# **Predicting Movie Recommendations by Leveraging Deep Learning and MovieLens Data(Part 3)**

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

**Overview:**

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. 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 post, we summarize our approaches in generating movie recommendations based on users or movie information using deep learning. We describe our process to improve the performance of the transformer and implement more deep learning applications for movie recommendation tasks such as autoencoders and word2Vec. Finally, we identify possible next steps to further improve upon our model.

In our [first blog post](https://roma-coffin.medium.com/predicting-movie-recommendations-by-leveraging-deep-learning-and-movielens-data-17661ba4c3b1) we discussed 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 vanilla embedding baseline model, we achieved a minimum RMSE at epoch 30 of 0.877, so the leading MSE validation loss was 0.763. From our baseline model, we went on to our [second blog post](https://roma-coffin.medium.com/predicting-movie-recommendations-by-leveraging-deep-learning-and-movielens-data-part-2-20baff367c0b) where we implemented new deep learning architectures for collaborative filtering for our recommendation task applications. Specifically we developed a transformer and began to tune it. Since our second blog, we have dug deeper into our transformer models and developed advanced deep learning applications for movie recommendation tasks like autoencoders and word2Vec.

**Transformer:**

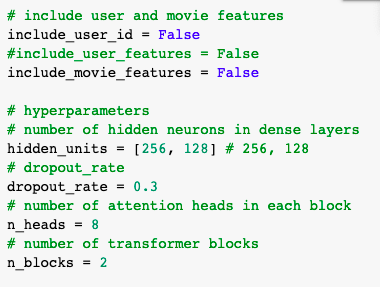
To enhance our models’ performance further, we explored making improvements on the baseline behavior sequence transformer (BST) model and our first attempt on a tuned transformer model from our last blog post([blog post 2](https://roma-coffin.medium.com/predicting-movie-recommendations-by-leveraging-deep-learning-and-movielens-data-part-2-20baff367c0b))**.** When implementing the transformer model we aimed to predict the rating of target movies using users’ past movie rating history as well as information about their demographics. The major updates from our previous transformer models include using the MovieLens 10M dataset (and therefore dropping user demographic features) and further hyperparameter tuning, including with movie sequence length. It was interesting to find that by using more data (MovieLens 10M) without user demographics performs better than less data (MovieLens 1M) with the user demographic information. The table below describes the tuned models and their associated test RMSE, all of which used the MovieLens 10M dataset and used Adagrad as an optimizer . As seen from the table below our tuned BST 5 and tuned BST 6 models achieved the best results. We will elaborate further on both of these models.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

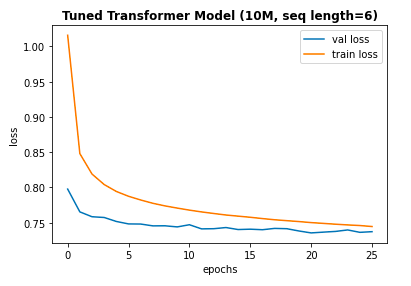
*\*Tuned BST 5 Also extended sequence length from 4 to 6*

*\*\*Tuned BST 6 Kept sequence length 6 and added userid and movie feature input embeddings*

The tuned BST 5 is our tuned transformer model that was able to achieve the second lowest test RMSE of 0.863. Compared to the previously tuned BST models, it is worth noting that by extending the length of the movie and rating sequences from 4 to 6, we boosted our model predictions. This tuned transformer-based model did not encode movie features or user IDs into embeddings. The model had 2 fully-connected layers with 256 and 128 neurons, a dropout rate of 0.3, 2 transformer blocks, 8 heads, and it was trained with a batch size of 256. It appears that using slightly longer sequences improved upon our model’s ability to predict ratings, given our positional embedding to capture sequence order and the ability of multi-headed attention to capture both short and longer-term dependencies.

