

CS 410 Tech Review: Search Ranking and Recommendation System

Grace, Mu-Hui Yu (Fall 2021), [Google Doc link](#) for this file

Overview

Recommendation algorithms are part of all major online businesses these days in terms of helping businesses improve conversion rates, product click-thru, and other key metrics. Earlier recommendation systems were purely based on matrix factorization, which only considered what individual users view history and rating. However, recent studies have explored various methods and biases to develop a recommendation system, and they tried to include more product-related and customer-related features (e.g. user demographic information) to create a better personalized recommendation.

In the meantime, from traditional ranking systems to deep learning, more and more systems are being implemented on real-time platforms nowadays. It's not a new area of study, yet there are always substantial improvements in recommendation quality for all systems.

This technical review aims to provide different perspectives and methods on academic papers and sharing keynotes from one of the world-leading platforms "YouTube". In the coming sections, I would introduce two different methods YouTube used in their recommendation system, including Deep Neural Network (2016) and Multitask Ranking System (2019). I focus specifically on the practical implementations of each system and summarize the main ideas. Also, I would implement these two models in Python (real code for Deep Neural Network and pseudocode for Multitask Ranking System), and share my results and comments below.

Paper 1 - Deep Neural Networks for YouTube Recommendations (P. Covington et al, 2016)¹

Overview

This is one of the most famous YouTube recommendation system papers by Paul Covington, Jay Adams, and Emre Sargin. To create a recommendation system for YouTube, we may face these problems: **scale** (how to learn from billions of users), **freshness** (500+ hours of new video posted on YouTube per minute), and **noise** (some implicit bias affects the click-thru result).

This paper focuses on solving the first problem: the large user scale, by using the combination of the candidate generator system and the ranking system. Instead of sorting the results based on user watch history and click-thru only, this model also takes other demographic and heterogeneous features into account, such as user language, video language, previous impressions. The first part filters candidate videos from millions to hundreds based on user preferences, watch history, demographic features, etc. And the second part finishes the candidate ranking and provides the result to users.

¹ Paul Covington et al, "Deep Neural Networks for YouTube Recommendations", 2016, <https://static.googleusercontent.com/media/research.google.com/ko/pubs/archive/45530.pdf>

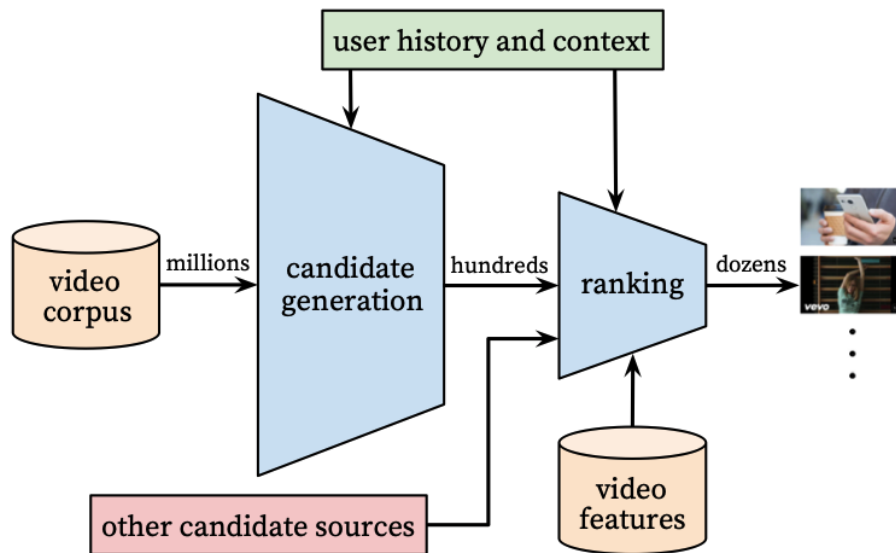


Fig1: Youtube Recommendation System Architecture (from the paper)

