Recommendation Systems 3693 Final Project Report

by Gil-ad Katz and Loten Noy

Based on: *Deep Neural Networks for YouTube Recommendations .2016*

Fall 2023

Abstract:

As part of RecSys 3693 course at RU, we chose to discuss and implement the paper of *Deep Neural Networks for YouTube Recommendations*. This paper concerns the YouTube recommendation system, considered one of the world's largest and most sophisticated industrial recommendation systems. It is structured around the two-stage modeling approach, detailing the deep Candidate Generation and deep Ranking models. Our work focused on the preliminary **Candidate Generation** model, considered a more innovative and groundbreaking practice in recommendation systems. Our enhancement of the originally proposed Candidate Generation model, using the Transformer Encoder layer to improve context embedding of the sequenced input features, achieved significantly better results than the original YouTube proposed model and surpassed an additional baseline model. Finally, we elaborate on future areas we believe should be further explored using the proposed mechanism over YouTube's data and original architecture.

1. Introduction:

1.1 Anchor Paper

Deep Neural Networks for YouTube Recommendations focuses on the impact of deep learning on YouTube's video recommendation system. YouTube recommendations assist over a billion users in finding personalized content from a constantly growing pool of videos. Recommending YouTube videos poses three significant challenges: scale, freshness, and noise. The recommendation system must handle YouTube's massive user base and corpus with highly specialized distributed learning algorithms and efficient serving systems. It should also be responsive to newly uploaded content and the latest user actions while balancing new content with established videos. Historical user behavior on YouTube is difficult to predict due to sparsity and unobservable external factors, and noisy implicit feedback signals must be modeled. As a solution for the abovementioned problems, the paper highlights the importance of deep learning techniques in addressing these challenges in the YouTube recommendation system. The writers of the paper suggested an innovative two-stage architecture to improve the methods in practice of those days (Figure 1).

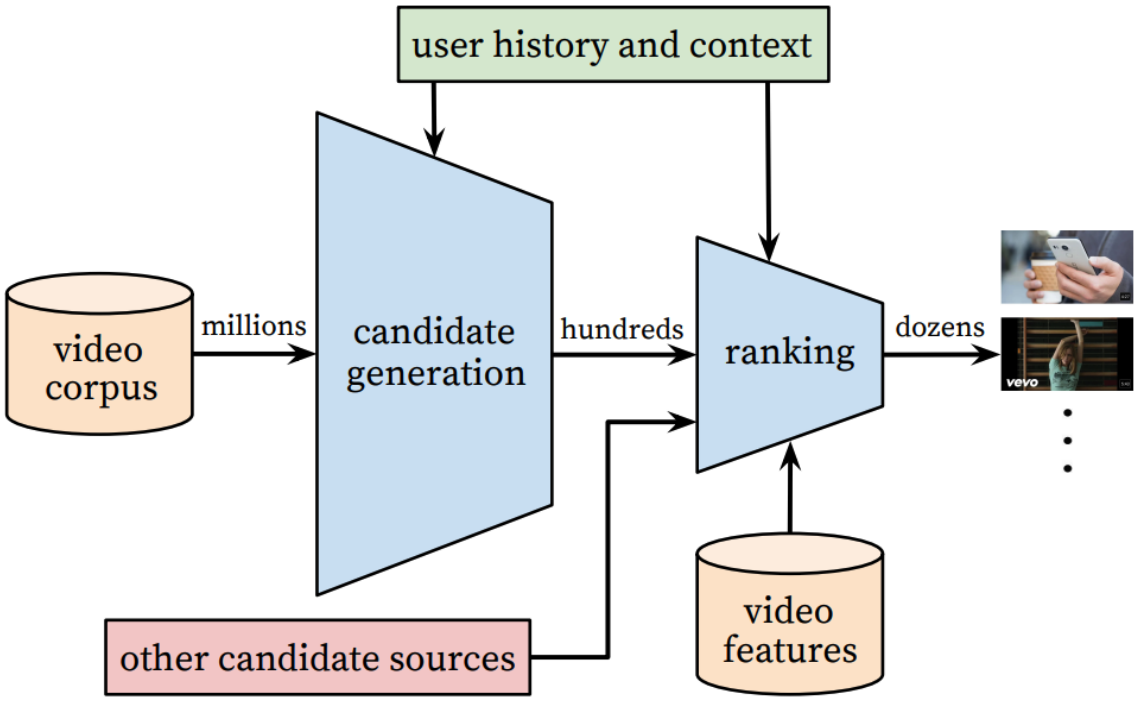

Figure 1: Original paper two-stage architecture. The *Candidate Generation* model filters the massive YouTube corpus into a few thousand recommendations per user. The *Ranking* model is then responsible for ranking those recommendations (and optionally additional out-sourced recommendations) based on user-item context.

