User Profiling in Recommender Systems

Jacob Peterson
pet02873@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Recommender systems (RSs) are systems that look to create specific recommendations for users based on their activity on a certain service. Today recommender systems play a crucial role in providing a good user experience on many different platforms, and help users overcome the information overload problem. They are a large part of many different platforms like E-commerce (Amazon, eBay), social networking websites such as Twitter, music services such as Spotify, video streaming services such as Netflix, among others. One of the most important aspects of recommender systems is user profiling. It provides RSs with the proper information to make an accurate recommendation for a user. In this paper I will discuss various ways in which user profiling methods are being improved in order to provide better recommendations. The methods that I've decided to look at include a Hierarchical User Profiling framework, and a tag-based user profiling approach that uses game theory to find the best trade-off between generality and detailedness. By looking at these methods I'll be able to analyze how effective new approaches to user profiling can be, and how they will create better recommendations to enhance the user experience.

1 Introduction

In today's world there is an increasing number of platforms with massive amounts of information for users to sift through. This is known as the information overload problem. The information overload problem is simply the difficulty in understanding when too much information is present. This can be seen in many platforms. A clear example of this can be seen in E-commerce websites. It seems that every day there are more and more items for users to purchase. While these services are convenient, the amount of information they contain can create stress for users and decrease their experience on these platforms. Because of this recommender systems (RSs) have been developed to help guide users by generating personalized content for them. RSs are algorithms that suggest relevant items to a user based upon their previous interactions. In these systems an accurate and dynamic representation of the user's interests is extremely important. This representation is known as a user profile. User profiles are used to generate recommendations for the user. The profiling phase is often the most important step in RSs.

This paper provides background information that is necessary to understand concepts of user profiling. Then it discusses two approaches to improve user profiling. Section 3 covers a hierarchical user profiling framework that is capable of representing a user's interests at multiple levels. Then in Section 4 a tag-based user profiling technique is covered that uses game theory, which looks to find the optimal trade-off between the generality and detailedness in the representation of a user is covered. Finally the paper goes over conclusions drawn from these frameworks.

2 Background

This section of the paper provides background knowledge necessary to understand the concepts that are covered and is crucial for understanding user profiling methods.

2.1 Neural Networks

The concept of neural networks (NN) is very important when trying to understand RSs and many other deep learning techniques. Fundamentally NNs are comprised of neurons and connections between them. A neuron is a function that may take one or many inputs and generate an output. Neurons are organized into layers such that, neurons in a given layer provide their output as input for the next layer. The connections between neurons are also very important as they connect the output of a neuron and send it to the next neuron as input. Connections maintain a weight parameter. The weight of a connection determines how important it will be to the target neuron and is determined through a process known as training. The training process involves finding a set of weights that give the desired results for the network. The process is iterative, where each iteration provides a small update to the set of weights. Through training we seek to find a set of weights that will give accurate results for whatever problem the network is trying to solve. One useful example of a problem that a NN can solve is reading handwritten characters.

2.2 Recurrent Neural Networks

In recommender systems, recurrent neural networks (RNN) are a common approach to generate recommendations. A RNN is very similar to the traditional NN. However, RNNs introduce the idea of memory by including different kinds of connections. Unlike NNs, where the output of each neuron is fed forward through the network, RNNs sometimes feed

output from layers back into the previous layer. This is a very effective way of modeling sequences of input and dynamic behavior, which is particularly important in the hierarchical user profiling framework. Even though recurrent neural networks are more complex due to their nature, the methods for training RNNs are similar to that of NNs.

