**DEEP LEARNING APPLICATIONS FOR**

**MOVIE RECOMMENDATION**

Annie Phan

Drew Solomon

Roma Coffin

Yuyang Li

**Abstract:** Recommendation systems use consumer data to develop personalized recommendations to customers. Deep learning models in recommendation systems have become quite prevalent due to overcoming limitations of other approaches and many times increase prediction accuracy. In this paper, a vanilla embedding model for movie recommendations is established through an existing model based on prior research by James Le, a Data Advocate at Superb AI Inc.. We present our experience with implementing three deep learning approaches for a movie recommendation engine: transformer-based, autoencoder, and word2vec and use MovieLens datasets to perform our analysis. Our results support and partially improve the accuracy of these methods, and highlight the appropriate contexts for each approach based on their prediction behavior for tailored movie recommendations.

1. **Dataset and Recommendation Task Overview**

There has recently been a lot of growth in the market for streaming services as they have increased in demand. Recommendation systems have recently increased in popularity as they have proven to be successful in enhancing user-experience by using consumer data to develop personalized preferences to customers. Many recommender systems function by suggesting the latest products or services that suit the products or services users have purchased in the past. Deep learning models in recommendation systems have become quite prevalent because they overcome limitations of other approaches and many times increase prediction accuracy. In this paper, a vanilla embedding model for movie recommendations is established through an existing model by James Le, a Data Advocate at Superb AI Inc. Our paper will be using the MovieLens dataset to perform analysis. The MovieLens dataset, which is generated by the GroupLens research lab, consists of 100k, 1M, or 10M ratings on 3,900 movies from 6,040 MovieLens users. We present our experience with implementing three deep learning applications for a movie recommendation engine: transformer, autoencoder, and word2vec and use MovieLens datasets to perform our analysis. The behaviors of these approaches when executing recommendation tasks explore the context in which they are significant. Additionally, these proposed approaches can provide more accurate personalized movie recommendations when compared with current methodology.

1. **Exploratory Data Analysis**

Datasets typically need to be cleaned and understood in order to be analyzed appropriately. In this section, we will highlight some of the exploratory data analysis (EDA) that was performed that is of particular importance when exploring the MovieLens dataset. To get a sense of the distribution of ratings in the dataset, the bar chart (see figure 1) reveals the dataset is skewed to the left and the rating 4 was the most dominant(mode). Additionally, a bar chart (see figure 2) was also created to see the number of occurrences for each genre. The Popularity of Genres bar chart shows that comedy and drama are the most prevalent genres.