1.2 Original Architecture

The YouTube recommendation system is made up of two neural networks: one for candidate generation and one for ranking. The Candidate Generation network (Figure 2) retrieves a subset of relevant videos from a large corpus based on the user's YouTube activity history. The ranking network (Figure 3) assigns a score to each video and presents the highest-scoring videos to the user. This two-stage approach allows recommendations to be made from a large corpus while

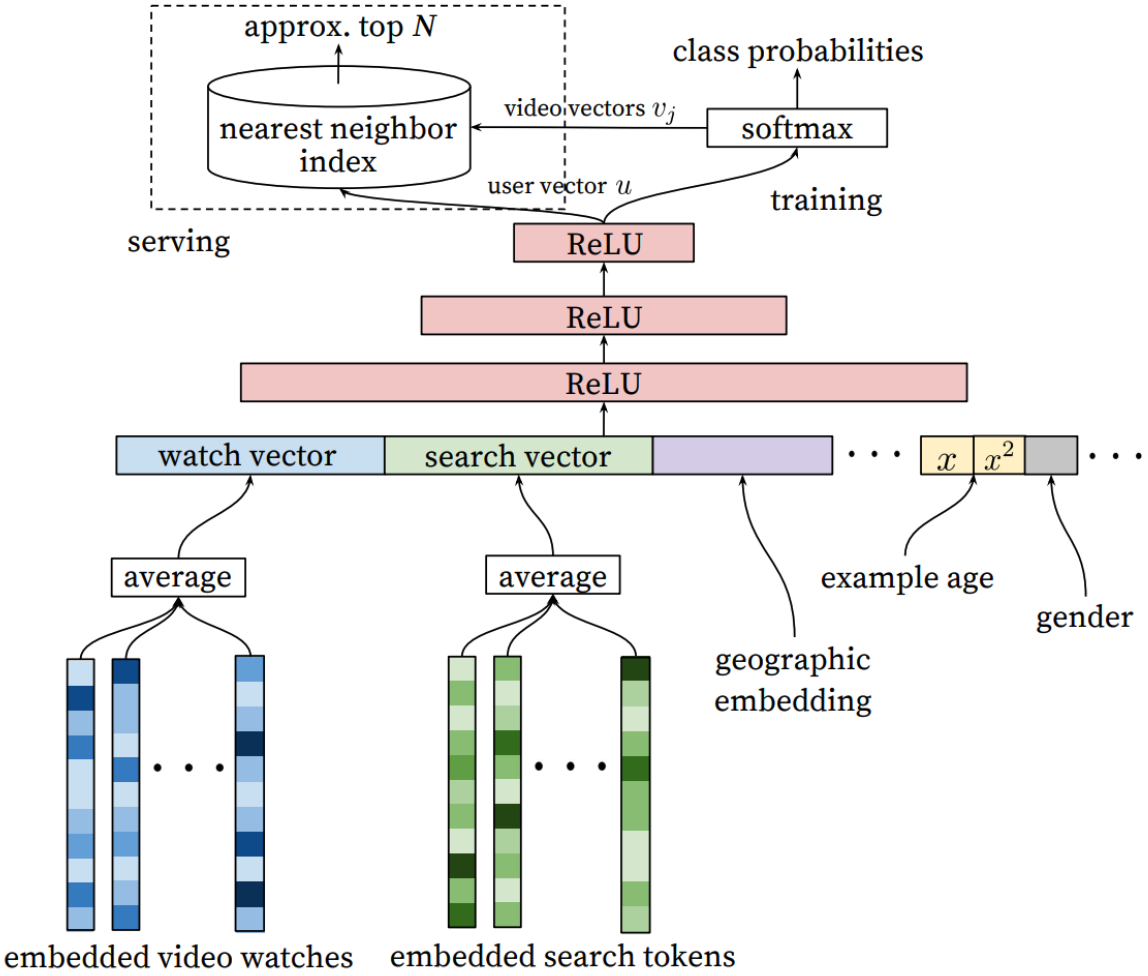

Figure 2: Original paper Candidate Generation model

still ensures personalization. Offline metrics are used for iterative improvements, but A/B testing is relied on for the final determination of the effectiveness of an algorithm or model, as live experiment results are not always correlated with offline experiments.