2.3 User Profiling

Many RNN based recommendation systems generate a user's profile vectors to create recommendations. A profile vector is a vector that hierarchically contains things that the user is interested in, and are meant to "formally represent users' interests by deeply understanding their historical interactions" [4]. For example, if user u is interested in three categories of movies, action, comedy, and romance in that order (where the category they are most interested in is action, then comedy, and then romance), then their profile vector may look something like $p_{categories_u} = \{54, 0, 0, 36, 0, 14, 0\}$. This vector represents the user's interest in all categories. What that means is that categories that the user is not interested in will be represented with a zero in the vector, and categories that the user is most interested in will be represented by numbers that correspond to their level of interest. From the example, the user is most interested in action movies. In the profile vector, $p_{categories_u}$, this is represented with a 54. In recommender systems this is used for candidate generation, where a candidate is a possible recommendation. So, if our RS generated $p_{categories_u}$, then we would almost certainly recommend things that are similar to action movies to u. One limitation of this technique in many systems is the fact that they lack the ability to represent a users' interests at various levels. This means that these RSs will only generate one profile vector instead of multiple. The *Hierarchical User Profiling* framework seeks to solve this problem by generating a set of profile vectors to represent user interests at varying levels of granularity.

2.4 Generality and Specificity

Another important aspect of recommender systems is the idea of generality and specificity. This is important for creating satisfactory recommendations. A user profile needs to be both general enough to adequately model a user, and specific enough to pinpoint topics that will be interesting or niche to them [3]. This requirement is addressed in Section 4.3 by using a two-player zero-sum game.

2.5 Two Player Zero-Sum Games

This game is represented by a *payoff matrix* and is played by each player simultaneously. A simple example of a payoff matrix can be seen in Table 1. There are two players that play the game, a row player and a column player. The row player picks a row and the column player picks a column at the same time. Each round the result is found in the payoff matrix which represents each player's loss or gain; one player's loss is the others gain. How the player plays the game is represented by a normalized probability distribution, where each choice a player can make is given a probability. For example, if the row player has a .25 probability of choosing row X, then they will choose that row approximately 25% of the time.

	Even	Odd
Even	1/-1	-1/1
Odd	-1/1	1/-1

Table 1. The payoff matrix for a simple game where each player chooses an odd or even integer. If the sum of the two numbers is even then the row player gains a point, if the sum is odd then the column player gains a point. This is a zero-sum game as the result of each round nets zero utility.

3 Hierarchical User Profiling

Hierarchical User Profiling (HUP) is a user profiling framework that looks to decipher a user's real-time interests. It does this by categorizing these interests into levels of granularity. To do this it creates a collection of profile vectors for multiple levels, which are the micro-level, item-level, and category-levels. Doing this creates a more informative model of the user. HUP is meant to solve some of the problems that other modern recommender systems have. These limitations are, that they don't build hierarchical user profiles that can show granularity in user interests, they don't harvest as much data from users that is easily available, and that they don't cope well with the fact that user interests are fluid and constantly changing.

3.1 Layered Approach

HUP overcomes these limitations by using a Pyramid Recurrent Neural Network (PRNN) to generate a user's profile vectors. A PRNN consists of a series of RNNs that are layered on top of each other. Most traditional recommendation methods use a single RNN to generate recommendations. HUP's PRNN uses a layered approach to achieve granular classification of user interests. Each layer in the PRNN is an RNN that focuses on generating a profile vector for that layer. These layers are the micro-level RNN layer, the item-level RNN layer, and several category-level layers. Each layer transforms a user's context vectors, discussed in Section 3.5, into a hierarchical user profile, which is simply the collection of a users profile vectors. [4]

3.2 Input and Embedding

The first layer of the PRNN is the input and embedding layer. Embedding is essentially the process of mapping variables to a vector with lower-dimensionality. The input and embedding layer considers a user u and their set of micro-behaviors (MBs). A micro-behavior is a finely-granular action that the