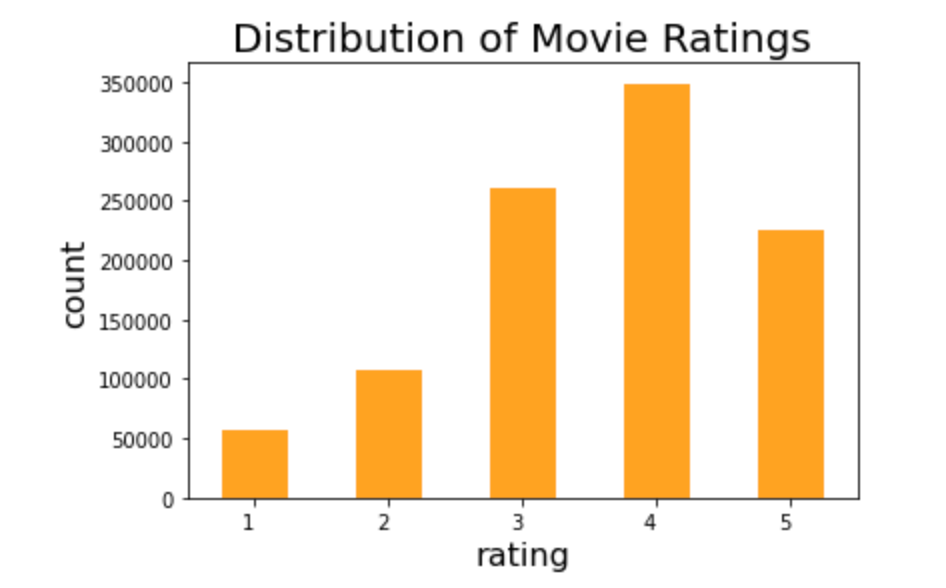


Figure 1:Bar chart of the count for each rating

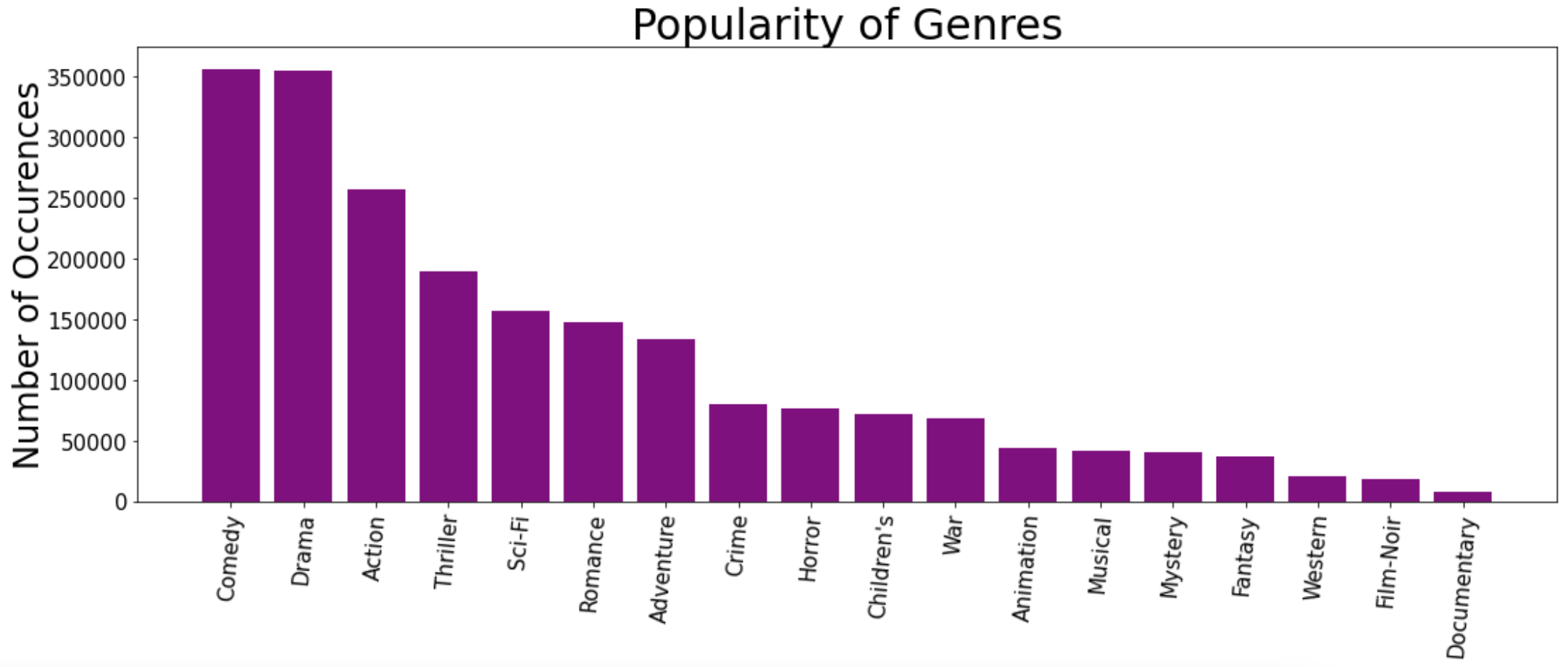


Figure 2:Bar chart of the number of occurrences for each genre

1. **Baseline Model**

**a. Model overview**

Our baseline model is based off of James Le’s deep learning model (Le, James 2018). 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.

**b. Methodology**

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.

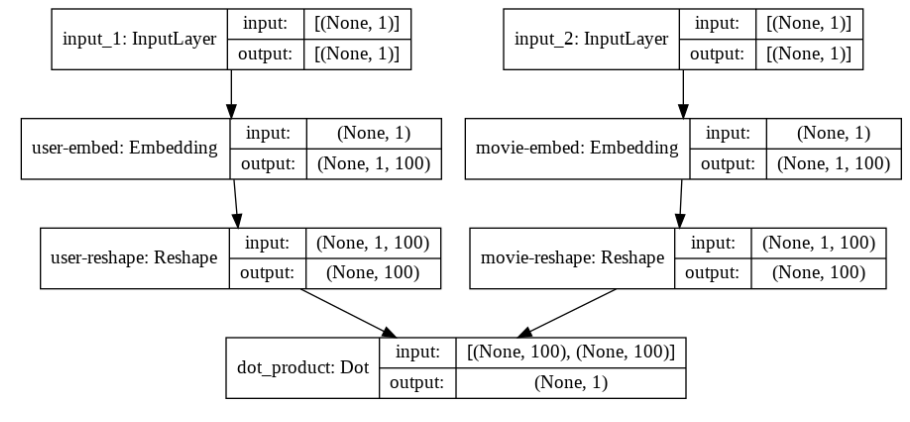


Figure 3: Baseline Model Structure

**c. Performance & Results**

We chose MSE as the loss function (take the root at the end for RMSE) and the callbacks monitor validation loss so that every time improvement occurs, model weights are remembered. We achieved a minimum RMSE at epoch 30 of 0.8773, so the leading validation loss is 0.7627.

**d. Implications**

Our baseline model gives us a relatively acceptable result, which means it does quite well in the movie recommendation under this context.

1. **Deep Learning Application 1 - Transformer: Behavioral Sequence Transformer**

To leverage sequential information from users’ behavior (beyond movie or user similarity), we implemented and tuned a transformer model to predict the ratings of target movies using users’ past movie viewing/rating history. Transformers are a type of encoder-decoder architecture that rely solely on attention mechanisms, without convolutions or recurrence (Zhang et al., 2020). In contrast to recurrent neural networks (RNN), transformers can process sequences in parallel, all at once. Moreover, their multi-head attention mechanisms allow transformers to focus on and capture relationships across both short and longer ranges. To harness the order of inputs, positional encoding is added to the sequence embeddings (Voita , 2021).

1. **Model overview**

For our task of predicting target movie ratings from users’ rating history (a benchmark example of explicit feedback), we implemented a baseline transformer model based on Qiwei Chen et al. (2019) Behavior Sequence Transformer (BST), initially designed for Alibaba e-commerce recommendations.

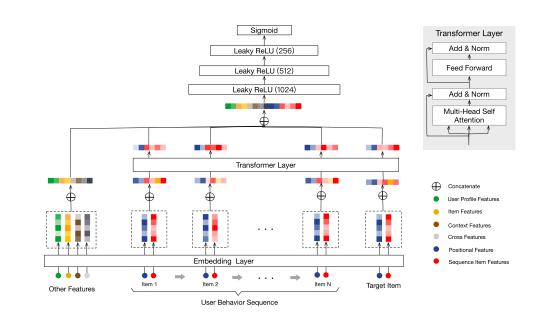


Figure 4: The BST model architecture.

