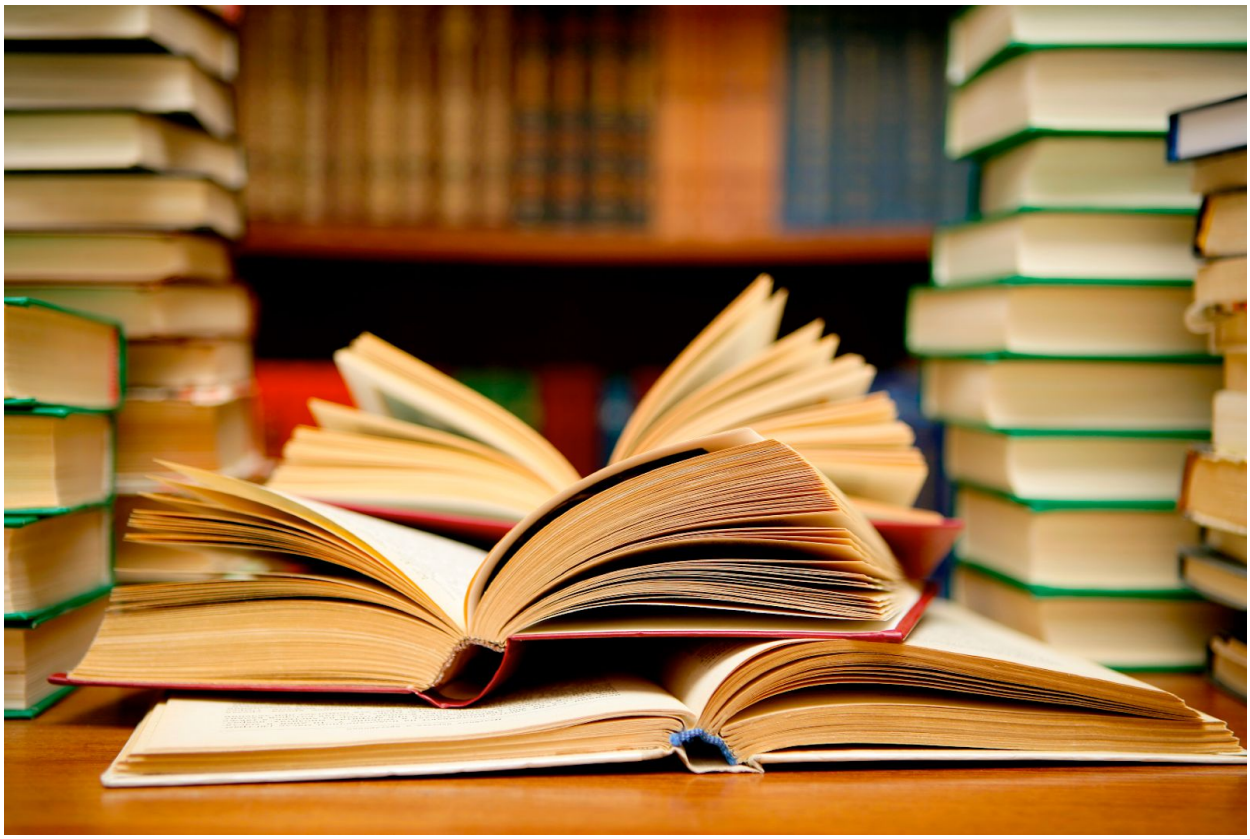
Summary	In this project we perform a comparison of emerging deep learning approaches to recommendation with classic and current state of the art approaches using the example of personalized book recommendations.
URL	https://github.com/manoj0237/DeeplearningandAI-finalproject
Category	Deep learning
Environment	Python 3, Keras, Surprise, Scikit-Learn
Status	In progress
Feedback Link	
Author	Manoj Karru, Nishant Gandhi, Emily Strong

Personalized Book Recommendations



04.14.2019

Emily Strong
Manoj Karru
Nishant Gandhi

Overview

Many companies face the challenge of providing personalized recommendations to users based only on the users' interactions with content on the site. Two classic approaches to this problem are logistic regression and collaborative filtering. Recent advances in deep learning have introduced counterparts to these approaches: collaborative deep learning and wide and deep learning. There is the question however of how these emerging techniques compare to the current state-of-the-art contextual multi-armed bandit approach. In this project we will compare the performance of these five algorithms in the problem of personalized book recommendations.

