Deep Neural Networks for YouTube Recommendations

Covington et al. Google

2016 ACM RecSys

Presented by Haejune Lee

Main point

Two deep neural network

- Candidate generation
- Ranking

Online A/B testing : Impression

Large dataset handling

Able to use candidate video generated from other source

Feature handling

Challenges

- **Scale**: **User base와 video가 너무 많음**

  Highly specialized distributed learning algorithms and efficient serving system is needed

- **Freshness**: Video가 계속 업로드되는 **dynamic corpus**(집단)

  RecSys가 새 content를 반영할 수 있게 **Responsive** 해야 / Balancing between new with old content

- **Noise**: **Ground truth** 추출 어려움

  Sparce & unobservable external factors, noisy implicit feedback and poor structured metadata

DNN for RecSys

Candidate generation

- User history input → hundreds of video candidate
- Collaborative filtering: Broad personalization not individual

Ranking

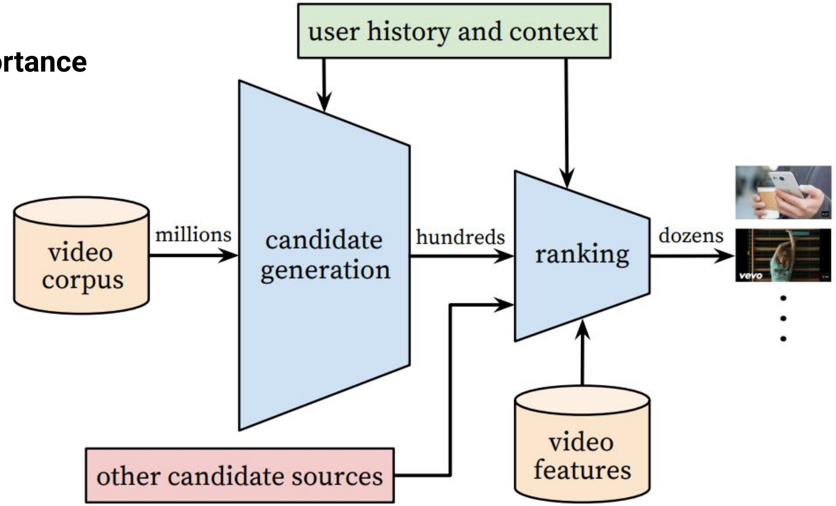
- Assign score to each candidate video to distinguish **relative importance**

Pros

- **Large corpus handle** 가능 → A few personalized videos
- Blending candidate generated **by other sources**

Testing

- Offline metric + Online A/B testing



Recommendation as multiclass classification

Recommendation 문제를 time t 에 특정 video를 보는 w_t 의 **classification** 문제로 치환

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

$u \in \mathbb{R}^N$: high-dim embedding of user & context pair

$v_j \in \mathbb{R}^N$: high-dim embedding of candidate video

- Embedding이 **sparse entity를 dense vector를 mapping**
- Only use implicit feedback (Complete a video)
- Top-N