Source: Chen, Qiwei, et al. *Behavior sequence transformer for e-commerce recommendation in Alibaba*. Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data. 2019.

The inputs to the BST model include the user’s behavior sequence (including the target item and positional features), and “Other Features”. The inputs are embedded as low-dimensional vectors. Then, the transformer layer learns deeper representations through the multi-head attention pooling. Then, embedded Other Features are concatenated with the transformer layer’s output. These hidden features are then passed through dense layers with leaky ReLU to capture non-linear relationships. Finally, a sigmoid function is used to generate the output predictions (Chen et al., 2019).

Our baseline model followed a Keras implementation of the BST for the MovieLens 1M dataset with just the encoder individually (Keras Team, 2020). To improve upon this baseline, we tuned the model hyperparameters, adjusted the model inputs, and expanded the approach to a larger dataset.

1. **Methodology**

To prepare the data for prediction, we processed the user, movie, and rating data into pairs of movie and rating lists for each user. Then, we split these lists into sequences (initially of length 4). During batching, the final movie or rating of a sequence is kept as the target to predict. We then created our model input layers, and defined our encoder to convert input features into lower-dimensional embeddings. Then, we created the BST model with the embedding layer, positional encoding, and a transformer block with multi-head attention, layer normalization, residual connections, and a position-wise feed-forward network. Finally, we trained the BST model on our training data with a validation set to monitor overfitting, and evaluated the model on withheld test data. Given our continuous predictions, we chose RMSE as our evaluation metric, to measure the difference between our predicted and actual target ratings and penalize larger errors (compared to MAE).

*Parameters*

To improve our model performance on the 1M ratings dataset, we tuned hyperparameters that determine the model’s structure and the training process:

|  |  |
| --- | --- |
| **Model structure hyperparameters** | **Training process hyperparameters** |
| Number of fully-connected layers [1-2] | Batch size [256; 512] |
| Number of hidden units in each FC layer [256, 128] | Optimizer [Adam, Adagrad] |
| Number of transformer blocks [1-4] | Learning rate [0.01-0.03] |
| Number of attention heads per block [2-12] | Early-stopping [True; False] |
| User ID embedding [True; False] |  |
| User features embedding [True; False] |  |
| Movie features embedding [True; False] |  |

Table 1: Hyperparameters

For our baseline model, we got an RMSE of 0.961 on the 1M data. From this model, we added more input features into the encoder, and tuned the available hyperparameters. Specifically, we encoded features such as user’s occupation, gender, zip code and genre of movie. We also tuned the above hyperparameters.

1. **Results - 1M**

*1M Dataset*

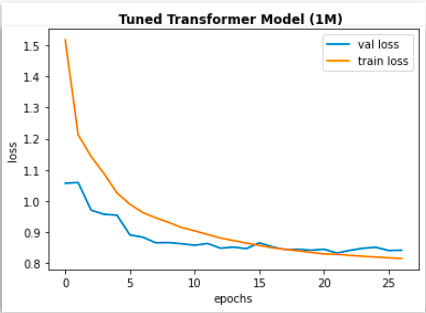


Figure 5: Train and Validation Epoch vs Loss Graph for Tuned Transformer Model (1M)

Through our tuning, we achieved a minimum RMSE at epoch 26 of 0.920, on the 1 million ratings data. This tuned transformer model had 2 transformer blocks with 8 attention heads each, 1 FC layer with 128 neurons, a drop out rate of 0.5, a learning rate of 0.02, using the Adagrad optimizer.

*10M Dataset*

We then applied our transformer-based model to the larger 10M MovieLens dataset, with 10 million ratings and over 70’000 users (GroupLens, 2021). By contrast to the 1M data, this dataset does not contain user demographics such as age or gender. We found that more ratings outweighed the lack of demographics for prediction, as we achieved a 0.876 RMSE with our baseline transformer model.

To improve upon this, we further tuned the model structure and training hyperparameters. In addition to our previous hyperparameter tuning, we also experimented with a longer input sequence length.

1. **Results - 10M**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model (10M Dataset)** | **FC Layer Units** | **Dropout Rate** | **Attention Heads** | **Transf. Blocks** | **Learning Rate** | **Batch Size** | **RMSE** |
| Baseline BST | [256, 128] | 0.1 | 3 | 1 | 0.01 | 256 | 0.876 |
| Tuned BST1 | [128] | 0.1 | 8 | 2 | 0.01 | 512 | 0.905 |
| Tuned BST2 | [128] | 0.5 | 8 | 2 | 0.02 | 512 | 0.918 |
| Tuned BST3 | [256, 128] | 0.4 | 10 | 3 | 0.02 | 512 | 0.889 |
| Tuned BST4 | [256, 128] | 0.4 | 3 | 2 | 0.02 | 256 | 0.879 |
| Tuned BST5\* | [256, 128] | 0.3 | 8 | 2 | 0.02 | 256 | 0.863 |
| Tuned BST6\*\* | [256, 128] | 0.3 | 8 | 2 | 0.02 | 256 | 0.860 |

*\*Tuned BST 5 - extended sequence length from 4 to 6*

*\*\*Tuned BST 6 - kept sequence length 6 and added user ID and movie feature input embeddings*

Table 2: Tuned Transformer Results

While the baseline BST outperformed most tuned models, the BST models trained on input sequences of length 6 (rather than 4) yielded better predictions. Thus, using slightly longer sequences improved upon our models rating predictions, perhaps due to the transformer’s ability to capture multi-range dependencies.

