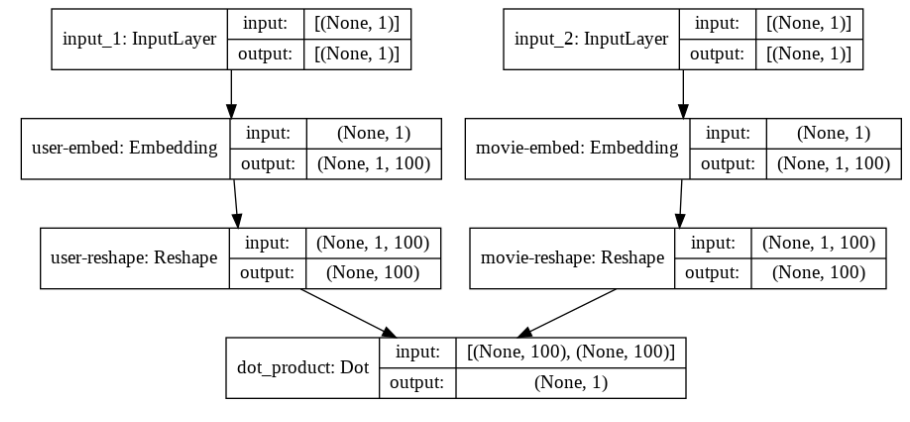
**Predicting Movie Recommendations by Leveraging Deep Learning and MovieLens Data**

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

**Overview:**

Continuing on from our [previous post](https://roma-coffin.medium.com/predicting-movie-recommendations-by-leveraging-deep-learning-and-movielens-data-17661ba4c3b1), our main objective now is to implement new deep learning architectures for collaborative filtering for our recommendation task application. In particular, we aim to develop a transformer and tune it.

In our last blog post we discussed that we want to incorporate deep learning models into recommendation systems. Our goals include finding new tasks and building better movie recommendation systems that more accurately provide personalized content for the modern consumer. We also went over a brief overview of the MovieLens dataset, the associated data collection processes, our EDA process, as well as our model methodology which is based off of [James Le’s research](https://le-james94.medium.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223). With our initial baseline model, we achieved a minimum RMSE at epoch 30 of 0.8773, so the leading MSE validation loss was 0.7627. Here, we improve upon our baseline model through using fine-tuning and transformers.