Negative sampling

- 100x speedup over traditional softmax
- 충분한 sampling으로 성능 보장

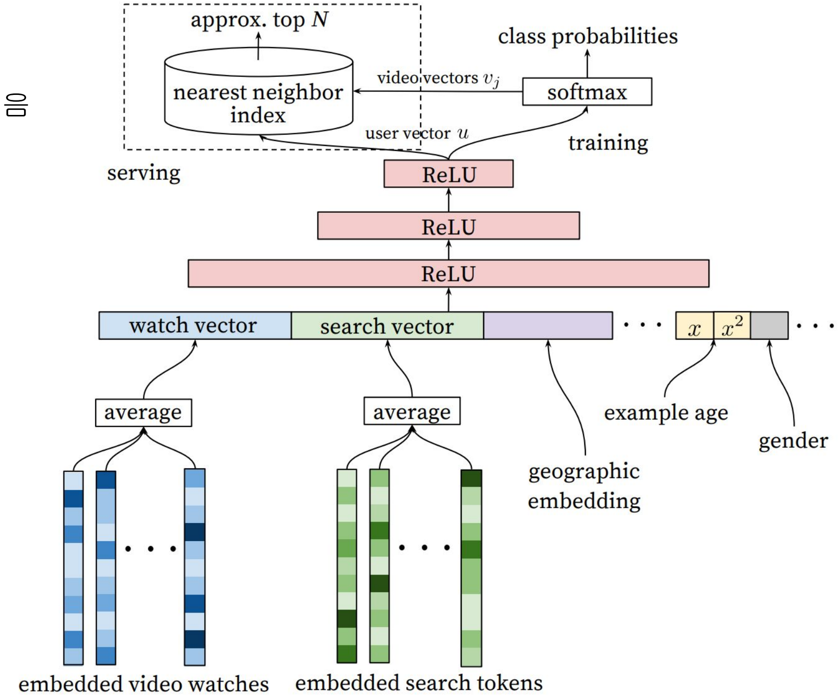
Model architecture

Network composition

- Feature들을 **fixed vocabulary**의 embedding으로 나타냄
- 각 **embedding을 averaging**하여 Feedforward network로 넣음
- Gradient descent backpropagation

DNN as a generalization of matrix factorization

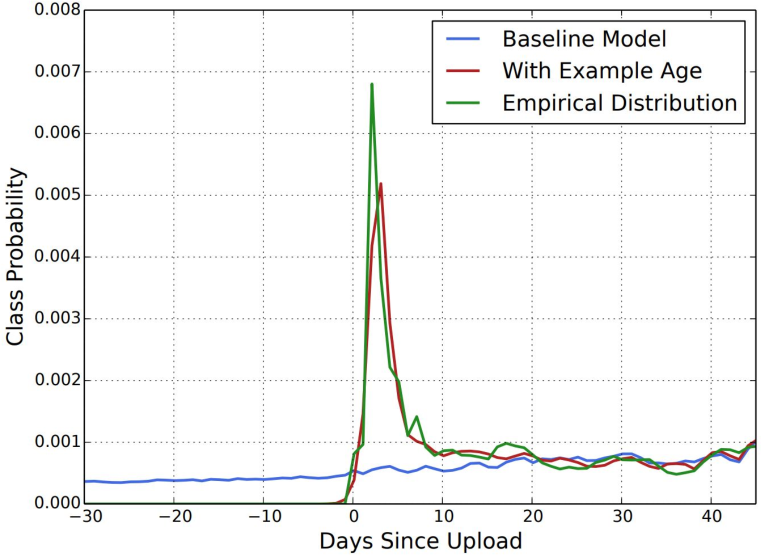
- Able to add continuous / categorical features
- Averaging tokenized search queries
  → **Summarized dense search history**
- Efficient to add additional informations