*Tuned BST 5*

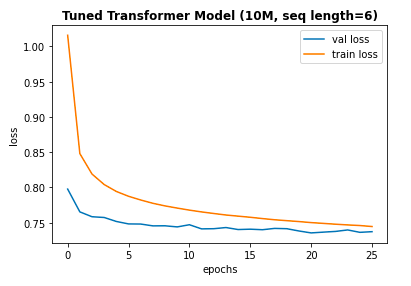
**

Figure 6: Train and Validation Epoch vs Loss Graph for Tuned Transformer Model (10M, seq length=6)

*BST-5 RMSE: 0.863*

From the current best BST-5 model, we added user ID and movie feature embeddings into the encoder, to test their impact on predictions. However, the additional embeddings did not yield a significant improvement.

*Tuned BST 6*

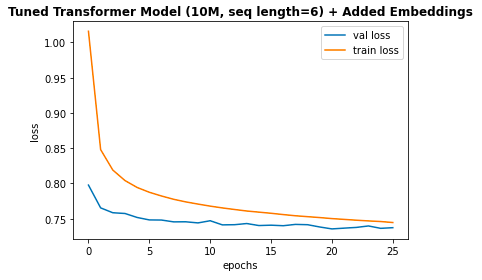
**

Figure 7: Train and Validation Epoch vs Loss Graph for Tuned Transformer Model (10M, seq length=6) + Added Embeddings

*BST-6 RMSE: 0.860*

1. **Implications**

The transformer-based model allows predictions based on past user behavior, with attention to feature relationships across multiple ranges. Thus, the BST model may capture users’ long-term movie preference patterns as well as short-term shifts (such as in mood or weather). In order to harness the sequential information, positional encoding must be added given the lack of recurrence or convolution. Our future steps would be to construct a decoder to interpret embeddings into predictions that we can rank and use to generate top-K movie title recommendations.

1. **Deep Learning Application 1 - Autoencoder: Deep Autoencoders**
2. **Model overview**

Autoencoders are a type of neural network that learns a representation or encoding of a set of data in an unsupervised manner. More specifically, autoencoders reduce data dimensions by encoding for a set of data and training a network to ignore the noise in the data. AutoRec is a compact and efficiently trainable autoencoder model for collaborative filtering (CF) in recommendation tasks that was created by a group of researchers at Australian National University, as published in the paper “AutoRec: Autoencoders Meet Collaborative Filtering” by Sedhain in 2015. AutoRec, at the time, had outperformed state-of-the-art CF techniques, including matrix factorization, RMB-CF, and LLORMA on the MovieLens dataset while also having significantly less computational time and parameters (Sedhain, 2015). As such, in our project, we seek to recreate, understand, and refine a deep learning implementation of AutoRec in keras/tensorflow for the 1M MovieLens dataset as another application for the recommendation task in our project. Our work is based off of graduate researcher in Continual Learning and Recommender Systems at the University of Toronto, [Zheda Mai](https://github.com/RaptorMai/Deep-AutoEncoder-Recommendation/blob/master/Report.pdf). In our project, we focused on testing a wider range of hyperparameters and structures, extending some experiments, and creating a pipeline for these models to predict ratings and recommend movies for users based off of Zheda’s AutoRec and Deep AutoRec implementation.

On a high level, in rating-based collaborative filtering, the number of users (let’s assign the number of users as m) and items/ movies in our dataset (let’s assign the number of items as n) is represented by a matrix R with dimension m x n, where m = number of rows - represented by the users ID and n = number of columns - represented by the movies ID and each entry R(i,j) is the ratings decided by the ith user to the jth item. As such, in this project, we implement an user-item-based autoencoder that takes in this matrix as input, project it into a low-dimension hidden (or latent) space and reconstruct another matrix in the output space with the same dimension that predicts missing ratings to recommend movies to the users.

1. **Methodology**

To allow the autoencoders to perform this task, we preprocess our MovieLens 1M dataset into the described matrix with the following steps. First, we split the data into approximately 80% of the data being assigned to the train sets, 10% to the validation set, and 10% of the data to the test set. When splitting, we stratified the data by user\_id to ensure that we have similar class balances across the different sets to ensure that the model can learn and infer without bias. Second, we transform the data in these different sets into the user-item rating matrix as described above. Since we have 6040 unique users and 3952 unique movies in our dataset, the matrix has the shape 6040 x 3952. When transforming, we have many default ratings for unobserved ratings, which can take values from 0 to 5. This set up allows us to have multiple training sets with different default ratings, which allows us to test the performance of the model with different default ratings. This set up is important because many projects that explored AutoRec for CF in recommendation use different values for the default ratings, which result in different RMSE scores.

Based off of Zheda’s implementation, we have customized loss functions to calculate the masked RMSE and loss (masked MSE) of the autoencoders and Keras does not have a default masked MSE function. These functions are necessary because according to the [paper](http://users.cecs.anu.edu.au/~u5098633/papers/www15.pdf) “AutoRec: Autoencoders Meet Collaborative Filtering,” it is crucial that the model only considers non-zero rating during inference because it simply doesn’t make sense to count ratings that are 0!

The autoencoders models we re-implement are AutoRec, which was the model proposed in the “AutoRec: Autoencoders Meet Collaborative Filtering” paper, and Deep AutoREc (Deep AE), which is a deep learning implementation of AutoRec. Deep AutoRec consists of more hidden layers and uses activation functions with non-zero negative and boundless positive parts and also dropout layers following the latent layer to prevent overfitting and learn more complicated representations.

