MŰEGYETEM 1782

Session-Based Recommender System with Recurrent Neural Networks

Ömer Halit Cinar – EGX6CZ

Advisor: Grad-Gyenge László György

Introduction

In the dynamic world of online retail, personalized recommendations have become a not only an added benefit for customers, but an expected part of the shopping experience. As a student fascinated by the intersection of data science, I undertook a project that allowed me to further explore this aspect of online retail. The goal of the project was to implement a session-based recommender system using recurrent neural networks (RNNs) on a popular retail dataset - the RetailRocket dataset.

A recommender system is an intelligent system that makes personalized suggestions to users about products, services, or information. Recommender systems are used by large companies like Netflix or Amazon to improve the user experience. The focus of this project was to explore session-based recommender systems that offer recommendations based on a user's activities within a single session.

Traditional machine learning methods often reach their limits when dealing with sequential data, as these methods typically treat data points as independent. However, in a session-based recommender system, a user's previous actions within a session are likely to influence their subsequent actions. RNNs has the ability to handle sequential data therefore it offers an interesting way to solve this problem.

Background & Related Work

Recommendation systems have become an important tool in the arsenal of any digital platform that wants to increase user engagement and satisfaction. Traditionally, these systems have relied on collaborative filtering or content-based methods. Collaborative filtering recommends items to a user based on the preferences of other users with similar behavior. Conversely, content-based methods provide recommendations based on the characteristics of articles that a user has previously interacted with.

As the digital landscape evolved, so did the complexity and diversity of user behavior. As a result, these traditional approaches gradually reached their limits. For example, they often required a significant amount of historical data about user-item interaction to work effectively. However, in many scenarios, such as when a new user visits a website, this interaction history is not available, leading to the cold start problem.

To overcome these challenges, the concept of session-based recommender systems has become increasingly popular. Unlike traditional systems that rely on long-term user history, session-based recommender systems focus on a user's actions within a single session, which makes them robust to the cold-start problem.

However, dealing with session data presents several challenges. These data points are sequential, meaning a user's actions are often influenced by their previous actions within the same session. Traditional machine learning models that treat data points as independent have difficulty with such data.

This is where recurrent neural networks (RNNs) make their grand entrance. RNNs have the unique ability to process sequential data because they maintain a hidden state that captures information about the progression of the sequence. This makes RNNs particularly well suited for session-based recommendation tasks.

Dataset Overview:

The project uses the RetailRocket Recommender System dataset, a comprehensive collection of real-world user behavior data from a large online retail platform. The dataset contains a variety of user interactions, categorized into events such as 'view', 'add to cart', and 'transaction'. However, for the purposes of this project, we focus exclusively on 'View' events, which indicate that a user has viewed a particular product.

The information which the dataset has:

- Event: describes the type of interaction between a user and an item. For the purposes of this project, we will only consider the 'view' event, which represents users visiting specific item pages.
- VisitorID: Unique identifier for each user.
- ItemID: Unique identifier of a product that a user interacts with. For compatibility with PyTorch and efficient computation, these IDs are encoded using Sklearn's LabelEncoder.
- Item_type: A list consist of clothing, shoes, accessories, bags, jewelry

This sample is split into training (80%) and testing (20%) data to validate the performance.

The RetailRocket dataset represents a realistic picture of user-item interactions in an online retail environment. Each interaction, encapsulated as a row, contributes to a session, and these sessions form the basis for our recommender system. The overall goal is to use these sessions to predict the next product a user might be interested in, providing valuable insights for personalized product recommendations.

Data Preprocessing

For this kind of tasks, it is crucial to prepare the data in a right format. In this project, several data preproocessing steps were taken to prepare RetailRocket dataset for the Vanilla RNN, LSTM, Item2Vec models.

Firstly, a data cleaning operation was performed. Given the nature of the RetailRocket dataset, where we are only interested in the 'view' events for this project, all other events were filtered out from the dataset. This helped us focus on the interactions that provide information about the user's preferences.