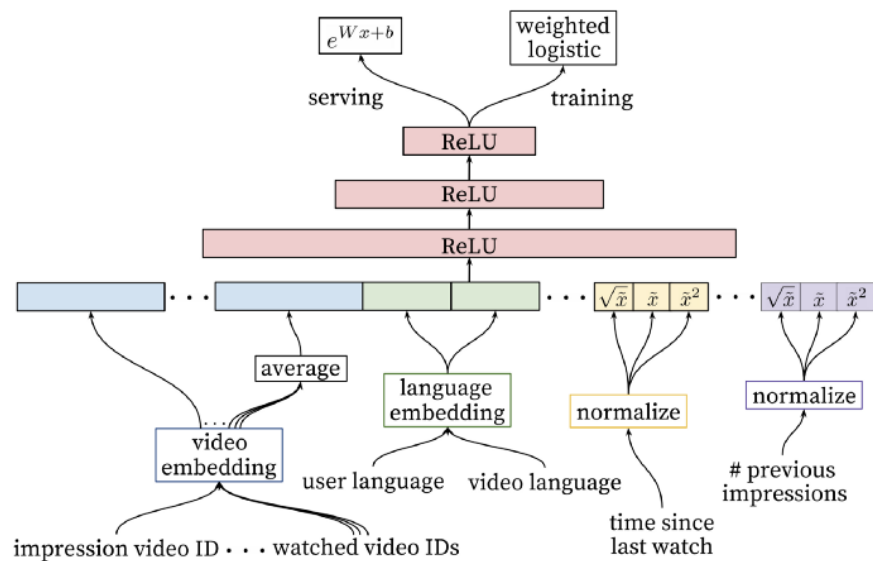


Fig1: Deep candidate generation model architecture (from the paper)

Python Implementation

This python implementation follows this tutorial article on Medium: [Implementing the YouTube Recommendations Paper in TensorFlow — Part 1](https://theiconic.tech/implementing-the-youtube-recommendations-paper-in-tensorflow-part-1-d1e1299d5622)² by Revathi Prakash.

Here I use the sample data set from [MovieLens](https://grouplens.org/datasets/movielens/). The dataset contains 100,000 ratings from 943 users on 1682 movies. The sample output looks like the following.

² Revathi Prakash, "Implementing the YouTube Recommendations Paper in TensorFlow — Part 1", 2020, <https://theiconic.tech/implementing-the-youtube-recommendations-paper-in-tensorflow-part-1-d1e1299d5622>

movies.head(3)

	movie_id	title	release_date	video_release_date	imdb_url	genre_unknown	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	File
0	0	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	1	0	0	0	0	
1	1	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	0	0	0	0	0	
2	2	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	0	0	0	0	0	

Image 1: Movie list

[6] rating_details_sample.head(10)

	user_id	movie_id	rating	unix_timestamp	title	release_date	video_release_date	imdb_url	genre_unknown	Action	Adventure	Animation	Children	Comedy	Crime	D
0	195	241	3.0	881250949	Kolya (1996)	24-Jan-1997	NaN	http://us.imdb.com/M/title-exact?Kolya%20(1996)	0	0	0	0	0	0	1	0
1	195	256	2.0	881251577	Men in Black (1997)	04-Jul-1997	NaN	http://us.imdb.com/M/title-exact?Men+in+Black+...	0	1	1	0	0	1	0	
2	195	110	4.0	881251793	Truth About Cats & Dogs, The (1996)	26-Apr-1996	NaN	http://us.imdb.com/M/title-exact?Truth%20About...	0	0	0	0	0	1	0	

Image 2: Rating details

In the article, the author used the bare minimum architecture to combine all basic elements that are mentioned in the YouTube candidate generation network. Including:

- Input layer
- Embedding layer
- L2 normalization layer
- Embedding aggregator
- Concatenate layer
- ReLU and Batch Norm layers
- Dense output layer
- Loss function

It is noted that for the loss function, the author used only `sparse_categorical_crossentropy` to reduce the loss between the next predicted video and what the user has actually watched from the video corpus, while the author didn't consider the concept of negative sampling (with the help of sampled softmax loss as it implemented in TensorFlow). Also, according to the paper, for the aggregator, taking the component-wise average was found to work the best: *"The network requires fixed-sized dense inputs and simply averaging the embeddings performed best among several strategies (sum, component-wise max, etc.)."*

The model architecture in the simplest form looks like the following.