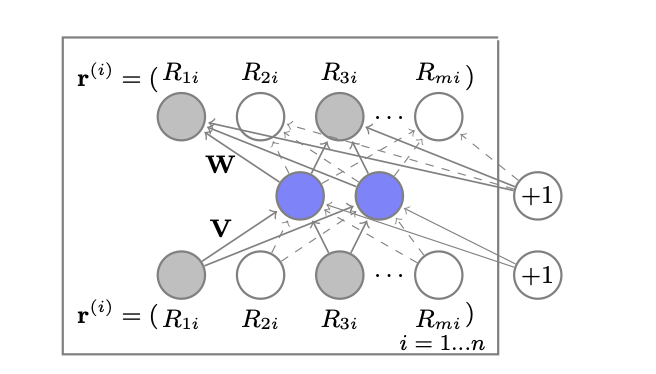


Figure 8: AutoRec structure (Sedhain, 2015)

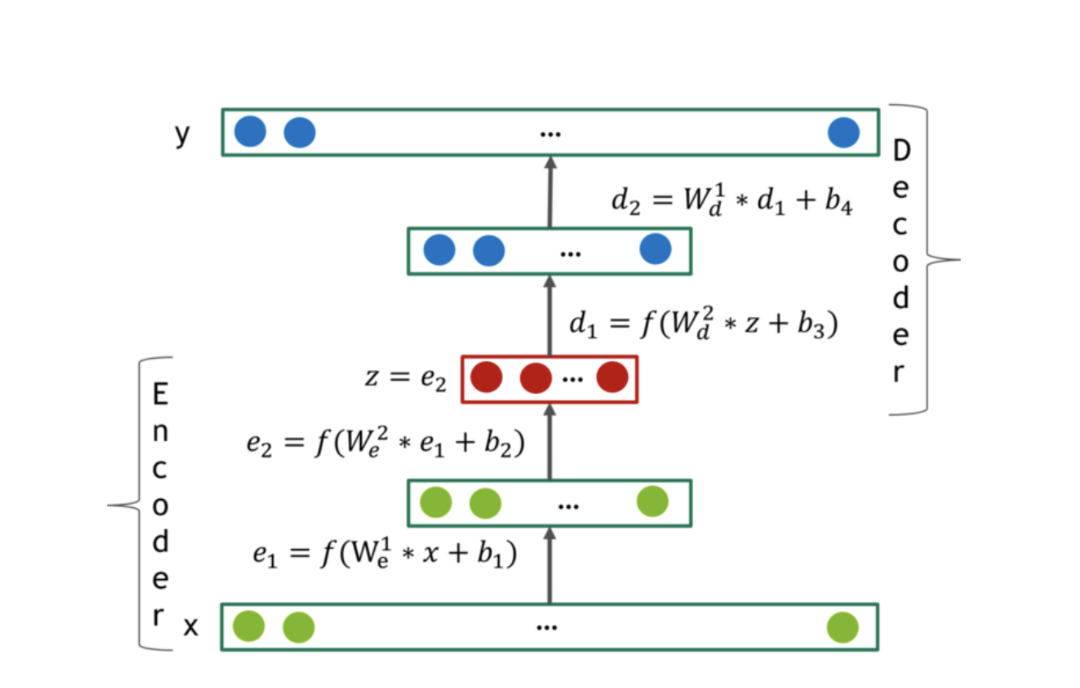


Figure 9: DeepAutoRec Structure (González-Fierro, 2018)

We tested a wide of parameters, particularly the number of layers, number of neurons in each layer, and different activation functions, the regularization lambda, drop out rate, and learning rate for both Autorec and Deep AE. We also tested various train matrices with different default rating values on the 2 models. We focused on the activation functions with non-zero negative parts and boundless positive parts (selu, elu, and LeakyReLU) because the autoencoder performs a downsample (or upsample) of the input matrix into a latent representation and then upsample (or downsample) of this matrix to create an output matrix. As such, it is important that the action values outputted by the activation functions don’t have negative or zero values, which can create issues of undefined and exploding/ vanishing gradients during back propagation and that the operations in the activation functions have an inverse operation to allow autoencoders to perform up-then-down sampling or vice versa.

Some behaviors we observed was that having 1 encoder layer, 1 latent layer, and 1 decoder layer gives us the best performance. The number of neurons in each layer following the 2 configurations [512, 256, 512] and [256, 512, 256] both gave similarly the best performance but since the configuration [256, 512, 256] has half the amount of parameters, we stuck with this option. By having additional regularization parameters, the test performance decreases. We also saw that for AutoRec, which has shallower structure. when the default rating has average=True, the model converged quicker but was much more noisy when compared to when the default rating was zero. However, for Deep AE, which was deeper, when the default rating was zero, the model converged quicker and was less noisy when compared to when the default rating was average.

To explore options to enhance our models’ performance further, we came up with the hypothesis that since the autoencoders we implemented had computationally efficient training time and parameters, we can train the model on a larger dataset to allow the model to learn even more latent representation. We decided to load in the next large MovieLens dataset following our current 1M dataset, which is the MovieLens 10M dataset. We wrote a loop to subset 1M points of the entire 10M data every time and calculate the number of user IDs and number or movie IDs to create the user-item matrix needed for the autoencoder (described above) in every loop. Unfortunately, the RAM of our Google Colab, even with the Colab Pro account, could not handle the computation. As such, we weren’t able to conduct this experiment, but fortunately we were able to train our transformer (referred to the previous section) on the MovieLens 10M dataset. In the future, we hope to find more ways to train the autoencoders on larger MovieLens dataset, such as through more efficient ways to subset and iterate through the data or use higher RAM/ higher storage platforms.

