# Informative Transformer Encoder for Movie Recommendation Task

**Tor Nitayanont** TOR_N@BERKELEY.EDU
*Department of Industrial Engineering and Operations Research*

## Abstract

In this work, we use the transformer encoder in a movie recommendation task. While common approaches to the recommendation task is to either rely on matrix factorization techniques or information from the movie sequence, the main idea of this work is to design an incorporation of heterogeneous features of users and movies into the model. Results from the experiments reveal improvement that these side information brings in, however, to a lesser degree than expected. We also provide comparison between the use of cross entropy loss and the binary cross entropy loss.

## 1. Introduction

The use of deep learning in recommender systems is considered relatively new compared to areas like computer vision and speech. This is because traditional methods like collaborative filtering alone can deliver decent performance of the models. In the early stage of deep leaning in the recommender field, they failed to enhance the performance to a significant extent. After several years, many deep learning approaches have progressed, and they have been integrated with traditional methods of recommender systems smoothly.

## 2. Preliminaries

An article on the use of deep learning in the recommender systems of Netflix by Steck et al. (2021) outlined several works on recommender systems that leveraged the capability of deep learning tools. Among them are the use of autoencoders to encode heterogenous information about users and movies, and the use of sequential models to handle the sequential nature of the movies viewing of users. There are several works such as the works developed by Kiran et al. (2020) and Wei et al. (2017) that utilized autoencoders, specifically the Stacked Denoising Autoencoders, to learn useful representations before passing on the encoded data to an optimization model like collaborative filtering or one of its equivalents that are based on matrix factorization. In these works, the point about the sequential nature of the problem was not addressed.

In another work on the use of sequential model where the author developed the Hierarchical Recurrent Neural Network based on the session-based RNN, the author also noted that one possible direction is to use the users and items features (Quadrana et al., 2017).

Deep neural networks for youtube recommendations by Covington et al. (2016) incorporates a number of side information about the users and the videos as well as the time duration that each user spent on each video. However, this work lacks the sequential aspect as the embedding vectors of the videos are averaged into a single vector before being fed into the network.

In this work, we would like to use a model that does both aspects: take in the side information or the heterogeneous features to be encoded as well as capturing some sequential behaviors of the users. An important aspect of our work would be the method to construct the input based on the information from both sides.

## 3. Problem Formulation and Loss Function

Each input to the model corresponds to a watch history of a particular user. It contains information about this particular user, $m = 10$ movies that this user has watched consecutively, IDs of these 10 movies, and the ratings that this user has given to each of these 10 movies. Our goal is to get a list of $k = 5$ movies that this user is likely going to watch next. The output from our model is a tensor of length equal to the number of movies that we have in total, which is 1000 in our case.

From the dataset, we can obtain the target tensor, which is the actual list of movies that the user actually watched after. We train the model on two different losses, which then results in two different models that we will evaluate in the experiment.

The first loss is the Cross Entropy Loss. We frame the problem as a multi-class classification problem. The target is only the next movie that the user watches after the given 10 movies. Although we only minimize the loss with respect to only one next movie, when we evaluate the model, we look at the five movies that have highest predicted probability (from softmax) and compare them to the actual five movies that are watched.

In the second scenario, we treat our problem as a multi-label classification problem, with the Binary Cross Entropy Loss. The difference from the first scenario is that the target tensor now involves all of the five movies that the user will watch. Suppose the IDs of the movies that this user watched after the sequence of the given 10 movies are $i_1, i_2, i_3, i_4$ and $i_5$, then the target tensor is a zero tensor of length 1000 with elements at the indices $i_1, i_2, i_3, i_4$ and $i_5$ being ones. Instead of using the softmax function in the final layer, the sigmoid function is applied instead.

Note that we look at the next five movies that the user will watch rather than just one next movie to make the difficulty of the task more realistic. In reality, people have multiple movies that they would like to watch in mind, and sometimes, the order can be arbitrary. This is also why we do not consider the order of the five movies in the prediction and frame the problem as a classification problem rather than a sequence prediction task. The evaluation metrics, or the accuracy, will be discussed in the experimentation part.

Regarding future works, there are many more features about users and movies that can be incorporated such as directors of the movies, languages, titles. The training and the target sequences can cover a longer time horizon to make the task resemble the reality more closely. We can also experiment with other model architectures.