Goals

Evaluate the performance of emerging deep learning approaches to recommendation against classic and state-of-the-art techniques. The techniques we investigate are:

- Collaborative Filtering
- Logistic Regression
- Contextual Multi-Armed Bandits
- Collaborative Deep Learning
- Wide and Deep Learning

Process

1. Literature Review
2. Data Wrangling
3. Exploratory Data Analysis and Feature Engineering
    - Text features of the books are embedded using TF-IDF and their latent features extracted with NMF
    - User-book interaction history will be used to generate additional features with NMF
4. Development of models and comparison of approaches
5. Deploy notebook with best model

Milestones

Timeframe	Delivery
April 11-19	Exploratory Data Analysis & Feature Engineering

April 11-20 Literature Review, Model Building and Preliminary Training

April 21-24 Model Tuning and Evaluation

April 25-26 Final documentation and deploy best models

Deployment Details

1. Language: Python
2. Frameworks: Keras, Scikit-Learn, Surprise
3. NLP Tools: TF-IDF, latent features with NMF
4. Demo Deployment: Jupyter Notebook on AWS

Methodology

Data Set

Data set: https://www.kaggle.com/zygmunt/goodbooks-10k

The GoodReads data set is scraped from public data on GoodReads.com. The data set is for the most popular 10k books on the site, with over 6 million user ratings. It includes book metadata and tags.

The public data on GoodReads does not contain any user data beyond their interaction history, so we will need to engineer features for the models that require them.

For the rating data we removed all ratings from users 1-5 to use for cold-start evaluation. For the remaining 53419 users, we performed a randomized 70-30 train-test split. Both sets contain data for each warm-start user.

Feature Engineering

For each book, the GoodReads data set contains:

- Book ID
- Number of editions
- Authors
- Original publication year
- Original title
- Edition title with series name
- Language
- Average rating
- Number of ratings for the edition
- Aggregate number of ratings for all editions

- Number of text reviews for the edition
- Separate counts for each of the rating options (1-5)
- Tags

We one-hot encoded the author and language and used TF-IDF to embed the edition title, original title, and tags. We then used non-negative matrix factorization to extract 25 latent features from these to model the topic of the work. With the scaled numeric features, this gives 36 books features.

For user features, we calculated the average rating from each user in the training set. We then identified all of the books that the user had rated 4 or 5 stars. For those books, we averaged the scaled numeric features and 500 components from the embedded features that explain 78% of their variance. We then extracted 25 latent features from these averages to model the user's book preferences.

Model Evaluation

We will evaluate each model based on replay with the historic data, a method of evaluation in which only those recommendations that have historic ratings are scored. In this method, the model makes recommendation for an historic test set, and the top-k recommendations are evaluated.

The specific metrics we will use are mean average precision for the top-k recommendations, and average precision and recall for all users for all of their ratings in the test set. We will look at the top 10 recommendations, similar to what might be displayed for a website user.

We will look at both metrics for predicting exact rating, as well as binarized scores defined as whether or not the user enjoyed the book. For models trained on the 5 star data, we will calculate a binarized mAP after prediction to directly compare the performance of these models to the models trained on binary data.

Collaborative Filtering

Collaborative filtering is a classic approach to recommendation in which similarity of items or users is the basis of the recommendation. A common user-user technique is K-Nearest Neighbors, where the ratings for all items are projected onto an n-dimensional space and the k neighbors are selected based on cosine similarity. In 2007, a matrix-factorization approach emerged as a winner of a contest run by Netflix. This approach uses singular value decomposition to model the items based on their ratings. Both of these approaches use only the user-content interaction history and cannot make recommendations to new users since they do not have historic data for the similarity comparisons. A common

solution to this is to require new users rate a few items before the system makes recommendations.

We use the Surprise library to evaluate both approaches with our data set for a baseline comparison. The Surprise provide multiple algorithm packages, among which we used SVD for matrix factorization approach & KNNBasic for K-Nearest Neighbours approach.