Furthermore, we followed and expanded on methods that Zheda suggested to investigate the performance of the autoencoders for collaborative filtering even more. These approaches include: adding noise, adding demographic information as predictive features, and training the model on a specific demographic subset and predict for a user in that subset.By incorporating additive Gaussian noise and multiplicative dropout noise (we tested with noise\_factor = 0.1, 0.2, 0.3, 0.4, 0.5) we did not see an improvement to our model performance. We believe this is due to the fact that the default ratings influence the scores (as mentioned above) so by including noise, the default rating is also altered. By incorporating additional predictive features had minimal effect on the performance of both the AutoRec and DeepAE models. This is because the dataset already has enough features to predict from these users, which were the previous 3952 rating features and these rating features are much more predictive of potential ratings from the users compared to demographic information. When we already have 3952 rating features for prediction, 30 demographic features have minimal impact.

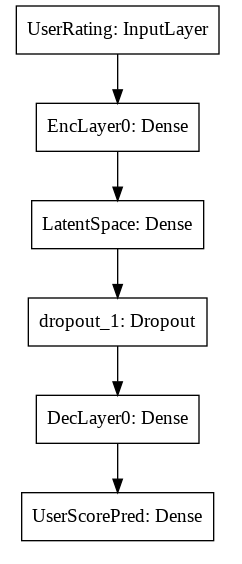
The experiment led us to think more deeply about the predictive power of features commonly available in datasets for recommendation systems of different companies. In the past, many companies used such basic demographic information to feed into their recommendation algorithm due to the nascent nature of AI technology and Big Data, for instance, Amazon also took this approach in the early days of their recommendation algorithm (Larry, 2020). As time evolves, we realize that features such as age, gender, and occupation of the users aren’t very sophisticated predictors of users’ preferences for movies, because there isn’t very high correlations between demographic information and user’s likeness for a movie. Nowadays, companies develop more sophisticated methods and knowledge to collect and use user information to bolster the performance of their recommendation algorithm, such as online interactions among users, online presence and status, user’s search terms, device types, etc (Underwood, 2020). In our case, all of the MovieLens datasets currently available have yet to had such sophisticated information, and while the datasets are continually being updated in terms of duplicating in the numbers of users and movies, we think that it would also be extremely effective and interesting for GroupLens to include more sophisticated user info that are more predictive of user’s ratings and movie preferences.

Finally, we conducted an experiment that involved looking into demographic-group specific recommendations to improve model’s performance. This approach, though unnamed in Zheda’s project, is actually called “demographic-based recommender system”[[1]](#footnote-0) based on our personal research, in which we categorize users based on a set of demographic classes. This approach is based on the notion that though all ratings from a user stem from the same distribution but different users would not necessarily have the same exact distribution. We can create clusters of users with similar characteristics in terms of age, gender, and occupation and train the autoencoder on this cluster then predict for users in that cluster. When conducting this experiment on the most populous demographic cluster, which is male users within age 18-24, we have 1538 users. When training Deep AE on this cluster, we have a test masked RMSE of 0.88, which is lower than our best autoencoder’s test RMSE of 0.86 (more about this model in the next session), which is the Deep AE model trained and predicted on the full 6040 x 3592 user-item matrix. This makes sense because we have less data points (less users) in this demographic subset. When carrying out ratings predictions and movie recommendations for a specific user (the user with the ID = 2000), who happens to be a male between 18 - 24 years old, Deep AutoRec got 63% of his ratings correct, which is also lower than what Deep AutoRec trained on the full 6040 x 3952 user-item matrix had (70%, and we will talk more about this in the next session). In a second experiment within this “demographic-based” approach to further explore the behavior of the model, we only clustered user with the age of 18 - 24 years old to train and predict the model on, since doing so will give me a larger subset of users so there will be more data points (we got 2096 users). With this configuration, we got a higher masked RMSE and more accurate prediction than Deep AE for male in the age of 18 - 24 years old (test masked RMSE = 0.87 and prediction accuracy = 67%). However, this is still lower than our best model: Deep AE trained on the full dataset. Please refer to the following screenshots for the results we got with this model and configuration.

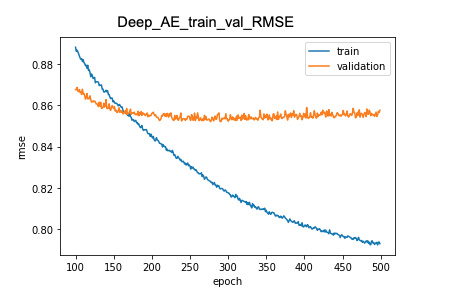
Though our “demographic-based recommender system” experiment didn’t seem to significantly bolster the performance of the autoencoders, we wouldn’t immediately disregard this approach. We are aware that the features available for the MovieLens datasets and our approaches air on the side of simpler, more basic approaches. Research has shown that for “demographic-based recommender system” (Underwood, 2020) to significantly bolster the performance of the recommender, extensive market research is needed.

1. **Performance & Results**

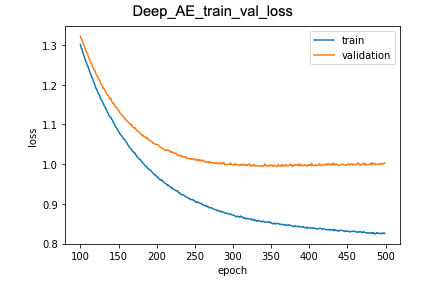
After various experiments and approaches, we’ve concluded that our best Autoencoder Model is the Deep AutoRec model, trained on the full 6040 x 3952 the training matrix with default rating set to 0 with the set of parameters: layers = [256, 512, 256], dropout = 0.8, activation and last\_activation = 'selu’, and regularization alpha of encoder and decoder = 0.001. We achieved a masked RMSE of 0.858 and loss of 0.9957. Since our baseline is 0.877, we lowered our RMSE by 0.019 which is a slight improvement. Additionally, for the Recommendation Pipeline the predicted ratings and recommend unseen movies for the user with ID 2000 resulted in 70% of his ratings being correct.