****

*Tuned hyperparameters for BST 5*

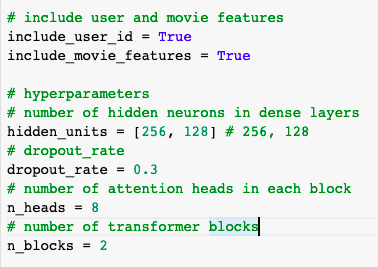
**

*Train and Validation Epoch vs Loss Graph for BST 5*

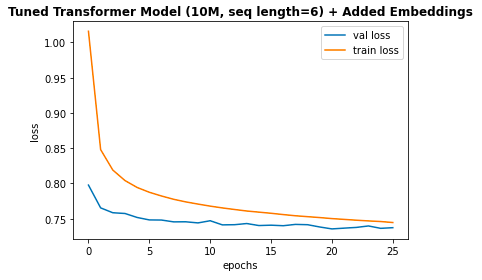
****

*Test RMSE for BST 5*

From the BST 5 model, the tuned BST 6 included user ID and movie features into the encoder embeddings, beyond the movie sequence and target movie embedding, and was able to achieve the lowest test RMSE of 0.860. However, this does not represent a significant improvement upon BST 5’s RMSE of 0.863. Otherwise, the sequence length and other parameters were kept the same. Thus, interestingly, adding the user ID and movie features as embeddings in the encoder did not meaningfully improve the accuracy of target movie rating predictions, for the 10M movie rating data. Therefore, we believe the transformer was able to sufficiently capture relationships between movies and users without the additional embeddings.

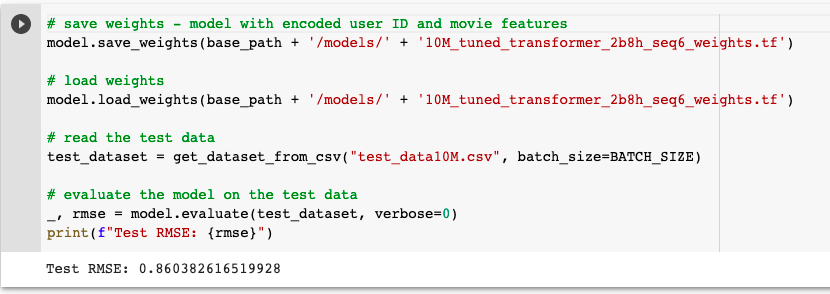


*Tuned hyperparameters for BST 6*

**

*Train and Validation Epoch vs Loss Graph for BST 6*

As the training and validation loss curves for both BST 5 and BST 6 indicate, we did not observe overfitting and the training over the 25-30 epochs was relatively smooth.



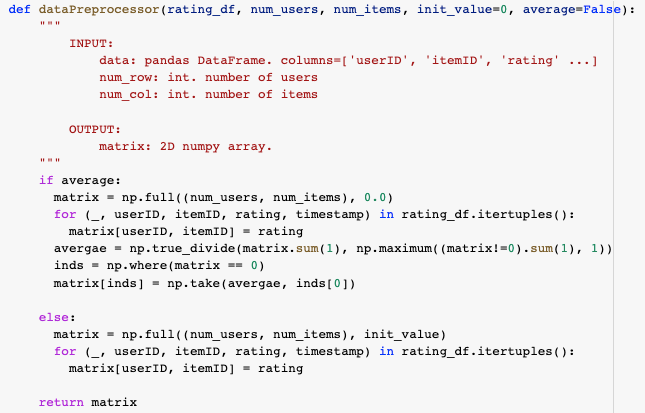
*Test RMSE for BST 6*

**Autoencoder:**

Autoencoders are a type of neural network that learns a representation or encoding of a set of data in an unsupervised manner. Autoencoders do so by reducing data dimensions through 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 the Australian National University, as published in the paper “AutoRec: Autoencoders Meet Collaborative Filtering” 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. 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.

