Erasmus
University
Rotterdam

ERASMUS UNIVERSITY ROTTERDAM

MASTER THESIS

The Effectiveness of Transformer and Recurrent Neural Networks in Providing Session-based Recommendations

Name Student:
Rowan DOESBURG

Student ID number:
457229

Supervisor:
Luuk van MAASAKKERS

Second Assessor:
Clement BELLET

A thesis submitted in fulfillment of the requirements
for the degree of

MASTER IN DATA SCIENCE AND MARKETING ANALYTICS

Erasmus School of Economics

April 29, 2022

ERASMUS UNIVERSITY ROTTERDAM

Abstract

Erasmus School of Economics

Master in Data Science and Marketing Analytics

The Effectiveness of Transformer and Recurrent Neural Networks in Providing Session-based Recommendations

by Rowan DOESBURG

As scenarios in which companies only have access to session-based information are becoming more prevalent, the importance of adequate session-based recommendation systems is rising. The aim of this study is to examine how Transformer neural networks perform in providing session-based recommendations compared to recurrent neural networks (i.e, LSTM) and other popular recommendations algorithms, such as k-Nearest Neighbour approaches. By bench marking the effectiveness of Transformers in a business context, we contribute to the literature on session-based recommendation systems and the use of Transformers in business in general. In this study, we used transactional data containing over 3 million shopping sessions sourced from Instacart, an online grocery delivery service. We found that, similar to the developments in Natural Language Processing, both Transformers trained with an MLM objective (e.g., BERT) and an RTD objective (e.g., ELECTRA) are an improvement over recurrent neural networks (i.e., LSTM) in providing session-based recommendations. However, they still appear to experience difficulties in steadily outperforming simpler k-Nearest Neighbour algorithms. Also, the importance of positional encoding for the Transformer architecture is emphasized and proven to gain significance as the length of a session increases. Lastly, we have proven the added value of contextual information for the performance of Transformer session-based recommendation systems.

Contents

List of Figures

1 Introduction

In the last decade, there has been an immense increase in online business activities worldwide. Due to an increased usage, the Internet has developed into a global marketplace for various goods and services, so-called 'electronic markets' (Javadi et al., 2012). Even traditional retailers have begun to develop, add and offer online components of their business. As a result of the abundance of choice and relatively low search and switching costs in the online environment (Zhang et al., 2011), companies have experienced increased competition of global competitors. Hence, various studies (e.g., Krizanova et al., 2019; Schwarzl and Grabowska, 2015; Lissitsa and Kol, 2016) have emphasized the crucial importance of distinct digital marketing strategies for both online and traditional businesses.

A well-established and traditional element of a digital marketing strategy is the creation of online advertising campaigns to reach specific target audiences. Previous research (Schwarzl and Grabowska, 2015) confirms the beneficial nature of these activities by emphasizing the importance of timing and targeting for the effectiveness of these campaigns. In contrast to the traditional offline markets, customers in the online environment are presented with an immense, almost infinite number of alternatives and choice (Dimoka et al., 2012). Due to our bounded rationality, we are not able to trade-off all the benefits and disadvantages of all alternatives, making us prone to information overload. Therefore, a number of studies (e.g. Haubl and Trifts, 2000; Sahoo et al., 2012) have identified the importance of interactive tools (e.g., personalized recommendations) that assist customer decision making for both the quality and efficiency of their purchase decision and the corresponding satisfaction with it. In particular, personalized recommendation systems help customers by narrowing down more than thousands of products to a smaller selection of items that they are likely to prefer (Sahoo et al., 2012). These assumptions about the customer are often based on a customer's previous interactions with, and purchase behaviors on the particular website (Lops et al., 2010).