We chose to focus on the Candidate Generation model alone for the following reasons:

- Modeling the data, and the YouTube video items specifically, as a multi-class classification problem was an innovative idea at the time (and still is). YouTube has an immense corpus of videos, and the paper writers chose to produce the candidates by predicting the next watched video of a given time window. This approach was unique in our perspective; thus, we wanted to explore it furthermore.

- Available datasets: YouTube did not publish the datasets used in the original article, nor could they, since the data is too big and under PII commercial protection. Given that

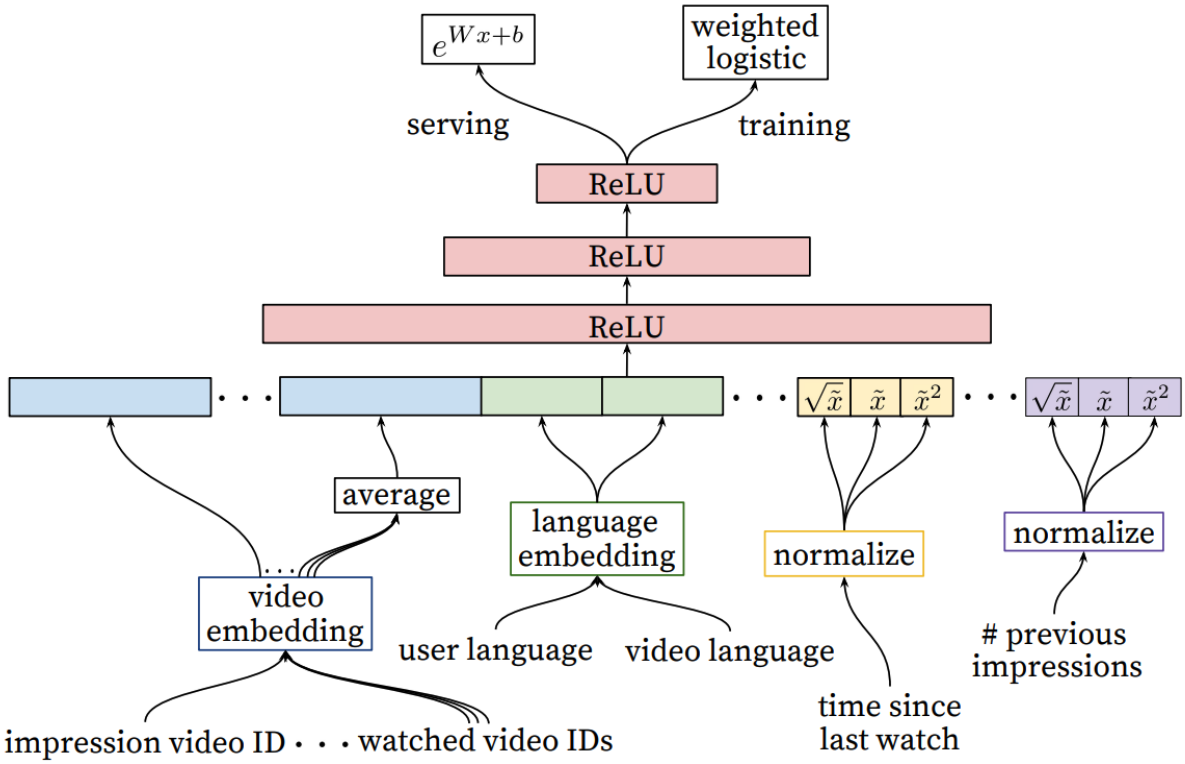

Figure 3: Original paper Ranking model

Impressions-based data used in the Ranking model was unavailable, we focused on collaborative filtering-based data used by the Candidate Generation model and available in numerous common open-sourced recommendations datasets.

2. Suggested Improvement

2.1 Transformer-Based Candidate Generation

One implementation decision in the original paper seemed odd at first glance. User history data in the Candidate Generation model was embedded and aggregated to form a single vector, representing the context of the given sample's timeframe window. However, this window is sequential, and averaging it might lose the information on the order of items within the sequence. I.e., videos at the beginning of 100-length window relevancy probably differ from those in the last

places. Moreover, items in the same window might have contextual connections with one another. I.e., a user binge-watching a series would probably result in a sequence where adjacent items are part of the same season.