An example training input:

User ID	Book 1	Book 2
1	5	1
2	0	3
3	1	0

Logistic Regression

Logistic regression is commonly used in industry due to its simplicity and effectiveness. For logistic regression we will combine our engineered user features and content features to train a logistic regression using scikit-learn that will predict how the user would rate the content.

The logistic regression uses a combination of user and content features to make predictions for the given user for the given content. Both of these can be generated from non-interaction data depending on the data available, which reduces the impact of cold starts. However for our data set all of our user features will need to be generated from the interactions. We will thus need to use a vector of zeros to represent the user features and the output will be the default output of the logistic regression.

An example training input:

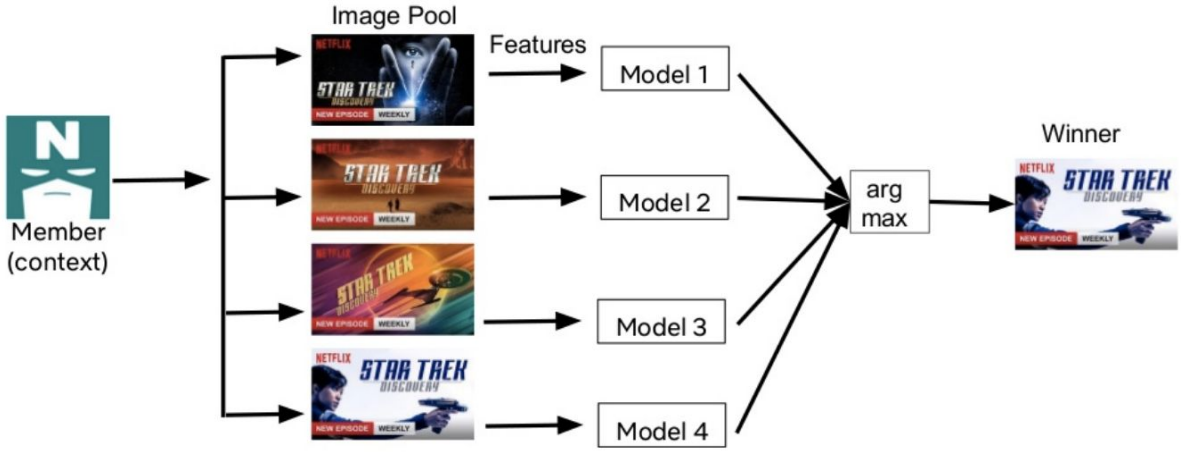
User Features	Content Features	Rating
User 1 Features	Book 1 Features	5
	Book 2 Features	3
	Book 3 Features	1
User 2 Features	Book 1 Features	3
	Book 2 Features	2

	Book 3 Features	1

Contextual Multi-Armed Bandits

Contextual multi-armed bandit algorithms are the current state-of-the-art family of algorithms used in recommendation problems. These are reinforcement learning algorithms that leverage probabilistic and regression techniques to address the exploration-exploitation tradeoff. Since none of the other methods we are using involve reinforcement learning, we only perform an offline experiment.

To given an example of how this works, Netflix (PRS Workshop 2018) uses contextual multi-armed bandits to select what artwork to show to customers for each movie displayed as shown in the figure below. Each user has a unique context of preferences that they feed into models trained for each piece of artwork. These models would predict the probability of the user playing the movie. The maximum predicted value is chosen, but will have an exploration component to it. This may be incorporated directly into the model output or might be a technique such as an epsilon greedy approach in which with probability epsilon a random option is explored, and with probability 1 - epsilon the max is greedily selected.



Source: https://www.slideshare.net/JayaKawale/a-multiarmed-bandit-framework-for-recommendations-at-netflix

The specific algorithm we used is LinUCB (Chu 2011). This algorithm trains a ridge regression for each item available for recommendation, calculates a predicted value for the given user and uses the covariance to calculate the upper confidence bound on the prediction. The item with the highest predicted upper confidence bound is recommended.

Since a ridge regression is used, new items will still have models initialized with an identity matrix, and new users would receive the default recommendations.

