UNIVERSITY OF CALIFORNIA

Los Angeles

Survey of Deep Learning Techniques for Recommendation Systems

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Henry Yung

2022

ABSTRACT OF THE THESIS

Survey of Deep Learning Techniques for Recommendation Systems

by

Henry Yung
Master of Applied Statistics
University of California, Los Angeles, 2022
Professor Yingnian Wu, Chair

Various practitioners in building recommendation systems currently leverage deep learning techniques. This thesis surveys several deep learning techniques to build recommender systems that serve personal recommendations. The paper begins with the traditional collaborative filtering model and builds several neural networks. The neural networks had similar structures but the main difference was the various feature sizes used to measure improvement on predictions. The neural networks were also fine-tuned by using dropout to prevent overfitting. Ensemble techniques are later applied in order to enhance recommendations. Finally, a bidirectional long short term memory (LSTM) was built to serve recommendations based on implicit feedback. The models are tested with an anime dataset from the web. The models were able to predict reasonable recommendations and both implicit and explicit recommendations can be made for users.

The thesis of Henry Yung is approved.

Dave Anthony Zes

Maryam Mahtash Esfandiari

Hongquan Xu

Yingnian Wu, Committee Chair

University of California, Los Angeles

2022

To my wife and family…

thank you for your support and encouragement

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

In the age of big data, there has never been a time when information is more available to users and companies. Every year companies invest billions into leveraging user information to gain a competitive edge over rivals. One area of interest in using big data to drive profits from user information is recommender systems. Recommender systems, generally, are algorithms aimed at suggesting relevant items to users. A recommender system makes a list of all possible items and ranks them according to the preference of a user. This personalized and ranked list is often referred to as recommendations. Recommendations are usually based on user history, known as interactions. A recommender system takes the user's interaction and makes the best recommendation possible. This paper aims to survey several recommendation techniques and assess the quality of the recommendations. The paper first describes the dataset and the features used for recommendation models. From there, the difference between implicit and explicit signals are analyzed in order to use the correct methodologies. The fourth chapter describes the different recommender systems built from the dataset. In addition, the components used to build the model, such as losses and offline evaluation metrics, are explained. Chapter five describes the model's results for the implicit and explicit recommenders. Finally, the last few chapters focus on the recommendations made from the model and possible enhancements for the future.

CHAPTER 2

Dataset

The dataset created by Matěj Račinský [10] is available on Kaggle, a website that hosts datasets and daily competitions for data scientists worldwide. The dataset contains information from Animelist, a popular website consisting of thousands of anime and manga for their userbase, divided into eight comma-separated value files. However, there were only three files used to build the anime recommenders. The first file, Useranimelist, contains the list of all animes registered by the user with their respective score, watching status, and the number of episodes watched. The file contains around 80 million records with 14,478 unique animes and 283,044 distinct users. The second comma-separated value file is watching status, which contains every possible level of the user's watch status. Watching status in the Animelist includes an integer value denoting the user's watch status. The watching status comma space value file is a mapping dictionary to convert those integers values to an actual status. For recommendation, only the watching status of 1 and 2 (currently watching and completed) are used, resulting in 13,597 unique animes. The third file is the clean user file. This file is where the author fixed some data issues. Several users had more watched episodes than what existed in the actual series. The author also fixed watch times and seen episodes. Some users had unrealistic prolonged inactivity such as 1900 showing up as their last active year. These were fixed by the author who updated those dates with the user's last watched date. The final file is the animelist. The animelist contains the list of animes and their relevant information, such as title, genre, and producer. These four files were used to build the recommendation system by filtering the Useranimelist by users in the clean anime file and filtering by current watching status. The Useranimelist is the main file used, and further details are listed below.

User Anime List Details:

- Username: The username associated with the user in the website
- Anime id: Anime id for the website animelist. (Example: 1).
- My watched episodes: number of episodes in the series watched by the user
- My start date: The date the user first watched the anime series
- My finished date: The date the user finished watching the anime series
- My score: A score given by a user from a scale of 1 to 10. 0 if user did not assign score
- My status: The current watching status for the anime from the user
- My rewatching: Column indicating if the anime series was rewatched
- My rewatching episode: Column indicating which epsiode in the series was rewatched
- My last updated: Timestamp showing last update of the user for the series.
- My tags: Tags that users added to the series

CHAPTER 3

Explicit vs Implicit Signals

3.1 Explicit Signals

Users can give explicit signals that can be leveraged for recommendation systems. These are signals that the users directly express or give. An example of an explicit sign would be the MovieLens website, where users rate movies on a scale of 1 to 5. A higher rating indicates a higher preference for the film. Another example of an explicit signal would be Netflix's like or dislike button for a movie or series. Although explicit signals are helpful in recommendations, specific characteristics must be understood to use them. The following list includes some of those items[5].

- Users are directly giving feedback for the specific item
- Signals can be either positive or negative
- There is absolute measurement for explicit signals
- Explicit signals are sparse

3.2 Implicit Signals

Users can also indirectly give signals that can be used for recommendation systems. These signals can include the number of watched episodes, minutes spent on specific content, or even a user's interaction history. There are critical differences between implicit and explicit signals. Implicit signals are numerous and readily available. Every time a user clicks or watches a video, it is recorded and logged. Another critical difference between implicit and explicit signals is that those implicit responses are considered positive. A user can click on content by accident, but the click would still be regarded as positive. However, an implicit signal can show a degree of preference. For example, users clicking on a video link by accident would register a low click count because they did not want to watch the content. However, if users continue to click on a link multiple times, it shows a higher and stronger preference for the item. By understanding the characteristics of implicit signals, recommender systems can be utilized to give meaningful suggestions. The items below list the features of implicit signals.

- Users are indirectly giving feedback for the specific item
- Signals are only considered positive
- Implicit signals measurements are relative
- There is usually a high abundance of implicit signals available for recommendation use

3.3 Explicit vs Implicit Signals - Anime Recommendation

The rating will be used as explicit feedback for the anime dataset to build several recommender systems. The users can rate any anime or movie on a scale from 1 to 10, with 10 indicating the highest preference. Note, in the dataset, that some scores were given a rating of zero. A rating of zero does not indicate the lowest score possible, but rather the dataset states that a zero rating means the user did not give a rating. Therefore, these zero ratings will not be used for explicit recommendation as they are not truly low scores. In addition, the animelist comma separated value file is the user interaction history with a watching status field monitoring the user's account. The field status shows the series or movie that a user has began or completed. The user's interaction history sequence will be used as implicit signals for recommendations.

CHAPTER 4

Model Methodology

4.1 Explicit Collaborative Filtering Recommendation System

4.1.1 Matrix Factorization

The first recommendation system built as a benchmark for the other explicit models is the matrix factorization model[7]. In collaborative filtering, matrix factorization model maps users and items to a joint latent factor space of dimensionality. Each user is then accompanied by their own item vector of g, and each item is accompanied by their own user vector h. These matrices are then multiplied together to estimate the rating between the user and item in the equation 4.1. Figure 4.1 below also illustrates the details further by showing how explicit signals are decomposed.

$$\widehat{r}_{ui} = g_i^T h_u \tag{4.1}$$

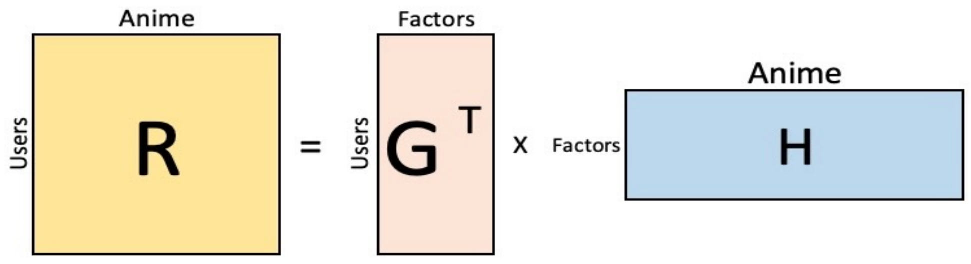

Figure 4.1: The above figure illustrates the matrix decomposition

The recommender then minimizes the squared error with regularization λ in equation 4.2 on the known ratings.