Due to the increased availability of online customer data, a growing number of companies (e.g., Amazon, Netflix and Spotify) have attempted to improve these product recommendations by applying increased personalization (Xiao and Benbasat, 2007). Similarly, the increased usage and availability of customer data has also resulted in the personalization of advertisements (e.g., on Youtube and Facebook) and even discounts (e.g., Albert Heijn). The importance and potential of personalization was emphasized by strategy consultancy giant Accenture in a research conducted in 2017 (Wollan et al., 2017). By applying personalization technologies, companies are able to target their audiences how, when and with what the customer would most likely prefer (Thirumalai and Sinha, 2013). Logically, increased personalization has become an even more essential element of the digital marketing strategy of companies (Arora et al., 2021). However, it can be challenging to identify these individual customer preferences as, with the current state-of-the-art recommendation systems, it requires a considerable amount of information about a customer (Ricci et al., 2011). As a result, when no or little previous information about a customer is available, the current recommendation systems can experience difficulties in providing high quality recommendations.

With the introduction of the General Data Protection Regulation (GDPR) in the European Union in 2018 and the increasing awareness on online privacy and data protection among customer (Boerman et al., 2021), the odds of experiencing this information shortage has increased. Hence, the necessity for recommendation systems that can handle small amount of session-based customer information for generation of so-called session-based recommendations has increased.

As the absence of a considerable amount of data increases the complexity of recommendation task (Geron, 2019), it might be interesting to consider more advanced techniques (e.g., deep learning) that were previously used in other 'regular' recommendation systems (Covington et al., 2016). One of these techniques are neural networks, a supervised machine learning technique that maps linear and non-linear relationships and patterns in data (Geron, 2019). A special type of neural network called recurrent neural networks (RNN) have the capacity to perform well in modeling complex sequence or time-varying data (Fausett, 1994; Medsker and Jain, 1999). Hence, these models are applied in a wide range of technologies and problems, such as speech and text recognition and generation (Li and Yang, 2017). Recently, developments in Natural Language Processing (NLP) has inspired researchers to experiment with these techniques in business contexts. For example, Gabel et al. (2019) mimicked the rationale behind word embeddings (i.e. numerical representations of the semantic meaning of words) in a marketing context by creating product embeddings (i.e. numerical representations of the products). Similar to human language, in which the meaning of words are derived from the context in the sentence, the embedding (meaning) of a word is derived from the relationship to the embeddings of other words in the sentence (Li and Yang, 2017). On a similar note, product embeddings derive meaning from their context (e.g., products bought together in the same session). In Natural Language Processing, the contextual information of word embeddings is used as input for next word prediction with the help of neural networks (Li and Yang, 2017). In theory, product embeddings could be utilized in a similar manner (Hidasi et al., 2015). In Natural Language Processing tasks including next word prediction, the most dominant and best performing type of neural network in the recent years was the Long-Term-Short-Memory Network (LSTM), a variation of the recurrent neural network architecture (Hochreiter and Schmidhuber, 1997; Geron, 2019). However, in 2017, Vaswani et al. introduced the Transformer neural network architecture, which outperformed the LSTM in most Natural Language Processing tasks. Considering its outstanding performance in Natural Language Processing, it would be interesting to investigate how the Transformer performs compared to the more commonly used LSTM (e.g., in Zhu et al., 2017; Shafqat and Byun, 2020; Weinzierl et al., 2020; Fuentes et al., 2021) in business-related contexts and, more specifically, in next item prediction for session-based recommendations.

Therefore, the aim of this study is to examine how Transformers perform in predicting purchase behavior based on session-based data compared to LSTMs and other popular recommendations algorithms. By bench-marking the effectiveness of Transformers in a business context, we contribute to the literature on session-based recommendation systems and the use of Transformers in business in general. Moreover, this paper contributes to the literature by investigating the effect of various training objectives and the addition of contextual information on the performance of the Transformer in session-based recommendations. Furthermore, considering that, in online marketing, a small increase in recommendation quality could yield a significant growth in sales (Arora et al., 2021), the importance of high-quality recommendations in today's online market has increased. Hence, from a marketing perspective, it is highly relevant to compare these techniques to investigate the potential of Transformers as session-based recommendation systems and identify the most effective technique for companies to use.

This paper is outlined as follows. First, related work in (session-based) recommendations and neural networks is discussed. Second, the processing of the empirical data used in this study is described. Third, the research methodology and technical details are elaborated on. At last, the results, findings, academical and managerial implications, and conclusions of this study are discussed and reviewed.

