Deep Neural Networks for YouTube Recommendations

최서웅

Introduction

Challenge of Youtube Recommendation System

1. **Scale** : Handling YouTube's massive user base and corpus

2. **Freshness** : Responsive to model newly uploaded content and latest actions taken by the user

3. **Noise** : Model noisy implicit feedback signals, Metadata is poorly structured

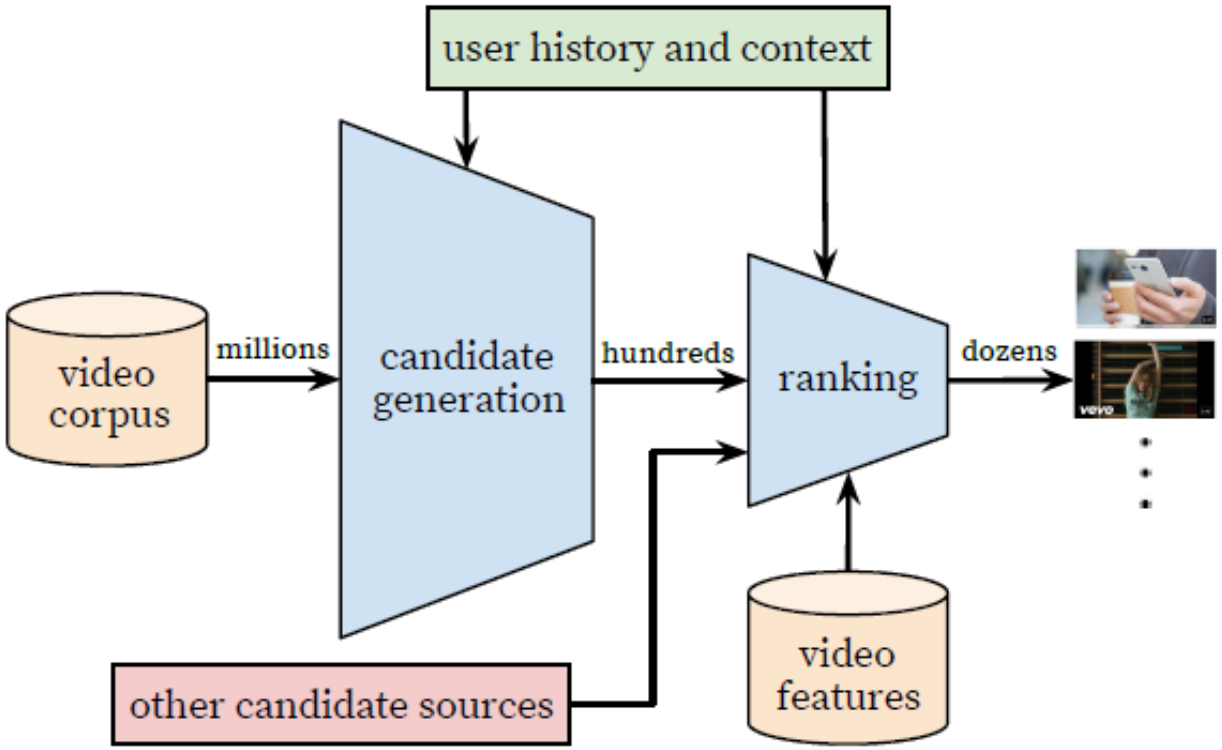
=> Robust to these datasets.

Introduction

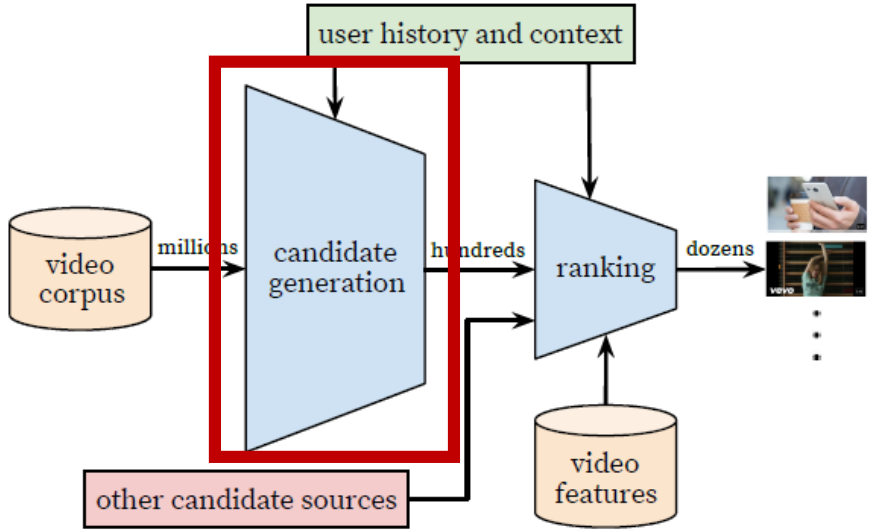
Using DNN…

1. Learn approximately one billion parameters
2. Trained on hundreds of billions of examples.

System Overview



System Overview



Two Neural Networks

1. Candidate Generation

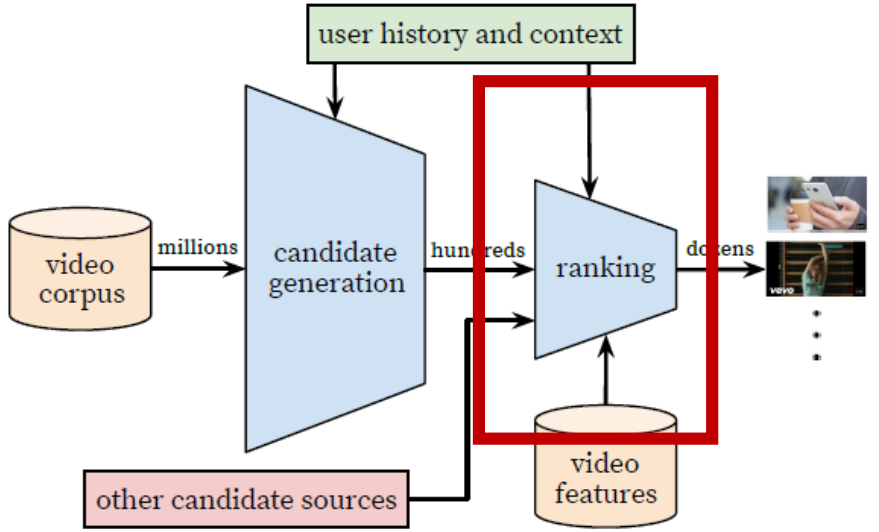
Input : User's YouTube activity history

Output : Small subset (hundreds) of videos from a large corpus

Broad personalization via collaborative filtering

Similarity between users : IDs of video watches, Search query tokens, demographics

