Case 1

As the features of the video has been given, we can recommended videos to user based on his history. It will allow us to recommend video that interests user.

Algorithm 1 : (code file saved as Algo1.py)

1) Gather video watched by user and randomly sample other videos from database and keep csv of both of these different.
2) Combine columns of tags, category and title and preprocess the text by performing series of techniques on it.
3) Create TF-idf matrix for the processed text.
4) Convert features of each video into vectorial form using Word2Vec.
5) Calculate Average of Word2Vec vectors with TF-idf (Just take the video vectors and multiply it with their TF-idf scores. Take the average and it will represent video vector.)
6) Calculate cosine similarity between videos that user watched and videos that were randomly sampled from database and store. (As we are using dictionary which stores VideoID as key and video vector representation as value, we can skip when sampled video is same as video from user's history. We can achieve this by comparing their keys).
7) Closer the vectors, more similar they are, recommend videos to the user that have largest 'n' values. (as Cos of less angle is greater than cos of large angle between vectors).

The combination of using Word2Vec with TF-idf is very useful. Over here the TF-IDF of a word acts as a weight for the Word2Vec vector. The Word2Vec model helps in determining the semantics of the word whereas the TF-IDF helps us in determining the weight of a particular word. Thus, the TF-IDF weighted Word2Vec model will successfully identify the importance of the words while it withheld the semantic meaning of the given word.

This model doesn't need any data about other users as the recommendations are specific to the user. This algorithm recommends similar new videos according to user's history that user might like by capturing the interest of the user.

Case 2

In this case, we cannot trust the tags as users might not tag videos properly. In general, the features that makes most sense to humans might not make sense to our Model. In those cases, it is much better for the Model to select what parameters are better to make predictions, i.e., which makes more sense to the Model than to humans. Still, we have to supply the number of latent parameters for which Model will learn the values.

Now, to select the label, we have to give high score to that video that user watched for a longer time than to a video that user has watched for a short period of time. So, a formula to complete this can be (seconds user watched a video) / (total seconds video has). But, when I saw The yovo app, time was not mentioned on UI. So, I assumed that it doesn't use time as it has short videos. Hence, for labeling the watched video, I came with this strategy for The yovo app:

(Badge : the little image which appears on the screen after starting of the video plays, which lets user to continue to the next portion of the video)

1) If a user has not clicked any of the Badge, assign that video a 0. (0 click)
2) If a user has clicked any of the left or right Badge, assign that video a 2.5. (1 click)
3) If a user has clicked remaining left or right Badge, assign that video a 5. (2 click)

So, if a user doesn't like the video, user might not press the tag and would continue to the next video. If user somewhat likes the video, one of the tag will be pressed. If user really enjoyed the video, user will press both of the Badges to watch the full video. In this way, we can label the watched video whether user liked this video or not.

Hence, Deep Neural Network (DNN) can be trained as a regression problem, where the model learns weights to predict the score (0, 2.5, 5) for any user-video pair.

In the proposed Model, I have applied concept of Embeddings. Embeddings are nothing but matrices of weights. They allow us to represent input features as dense continuous vectors (i,e., removes the problems encoding techniques like hot-one-encoding and hence, removes 'curse of dimensionality'). Embeddings also removes the problem of other ML models like tags, category etc by selecting features by itself (although we have to give number of features) which it finds useful. They are also called latent features.

Simpler models can directly compute loss by taking dot product of user embeddings and video embedding to generate label. But to get better behavior, I have constructed a DNN on top of embeddings to get the better behavior (MLP embedding). So, during training, this model learns the parameters of DNN and embeddings (latent features).

This model also incorporates another set of user-video pair embeddings. This set of embeddings is same as matrix factorization (MF) method by taking dot product of user-video pair embedding.

Hence, the proposed DNN combines MF with MLP embedding to get the robust result.

To generate the recommendations, it combines the predictions of both MF and MLP.

