Deep Cooperative Neural Network (DeepCoNN) for Movie Rating Prediction

Colby Wise
Columbia University
cjw2165@columbia.edu

Michael Alvarino
Columbia University
maa2282@columbia.edu

Richard Dewey
Columbia University
rld2126@columbia.edu

Abstract

For our project we implemented the Deep Cooperative Neural Network (DeepCoNN) architecture outlined in the arXiv working paper "Joint Deep Modeling of Users and Items Using Reviews for Recommendation" (Zheng, Noroozi, et. al. 2017) to predict user movie ratings from user review text data. The DeepCoNN methodology models users and items jointly using review text in two cooperative neural networks. One network learns feature representations from groupings of users and their reviews, while the other learns features from items and user reviews; both networks share a final interaction layer. The following paper describes our implementation of DeepCoNN architecture using movie review data and presents our results.

Project GitHub Repository

1. Introduction

The literature on recommendation systems is deep and until recently has traditionally focused on well-known matrix factorization algorithms, such as collaborative filtering (CF), as popularized by Netflix and Spotify. Collaborative filtering is advantageous to other methods because it is relatively easy to implement and computationally efficient. On the other hand, the drawbacks of collaborative filtering in real-world applications lead to at times insurmountable challenges. To clarify, CF suffers from the "cold-start" problem in that it requires user ratings to make predictions. When users have no or limited prior ratings history relative to the total amount of items to rate this is called data *sparsity*. For sparse user data it's difficult to accurately predict ratings/recommendations thus leading to poor model performance and generalization [9].

Recently strong empirical results in Deep Learning particularly in natural language modeling via convolutional neural networks (CNN) and recurrent networks (RNN) to capture complex feature interactions in textual data have lead recent recommendation systems ($RecSys$) research leverage incorporate deep learning. CNN models [1] and RNNs [3], such as *Long Short-Term Memory* (LSTM) and *Gated Recurrent Units* (GRU) have shown the ability to help with the generalization problem and improve model accuracy in textual data sets.

1.1. Related Work

Our experiments focus on expanding the $DeepCoNN$ model thus our primary reference will be the original *Joint Deep Modeling of Users and Items Using Reviews for Recommendation* paper. While the authors do not provide an open-source code repository on-line, after reaching out them via email they provided guidance on model replication. Additionally, we found a similar code base on-line that utilized video game review data [8]. Related papers we referenced for re-constructing $DeepCoNN$ are $DeepFM$ [4] which similar to our model, combines a form of matrix factorization to capture user/item interactions with a CNN architecture; $TransNets$ [3], which extends $DeepCoNN$ to examples where the users review is not available; and for background on factorization machines [2].

Our reading of other papers that jointly train multiple neural networks such as *Cooperative Neural Net Ensembles* [9] and *Collaborative Recurrent Neural Networks for Dynamic Recommender Systems* [10] also informed our research. In particular we drew inspiration from papers that used more advanced network architectures such as *A Recurrent Neural Network Based Recommendation System* [11] which for instance use LSTM and GRU architectures to predict user preferences on $Yelp$ and $Trip\text{-}Advisor$ from text reviews. This led us to extend the DeepCoNN network by testing RNN architectures such as (LSTM | GRU), which both provided better out of sample performance. Finally, research on word embeddings in NLP such as *A Comparative Study of Word Embeddings for Reading Comprehension* [12] helped inform our use of pre-trained word embeddings.

It should be noted that our original proposal was to replicate Google's *Wide & Deep* model [5]. We spent a considerable amount of time going down this path, but ultimately came to the conclusion that we could not source similar high quality categorical data as per the original paper. Furthermore, we decided to switch to the $DeepConn$ model due to it's greater relevancy to our data set and given the success of

the paper which was recently accepted into the 2017 ACM International Conference on Web Search and Data Mining ($WSDM$). Lastly, we found the paper to be more innovative and challenging.

2. Methodology

This section outlines the methodology used in the project including an overview of the problem formulation, an overview of our data set and preprocessing, and details of model architectures.

2.1. Problem Formulation

In order to make better movie recommendations how can we more accurately predict a users rating for an unseen movie based on what the user has previously seen? Furthermore, can we improve generalization of our model when information on a users past movie ratings is limited by utilizing user review data?