## 4. Informative Transformer Encoder

The idea of the Informative Transformer Encoder model is motivated by the availability of the side information of users and movies. The input to the model includes these information, as well as information from user-movie pairs like ratings. The network architecture that we use is the transformer encoder. We discuss below why we select this network architecture.

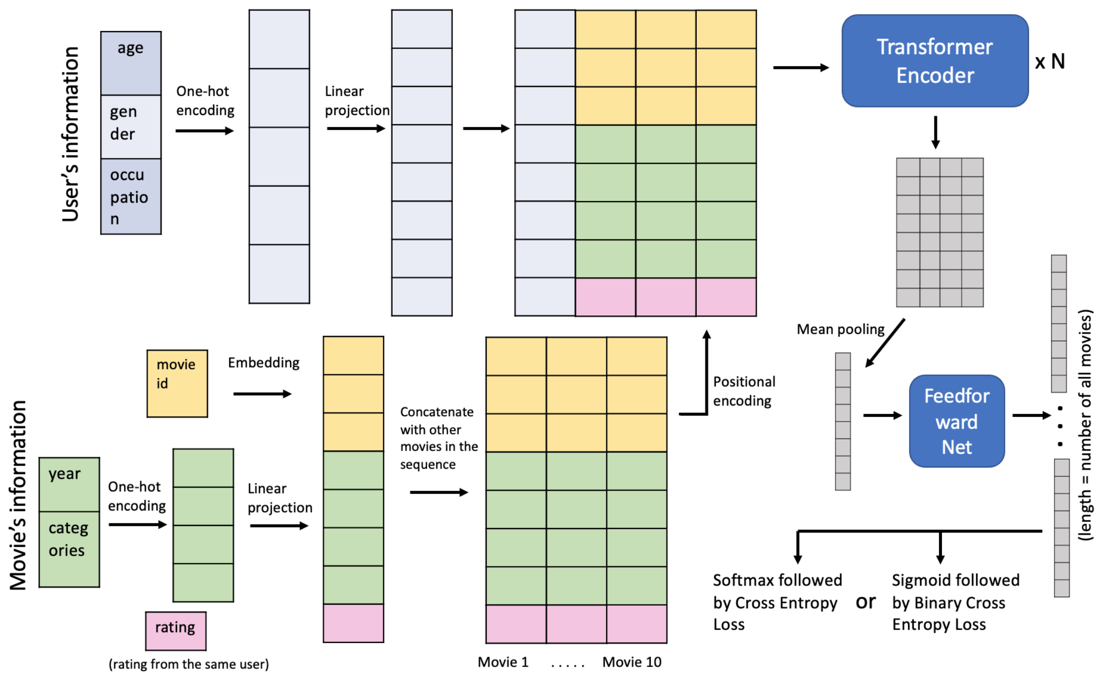


Figure 1: Construction of the input tensor and the architecture of the Informative Transformer Encoder

Before the information is fed into the encoder, we construct input tensors as follow. First, we apply one-hot encoding to the categorical features of users and movies. The resulting tensors are then linearly projected. For each movie, we obtain the embedding tensor from its ID. The linear projection of movie feature tensors are stacked on top of the movie embedding tensors. For each user, we have a sequence of $M$ movies that the user has watched. The linear projections of these $M$ movies are then concatenated together, followed by positional encoding to factor in the sequentiality of the watch history. The user feature tensor is then prepended to this stack of movie tensors.

Last but not least, one last piece of information that we utilize is the movie rating given by this user. Ratings have always been one of the most used information, especially by the matrix factorization approaches. Although rating is a value of a user-movie pair, we can treat it as one of the movie features since each tensor input is built for each individual user. The rating of each movie is concatenated after the linear projection of movie features before the the user tensor is prepended.

The constructed tensor is fed to a transformer encoder, which consist of $N$ stacks of a self-attention layer and a feed-forward net. There are several reasons that we choose the transformer encoder architecture. First, the self-attention layer learns the dependence between different positions in the sequence. Often, we watch movies based on what we have seen. Another interesting behavior is to also watch movies based on what we will watch later in the future. Hence, it is important that we choose a model that is capable of learning this relationship between different positions in the sequence.

Second, with transformer, or the self-attention mechanism in particular, we have access to all movies in the sequence simultaneously regardless of whether the movie was watched very recently or several steps back. When watching movies, the role that sequentiality plays is not as evident as, for instance, in natural language processing. Thus, transformer might be a more appropriate choice than RNN or LSTM. However, this is not to say that the watching order has no significance. The transformer can still learn that due to how we concatenate movie tensors in order. It is also aided by the use of positional encoding.

