**Recap and Overview**

* Short Recap of what we did in blog post 1 and our goal in this blog post: implement a new DL application for collaborative filtering model for recommendation task - building a transformer from scratch and tuning it.

**Improvements on the baseline deep learning approach of matrix factorization - collaborative filtering model (~ 3 paragraphs)**

* Before jumping into the transformer, we seek to improve a bit on our old DL application of the baseline matrix factorization - collaborative filtering model we built with embedding and dot product layers. We accomplished two improved models with this application
* Tuned matrix factoring algorithm using deep learning, model 1
  + Changes made compared to baseline: add dropout 0.1, concatenate the embedding user and movie layers together, add dropout 0.1, add dense layer with 100 hidden paragraph neurons and relu activation, dropout 0.1, add dense layer with 1 hidden neuron and linear activation
  + Test RMSE result, at which epoch
* Tuned matrix factoring algorithm using deep learning, model 2
  + Changes made compared to baseline: running the output of the dot product through a sigmoid layer and scaling the result using min and max ratings of the data to introduce non-linearity, pulling the embedding layer and reshape operations into a separate class
  + Test RMSE result, at which epoch
* Recap that this serves the purpose to understand the intuition of tuning DL models for collaborative filtering recommendation tasks more as well as to gauge between the “algorithm” and “application” approach which one is a better focus for our project. After realizing that there is more space for learning and challenges with the application approach as we can broaden our understanding of different kinds of DL models and explore the more up-to-date and robust applications of current DL implementations in recommendation systems in the industry, we decide to focus on implementing the transformer as a new application → introduce to next session

**Application: Implementing a Behavioral Sequence Transformer for Collaborative Filtering**

* **Baseline BST Transformer**
  + Short recap of transformer use in recommendation(~ 3- 5 sentence): mention that we are implementing a Transformer to conduct Collaborative Filtering
  + Overview of the BST model, refer to this paper (~1 paragraph): <https://arxiv.org/abs/1905.06874>
  + Overview of how the encoding the input features: read the “Encode input features” in <https://keras.io/examples/structured_data/movielens_recommendations_transformers/> : we transform the movie ratings data into sequences then encode these as the input features for the transformer
    - Sort the ratings data using the unix\_timestamp, and then group the movie\_id values and the rating values by user\_id.
    - Split the movie\_ids and ratings list into a set of sequences of a fixed length. Set the sequence\_length and step\_size to change the length of the input sequence to the model and control the number of sequences to generate for each user
    - process the output to have each sequence in a separate records in the DataFrame and join the user features with the ratings data.
    - Each categorical user/ movie feature is encoded using layers.Embedding, with embedding dimension equals to the square root of the vocabulary size of the feature.
    - Concatenate a multi-hot genres vector for each movie with its embedding vector, and processed using a non-linear Dense layer.
    - Add a positional embedding is to each movie embedding in the sequence, and then multiplied by its rating from the ratings sequence.
    - Concatenate the target movie embedding to the sequence movie embeddings to create a tensor with the shape of [batch size, sequence length, embedding size], as expected by the attention layer of the transformer
    - The method returns a tuple of: encoded\_transformer\_features and encoded\_other\_features.
  + Split: 85% train and 15% test, no validation set
  + Test RMSE result, at which epoch
  + Training loss plot
* **Tuned BST Transformer**
  + Different splitting: train - 80%, val - 10%, test - 10%, needs validation test to do early stopping on validation loss to detect under or overfitting
  + Include more features: user - age, occupation, gender, movie - movie IDs sequence, genre, ratings sequence
  + Tuned parameters: number of attention heads, number of transformer blocks, number of hidden neurons in dense layers, number of dense layers, drop out rate, learning rate, batch size
  + Best hyperparameter combination:
  + Test RMSE result, at which epoch:
  + Training loss plot: describe it a bit, looks great, loss decrease quickly in the beginning and plateau out

**Evaluation and Next Steps**

* **Challenges we faced:** understanding a complicated new transformer, figuring out how to build a transformer from scratch then tune it, understanding more about recommendation tasks - emphasize that we haven’t learn this in class so we have done a lot of self-learning, research, trial and errors on our on
* **Plan:** improve transformer further with more tuning techniques, try different DL applications to build movie recommendation system such as autoencoder, skip-gram, bag of words, etc, for both collaborative filtering and content-filtering