Deep Neural Networks for YouTube Recommendations

1 Citation

Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." Proceedings of the 10th ACM conference on recommender systems. ACM, 2016.

`https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45530.pdf`

2 Abstract

High level description of YouTube video recommendation system, which has two pieces: candidate generation and ranking.

3 Introduction

Recommending YouTube videos is hard because there are many videos (scale), new ones are created all the time (freshness), and we don't have good groundtruth on user satisfaction (noise). We train a billion parameter Tensorflow model on hundreds of billions of examples.

4 System Overview

Candidate generation fetches a few hundred videos per user. It's a collaborative filtering model based on user watch history and demographics. We then score each video with a ranking model that uses a rich set of features. We use offline metrics like precision, recall, and ranking loss, but A/B testing is our true metric of success.

5 Candidate Generation

We seek to predict which video a user will watch at time t, called w_t given the user U and context C. We use embedding vectors u and v_j, which represent the user embedding and embedding of video $j \in V$. This gives:

$$P(w_t = i|U,C) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}} \tag{1}$$

Our positive examples are (user, video) pairs where the user fully watched the video. Other feedback signals, such as video likes, are too sparse.

We generate negative samples with random sampling and importance weighting. This performs better than hierarchical softmax because it's hard to create a good hierarchy.

Once we have user and video embeddings, we can find the nearest neighbor videos for a given user and use those as candidates for ranking.

Our model takes our features, encodes them as fixed sized vectors, pushes these vectors through a feedforward network, and finally predicts which of the given videos (one positive example and many negative examples) the user will watch. We represent user searches and user previous video watches as a bag of IDs. We learn an embedding vector for each ID and average the embedding vectors to produce a fixed sized vector representation for a bag of IDs. We include other features like user age, gender, and geographic location (we learn an embedding of user geographic location in a separate system). Continuous features are normalized to lie between 0 and 1.

Since users prefer fresh content, we include the age of the video as a feature.

We collect positive examples wherever we can, even from YouTube videos embedded on other websites. This gives us a better picture of what videos users like. We limit the number of positive examples each user can produce to avoid overrepresenting active users. We find that using a bag of IDs rather than sequence of IDs prevents the model from just recommending the same content that a user just watched.

When creating features, we consider a positive example and only consider actions the user took BEFORE they watched that video. This avoids feature leakage.

Adding features and making the model deeper yields better performance. Later layers are less wide than previous layers.

6 Ranking

We can use more features in the ranking phase because we only have a few hundred candidates. During A/B testing, our key metric is watch time per impression. This is better than click through rate because it fights against clickbait.

Our features are categorical (bag of IDs) or continuous (floating point number). Categorical features are univalent (i.e. one ID) or multivalent (i.e. multiple IDs). A feature can describe the video or the user/context. The latter is computed once per request while the former is computed once per candidate video in the request.

We use hundreds of features, roughly half of which are categorical. We spend a lot of time engineering features. The most useful features describe how users previously interacted with the video's channel or the topic of the video. The score that the candidate fetcher gave the video is also important. Knowing whether a video was recommended to this user before is also important.

To pick the embedding length for a categorical feature, we set it proportional to log(number of distinct IDs). To limit the size of the embedding lookup table, we learn embeddings only for the most frequent IDs and use an out-of-vocabulary embedding for the rest. We share embedding lookup tables when different features use IDs from the same space. Almost all the model parameters are due to the embedding lookup tables.

To normalize continuous features, we compute a CDF of feature values and represent the feature with its quantile. Suppose x is a normalized continuous feature, we find gains by including x^2 and $\sqrt{x}$ as features as well.

Positive examples are tagged with the watch time (i.e. how long did user watch the video). Our model is trained with weighted logistic regression under cross entropy loss. Negative samples have unit weight and positive examples are weighted by their watch time. With this formulation, we can take our model score for a video, call it s, and compute e^s to get expected watch time. See paper for details.

To compare different architectures, we look at their weighted per-user loss. To do this, we run an A/B test. We consider impressions where a positive example eceived a lower score than a negative example. The watch time of the positive example is called the mispredicted watch time. A user's weighted loss is their total mispredicted watch time divided by their total watch time.

Wider, deeper models do better, at the cost of increased CPU usage at inference time. A 1024-512-256 model worked the best for us. Using x^2 and $\sqrt{x}$ as feature transformations cut loss by 0.2%. Using importance weighting cut loss by 4.1%.

7 Conclusion

We formulate recommendation as candidate generation and ranking. We generate candidates with a deep collaborative filtering model. We train our models to predict the next watch, which is a good surrogate problem for training. Using the age of a video as a feature prevents the model from recommending old videos. For ranking, we weight examples to predict watch time, which is better than predicting click through rates.