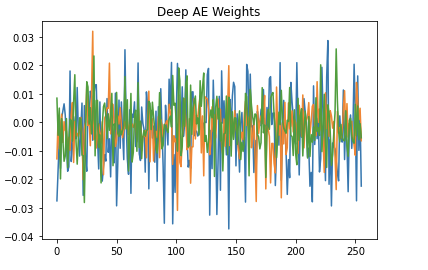
*Figure 10: Best Autoencoder: Deep AutoRec Model Structure*



*Figure 11: Train and Validation Epoch vs RMSE Graph for Deep AutoRec*



*Figure 12: Train and Validation Epoch vs Loss Graph for Deep AutoRec*



*Figure 13: Weights of Deep AutoRec*

**

*Table 3: Movie Ratings Prediction of User with ID = 2000 for Deep AutoRec*

**

*Percentage of Ratings Correct of User with ID = 2000 for Deep AutoRec*



*Table 4: Recommended Movies for User with ID = 2000 based on Predicted Ratings for Deep AutoRec*

1. **Implications**

Autoencoders are classifical implementations of recommendation engines, as the depth of the layers can exponentially reduce computational costs since more layers can exponentially decrease the dimension of the training data needed to learn some functions while also yielding better compression compared to shallow autoencoders (Goodfellow, 2016). Moreover, some of the components of the autoencoders, such as the encoder, can be separately trained on several independent data subset and re-used in other end-to-end neural-networks while still keeping the potential to be globally optimized by back-propagation (Martínez, 2019). However, one drawback of Autoencoders is that transforming the data points into a user-item matrix can be computationally expensive for large datasets, as we have seen in the case of the 10M MovieLens datasets.

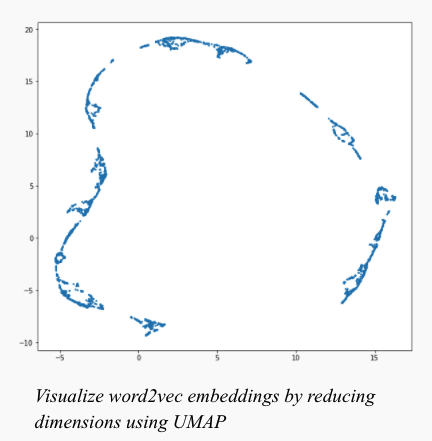
As such, it seems like Autoencoders might be a good application for smaller and medium-sized companies, such as start-ups, who had smaller and more niched sets of customers to serve. In the future, we would like to explore ways to further train and predict our autoencoder on larger datasets with a variety of more sophisticated user features as well as exploring more advanced deep learning implementations that can be combined with autoencoders to bolster performance.

1. **Deep Learning Application 3: Word2Vec - Word2Vec for Content Filtering**
2. **Model overview**

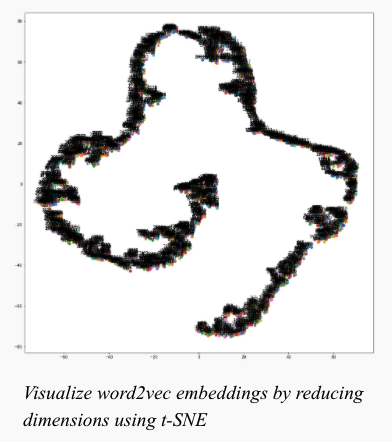
Content-based filtering is based on the idea that users who previously had similar taste will also have in the future. Word2Vec is a neural network model which learns word embeddings from large datasets. We implemented a Word2Vec content-based filtering model for this task to recommend movies based on users’ previous preferences.

1. **Methodology**

Our approach started with preprocessing data and splitting data into train and validation sets. Then we created Word2Vec embeddings and built a model which has a vocabulary of 3396 unique words with vectors of size 100 each. We also generate visualizations of Word2Vec embeddings by reducing dimensions using t-SNE and UMAP.



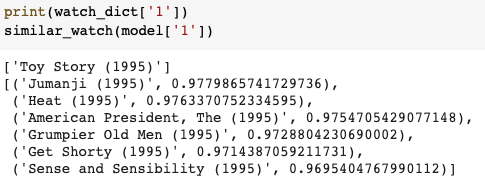
*Figure 14: Word2Vec embeddings visualization using UMAP*



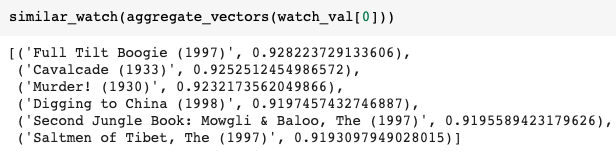
*Figure 15: Word2Vec embeddings visualization using t-SNE*

1. **Performance & Results**

In order to make the recommendations, we generated two functions. One takes a movie as an input and returns the top 6 similar movies. And the other one takes the average of all the vectors and returns the top 6 similar movies. The results seem to be pretty relevant, as shown below.



*Figure 16: Recommendations outputs 1*



*Figure 17: Recommendations outputs 2*

1. **Implications**

It turns out that Word2Vec content-based filtering works well under this context. And based on the reflection, we might want to try Word2Vec collaborative filtering in the future.