These two questions are the crux of our problem. Prior to deep learning, standard approaches for recommendation systems ($RecSys$) used collaborative filtering which relies on decomposing users, items (i.e. movies), and ratings into latent feature matrices. The interaction between users and items is captured by matrix multiplication of the weight matrices of these latent features. One common CF method includes using the cosine similarity measure between all pairs of movies that users have rated:

Where, m_i and m_j refer to movie vectors of ratings of users who have rated both movies:

$$cos(\theta) = \frac{\vec{m_i} \bullet \vec{m_j}}{||\vec{m_i}||_2 \times ||\vec{m_j}||_2}$$

This yields a movie-to-movie similarity matrix of dimensions $M \times M$ with ones along the diagonal. Thus, the predicted rating for movie m_2 for $user_1$ would be calculated using similarity measures between (m_2, m_1) and (m_2, m_3) weighted by the respective ratings for m_1 and m_3.

From the above formulation we can clearly see the major disadvantage of this approach: $sparsity$. When ratings are limited the movie-to-movie matrix is mainly zeros thus limiting predictive ability. Current research like $DeepCoNN$ have attempted to improve accuracy of sparse data by using text data that users write after watching movies. This text data is used as input for training neural networks and offers additional insights versus only numerical ratings.

2.2. Data Overview & Preprocessing

The Amazon Instant Video Review 5-core data is a JSON file with nine values per entry of which we use the following four:

	# Reviews	# Users	# Movies	Training%	Test%
Data	18,563	5033	1685	90%	10%

Exhibit 1. Amazon Instant Video Dataset Overview

ReviewerID - *user ID*
Asin - *movie ID*
ReviewText - *review text*
Overall - *movie rating*

2.2.1 Grouping Users & Reviews

The DeepCoNN model formulation that we'll focus on in this research uses two jointly modeled neural networks. The first network Net_i uses all of the text reviews for a given user. The second network Net_p uses all of the text reviews for a particular item (movie). To replicate the paper, we group users with all their reviews and movies with their reviews into two dictionaries such that each key in the dictionary is a unique $reviewerID$ ($Asin$) accompanied by a list of text reviews.

2.2.2 Data Preprocessing

Using Keras built-in preprocessing API each review is tokenized into separate lowercased words with all punctuation removed. Then we use our embedding mapper to transform all reviews into their GloVe vector representation. Given review lengths typically differ in the number of words per review, prior to aggregating all user reviews and movie reviews into input matrices we pad ($truncate$) reviews such that input matrices are of equal dimensionality. It's important to note that our original data set includes 37,000 ratings however of these there are only 18,563 text reviews. Given we are training our model on text reviews, we limit our data set to only ratings that have reviews thus truncating our data set down to 18,563 from 37,000. The 18,633 unique reviews are comprised from 5033 unique users and 1685. Hence on average there are 3.70 reviews/user and 11 reviews/movie. Lastly we split this data set into a randomized training and test using a 90%, 10% split. *Exhibit 1* summarizes this.

2.2.3 Pre-trained Embeddings

Word embeddings map words in our review text to n-dimensional numerical vectors. The paper does not explicitly state a standard word embedding method i.e. (GloVe, Word2Vec, etc) thus we decided to use Global Vectors for Word Representation (GloVe.6B) 50-dimensional and 100-dimensional pre-trained embeddings [10]. GloVe is a log-bilinear model with a weighted least-squares objective. Intuitively, the model observes that ratios of word-word co-occurrence probabilities have the potential for encoding

some form of meaning. Thus GloVe embeddings help capturing text structure from our review data. The advantage of using pre-trained embeddings vs learning the embeddings from the data is that one can shorten model training times while potentially increasing the quality word-vector representations in the model. The disadvantage is that our text corpus may significantly differ from the corpus used to train GloVe.

2.3. Model Architectures

CNNs and RNNs are amongst the top performing techniques across a range of NLP tasks such as sentiment classification, question-answering, etc. Given there success for text-based modeling we focused on trying to improve the DeepCoNN-DP model using these models. In the subsections below we outline each type of model used on our experiments. Prior to that, we provide a detailed overview of the original $DeepCoNN$ model presented in the paper.

The baseline architecture of $DeepCoNN$ is depicted below and taken from the original research paper. As previously stated, $DeepCoNN$ utilizes two parallel neural networks trained in tandem. One network learns latent factors representations for users from reviews and the other learns factors for movies.