Image 3: Candidate Generator Network Architecture

For the training process, I followed the author's framework. I picked an arbitrary user, trained the network for 1000 epochs, and observe the results to verify the article. According to the article, it claims that for users with large watch histories, the network could predict the next video correctly. For some of the other users, the actual video the user watched was in the top 20 video recommendations. Here is the actual result I got from my implementation.

One problem I found during the implementation was a code error from the author while running the training process, `model.fit()`. I believe that was caused by an out-of-bounds index error, which makes the list not match the other list elements' lengths.

```
model.fit([
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['title_d']),
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['like']),
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['dislike']),
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['all_genres'])
],
    user_title_list_e['predict_labels'].values, callbacks=[tensorboard_callback],
    steps_per_epoch=1,
    epochs=1000,
    verbose=1
)
```

Train on 500 samples
Epoch 1/1000
WARNING:tensorflow:5 out of the last 8 calls to <function L2NormLayer.call at 0x7f5fe2c1fef0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings i
WARNING:tensorflow:5 out of the last 8 calls to <function MaskedEmbeddingsAggregatorLayer.call at 0x7f5f8a7323b0> triggered tf.function retracing. Tracing is expensive and the excessive
500/500 [=====] - 2s 5ms/sample

InvalidArgumentError Traceback (most recent call last)
<ipython-input-17-f47d5f0470fb> in <module>()
 9 steps_per_epoch=1,
 10 epochs=1000,
--> 11 verbose=1
 12)
 11
 12)

11 frames
/usr/local/lib/python3.7/dist-packages/six.py in raise_from(value, from_value)
InvalidArgumentError: indices[3818] = 4502 is not in [0, 4500]
[[{node: model/aggregate_embeddings_3/PartitionedCall/RaggedMask/boolean_mask/GatherV2}}] [Op: __inference_distributed_function_3953]

Function call stack:
distributed_function

SEARCH STACK OVERFLOW

Image 4: out-of-bounds index error

I fixed the code by adjusting the length of INPUT_DIM and downgrade my Tensorflow. I got the same result as the one from the article.

```
model.fit([
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['title_d']),
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['like']),
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['dislike']),
    tf.keras.preprocessing.sequence.pad_sequences(user_title_list_e['all_genres'])
],
    user_title_list_e['predict_labels'].values, callbacks=[tensorboard_callback],
    steps_per_epoch=1,
    epochs=1000,
    verbose=1
)
```

Epoch 876/1000
1/1 [=====] - 0s 29ms/step - loss: 4.6698
Epoch 877/1000
1/1 [=====] - 0s 30ms/step - loss: 4.6924
Epoch 878/1000
1/1 [=====] - 0s 30ms/step - loss: 4.6815
Epoch 879/1000

Image 5: Training results of the candidate generator network

To summarize the implementation of this simple deep learning model structure, it is far from perfect, but it is a step in the right direction. As we learned in the Text Information System lectures and the following paper in this assignment, we can always add more features and vary the model architecture to learn more complex patterns and make the output more reliable.

Paper 2 - Multitask ranking system (Z. Zhao et al, RecSys 2019)³

Overview

The goal of this paper is to recommend what video to watch next on YouTube based on a large-scale multi-objective ranking system.

It's a quite practical framework that solves the two main problems as below when developing a recommendation system.

1. **Multiple competing ranking objectives (Multitask Learning)**: Besides the clicking rate, there are different implicit signals in the system that need to be considered. For example, watching time length, like rate, users may want to share with friends (social impact)...etc. All these signals could help us in terms of optimizing the ranking system, while sometimes it may bring conflicting objectives as well. For example, we may want users not only to watch a video but also to click the "like" button and share it with their friends.
2. **Implicit selection biases in user feedback (Selection Bias)**: there is often implicit bias in the system as well. For example, users might have clicked and watched a video simply because it was being ranked high, not because it was the one that users liked the most. Another example would be, as a result of the position, users are more likely to click on the first recommendation, even though lower placed videos might yield higher engagement and satisfaction.