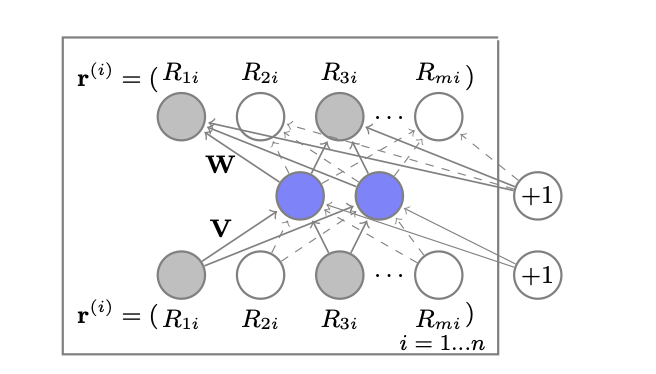
To allow the autoencoders to perform this task, we preprocess our MovieLens 1M dataset into the described matrix with the following steps. First, we split the data into approximately 80% of the data being assigned to the train sets, 10% to the validation set, and 10% of the data to the test set. When 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. For example if a user is only in training not in test, then the RMSE will be 0 for this user while if the user is only in test and not in training then the RMSE will be higher for this user. Second, we further transform the data in these different sets into the user-item rating matrix as described above. This is done by the function dataPreprocessor, please see the image below for this function. The function transforms the data frame into a 2-D matrix where rows represent the users, columns represent the movies, and each entry is a rating of a specific user for a specific movie. Since we have 6040 unique users and 3952 unique movies in our dataset, the matrix has the shape 6040 x 3952. This allows the Autorec to take in this matrix to learn the representation of that matrix through the latent layer and outputs a matrix with the same dimension that shows this learnt representation. In the data Preprocessor function, the init\_value is the default rating for unobserved ratings, which can take values from 0 to 5. When the average is set to True, the unobserved rating will be set as the average of all the ratings the user has put. 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.



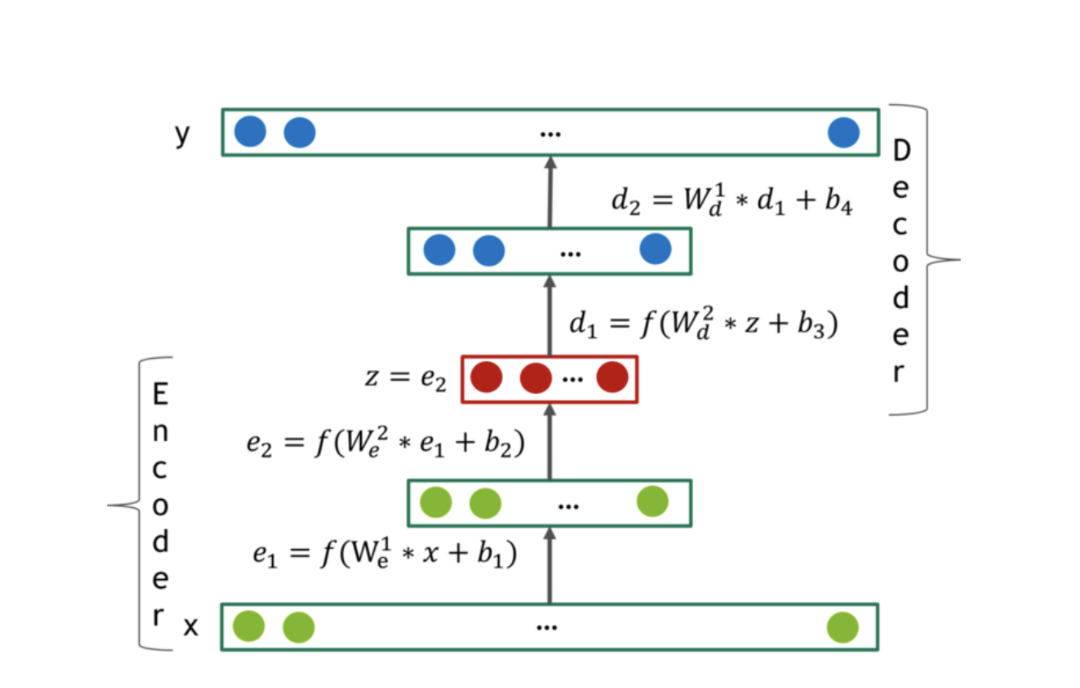
Based off of Zheda’s implementation, we have customized loss functions to calculate the masked RMSE and loss (masked MSE) of the autoencoders. These are the functions called masked\_mse, masked\_rmse, and masked\_rmse\_clip. These functions are necessary because according to the [paper](http://users.cecs.anu.edu.au/~u5098633/papers/www15.pdf) “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! As Keras does not have a default masked MSE function, the masked MSE and masked RMSE functions we developed based on Zheda’s approach takes the form of this equation, with ri = the actual rating that the user gave, yi = the predicted rating and mi = the masked values of the ratings, which takes on mi = 1 when the rating is non-zero and mi = 0 when rating is 0. Please refer to this photo for the equation:



The autoencoders models we re-implement are AutoRec, which was the model proposed in the “AutoRec: Autoencoders Meet Collaborative Filtering” paer, and Deep AutoREc (Deep AE), which is a deep learning implementation of AutoRec. Deep AE 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.



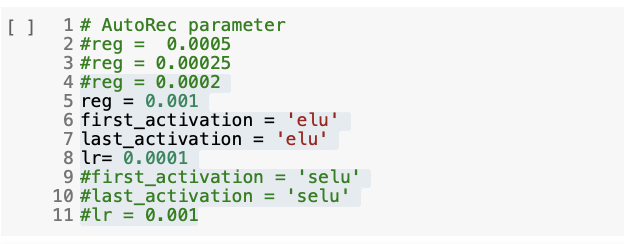
This image is sourced from [Sedhain](http://users.cecs.anu.edu.au/~u5098633/papers/www15.pdf) and describes the structure of AutoRec



This image is sourced from [González-Fierro](https://miguelgfierro.com/blog/2018/introduction-to-recommendation-systems-with-deep-autoencoders/) and describes the structure of DeepAutoRec (Deep AE )

While Zheda had tested various parameters, particularly the number of layers, number of neurons in each layer, and different activation functions, we decided to test even wider range of these parameters as well as some other parameters that Zheda hadn’t tested, such as the regularization lambda, drop out rate, and learning rate for both Autorec and Deep AE.

Please see the screenshot below for the set of parameters that we tested with AutoRec



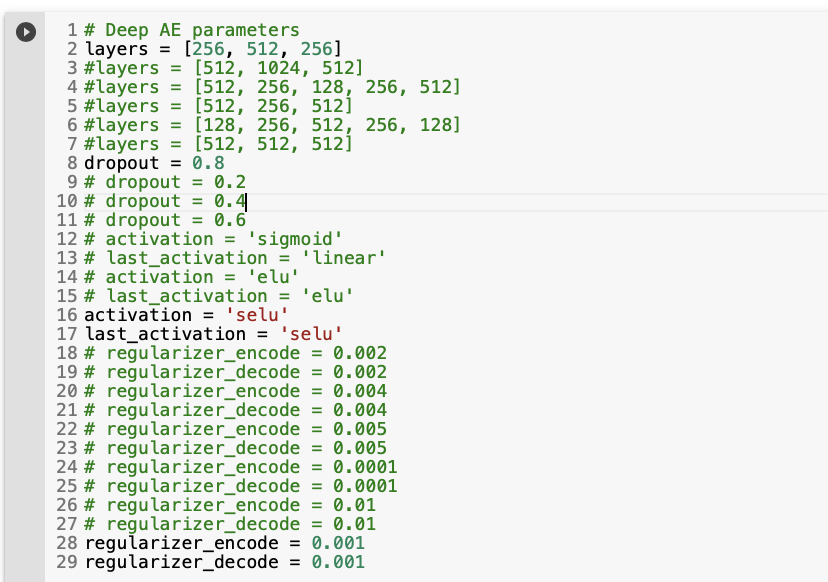
Autorec parameters tried

We also tested AutoRec with Leaky ReLU for the activation functions. In Keras, it is required that LeakyReLU be written as a standalone layer, we need to write a separate AutoRec\_LReLU\_model for AutoRec with Leaky ReLU.