Given that, we present an enhancement of the originally proposed Candidate Generation model. We use a multi-head self-attention mechanism encapsulated in a Transforer Encoder layer to extract the essence of context between the items in the same window and better represent the user's window history.

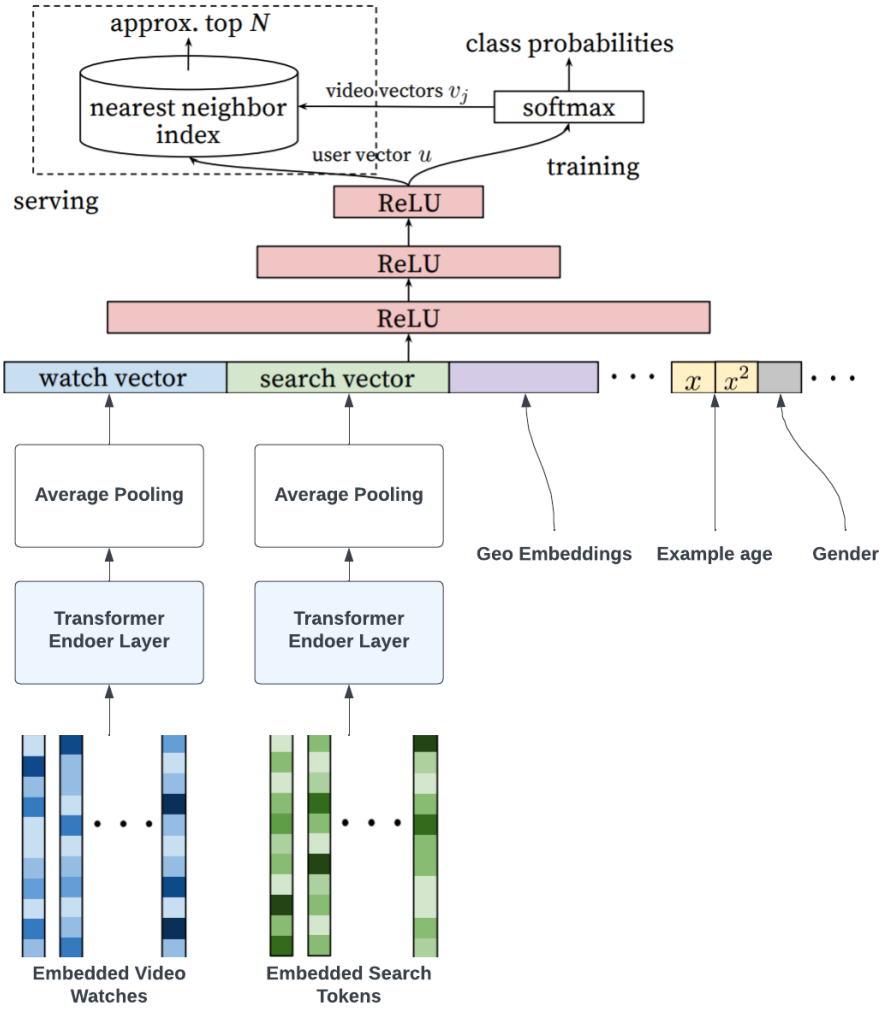

Figure 4: Our Transformer based Candidate Generation model proposal. In our proposal, the embeddings of the window's video watches and search tokens are first fed into a unique Trasnforemer Encoder layer, where multiple self-attention heads process them, capturing the dependencies between the embeddings and producing a sequence of contextualized embeddings. Then we max-pool the embeddings the generate a single window-history vector.

2.2 Attention and Self-Attention

Attention and self-attention mechanisms are critical components of deep learning models that have revolutionized the field of natural language processing (NLP). These mechanisms allow the model to selectively focus on relevant parts of the input, enabling it to make more informed predictions. The **attention** mechanism allows the model to attend to specific parts of the input sequence, providing a weighted sum of the inputs used as context for the subsequent layer. On the other hand, **self-attention** (Figure 5) allows the model to attend to different parts of the same input sequence, allowing it to capture dependencies between different sequence components. Self-attention is particularly effective in capturing long-range dependencies, making it a powerful tool for modeling sequences such as natural language text. These mechanisms have played a key role in many recent breakthroughs in NLP, including the development of transformer models such as BERT and GPT-3.