$$\underset{g*,h*}{\operatorname{argmin}} \sum_{(u,i \in \kappa)} (r_{ui} - g_i^T h_u)^2 + \lambda(||g_i||^2 + ||h_u||^2) \tag{4.2}$$

4.1.1.1 Matrix Factorization Optimization

There are two approaches to minimizing the regularized squared error in equation 4.2 above. The first approach is to apply stochastic gradient descent optimization. Stochastic gradient descent optimizes the equation by predicting the difference between the predicted rating and the actual. It then modifies g_i and h_i weights by a degree proportional to the learning rate η towards the opposite direction of the gradient. The process continues until the algorithm loops through the entire training data. For example, the equation updating g_i and h_i is shown below.

$$e_{ui} \approx r_{ui} - g_i^T h_u \tag{4.3}$$

$$g_i := g_i + \eta(e_{ui} * h_u - \lambda * g_i) \tag{4.4}$$

$$h_u := h_u + \eta(e_{ui} * g_i - \lambda * h_u) \tag{4.5}$$

The second approach to solving equation 4.2 is alternating least squares (ALS). The idea behind ALS is to rotate and fix one of the variables g_i or h_i. ALS then recalculates the other variable as a least-square problem when one of the variables is fixed. This process is repeated with fixes between the two variables until convergence. In order to use this algorithm and optimization, the Apache Spark library is used. Apache Spark is a general-purpose clustering computer system with the ALS algorithm built-in. The library API is convenient as users can feed their dataset, reference the specific columns to use, and then train and return recommendation results. It also allows the user to specify the regularization and the number of iterations to run for the algorithm. The anime dataset was fed into this library and served as a benchmark for the other models.

4.1.2 Neural Network

Neural networks can also build recommendation systems. Neural networks were built and designed around how the human brain works. A neuron is connected by synapses that interconnect to perform an activity. These neurons are all connected, forming an entire network. Similarly, a neural network in machine learning uses input to feed into a network that predicts an output. Figure 4.2 illustrates a simple neural network that generates an output. The inputs are all first connected to the nodes of the hidden layer. Afterwards, the hidden layer feeds into the output and generates a prediction.

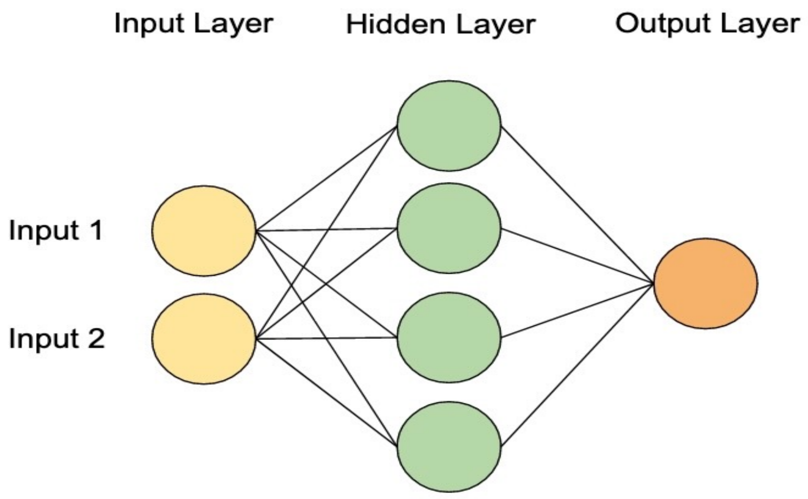

Figure 4.2: Simple Neural Network with one hidden layer

The neural network training process can be broken down into three significant steps. The first step in training is to perform forward propagation, where the inputs are fed into the network to generate an output. Then, after generating an output, an error is calculated and minimized based on a specific cost function. Finally, backpropagation is performed on the error to find its derivative for each weight and then updated to the model. This process is repeated for the specified epochs or until convergence.

4.1.2.1 Loss Function - Mean Absolute Error

Cost function for a neural network supports the model in understanding the actual and predicted value differences. For example, the cost function used to build the anime recommender systems is the mean absolute error (MAE)[1]. The mean absolute error shown in the equation takes the absolute difference between y, the actual value, and y', the predicted value. Therefore, MAE is advantageous as a loss function as it is not as sensitive to data outliers.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - y_i'| \tag{4.6}$$

4.1.2.2 Activation Function - ReLU Activation

ReLU, or rectified linear unit, is the primary activation function used for the anime neural networks. One significant advantage of ReLu is that its activation function is still non-linear. However, at the same time, its derivatives for positive values will always be positive, resulting in an advantage over other activation functions such as tanh because ReLU can deal with vanishing gradients. Vanishing gradients for activation function occur when the derivative with respect to its input diminishes as x becomes too large, resulting in the gradient terms approaching zero. The ReLU activation function avoids this problem, making ReLU one of the most popular activation functions in deep learning.

$$\phi(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x >= 0 \end{cases} \tag{4.7}$$

4.1.2.3 Optimizer- Adam

Adam is the main optimizer used in training the anime recommender systems. The optimizer combines two gradient descent methodologies, root mean squared propagation (RMSP) and momentum, to improve performance. The equation shows the mathematical aspect is as follows:

$$
\begin{aligned}
p_t &= \beta_1 p_{t-1} + (1-\beta_1) g_t \\
q_t &= \beta_2 q_{t-1} + (1-\beta_2) g_t^2 \\
\widehat{p_t} &= \frac{p_t}{1-\beta_1^t} \\
\widehat{q_t} &= \frac{q_t}{1-\beta_2^t} \\
w_{t+1} &= w_t - \frac{\widehat{p_t}\eta}{\sqrt{\widehat{q_t}+\epsilon}}
\end{aligned}
$$

where p_t and q_t are the estimates of the first moment and second raw moment, respectively. $\widehat{p_t}$ and $\widehat{q_t}$ are the bias-corrected weight parameters of the first and second moment. β_1 and β_2 are the decay rate of average gradient for the first and second moment. ϵ is a small constant. Combining these variables into our equation, the weights are updated.

4.1.2.4 Evaluation Method - Root Mean Square Error

The evaluation method used to evaluate the explicit rating of the model will be the root mean square error. The root mean square error takes the squared difference between the actual vs. predicted values and then divides that difference by the total value. Finally, RMSE takes the square root of the previous calculation for a score. One advantage of using RMSE over MAE is that it does not use absolute value for its calculation.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} |y_i - y_i'|^2} \tag{4.8}$$

4.1.2.5 Overfitting

One of the significant challenges of training neural networks is overfitting. Overfitting occurs when the model tries to learn from the noise of the training data. This action causes the model to use these nuisances in making predictions. However, when new data is presented, the model cannot generalize and makes incorrect predictions. Neural networks can easily overfit as they are learning millions of parameters during the forward and backpropagation stages to minimize the loss function or error. Although the initial results can look promising for the training data, the validation results show a different story. In order to prevent overfitting, some techniques are applied in model training.

4.1.2.6 Remediation for Overfitting

There are several methods use to reduce overfitting. The following methods are listed and used during model training.

- Dropout - One of the simplest methods employed to prevent overfitting is using dropout. Dropout is a regularization learning method that randomly thins out nodes during training. Figure 4.3 illustrates the dropout process for a two-layer network. The cross nodes on the right show the nodes that are dropped out. Nodes are kept with a parameter of p, and the other nodes are left out with a probability of 1 - p. This method results in a thinned neural network. Dropout can be applied at the layer level with the specified probability. For example, it is common to employ dropout at the probability of p at 0.5. However, tuning and experimentation are usually needed to find the best fit.