Algorithm 1 LinUCB: UCB with Linear Hypotheses

Inputs: $\alpha \in \mathbb{R}_+, K, d \in \mathbb{N}$
$A \leftarrow I_d$ {The d-by-d identity matirx}
$b \leftarrow \mathbf{0}_d$
for $t = 1, 2, 3, \ldots, T$ **do**
  $\theta_t \leftarrow A^{-1}b$
  Observe K features, $x_{t,1}, x_{t,2}, \cdots, x_{t,K} \in \mathbb{R}^d$
  for $a = 1, 2, \ldots, K$ **do**
    $p_{t,a} \leftarrow \theta_t^\top x_{t,a} + \alpha\sqrt{x_{t,a}^\top A^{-1} x_{t,a}}$ {Computes upper confidence bound}
  end for
  Choose action $a_t = \arg\max_a p_{t,a}$ with ties broken arbitrarily
  Observe payoff $r_t \in \{0, 1\}$
  $A \leftarrow A + x_{t,a_t} x_{t,a_t}^\top$
  $b \leftarrow b + x_{t,a_t} r_t$
end for

This is a simple ridge regression with the exploration calculated using the covariance matrix A.

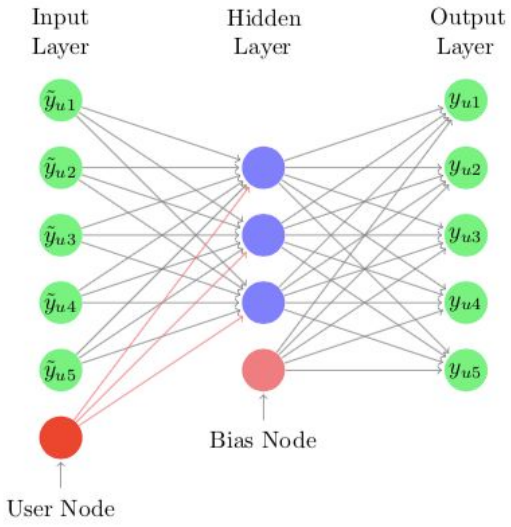
An example training input:

User Features	Book ID	Rating
User 1 Features	1	5
	2	3
	3	1
User 2 Features	1	3
	2	2
	3	1

We also experimented with a variant called hLinUCB (Wang 2016) that models the hidden features of the user-item interaction, however after getting odd results we looked at the authors' code and found their implementation differed from what is in their paper in non-trivial ways and decided not to pursue it further since multi-armed bandits is not the focus of this project.

Collaborative Deep Learning

For collaborative deep learning we used Collaborative Denoising Auto-Encoders (Wu 2016). This is a technique that treats the missing ratings in the interaction data as noise and uses auto-encoders to “denoise” them and predict their values. CDAE further attempts to improve the predictions by injecting latent features of the user.



The authors have released a dataset-specific version of their code in Tensorflow which we used as a reference for how to design the architecture, though our implementation is in Keras.

Our implementation differs from theirs in two ways. First, they train separate weights for the user features of every user. The reasoning is that each user has unique interactions with the content. However, their argument does not adequately address the fact that leveraging user similarity is a fundamental aspect of collaborative filtering and recommendation in general, so we instead implemented shared weights for the users. Second, they used a single hidden layer however many papers (e.g. Kuchaiev 2017) have demonstrated the effectiveness of using multiple hidden layers in improving recommendations so we experimented with addition additional layers in our tuning.

The latent features of new users are represented by a vector of zeros. Because we had to use the full data set as the denoised data, we performed a 70-30 split of the users and used the training data of the 70% as the “noisy” training data and used the training data of the 30% as the “noisy” test data. As a result of this, we indirectly tested how well the model performs with users it has never seen before that do have latent features available.

An example training input:

User Features	Book 1	Book 2
User 1 Features	5	1
User 2 Features	0	3
User 3 Features	1	0

Wide and Deep Learning

Our final approach is the wide and deep learning algorithm (Cheng 2016) developed by engineers at Google. This combines a generalized linear model with a feed-forward network to create an ensemble method. The inputs of the two models are combined and run through a logistic regression to make the final prediction. The architecture is designed for binary classification so the weighted sum of the two is taken in that version, however for our 5 star rating prediction we had to concatenate the outputs instead.