System Overview



Two Neural Networks

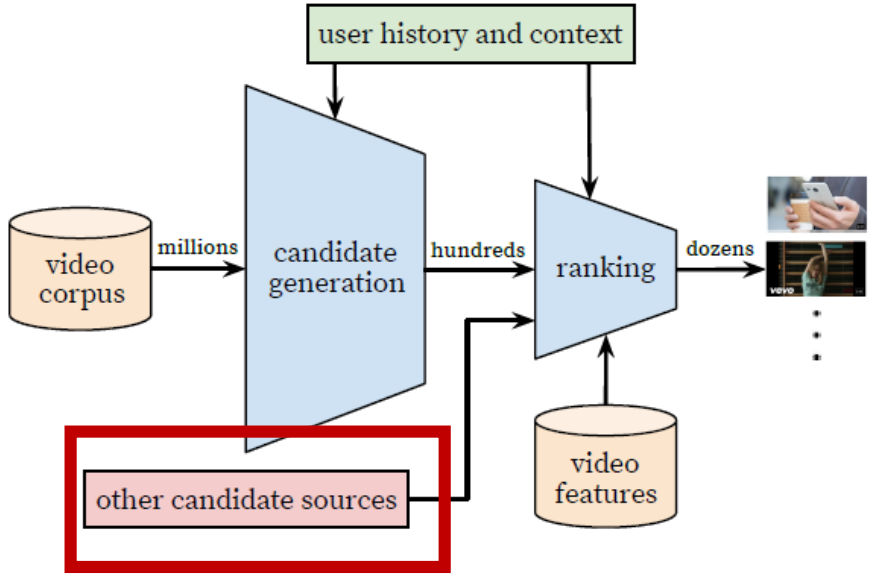
2. Ranking

Presenting a few “best" recommendations

- Using features of video and user, assign a score to each video
- The highest scoring videos are presented to the user, ranked by their score.

System Overview

Two stage approach



Make personalized recommendation from a very large corpus of videos

Enables blending candidates generated by other sources

System Overview

During development : Use offline metrics (precision, recall, ranking loss, etc.)

However for effectiveness of model, rely on **A/B testing** via live experiments.

Can measure subtle changes in click-through rate, watch time, and many other metrics that measure user engagement in live experiments.

- Live A/B results are not always correlated with offline experiments

Candidate Generation

Recommendation as classification

Pose recommendation as extreme **multiclass classification**

$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

$u \in R^n$: high-dimensional embedding of the user, context pair

$v_j \in R^n$: embeddings of each candidate video

V : Video corpus , U : User, C : Context

$w_t = i$: Watch video i at time t

Embedding : mapping of sparse entities into a dense vector in R^n.

Learn user embeddings u from DNN

Candidate Generation

Recommendation as classification

Although explicit feedback mechanisms exist on YouTube, we use the **implicit feedback** of watches to train the model.

Positive example = User completing a video.

This choice allows us to produce recommendations deep in the tail where explicit feedback is extremely sparse.

Candidate Generation

Efficient extreme multiclass

1. **Negative Sampling**

Use negative sampling to efficiently train model.

Cross-entropy loss is minimized for the true label and the sampled negative classes

Several thousand negatives are sampled(100 times speedup)

Candidate Generation

Efficient extreme multiclass

$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

2. **Serving Time**

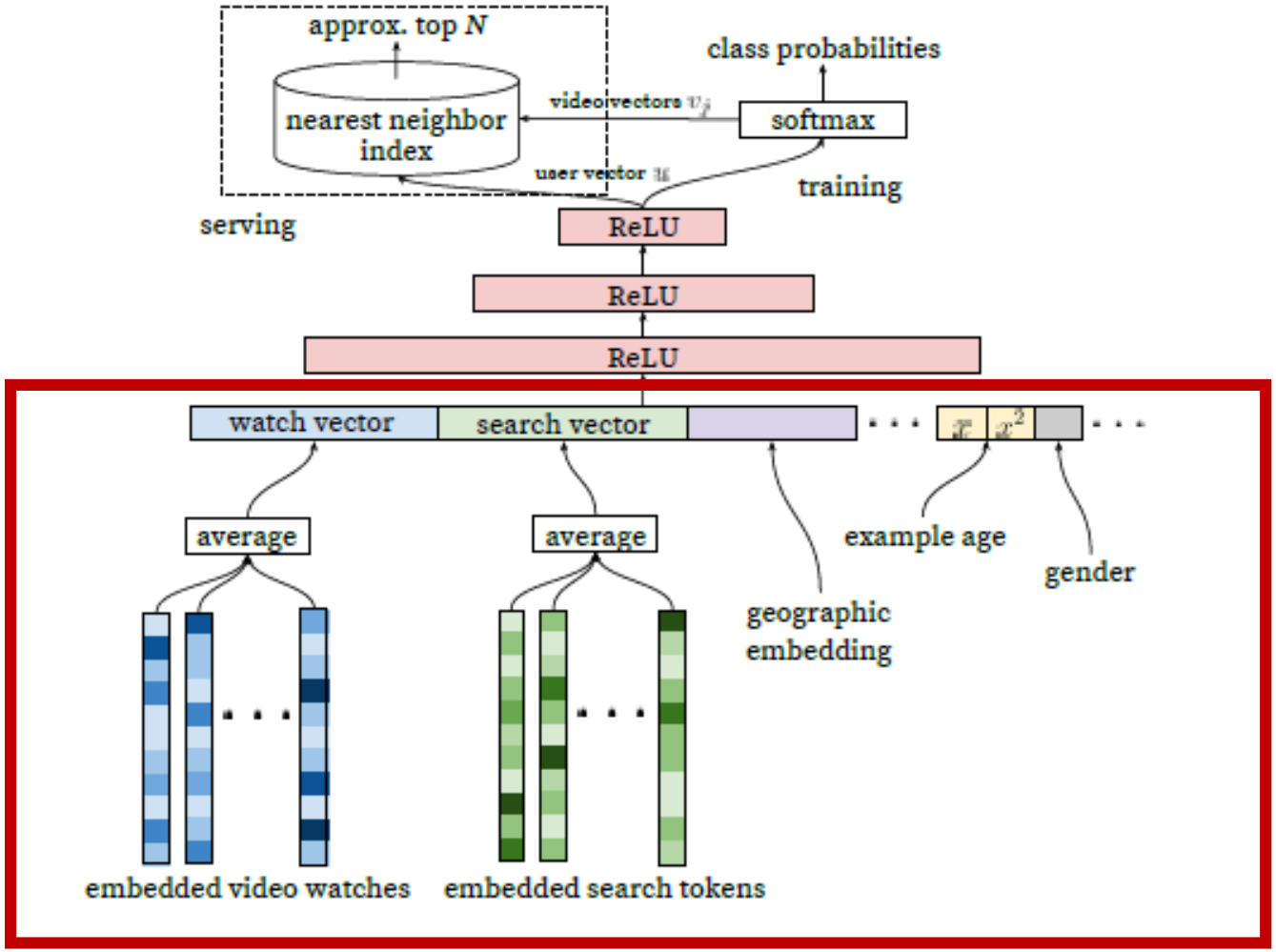
Compute the most likely N classes (videos) in order to choose the top N.