- Early stopping with patience - Another method that is commonly employed for regularization is early stopping. Early stopping is a machine learning method that stops the entire training process for a model when a negative metric threshold has been reached. It will then restore the weights to the previous iteration that is the most beneficial. Frequently, the patience parameter is employed with early stopping. The patience parameter waits several iterations after a negative threshold is reached before wholly stopping the training process. The parameter gives the model a chance to continue training in case there is noise in the training data. For example, it is usually common to apply early stopping to validation loss to prevent overfitting. Early stopping will usually stop training the model once validation loss increases. However, the patience parameter is also set to wait for a couple more training iterations to see if the validation loss will decrease. If validation loss does decrease, the model will continue training, but it will stop training if the validation loss continues to increase or stay above the lowest validation loss.

- Reducing Model Complexity - The final method to avoid overfitting is to reduce model complexity. As programming languages have simplified the methods to build sophisticated neural networks, it can be tempting to add more layers or features to machine learning models to increase performance. Syntax to add another layer or feature is trivial as it is sometimes as simple as adding one line of code. However, adding more layers to a neural network does not always guarantee model improvement. Many times, due to complexity, the model will overfit. By reducing a model's complexity, the results will often be better.

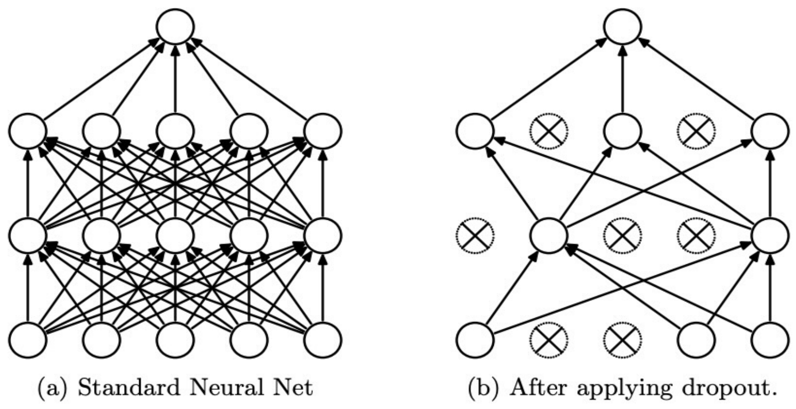

Figure 4.3: Dropout with two layers. [11] The right side shows the network after dropout.

4.1.3 Ensemble Learning

Ensemble methods can also be utilized with neural networks. Ensemble learning is a technique that combines several different machine learning algorithms in order to improve prediction accuracy. The ensemble method used to create the final machine learning for the recommender system is stacking. Stacking shown in figure 4.4 involves training several machine learning models known as beta learners or level-0 learners to make predictions. Predictions are then stacked together and fed into a final model, often known as a meta learner or level 1 model. For continuous variables, the meta learner can be any regression model. The stacking ensemble method has shown success in many real-life applications such as predicting short-term energy consumption [2] to time-series forecasting [9]. However, there are disadvantages to using stacking ensemble methods. One major disadvantage is that stacking can be expensive and time-consuming to implement. Stacking requires two or more models, which take time to train. Afterwards, the predictions are fed into the meta learner, which uses up even more time. At the same time, each beta learner needs some amount of reasonableness and feature tuning in order for stacking to work. Therefore, the models cannot be expected to perform well without proper training. Nevertheless, stacking ensembles can generate great results when adequately tuned and trained.

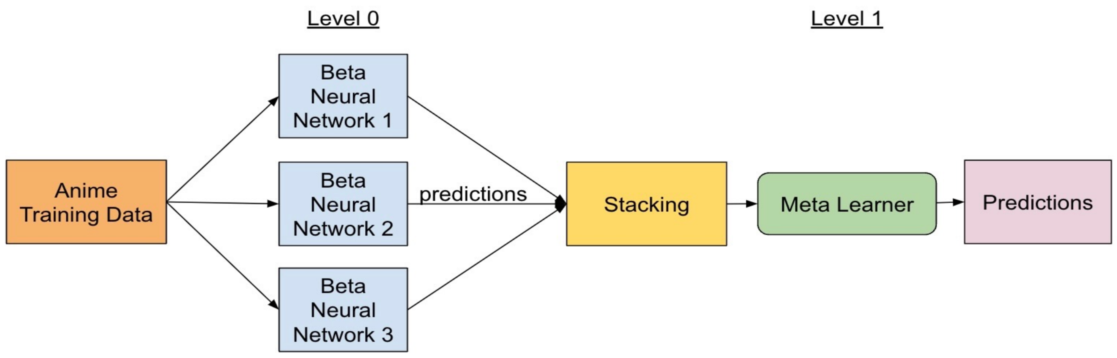

Figure 4.4: Beta and Meta Learner in Ensemble Learning

4.2 Implicit Recommendation Systems

The anime dataset can also be converted for implicit recommendations. Although the anime dataset was designed for explicit ratings, there is a user interaction history given the volume of the data. The user interaction history can be converted into a series of sequences ordered by timestamp as an input for a recurrent neural network. Several papers have used RNN modeling to perform recommendations, and they have heavily inspired this idea.

4.2.1 Recurrent Neural Network

A recurrent neural network is similar to a standard neural network where inputs are fed into a hidden layer and then onto the output. However, the main difference is that a recurrent neural network operates over sequences of vectors. [6] It can operate the sequential data either from the input, output, or both. Furthermore, a recurrent neural network's hidden layer receives input from the previously hidden layer, unlike a standard artificial neural network. Finally, another difference between a neural and recurrent neural network is the training process. A recurrent neural network uses backpropagation through time (BPTT), where the overall loss is the sum of all the losses through time. Therefore, recurrent neural networks can learn to use historic sequences for better recommendation results.

4.2.1.1 LSTM

Although RNNs are great at learning sequences over traditional machine learning models, they do have disadvantages. One major issue for RNN is the problem of vanishing and exploding gradient. In order to overcome this in sequential learning, Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) are applied for training. Figure 4.5 shows the architecture of an LSTM. An LSTM contains three gates: the input, forget, and output. The input gate decides which new information will be stored in long-term memory. The forget gate decides which long-term memory should be kept or discarded. Finally, the output gate will take the current input and newly computed long-term memory to produce the hidden states, which will be passed onto the next cell.

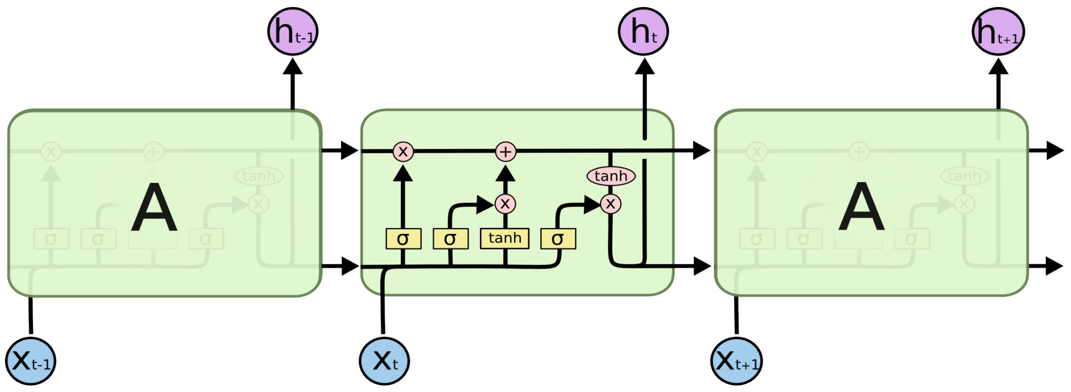

Figure 4.5: An exmaple of a LSTM Cell [8]

4.2.1.2 GRU

Like LSTM, GRU also deals with the vanishing and exploding gradient issue for RNN. Figure 4.6 has two gates, the update and reset gate. The gates, like LSTM, are designed to filter out irrelevant information. The reset gate is derived and calculated using the hidden state from the previous time step and current input data. It uses an activation function to filter out less significant values. The update gate is also derived and calculated using the hidden state from the previous and current timestamp. Each gate, though, is unique, which allows the gates to serve their specific purpose. After the update gate is created, the hidden state is updated.