*AutoRec with Leaky ReLU*

Please see the screenshot below for the set of parameters that we tested with Deep AutoRec



*Deep AE parameters attempted*

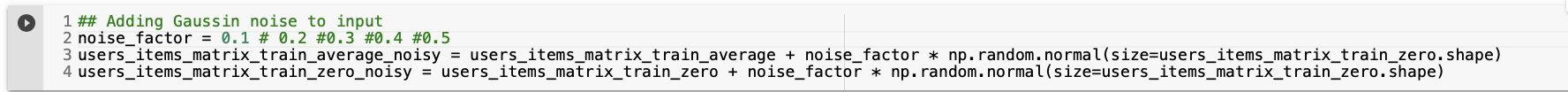
Some behaviors we observed was that having 1 encoder, 1 latent layer, and 1 decoder gives us the best performance. The number of neurons in each layer following the 2 configurations [512, 256, 512] and [256, 512, 256] both gave similarly the best performance but since the configuration [256, 512, 256] 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 converges quicker but is 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.

A quick note that we focused on the activation functions with non-zero negative parts and boundless positive parts (selu, elu, and LeakyReLU) instead of the most (ReLU, sigmoid, linear). This is 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. ReLU was a bad fit for the autoencoder because reLU’s max transformation doesn’t have an inverse, which creates difficulties for the autoencoder. Sigmoid and linear were tested because these functions had been previously implemented in Sedhain’s AutoRec paper, so we re-implemented them to compare and evaluate results with new activation functions added.

To explore options to enhance our models’ performance further, we implemented some ideas of our own. First, 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. Since the MovieLens datasets have many versions with different data points, we thought that this experiment was appropriate, given that these datasets all come from the same underlying distributions as well. 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 predictive features, and training the model on a specific demographic subset and predict for a user in that subset.

The first option was to add noise to the model to increase generalization capacity. 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, which given the fact that



*Adding Gaussian noise to input*

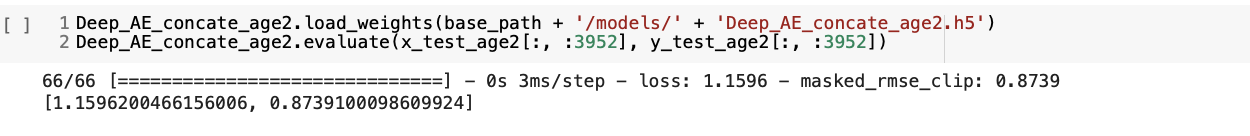
The second option was to give the model more demographic features of the users to predict. We did so by concatenating available demographic features in the dataset, which are gender, age and occupation, to the user-item matrix and applying one-hot encoding to this matrix. After one-hot encoding, we have an additional 30 demographic features, which when concatenated to our original user-item matrix give us a new user-item matrix with shape 6040 x 3982. We found that incorporating additional 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, which in our cases - age, gender, occupation info - are quite general, basic and doesn’t contain much indication of a user’s ratings for a movie. In short, 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. First, in the past, many companies use 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[[1]](#footnote-0). 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. For instance, features such as online interactions among users, online presence and status, user’s search terms, device types, etc [[2]](#footnote-1) are being increasingly collected, studied, and built into recommendation algorithms, which allow industry leaders, such as YouTube and Spotify to build outstanding recommendation algorithms. 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 (the organization that created the MovieLens dataset) 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”[[3]](#footnote-2) 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, which means our user-item matrix becomes a matrix with shape 1538 x 3892, but we will only use the rating features to train and predict the model on since the demographic information has already been accounted for, so we will index the matrices to have shape 1538 x 3592 to train and predict the model. 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 AE got 63% of his ratings correct, which is also lower than what Deep AE trained on the full 6040 x 3952 user-item matrix had (70%, and we will talk more about this in the next session). Please refer to the following screenshots for the results we got with this model and configuration.