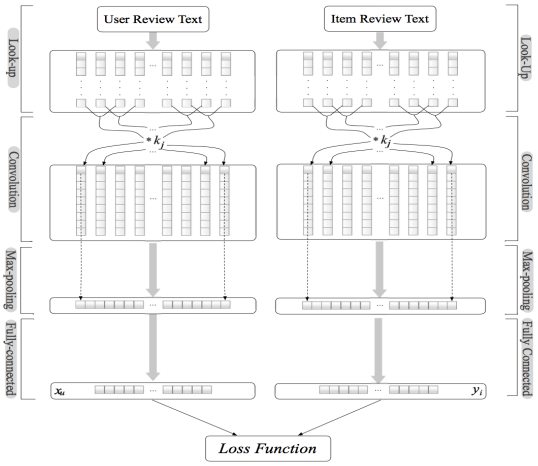


Figure 1: The architecture of the proposed model

Note that in the original paper the first layer of the network is a "lookup" layer that transforms review text into embeddings. As discussed above, we create embedding prior to feeding into the network thus we omit a explicit "lookup" layer. Since the embedding layer uses pre-trained embedding and this is non-trainable our method is similar. The table on the following page summarizes our baseline DeepCoNN-DP model.

To train two networks simultaneously using a single loss function, the paper combines the outputs of both networks by concatenation. From here the interaction of user review features with movie review features is done via a *factorization machine* (FM) of which the details are not provided. However, the goal of the FM is to capture second-oder interactions between users and movies. The $DeepCoNN$ loss function including FM:

Baseline DeepCoNN-DP Model	
CNN Layer	1
$HiddenSize$	64
$Filters$	2
$KernelSize$	8
$Strides$	6
$Activation$	ReLU
Max Pooling	1
Flattened	1
Fully-Connected	1
$HiddenSize$	32

Figure 1. DeepCoNN Baseline Architecture

$$\textbf{DeepCoNN-FM} = \hat{\beta}_0 + \sum_{i=1}^{|\hat{z}|} \hat{w}_i \hat{z}_i + \sum_{i=1}^{|\hat{z}|} \sum_{j=i+1}^{|\hat{z}|} \langle \hat{v}_i, \hat{v}_j \rangle \hat{z}_i \hat{z}_j$$

Where:
$\hat{\beta}_0$ is the global bias
$\hat{w}_i$ models strength of i^{th} variable in $\hat{z}$
$\langle \hat{v}_i, \hat{v}_j \rangle$ is the 2^{nd} order interaction

After contacting the authors, we were told that taking the dot product of the output from the fully connected layers of Net_u and Net_i should closely resemble the factorization machine approach and results focusing on first-order interactions. These results are further discussed in the conclusions and results section. The loss function for the dot product approach is defined as follows:

$$\textbf{DeepCoNN-DP} = \hat{\beta}_0 + \sum_{i=1}^{|\hat{z}|} \hat{w}_i \hat{z}_i + x_u^T x_i$$

Optimization in the paper is done via $RMSprop$ which we replicated. However in further experiments we switched to $Adam$ after finding it led to similar results. Hence we decided to use it in all experimental architectures to ensure consistency between models. $Adam$ is an adaptive version of gradient descent that controls the step size with respect to the absolute value of the gradient. Lastly, note that the paper did not mention of the use of regularization.

2.3.1 CNN Architecture

CNNs are very popular within the image processing field and its applications. For text applications, similar to image

	CNN	LSTM	GRU
GloVe Embedding	50 | 100	50 | 100	50 | 100
Hidden Layer	1	1	1
Units	64	64	64
Activation	ReLU	Tanh	Tanh
Dropout %	0.10	0.10	0.10
Recurrent Drop. %	-	0.20	0.20
Dense Layer	1	1	1
Units	64	64	64
Activation	ReLU	ReLU	ReLU
Optimizer	Adam	Adam	Adam

Figure 2. DeepCoNN Architecture Comparisons

processing, CNNs make use of temporal convolution operators known as filters. In practice, filters are applied at multiple resolutions and coupled with non-linear activation functions and pooling techniques such as max pooling. The convolution operation is a linear mapping over n-gram vectors which helps learn word representations. See Figure 1 for a depiction of the architecture used in the original paper.

In our initial model experiments we decided to innovative upon the DeepCoNN model by adding regularization via dropout, and expand the number of nuerons in the hidden (CNN) and dense layers. Additionally, for consistency when experimenting between models we utilized Adam as our optimizer. Our implementation of this is detailed in Figure 2 above.