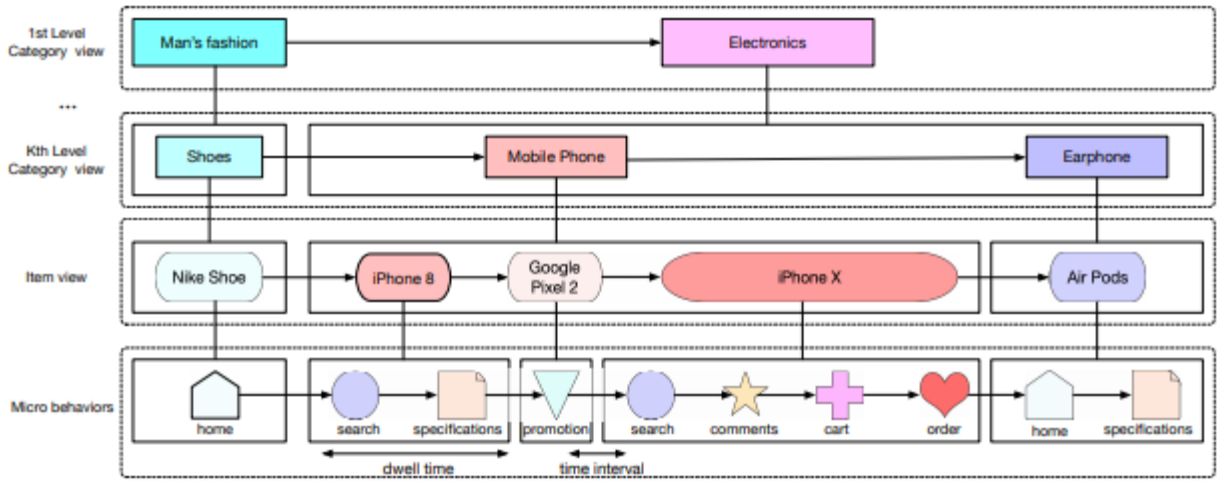

Figure 1. A Hierarchical view of micro-behaviors showing user interests with increasing granularity [4].

user performs. Things like searching, browsing, viewing an item's specific details, adding an item to the cart, and purchasing an item are all examples of micro-behaviors that are considered. The users MBs are the input for the recommendation model. The set of a user's MBs is $X = \{x_1, x_2..., x_n\}$. Formally, a micro-behavior is a sextuple, $x_i = (t_i, v_i, c_i, b_i, d_i, g_i)$, containing pertinent information for an action. It holds the time the action took place (t_i) and the item the action took place on (v_i). The categories the item is in, c_i. Which is a set of categories that an item belongs to, listed as $\{c_i^{(1)}, c_i^{(2)}, ..., c_i^{(k)}\}$, $c_i^{(1)}$ is the most granular, meaning that it represents the most generalized user interests in categories, on the other hand $c_i^{(k)}$ has the finest granularity, meaning that it represents to most specific level of category interests. For example, in Figure 1 the *Google Pixel 2* is in the *Mobile Phone* category and the *Electronics* category. The *Mobile Phone* category is more specific than the *Electronics* category, meaning it has finer granularity. Also included is the type of MB (b_i); this could be things like searching for an item, or adding an item to a cart. The dwell time (how long an action took place, d_i), and the time interval from this MB and the next (g_i) are also included. Each MB, x_i, is transformed into an embedding vector e_i, which is a low-dimensional dense vector.

3.3 Subsequent Layers

Figure 1 shows an example of how a user's interests have been arranged hierarchically. At the bottom are the micro-behaviors; here you can see things like searching for an item or adding an item to the cart, along with the MBs dwell time. Above this is the item layer. This shows the specific items a user is interested in. An example of this from the figure can be seen where user has searched for, and looked at specifications for an *iPhone 8*; because of this it is inferred that the user is interested in the *iPhone 8*. Further above this are the category layers. The iPhone 8 is in the *Mobile Phone* category and the *Electronics* category. This shows decreasing granularity, the item view being the finest-granularity and the 1st Level Category view being the most coarse grained. The PRNN models this structure in the following layers.