For the first problem, different objectives need to be optimized. The exact objective function is not defined but the objective is split into two types of objectives:

1. **Engagement objectives**, such as user clicks, and degree of engagement with recommended videos
2. **Satisfaction objectives**, such as user liking a video on YouTube, and leaving a rating on the recommendation.

For the second problem, how to effectively and efficiently learn to reduce such biases is an open question.

Solutions and Objectives

The model described in this paper focuses on two main goals. A Wide & Deep model architecture was used, which combines the power of a wide model linear model (memorization) with the power of a deep neural network (generalizations). The Wide & Deep model will make a prediction for each of the defined objectives (both engagement and satisfaction). The objectives are divided into binary classification problems (e.g. liking or disliking a video) and regression problems (e.g. the rating of a video). On top of this model, a separate ranking model is added. This is simply a weighted combination of an output vector containing the various predicted objectives. These weights are manually adjusted to achieve the best possible performance of the various objectives. To improve performance, advanced methods such as pairwise or listwise approaches are proposed, but they are not implemented in production due to the increasing computation time.

³ Zhe Zhao et al, "Recommending What Video to Watch Next: A Multitask Ranking System", 2019, <https://daiwk.github.io/assets/youtube-multitask.pdf>

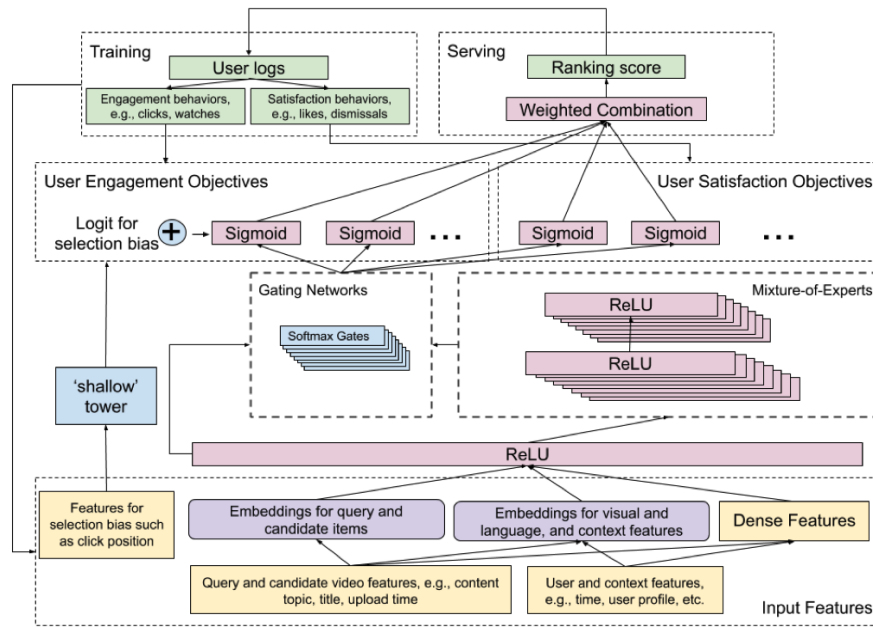


Image 6: Complete architecture of the model

The detailed solutions to these two problems/goals are as follows.

1. Extends the Wide & DeepModel⁴ architecture by adopting Multi-gate Mixture-of-Experts (MMoE)⁵ model for multitask learning

In order to learn and estimate multiple types of user behaviors, this paper used MMoE to automatically learn parameters to share across potential conflicting objectives. The MMoE model is based on efficiently sharing weights across different objectives. The MMoE architecture modularizes the input layer into experts, each of which specializes in different aspects, as the features of the current video (Content, title, topic, upload time, etc.) and the user that is watching (time, user profile, etc.), are used as input. This enhances the representation learned from a complex feature space generated by multiple modalities. The objectives can then choose which experts to share or not share with others by leveraging multiple gating networks.