In the original implementation, for the generalized linear model the cross product of binary features was calculated to capture interactions and the feedforward network included an embedding layer. Since we are using latent features instead of categorical features, we are omitting these steps.

Features for new users are represented by a vector of zeros.

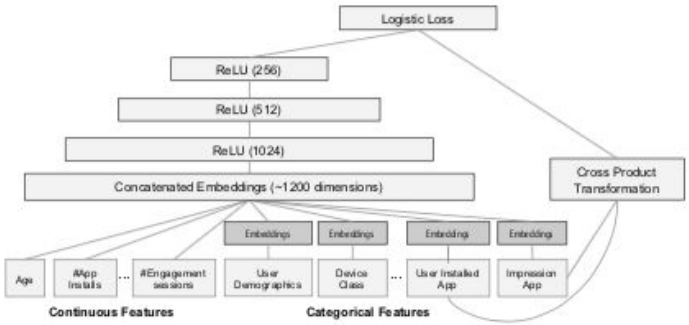


Figure 4: Wide & Deep model structure for apps recommendation.

The authors have released a [dataset-specific version] of their code in Tensorflow which we used as a reference for how to design the architecture, though our implementation is in Keras.

An example training input:

User Features	Content Features	Rating
User 1 Features	Book 1 Features	5
	Book 2 Features	3
	Book 3 Features	1
User 2 Features	Book 1 Features	3
	Book 2 Features	2
	Book 3 Features	1

Results & Discussion

Best Results for Each Algorithm:

Algorithm	**Test mAP** [*binary mAP*]	**Test Precision**	**Test Recall**	**Cold Start mAP**	**Cold Start Precision**	**Cold Start Recall**
Collaborative Filtering (KNN)	0.0841 [0.6033]	0.5071	0.4214	N/A	N/A	N/A
Collaborative Filtering (SVD)	0.0949 [0.5845]	0.5376	0.4738	N/A	N/A	N/A
Logistic Regression	0.646	0.759	0.897	0.7493	0.5519	0.9443
LinUCB	0.6075	0.7478	0.9154	0.3967	0.7178	0.6149
CDAE	0.6628	0.7989	0.0378	0.52	0.7344	0.1638
Wide & Deep	0.5929	0.7604	0.9023	0.6279	0.5580	0.9547

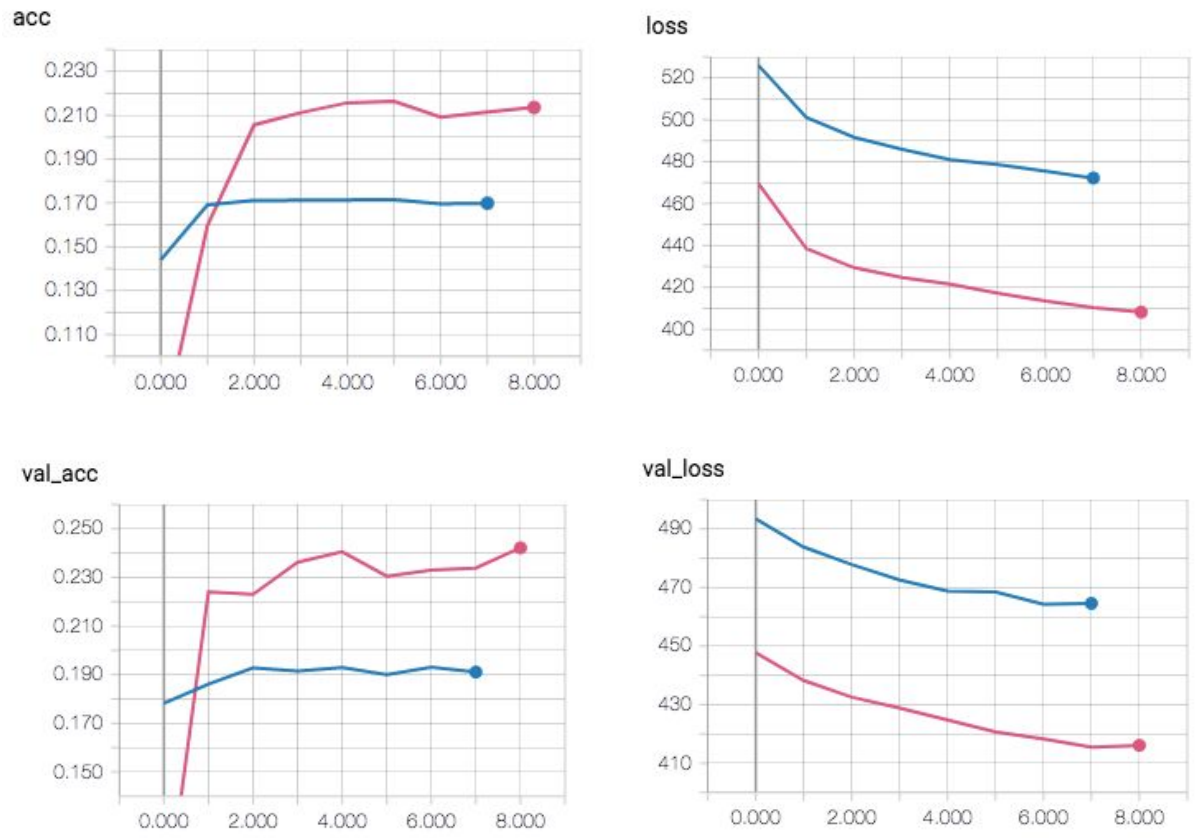
Full results can be viewed at:
https://docs.google.com/spreadsheets/d/1dufvigG9mKmshKDB6gZJ1GuJgwkdFBMt6j2nRqLKnVI/edit?usp=sharing