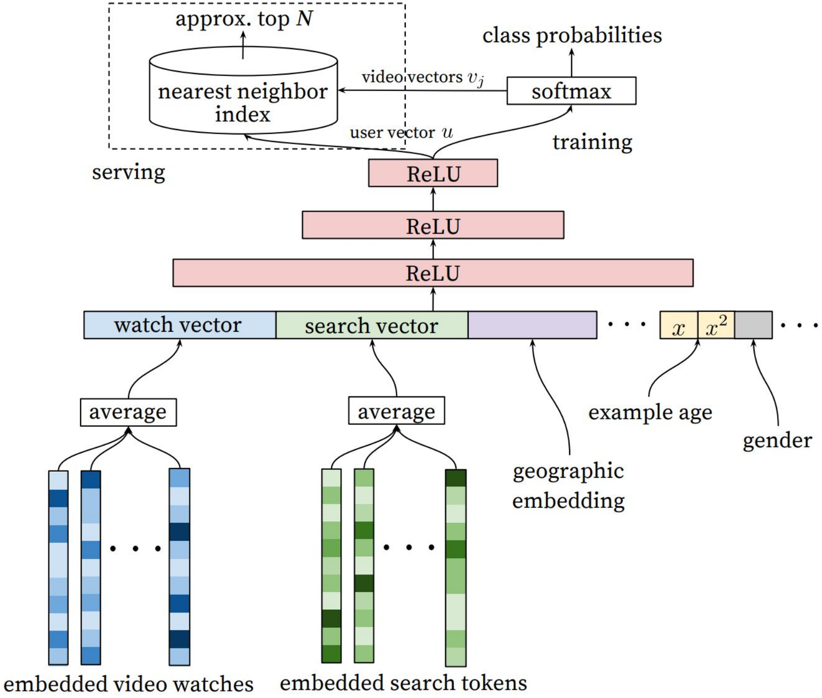


Model architecture

Fresh content to be recommended

- Bootstraping & Propagating viral content problem
- **Age example** to inhibit bias toward to the past





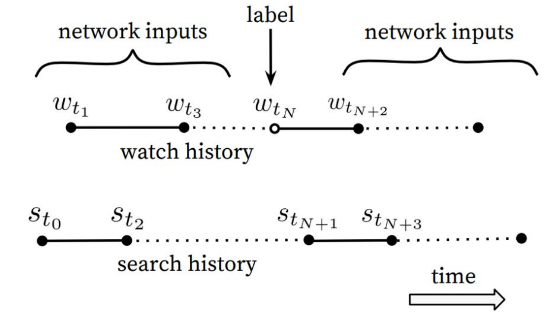
Label and context selection

Surrogate problem

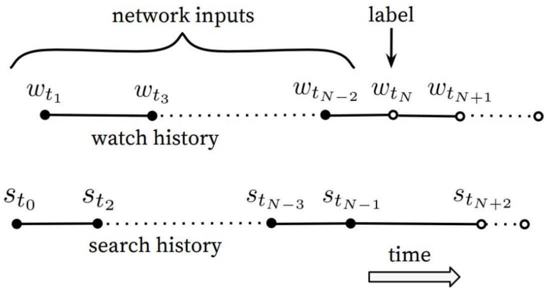
- Transfer recommendation problem into accurately predicting ratings
- A/B test가 검증에 필요

Additional Considerations

- Propagate not recommended & watched video to other users
- Restrict heavy users : Fixed number of training example per user
- **Withhold information from the classifier**: Prevent overfitting on surrogate problem
  search query를 순서가 없는 토큰 집합으로 표현
- Rollback dataset to the past: Reflect **asymmetric consumption pattern**