Another reason that we use the transformer architecture is that it allows us to incorporate side information about the users and movies conveniently. Also, we only use transformer encoder layers rather than the entire transformer, which contains both encoders and decoders, since it suits better with our classification formulation.

After the transformer encoder, we apply mean pooling to the output from the encoder. This reduces the tensor to one dimension, where the length is equal to the embedding dimension of the encoder. Then, the mean tensor goes through one last layer of a feedforward network. The output is a tensor whose length is equal to the number of movies in our movie dictionary. Finally, we apply another function on the tensor depending on the loss function on which we train the model. If we are minimizing the cross entropy loss, we apply the softmax function. With the binary cross entropy loss, we apply the sigmoid function.

## 5. Experiment

### 5.1 Datasets

The dataset that we use is the MovieLens 1M Dataset (Harper and Konstan, 2015). The dataset consists of 1) users' information: IDs, genders, ages and occupations 2) movies' information: titles, years, categories (each movie may belong to multiple categories) and 3) ratings that the users gave to each movie that they watch along with the timestamp, which is useful for obtaining the sequence of movies that each user watched. There are 6040 users, 3883 movies and 1000209 ratings. However, among these 3883 movies, some of them are only watched by a few users. We only work on 1000 most-watched movies and filter out other movies. We also filter out users that have watched too few movies by keeping only those who watched at least 15 movies since we need 10 movies in the training sequence and five in the target, as mentioned in Section 3. The two filtering steps result in a smaller dataset that has 5958 users, 1000 movies and 747869 ratings.

We obtain training sequences by partitioning the movie sequence of each user into subsequences of length 10. The corresponding targets are constructed, as explained in Section 3, using 5 subsequent movies. As a result, we obtain 69028 sequences of 10 movies, of which randomly sampled 90% make up the training set while the rest is the validation set.

### 5.2 Models

We build models based on the Informative Transformer Encoder architecture illustrated in Section 4. We experiment with some variations of the model according to the input to the model as well as the loss function. The two loss functions are the cross-entropy loss function and the binary cross-entropy loss function. Regarding the inputs, we vary the level of side information included. First, we train the model using side information of both users and movies. Since we would like to see how involving more information into the model would help boost prediction performance, we also train the model using only the information about users, only the information about movies, and finally, with no side information rather than a sequence of movies IDs.

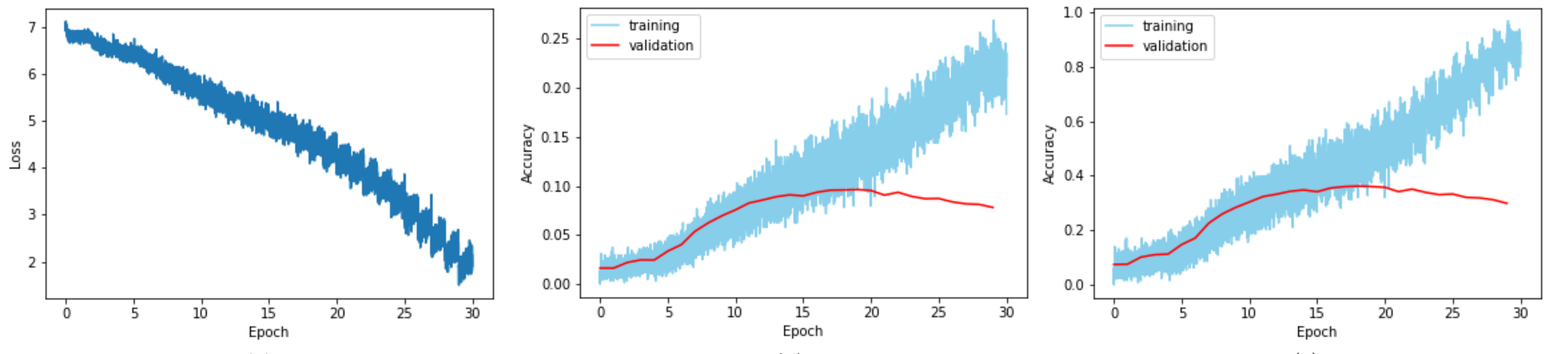
### 5.3 Evaluations

**Accuracy 1**: For each sample, we count the overlap of the five predicted movies and the actual five watched movies. For instance, when the predicted list is $[12, 437, 1, 82, 554]$ and the target list is $[23, 48, 911, 12, 82]$, the accuracy score is $\frac{2}{5}$.