Once the data was cleaned, we had to encode the item IDs. This was accomplished using a LabelEncoder, which transformed each unique item ID into a unique integer. This step was necessary because the models we used (Vanilla RNN and LSTM) require numerical input.

The next important step was to split the data into training and testing sets. A typical 80-20 split was used, with 80% of the data utilized for training and the remaining 20% for testing. It's essential to have separate datasets for training and testing to evaluate the model's performance. This ensures that the system's recommendations aren't biased towards the data it was trained on.

Next, the session data was prepared in a format suitable for the models. For this purpose, a SessionDataset class was defined, which organized the data into sequences. In this case, each sequence or session was composed of five item IDs, with the target being the item viewed next. This format is necessary for session-based recommendation tasks and is a typical approach for models like Vanilla RNNs and LSTMs that handle sequential data.

For the Item2Vec model, an additional step was taken to generate item pairs. Each user session was treated as a sentence and items as words in that sentence. Item pairs were then generated using a sliding window over the 'sentences'. The Item2Vec model learns embeddings for items based on these pairs.

Lastly, the data was loaded into DataLoader objects, which does the process of serving the data to the models during the training process. This enables efficient memory usage and optionally allows for data shuffling and multi-threaded loading.

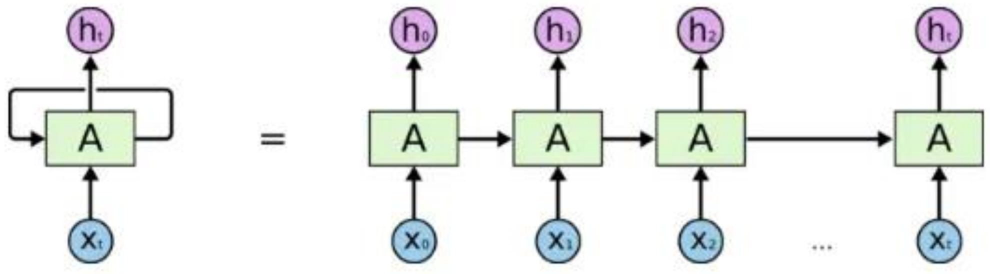
These preprocessing steps are vital to ensure the data is in the best possible format for modeling, ultimately affecting the performance of the recommender system. By ensuring the data is correctly cleaned, encoded, and formatted, we can make the most of our models and provide the best possible recommendations.

Model Architecture

The session-based recommender system in this project uses three main methods; Vanilla Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) network, and the Item2Vec model. This section will outline each model's unique architecture and how they process sequences of user-item interactions.

1. Vanilla Recurrent Neural Network (RNN)

The Vanilla RNN model is the fundamental architecture in this project. The RNN contains an input layer, a hidden layer, and an output layer.



An unrolled recurrent neural network.

The formula for the current state:

$$h_t = f(h_{t-1}, x_t)$$

Activation Function:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

‘’**W** is *weight,* **h** is the *single hidden vector,* **Whh** is *the weight at previous hidden state,* **Whx** is the *weight at current input state,* **tanh** is the *Activation funtion,* that implements a Non-linearity that squashes the activations to the range[-1.1]”