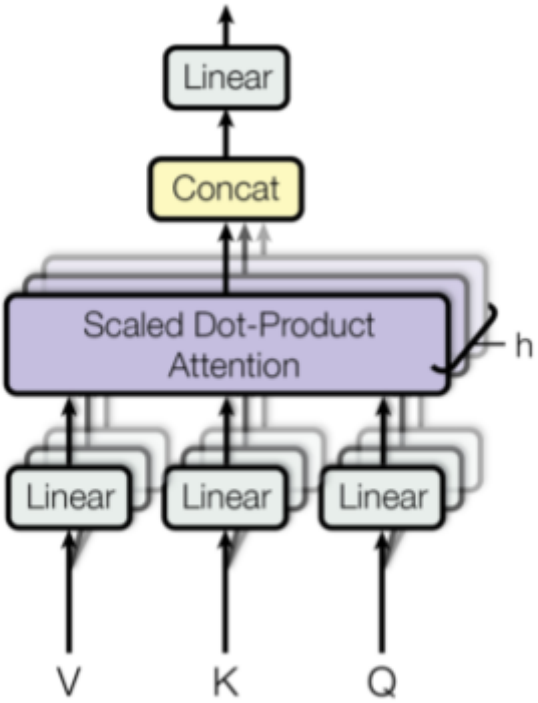

Figure 5: Self-Attention head architecture

2.3 Transformer Encoders

Transformers and Transformer encoders are based on the self-attention mechanism, which enables them to selectively attend to different parts of the input sequence to make more informed predictions. Transformer models consist of an **encoder-decoder** architecture, where the encoder is responsible for encoding the input sequence, and the decoder is responsible for generating the output sequence.

Transformer encoders use multiple stacked self-attention layers and feed-forward neural networks. Each encoder layer attends to the previous layer's output to capture dependencies between different parts of the input sequence.
The transformer architecture has been shown to outperform previous state-of-the-art models in various NLP tasks, including language modeling, machine translation, and question-answering.

3. Experiments

3.1 Dataset

We chose the MovieLens100K dataset to examine the quality and performance of our implementations of the original, baseline, and enhanced models.

It is a popular dataset often used in recommender system research and consists of user ratings on a scale of 1 to 5 for approximately 100,000 movie ratings from 943 users on 1,682 movies. The dataset includes additional information, such as movie titles, genres, and release year. The MovieLens 100k dataset is commonly used as a benchmark dataset for evaluating the performance of different recommender system algorithms.

Similarly to the original paper, we modeled the data by implicit rating, treating any user interaction with a movie as a positive rating, disregarding the actual rating given to a movie by a given user.