The second layer of the PRNN is the micro-level RNN layer. This layer is meant to show a user's most fine grained interests based on the very specific actions, micro-behaviors, they have performed. The input for this layer at time i is e_i and comes from the embedding layer. The hidden state of the micro-level RNN is updated after taking each MB as input. The output of this layer is given to the next layer up, the item-level layer.

This layer shows a users interests in specific items at a specific time. The input taken by the RNN of this layer is the concatenation of the item embedding (e_{v_i}) and the output of the micro-level layer. The state of the item-level RNN is updated when the user switches their focus from one item to another. This can be seen in the *item view* row of Figure 1. The user is initially focused on *Nike Shoe*, then switches their focus to *iPhone 8*. The output of this layer is given to the category-level layer.

The category-level RNN layers model a users category level interests. These are the most granular of all of the layers, and work to extract the users most generalized interests. Figure 1 gives a few examples of categories. Items are organized into a hierarchy of categories; in Figure 1 the item *iPhone 8* is in the *Mobile Phone* category and the *Electronics* category. *Electronics* is in a higher level category that includes *Mobile Phones*. This layer needs to be run on each of the category layers that are being considered at time i, so the category-level RNN is ran K times, once for each category layer. The input for the Kth category at time i is the concatenation of the category embedding ($e_{c_i}^{(K)}$) and the output of the item-level layer. For higher-level category layers the input is $x_C^{(k)}$, the combination of the category embedding

($e_{c_i}^{(k)}$) and the output of the (k-1)th level category layer. For each category-level the hidden state is updated when the user shifts their focus from one category to another. A simple example from Figure 1 can be seen in the *1st Level Category view* where the user shifts their focus from *Men's fashion* to *Electronics*.

3.4 Behavior-LSTM

A long short-term memory cell (LSTM) is a component of an RNN that excels at making predictions from sequences of data. LSTMs were created to deal with the *vanishing gradient problem*, which is a problem that can be seen when training RNNs. Essentially during the training process connection weights are updated proportional to a partial derivative of an error function, known as a gradient. The problem comes from the fact that sometimes this gradient approaches zero or infinity, so when the connection weights are updated they will be either vanish (they approach zero) or explode (they approach infinity). This means that the RNN will effectively stop training properly [8][7]. Additionally, LSTMs are a solution to the short-term memory problem that RNNs tend to have. This problem comes up when RNNs deal with long sequences of data. When this happens RNNs have a hard time considering earlier information as well as current information. To overcome this, LSTMs include mechanisms called gates. Gates are able to learn which data is important and should be kept, or which data is unimportant and should be thrown out. Typically an LSTM contains at least three gates, a forget gate, input gate, and output gate. Each gate serves an important purpose. The forget gate decides which information should be kept or discarded, the input cell works to update the cell by deciding which values will be updated, and the output gate decides what next hidden state of the LSTM will be. [6]

Due to the fact that users' interests are constantly changing Behavior-LSTM has been created to track a users "real-time interests" more accurately. Time-LSTM is a similar LSTM that tracks a user's sequential behaviors by tracking the time between user purchases. However, it cannot track micro-behavoir type and dwell time information. [4] These two things are crucial to HUP so the creation of Behavior-LSTM was a necessity. There are two key components that have been implemented. First, the time gate, $\mathcal{T}$, approximates how much information should be maintained and given to the next cell state by capturing the time intervals between each MB. Secondly, the behavior gate, $\mathcal{A}$, calculates the importance of the current behavior using its type and dwell time information. There is one exception to this; in the micro-level layer only the behavior type is considered as many MBs are instantaneous events that do not have any dwell time associated with them. However, each MB's type is very important for modeling a user's interests. Furthermore, the behavior gate is extremely important in the item and category layer as dwell time can be a very good way of determining a user's interest in a category or item. For example Figure 1 shows the dwell time for various items. The dwell time for *iPhone X* is much longer that that of the *iPhone 8*, so, it can be surmised that the user is more interested in the *iPhone X*. The input for Behavior-LSTM is a tuple, $(e_t, a_t, \Delta t)$. e_t is the embedding vector of the input, a_t is the embedding vector of the behavior type (or dwell time in the case of MBs), and Δt is the embedding vector that contains information about the time between this behavior and the next.