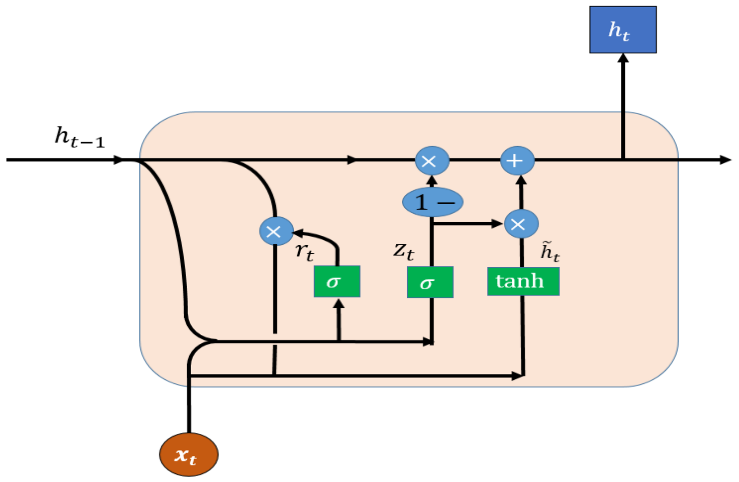

Figure 4.6: An example of a GRU Cell [4]

4.2.1.3 Categorical Cross-Entropy Loss

For implicit recommendation, the categorical cross-entropy loss is used for multi-class classification. The predicted values are first converted into one hot encoded values before cross-entropy can be leveraged. Then, the equation calculates the average difference between the probability of the actual and predicted classes. Finally, the loss is minimized, with a lower loss being preferable.

$$\text{CEL} = -\sum_{i \in \mathcal{C}} y_i \cdot \mathsf{log}[\,\widehat{y_i}\,] \tag{4.9}$$

4.2.1.4 NDCG@k

Net Discounted Cumulative gain (NDCG) is one evaluation metric for scoring a recommendation system. NDCG has become one of the more popular choices in evaluating recommendation systems. Equation 4.10 in [12] gives a score to correct recommendations that are higher on the ranking list as they are usually more relevant to user's taste.

$$\text{NDCG(u)@k} = \sum_{i=1}^{k} \frac{G_{ui}}{\mathsf{log}(1+i)} \tag{4.10}$$

where k represents the sample size of the recommendation. G_{ui} represents the gain of the item at the ith position. A higher score means better recommendations based on this evaluation metric.

4.2.1.5 MRR@k

The mean reciprocal rank (MRR) evaluates the relevance of the model's highest rank for a predicted value at k. If the predicted item was not in the kth ranking, the mean reciprocal rank would return zero. The reciprocal rank focuses on finding the best item in a recommendation system.

$$\text{MRR} = \frac{1}{Q} \sum_{k=1} \frac{1}{R_k} \tag{4.11}$$

4.2.1.6 Recall@k

In recall@k, the formula counts the relevant recommendations at the top k and divides them by the user's total number of relevant items.

$$\text{Recall@k} = \frac{\text{Recs and relevant items}}{\text{\# of relevant items}} \tag{4.12}$$

CHAPTER 5

Model Training and Results

5.1 Explicit Model Training

The following section describes the training process for building the recommender system on the anime dataset. The three neural networks were built with the same architecture where the user and anime id are first embedded into their layers. The embedded layers contain the unique counts of user and anime ids. Next, the embedded layers are flattened, concatenated, and fed into the dense layers for model output. Figure 5.1, 5.4, and 5.7 illustrates the design for the three networks. The main difference in the networks is the number of features used in training. These features used were 100 for the first model, 128 for the second, and 150 for the third model. A dropout of 0.5 was employed across all three neural networks. Early stopping and patience with a parameter of 3 are also utilized to prevent overfitting. Finally, The learning rate was kept constant at 0.001. Figures 5.2, 5.3, 5.5, 5.6, 5.8, and 5.9 show the training and validation curves for the three neural networks. The curves show learning as they decrease to the 15 epoch, with the lowest loss at around 1.5. These three models are then combined to create a stack ensemble model with the methodology mentioned in section 4.13. As these are explicit ratings with continuous output, two ensemble models were built using linear and ridge regression as the level 1 meta learner.