2.3.2 LSTM Architecture

Long Short-Term Memory (LSTM) cells retain and control information flow outside the normal flow of a simply recurrent network (RNN) via the use of gated cells. Symbolically, information can be stored, written, or read from a cell, similar to how data is manipulated in a computers memory. LSTM cells"decide" what is stored, written, or read by opening and closing gates. In practice these actions are implemented with element-wise multiplication by sigmoids which range from zero to one. A graphical overview of an LSTM cell is depicted below in Figure 3 [14]. A summary of our implementation can be found in Figure 2 above.

Thus gates can block or pass on information based on its strength, which they control through their own sets of weights. LSTMs are well-known variants of traditional RNNs and were introduced by Hochreiter and Schmidhuber in 1997. Furthermore, they are popular with the natural language processing given their above abilities, addition to, their ability to counteract the vanishing-gradient problem; and because standard stochastic gradient descent-based learning techniques can be used given their differentiability. The downside of LSTMs is the magnitude of parameterizations increases leading to longer training times, which we discuss in the results section. As discussed above, in all of our experiments we implemented regularization via recurrent dropout and/or regular dropout. We also expanded the number of hidden units and changed the optimizer.

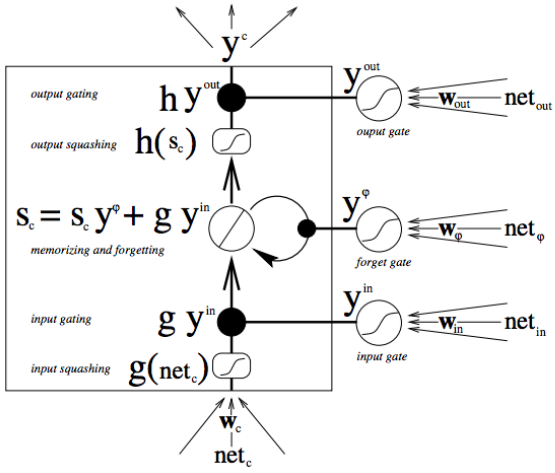


Figure 3. LSTM Gated Cell

2.3.3 GRU Architecture

We also explored Gated Recurrent Units (GRU) which were introduced in 2014 by Cho et al. [15]. GRUs simplify the LSTM architecture by combing the forget and input gates into an update gate and merging the cell state with the hidden state. Thus GRU have fewer gates than LSTMs and eliminate the separate memory component. As you would expect, leads to fewer parameterizations. Figure 4 shows the mathematics of GRU gates [16]. A summary of our implementation can be found in Figure 2 above.

$$\begin{aligned}\mathbf{s_j} = R_{GRU}(\mathbf{s_{j-1}}, \mathbf{x_j}) =& (\mathbf{1} - \mathbf{z}) \odot \mathbf{s_{j-1}} + \mathbf{z} \odot \mathbf{h} \\ \mathbf{z} =& \sigma(\mathbf{x_j W^{xz}} + \mathbf{h_{j-1} W^{hz}}) \\ \mathbf{r} =& \sigma(\mathbf{x_j W^{xr}} + \mathbf{h_{j-1} W^{hr}}) \\ \mathbf{h} =& \tanh(\mathbf{x_j W^{xh}} + (\mathbf{h_{j-1}} \odot \mathbf{r}) \mathbf{W^{hg}})\end{aligned}$$

$$\mathbf{y_j} = O_{LSTM}(\mathbf{s_j}) = \mathbf{s_j}$$

$$\mathbf{s_j} \in \mathbb{R}^{d_h},\ \mathbf{x_i} \in \mathbb{R}^{d_x},\ \mathbf{z}, \mathbf{r}, \mathbf{h} \in \mathbb{R}^{d_h},\ \mathbf{W^{x\circ}} \in \mathbb{R}^{d_x \times d_h},\ \mathbf{W^{h\circ}} \in \mathbb{R}^{d_h \times d_h},$$

One gate ($\mathbf{r}$) is used to control access to the previous state $\mathbf{s_{j-1}}$ and compute a proposed update $\mathbf{h}$. The updated state $\mathbf{s_j}$ (which also serves as the output $\mathbf{y_j}$) is then determined based on an interpolation of the previous state $\mathbf{s_{j-1}}$ and the proposal $\mathbf{h}$, where the proportions of the interpolation are controlled using the gate $\mathbf{z}$.

Figure 4. GRU Mathematical Overview

2.4. Implementation Details

To implement the DeepCoNN-DP model we used *Python*. Modeling construction was done via *Keras* version 2.1.2 with a *TensorFlow* version 1.4.0 back end. Data preprocessing was done mainly via *pandas*, *numpy*, *Keras preprocessing API*, and *sklearn*. Please see the project

GitHub repository for specific code implementation details. For training we used one Tesla K80 GPU on Google Cloud.

