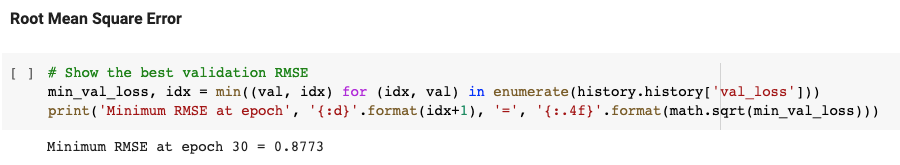
James’ deep learning model is our baseline model. The deep learning model is similar to the idea of matrix factorization, except that to make recommendations, the values from embedding the matrix are captured and learned. By embedding the matrix, a categorical factor is translated to continuous-valued. Efficiency is increased in terms of expressing larger dimensional vectors to a smaller dimensional space.

The image below depicts the baseline structure of the model which uses a sparse matrix factoring algorithm. The model consists of a left side which takes a user id into the input layer and from there the embedding layer develops users through a latent factors matrix, it will output the latent factor vectors for the user. The right side is very similar except it uses a movie id for the input layer and the embedding layer develops movies which will result in latent factor vectors for the movie. Finally, the merge layer outputs a rating based on the dot product of the latent factor vectors users and movies.



*Baseline deep learning model structure*

James chose RMSE as an evaluation metric as it is one of the more favored metrics used to assess accuracy for predicting ratings. We mimic that approach when implementing the baseline model. The model is compiled using MSE as the loss function(take the root at end for RMSE) and the callbacks monitor validation loss so every time improvement occurs, model weights are remembered. We achieved a minimum RMSE at epoch 30 of 0.8773, so the leading validation loss was 0.7627. It is of note that in terms of run time, he used CPU and it took 3 hours, our model used GPU and took about 45 minutes for 30 epochs. We should keep this in mind in the future, as our process seems to be running faster than his.



The minimum RMSE is *0.8773* at epoch 30

In addition to the RMSE calculation, we also include the recommendation from the baseline model that we based our model off of. Please see below for a list of 20 recommendations of unrated movies sorted by prediction value when the user\_id =2000

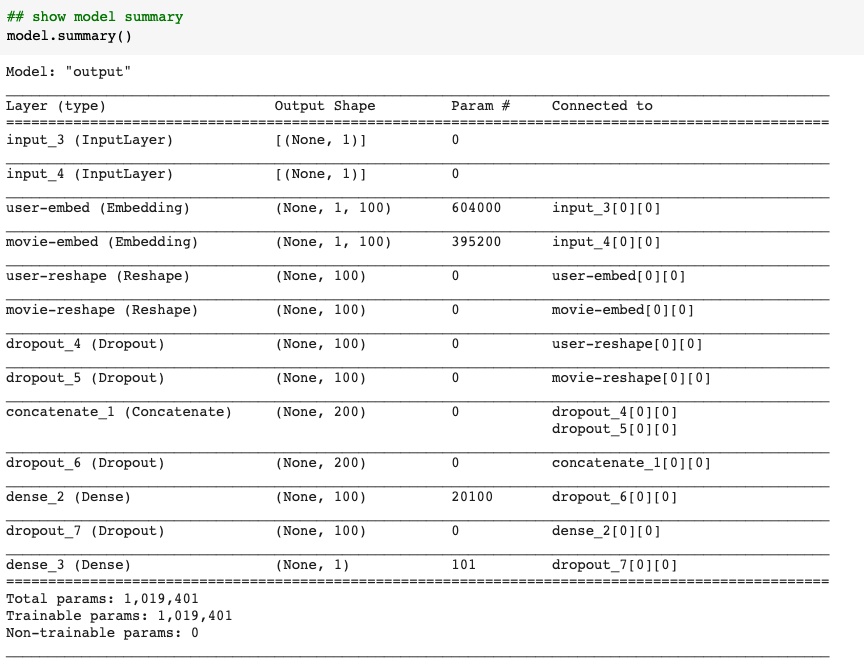
*Movie Ratings Prediction of User with ID = 2000*

*Recommended Movies for User with ID = 2000 based on Predicted Ratings*

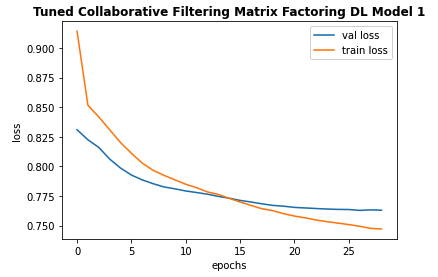
**Updates to Baseline Model:**

We now aim to improve upon our baseline model through using fine-tuning and transformers. We implement new deep learning architectures for collaborative filtering for our recommendation task application.