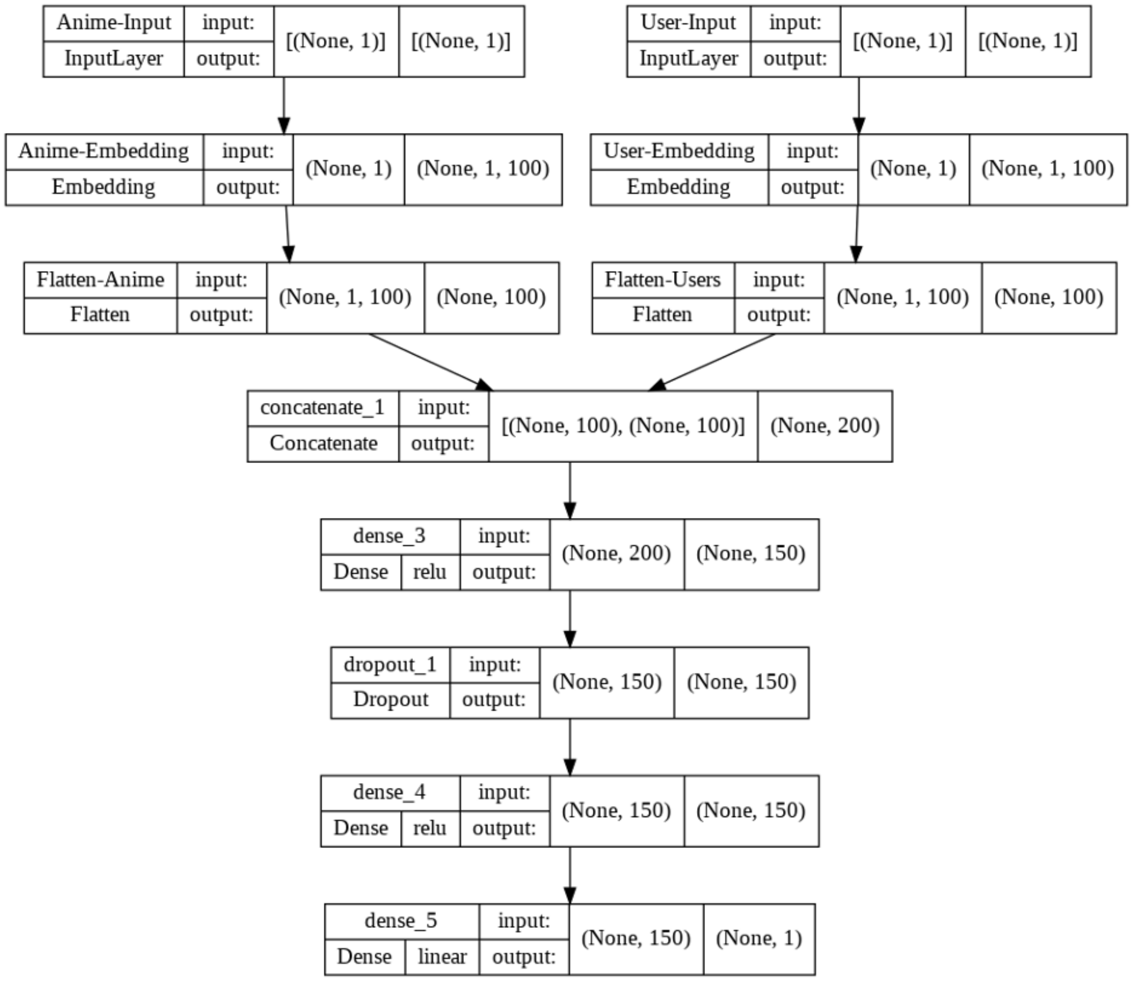

Figure 5.1: Beta Neural Network 1 Architecture

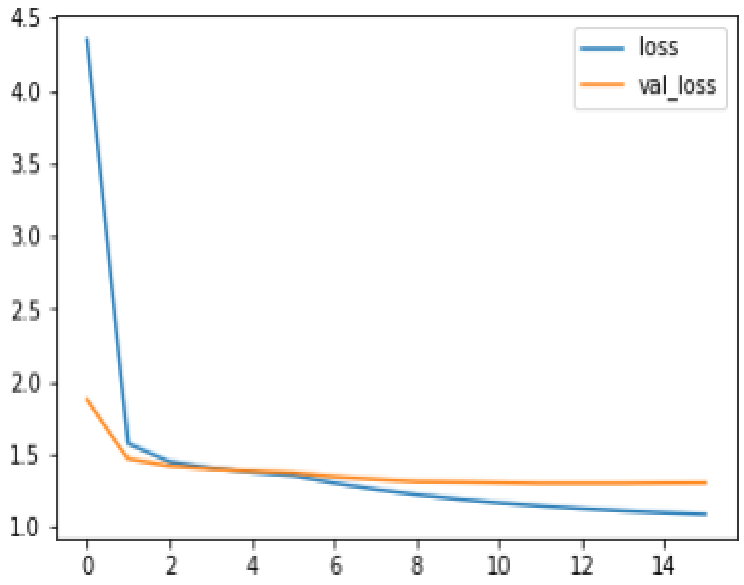

Figure 5.2: Beta Neural Network 1-MAE Training and Validation Loss

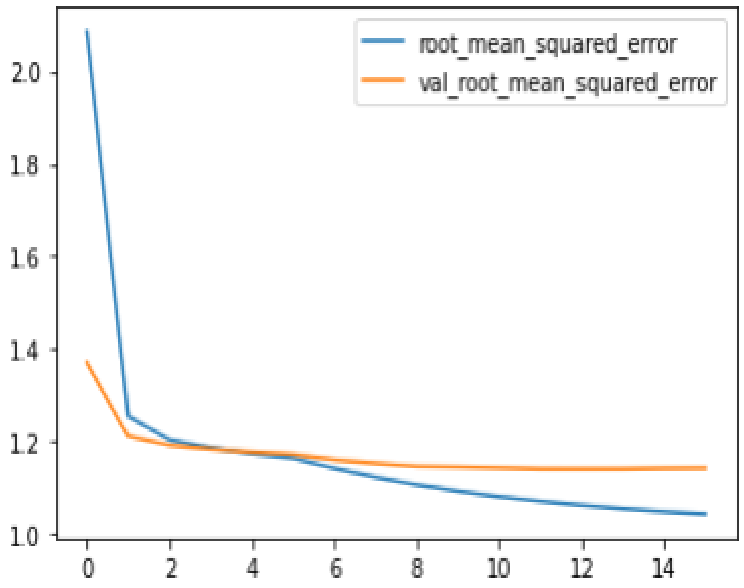

Figure 5.3: Beta Neural Network 1-RMSE Training and Validation Curve

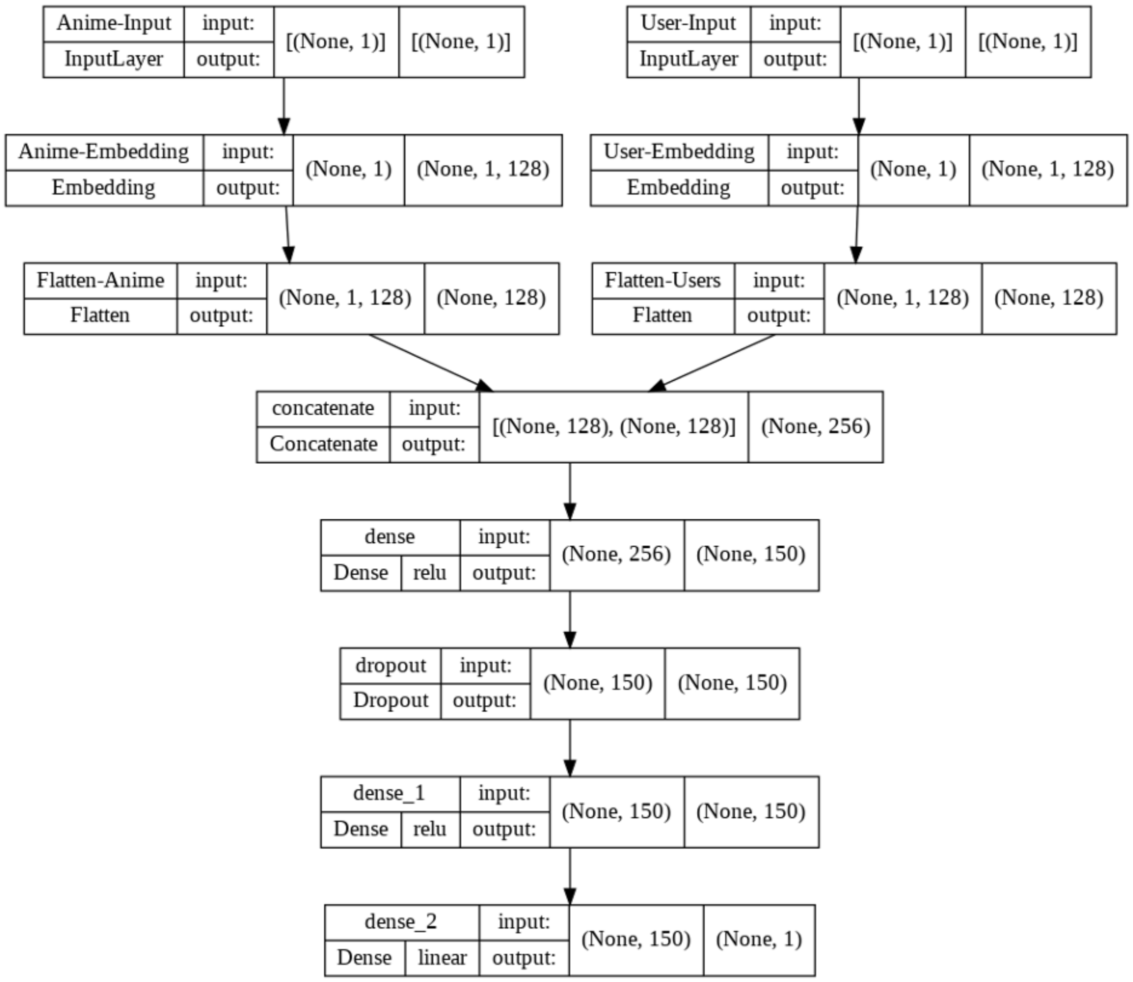

Figure 5.4: Beta Neural Network 2 Architecture

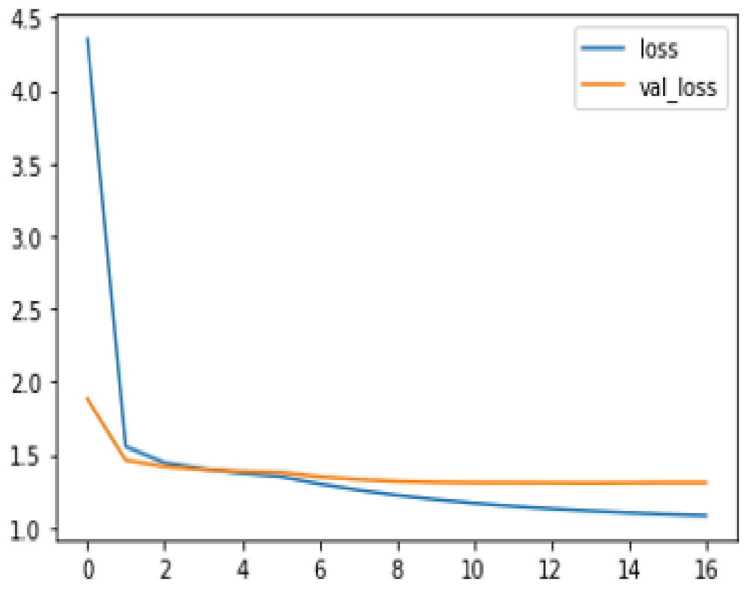

Figure 5.5: Beta Neural Network 2-MAE Training and Validation Loss

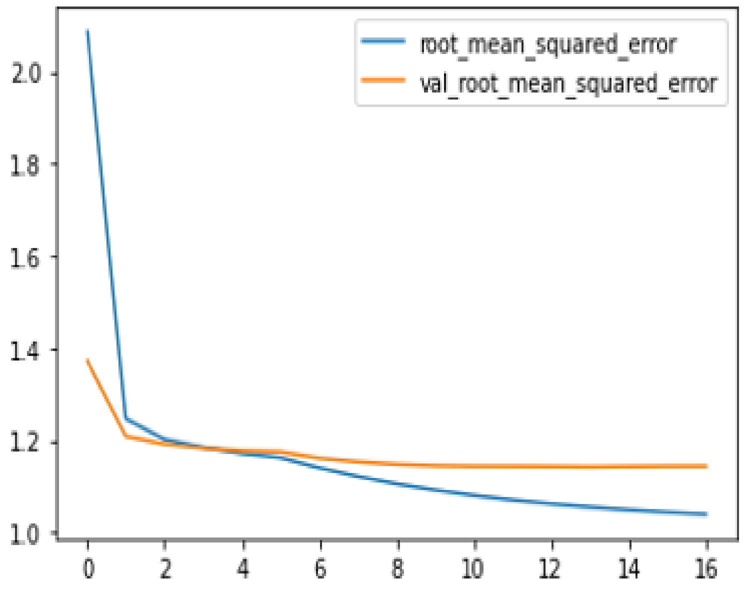