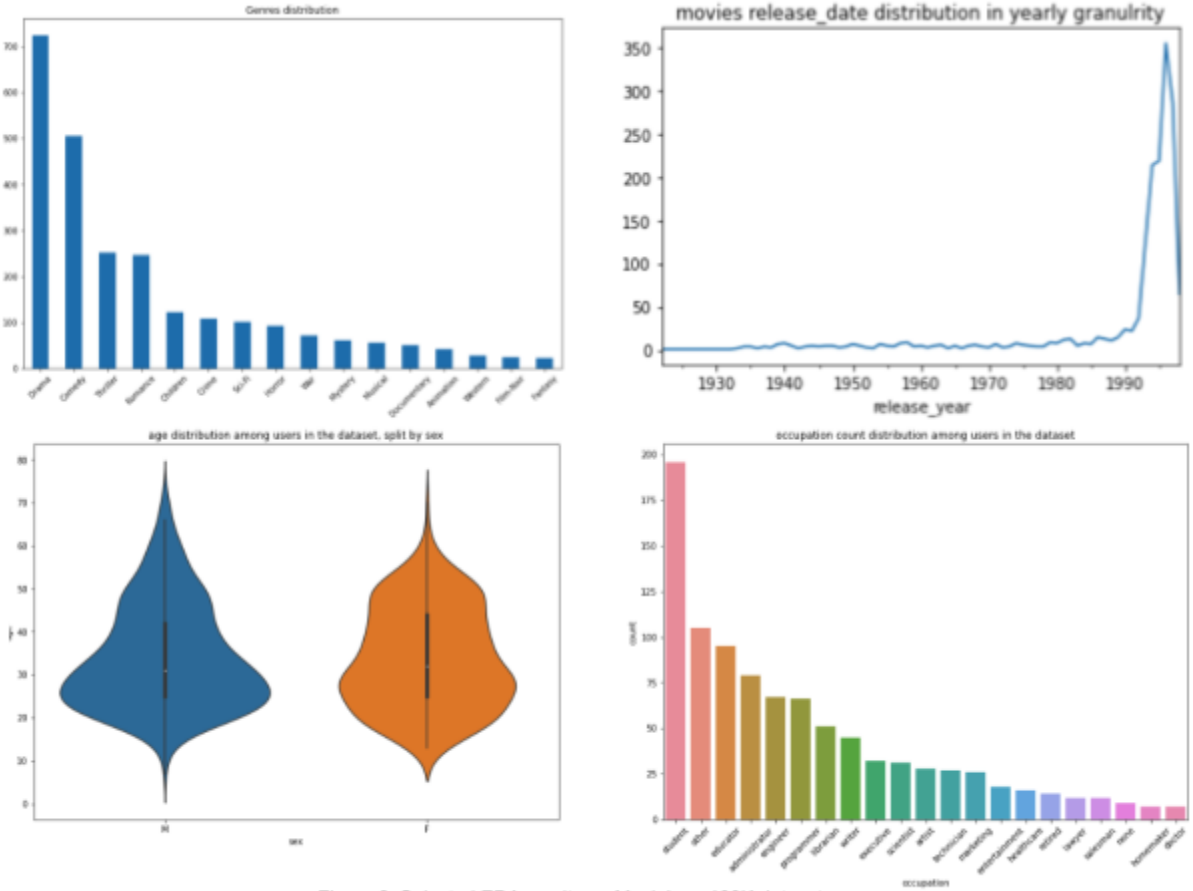

Figure 6: Selected EDA results on MovieLens100K dataset

3.2 Data Modeling and Preprocessing

We created a sample of item, user, and rating features based on the elaborative feature engineering work done in the original paper.

	user_id	positives	search	label	example_age	features	negatives
40623	180	[1337, 1356, 1328, 1348, 1331, 1319, 980, 812,...	[1, 5, 6, 7, 8, 14, 16, 17]	1287	-0.581000	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ...	[1609, 792, 18, 666, 972, 1499, 1233, 1238, 43...
8849	495	[1613, 355, 154, 1156, 1132, 93, 553, 1228, 10...	[1, 2, 3, 4, 5, 8, 9, 11, 12, 13, 14, 15, 16, 17]	276	-0.500483	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[1051, 558, 627, 992, 1081, 80, 1522, 1639, 11...
26196	415	[252, 627, 236, 224, 844, 472, 297, 272, 20, 2...	[1, 2, 3, 4, 5, 6, 8, 12, 14, 15, 16]	823	-0.592914	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[408, 401, 175, 1120, 608, 1528, 1466, 395, 16...

Tabel 1: examples dataframe generated based on MovieLens100K dataset

- Item features: we created a one-hot encoding map based on the movies genres to express the items' different content characteristics.
- User features: we created a one-hot encoding map of the user's occupation and gender. For geolocation, we used a clustering mechanism to group the given zipcodes provided within the dataset into five groups and encoded that categorically similar to the occupation feature. At last, we added the user age normalized form.
- Window features: we created three k-length window context lists per window.
    - K **Positive** videos (videos the user watched).
    - L (>>K) **negative** videos (sampled videos not in the window of the current user).
    - K **search tokens** are the textual representation embedding of the items' video categories (genres) in the current window.

  In addition, the k+1 latest video of this window is regarded as a "label," and its age was added to the features vector as the "example_age" feature.

Ultimately, we balanced the dataset by user id and label, ensuring all users and target videos have sufficient training examples to avoid unbalanced dataset biases.

3.3 Implementation Details

Our original Candidate Generation model Pytorch implementation takes in three types of input embeddings: positive items, negative items, search tokens, and an additional fixed-size context features vector. Each type of input embedding is first passed through an Embedding layer to obtain a dense representation, then aggregated using an EmbeddingsAggregatorLayer, which computes an

average of the embeddings. The aggregated embeddings are then normalized using an L2NormLayer.
The network is trained to predict the next watched video, as proposed in the original paper.
The normalized embeddings are concatenated with the features vector and passed through several fully connected layers with ReLU activations and batch normalization before being mapped to a final output of size n_items using a linear layer and a log softmax activation. Finally, the network outputs a n_items long normalized probabilities vector used to rank and select the top k candidates, sorted by probability score (unlike the original paper, where they used an additional KNN model to cache item-user similarities for efficiency purposes).
Our **baseline** model is a simple Popularity based deterministic model that emits the most popular items as candidates, ranked by popularity.
Our **Transformer-based** implementation performs a flow similar to the original one, with an additional stage of Transformer Encoding as described earlier, using four self-attention heads and a 2048 fully connected layer with a 0.1 dropout regularization rate a ReLu activation (standard settings). We initially implemented a positional encoding layer that produced worse results, so it was omitted from the results presented in the report.