Model 1 is our tuned matrix factorization-based collaborative filtering model(see image below). We made changes to the baseline model by integrating regularization techniques such as dropout. We added dropout of 0.1, concatenated the embedded user and movie layers together, put dropout of 0.1, built in a dense layer with 100 hidden paragraph neurons and relu activation, inserted dropout of 0.1, and lastly included a dense layer with one hidden neuron and linear activation. We achieved a minimum RMSE at epoch 27 of 0.8734. Since our baseline is 0.8773, we lowered our RMSE by 0.0039 which is a slight improvement.

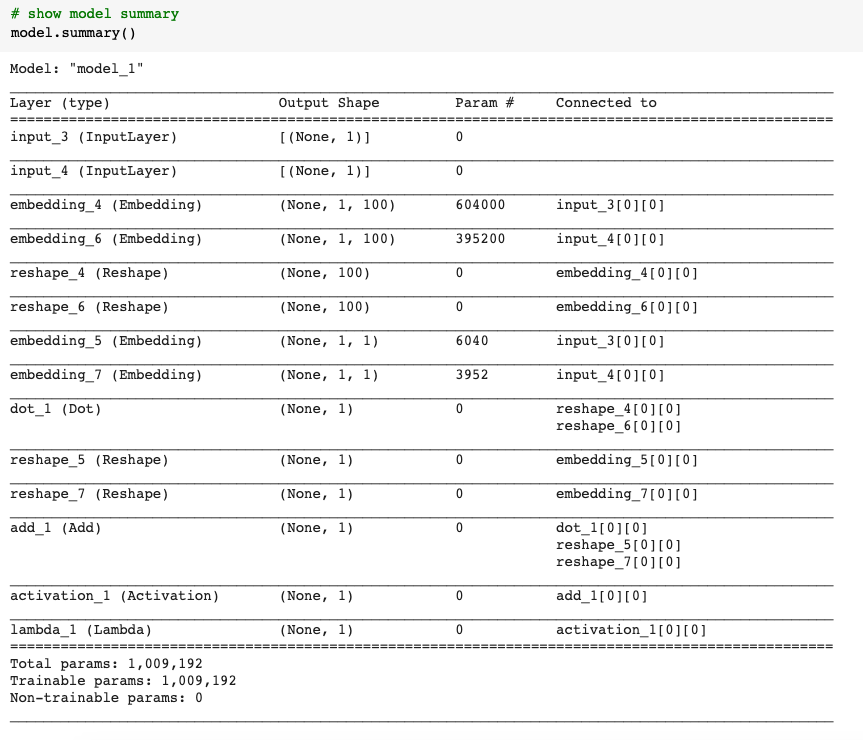


*Model Structure for Model 1*

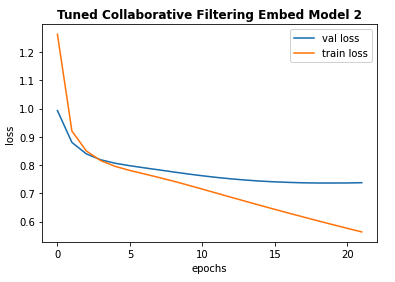


*Train and Validation Epoch vs Loss Graph for Model 1*

Model 2 is another version of our tuned matrix factorization-based collaborative filtering model . We made changes to the baseline model by running the output of the dot product through a sigmoid layer and then scaling the result using minimum and maximum ratings of the data to introduce non-linearity. Additionally, we pulled the embedded layer and reshaped operations into a separate class. We achieved a minimum RMSE at epoch 20 of 0.8583. This time we saw a larger improvement in RMSE of 0.0290, from our baseline results of 0.8773.