Our results reflect why logistic regression is so commonly used in industry. Without any tuning, it performed as well as our best deep learning models (see architecture diagrams

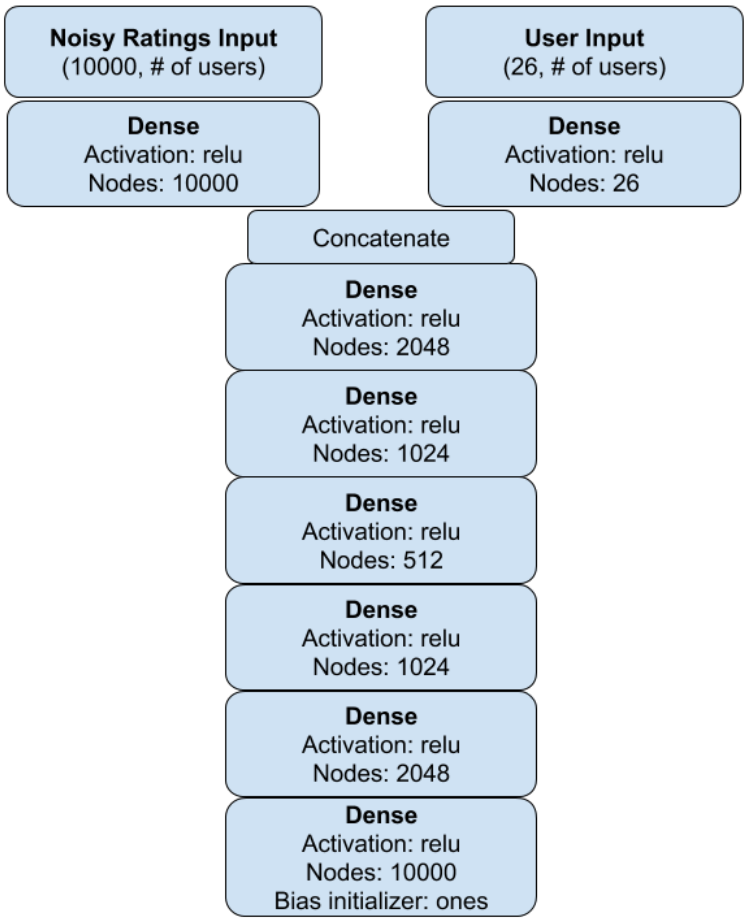
below) in precision and mAP, and as well as the Wide & Deep architecture in recall. It also gave the best results with cold start users.