Figure 5.6: Beta Neural Network 2-RMSE Training and Validation Curve

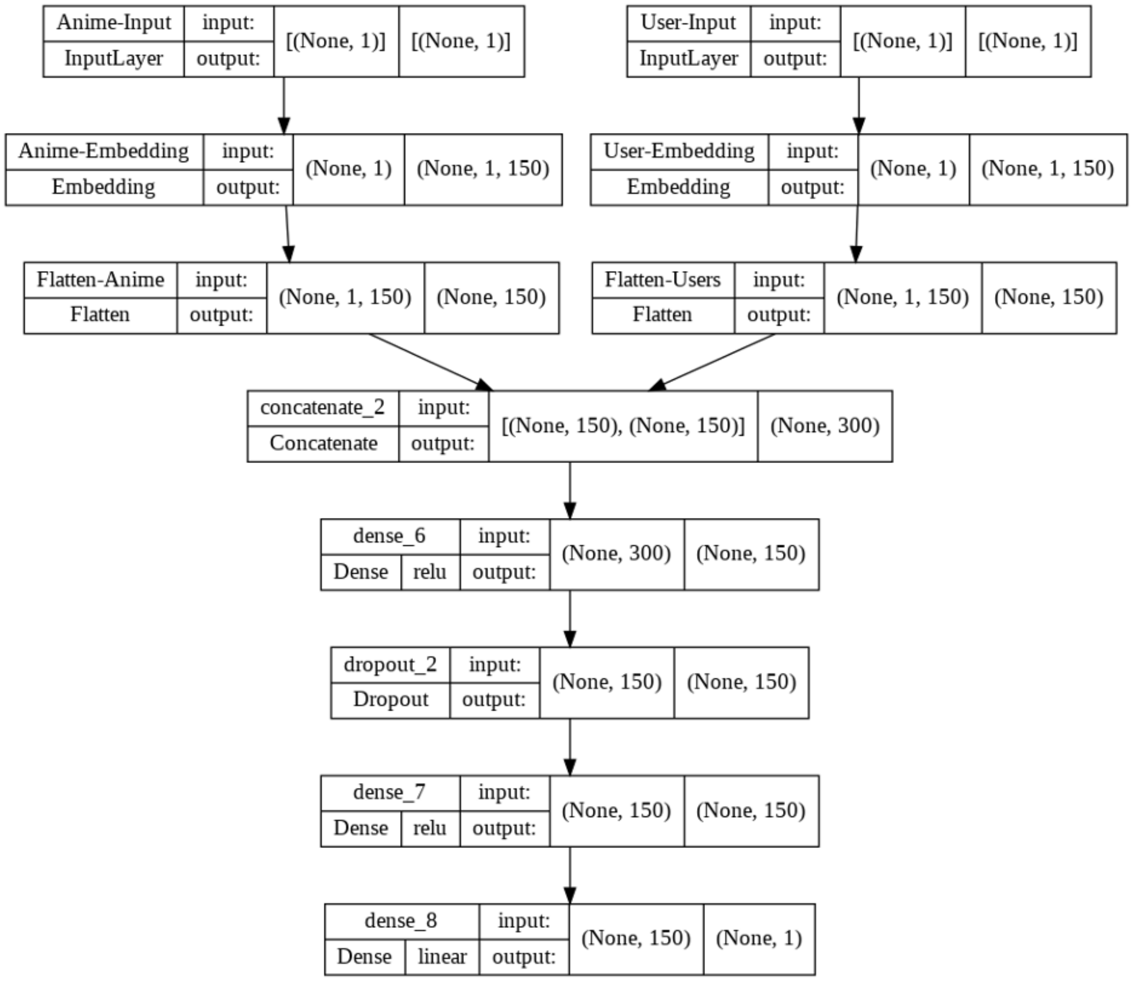

Figure 5.7: Beta Neural Network 3 Architecture

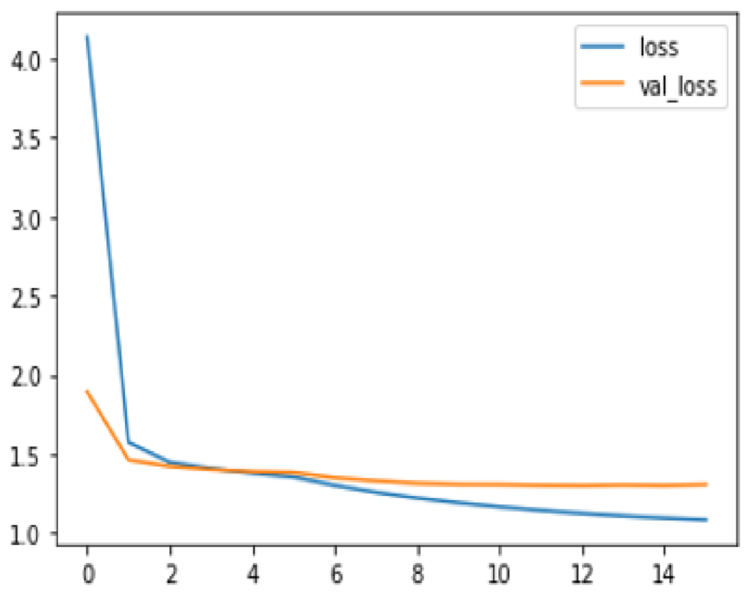

Figure 5.8: Beta Neural Network 3-MAE Training and Validation Loss

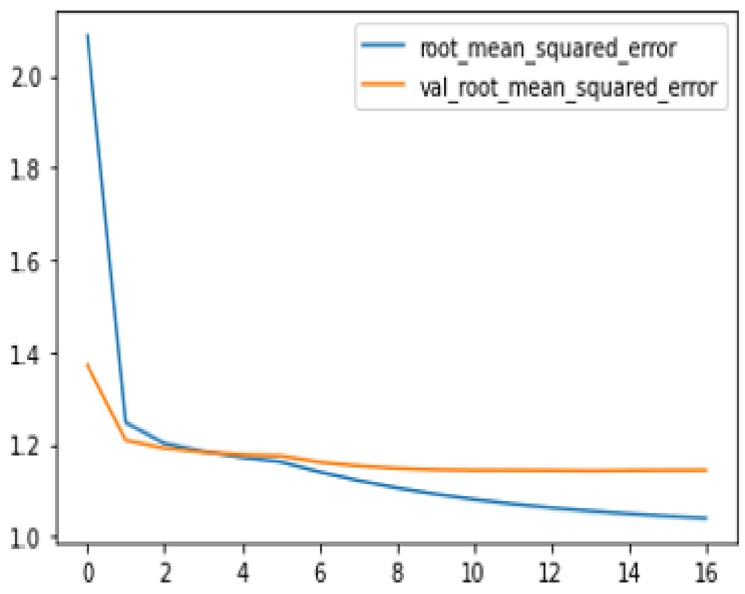

Figure 5.9: Beta Neural Network 2-RMSE Training and Validation Curve

5.2 Explicit Model Results

Num	Model	RMSE
1	Beta NN-100	1.152
2	Beta NN-128	1.157
3	Beta NN-150	1.149
4	Ensemble-Linear	1.130
5	Ensemble-Ridge	1.130
6	ALS-Spark (Baseline)	1.136