**Accuracy 2**: For each sample, we only check if there is an overlap between the predicted movies and the watched movies. If there is, the accuracy is 1. If the two lists do not overlap at all, the accuracy is 0. For the example given above, the accuracy score is 1.

### 5.4 Results

First, we use the input that includes information about both users and movies. We build two models based on the cross-entropy loss (CE) and the binary cross-entropy loss (BCE).

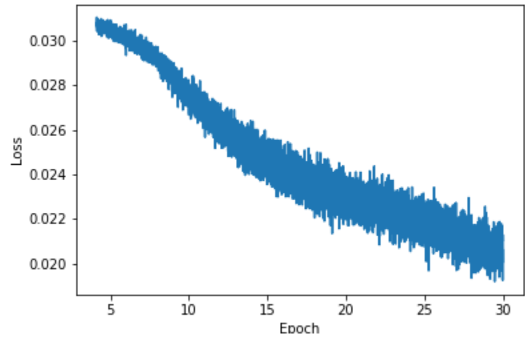


(a) (b) (c)

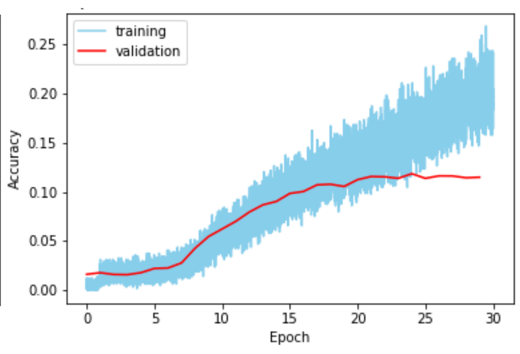
Figure 2: Performance when the model is trained on the CE loss and side information about users and movies are used. (a) Training Loss (b) Accuracy 1 and (c) Accuracy 2

Figure 2 displays the results of the model that is trained on the CE loss using side information from users and movies. After 30 epochs, the training accuracy of both types as well as the training loss still continue to improve. However, the validation accuracy of both types already start to decline. The highest of Accuracy 1, which counts how much in each predicted list overlaps with the actual watch list, that the model achieves is 9.63%. Accuracy 2, which is the fraction of predicted sequences that overlap with at least one movie in the actual watch sequences, reaches the highest value at 36.23%.

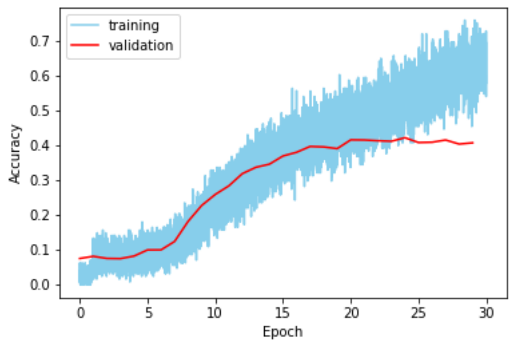
We are able to obtain better results when we train the model, using the same information, on the BCE loss, as shown in Figure 3. The best Accuracy 1 and Accuracy 2 are 11.85% and 42.28%. These accuracies are higher than those obtained by the CE loss by 23% and 17%. Similar to the CE loss, the model trained on the BCE loss also reach the highest validation accuracy after around 15 epochs before declining whereas the training accuracy and the training loss continue to improve. Note that for the BCE loss figure, we only show the loss after epoch 5 because the loss was much larger before drastically dropped to the magnitude in the plot after epoch 5. If the loss before that is to be included, the change in the loss after epoch 5 will be difficult to see on that scale.