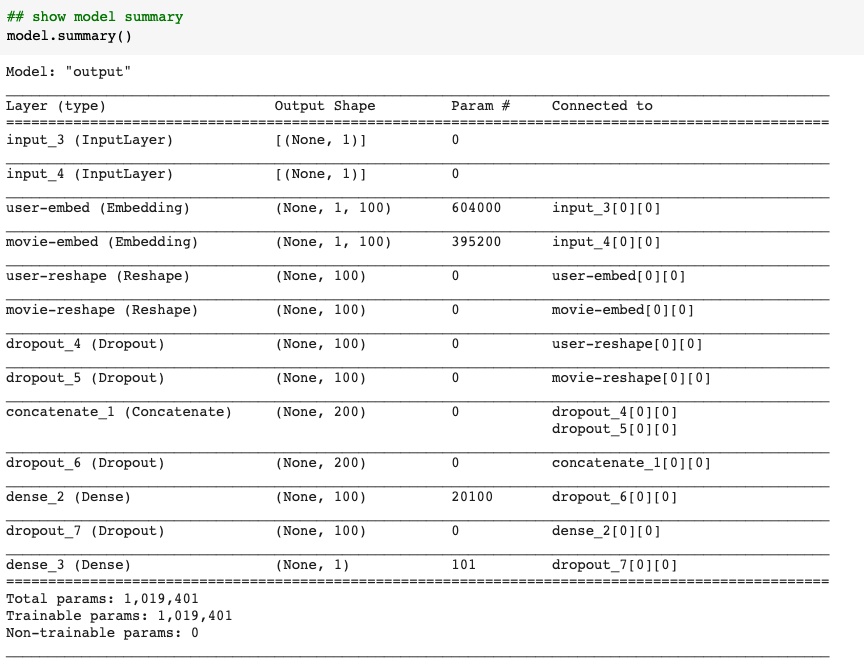


*Baseline deep learning model structure*

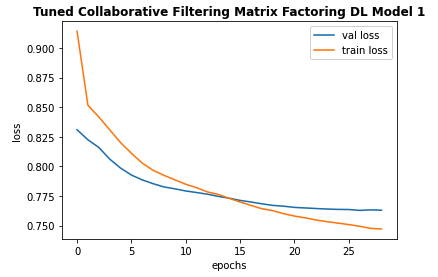
**Updates to Baseline Model:**

To enhance our models’ performance further, we explored making improvements on the baseline matrix factorization collaborative filtering model. So instead of only looking at how alike one user is to other users and then making recommendations based on their preferences, our method includes sequential user behavior, allowing us to tailor content recommendations based on users’ movie rating history. For further improvement, we continued to fine-tune the models to boost our models’ accuracy. In addition, we better understood what was and was not working through this hands on approach. Through trial-and-error, we found which fine-tuned models worked best for our task. In addition, we describe the details of two matrix factoring models we tuned and their outcome.

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.