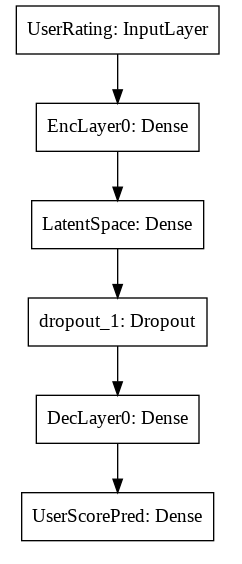
We also conducted a second experiment within this “demographic-based” approach to further explore the behavior of the model. We decided to only cluster user with the age of 18 - 24 years old to train and predict the model on, since doing so will give me a larger subset of users so there will be more data points (we got 2096 users compared to just 1538 users if we selected users that are male in the age of 18 - 24 years old). 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.



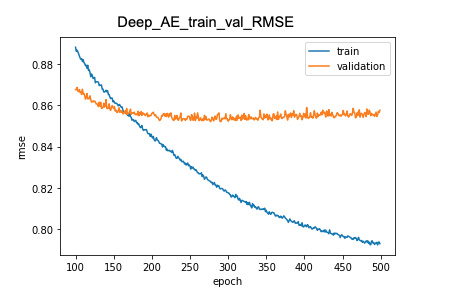


Though our “demographic-based recommender system” experiment didn’t seem to significantly bolster the performance of the autoencoders, we wouldn’t immediately disregard this approach. We are aware that the features available for the MovieLens datasets and our approaches air on the side of simpler, more basic approaches. Research has shown that for “demographic-based recommender system” [[4]](#footnote-3) to significantly bolster the performance of the recommender, extensive market research is needed.

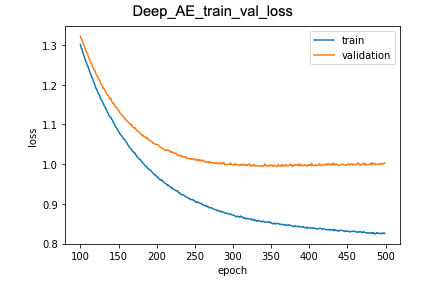
After various experiments and approaches, we’ve concluded that our best Autoencoder Model is the Deep AutoRec model trained on the full 6040 x 3952 training matrix with default rating set to 0 with the set of parameters: layers = [256, 512, 256], dropout = 0.8, activation and last\_activation = 'selu’, 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.



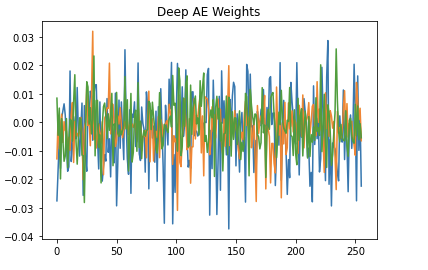
*Best Autoencoder: Deep AutoRec Model Structure*



*Train and Validation Epoch vs RMSE Graph for Deep AutoRec*



*Train and Validation Epoch vs Loss Graph for Deep AutoRec*



*Weights of Deep AutoRec*

**

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

**

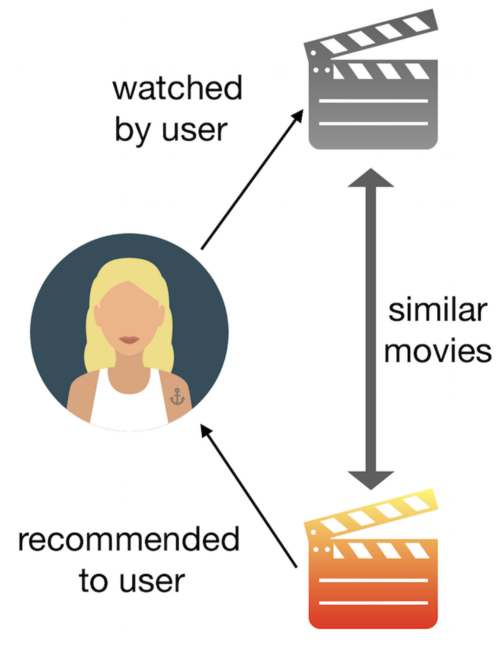
*Percentage of Ratings Correct of User with ID = 2000 for Deep AutoRec*