Use **Nearest neighbor search** in the dot product space

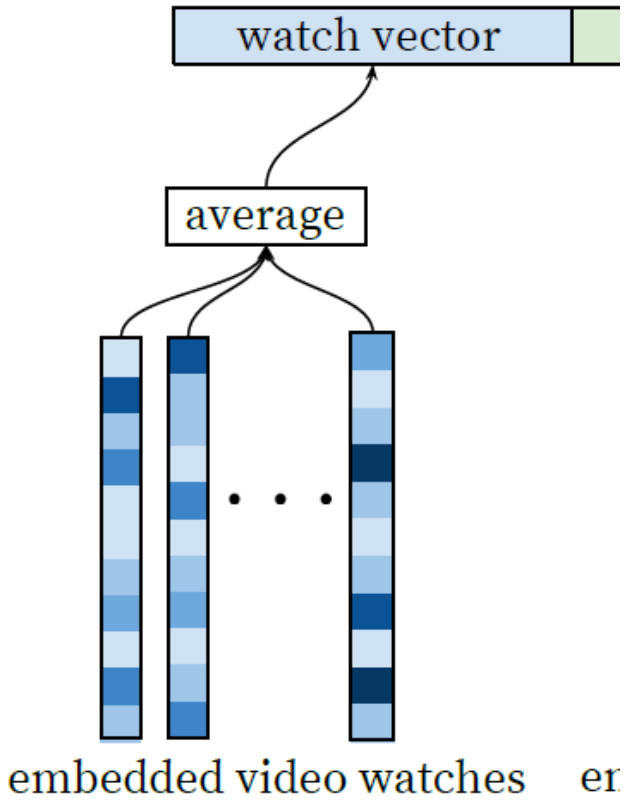
Not particularly sensitive to the choice of NNS algorithm.

Model Architecture(Candidate Generation)

Input Structure



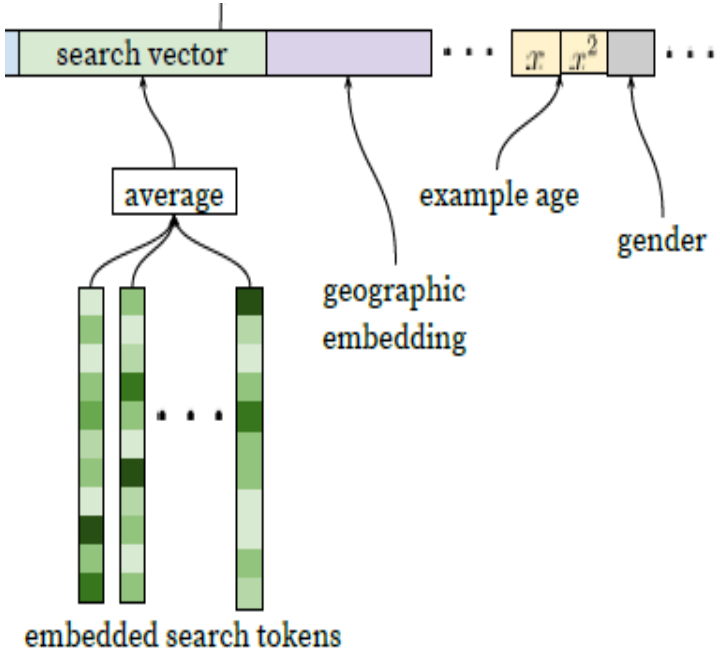
Model Architecture(Candidate Generation)



Watch Vector

- Learn high dimensional embeddings for each video.
- User's watch history is mapped to a dense vector representation by simply averaging the embeddings
- Embeddings are learned with all other model parameters.

Model Architecture(Candidate Generation)



Continuous and categorical features can be easily added to model

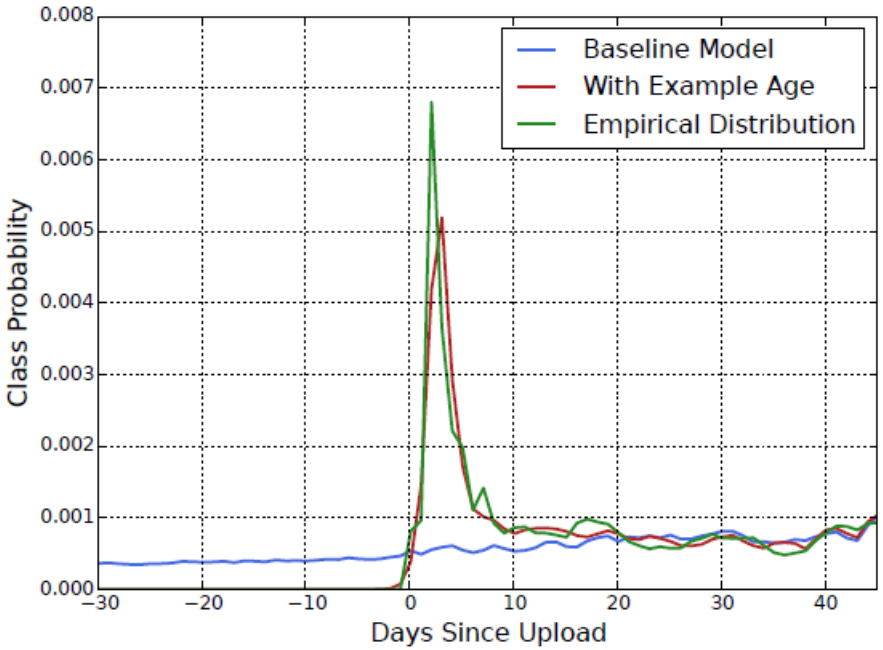
Search history

- Search history is treated similarly to watch history
- Each query is tokenized and embedded.
- Averaged embedded queries represent a summarized dense search history

Demographic Feature

- Make recommendations behave reasonably for new users.
- User's geographic region and device are embedded and concatenated.
- Simple binary and continuous features such as the user's gender, age are input directly into the network(normalized to [0, 1].)