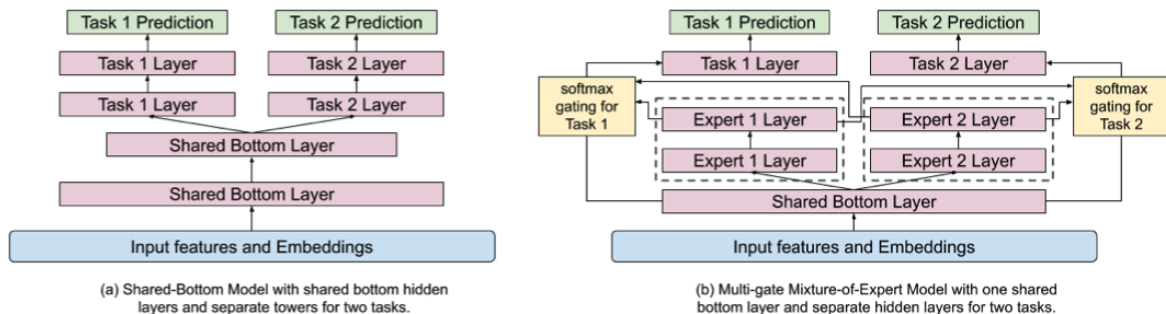


Image 7: Replacing shared-bottom layers with MMoE

⁴ Heng-TzeCheng et al, "Wide & Deep Learning for Recommender Systems", 2016, <https://arxiv.org/abs/1606.07792>

⁵ Jiaqi Ma et al, "Modeling Task Relationships in Multi-task Learning with Multi-gate Mixture-of-Experts", 2018, <https://dl.acm.org/doi/pdf/10.1145/3219819.3220007>

The shared bottom layer is divided into multiple experts, each of which is used to predict the various objectives. There is a gate function for every objective. This gate function is a softmax function with input from the original shared layer as well as the various expert layers. This softmax function will determine which expert layers are critical for the various objectives. From the result (image 8) we know that different experts are more important for different goals. When compared to models with a shared-bottom architecture, training in the MMoE model is less affected if the different objectives have a low correlation.

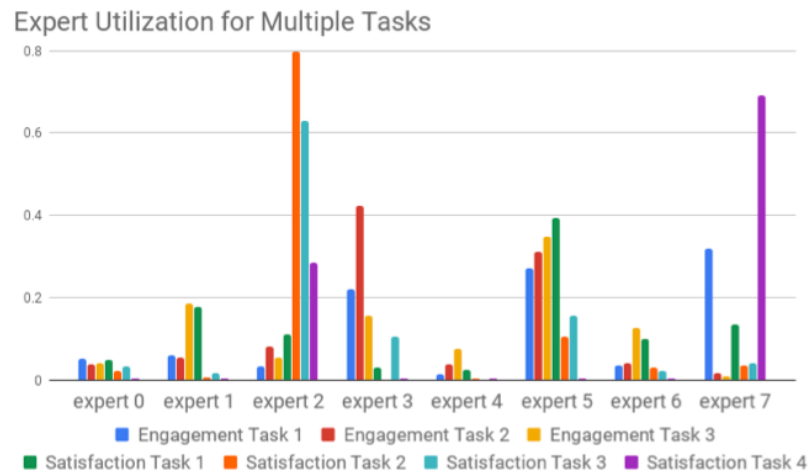


Image 8: Expert utilization for multiple tasks on YouTube

2. Introduces a shallow tower to model and remove selection bias

The model's main focus is on reducing the selection bias (e.g., position bias) in the system caused by the position of the recommended videos. The paper proposes adding a shallow tower, which can simply be a simple linear model with simple features such as the position of videos clicked on and the device used to watch the video, to the main model to model and decrease selection bias using biased training data.

The shallow tower collects input relevant to the selection bias, such as the ranking order decided by the current system, and outputs a scalar that serves as a bias term to the final prediction of the main model, which is a critical component of the Wide & Deep model architecture. As a result, the model will be more focused on the video's position. To prevent the position feature from becoming overly important in the model, a 10% dropout rate is used during training. If you do not use the Wide & Deep architecture and add the position as a single feature, the model may not focus on it at all.