3.5 Attention Layers

In a given RNN an attention mechanism can be implemented to allow the network to focus on more specific parts of the given input. This can increase the accuracy of predictions that the RNN will make, thus improving performance [1]. Additionally, through using attention, one is able to minimize long-term dependency issues, and provide better interpretations of data [4]. An attention mechanism generates a series of context vectors. Context vectors are what allow the network to focus on certain parts of the input.

HUP uses multiple attention layers which are a part of their corresponding RNN layer. For example, the item-level attention layer is a part of the item-level RNN. The layers are referred to as the micro, item, and category-level attention layers. The context vectors that the attention layers generate are named s_m (micro-level), s_i (item-level), and a set of category context vectors $s_c = \{s_{c_1}, ..., s_{c_K}\}$. Using the attention mechanism allows for fully connected layers in the neural network. Each layer of the RNN transforms the user's context vectors from its attention layer into hierarchical user profiles in corresponding levels with increasing granularity. [4]

3.6 HUP Experimentation and Results

HUP has been tested by generating recommendations at the item and category level. The system is given a user u and their sequence of micro-behaviors. It then creates a set of hierarchical profile vectors for u. These represent their interests in items and categories by understanding their historical interactions. [4]

Item level recommendations are generated by first selecting a set of candidate items that are similar to those that the user has previously browsed. Then the *cosine similarity* is calculated for each candidate item embedding, e_{v_i} and the user's item level profile vector p_i as a ranking score. Cosine similarity is simply calculated by finding the cosine of the angle between two vectors. A smaller cosine means that the two vectors are more similar. The top scored items are then selected to be recommendations. A similar process is then done for the category layer. However, here the category embedding, $e_{c_i}^{(K)}$, and the Kth category profile vector $p_{c_u}^{(K)}$ are used.

The same dataset that was used in study [10] and has been used to test RIB (another micro-behavoir based recommenation framework) has also been used to test HUP. It uses the *"JD Micro Behaviors Datasets"*, a collection of user's micro-behaviors in two categories, appliances and computers.

The results of HUP were compared with those of several baseline methods and three state-of-the-art RNN-based recommendation frameworks. To measure the effectiveness of each system two common metrics, Recall@K and MRR@K, were used to compare HUP to the other models. The experimental results have shown that HUP outperforms all other methods by a significant amount, especially in the item recommendation category. More specifically, "HUP outperforms state-of-the-art method by 3.4%, 6.1% in Recall@20 and 6.7%, 9.1% in MRR@20 for the 'Appliances' and 'Computers' datasets respectively" [4]. Recall@20 and MRR@20 simply refer to the two metrics calculating how well 20 recommendations were generated for a user. For category recommendation performance gains are smaller, but still notable.

4 Tag-Based user profiling using game theory

The most important aspect of a recommender system is the user's experience. Creating personalized content for a user is an effective way to increase their satisfaction. To achieve a good experience an RS must be able to ascertain a user's particular tastes and preferences. A common way of doing this is to create a user-profile using item tags and looking at how often users associate with certain tags. Item tags can be used to explain an item, as we can see this in the Internet Movie Database (IMDb). For example, *The Avengers (2012)* has many tags associated with it, including things like superhero, alien invasion, and marvel cinematic universe. While tag-based strategies are commonplace, it can be argued that many of the frequentist approaches oftentimes don't obtain a proper level of specificity in the users preferences [3]. A frequentist approach simply generates a user profile based on the amount of time a user associated with a tag. To address this a game theoretic approach can be employed to find a more suitable trade-off between the generality and the detailedness of the user profile. In this section I will cover a user-modeling approach that uses tag-clustering to create topics and game theory to learn a users preferences to create a more satisfying user experience.