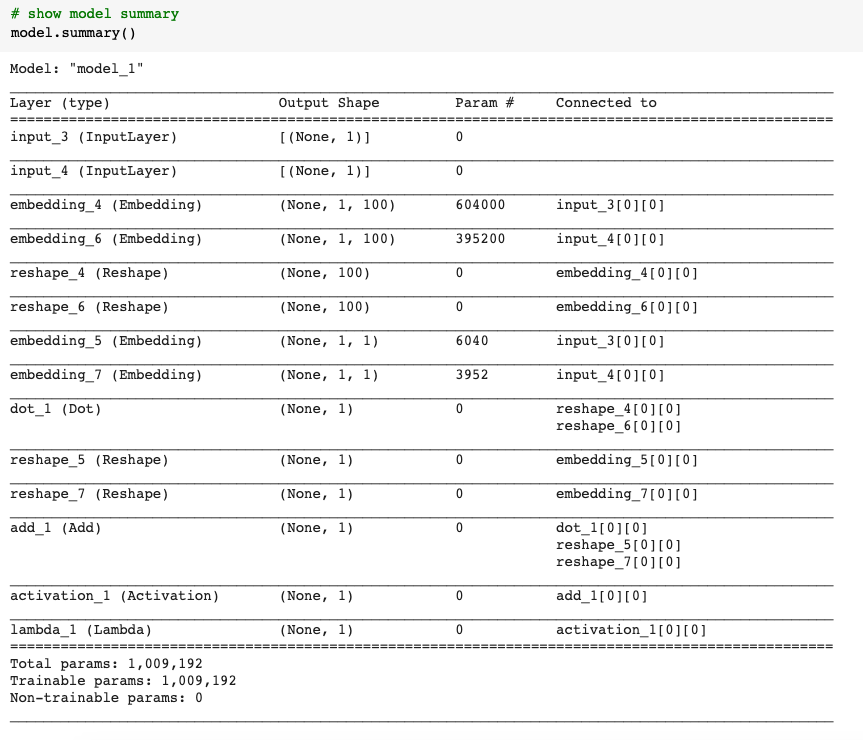


*Model Structure for Model 1*

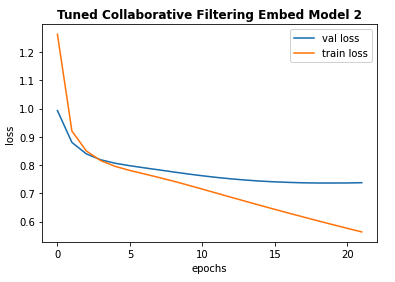


*Train and Validation Epoch vs Loss Graph for Model 1*

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*

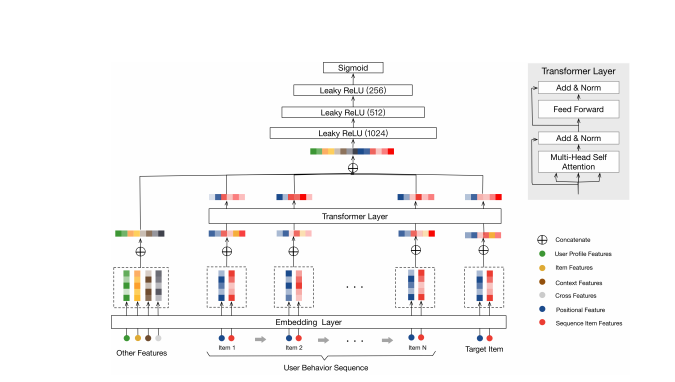


*Train and Validation Epoch vs Loss Graph for Model 2*

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.