3.4 Experiments setup & Hyper-Params

We used a multiclass Cross-Entropy Loss to measure the loss of the next video classification problem with an Adam optimizer using the following hyper-params:

Parameter	Value
EMBEDDING_DIMS	256
USER_DIM	64
LEARNING_RATE	1e-4
WEIGHT_DECAY	1e-5
FC_LAYERS	(2048, 1024, 512, 256)

Adjusted after multiple runs with different configurations:

- Fully connected dimension: tried all of the original paper suggestions starting from (512, 256) and adding 1024 and 2048 layer gradually until we received the best results.
- Embedding dimension ranging from 2^5 to 2^8
- Learning rate running from 1e-3 to 1e-5

We used 16-length history windows, with **100** negative examples each.
The dataset was split into train (0.75) and validation (0.25) sets.
We trained both models for **ten** epochs after observing significant degradation in the model's loss reduction at ~ epoch 10, where we observed substantial overfitting of the original paper model and validation loss increase.
We trained the models and analyzed the results using loss scores and MRR@k and Hit-rate@k ranking metrics to compare the results of the trained models on predicting the next video at the top k candidates per validation example.
We mostly used Google Colab and a GPU runtime to conduct the training experiments with different hyper-params configurations.

3.5 Results

In terms of loss, we observed to following results:

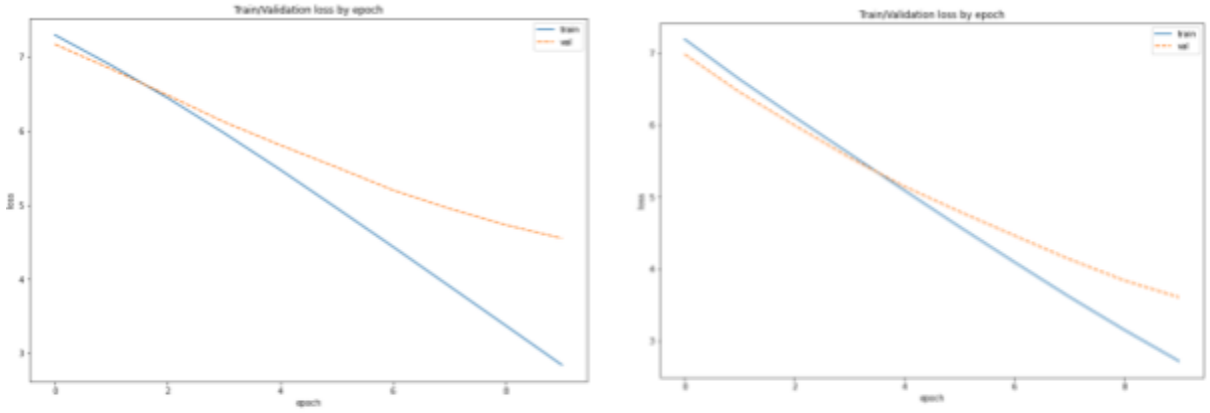

Figure 8: Left - Train/Validation loss of the **original paper model** over epochs. Right - Train/Validation loss of the **Transformer-based model** over epochs.

Overall, the Transformer-based model achieved better results in both train and validation loss reduction scores over epochs and a better fit for train-validation loss differences.
Model candidates' rank relevancy produced the following results:

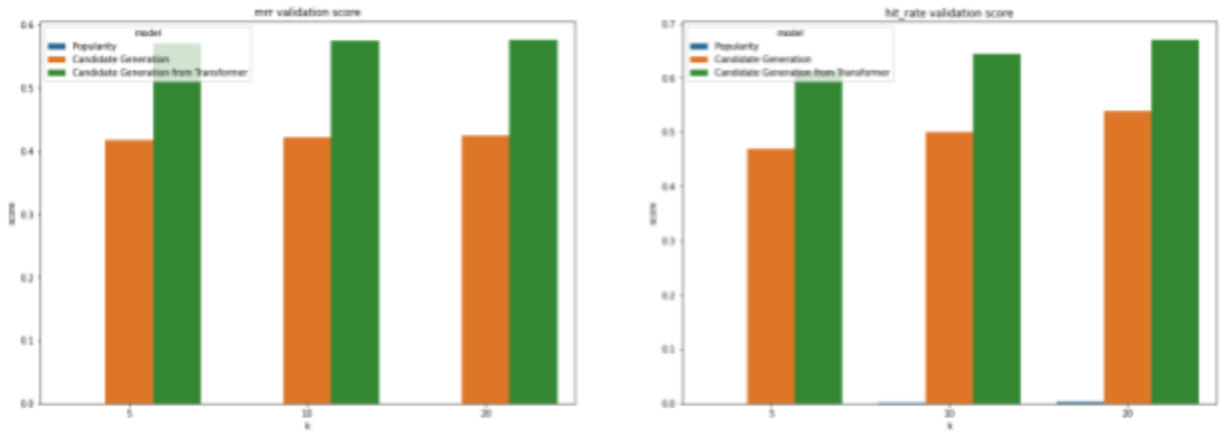

Figure 9: MRR@k and Hit-rate@k score for k = 5/10/20 split by model architecture

MRR:

#	Model	MRR@5	MRR@10	MRR@20
0	Candidate Generation	0.418440	0.422599	0.425248
6	Popularity	0.000000	0.000156	0.000245
12	Candidate Generation from Transformer	0.570987	0.575266	0.577130

Hit Rate:

#	Model	Hit Rate@5	Hit Rate@10	Hit Rate@20
1	Candidate Generation	0.469062	0.500469	0.539531
7	Popularity	0.000000	0.000937	0.002188
13	Candidate Generation from Transformer	0.612500	0.644062	0.671016

Table 2: HR and MRR results @k comparisons

In terms of rank based metrics, we see a significant improvement in comparison to the baseline model (Popularity) and to the originally proposed candidate generation model (Candidate Generation). Hit-rate@k numbers increased at ~ 0.15 compared to the original model, and MRR@k numbers increased similarly.

Additionally, we observed an increased training time for the Transformer-based candidate generation model, that increased in ~ 850% per epoch, probably due to the complexity of running four prallel attention heads per embedding.

4. Discussion

4.1 Interpreting the results

As suspected, an important signal was embedded in the context and inner connections of each window's history and item embeddings. We argue that not only the obvious sequenced lists (positive items, search tokens) benefited from the self-attention mechanism but also the non-sequenced list (negative items). Attention-based embeddings have been found to produce better results when applied to sequenced lists because they can capture the relationships between different items in the sequence. This is because attention mechanisms allow the model to selectively focus on specific parts of the sequence when processing it, giving it more context about the overall structure of the list. However, even non-sequenced lists can benefit from attention-based embeddings because the self-attention mechanism can enhance the representation embedding of each item by allowing the model to attend to the relationships between different items within the same list. By considering the dependencies between items in a list, self-attention mechanisms can create more informative embeddings that capture the underlying structure of the list, regardless of whether it is ordered or not.
This resulted in a much more robust representation of the window's context and improved the network's ability to generalize and perform the "next video" task more accurately.

4.2 Future enhancements

4.2.1 Experiment with more datasets

We used the MovieLens100K dataset due to restricted resources and the availability of a dataset more similar to the one used in the original paper. To do so, we hand-crafted some of the originally suggested features. One direction we believe should be explored is using the proposed method with more datasets that are closer, in essence, to the original YouTube dataset (implicit ratings, impressions data, etc.). This will also enable us to implement the *Ranking* model suggested in the original paper in a more straight-forward manner.

4.2.2 Automatic network-based feature extraction

The original paper suggested high load feature engineering processes that are less common in the deep learning community. Given the dataset presented in the paper, we would like to explore the possibilities of using some automatic tools/architectures to extract features. Such tools, even

attention-based ones, were unavailable when the original paper was published and should be explored in the context of the given problem.

4.2.3 Explore a fully Transformer-based solution

Our suggested solution uses Transformer encoding to represent better the embedding layers of the categorical items in the dataset. We suggest exploring the possibility of implementing a recommendation system that relies entirely on transformers for feature extraction and context understanding. Such a model would inject all input features into a Transformer encoder block (with multiple encoder layers) to produce a single embedding representation of the entire window. This would help discover additional connections between the items in the same window.

5. References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *ArXiv*.

[2] Covington, Paul & Adams, Jay & Sargin, Emre. (2016). Deep Neural Networks for YouTube Recommendations. 191-198. 10.1145/2959100.2959190.

[3] Kaggle MovieLens 100K Dataset page
https://www.kaggle.com/datasets/prajitdatta/movielens-100k-dataset