2 Related Work

In this chapter, past related literature on recommendation systems and neural networks is reviewed to identify the previous and current developments surrounding these subjects.

2.1 Recommendation Systems

In an online environment, it is crucial to learn and predict customer preferences or intentions to provide accurate recommendations. A vast body of research has studied the use of real-time predictive analytics in marketing in order to predict customer behavior (Shmueli and Koppius, 2011; Cui and Curry, 2005; Dhar et al., 2014). From a marketing perspective, promotion activities, such as discount and advertisements, based on these recommendations could convince the customer to engage in the purchase of a particular item. In this section, we discuss the two most prevalent techniques currently used to generate recommendations: content-based filtering and collaborative filtering.

2.1.1 Content-based Filtering

Content-based recommendation systems solely rely on previous purchases of, or ratings provided by the targeted customer (Lops et al., 2010; Ricci et al., 2011; Adomavicius and Tuzhilin, 2005b). In essence, it attempts to provide useful recommendations based on their previous purchases or interactions with the company. For example, a customer who regularly buys organic tomatoes will be given recommendations for other similar organic products and someone who provides a high rating for an action movie will be given recommendations for other similar action movies. In other words, it relies on the similarity between features of the items and maps that as the preference of the user (Lops et al., 2010; Ricci et al., 2011; Adomavicius and Tuzhilin, 2005b). There are, however, some limitations to this method. First, the currently used content-based recommendation systems need sufficient data to learn a customer preference (Ricci et al., 2011). This data is not always steadily available. For example, for new customers or guest customers, there might not be sufficient data to provide useful recommendations. Moreover, in order to select the most effective features of items, an extensive in-field domain knowledge is required (Lops et al., 2010; Adomavicius and Tuzhilin, 2005a). This makes the quality of the recommendations dependent of the knowledge of the person selecting the features. In some contexts, it is difficult to assure high quality feature selection for all products. For example, if a company, such as Amazon, bol.com or Ebay, allows third-party or private sellers to sell through their platform, there is often little control over the quality and usefulness of the features added to the product (description). Moreover, it would be very costly to hire domain experts to manually process each item's feature selection. Critics of this technique also argue that the over-reliance on content-based filtering might lead to customers being stuck in a 'similarity hole' in which they only get similar products recommended (Hurley and Zhang, 2011). For example, when a customer is searching for a horror movie on Amazon, the algorithm would most likely recommend other horror movies. The 'similarity hole' might weaken the potential and odds of recommendations that were more suitable or compatible to the customer. This is in line with the study of Park and Han (2013) that argued that, besides similar product recommendations, there is also a great importance of diverse recommendations.

2.1.2 Collaborative Filtering

The second widely used recommendation generating method that will be discussed is collaborative filtering (Adomavicius and Tuzhilin, 2005a; Ricci et al., 2011). In contrast to content-based recommendation systems, the personalized recommendations derived from collaborative filtering are often based on previous purchase behavior of customers that are similar to the targeted customer, or items similar to the items in question. These systems rely on the assumption that customers who bought the same products share customer preferences to some extent. Fundamentally, they mimic customer-to-customer recommendations by recommending items that similar customers have bought (Ricci et al., 2011). In movie recommendations, an example of collaborative filtering would be 'other users who also liked movie A, liked movie B. Therefore, we recommend movie B.' An advantage of collaborative filtering over content based filtering is that it is able to provide recommendations without having access to information on product features (Ricci et al., 2011). Also, there is lower risk of the similarity hole problem as it recommends items based on others' preferences. Nevertheless, this is also one of the limitations of collaborative filtering as this method requires an extensive set of rating information of other customers (Adomavicius and Tuzhilin, 2005b). If customers do not regularly rate their products, which is not uncommon in some industries, the system will have insufficient data to provide adequate recommendations. In line with this, one could also argue that the assumption that customers share customer preferences might not always be valid. If a customer has a specific taste that is not similar to other customers, the recommendations provided through collaborative filtering have a high chance of being of low quality (Delic et al., 2016; Tran et al., 2018).