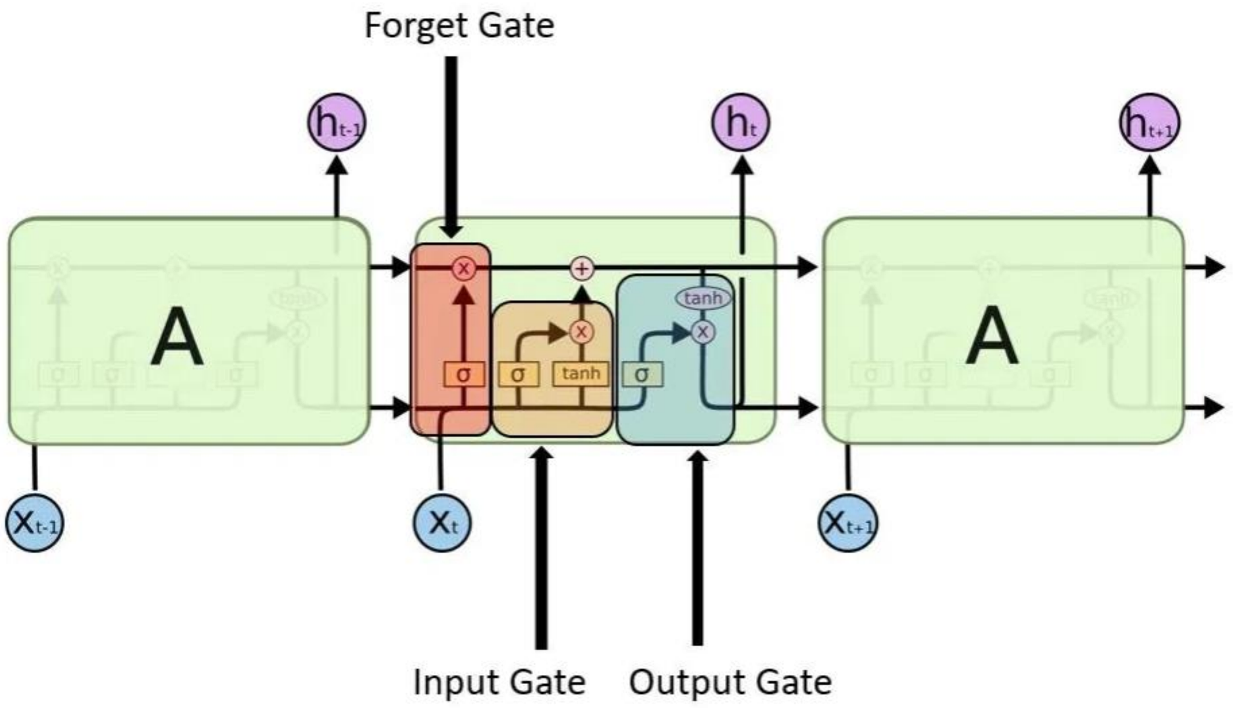
Output:

$$y_t = W_{hy}h_t$$

- The **input layer** accepts an encoded item ID as input. The input size equals the number of unique items in the dataset.
- The **hidden layer** keeps track of the 'state' or 'memory' of the network and plays a vital role in processing sequences of data. The hidden layer in this RNN model is implemented with a simple linear transformation followed by a non-linear activation function. The size of the hidden layer (i.e., the number of hidden units) is a hyperparameter to be chosen.
- The **output layer** produces the probabilities of the next item in the sequence. It is a linear layer.

In the forward pass of the RNN, the current input and the previous hidden state are combined to compute the current hidden state, which is then used to calculate the output probabilities. This allows the network to make use of the information from the sequence of item views by a user.

2. Long Short-Term Memory (LSTM) network
The LSTM is a type of RNN specifically designed to learn long-term dependencies. It is composed of an input layer, one or more LSTM layers, and an output layer.