Logistic regression, LinUCB and the Wide & Deep model all had higher recall than precision suggesting that they may be over-predicting likes. However, they all had acceptable precision and mAP indicating they are still performing satisfactorily. The collaborative denoising autoencoder on the other hand had a very low recall indicating it has a high bias in its predictions.

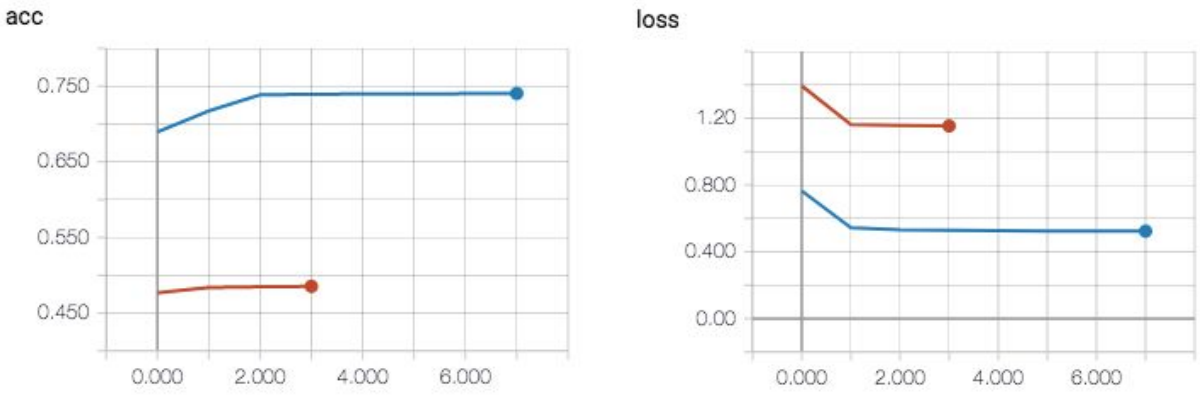
Interesting, when we looked at the binarized mAP for our 5-star models, the values were consistent with the mAPs for models trained directly on binary data. What this suggests is that both models are capturing the same interaction between user while the 5-star models are not able to distinguish preferences at a more granular level (1 vs 2, 4 vs 5).

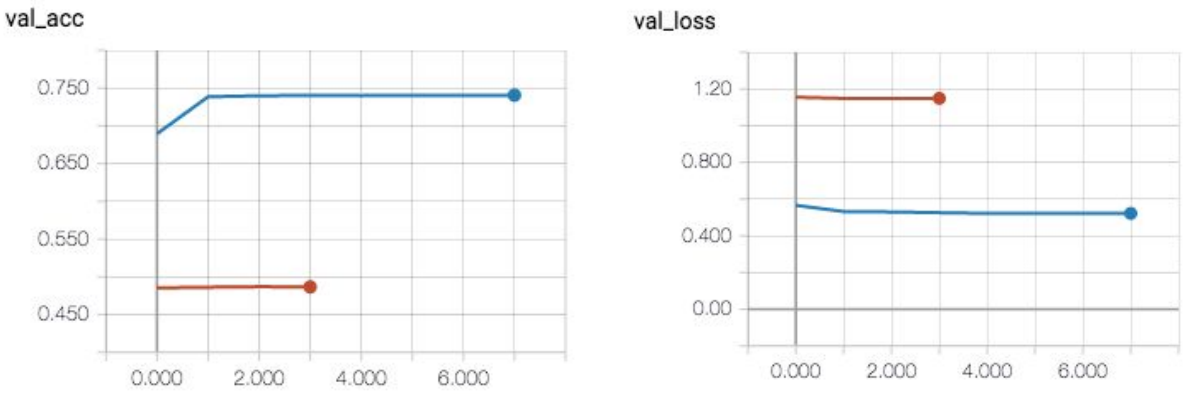


CDAE Best Models Training Plots. Pink: Binary prediction. Blue: 5 Star prediction.