2.1.3 Limitations of Current Recommendation Systems

Although the aforementioned methods have different advantages over the other, there are some common limitations in certain contexts (Jannach et al., 2020). First, both methods rely on previous information on either the customer's purchases or ratings. As a result, the so-called 'cold start' problem arises, in which recommendation systems have difficulties in providing new customers with qualitative recommendations as there is little to no information known about the customer (Lops et al., 2010). Second, these methods work best if one can assume that customer preferences do not change frequently and are context insensitive. However, it may not always be reliable to assume that predictions based on past purchase behavior or preferences will be an accurate forecast for future behavior as research shows that customer preferences can change over time (Koren, 2010). Consequently, most recommendation models that were based on specific prior customer preferences could have lower predictive power if customer preferences change after the training period (Sahoo et al., 2012). At last, with the introduction of the General Data Protection Regulation (GDPR) in the European Union in 2018, consumers have become increasingly aware of online privacy, data protection and online tracking (Boerman et al., 2021). Consequently, an increasing number of consumers prefer to browse anonymously or reject cookies used for tracking, which increases the likelihood of not having access to (sufficient) information about the customer in an online environment (Strycharz et al., 2021). Hence, in order to recommend the 'right product to the right customer at the right time', it would be valuable for companies and marketeers to have predictions of future customer purchase behavior that can handle frequently changing customer preferences and accurately map the current customer intentions.

2.2 Session-based Recommendations

A way to prevent the aforementioned limitations of the discussed recommendation systems is by solely using information that is provided by the customer in the current session (Ludewig and Jannach, 2018). In this sense, a session can be defined as the set of uninterrupted interactions of a user with a particular platform or company (e.g., adding product to the online shopping cart, or a sequence of clicks on the website of an online retailer). This provides several benefits. For example, by using only information about the current session, it is possible to make session-based recommendations within the rules of GDPR that reflect a customer's current intentions or preferences and that are less sensitive to the cold-start problem (Ludewig and Jannach, 2018).

2.2.1 Filtering Methods for Session-based Recommendations

It is interesting to discuss if and how adjusted versions of the aforementioned recommendation systems could be used for session-based recommendations. As mentioned before, content-based recommendation systems use product features to map customer preferences. However, research shows that these systems often need an extensive amount of data to provide useful recommendations and that the feature selection requires an in-depth knowledge of the domain (Ricci et al., 2011). Fundamentally, session-based information often consists of a smaller sequence of information, which could pose difficulties for content-based filtering. Moreover, the assurance of high quality feature selection is not always easily available, which decreases the effectiveness and robustness of this method across different contexts. Hence, these content-based recommendation systems are deemed to be inadequate for session-based recommendations. Collaborative filtering uses similarity in ratings or purchases across users to provide recommendations (Adomavicius and Tuzhilin, 2005a). Therefore, it does not need knowledge about the product itself to provide recommendations, making it robust within different contexts (Ricci et al., 2011). In the context of session-based recommendations, the aforementioned limitations of this method do not necessarily have to be problematic. Although there is no availability to the previous purchase behaviour of a particular customer, there is often anonymous purchase information available of previous customers. This, however, cannot be mapped to anonymous users, which results in treating every anonymous visiting user as a new customer. In essence, this method could use the currently selected products or interactions in a session as the variables to calculate similarity to other purchase sessions. Subsequently, the system could recommend items from the most similar sessions. In previous research, similar filtering approaches yielded promising results in session-based recommendations (Hidasi et al., 2016; Jannach and Ludewig, 2017; Kamehkhosh et al., 2017; Hidasi and Karatzoglou, 2018). Hence, an adjusted version of collaborative filtering is deemed to be adequate for session-based recommendations.

2.2.2 Advanced Methods for Session-based Recommendations

While considering other suitable techniques for session-based recommendations, it is advisable to identify what is asked by the task and the information that is available. As mentioned before, it is crucial to learn and predict customer preferences or intentions to provide accurate recommendations. In other words, in session-based recommendations, we would like to predict a customer next 'most desired product' based on the interaction of a user in the current session. Considering that we also have anonymous previous purchase sessions, we could find patterns between products within the previous purchases to provide a valid recommendation.