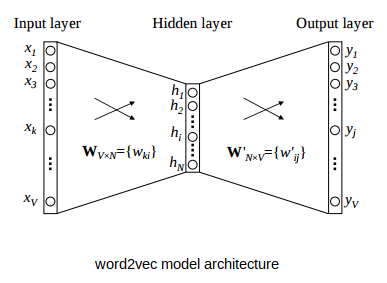
*Recommended Movies for User with ID = 2000 based on Predicted Ratings for Deep AutoRec*

**Word2Vec Content Based Filtering Model:**

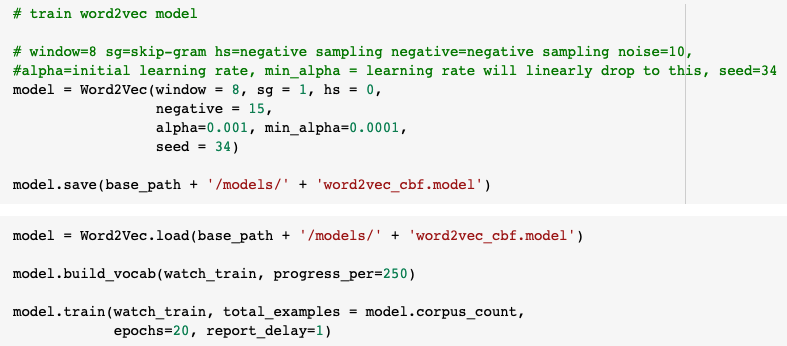
Word2Vec and content based filtering were attempted in hopes of enhancing our model. Word2vec models input Movies and translates them into vectors, and then trains on these movie embedding vectors. Content-based Filtering is based on the idea that users who previously had similar taste will also in the future. Our methodology was based on prior findings by a machine learning engineer and technical blogger [Sarang Desgmukh](https://github.com/SarangDeshmukh7). This approach works off of the principle based on users’ previous preferences to recommend movies.



*Word2Vec Content Based Filtering Concept*



When preprocessing data we merged the movies and ratings datasets together and dropped the missing values. We then randomly shuffled the users followed by splitting the data 90% train and 10% validation. Then we created word2vec embeddings and built a model which has a vocabulary of 3396 unique words with vectors of size 100 each.