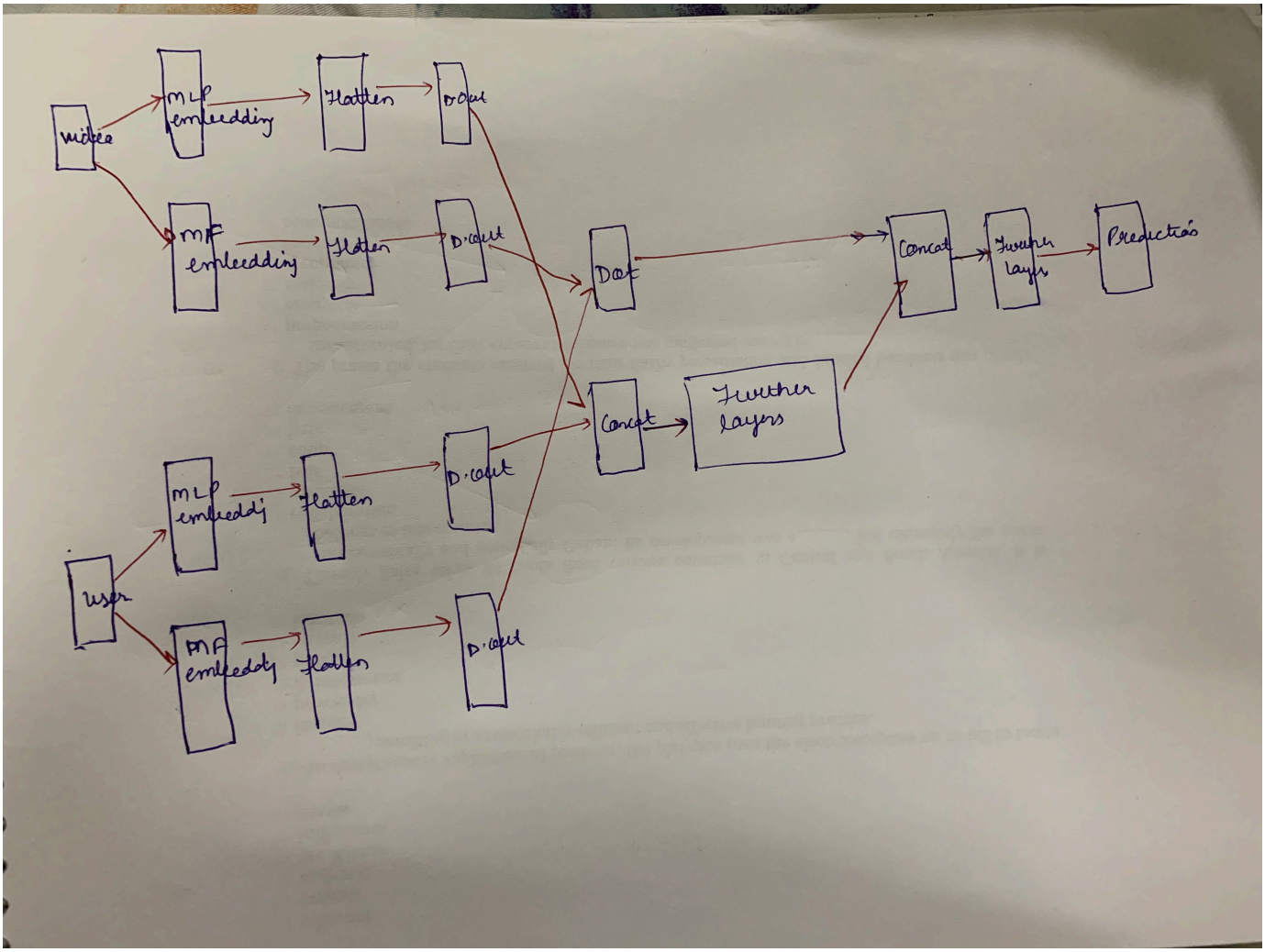
To let the DNN converge at a fast rate, we can help the model by forcing it's output between the range we want. I used softmax layer which takes any real number as input and outputs between 0 and 1. But, as this model uses labels between 0 and 5, I multiplied it's output by 5 to get the output of the model between 0 and 5. If we haven't used this tweak, model would take a lot longer to converge or it might not converge properly.

Algorithm 2 : (code file saved as Algo2.py)

1) Choose latent features of users, videos and matrix factorization.
2) Create embeddings for users, videos and matrix factorization by choosing appropriate input dimensions and output dimensions.
3) Build a DNN on top of it.
4) Start training (to let the model to converge fast, pass the output through sigmoid and multiply the result by 5 for rescaling).
5) Sample videos from database user has not seen and pass them through the model.
6) Give top 'n' recommendations to the user.

Hence, this gives user recommendation for videos which are most popular or liked by the community in general.