In the recent years, several studies in (session-based) recommendations have experimented with machine learning and Natural Language Processing (NLP) techniques, ranging from k-Nearest Neighbors to (Recurrent) Neural Network architectures (Hidasi et al., 2015; Hidasi and Karatzoglou, 2018; Hidasi et al., 2016). For example, in Natural Language Processing, words are often represented as word embeddings. Word embeddings are essentially numerical representations of the semantic meaning of words in the form of vectors. Similar to human language in which the meaning of words are derived from the context in the sentence, the embedding (meaning) of a word is derived from the relationship to other word embeddings in the sentence (Geron, 2019). Since the embeddings are numerical representations of words, it is possible to use them in calculations. A widely used example is that the word embedding of 'King' subtracted by the embedding of 'Man' and added to the embedding of 'Woman' should result in the embedding of 'Queen' (Ethayarajh et al., 2019). In NLP literature, these word embeddings are used as input for neural networks in next word prediction and in text or speech generation tasks (Li and Yang, 2017). The combined contextual information of word embeddings serve as a basis for next word prediction. Inspired by the use of word embeddings in NLP, Gabel et al. (2019) introduced the P2V-Map model, which demonstrated its capability of accurately mapping similarities and co-occurrences between products based on solely transactional basket data. A single observation in transactional basket data is the set of products bought by a customer in a single session and is thus comparable with the information available in session-based recommendations. In the process, P2V-Map generates and trains product embeddings, which derive meaning from their context (a transactional basket) and are therefore similar to the word embeddings seen in Natural Language Processing (Gabel et al., 2019). These product embeddings can be used to calculate similarity and co-occurrence between products. Subsequently, one could, in theory, estimate which products might fit in the existing shopping basket of a customer by using these co-occurrences (Gabel et al., 2019). This is similar to how word embeddings are used in NLP in next word prediction tasks, in which the embeddings are fed into a (deep learning) neural network (Li and Yang, 2017). Logically, the use of neural networks has gained popularity in session-based recommendation literature, showing promising results (Hidasi et al., 2015, 2016; Hidasi and Karatzoglou, 2018).

2.3 Neural Networks

As the name suggests, (artificial) neural networks are loosely based on the connections between neurons in our brains. Similar to how neurons are connected through axons and synapses, a neural network consists of connected nodes that pass on information. In machine learning, neural networks are often used as a supervised machine learning technique to map linear and non-linear relationships and patterns between two (sets) of variables (Geron, 2019). A minimal, basic neural network consists of an input layer and an output layer. Frequently, neural networks also contain one or more hidden layer(s), which are located in between the input and output layers. Each of the layers contain nodes that, if possible, receive information (input) from nodes in the previous layer and pass on information (output) to nodes in the next layer. Within each node, the input values are multiplied by weight coefficients to calculate a weighted sum, which is subsequently added to the bias and then used as input for an activation function to generate an output value. The bias is a constant, which is moderately comparable to the constant in linear function and can be used as an extra adaptable parameter to improve model performance (Geron, 2019). The use of activation functions is inspired by the functioning of neurons in our brains, in which these 'functions' decide if a neuron will be 'fired' to the next neuron or not. Similarly, in artificial neural networks, an activation function is a mathematical function that decides whether and in what form the information within a node should be passed onto nodes in the next layer (Geron, 2019).

A neural network learns by applying a technique called back-propagation (Rumelhart et al., 1985; Rumelhart et al., 1995). Initially, a preliminary neural network is generated by randomly initializing the parameters (i.e., the weight coefficients and the biases). In back-propagation, the optimization of these parameters starts at the computed output value (at the end of the network) by calculating the prediction error term of the current network settings (Rumelhart et al., 1995). Subsequently, the network 'back propagates' the error back through the network to determine how to adjust the weights and biases to minimize the (aggregated) error term. In short, it calculates to what extent the output of each node in the previous layer (before the output layer) contributed to the error term. Then, the algorithm computes to what extent these error contributions were caused by each node in the layer before. The algorithm repeats this process until it reaches the input layer. In this way, the algorithm is able to find the gradient of the error term in relation to each parameter. Using these gradients, the network adjusts the weight coefficients and the biases to minimize the error term through a gradient descent step (Rumelhart et al., 1995; Geron, 2019). The model iterates through this process until convergence is reached (i.e., loss moving towards a local or global minimum). Eventually, this process results in a lower aggregated error term, which improves the general predictive accuracy of the neural network (West et al., 1997; Geron, 2019).