Model Architecture(Candidate Generation)

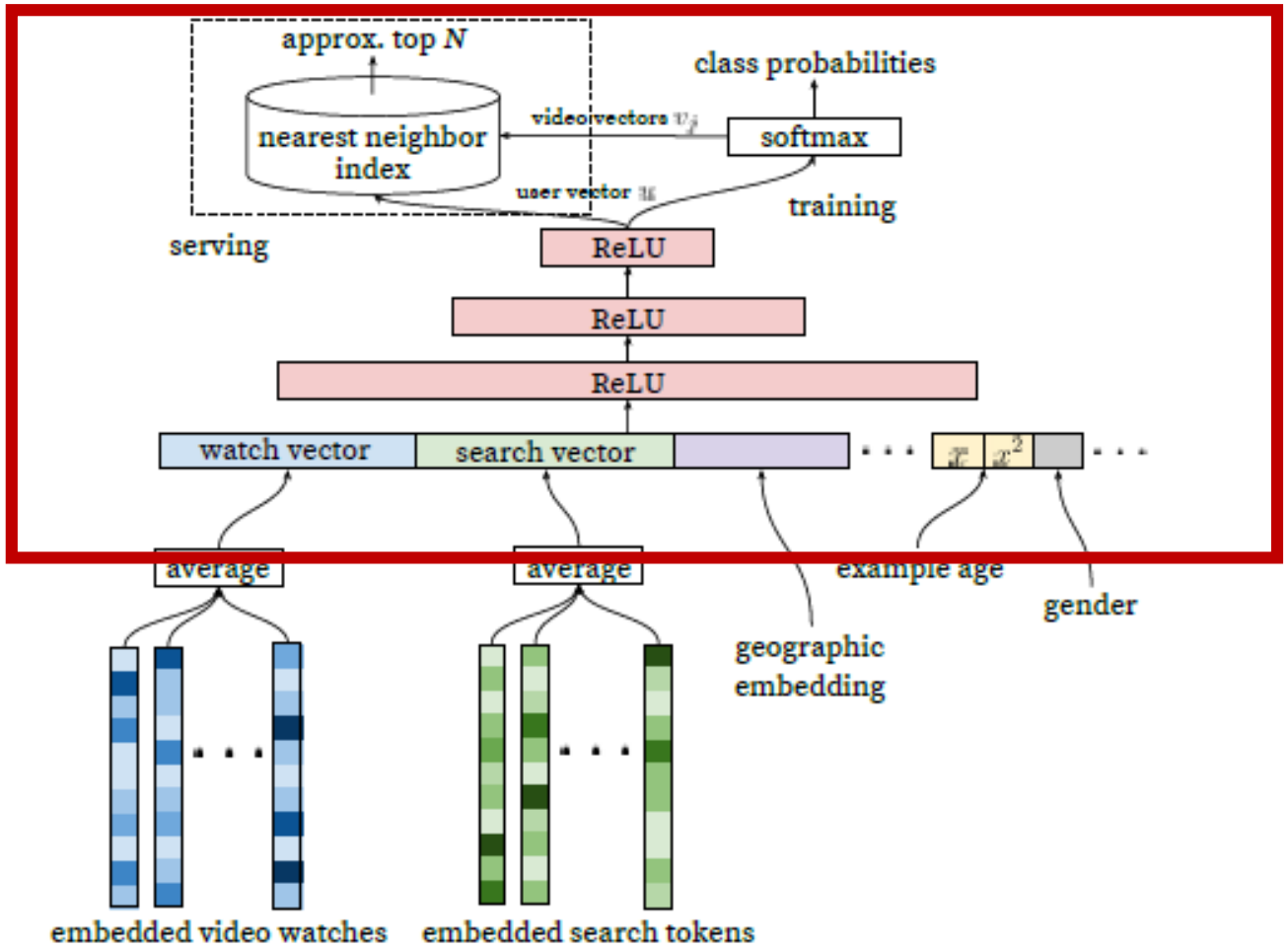


Example age

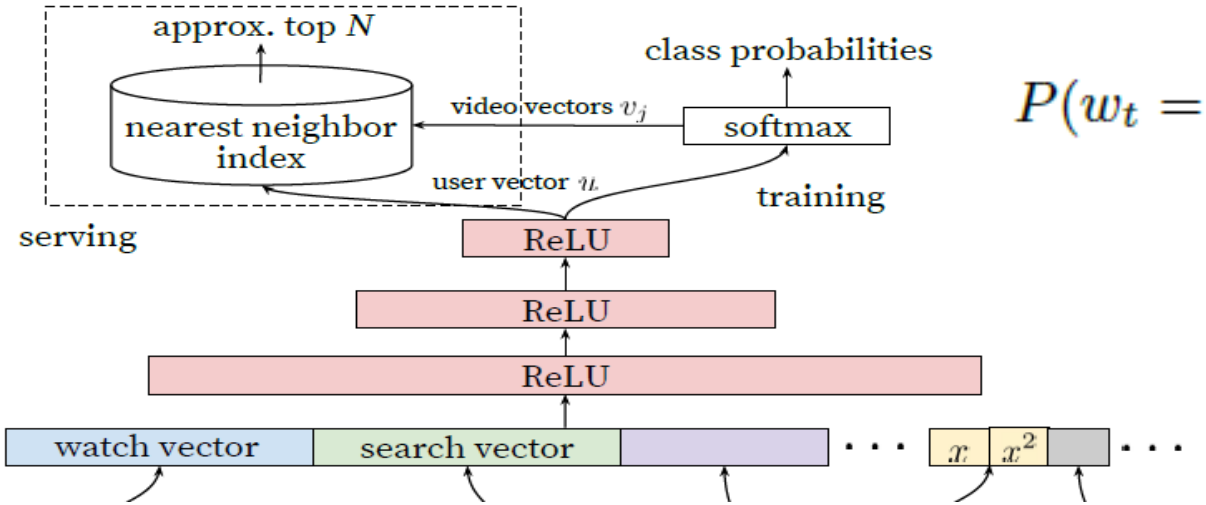
- Recommending this recently uploaded fresh) content is extremely important
- ML systems exhibit an implicit bias towards the past because they are trained to predict future behavior from historical examples
- To correct for this, use the **age** of the training example as a feature.