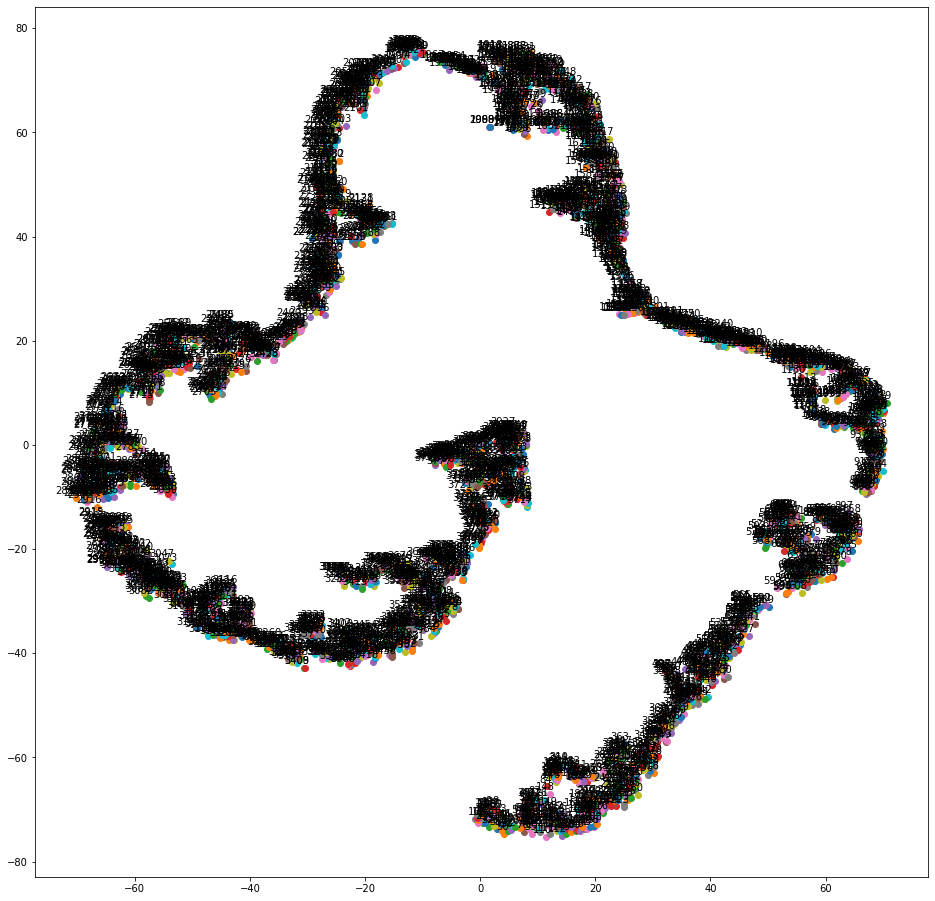


*Training set word2vec embeddings and specific parameters*

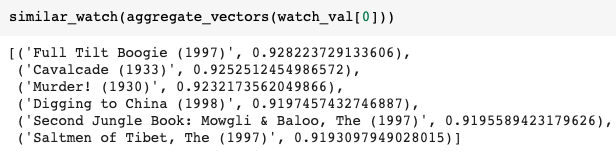
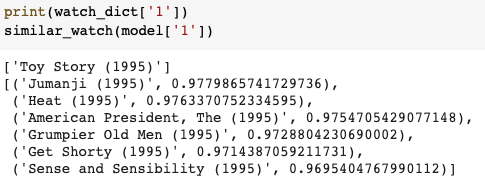
To better understand the data we generated visualizations of word2vec embeddings by reducing dimensions using t-SNE and UMAP. Since we cannot visualize these embeddings directly as the output was vectors of size 100, t-SNE and UMAP were leveraged to visualize these embeddings. Next, we pulled out the vectors of all the words in our vocabulary and created a dictionary of movie id and title. Lastly, 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.



*Visualize word2vec embeddings by reducing dimensions using UMAP*

**

*Visualize word2vec embeddings by reducing dimensions using t-SNE*

****

*Get recommendations outputs*

**Evaluation and Reflections:**

Reflecting on our results, we believe we could have boosted our performance by being able to use the larger movielens dataset such as the 10M and 20M movielens datasets. Ultimately we were only able to load in the 10M dataset for the transformer model. For the Autoencoder model this effort was unfortunately a failure because Google Colab repeatedly crashed and timed out during preprocessing. This project has been an interesting challenge to us as we have worked through understanding three very different complex deep learning applications. Our initial interest in exploring recommendation systems using MovieLens data has resulted in us starting from scratch to ending with building an impressive model(with the middle steps being a lot of self-learning, research, and trial and errors). Looking back on the work that we have completed we are proud of ourselves for our hand on approach in learning about recommendation systems, when we barely just covered it in class. Our successful collaboration has allowed the four of us to work well together. While there is always room for improvement, we are pleased with our progress towards generating movie recommendations, and our (deep) learning along the way.

**References:**

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

<https://miguelgfierro.com/blog/2018/introduction-to-recommendation-systems-with-deep-autoencoders/>

<https://www.analyticsvidhya.com/blog/2019/07/how-to-build-recommendation-system-word2vec-python/>

<https://github.com/sravanroy/RecommendationSystem-Word2vec/blob/master/word2vec.ipynb>

<https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing-25ad400eb243>

<https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>

<https://github.com/RaptorMai>

1. https://www.amazon.science/the-history-of-amazons-recommendation-algorithm [↑](#footnote-ref-0)
2. https://emerj.com/ai-sector-overviews/use-cases-recommendation-systems/ [↑](#footnote-ref-1)
3. https://emerj.com/ai-sector-overviews/use-cases-recommendation-systems/ [↑](#footnote-ref-2)
4. https://emerj.com/ai-sector-overviews/use-cases-recommendation-systems/ [↑](#footnote-ref-3)