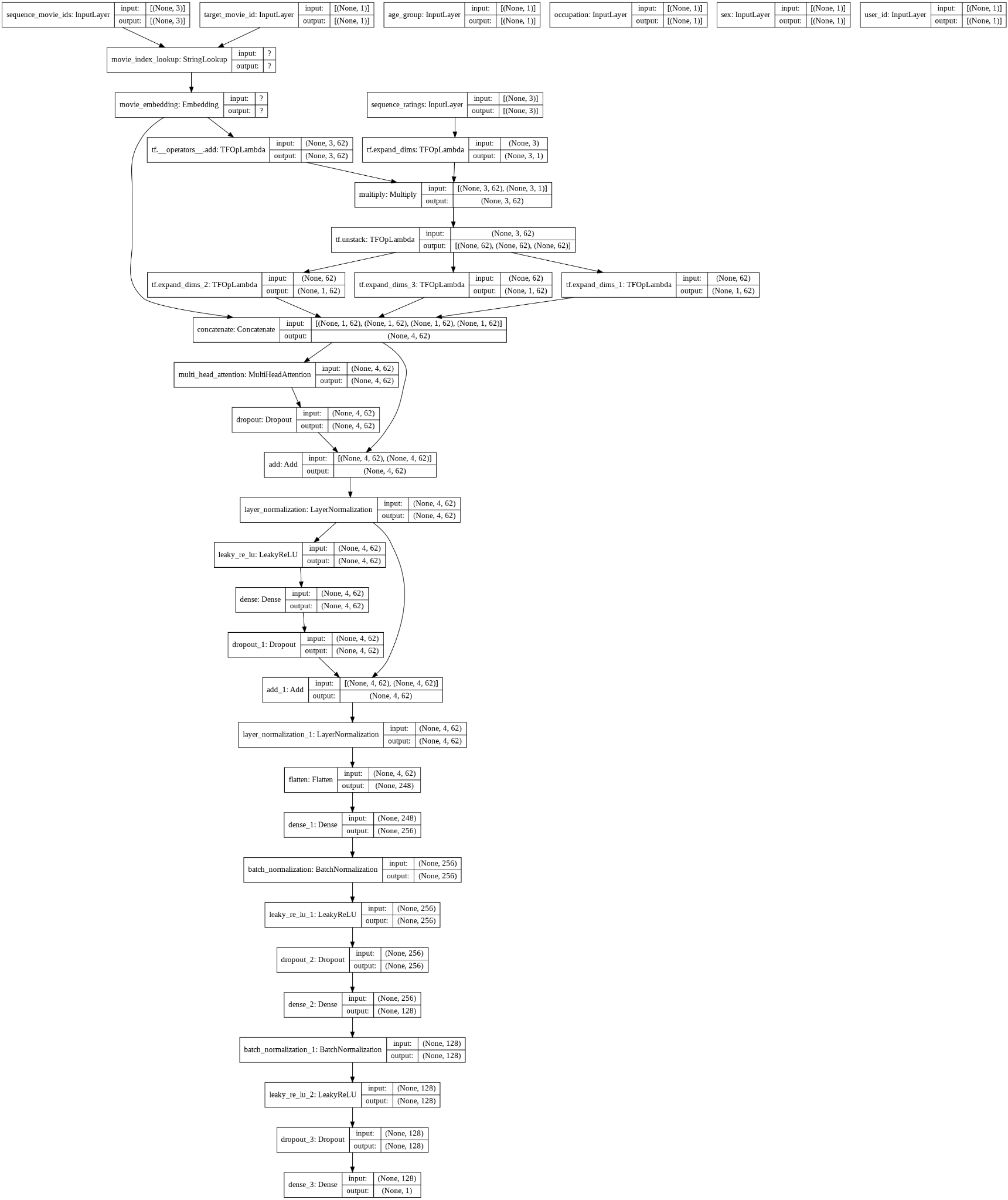


*Behavior sequence transformer(BST) model architecture*

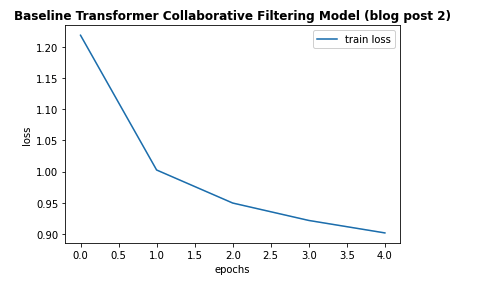
The baseline behavior sequence transformer (BST) model makes use of a transformer layer in its structure to incorporate users’ previous rating behavior for suggestion. Our baseline BST was based on the [Behavior Sequence Transformer (BST)](https://arxiv.org/abs/1905.06874) model, by Qiwei Chen et al.

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.