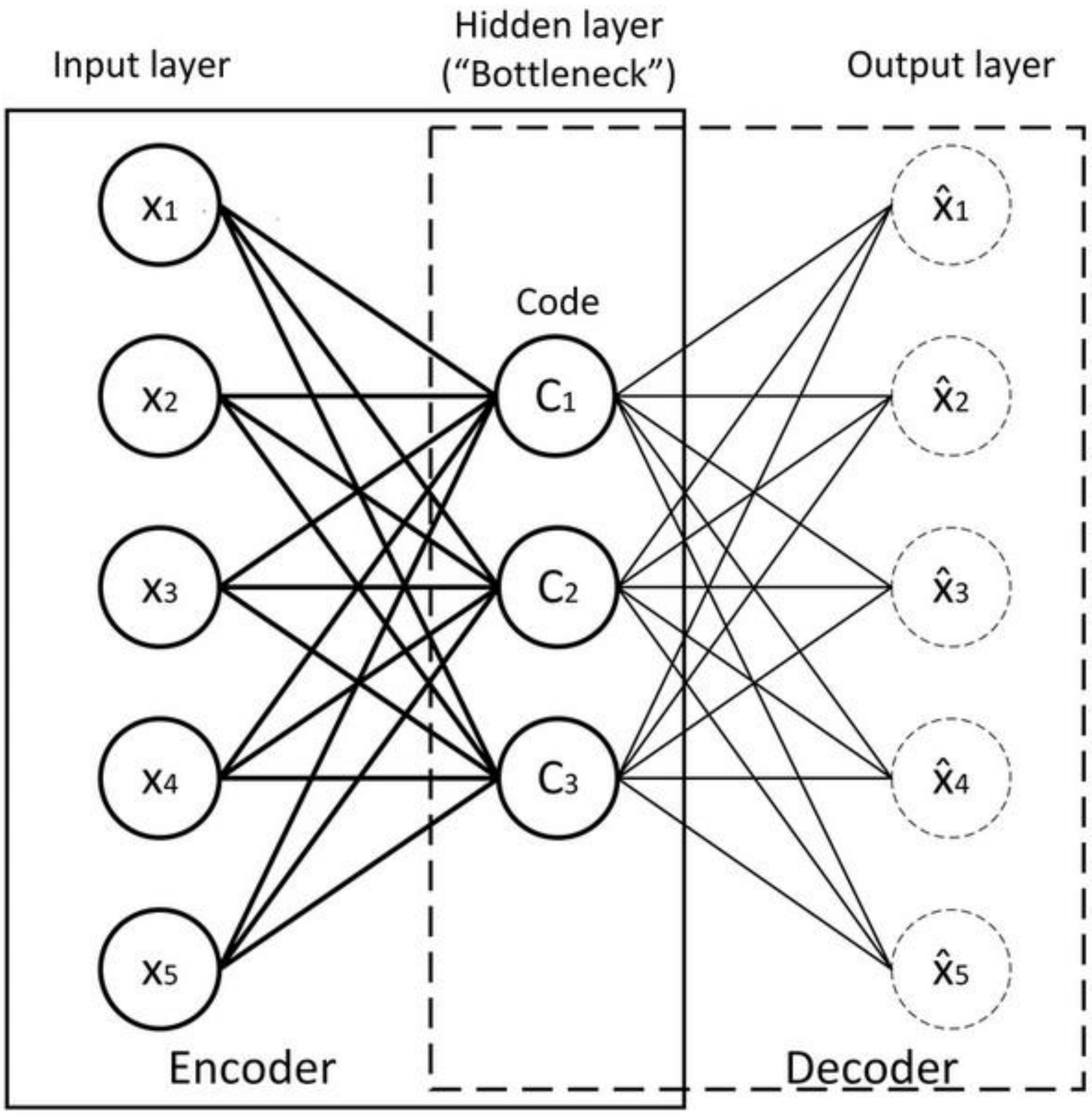


- The **input layer** is an embedding layer that maps each encoded item ID to a dense vector of a fixed size, also known as an 'embedding'. This helps in reducing the dimensionality and provides a more expressive representation of the items.
- The **LSTM layer** replaces the simple hidden layer of the Vanilla RNN. Each LSTM unit consists of a cell and three "gates" (input, forget, output), which collectively allow the LSTM to control the flow of information across timesteps.
- The **output layer** is the same as in the Vanilla RNN.

The LSTM model processes sequences of user-item interactions in a similar manner to the Vanilla RNN, but with a more complex update mechanism in the LSTM units, which helps it remember longer sequences and manage the vanishing gradient problem common in traditional RNNs.

3. Item2Vec

The Item2Vec model, based on the famous Word2Vec algorithm, learns dense vector representations for items in the same way that Word2Vec learns word embeddings. It treats each user session as a 'sentence' and items as 'words' within those sentences. It uses a shallow, two-layer neural network.