Model Architecture(Candidate Generation)

Neural Network



Model Architecture(Candidate Generation)



$$P(w_t = i|U, C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

- Features are concatenated into a wide first layer, followed by several layers of fully connected Rectified Linear Units (ReLU)

- The embeddings are learned jointly with all other model parameters through normal gradient descent backpropagation updates

Label and Context Selection

Surrogate problem

- Recommendation involves solving a surrogate problem and transferring the result to a particular context.
- Ex)Accurately predicting ratings leads to effective movie recommendations

- Surrogate learning problem has an importance on performance in A/B testing but is very difficult to measure with offline experiments.

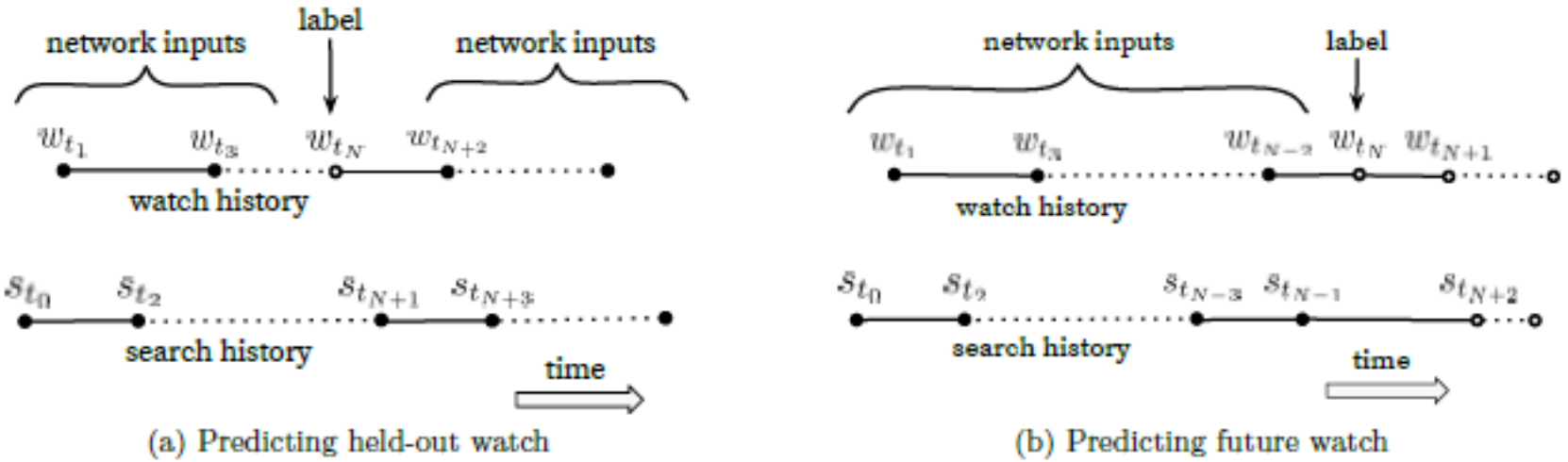
Label and Context Selection

Training Examples

- Training examples are generated from **all YouTube watches** (even those embedded on other sites) rather than just watches on the recommendations we produce.
- Otherwise, it would be very difficult for new content to surface and the recommender would be overly biased.

- Generate **a fixed number** of training examples per user.
- This prevented a small cohort of highly active users from dominating the loss

Label and Context Selection



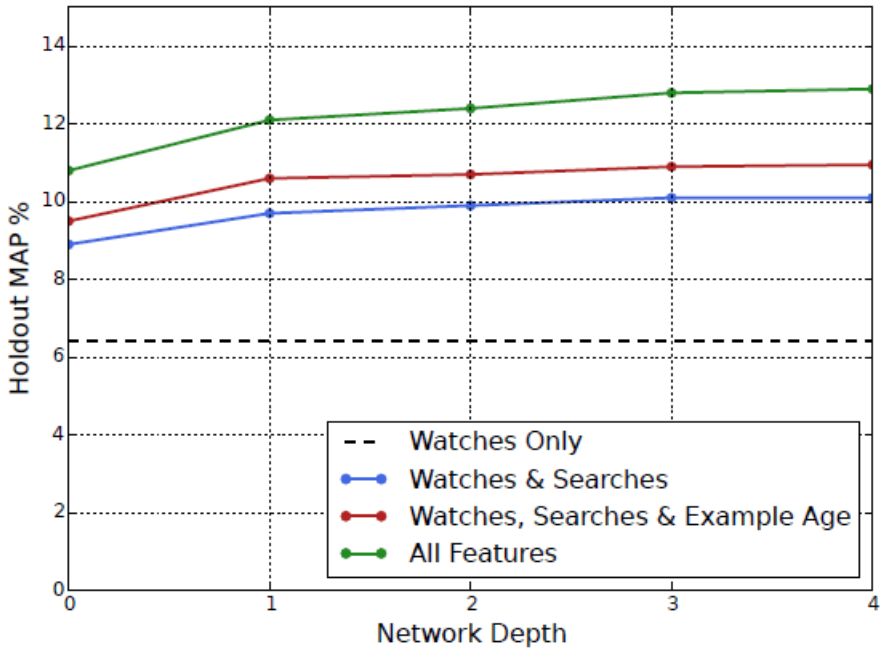
(a) Predicting held-out watch

(b) Predicting future watch

Predicting future watch

- Natural consumption patterns of videos : asymmetric co-watch probabilities
- Episodic series : watched sequentially
- Artists in a genre : most broadly popular to smaller niches
- Found much better performance predicting the user's next watch, rather than predicting a randomly held-out watch