(a) Predicting held-out watch



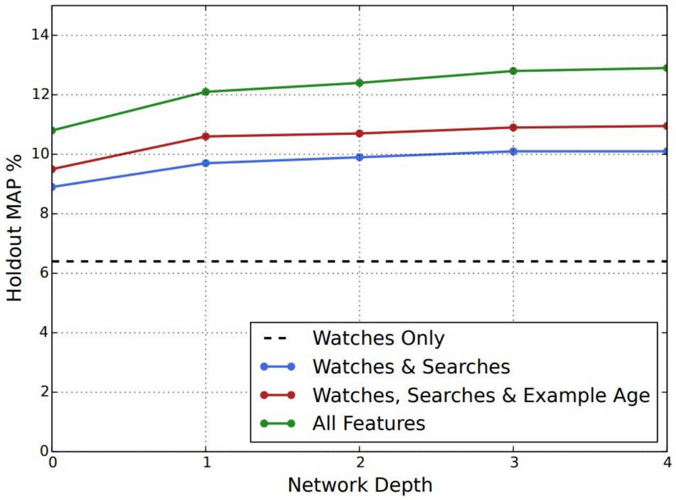
(b) Predicting future watch

Experiment with Feature and Depth

충분한 Feature와 Depth를 쌓는 것이 성능 향상을 보임

Tower pattern: input layer가 가장 크고, 다음 layer가 절반이 되는 구조

- 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



Further scoring using impression data

Categorical / Continuous features

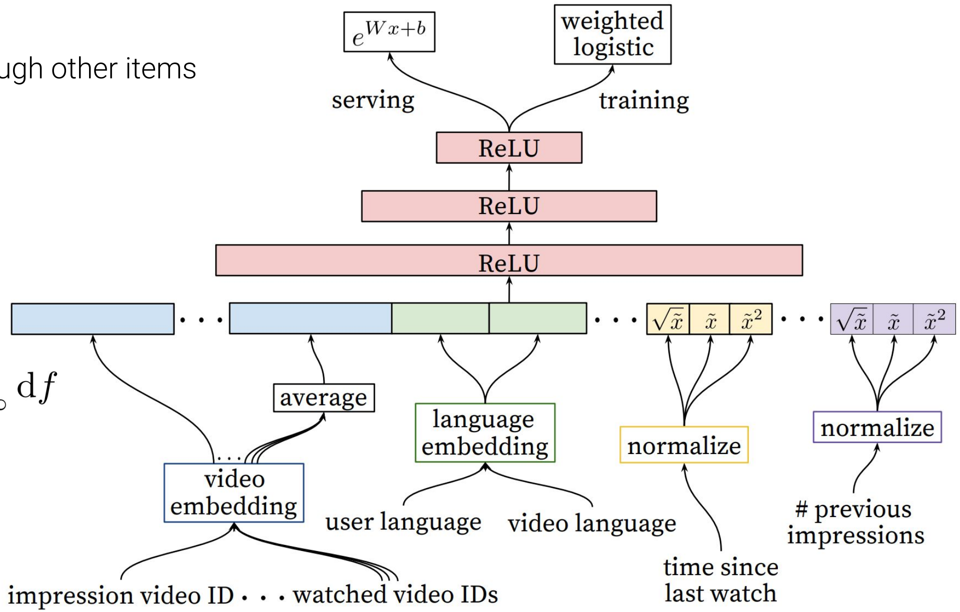
Properties of item (impression)/ Properties of user & context (query)

Useful features

- Past User-item interaction impression→ Generalize well through other items
- Frequency of past video impression
- Information used in candidate generation

Normalizing continuous features

- Scale, input distribution sensitive
- Normalize to [0,1) using cumulative distribution $\tilde{x} = \int_{-\infty}^{x} \mathrm{d}f$



Embedding

Unique ID space has separate learned embedding

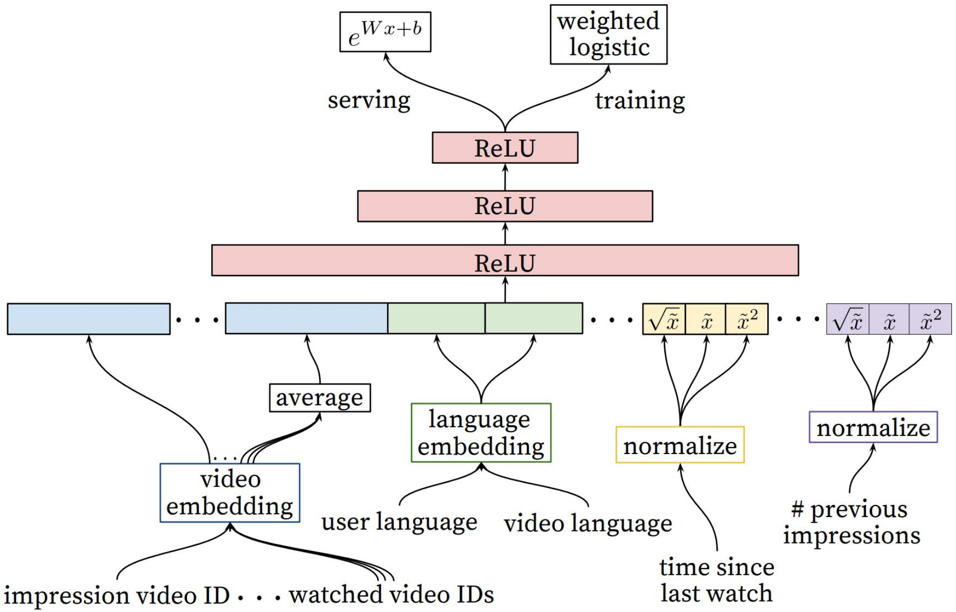
Categorical features using same ID space **share embedding**