4.1 Background on Tag-Based user profiling

To analyze this method tag-based user profiling must be understood. Firstly, there is a user (u) and a set of users (U) such that $u \in U$. Now, an item i is in the set of all items I, $i \in I$. Each item can be described by a subset of tags, T_i, from the set of all tags T. So, $\forall_{i \in I} \exists T_i \in T$. If you look at a specific user (u), then the set of items they have interacted with would be I_u [3]. When a tag is used more by u it will have a higher intensity, meaning that the user has more interest in that tag. Tags are often ranked hierarchically by their intensity and selected for the user-profile based upon that information. This is effectively how a recommender system learns a user's profile. This is similar to what is done in Section 3, where HUP creates hierarchical profile vectores. However, it has been proposed that additional information should included, like the co-occurrence of tags (what tags are being used together). Including the co-occurrence of tags allows for a user-profile to carry even more information than what can be achieved by a typical frequentist approach.[5] For example the user may have a preference for sci-fi action movies, that may lead to the action and sci-fi tag occurring often together. Including this information, that the two tags together has meaning to the user, recommender systems will be able to generate more accurate recommendations that will further satisfy the user.

Cluster Name	Cluster Tags
British Empire	british army, british empire, colonialism, dublin, independence, ireland
Teenager	teen comedy, teen drama, teen movie, teenage romance, teenage sexuality, teenager
Dog	dog, lassie, pet, talking dog

Table 2. This chart from [3] shows examples of tag clustering. The left column shows the name of a topic. The right column shows the various tags that are included in the topic.

4.2 Tag Clustering and Topics

Tag clustering is a process in which item tags are grouped into topics. Topics are seen as groups of similar or related tags. The dataset used is from *The Movie Database* (TMDb). TMDb is a community built movie and TV database that contains metadata for over 26,778 movies [3]. The metadata includes tags that have been created and assigned to movies by users. The database includes around 17,000 tags; for this reason topics are used instead of tags themselves as tags often represent aspects of an item that are too specific. Topics are generated using the hierarchical agglomerative clustering algorithm to build clusters. This is seen as the most effective way of clustering in this context [3]. The algorithm generates a set of clusters that are referred to as C. Each topic is named after the most occurring tag in the cluster. For example, the *teenager* cluster includes the tag "teenager" as the most popular tag. There can also be many different tags that represent the same meaning. Table 2 shows examples of tag clustering, where you can see tags in the right column and the cluster they belong to in the left column. The *teenager* cluster includes things like "first love", "teen drama",

and "teenage romance", which all convey similar meaning. So it makes sense to group them together.

4.3 User-profiling using game theory

The proposed method for finding a satisfactory trade-off between the generality and detailedness of recommendations using a game theoretic approach looks to understand each facet of consumed items [3]. There are two particularly important attributes that the method looks the maintain for a user profile. First, the user-profile needs to be *general* enough. This means that a user's profile contains enough general topics that are highly relevant to them. If the user is very interested in action movies the user profile will reflect that. Secondly, a user-profile must be *highly specific*. This means that the user-profile will include topics that the user rarely chooses but are important to them. Again, if the user is highly interested in action movies, then similar genres will be included that are highly specific to their interests.

These two concepts create a competitive environment and can be transformed into a two-player zero-sum game. The row player chooses topics that maximize the number of items covered (this player looks to maintain generality). Their opponent, the column player, chooses more specific items for the user. This does not consider the user's popular topics, rather it chooses items that are more niche to them (highly specific items). These two strategies force the players to select topics that are both general and specific.