(a)



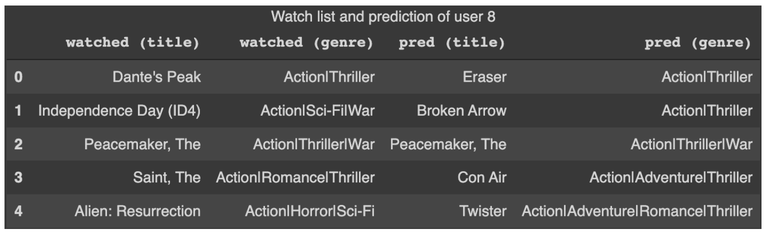
(b)



(c)

Figure 3: Performance when the model is trained on the BCE loss and side information about users and movies are used. (a) Training Loss (b) Accuracy 1 and (c) Accuracy 2

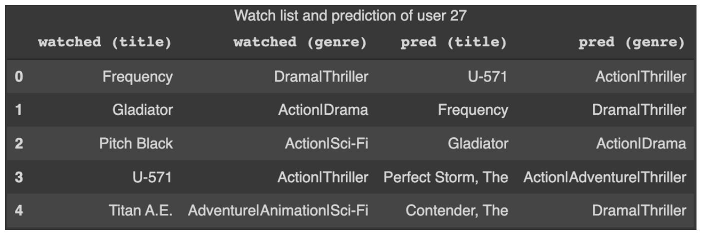
Since the model trained on the BCE loss yields better results, we only continue the experiment on models that are trained on the BCE loss for the brevity of the report. First, we look further into the prediction of the model trained on the BCE loss of three random users, displayed in Figure 4.



Watch list and prediction of user 8

| | watched (title) | watched (genre) | pred (title) | pred (genre) |
|---|---|---|---|---|
| 0 | Dante's Peak | Action\|Thriller | Eraser | Action\|Thriller |
| 1 | Independence Day (ID4) | Action\|Sci-Fi\|War | Broken Arrow | Action\|Thriller |
| 2 | Peacemaker, The | Action\|Thriller\|War | Peacemaker, The | Action\|Thriller\|War |
| 3 | Saint, The | Action\|Romance\|Thriller | Con Air | Action\|Adventure\|Thriller |
| 4 | Alien: Resurrection | Action\|Horror\|Sci-Fi | Twister | Action\|Adventure\|Romance\|Thriller |

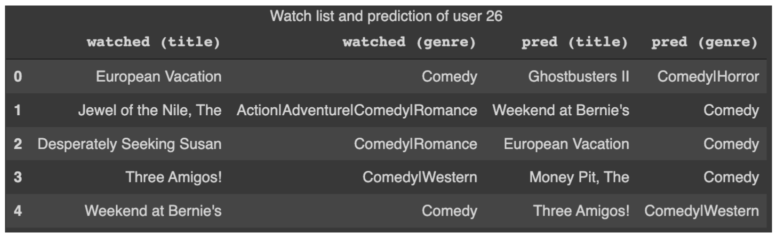
(a)



Watch list and prediction of user 27

| | watched (title) | watched (genre) | pred (title) | pred (genre) |
|---|---|---|---|---|
| 0 | Frequency | Drama\|Thriller | U-571 | Action\|Thriller |
| 1 | Gladiator | Action\|Drama | Frequency | Drama\|Thriller |
| 2 | Pitch Black | Action\|Sci-Fi | Gladiator | Action\|Drama |
| 3 | U-571 | Action\|Thriller | Perfect Storm, The | Action\|Adventure\|Thriller |
| 4 | Titan A.E. | Adventure\|Animation\|Sci-Fi | Contender, The | Drama\|Thriller |

(b)



Watch list and prediction of user 26

| | watched (title) | watched (genre) | pred (title) | pred (genre) |
|---|---|---|---|---|
| 0 | European Vacation | Comedy | Ghostbusters II | Comedy\|Horror |
| 1 | Jewel of the Nile, The | Action\|Adventure\|Comedy\|Romance | Weekend at Bernie's | Comedy |
| 2 | Desperately Seeking Susan | Comedy\|Romance | European Vacation | Comedy |
| 3 | Three Amigos! | Comedy\|Western | Money Pit, The | Comedy |
| 4 | Weekend at Bernie's | Comedy | Three Amigos! | Comedy\|Western |