Cardinality

Top-N after sorting based on click imbedding

Zero imbedding for the rest

Averaging multivalent categorical feature



Expected watch time

Weighted logistic regression

- Positive impression: observed watch time
- Negatvie impression: unit weight

Expected watch time

$$\frac{\Sigma T_i}{N-k} \approx E[T](1+P) \approx E[T]$$

Experiment

Loss: total amount of **mispredicted watch time** in Online test

Hidden layer의 넓이가 넓어질수록 성능 향상

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%

Dataset selection

User-item data & Impression data

YouTube

MovieLens 100K Dataset

MovieLens 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. 4/1998.

- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: https://grouplens.org/datasets/movielens/100k/

Dataset preprocessing

```
# 평점 3점 이상만 watch로 간주
rating_data = rating_data.loc[(rating_data['rating'].isin([3,4,5]))]
rating_data.drop(['rating', 'timestamp'],axis=1,inplace=True)
```

```
# search data 임의로 생성
search_data = rating_data.iloc[lambda x: x.index % 20 == 0]
search_data.rename(columns = {'user_id':'user_id','movie_id':'search_hist'},inplace=True)
search_dataset = search_data.groupby('user_id')['search_hist'].apply(list).reset_index()
```

```
# example_age 추가
movie_data['example_age'] = (pd.to_datetime("now") - pd.to_datetime(movie_data['release_date']))₩
          /np.timedelta64(1,'D')
```

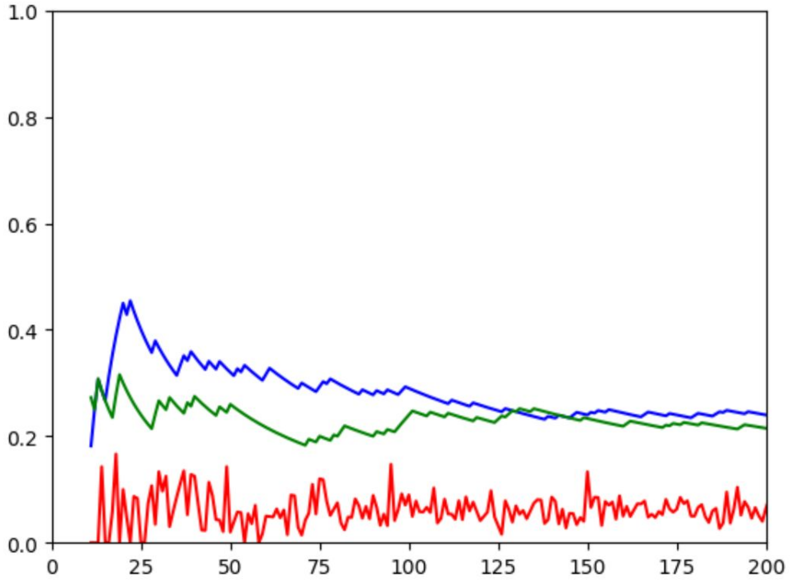
```
#  Negative sampling
total_movie_num = movie_data.shape[0]

def create_negative_sample(row, total_movie_num):
    all_movie_ids = set(range(total_movie_num))
    positive_set = set(row['watched_movies'])
    negative_set = list(all_movie_ids - positive_set)
    negative_sample = random.sample(negative_set, min(len(negative_set), 30))
    return negative_sample

train_dataset['negative_sample'] = train_dataset.apply(create_negative_sample, axis=1, total_movie_num=movie_data.shape[0])
```