2.5. Results

Our preliminary work consisted of implementing DeepCoNN to model text review data for movie rating prediction. We then tried to innovate upon the DeepCoNN framework by experimenting with two new architectures in the $LSTM$ DeepCoNN-DP model and GRU DeepCoNN-DP models as outlined above. Additionally, experimentation was done with the original CNN underlying architecture through regularization, hidden layer dimensionality, and optimizers. Equally important, we experimented with different GloVe word embeddings. Below we present and summarize those results.

We used our recreation of DeepCoNN-DP as the baseline model when comparing experiments. Mean Squared Error was used when comparing models. Exhibit 1 shows our training/testing split as part of the data preprocessing step.

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}(r_n - \hat{r}_n)^2$$

In the original paper the Amazon Music Instant Review data set was used. We have included a table in Figure 5 showing the original results of the authors from that experiment for convenience. Note that they achieved an MSE of 1.253 on the Amazon Music data set for the *DeepCoNN-DP* model and a MSE of 1.233 on the *DeepCoNN-FM* model.

Table 4: **Comparing variants of the proposed model. Best results are indicated in bold.**

Model	Yelp	Amazon Music Instruments	Beer
DeepCoNN-User	1.577	1.373	0.292
DeepCoNN-Item	1.578	1.372	0.296
DeepCoNN-TFIDF	1.713	1.469	0.589
DeepCoNN-Random	1.799	1.517	0.627
DeepCoNN-DP	1.491	1.253	0.278
DeepCoNN	**1.441**	**1.233**	**0.273**

Figure 5. JDM MSE Results - Amazon Music Instruments

From our experiments using the Amazon Movie Review data set we show that all experiments lead to similar MSE results as per the paper's findings. Interestingly, our results from all architectures using a simple dot product interaction (DeepCoNN-DP) instead of the papers best model of (DeepCoNN-FM) show similar or better MSE results than the original paper. While we should emphasize the difference in data sets, our findings suggest that further analysis and testing may be needed to prove the advantage of *DeepCoNN-DP* over *DeepCoNN-FM*. Case in point in the authors' original results presented in Figure 5 on the $Beer$ data set, the different in MSE for DC-DP and DC-FM is only 0.005.

	Embedding	Training Time (seconds)	MSE Loss
DC-DP	50d	262	1.0954
LSTM	50d	2095	.9835
GRU	50d	5273	1.0532
DC-DP	100d	262	1.0954
LSTM	100d	2095	.9835
GRU	100d	5273	1.0532

Figure 6. Results - Model Comparison

2.5.1 Architecture Experiments

Our first innovation beyond the described DeepCoNN model involved testing two new network architectures, an LSTM network and a GRU network. Both the LSTM and GRU networks showed significant improvements in MSE vs our baseline DC-DP implementation. The best model results were from GRU with an MSE of 0.93 on the 100d GloVe Embeddings. As expected LSTMs were close seconds in performance vs GRUs followed last by the CNN architectures. A summary of results per architecture and embedding is presented in Figure 6.

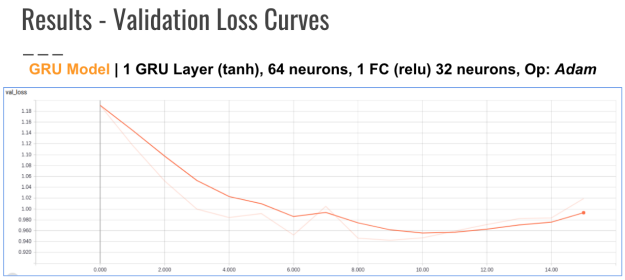
The downside of the RNN architectures particularly the LSTM was that training times were significantly slower than both GRU and cNN. In Figure 6 we can see that training the LSTM took **twice** as long as the GRU and almost **ten** times as long as the CNN on one Tesla K80 GPU. Thus significant compute resources and time must be allocated to train these RNN networks relative to the CNN baseline. Consequently, this may limit their real-world application for companies with prohibitive resource or latency constraints. Lastly, we performed naive experimentation with different hyperparameters such as optimizer learning rates and dropout %, but found that tuning only led to marginal differences and a more robust hyperparameter search was needed if given more time.