*Model Structure for Model 2*

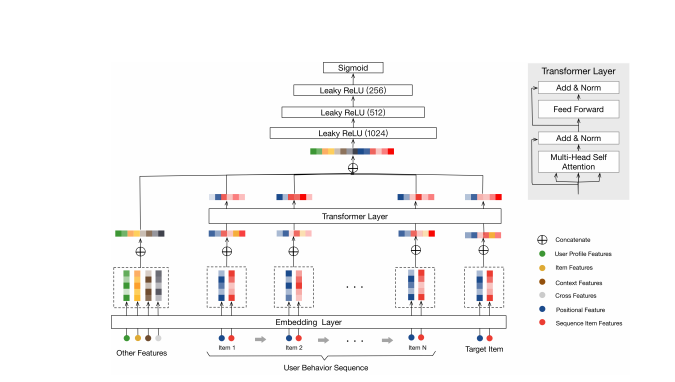


*Train and Validation Epoch vs Loss Graph for Model 2*

This hands on approach allowed us to really understand the intuition of tuning deep learning models for recommendation tasks. This will only be more beneficial to us as we implement the transformer into our model next.

**Implementing a Transformer**

To better capture the behaviors of the users in the MovieLens datasets a behavioral sequence transformer was implemented. Transformers - network architectures based on attention mechanisms - can help leverage consumers’ past behavior to tailor recommendations to them. Since transformers don’t need to process data in order, they help a lot with efficiency as training time significantly decreases.

