Implementation of Recommendation Framework based on Deep Reinforcement Learning(DRR) on News Data

Yue(Lynnc) Liang
University of Minnesota
Vishnu Chhabra
University of Minnesota

May 8, 2021

1 Introduction

With the rise of media consumption in today's world, recommender systems are growing more prevalent and important. Recommendation system plays an essential role in the business world. A good recommendation system translates to improved user experience and costumer satisfaction. In other words, the more efficient a company's recommender system is, the more likely it is that they will stand out among competitors and generate more revenue.

Collaborative filtering and content-based filtering are the two major paradigms to current, popular recommender systems. The traditional approach to the recommender system is to consider it as a classification or prediction problem as with collaborative filtering and content-based filtering. But these methods are faced with many challenges, including sparsity of data, latent association, and scalability. [3] Specifically, the sequential issue associated with traditional models can be addressed with reinforcement learning methods. Other methods like neural collaborative filtering and different deep reinforcement learning algorithms can also successfully address some of these limitations with additional improvements to current models.

In this project, the experimental focus will be on various reinforcement learning algorithms such as deep deterministic policy gradient (DDPG) using an Actor-Critic framework. We will formulate the recommender system as a reinforcement learning problem using the MIND dataset from Microsoft's News website.

2 Background and Literature

A recommender system generates prediction of what users will like, and it consists of three important components: candidate generations, scoring systems, and re-ranking systems [10]. There are many different methods to building a recommender system that considers all three of these components, and as businesses seek to improve their systems, more variations of methods and new methods are being introduced. In this section, we will be discussing traditional methods and a variation involving deep learning. This leads to a more in-depth view of different deep learning architectures.

And finally, we will be examining the more recent applications of Reinforcement-learning (RL) and deep reinforcement learning (DRL) algorithms in the area of recommender systems.

2.1 Traditional Methods

Traditional methods used in recommender systems treat the problem of filtering content as some combinations of prediction and classification. This involves clustering, nearest neighbor, and matrix factorization. The two common groups are collaborative filtering and content-based filtering.

Content-based filtering is a rather trivial approach to the recommender system. It essentially uses known user information as connective pipelines for recommendations. In other words, recommendations of items for a user does not depend on other user's data [10]. Having mentioned that, the focus in this section will be on collaborative filtering as it is more commonly used, more complex, and more efficient than content-based filtering. Furthermore, there are more notable works with collaborative filtering. There have also been variations of collaborative filtering methods introduced that leverage the power of deep learning to achieve more robust systems and to address certain shortcomings of the traditional methods.

2.2 Reinforcement and Deep Reinforcement Learning

Recommender systems taking into account user's interactions with items are inherently sequential. This temporal data broadens the scope beyond a prediction problem [13]. More specifically, it can be formulated as a Markov Decision Process (MDP) and solved using reinforcement learning methods [1]. With reinforcement learning, an agent interacts with the environment with the goal to maximize the reward. Applying this to the recommender system, the goal is to maximize user's satisfaction by recommending the best fitting items to the user. The agent is therefore the algorithm that is applied, the reward is user's satisfaction rating, and everything else is the environment. Combining methods of deep learning with reinforcement learning results in a reinforcement learning based recommender system, which can be further classified into reinforcement learning (RL) and deep reinforcement learning (DRL) [1]. Within these two categories, we will be discussing various algorithms specific to each.

2.2.1 Reinforcement Learning

The algorithm that will discussed in this category is the Monte Carlo Tree Search (MCTS) because unlike other tabular methods in RL, such as temporal difference, dynamic programming, etc, MCTS only needs to keep the information of a sampled experience, not the entire environment. This is preferred especially when the state and action spaces increase [1]. MCTS is a decision-time planning algorithm. It approximately solves sequential decision-making problems and is an improved variation of the Monte Carlo algorithm.

In a study conducted at UT Austin, MCTS was used to design better playlists by taking into account both song and transition preferences in real-time. For a music recommender system, there are two important components to consider: preference learning and planning for selecting the next song. The emphasis in this study is the planning approach using MCTS. The algorithm is able to execute many planning simulations and keep track of encountered states and actions within a tree [4]. The music recommender system implemented with MCTS was compared to a baseline, which sequentially generates playlists based on user preference using reinforcement learning methods with

naive planning. It was found that the advanced planning techniques of MCTS holds a significant advantage over the naive planning in the baseline comparison.