- The **input layer** is a one-hot encoded vector representing the 'current' item.
- The **hidden layer** serves as the 'embedding' layer. It projects the one-hot encoded input onto a dense embedding space.
- The **output layer** provides a probability distribution over all items, predicting the 'context' items given the current item.

This model is different from the others in that it's not explicitly sequence based. It learns the embeddings by predicting the context of an item in a user session.

Each of these models processes the sequences of user-item interactions differently, yet all can capture the sequential dependencies inherent in session data. The aim is to predict the next item a user will interact with based on their previous interactions, providing effective and personalized recommendations.

Training and Optimization

Training Process and Parameter Updates

The training process of the RNN-based recommender systems, including the Vanilla RNN, LSTM, and Item2Vec, were conducted using Stochastic Gradient Descent (SGD). The model parameters were updated iteratively through this process, with each iteration involving forward and backward passes through the network.

In the **forward pass**, the model received an input sequence of items viewed in a user session and generated predictions for the next item in the session. The predictions were probabilities distributed across all possible items, indicating the likelihood of each item being the next in the sequence.

Following this, the **loss function** measured the dissimilarity between the model's predictions and the true next item in the session. For this task, this project used Cross-Entropy Loss function, which is a standard choice for multi-class classification problems. It quantifies the difference between the predicted probability distribution and the actual distribution.

In the **backward pass**, gradients of the loss with respect to the model parameters were computed via backpropagation, which were then used to update the model parameters. Backpropagation through time (BPTT) was used in the context of RNNs to account for the temporal sequence of the data.

Computational Resources

The training process was conducted using PyTorch, a popular open-source machine learning library for Python, known for its flexibility and efficiency in handling dynamic computational graphs such as those in RNNs.

Performance Evaluation

The performance of the model was evaluated based on its predictive accuracy, which indicates how often the model correctly predicts the next item in a session.

However, accuracy alone may not always provide a comprehensive view of the model's performance, other metrics might also be considered such as precision, recall, or F1-score to consider both the false positives and false negatives.

In the context of session-based recommender systems, ranking-based metrics like Mean Reciprocal Rank (MRR) or Normalized Discounted Cumulative Gain (NDCG) can also be useful, as they evaluate the model based on the rank of the correct item in the list of recommendations.

During the evaluation phase of this project, the model's performance was tested using a separate test set that the model hadn't seen during the training process. This approach ensured an unbiased estimate of how well the model generalizes to unseen data.

LSTM Accuracy of the model on the test data was 88.2% where as vanilla RNN Accuracy of the model on the test data was 84.61%. It is clearly visible here that LSTM is more reliable and accurate then vanilla RNN.

As a result, the Recurrent Neural Networks implemented in this project, showed promising performance, demonstrating their capacity to capture the sequential dependencies in the user-item interactions and use this information to generate accurate next-item recommendations.

Further improvements to the model's performance might involve tuning hyperparameters, such as the learning rate or the hidden layer size or implementing more complex model architectures. Additionally, using more training data or applying data augmentation techniques could potentially improve the model's ability to generalize from the training data to unseen sessions.

Case Study/Application

According to Mordor Intelligence "The household robots market is valued at USD 7.99 billion. It is expected to reach USD 18.90 billion by the next five years, with a CAGR of 18.81% during the forecast period." It is a big market and expected to grow even more. Recommender systems can be a crucial part of this robots to increase the user experience quality. As the user begins to interact with the robot, the system can keep track of the sequence of commands or tasks requested in each interaction session.

For instance, a user begins a morning routine by asking the robot to make coffee, followed by reading the day's news updates, and then checking the weather forecast for the day. The system can interpret this series of commands as a user session and learns that these tasks typically follow each other in the given order.