In a two-player zero-sum game the outcome of each round can be found in the *payoff matrix*, M. The row player P and the column player Q play at the same time. The row player picks an item from all items that the user has interacted with, while the column player picks a topic from the set of topic clusters, C. The two players play with different strategies. Player P aims to pick a row such that it will minimize its losses. Player Q looks for a strategy to maximize its gains. More formally, P picks a row according to probability distribution p, and Q picks a column according to probability distribution q. p dictates the probability of the row Player P choosing each row. Likewise, q dictates which column Player Q will choose. Using linear programming it is possible to find the exact distributions of p and q. However, due to the size of the payoff matrix, the exact distributions of p and q are computationally difficult to find. Because of this the *fictitious play* algorithm has been implemented to approximate their distributions. Fictitious play is a learning process in which two players are playing a finite game repeatedly. The technique assumes that after each round of play the player will update their strategies (p and q) in response to the outcome of the round. [2] As play goes on the strategy distributions of each player will converge on the *Nash Equlibria*, which is defined as the solution to a non-cooperative game. The solution to a two-player zero-sum game is the final probability distribution that each player will play by.

4.4 Results From Ficticious Play

After ficticious play has completed the result is two probability distributions (p and q) that Player P and Player Q will play by according to their strategies. Again, Player P seeks to minimize its losses, and Player Q seeks to maximize its gains. The important part of this is distribution q, which shows the probability that a given topic will be chosen from C. A topic that has a high probability of being chosen will be highly relevant to the user. Conversely, topics with a low probability have a low relevance to the user. Given this information a user profile can be generated that includes topics that are both general (popular with the user) and highly specific (niche to the user). Using this profile recommendation systems are able to provide dynamic recommendations to the user.

5 Conclusions

It is clear that recommender systems are very important and for many people affect their day to day lives. With their increasing popularity, applications of RSs can be seen in many places. Video streaming services, like Netflix, will recommend media for its users. Likewise, E-commerce websites, like Amazon and Ebay, will recommend items for a user to purchase. These features are very convenient for users and heighten their experience on these platforms by generating personalized content specific to them.

In this paper, I've investigated two techniques which aim to create better representations of user interests in RSs, with the goal of creating more accurate and useful recommendations for the user. The first technique, HUP, creates a hierarchical representation of a given user's interests by creating a hierarchical user profile that is capable of representing the user's interests at various levels. This technique takes advantage of the user's micro-behaviors to create recommendations at the item and category level. The second technique for modeling the user's interests uses a tag-based approach and game theory principals to create a user profile that is capable of describing the users interests with both generality and detailedness. Both of these frameworks are flexible and can be used in many applications. Creating better representations of what is important to users is crucial to the advancement of recommender systems with the goal of creating an improved user experience. Creating better representations of user interests, which more closely reflect their interests, is one effective way of doing this and is an important field of study.

References

[1] N. Arbel. Attention in rnns, Mar 2019.

[2] U. Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135:572–578, 02 2007.

[3] G. Faggioli, M. Polato, and F. Aiolli. Tag-based user profiling: A game theoretic approach. In *Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization*, UMAP'19 Adjunct, page 267–271, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Y. Gu, Z. Ding, S. Wang, and D. Yin. Hierarchical user profiling for e-commerce recommender systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 223–231, New York, NY, USA, 2020. Association for Computing Machinery.

[5] E. Michlmayr and S. Cayzer. Learning user profiles from tagging data and leveraging them for personal (ized) information access. 01 2007.

[6] M. Phi. Illustrated guide to lstm's and gru's: A step by step explanation, Jun 2020.

[7] Wikipedia contributors. Long short-term memory — Wikipedia, the free encyclopedia, 2021. [Online; accessed 1-April-2021].

[8] Wikipedia contributors. Vanishing gradient problem — Wikipedia, the free encyclopedia, 2021. [Online; accessed 1-April-2021].

[9] Wikipedia contributors. Zero-sum game — Wikipedia, the free encyclopedia, 2021. [Online; accessed 9-March-2021].

[10] M. Zhou, Z. Ding, J. Tang, and D. Yin. Micro behaviors: A new perspective in e-commerce recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 727–735, New York, NY, USA, 2018. Association for Computing Machinery.

 UMM CSci Senior Seminar Conference, May 2021 Morris, MN.