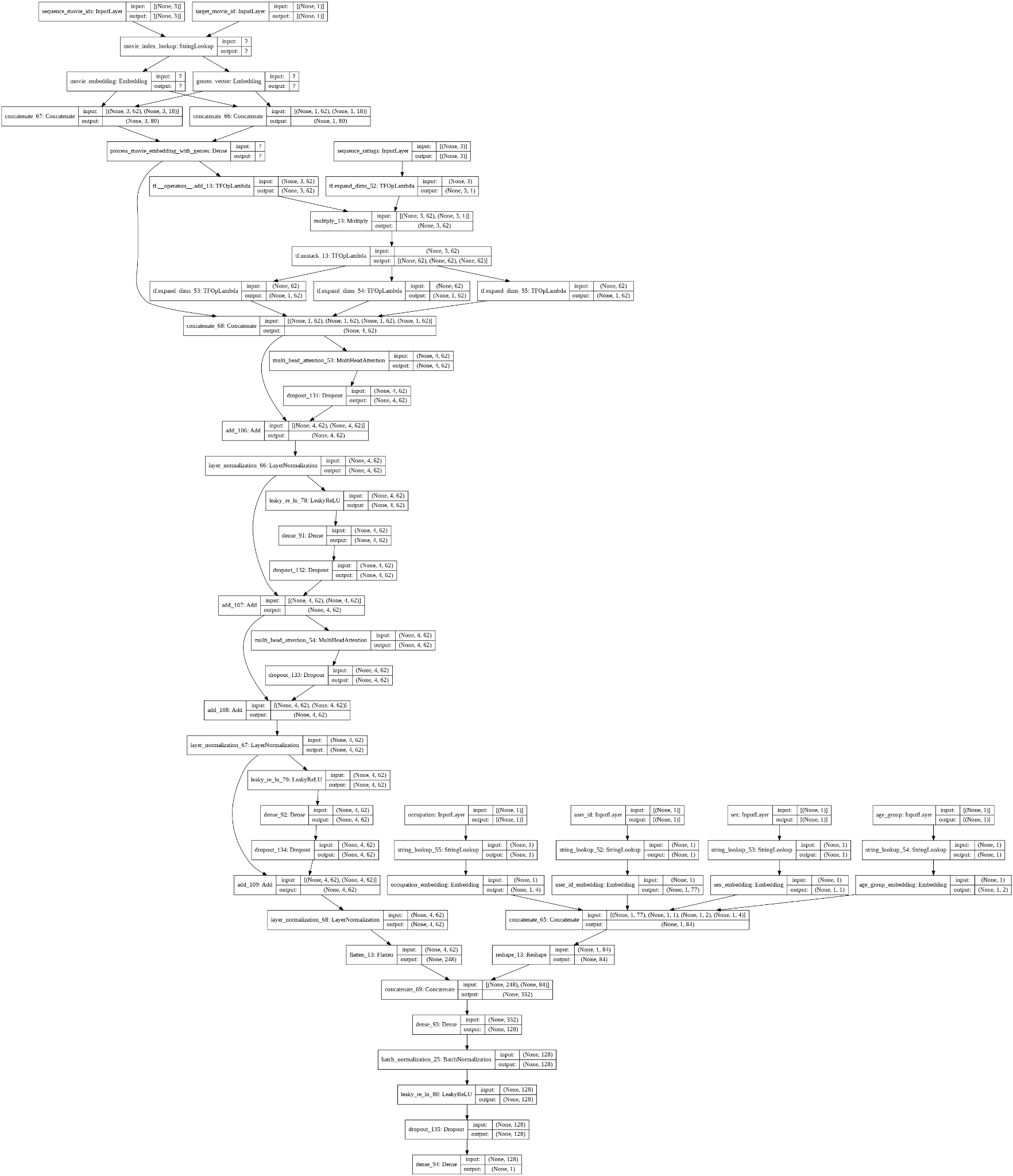


*Model Structure for Baseline Transformer*

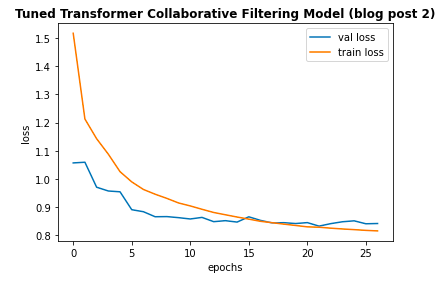


*Train Epoch vs Loss Graph for Baseline Transformer*

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.

**Next Steps:**

Our next step is to continue tuning and experimenting with transformers and try other recommendation tasks for fine-tuning, towards improving our RMSE. We will also aim to generate real movie predictions based on our final model. This has been an interesting challenge to us as we have yet to cover recommendation systems in class. As we investigate recommendation system techniques more we will refine upon our current attempts to enhance the model’s performance and generalizability. We intend to further improve upon the transformer by exploring various courses of actions such as:

* Autoencoder-to help with minimizing dimensions in data by training the network to ignore noise in data
* Skip-gram- to assist with predicting appropriate related words for a given input/word
* Continuous Bag of words-to help out with predicting an output/word by referencing the surrounding words/context

Additionally we hope to further improve our models’ performance by trying these approaches for content-based filtering as well. Stay tuned for our next blog post as we continue to further improve and apply our model!

**Citations:**

<https://www.onceupondata.com/2019/02/10/nn-collaborative-filtering/>

<https://medium.com/@jdwittenauer/deep-learning-with-keras-recommender-systems-e7b99cb29929>

<https://keras.io/examples/structured_data/movielens_recommendations_transformers/>

<https://github.com/noveens/svae_cf>

<https://le-james94.medium.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223>

<https://github.com/bradleypallen/keras-movielens-cf/blob/master/CFModel.py>

<https://fenris.org/2016/03/07/index-html>

<https://arxiv.org/abs/1905.06874>

<https://arxiv.org/abs/1706.03762>