This model architecture factorizes the label in training data into two pieces: the unbiased user utility gained from the main model, and the predicted propensity score learned from the shallow tower.

The results

The findings of this paper show that replacing shared-bottom layers with MMoE improves the model's performance in terms of both engagement (time spent watching recommended videos) and satisfaction (survey responses). Increasing the number of experts in the MMoE and the number of multiplications

improves the model's performance even further. This number cannot be increased in a live setup due to computational constraints.

Model Architecture	Number of Multiplications	Engagement Metric	Satisfaction Metric
Shared-Bottom	3.7M	/	/
Shared-Bottom	6.1M	+0.1%	+ 1.89%
MMoE (4 experts)	3.7M	+0.20%	+ 1.22%
MMoE (8 Experts)	6.1M	+0.45%	+ 3.07%

Table 1: YouTube live experiment results for MMoE

Further results show that using the shallow tower improves the engagement metric by reducing selection bias. This is a significant improvement over simply adding input features to the MMoE model.

Method	Engagement Metric
Input Feature	-0.07%
Adversarial Loss	+0.01%
Shallow Tower	+0.24%

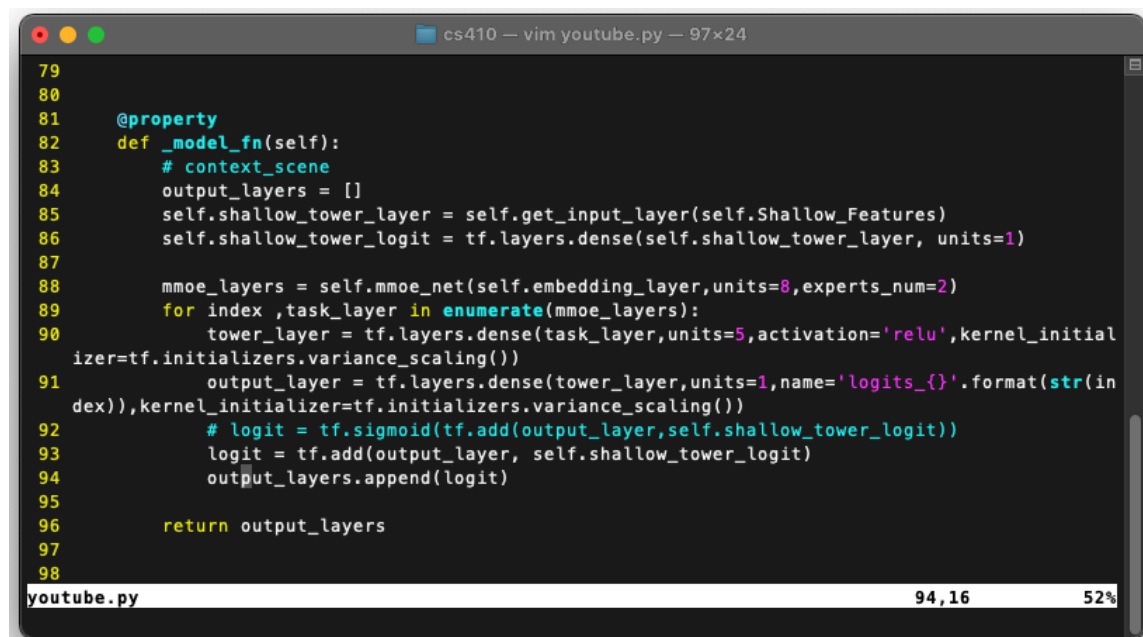
Table 2: YouTube live experiment results for modeling position bias

Python (pseudocode) Implementation

For this python pseudocode implementation, I took references from these two repositories on Github: "[Shicoder/Deep_Rec](#)" and "[QiuruiChen/AMultitaskRankingSystem](#)". The ideal steps of this implementation should be as follows.