Table 5.1: Explicit Model Results

Table 5.1 shows RMSE results for 6 models trained using explicit ratings. These results seem reasonable as the rating scores ranged from 1 to 10. Each beta NN could not beat the baseline ALS-Spark Model as each RMSE was higher on the validation results. However, the ensemble model combining all three beta neural networks outperformed the ALS model. These results show how ensemble models can improve recommendation results. It is interesting to note that linear and ridge ensembles had similar results. Of course, there is a high cost to employing ensemble and deep learning techniques. Neural networks and ensembles take time to tune and train, while ALS is already packaged and ready for deployment. However, deep learning techniques and ensembles can generate good results with proper tuning.

5.3 Implicit Model Training

In order to train implicit data, the user's interaction history is split into sequences for the LSTM network. The sequences have varying lengths, and padding was applied to keep the same length. The LSTM will also need output, and therefore the last sequence was used as a target variable. The average user in myanimelist had around 200 interactions. Applying the mean would overload the LSTM model and likely add no predictive value. Therefore, this experiment was done by breaking the sequences between the fifth and tenth percentile and sequences of 15 and 18 were tested. Based on the analysis of the data, the sequence of 18 was ultimately chosen. From there, the training data was fed into the bidirectional LSTM network. Figure 5.10 illustrates the bidirectional LSTM network where the first layer used 200 features and the second layer used 100 features. Dropout and early stopping with patience were employed again to prevent overfitting. Finally, the learning rate with the Adam optimizer was kept constant at 0.001. Figure 5.11 shows the training loss curve for cross-categorical entropy. An interesting observation can be noted when looking at the training loss. The training data had over 13597 distinct animes. With categorical cross-entropy, if we just random-uniformly guess the outcome, our expected loss is 9.51. By showing the final loss at 4.6612, the model shows a considerable improvement over chance.

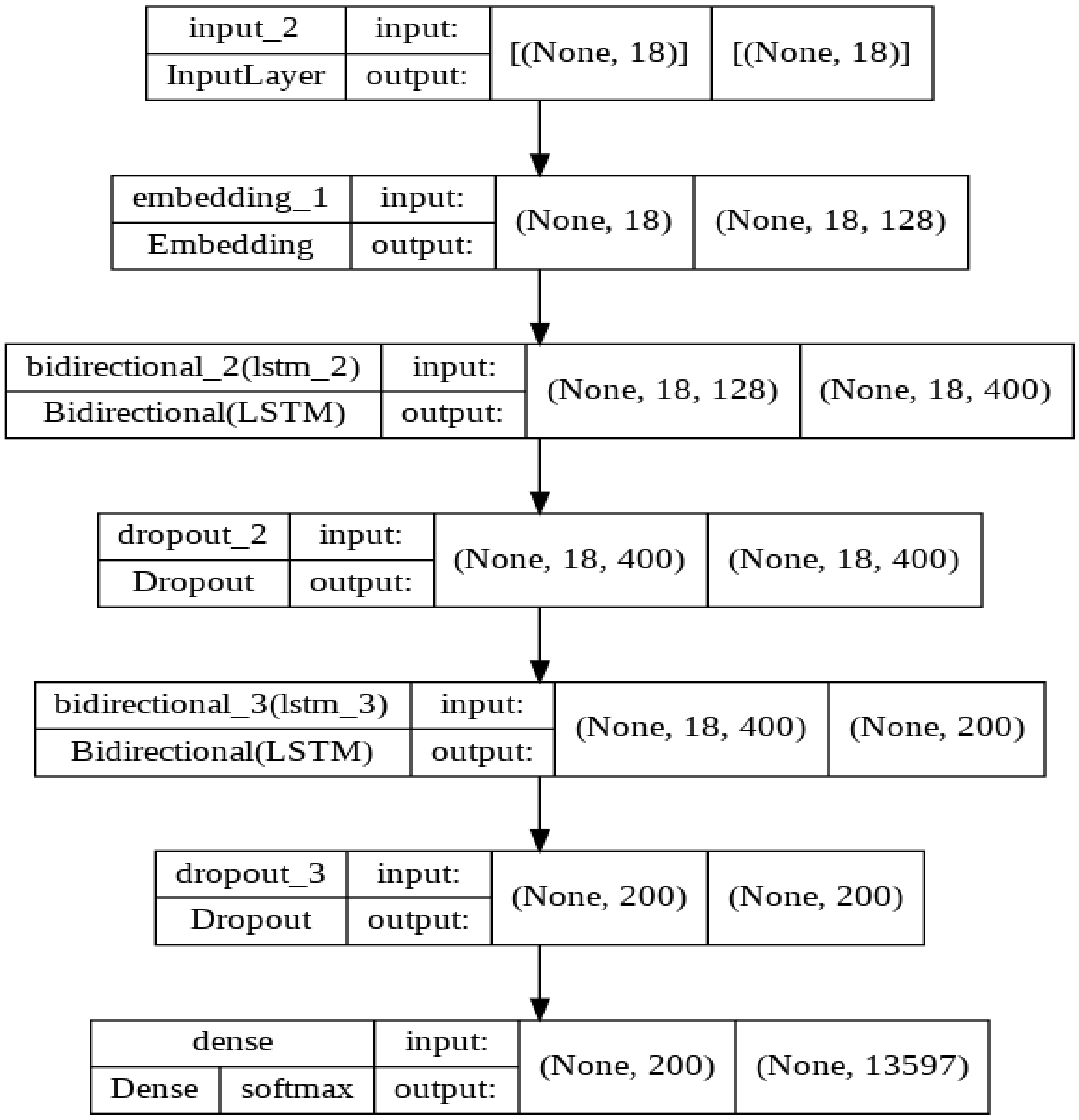

Figure 5.10: Bidirectional LSTM Architecture

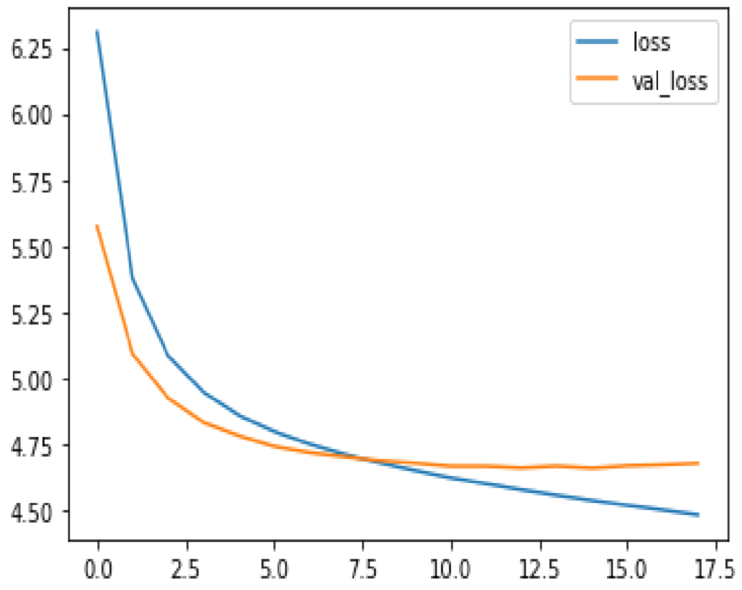

Figure 5.11: Bidirectional LSTM-Training and Validation Loss

5.4 Implicit Model Results

Num	Metric	Score
1	NCDG@5	0.2953
2	Recall@20	0.5065
3	MRR@20	0.2887

Table 5.2: Implicit Model Results

Table 5.2 shows the offline validation for the bidirectional LSTM trained on sequential data. These results seem reasonable as only the anime ids were used as sequential data. In a future iteration, contextual information such as genre can be used to improve results. The ranking metric of Recall@20 shows the system was able to predict the top 20 predictions more than half the time. In the gru4rec whitepaper, [3], the top recall@20 was around 0.66 showing some reasonableness in the offline validation.