Experiments with Feature and Depth



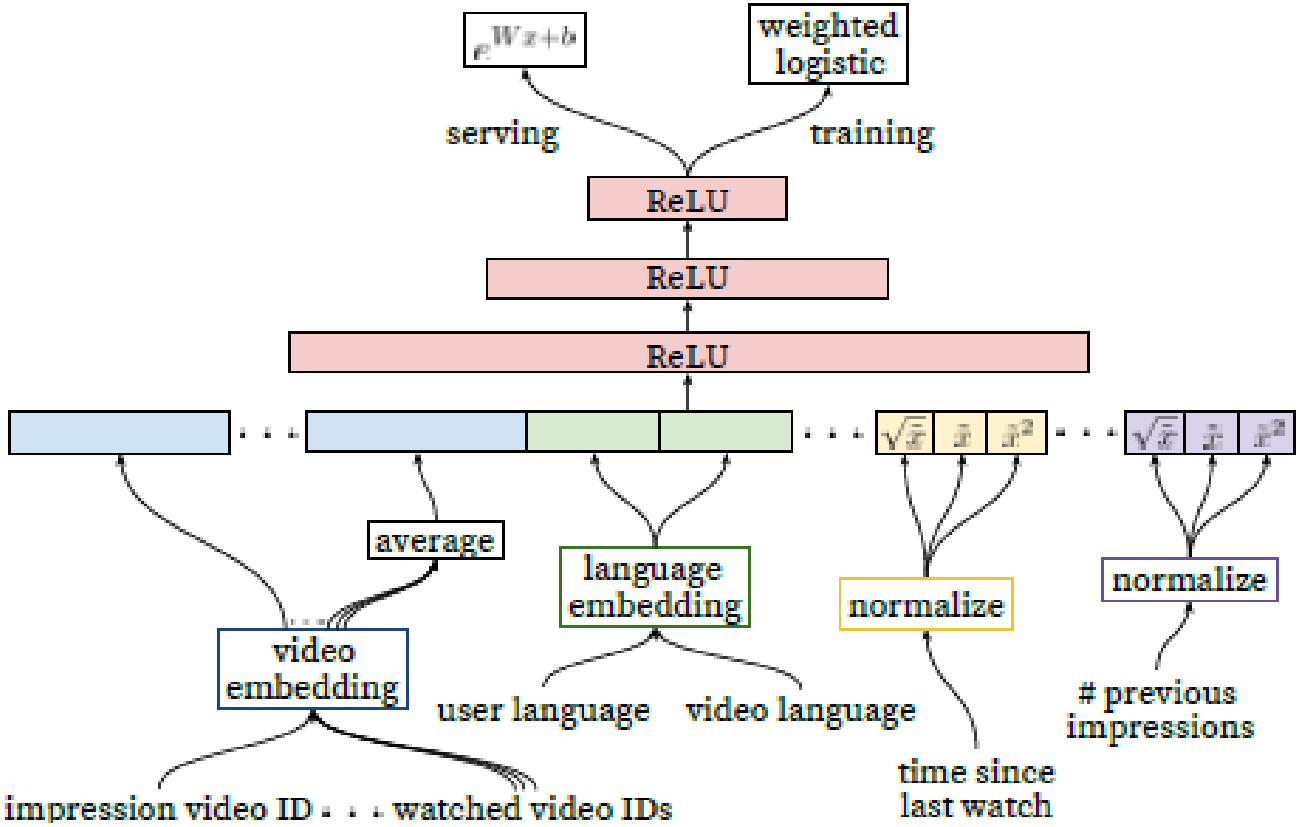
- Depth 0: A linear layer simply transforms the concatenation layer to match the softmax dimension of 256
- Depth 1: 256 ReLU
- Depth 2: 512 ReLU → 256 ReLU
- Depth 3: 1024 ReLU → 512 ReLU → 256 ReLU
- Depth 4: 2048 ReLU → 1024 ReLU → 512 ReLU → 256 ReLU

Network Structure

- 1M videos and 1M search tokens were embedded with 256 floats
- 50 recent watches, 50 recent searches
- Softmax layer outputs : 256
- The depth zero network is effectively a linear factorization scheme
- Network structure : tower pattern = bottom of the network is widest and decreasing

Feature ↑, Depth ↑ → Prediction ↑

Ranking



- Almost the same as the previous.
- However, it puts more efforts on feature engineering and applies the weighted logistic to put more attentions to the watch-time of each video

Ranking

- Role : Use impression data to specialize and calibrate candidate predictions

- Only a few hundred videos are being scored
- During ranking, have access to many more features describing the video and the user's relationship to the video

- Crucial for ensembling different candidate sources

Feature Representation

Data type

- Categorical Feature
  - Cardinality : Binary or Millions of possible values
  - Univalent : Contribute only single value ex) video ID
  - Multivalent : Set of values ex) N last videos watched

- Continuous Feature

Meaning

- Impression : describe properties of item
- Query : describe properties of the user/context

Feature Engineering

- Use hundreds of features
- Despite deep learning, considerable engineering resources transforming user and video data into useful features

Main challenge

- Representing a temporal sequence of user actions
- How these actions relate to the video impression (Relationship between user action and video impression)

Feature Engineering

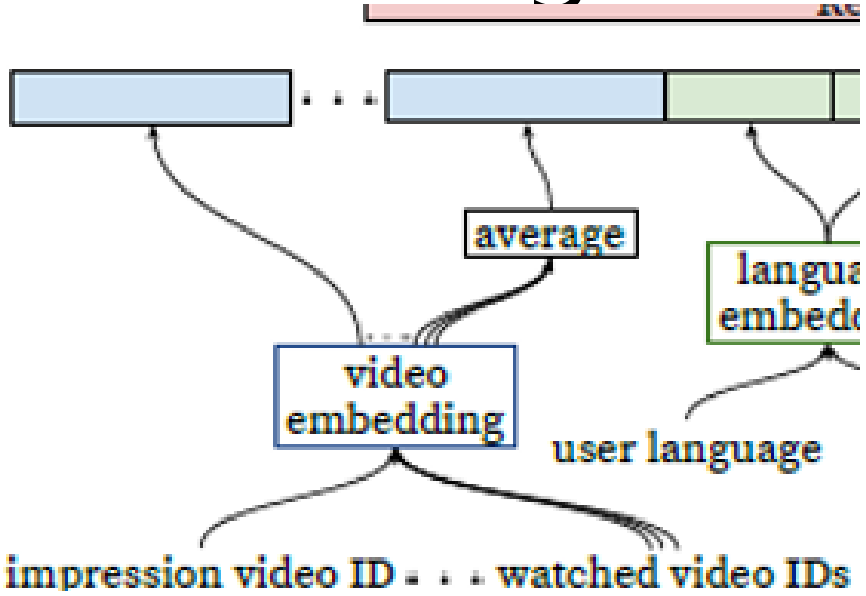
Important Signals

- User's previous interaction with the item itself and other similar items
- Ex) user's past history with the channel
  - Number of videos watched from this channel. Last time the user watched a video on this topic.