2.3.1 Recurrent Neural Networks

The simplest form of neural network is the feedforward neural network, which takes an input layer and calculates an output layer. In these neural networks, the data only moves in one direction (forward), which makes it less suitable for performing tasks on sequential data (e.g., text, audio and images) as this requires knowledge about previous states of the data (Geron, 2019). Opposed to feed-forward neural networks, recurrent neural networks (RNN) have the capacity to perform well in modeling sequence or time-varying data. Hence, these types of neural networks are applied in a wide range of technologies and problems, such as speech and text recognition, financial (stock) prediction and natural water inflows forecasting. In essence, RNNs are feed-forward neural networks with an implemented feedback loop, in which the input of the network at time step *t* consist of the input values from the input layers as well as the output values of the network at timestep *t-1* (Fausett, 1994). In this way, an RNN can use information from a previous timestep as input for the current timestep, enabling the processing of sequential data.

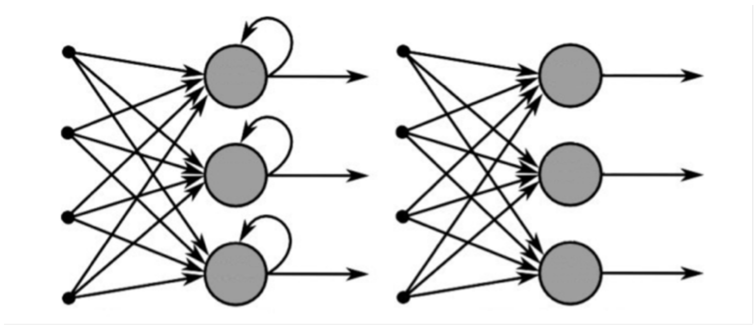


FIGURE 2.1: Recurrent Neural Network (Left) and Feed Forward Neural Network (Right) (Eliasy and Przychodzen, 2020)

Nevertheless, recurrent neural networks can also experience some difficulties when dealing with longer sequences (Hochreiter and Schmidhuber, 1997; Bengio et al., 1994). As mentioned before, neural networks learn by using back-propagation. The error-minimizing alteration of the parameters is based on (steepest) gradient descent optimization. This algorithm calculates a local minimum by using the derivatives of the loss function. The network computes the gradients of a parameter based on the contribution to the error term of the nodes in the layer above. In calculating the gradients, it uses the chain rule (Geron, 2019). In this process, two problems frequently arise. The first one is called the vanishing gradient problem. As successive gradients are multiplied through the chain rule, gradients tend to decrease in size as the back-propagation algorithm reaches earlier layers. As a result, the corrective adjustment done through the gradient descent step will almost be zero in these early layers, which prevents these layers from 'learning' (Bengio et al., 1994). Second, the gradients also frequently tend to increase immensely in size as the back-propagation algorithm reaches earlier layers. As a result, the corrective adjustment in these layers will be too large, which makes the network unstable. This is also known as the exploding gradient problem (Geron, 2019). As a recurrent neural network processes the sequence by using input values and the network's output of the previous time step as the input for the current time step, the aforementioned problems can arise in these networks (Bengio et al., 1994). The error term is often calculated at the end of a sequence and is used as the basis for the back-propagation process to optimize the network's parameters (Rumelhart et al., 1995). When back-propagating, the back-propagation algorithm 'unrolls' the RNN, making it essentially a very deep feed-forward network (Figure 2.2). As a result, the network is more susceptible of the exploding or vanishing gradient problem as the larger number of layers the network should back-propagate through increases the risk of vastly increasing or decreasing gradients (Bengio et al., 1994). Hence, these gradient descent learning algorithms tend to portray weak performance on problems that require long-term dependencies on inputs derived or computed in the past (Medsker and Jain, 1999).