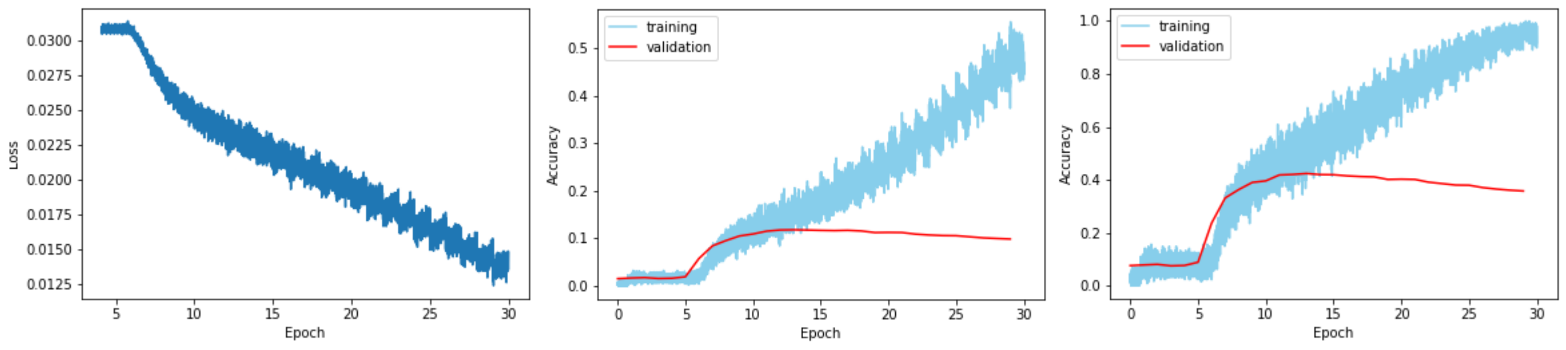
(c)

Figure 4: Predicted movies of three users given by the model trained on the BCE loss using side information of both users and movies

We can see that the model is able to predict correctly one movie for the first user and three out of five movies for the second and the third users. It is important to note that predicting only five movies and hoping that they would be mostly correct is quite unrealistic. On many streaming platforms, their recommendations include many more movies than this. Also, the order of movies that users watch depend on so many factors. In practice, these predicted five movies might not be watched immediately after the training sequence but could be watched a little later or get added to the saved list, for instance. Hence, getting some of the five predicted movies correct already reflects the capability of deep learning in a challenging task in the movie recommendation field.

Moreover, we can see that the model recommends movies of genres that are very similar to genres of those movies that are watched by the users. This demonstrates decent capability of the model to learn about users from their information and watch history. It is important to note that the user IDs are not part of the information given to the model, unlike movies whose IDs are embedded in the model.

Next, we would like to see how the models perform when side information is missing. We train the model on user information, on movie information, and with no side information. In these experiments, we maintain the input shape to the encoder. Hence, after movie information is removed, the dimension of the movie IDs embedding is adjusted accordingly.

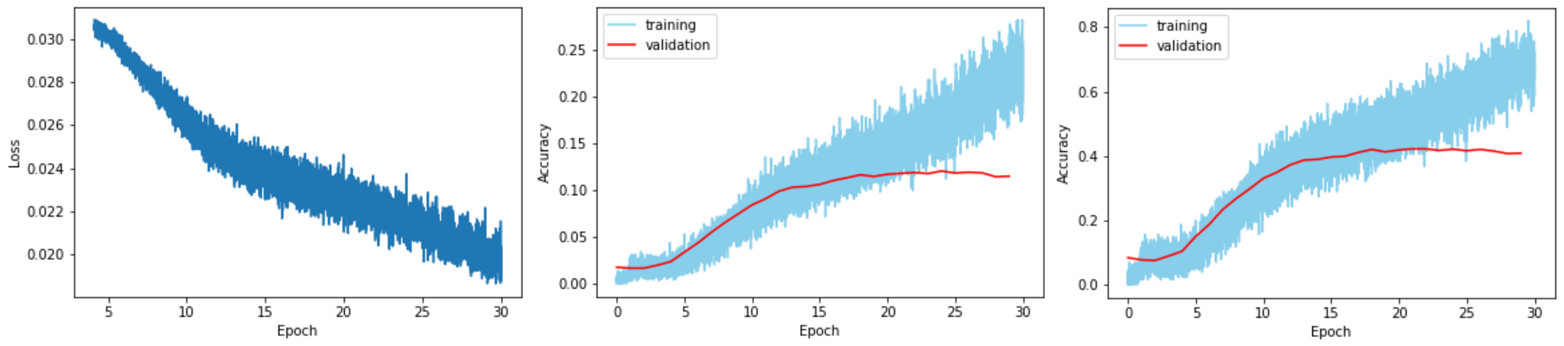


(a) (b) (c)