- Continuous features describing past user actions on related items are particularly powerful!

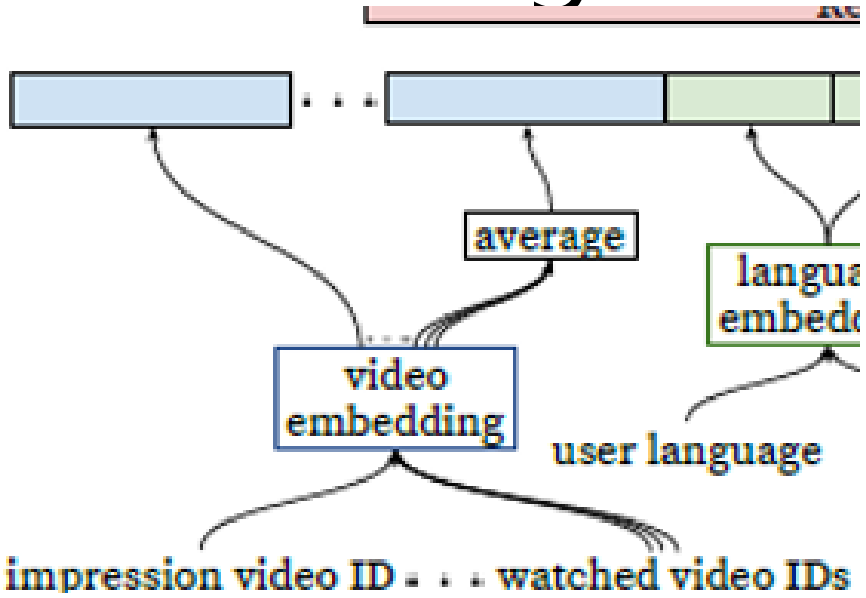
- + Don’t recommend videos that previous recommended but didn’t watch

Embedding Categorical Features



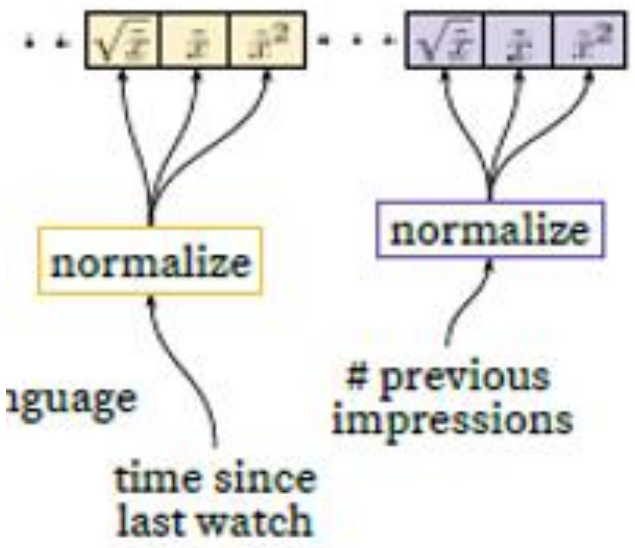
- Map sparse categorical features to dense representations
- Large cardinality ID spaces : Including only the top N (sorting based on their frequency in clicked impressions)
- Out of vocabulary : zero embedding
- Multivalent categorical feature embeddings are averaged

Embedding Categorical Features



- Categorical features share underlying embeddings
- For example, video ID
- Each feature is fed separately into the network
- Improving generalization, speeding up training, reducing memory requirements

Normalizing Continuous Features

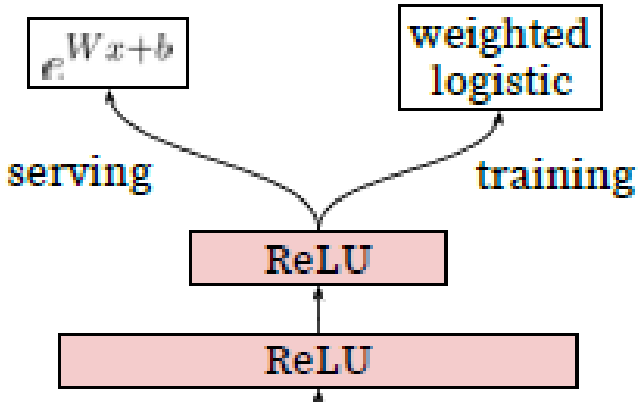


- Neural networks are sensitive to the scaling and distribution of their inputs
- Critical for convergence.

A continuous feature x with distribution f : transformed to [0, 1) by $\int_{-\infty}^{x} df$

Use $x^2, \sqrt{x}$:giving the network more expressive power

Modeling Expected Watch Time



- Predict expected watch time given training examples that are either positive (clicked) or negative (not clicked)
- Weighted logistic regression
  - Use Cross entropy loss
- Positive are weighted by the observed watch time on the video.
- Negative receive unit weight.

Modeling Expected Watch Time

$\frac{\sum T_i}{N-k}$: The odds learned by the logistic regression

T_i : The watch time of the i_th impression

N : Number of train examples

k : Number of Positive impressions

$$Odds = \frac{p}{1-p}$$

$$Odds(i) = \frac{w_i p}{1-w_i p}$$

$$ln(\frac{p}{1-p}) = \theta^T x \Rightarrow \frac{p}{1-p} = e^{\theta^T x} \Rightarrow p = \frac{1}{1+e^{-\theta^T x}} \Rightarrow p = sigmoid(\theta^T x)$$

Modeling Expected Watch Time

Assuming the fraction of positive impressions is small (which is true in our case), the learned odds are approximately

$$E[T](1+P)$$

$$Odds(i) = \frac{w_i p}{1-w_i p} \approx w_i p = T_i p = E(T_i)$$

Use e^x as the final activation function to produce these odds that closely estimate expected watch time

Experiments with Hidden Layers

Hidden layers	weighted, per-user loss
None	41.6%
256 ReLU	36.9%
512 ReLU	36.7%
1024 ReLU	35.8%
512 ReLU → 256 ReLU	35.2%
1024 ReLU → 512 ReLU	34.7%
1024 ReLU → 512 ReLU → 256 ReLU	34.6%

Weighted, per-user loss : the total amount mispredicted watch time as a fraction of total watch time
Consider both positive (clicked) and negative (unclicked) impressions
If negative receives a higher score than the positive, then consider the positive's watch time to be mispredicted watch time.