1. **Results Table Summary**

Below is a table that summarizes our different applications, key parameters, and results

|  |  |  |  |
| --- | --- | --- | --- |
| *Application* | *Recommendation Method* | *Parameters and Configurations of Best Model* | *Results* |
| Vanilla Embedding | Collaborative Filtering | Number of users = 6040  Number of movies = 3952  Number of dimensional embeddings for movies and users  = 100 | RMSE = 0.8773 |
| Behavioral Sequence Transformer | Collaborative Filtering | FC Layer Units = [256, 128]  Dropout Rate = 0.3  Attention Heads = 8  Transformation Blocks = 2  Learning Rate = 0.02  Batch Size = 256 | RMSE =0.860  (10M) |
| Deep AutoEncoders | Collaborative Filtering | Deep AE model  Number of users = 6040  Number of movies = 3952  Layers = [256, 512, 256]  Dropout rate = 0.8 Activation and last\_activation = 'selu’  Regularization alpha of encoder and decoder = 0.001 | masked RMSE = 0.858 |
| Word2Vec | Content Filtering | window = 8  sg = 1  hs = 0  negative = 15  alpha = 0.001  min\_alpha = 0.0001  seed = 34  progress\_per = 250  epochs = 20  report\_delay = 1 | probability of being the nearby word for every word in the vocabulary on average for top 6 movies recommended = 0.97 (using a movie’s vector as input)  probability of being the nearby word for every word in the vocabulary on average for top 6 movies recommended = 0.92 (using average of all the vectors of movie that user watches as input) |

Table 5: Results Summary

1. **Conclusion**

Our analysis shows that the deep learning applications: Transformer, Autoencoder, and Word2Vec can be valuable applications in recommender systems. This is supported by the performance of these approaches when executing recommendation tasks. These methods are able to achieve significant results which proves that these methods should be considered and used when performing future recommendation tasks. Future work includes further investigations on implementing a decoder in the transformer model to generate movie titles from the predictive ratings. Additionally, further research could explore the demographic-based approach for Autoencoders more, although it was not the best performing Autoencoder approach, it showed a lot of promising results. Lastly, one alternative for Word2Vec is to consider Word2Vec collaborative filtering. This is based on the fact that Word2Vec content-based filtering worked well in our analysis for recommendation systems.

Works Cited

Chen, Qiwei, et al. "Behavior sequence transformer for e-commerce recommendation in alibaba." *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 2019.

González-Fierro, Miguel. “Introduction To Recommendation Systems With Deep Autoencoders.” 2018. <https://miguelgfierro.com/blog/2018/introduction-to-recommendation-systems-with-deep-autoencoders/>

Goodfellow, Ian, et al. “Deep Learning.” *Deep Learning, An MIT Press Book*, 2016, [www.deeplearningbook.org](http://www.deeplearningbook.org)

Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). [Deep Learning](http://www.deeplearningbook.org/). MIT Press. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [978-0262035613](https://en.wikipedia.org/wiki/Special:BookSources/978-0262035613).

Grimaldi, Emma. “How to build a content-based movie recommender system with Natural Language Processing.” 2018. <https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing-25ad400eb243>

GroupLens. “MovieLens 10M Dataset.” *GroupLens*, 2 Mar. 2021, grouplens.org/datasets/movielens/10m.

Hardesty, Larry. “The History of Amazon’s Recommendation Algorithm.” *Amazon Science*, 21 Aug. 2020, www.amazon.science/the-history-of-amazons-recommendation-algorithm.

Joshi, Prateek. “Building a Recommendation System using Word2vec: A Unique Tutorial with Case Study in Python.” 2019. <https://www.analyticsvidhya.com/blog/2019/07/how-to-build-recommendation-system-word2vec-python/>

Karani, Dhruvil. “Introduction to Word Embedding and Word2Vec.” 2018. <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>

Le, James. “The 4 Recommendation Engines That Can Predict Your Movie Tastes.” 2018. <https://le-james94.medium.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-bbec857b8223>

Mai, Zheda. “[Deep-AutoEncoder-Recommendation](https://github.com/RaptorMai/Deep-AutoEncoder-Recommendation)” 2019. <https://github.com/RaptorMai>

Martínez, Gerard. “Autoencoders for the Compression of Stock Market Time Series.” *Medium*, 22 Apr. 2019, towardsdatascience.com/autoencoders-for-the-compression-of-stock-market-data-28e8c1a2da3e.

Roy, Sravan. “[RecommendationSystem-Word2vec](https://github.com/sravanroy/RecommendationSystem-Word2vec).” <https://github.com/sravanroy/RecommendationSystem-Word2vec/blob/master/word2vec.ipynb>

Sedhain, Suvash, et al. "Autorec: Autoencoders meet collaborative filtering." *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015.

<http://users.cecs.anu.edu.au/~u5098633/papers/www15.pdf>

Team, Keras. “Keras Documentation: A Transformer-Based Recommendation System.” *Keras,* 2020, keras.io/examples/structured\_data/movielens\_recommendations\_transformers/.

Underwood, Corinna. “Use Cases of Recommendation Systems in Business – Current Applications and Methods.” *Emerj*, 4 Mar. 2020, emerj.com/ai-sector-overviews/use-cases-recommendation-systems.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Voita, Lena. “Sequence to Sequence (seq2seq) and Attention.” *Seq2seq And Attention*, 2021, lena-voita.github.io/nlp\_course/seq2seq\_and\_attention.html

Zhang, Aston, et al. “Transformer.” *Dive into Deep Learning*, 2020, d2l.ai/chapter\_attention-mechanisms/transformer.html.

1. https://emerj.com/ai-sector-overviews/use-cases-recommendation-systems/ [↑](#footnote-ref-0)