Figure 5: Performance when the model is trained on the BCE loss and only side information about users is used. (a) Training Loss (b) Accuracy 1 and (c) Accuracy 2

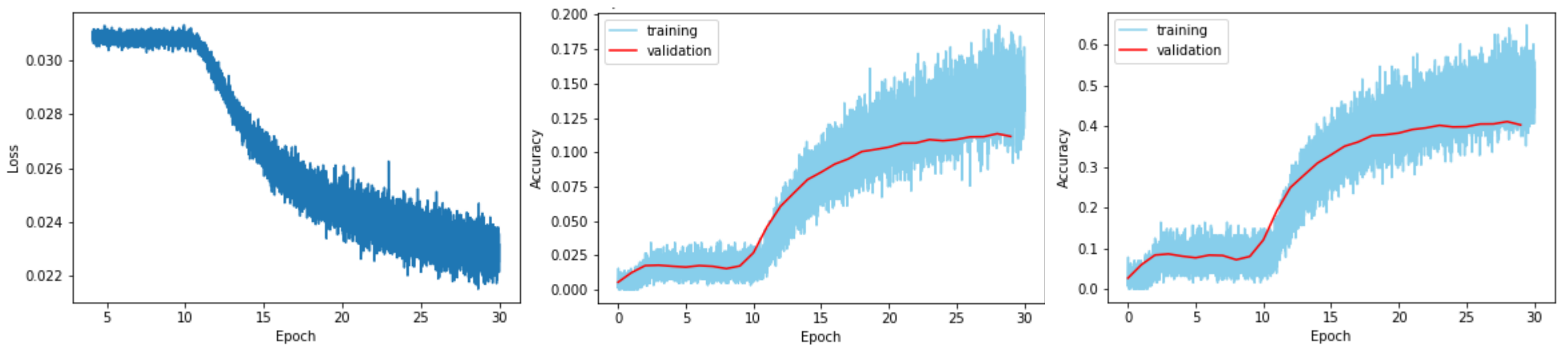
From Figure 5 and 6, we see that the performances of the model when the information from either the user side or the movie side is held back are very similar to when both information are available. With only user information, the highest Accuracy 1 and Accuracy 2 are 11.83% and 42.26%. With only movie information, the highest Accuracy 1 and Accuracy 2 are 11.86% and 42.28%. However, when we omit information from both sides (Figure 7), both accuracies drop slightly to 11.35% and 41.12%. The consistency among the first three experiments (Figure 3, 5 and 6) implies that information from one side, together with the movie sequence, is sufficient to imply information about the other side.

Although we see a slight decline in the performance of the model when no side information is given at all, it is still surprising that without any features of the users and movies, the model is able to achieve accuracy in the same range as when all information is available.



(a) (b) (c)

Figure 6: Performance when the model is trained on the BCE loss and only side information about movies is used. (a) Training Loss (b) Accuracy 1 and (c) Accuracy 2



(a) (b) (c)

Figure 7: Performance when the model is trained on the BCE loss and no side information is used. (a) Training Loss (b) Accuracy 1 and (c) Accuracy 2

## 6. Conclusion

In this work, we use a transformer encoder architecture in a movie recommendation task, which is framed as a multi-class classification and a multi-label classification problem. We show how to incorporate side information about users and movies into the input in addition to the movie sequence. Additional layers such as the mean pooling and the feed forward net are included after the transformer encoder. With the evaluation metrics that we use, the binary cross entropy loss appears more appropriate than the cross entropy loss, which is reasonable since we evaluate the models on more than one target movie. Our experiments also show that the use of side information provides benefit. However, the improvement is not as noticeable as expected. First, it could be implied that the side information is not very helpful, or it is not incorporated appropriately enough. However, it could also be implied that the transformer encoder is able to encode information about the users and movies by relying solely on the movie sequence, and hence, it is able to achieve decent accuracy.

The results also reveal that using information from only one side (either the user side or the movie side) yield accuracies that are as high as when information from both sides are leveraged.

## References

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

R Kiran, Pradeep Kumar, and Bharat Bhasker. Dnnrec: A novel deep learning based hybrid recommender system. *Expert Systems with Applications*, 144:113054, 2020.

Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 130–137, 2017.

Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. Deep learning for recommender systems: A netflix case study. *AI Magazine*, 42 (3):7–18, 2021.

Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 69:29–39, 2017.