2.2.2 Deep Reinforcement Learning

DRL is an improvement to RL in the sense that it is able to handle higher dimensional spaces. Deep Q network (DQN) is the most widely used algorithm in the DRL category. However, it is not efficient as it sometimes overestimate action values, randomly selects certain experiences regardless of significance, and is computationally expensive since it requires an iterative optimization at each step. This means that learning is inefficient and the process is computationally expensive [1]. Deep deterministic policy gradient (DDPG), a combination of DQN and deterministic policy gradient (DPG), operates in the continous space, and can therefore solve the issues. DDPG algorithm is used on the Actor-Critic framework for parameter and model training. The names means the following: the Critic measures the performance of the action taken, which is value-based, and the Actor controls how the agent behaves, which is policy-based. Essentially, how this works is that the actor, who controls the agent, first makes decisions randomly and the Critic observes the actions taken and provides feedback. The learning occurs when feedback is provided, upon which, the Actor will update its policy (policy function) as a result. At the same time, the Critic will also be updating its way (value function) to provide better feedback the next time. This reveals that there should be two neural networks running in parallel to implement the Actor-Critic framework [12].

To put this into perspective, we will examine a specific framework from a study done at Harbin Institute of Technology [6]. The framework implemented is named DRR, which adopts an Actor-Critic method to model the interactions between the user and items and considers both immediate and long-term rewards. It consists of the Actor network, the Critic network, and the state representation module which is significant to both the actor and Critic network. First, the Actor networks takes into account the user's state and generates an action based off of that. The inputs to the network are n most recent, positively interacted items. These are embedded and fed into the state representation module, which generates The state representation s, followed by two layers of ReLu and one Tanh layer. s is also transformed into an action, a (a policy of state s) and is defined as a ranking function of continuous parameter vectors. The ranking is defined using the action and returns the top ranked item. On the other hand, the Critic network is essentially a deep-Q network used to approximate the Q-value function with inputs s and a from the Actor network. The results with DRR shows a notable improvement compared to the baseline.

2.3 Evaluation Metrics

Since some of the major methods to implement a recommender system has been discussed, we now move on to evaluation methods to analyze the performance of the system. Recall@k, precision@k, and average precision@k are three of the most popular among the many evaluation methods. The 'k' in question is the recommendation per example. Recall@k is the ratio of true labels captured to the number of overall true labels, and precision@k is the ratio of true labels captured to the predictions made. In other words, recall emphasizes the amount of recommendations provided, and precision emphasizes on relevant recommendations. Given that, one of these metrics must be optimized while the other must be held above the minimum acceptance threshold [9]. However, precision@k and recall@k do not consider the ranking order of the recommendations, but average precision@k does. Average precision@k is the average of all precisions@k where the recommendation is a true positive.

It is also important to note other metrics that can be used to measure performance other than accuracy as accuracy alone can actually hinder the performance in recommender system. Based on research by GroupLens from the University of Minnesota, a focus on accuracy in these systems could result in less favorable recommendations [8]. This is because there can technically be a high accuracy rating, but the recommendations might not be relevant to the user. There are other metrics that can be considered such as diversity, coverage, serendipity, and novelty. Diversity measures how similar or dissimilar recommended items are for users, coverage measures the overall percentage of parameters that the system was able to recommend, serendity measures the success of having received an unexpected recommendation, and novelty measures how unknown recommended items are to the user [2]. These metrics fits into the user-centric approach to evaluating a recommender system, in [8], which combined with conventional metrics would make them accurate, and most importantly, helpful.

3 Experimental Design

From the literature review above, it is emphasized that the use of deep reinforcement learning is making headway in the area of recommender system since it combines the advantages of both deep learning and reinforcement learning. Hence, the experimental design of this project will be focused on a deep reinforcement learning using the Actor-Critic framework in combination with the DDPG algorithm.