The next morning, as soon as the user asks the robot to make coffee, the session-based recommender system can suggest the next action based on the previously observed session: "Would you like to hear today's news updates?" Once the news update is over, it can further suggest, "Do you want to know today's weather forecast?"

In addition, the recommender system can dynamically adapt to changes in user routines. If the user starts asking for a daily workout routine after the weather forecast, the system can quickly learn this new pattern and incorporate it into its suggestions, enhancing the robot's ability to predict the user's needs. This way, the session-based recommender system enables the robot to provide more personalized, making it a truly smart companion in the household.

For this project we would need to have the required hardware and the software components. The robot would need a mechanical bod, **Sensors, Actuators(**Depending on the tasks the robot is expected to perform, it might need different types of actuators to move its parts, pick up objects, etc.), **Embedded System(like Raspberry Pi or Arduino), Connectivity(like Wi-Fi or** Bluetooth), **Operating System, Session-Based Recommender System(**This would be the core software component driving the robot's personalized interactions.), **Natural Language Processing (NLP) and Data Security and Privacy**

Discussion

The implementation of our session-based recommender system, built on the powerful architecture of Recurrent Neural Networks (RNNs), demonstrates significant potential in providing personalized item recommendations based on sequential user-item interactions.

Our RNN model, including its variants (vanilla RNN, LSTM) and the item2vec approach, captures the inherent temporal dependencies in user sessions and effectively models them to make accurate predictions. The system's ability to suggest the next possible item a user might interact with was promising.

One of the primary strengths of this model lies in its capacity to handle large datasets with high cardinality in the item space, without the necessity of user or item profiles. Additionally, the flexibility of RNNs allows for continuous learning and adjustment to new patterns as more data becomes available, a valuable trait for evolving online platforms.

However, this model has some limitations. Training RNNs, especially on large datasets, can be time consuming. Furthermore, while the system works well with sequential data, it might face challenges in scenarios with sparse data or cold start problems, where little or no interaction history is available for a user or an item.

In comparison to traditional collaborative filtering or content-based recommender systems, the session-based recommender provides the advantage of being able to handle anonymous sessions and does not rely on user or item metadata. The usage of LSTM overcomes the limitations of vanilla RNNs, notably in learning long-term dependencies, thus improving the prediction accuracy.

For future work, exploring advanced architectures such as Gated Recurrent Units (GRUs), could enhance the model's performance.

In conclusion, the session based recommender system offers valuable information about user behavior and potentially improves the user experience. Although this approach can be improved, the results from this project shows us a promising direction for future research and development in the field of recommender systems.

Conclusion

The main objective was to provide accurate, personalized item recommendations based on sequential user-item interactions, focusing on the Retailrocket e-commerce dataset.

The findings confirmed the efficacy of RNNs in modeling sequential data and their superiority in capturing temporal dependencies, which are crucial for session-based recommendations. The models showed promising results, effectively suggesting the next possible items a user might interact with. This capability has significant implications for enhancing user experiences on e-commerce platforms, potentially increasing the sales.

However, the project also exposed some limitations and challenges as mentioned in Discussion part of the report. These findings point towards possible areas for future research and improvement, such as exploring advanced architectures, incorporating additional context, and optimizing the model for better computational efficiency.

In summary, this project demonstrated the potential of RNNs in session-based recommender systems, contributing to the ongoing evolution of personalized recommendation in e-commerce. The learnings from this project provide valuable insights for future research and development in this field.

References:

1. RetailRocket Dataset, (2015). RetailRocket recommender system dataset. Available at: https://www.kaggle.com/retailrocket/ecommerce-dataset.
2. PyTorch. https://pytorch.org/.
3. https://www.mordorintelligence.com/industry-reports/household-robots-market
4. https://www.researchgate.net/figure/Illustration-of-an-autoencoder-with-1-hidden-layer_fig2_340822637
5. https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e
6. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
7. https://wiki.pathmind.com/lstm
8. https://www.cs.princeton.edu/courses/archive/spring16/cos495/slides/DL_lecture9_RNN.pdf