Model construction

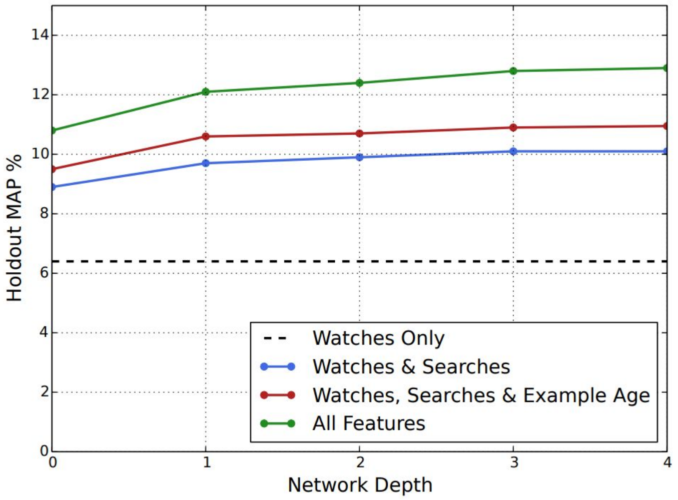
```
# Input layers
input_watched_movies = tf.keras.Input(shape=(None, ), name='watched_movies')
input_age = tf.keras.layers.Input(shape=(1), name='age')
input_gender = tf.keras.layers.Input(shape=(1), name='gender')
input_samples = tf.keras.Input(shape=(None, ), name='samples')

# Embedding layers (영화 기록을 임베딩 하기 위한 embedding layer, search data는 사용 안함)
features_embedding_layer = tf.keras.layers.Embedding(input_dim = MOVIE_NUM+1, output_dim = EMBEDDING_DIMS, mask_zero=True, name='features_embedding'
average_embedding_layer = Avg_Embedding(name='features_embedding_average')

# Dense layers
dense_1 = tf.keras.layers.Dense(DENSE_UNITS, activation='relu', name='dense_1')
dense_2 = tf.keras.layers.Dense(EMBEDDING_DIMS, activation='relu', name='dense_2')

#------------------------#
# Model1 connection
watched_movies_embedding = features_embedding_layer(input_watched_movies)
sample_movies_embedding = features_embedding_layer(input_samples)

average_embedding = average_embedding_layer(watched_movies_embedding)
concat_features = tf.keras.layers.concatenate([average_embedding, input_age, input_gender], axis=1, name='concatenate_features')
dense_1_out = dense_1(concat_features)
dense_2_out = dense_2(dense_1_out)
#
dot_product = tf.keras.layers.dot([dense_2_out, sample_movies_embedding], axes=(1,2), name='dot_product')
output = tf.keras.layers.Activation('softmax', name = 'class_probabilities')(dot_product)

model1 = tf.keras.Model(inputs=[input_watched_movies, input_age, input_gender, input_samples], outputs=[output])
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LEARNING_RATE), loss='sparse_categorical_crossentropy', metrics=['acc'])

model1.summary()
```

Search data difference

Search data를 포함(b) / 포함X(g)



- 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



Conclusion

Two deep neural network: Dense represenation embedding / logistic regression

Surrogate problem: classifying a future watch

Feature engineering

Youtube RecSys

1. The YouTube Video RecSys (2010)

    시청한 video 유사도 기반 video set 매핑, top-N

2. DNN for YouTube (2016)
3. Recommending What Video to Watch Next: A Multitask Ranking System (2019)

    Wide & Deep 확장, Multimodal feature 활용

QnA