Feature ↑, Depth ↑ → Prediction ↑

Conclusions

Use DNN architecture for recommending YouTube videos

- Two Problems : Candidate generation and ranking
- Assimilate many signals and model their interaction with layers of depth

Using the age of the training example

- Removes an inherent bias towards the past
- Allows the model to represent the time-dependent behavior.
- Improve offline holdout precision results and increase the watch time.

- Outperform linear and tree-based methods for watch time prediction
- Benefit from specialized features describing past user behavior with items

Weighted Logistic regression

- Allows us to learn odds that closely model expected watch time.

Implementation

Dataset : Movielens-100k

Rating : ['userId', 'movieId', 'rating', 'timestamp']
Item : ['movieId', 'title', 'release_date', 'video_release_date', 'IMDB']
User : ['userId', 'age', 'gender', 'occupation', 'zip_code']

Only Candidate Generation

Feature engineering

	userId	windows	disliked_movies	watched_genres	labels	user_feat	example_age
0	1	[168, 172, 165, 156, 166, 196, 187, 14, 250, 127]	[245, 260, 264, 126, 237, 11, 8, 143, 94, 145,...	[18, 11, 12, 10, 5, 8, 9, 19, 15, 7, 14, 20, 1...	181	[1.0, 0.0, 0.25757575757575757, 1.0, 0.0, 0.0,...	0.020960
1	1	[172, 165, 156, 166, 196, 187, 14, 250, 127, 181]	[245, 260, 264, 126, 237, 11, 8, 143, 94, 145,...	[18, 11, 12, 10, 5, 8, 9, 19, 15, 7, 14, 20, 1...	117	[1.0, 0.0, 0.25757575757575757, 1.0, 0.0, 0.0,...	0.030940
2	1	[165, 156, 166, 196, 187, 14, 250, 127, 181, 117]	[245, 260, 264, 126, 237, 11, 8, 143, 94, 145,...	[18, 11, 12, 10, 5, 8, 9, 19, 15, 7, 14, 20, 1...	109	[1.0, 0.0, 0.25757575757575757, 1.0, 0.0, 0.0,...	0.032687
3	1	[156, 166, 196, 187, 14, 250, 127, 181, 117, 109]	[245, 260, 264, 126, 237, 11, 8, 143, 94, 145,...	[18, 11, 12, 10, 5, 8, 9, 19, 15, 7, 14, 20, 1...	1	[1.0, 0.0, 0.25757575757575757, 1.0, 0.0, 0.0,...	0.049583

Liked_movies : rating >= 3
Windows : Movies in "liked_movies" sorted in the order that user watched and then truncated to the length of WINDOW_SIZE.
Labels : Next movie after windows.
Watched_genres : Genre of Liked movies
User_feat : [gender(one hot encoding), age(normalized), occupation(one hot encoding)]
Example_age : Now - Release_date (Normalized)
Row per User >= 100

DNN Structure

User, Movie Embedding dim : 128
Epoch : 100
Learning rate : 0.01
Genre embedding dim : 16
ReLU : 512->256->128
Input width : embedding dim + genre embedding dim + user embedding dim (user_feat) + example age = 171

```
self.dense_1 = nn.Linear(embedding_dim + self.genre_embedding_dim + self.user_embedding_dim  + 3 , 512)
self.batch_norm_1 = nn.BatchNorm1d(512)
self.dense_2 = nn.Linear(512, 256)
self.batch_norm_2 = nn.BatchNorm1d(256)
self.dense_3 = nn.Linear(256, 128)
self.batch_norm_3 = nn.BatchNorm1d(128)
self.dense_output = nn.Linear(128, embedding_dim)
```

DNN Structure

```
def forward(self, windows, watched_genres, user_feat, watched_movie_age):


    windows_embedded = self.movie_embedding(windows)
    windows_embedded = self.l2_norm(windows_embedded)
    windows_embedded = windows_embedded.mean(dim=1)

    watched_genres = self.genre_embedding(watched_genres)
    watched_genres = self.l2_norm2(watched_genres)
    watched_genres = watched_genres.mean(dim=1)

    windows_embedded = windows_embedded.float()
    watched_genres = watched_genres.float()
    watched_movie_age = watched_movie_age.float()
    user_feat = user_feat.float()

    # concatenate all the embeddings

    x = torch.cat([windows_embedded, watched_genres, user_feat, watched_movie_age, torch.pow(watched_movie_age, 2), torch.pow(watched_movie_age, 0.5) ], dim=1)
    # Pass through dense layers with ReLU and Batch Norm
    x = torch.relu(self.batch_norm_1(self.dense_1(x)))
    x = torch.relu(self.batch_norm_2(self.dense_2(x)))
    x = torch.relu(self.batch_norm_3(self.dense_3(x)))

    # Output layer
    user_embedding = self.dense_output(x)
    return user_embedding
```

Learn genre, movie embedding as well as user embedding

Loss

```python
def loss(self, user_embedding, target_movies, negative_samples):
    target_movie_embeddings = self.movie_embedding(target_movies)
    negative_movie_embeddings = self.movie_embedding(negative_samples)

    target_scores = (user_embedding * target_movie_embeddings).sum(dim=1)
    negative_scores = (user_embedding.unsqueeze(1) * negative_movie_embeddings).sum(dim=-1)
    logits = torch.cat([target_scores, negative_scores], dim=1)
    targets = torch.zeros(logits.shape[0], dtype=torch.long, device=logits.device)

    loss_fn = nn.CrossEntropyLoss()
    loss = loss_fn(logits, targets)

    return loss
```

Using Cross entropy Loss
Negative sample = disliked movies

Result

Top 100 MAP(Mean Average Precision) for test dataset

```
mAP@100 = 0.18720094725082236
rmAP@100 = 0.12183775140522979
```