Architecture :

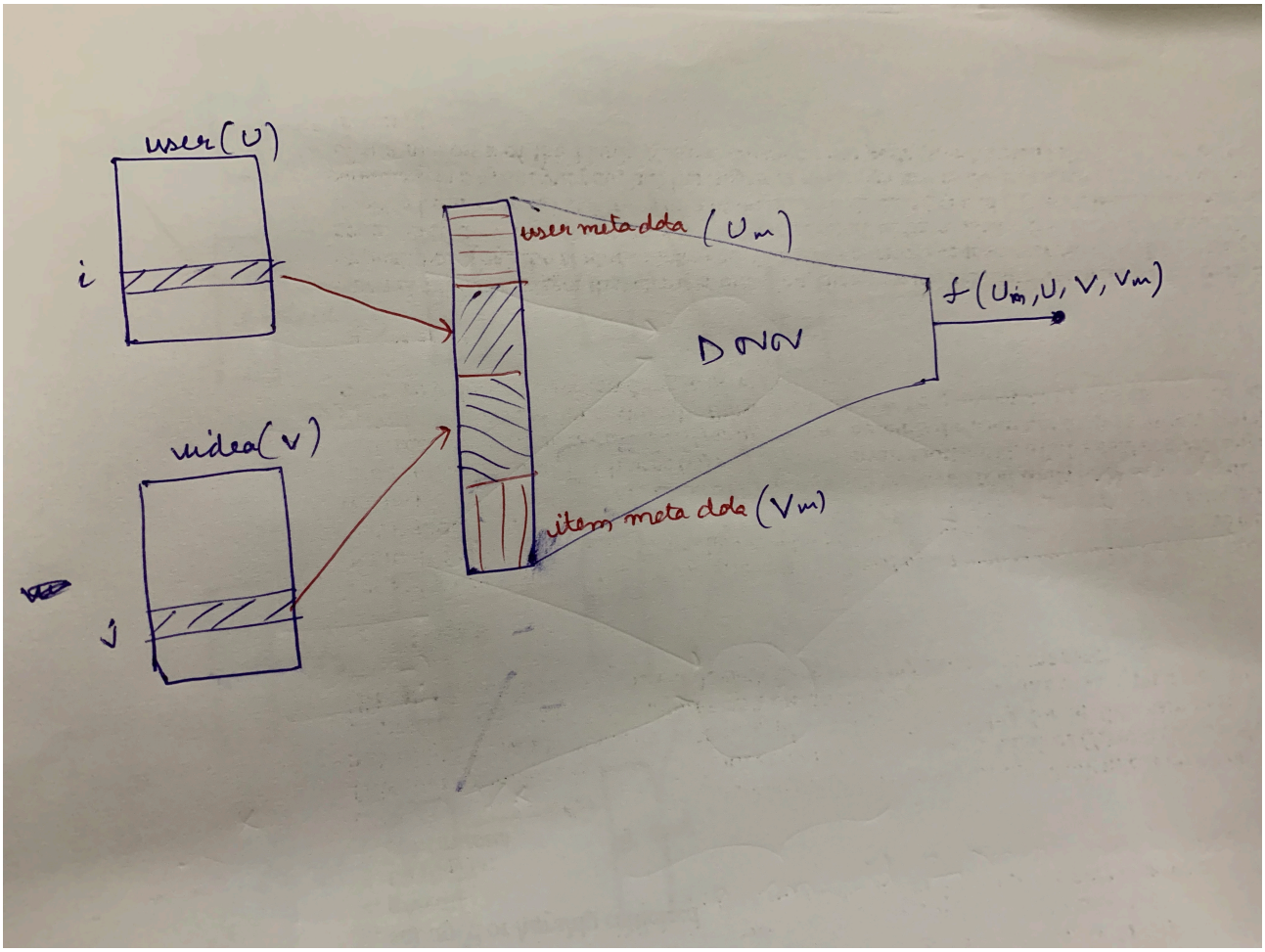


Using embeddings in our model has another advantage. In this question, it is given that tags might be wrong but contex and category is correct. So, we can add contex and category on top of video embeddings and can also add user's metadata on top of user embeddings to get more robust result from the model.

Algorithm 3 : (code file saved as Algo3.py)

1) same as Algorithm 2.
2) Create embeddings for users, videos and also for user's meta, video's meta by choosing appropriate input dimensions and output dimensions.
3) Rest is same as Algorithm 2.