The recommender system being implemented in this project will use the MIND dataset, a large-scale news dataset containing 160k articles from Microsoft News. The data must first be parsed to prepare it to be fed to the model. This also includes performing data-embeddings and assembling the embeddings. Creating the embeddings is a critical step to prepare the data because it allows for a continuous state representation of the user with respect to their sequential action, in other words, it solves the continuous control problem [6]. In practice, this involved generating vector representation of the news articles. The reward is defined as such: if the user clicks on the recommended news link there would be a $+1$ reward. If not, it would result in a -1 reward. When defining the Actor-Critic model, we will of course split this into their respective parts and approach each separately. This will follow the overview described above for the general actor-critic framework. Our experiment is to try the different step size and see what happens with different step size for the model.

3.1 Data File Preparation

REF: https://github.com/msnews/msnews.github.io/blob/master/assets/doc/introduction.md

We used MIND dataset and generated two input files to help our agent learn the policy of recommending news for users. MIND dataset for news recommendation was collected from anonymized behavior logs of Microsoft News website.

3.1.1 Input File 1 - behavior.csv

This input file contains 4 columns including newsId, userId, ratings, and timestamp. Rating information is used as the rewards in our model. If we have users clicked on the news from recommandation list recorded in MIND dataset, we would think this action means that user is interested in the specific news and will have the Ratings(Reward) as $+1$. For the news that were not clicked

from recommandation list recorded in MIND dataset, we would think users were not interested in those news at all and we would give the rating for those news as -1. This file will help our agent know whether further actions bring them positive/negative rewards. As a result, it helps the agent learn the best action it would take at specific state. The behavior data has the following columns: $n = 50,000$, *UserId: Id of user, Time: impression time, History: The news click history, Impressions: List of news displayed in this impression and user's click behaviors on them*

3.1.2 Input File 2 - news.csv

The second data file is the news.csv file, which contains the basic information about each news. It has the news ID, news category and the URL of the news and so on. What we mainly need is the news ID, and entities column so that we can extract the embedding information from the embedding data. It has following columns: $n = 51,281$, *News ID: the id for the news, Category: type of the news, Sub-category: sub-type of the news , Title, Abstract, URL, Entities: Contains the embedding ID for extracting the embedding information, Abstract entities: contains the embedding ID for the abstract information*

3.1.3 Input File 2 - newsembeddings.pkl

This input file contains more than 50k news as the key, and one tensor with 100 elements for each news. The tensor with 100 elements will be used as the descriptions of the news. The value will be between -1 and $+1$ to represent whether the news is positive-related or negative-related to specific topics.It has the following columns: *Embedding ID, Embedding values*

3.2 Data preprocessing:

First, missing values in the embedding data are filled with 0. The embedding data ID are in Wikidata Q identifier format, wich contains the embedded result from the news data. In the news data, we have title entities and abstract entities, which contain the Q ID for the embedding data. We matched the ID from news data and the embedding data to generate the embedding data for our model.

In the behavior.csv data, we used the news id in the history and the impression information to match with the news embedding ID, so that we can replace the Q id with the news id. The behavior.csv data is generated with only user id, news id, and ratings(+1, -1) and time stamp of the news that are looked through.

The major procedure can be described as the following:

- Match each user with each news
- Match news with each embeddings
- Calculate the mean values for each embeddings to generate news embeddings
- Scale the embedding values for model

4 Methodology

The model framework we are using for this project is the Actor-Critic Framework(ACF). In the Figure1 we can see that it has two major components. The first component is the Actor network(a.k.a Policy network). This network is mainly used for generating the actions a based on its state s. In ACF, the user state means that it is the embeddings of user's n latest positively interacted items. Then those embeddings are fed into the state representation module to state vector s. For example, we use t to denote time stamp, then the state can be defined as $s_t = f(H_t)$ where $f(.)$ is the state representation module.$H_t = i_1, i_2, ..., i_n$ stands for the embeddings of the latest positive interaction history(i.e. latest news look-through history). When recommender recommends an item with positive feedback, then the state is updated to $s_{t+1} = f(H_{t+1})$ where $H_{t+1} = i_2, i_3, ..., i_t$. If negative feedback, then $H_{t+1} = H_t$. In the end, the state s is transformed into an action $a = \pi_\theta(s)$ as the output of the Actor network. [7]