1. Generate the candidate models
2. Ranking (should be efficient)
 - Model
 - Input: Given a query, candidate, and context
 - Output: predict the probability of user taking actions such as user clicks, watches, likes, and dismissals
 - Measure
 - Engagement objects, binary classification, user clicks, regress, time spent
 - satisfaction, binary classification, like or not, rating
 - Loss
 - Binary classification: cross-entropy loss
 - Regression task: squared loss
3. Implicit bias steps
 - a. Selection bias (ranking order decided by the current system)
 - b. shallow tower
 - c. a scalar serving as a bias term to the final prediction of the main model

Here I use the dataset from [youtube-dataset/conspiracy](#). The core model structure is as follows, including the shallow_tower_logit for each training task. The first embedding layers input at line 88 is the feature data from the initial data.

A screenshot of a vim editor window titled 'cs410 - vim youtube.py - 97x24'. The window displays Python code for the MMoE model structure. The code is as follows:

```
79
80
81 @property
82 def _model_fn(self):
83     # context_scene
84     output_layers = []
85     self.shallow_tower_layer = self.get_input_layer(self.Shallow_Features)
86     self.shallow_tower_logits = tf.layers.dense(self.shallow_tower_layer, units=1)
87
88     mmoe_layers = self.mmoe_net(self.embedding_layer, units=8, experts_num=2)
89     for index, task_layer in enumerate(mmoe_layers):
90         tower_layer = tf.layers.dense(task_layer, units=5, activation='relu', kernel_initializer=tf.initializers.variance_scaling())
91         output_layer = tf.layers.dense(tower_layer, units=1, name='logits_{}'.format(str(index)), kernel_initializer=tf.initializers.variance_scaling())
92         # logit = tf.sigmoid(tf.add(output_layer, self.shallow_tower_logits))
93         logit = tf.add(output_layer, self.shallow_tower_logits)
94         output_layers.append(logit)
95
96     return output_layers
97
98
```

The status bar at the bottom shows 'youtube.py', '94, 16', and '52%'.

Image 9: YouTube MMoE model structure

For this assignment, I didn't provide the training results since there are still some training problems and bugs in the implementation code of the training stage. Nevertheless, the model structure works well in the validation and testing stages.

Insights and Conclusion

It's very interesting to compare these two papers and see how they conduct the recommendation system in YouTube. I also learned some different techniques in the domain of video recommendation.

Practical insights when doing video recommendation

1. Usually, we only need to calculate the user features once initially. But for video recommendation, we may need to recalculate several times depending on the number of videos.
2. For video recommendation, the basic idea from the first paper is to consider the user-related objectives such as user preference and feedback to certain videos, channels, and topics. From the second paper, we also find there potentially important features such as engagement objects, binary classification, user clicks, regress, time spent satisfaction, binary classification, like or not, rating, etc.
3. It's important to include the number of views of the user on a certain video. So we won't always recommend the same video.
4. According to the first paper, the most important signals are those that describe a user's previous interaction with the item itself and other similar items.
5. From my point of view, the engagement tasks and satisfaction tasks from the second paper represent the objective and subjective feedback. The former is something easier to track as long as the user interacts with the app, while the latter may require more effort and may not be as accurate as the other.

Learned from those models structures

1. You can design a network that predefines some features that are important to you by using a wide and deep model.
2. When you need a model with multiple objectives, MMoE models can be useful.
3. Using shallow tower to solve the Implicit selection bias could be a great game-changer for all types of recommendation tasks.

Future Directions

1. In the first paper, to prevent users from seeing the same recommended video, the model changed the sequential data into random order as features. However, since now we have the Attention Mechanism in Deep Learning, we can probably improve this part in the model.
2. For the second paper, the authors mentioned several future opportunities including (1) exploring new model architecture for multi-objective ranking which balances stability, trainability, and expressiveness (2) Understanding and learning to factorize, and (3) Model compression.

Other Interesting remarks

1. Despite Google's excellent computational infrastructure, they must exercise caution when it comes to training and serving costs.
2. Even with a great and complex model architecture, humans still manually adjust the weights in the final layer, which determines the actual ranking based on the various objective predictions.