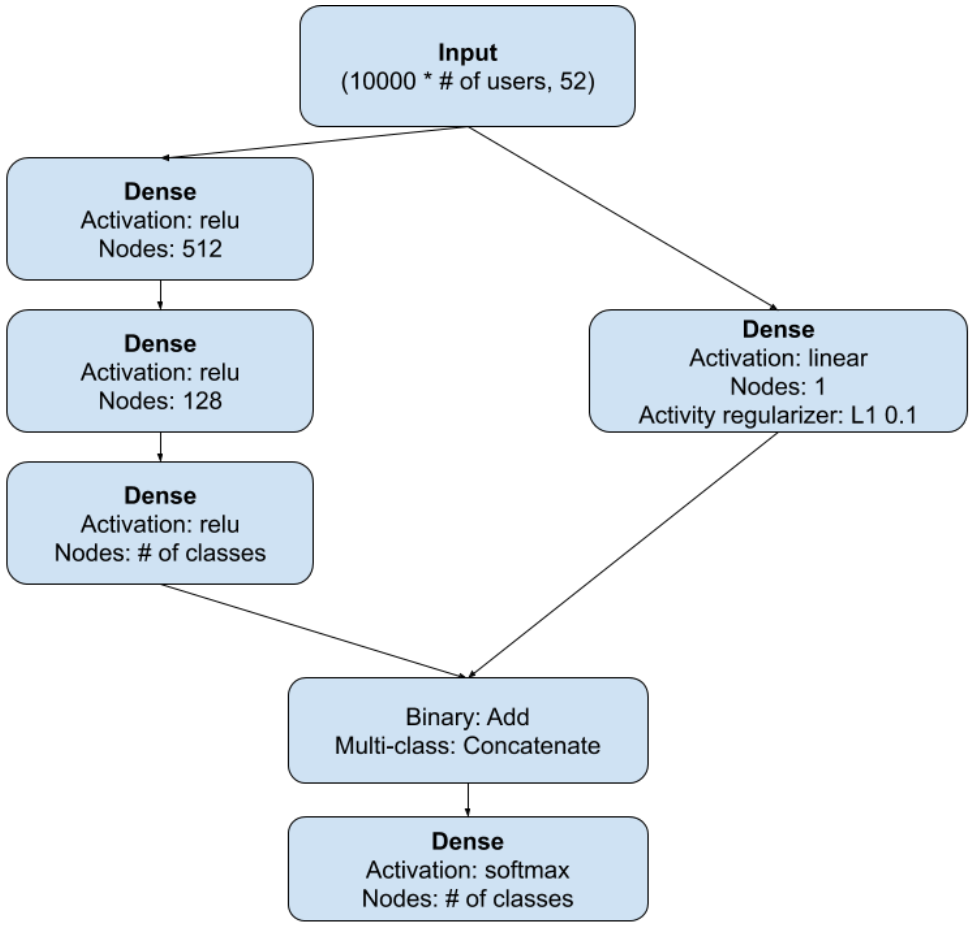


CDAE Tuned Architecture. Trained with Adagrad optimizer and a batch size of 128.





Wide-Deep Best Models Training Plots. Blue: Binary prediction. Brown: 5 Star prediction.



Wide and Deep Tuned Architecture. Trained with Adagrad optimizer and a batch size of 128.

References

Cheng, Heng-Tze, et al. "Wide & deep learning for recommender systems." *Proceedings of the 1st workshop on deep learning for recommender systems*. ACM, 2016.

Reference code: https://github.com/tensorflow/models/tree/master/official/wide_deep

Chu, Wei, et al. "Contextual bandits with linear payoff functions." Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011.

Kawale, Jaya and Fernando Amat. "A Multi-Armed Bandit Framework for Recommendations at Netflix." PRS Workshop, June 2018. https://www.slideshare.net/JayaKawale/a-multiarmed-bandit-framework-for-recommendations-at-netflix

Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37.

Kuchaiev, Oleksii, and Boris Ginsburg. "Training deep autoencoders for collaborative filtering." *arXiv preprint arXiv:1708.01715* (2017).

Wang, Huazheng, Qingyun Wu, and Hongning Wang. "Learning hidden features for contextual bandits." *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2016.

Wu, Yao, et al. "Collaborative denoising auto-encoders for top-n recommender systems." *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 2016.

Reference code: https://github.com/gtshs2/Collaborative-Denoising-Auto-Encoder