Then we have the Critic network, which is shown in the Figure1 middle part. This is a Deep Q-network parametrized as $Q_\omega(s, a)$ to approximate the true state-action value function (Q value function) $Q^\pi(s, a)$ The input of the Critic network is the concatenation of the actions generated by the Actor network and the original state representation module. The Q-value function embodies the traits of the action policy generated by the Actor network(Policy network). The parameters of the Actor network are updated in the direction of improving the performance of action a(i.e. boosting $Q_\omega(s, a)$. By using the deterministic policy gradient theorem [11], we will be able to update the Actor network by using the sampled policy gradient, which is trying to maximize expected Q-value of the policy:

$$J(\theta) = \mathbb{E}[Q(s, a)|_{s=s_t, a_t=\pi_\theta(s_t)}]$$

$$\nabla J(\pi_\theta) = \frac{1}{N} \sum_t \nabla_a Q_\omega(s, a)|_{s=s_t, a=\pi_\theta(s_t)} \nabla_\theta \pi_\theta(s)|_{s=s_t}$$

And the at the same time, the Critic network is updated using the temporal-difference learning approach, by minimizing the loss function:

$$L = \frac{1}{N} \sum_i (y_i - Q_\omega(s_i, a_i))^2$$

where y_i is defined as the following:

$$y_i = r_i + \gamma Q_{\omega'}(s_{i+1}, \pi_{\theta'}(s_{i+1}))$$

The target network [5] technique is also used in our ACF where ω' and θ' are the parameters of the target Critic and Actor network. The whole parameter space are updated based on DDPG algorithm. [11]

Here we will not elaborate the state representation module. Here we used the proposed method by this paper. [7]

5 Analysis of Results

Here we utilized the reconstruction error based on the recommender system as the main check for the model. Firstly, we used the model to generate the recommended news for the user, call it as x.

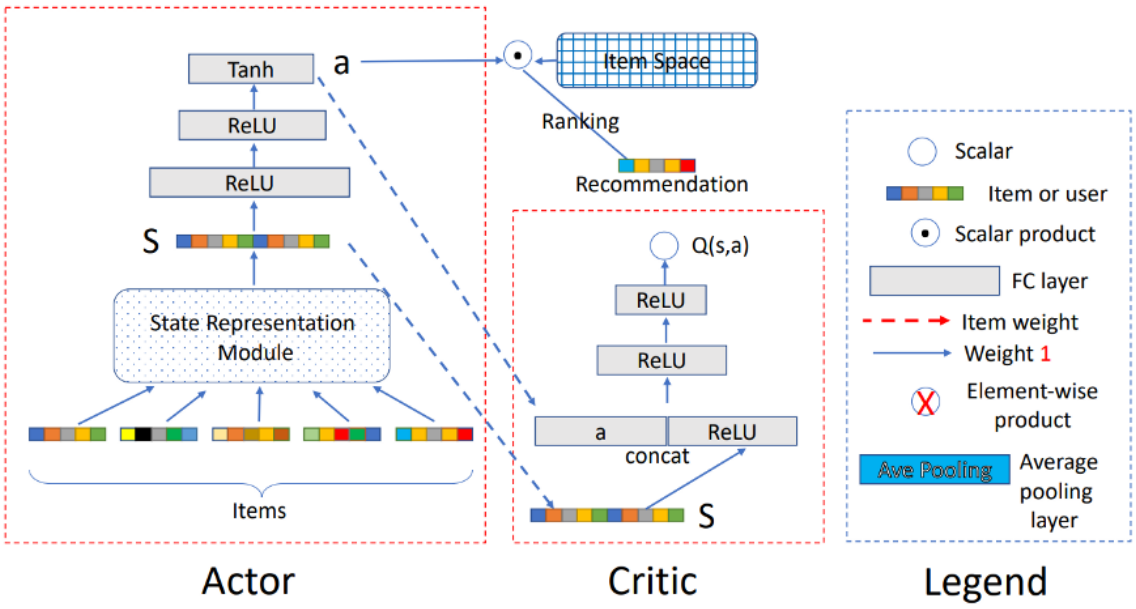


Figure 1: Actor Critic Framework

Then we feed that generated item embeddings x into 5 layer Autoencoder. Then we calculate the reconstruction error of x and $ae(x)$. The reconstruction error is defined as the following:

$$err = (x - ae(x))^2$$

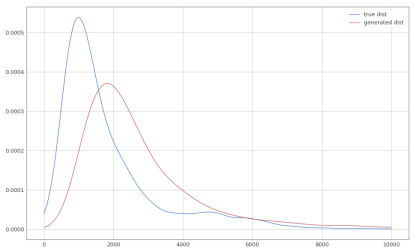
The smaller the error is, the better the reconstruction is. We do the same thing for the truely positively rated item also. Then we put both reconstruction error into the Gaussian kernel density function and generate the distribution curve. If their curve matches well, it means that the recommendation system works well. If not, it means the recommendation system does not work.