Architecture :

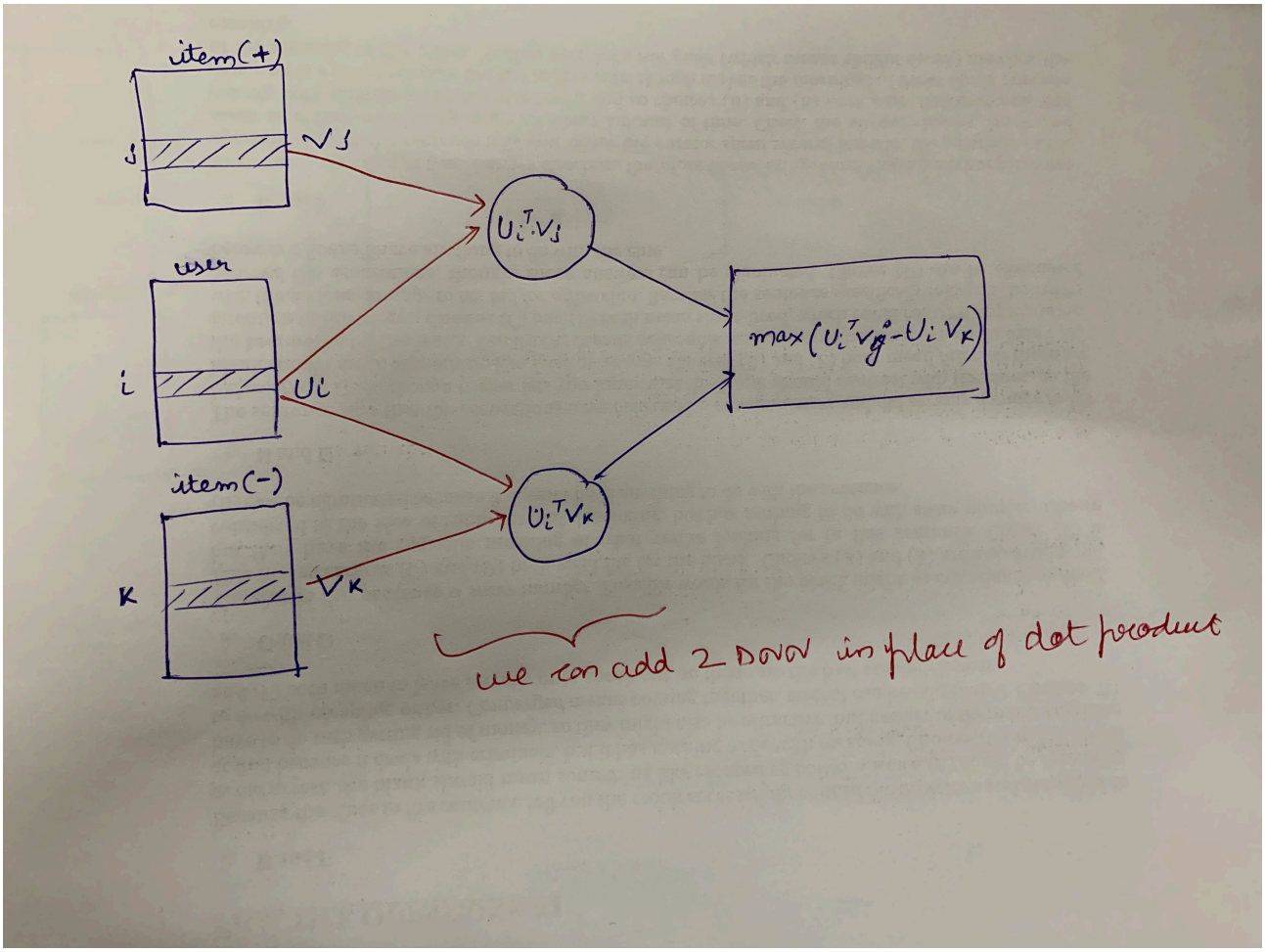


To give the users control, we can ask users for each recommendation we give, whether that recommendation is useful for them or not. Asking from users can be implicit and explicit. We can implicitly see whether users didn't watch the video we recommended with the help of labels we generate (0, 2.5, 5) after user sees the video. If the video we recommended has label 0, then user didn't watch the video properly and hence, didn't like it. Similarly, if the video we recommended has label 5, then user watch the video properly and hence, liked it.

So, we can design another network without adding any parameters or little to the model (as increasing the parameters of the model increases the computation required). We can take the same user-video embedding matrix and feed to the simpler DNN for the videos that had negative feedback from the user. The architecture of such model is given below.

We train both the models and maximize the difference between output from the positive model and negative feedback model.

Architecture:



These types of models tend to recommend movies that are popular (there popularity bias and suffers from cold start problem). These problems are removed by content based recommendation system.

So, to get the best of both the worlds, we can design a model that is a combination of both; content based model and collaborative model. The final ranking can be done to choose recommendations from both of the models. Recommendations from content based filtering will recommend videos that user like and collaborative filtering will give those videos that are popular or will let the user to discover it's taste by recommending videos similar user like him likes the video that this user has not seen yet.