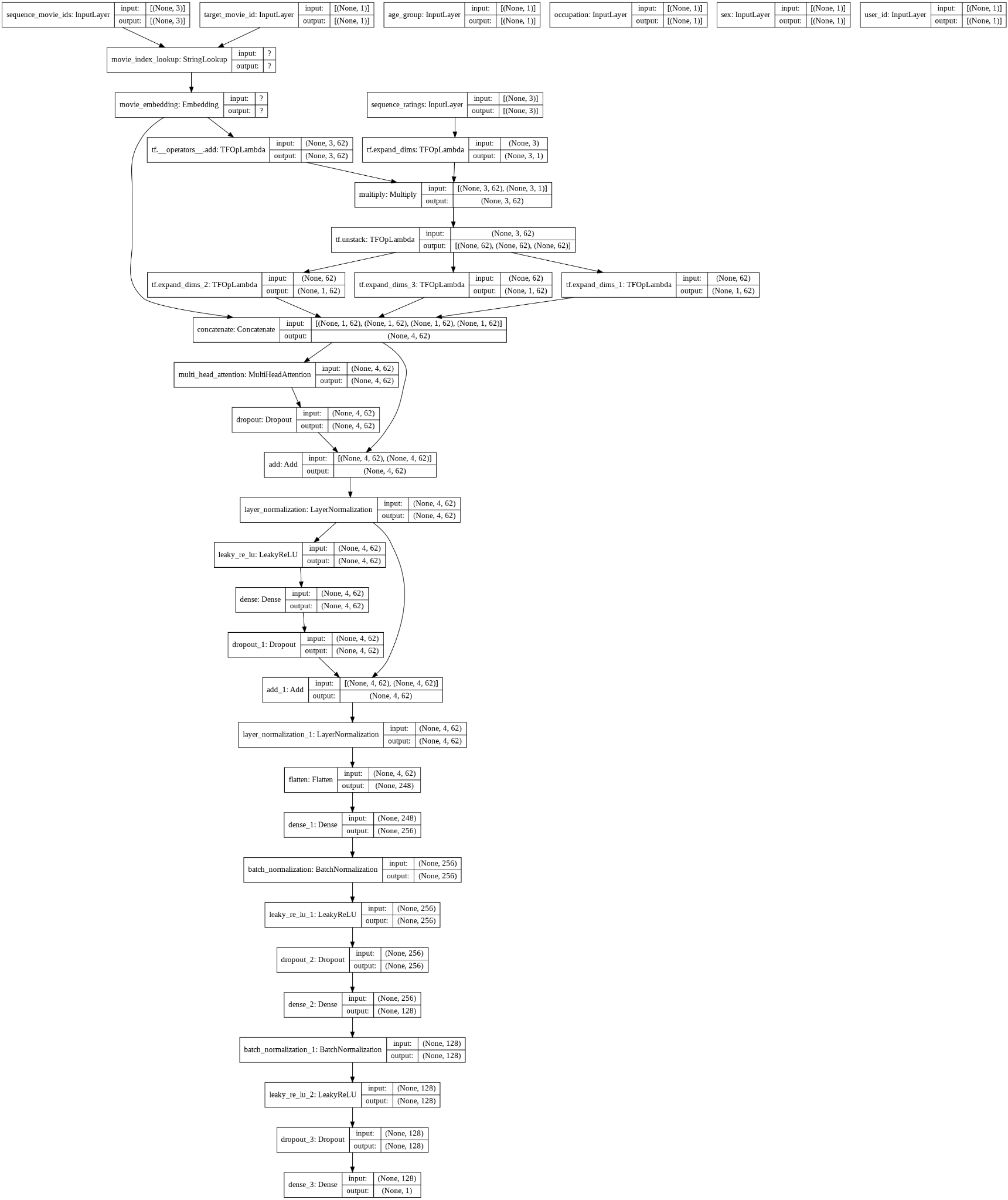


*Behavior sequence transformer(BST) model architecture*

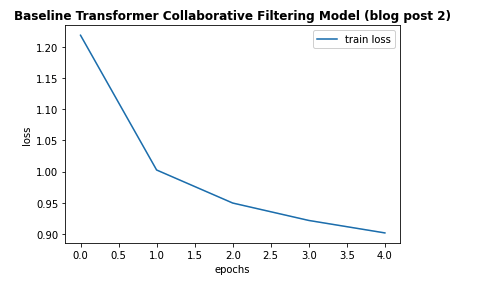
The baseline behavior sequence transformer (BST) model makes use of a transformer layer in its structure to incorporate users’ previous rating behavior for suggestion. Our baseline BST was based on the [Behavior Sequence Transformer (BST)](https://arxiv.org/abs/1905.06874) model, by Qiwei Chen et al.

In this model, we transform the movie ratings data into sequences then encode these as the input features for the transformer. First, we sort the ratings data using unix\_timestamp, and group the movie\_id values and the rating values by user\_id. We then split the movie\_ids and ratings list into a set of sequences of a fixed length and set the sequence\_length and step\_size to change the length of the input sequence to the model and control the number of sequences to generate for each user. We then process the output to have each sequence in a separate record in the DataFrame and join the user features with the ratings data.

This model inputs the users’ rating behavior and embeds them as vectors.We then concatenate a multi-hot genres vector for each movie with its embedding vector, and processed using a non-linear Dense layer. Then, we add a positional embedding to each movie embedding in the sequence, and then multiplied by its rating from the ratings sequence. Finally, we concatenate the target movie embedding to the sequence movie embeddings to create a tensor with the shape of [batch size, sequence length, embedding size], as expected by the attention layer of the transformer. We achieved a minimum RMSE at epoch 4 of 0.9612 from our baseline BST model.