2.5.2 Embedding Experiments

As discussed above using GloVe embeddings has several advantages and disadvantages however we felt the advantages were greater. Intuitively, using pre-trained embeddings could help with the cold start problem that occurs in textual models when initially trying to learn a vocab from a new corpus. After implementing our new architectures we further innovated by testing the performance of different GloVe embeddings, namely 50d vs 100d. GloVe 100d increases the token size from roughly 400,000 unique tokens to roughly 1.5 million tokens. Our goal was to see how this increase effected the generalization of our model.

Results show that the increased token size led to improved performance as noted in Figure 6. Particularly, on our best model (GRU DeepCoNN-DP) we saw a 0.07 decrease in MSE from 50d to 100d. Comparing across models we saw decreases in each architecture when increasing GloVe from 50d to 100d.

2.5.3 Regularization



We experimented with dropout regularization in all architectures, plus recurrent dropout in the RNN architectures. While the original paper did not use regularization in the baseline CNN architecture, LSTMs/GRUs has a tendacy to easily over-fit and regularization is typically prudent. Below we show loss curves for all of our architectures using the 100d GloVe embeddings. For the LSTM and GRU architectures we used a recurrent dropout (removing layers) and standard dropout of masking the activation of some nodes. From the graphs we can see that even with regularization both the LSTM network and GRU network begin to over-fit half way through training. From our experiments we found that modest regularization helped smooth the loss curve and decrease over-fitting. For the CNN architecture, we found over-fitting to be less apparent which may be why the authors in the original paper omitted specifics on regularization. Regularization is summarized in Figure 2.

3. Conclusion

The DeepCoNN model represents a step forward in recommendation systems and the typical collaborative and content-based filtering systems that were traditionally used. This was confirmed by our research into predicting movie rankings. Our research indicates that the DeepCoNN model can be incrementally improved by utilizing different word embedding structures and regularization. Moreover significant improvement to accuracy are available for a move into different architectures. The tradeoff with these new architectures is that they are significantly more complex and more time consuming to train. Practitioners may well continue to focus on collaborative filtering and the standard DeepCoNN model, but with improved compute power, these new architectures should gain acceptance and warrant further research.

Model	MSE	Training Time	Epochs
DeepCoNN-DP	1.2645	10 min	20

Exhibit 2. DeepCoNN-DP Initial Results

The DeepCoNN model shows superior performance over standard collaborative filtering techniques and the Wide and Deep model proposed by Google researchers. Our research into other architectures and regularization methods show that further improvements to DeepCoNN are available.

4. References

[1] Zheng, Lei, Noroozi, Vahid and Yu, Philip. Joint Deep Modeling of Users and Items Using Reviews for Recommendations. arXiv working paper, June 2017

[2] Rendle, Steffan. Factorization Machines. ICDM 2010

[3] Catherine, Rose and Cohen, William. TransNets: Learning to Transform for Recommendation. arXiv working paper, June 2017

[4] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He DeepFM: A Factorization-Machine based Neural Network for CTR Prediction arXiv working paper, March 2017

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & Deep Learning for Recommender Systems. arXiv working paper, June 2016

[6] A. Baylen. Github Repo $Uifud$ https://github.com/Praznat/uifud As of 11/27/17

[7] J.Pennington, R. Socher, C.Manning. Global Vectors for Word Representation version: 6B.50d.txt. https://nlp.stanford.edu/projects/glove/ Available as of 11/27/17

[8] Image-based recommendations on styles and substitutes J. McAuley, C. Targett, J. Shi, A. van den Hengel SIGIR, 2015

[9] Balakrishnan Anusha, Dixit Kalpit. DeepPlaylist: Using Recurrent Neural Networks to Predict Song Similarity. semanticscholars.org

[10] Global Vectors for Word Representation https://nlp.stanford.edu/projects/glove/

[11] Collaborative Recurrent Neural Networks for Dynamic Recommender Systems Young-Jun Ko, Lucas Maystre, Matthias Grossglauser, 2016

[12] A Recurrent Neural Network Based Recommendation System David Zhan Liu, Gurbir Singh

[13] A Comparative Study of Word Embeddings for Reading Comprehension Bhuwan Dhingra, Hanxiao Liu, Ruslan Salakhutdinov, William W. Cohen

[14] A Beginner's Guide to Recurrent Neural Networks and LSTMS https://deeplearning4j.org/lstm.html

[15] Learning phrase representations using RNN encoder-decoder for statistical machine translation. Kyunghyun Cho, Bart van Merrienboer, C aglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio., 2014

[16] A Primer on Neural Network Models for Natural Language Processing Yoav Goldberg, draft 2015