6 Conclusion and Future Applications

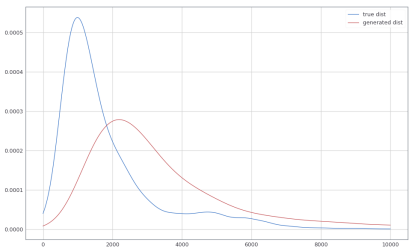
From the Figure2, we can see that the true distribution of recommended items are pretty close to the Actor Critic model system generated result for recommended items. It tells us that the RL based recommendation system works well based on the data we have.

The value policy plot gives us a sense of how the model was trained. We can see that the value function (Q-value function) converges well after a long run. However, the policy curve shows that it begins to converge(curve starts to be flat) but changes to diverge after 12500 iterations.

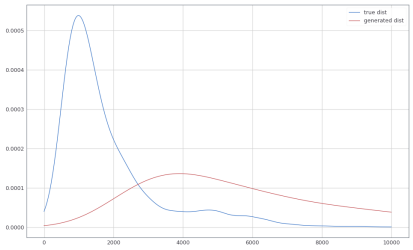
After long run, the generated distribution starts to become more and more dissimilar to the true distribution. Here we believe that the model starts to overfit with more and more iterations goes on. In the Figure2, we can see that the red curve becomes more and more flat in the future runs. The generated items by the RL model does not represent the truely positive feedback items from



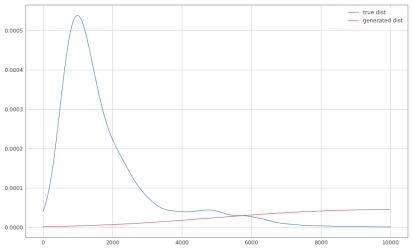
(a) 6000 iterations



(b) 10000 iterations

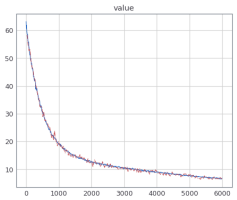


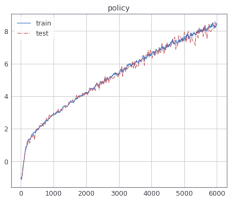
(c) 15000 iterations



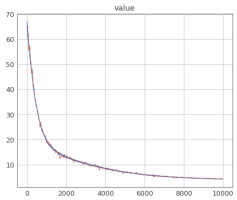
(d) 20000 iterations

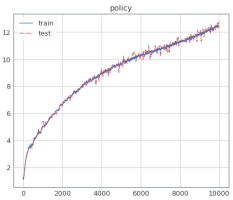
Figure 2: True recommendation Reconstruction error kernel density plot vs Generated recommendation Reconstruction error kernel density



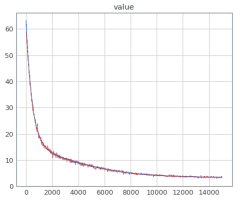


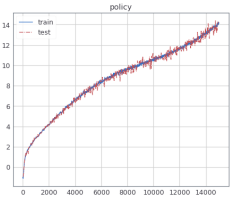
(a) 6000 iterations



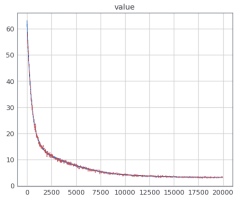


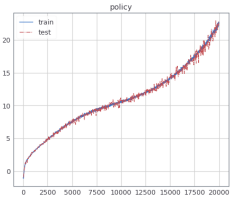
(b) 10000 iterations





(c) 15000 iterations





(d) 20000 iterations

Figure 3: Value Policy plot for training data and testing data

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.4000  0.3981  0.4020  0.3990  0.3983  0.3995  0.0014
MAE (testset)     0.2208  0.2203  0.2227  0.2201  0.2207  0.2209  0.0009
Fit time          57.32   58.19   59.12   58.04   58.28   58.19   0.57
Test time         12.12   11.29   15.55   10.80   11.06   12.16   1.75
```

(a) SVD

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.3684  0.3677  0.3709  0.3669  0.3670  0.3682  0.0015
MAE (testset)     0.1337  0.1337  0.1349  0.1335  0.1330  0.1338  0.0006
Fit time          33.98   31.03   32.36   32.53   31.15   32.21   1.08
Test time         92.26   93.58   92.75   91.26   97.10   93.39   2.00
```