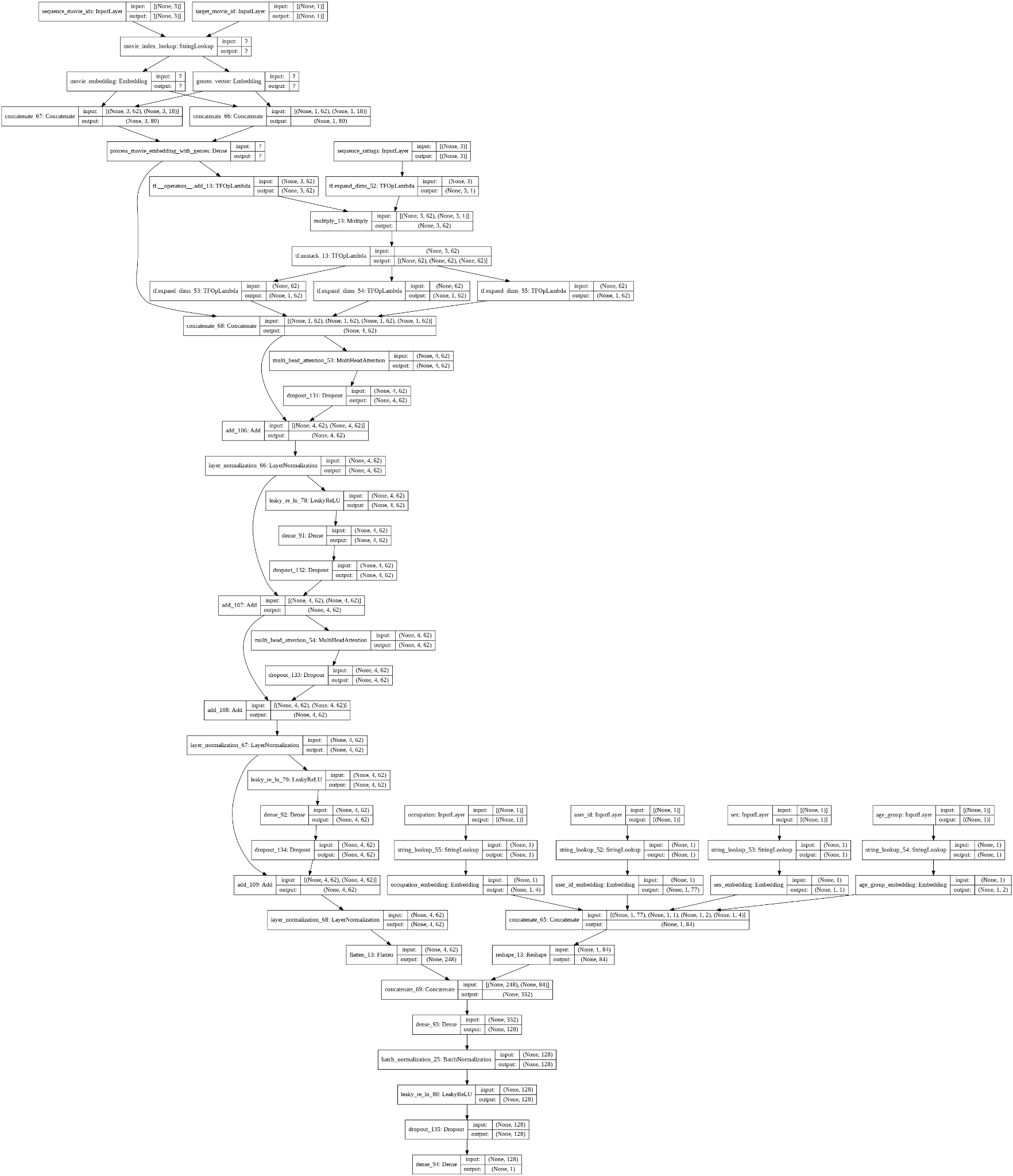


*Model Structure for Baseline Transformer*

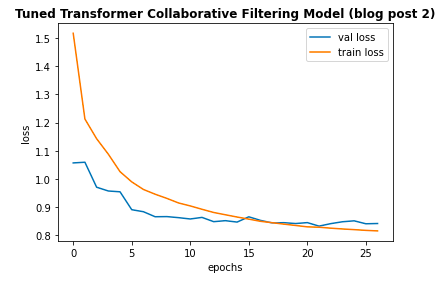


*Train Epoch vs Loss Graph for Baseline Transformer*

The tuned baseline transformer is our tuned transformer model (see images below). Compared to the baseline transformer which only splits the ratings data into 85% train and 15% test, we performed more splitting with ~80% of the ratings data in the train set, 10% in validation set, and 10% in test set. We feel that a validation set is necessary to incorporate early stopping on validation loss to detect under or overfitting. We made changes to the baseline transformer model by incorporating more input features into the encoder, and changing the model architectures. Beyond the baseline, we encoded features such as user’s occupation, gender, zip code and genre of movie. With this, our tuned transformer model may attend to richer information in capturing sequential signals underlying users’ behavior. We integrated early stopping based on validation loss. Additionally, we tuned many of the parameters such as number of attention heads, number of transformer blocks, number of hidden neurons in dense layers, number of dense layers, dropout rate, learning rate, and batch size. Through trial-and-error, we achieved a minimum RMSE at epoch 26 of 0.92. Compared to our transformer baseline results of 0.9612, we made great improvements by lowering our RMSE by 0.0412! To achieve this, our new best model has 2 transformer blocks with 8 attention heads each, only one fully-connected layer with 128 neurons, a drop-out rate of 0.5, and an Adagrad optimizer.



*Model Structure for Tuned Transformer*

* + 

*Train and Validation Epoch vs Loss Graph for Tuned Transformer*

As the training and validation loss curves indicate, the model did not appear to overfit. Moreover, the training was relatively smooth over the 25 epochs.