CHAPTER 6

Discussion - Recommendation Result

After model training and evaluation, this section discusses the results from the built recommendation systems. One user was randomly selected for recommendation. Table 6.3 shows the last twenty animes that the user had watched. Additionally, an analysis was done on genres from the user's interaction history by taking the top three genres from each anime and aggregating them. Figure 6.1 shows this specific user has strong affinity for action but at the same time, the user frequently watches horror and supernatural genres as well. Table 6.1 and Table 6.2 show the top five recommendations from the Ensemble Neural Network and Bidirectional LSTM model for this specific user. Both system were able to recommend action genres which is the user first choice. However, it seems the bidirectional LSTM was consistently able to recommend horror and supernatural genres as well. The bidirectional LSTM was trained on only the sequences of the User's behavior and did not contain any contextual information about genres. Therefore, the bidirectional LSTM being able to recommend reasonable anime is interesting.

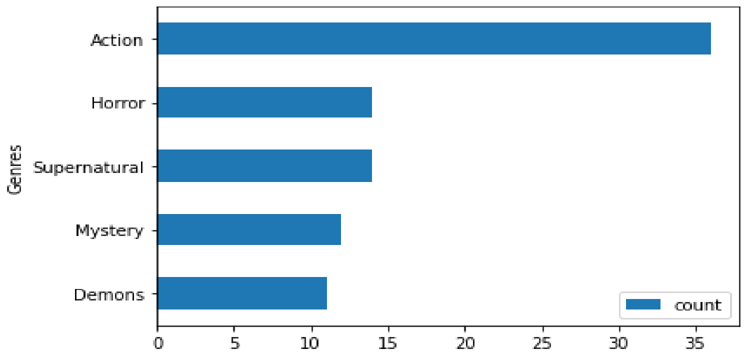

Figure 6.1: Genre count of random user

Rank	Title	Genre
1	Jikuu Bouken Nuumamonjaa	Action, Comedy, Fantasy
2	Chirin no Suzu	Adventure, Drama, Fantasy
3	Mobile Suit Zeta Gundam- Heir to the Stars	Military, Sci-Fi, Space
4	Lupin III:Seven Days Rhapsody	Action, Adventure, Comedy, Seinen
5	Joshikousei	Comedy, Slice of Life

Table 6.1: Ensemble Recommendation-Ridge Regression Recommendations

Rank	Title	Genre
1	Elfen Lied	Action, Horror, Psychological
2	Deadman Wonderland	Action, Sci-Fi, Horror
3	Claymore	Action, Adventure, Super Power
4	Another	Mystery, Horror, Supernatural
5	Vampire Knight: Guilty	Mystery, Supernatural, Drama

Table 6.2: Bidirectional LSTM Recommendations

Title	Genre	Score	Dateline
Biohazard:Degeneration	Action, Horror, Sci-Fi	10	4/16/17 5:43
Zetman	Action, Drama, Horror	9	4/16/17 0:48
Gungrave	Action, Drama, Sci-Fi	7	4/16/17 0:45
Ghost Hunt	Comedy, Horror, Mystery	9	4/16/17 0:41
Kikoushi Enma	Horror, Demons, Supernatural	10	2/22/16 18:50
Blassreiter	Action, Sci-Fi	9	2/18/16 18:09
Darker than Black: Gemini	Action, Sci-Fi, Mystery	9	2/11/16 16:39
Darker than Black: Gaiden	Action, Sci-Fi, Mystery	10	2/11/16 16:37
Darker than Black: Sakura	Sci-Fi, Comedy, Parody	10	2/11/16 16:36
Darker than Black: Keiyakusha	Action, Sci-Fi, Mystery	10	2/11/16 16:35
Basilisk: Kouga Ninpou Chou	Action, Adventure, Historical	9	12/23/15 17:18
Hakuouki Movie 2: Shikon Soukyuu	Action, Historical, Supernatural	10	12/2/15 15:45
Hakuouki Movie 1: Kyoto Ranbu	Action, Historical, Supernatural	10	12/2/15 15:45
Gankutsuou	Drama, Mystery, Sci-Fi	0	11/25/15 18:59
Devil May Cry	Action, Demons, Fantasy	9	11/25/15 18:45
Tokyo Ghoul	Action, Mystery, Horror	9	11/18/15 18:46
Petshop of Horrors	Horror, Josei, Mystery	10	11/18/15 17:14
Mononoke	Mystery, Historical, Horror	10	11/18/15 17:12
Ayakashi: Japanese Classic Horror	Mystery, Historical, Horror	10	11/18/15 17:11
Witch Hunter Robin	Action, Magic, Police	10	4/29/15 3:59

Table 6.3: User Interaction History

CHAPTER 7

Conclusion

7.1 Limitations

Both models could generate reasonable predictions but improvements can always be made. However, neural networks for large datasets are computationally and resource intensive so there was a limitation for further experimentation. Training can take many hours or days to complete which limits the number of iterations that can be created. In addition, while on-demand pricing for Amazon Web Services seem reasonable on an hourly rate, the costs can quickly add up while completing the necessary training.

7.2 Application and Future Work

Several machine learning models were built to create anime recommendations for users. The first model is an ALS model from apache spark that was served as a baseline model. Afterwards, several neural networks were built using user and anime id to predict ratings. These neural networks were then combined to create ensemble models to enhance performance. Finally, a bidirectional LSTM network was built to predict recommendations based on sequences for implicit recommendations. For future work, different ensemble methods can also be used and experimented with for the explicit models to see if improvements can be made. The model can be improved on the implicit recommendations by leveraging contextual information and concatenating them for richer insight. In the meantime, the models in this paper can serve as a guideline to build recommender systems either explicitly using measurable feedback or implicitly using the user interaction history.

REFERENCES

[1] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)? arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

[2] Federico Divina, Aude Gilson, Francisco Gómez-Vela, Miguel Garcia Torres, and José Torres. Stacking ensemble learning for short-term electricity consumption forecasting. *Energies*, 11:949, 04 2018.

[3] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. 1:7, 2015.

[4] Zhelin Huang, Fangfang Yang, Fan Xu, Xiangbao Song, and Kwok-Leung Tsui. Convolutional gated recurrent unit–recurrent neural network for state-of-charge estimation of lithium-ion batteries. *IEEE Access*, PP:1–1, 07 2019.

[5] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Comparison of implicit and explicit feedback from an online music recommendation service. 1:3, 01 2010.

[6] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. 1:1, 2015.

[7] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[8] Christopher Olah. Understanding lstm networks. 1:1, 2015.

[9] Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. 1:3, 12 2014.

[10] Matěj Račinský. Myanimelist dataset. *Kaggle*, 2018.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] Richong Zhang, Han Bao, Hailong Sun, Yanghao Wang, and Xudong Liu. Recommender systems based on ranking performance optimization. *Frontiers of Computer Science*, 10:273, 07 2015.