(b) KNN

Figure 4: SVD and KNN result

user anymore. Number of epochs(iterations) as a hyperparameter would be an important factor to consider.

Different initilized policy value might have different result. We might want to try to figure out what would be the best initilized policy value to start with.

Reinfocement learning models are mainly used for some gaming application or interation based application including user and environment components. Applying the model on the recommendations data is still very novel nowadays. State representation module is a very important component so that we can transform our available data to make it doable for RL based models.

7 Discussions and Limitations

The main point of the project to implement the RL based recommendation model for news data and get a sense of how RL models are run for recommendation purposes. In the future, we can focus on more on the evaluation method on the model and generate the some kind of rating prediction, so that we can compare some evaluation metric with our traditional recommendation model result. Here are some traditional model result (SVD and KNN) shown in Figure4 From the result in Figure4 we can see that KNN model achieves smaller RMSE for predicting the ratings.

Different policy initialization gives us different result. In the future, we should do more investigation of the policy initialization algorithms and try different method to see whether they are giving us different result and figure out why that happens.

Actor-Critic Framework is just one of the common framework in reinforcement learning. In the future, we can try to implement more different model(i.e. Twin Delayed Deep Deterministic Policy Gradient, Trust Region Policy Optimization) and compare their results.

Futhermore, using the evaluation metric such as precision @k or recall @k would be a nice evaluation for our project. Due to the time limitation and computing resource limitation This part was not ready yet. In the future, we can check how to generate the precision @k evaluation for both traditional methods and Reinforcement Learning based methods.

References

[1] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. "Reinforcement Learning based Recommender Systems: A Survey". In: (2021).

[2] Anna B. *Recommender Systems - It's Not All About the Accuracy*. June 2016. URL: https://gab41.lab41.org/recommender-systems-its-not-all-about-the-accuracy-562c7dceeaff.

[3] Roger Chua. *A simple way to explain the Recommendation Engine in AI*. June 2019. URL: https://medium.com/voice-tech-podcast/a-simple-way-to-explain-the-recommendation-engine-in-ai-d1a609f59d97.

[4] Elad Liebman et al. "Designing Better Playlists with Monte Carlo Tree Search". In: (2019).

[5] Timothy P Lilicrp et al. "DDPG: CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING". In: ().

[6] Feng Liu et al. "Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling". In: *Shenzhen Key Laboratory of Internet Information Collaboration* ().

[7] Feng Liu et al. "Deep reinforcement learning based recommendation with explicit user-item interactions modeling". In: *arXiv preprint arXiv:1810.12027* (2018).

[8] Sean McNee, John Riedl, and Joseph Konstan. "Being accurate is not enough: How accuracy metrics have hurt recommender systems". In: Apr. 2006, pp. 1097–1101. DOI: 10.1145/1125451.1125659.

[9] Giorgos Papachristoudis. *Popular evaluation metrics in recommender systems explained*. Apr. 2019. URL: https://medium.com/qloo/popular-evaluation-metrics-in-recommender-systems-explained-324ff2fb427d.

[10] Abhijit Roy. *Introduction To Recommender Systems- 1: Content-Based Filtering And Collaborative Filtering*. July 2020. URL: https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421.

[11] David Silver et al. "Deterministic policy gradient algorithms". In: *International conference on machine learning*. PMLR. 2014, pp. 387–395.

[12] Thomas Simonini. *An intro to Advantage Actor Critic methods: let's play Sonic the Hedgehog!* Mar. 2018. URL: https://www.freecodecamp.org/news/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d/.

[13] A. Zimdars, D. M. Chickering, and C. Meek. "Using temporal data for making recom- mendations". In: (2001).