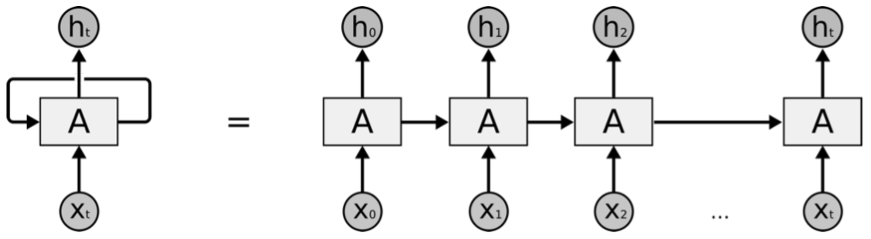


FIGURE 2.2: Unrolled Recurrent Neural Network (Olah, 2015)

To solve this problem, Long Short-Term Memory based neural networks (LSTM) were invented (Hochreiter and Schmidhuber, 1997). These are recurrent neural networks with additional operators, also referred to as 'gates', that determine which information from the previous time state should be transferred over to the current one. By implementing these gates in the architecture, the chain rule is circumvented, resulting in the avoidance of the vanishing gradient problem (Hochreiter and Schmidhuber, 1997). Consequently, these type of networks are the most commonly used neural networks in which long-term sequences are important (e.g., speech, text and image recognition or next word prediction) (Geron, 2019). Similar to 'basic' recurrent neural networks, every output in the LSTM depends on the output of the previous time step. The input data can therefore only be 'fed' to these models in a sequential manner, which results in a relatively long training time for LSTMs.

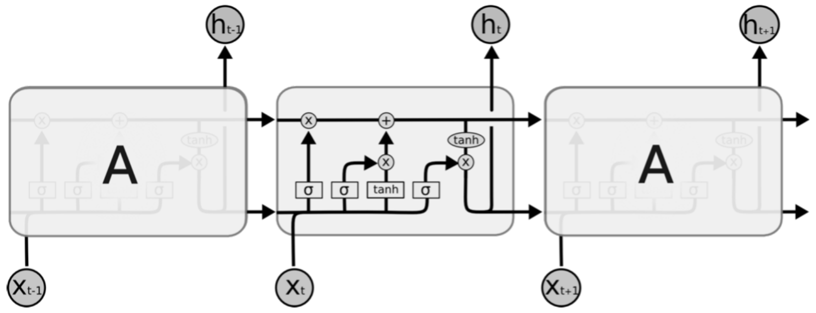


FIGURE 2.3: Long-Term-Short-Memory Neural Network (Olah, 2015)

In the context of session-based recommendations, the information in a session (e.g., interactions happening in a particular session) could be considered as a sequence of actions. In that sense and taking into account the good performance of the LSTM in next word prediction tasks, some researchers have experimented with the LSTM (or recurrent neural networks in general) for next-item prediction and session-based recommendations (e.g., Zhu et al., 2017; Shafqat and Byun, 2020; Weinzierl et al., 2020; Fuentes et al., 2021; Hidasi et al., 2015, 2016; Hidasi and Karatzoglou, 2018)

2.3.2 Transformer Neural Networks

In 2017, Vaswani et al. introduced their transformer neural network architecture, which has gained a lot of popularity since its introduction. The Transformer is able to process data in a non-sequential manner (e.g., in parallel instead of word by word). As a result, the Transformer allows for a high training speed and lower training time. Moreover, the most differentiating element of this architecture is the attention mechanism in the multi-headed self-attention sub-layers (see Figure 2.4). In short, self-attention allows the model to consider other relevant items in the input to improve the encoding or 'understanding' of the current item (Geron, 2019). For example, in the sentence "Chris smiled at Anna because he was happy", self-attention allows the model to learn that 'he' is referring to 'Chris' and not to 'Anna', giving 'Chris' higher attention when learning the representation of 'he'. Multi-headed attention allows the model to have 'multiple attentions', meaning that the model can base each attention on a different set of characteristics of the word of interest (Vaswani et al., 2017).

In essence, due to the added attention mechanism and non-sequential data processing, transformer networks are better capable to 'memorize' earlier information compared to LSTMs as they cannot 'forget' information. Moreover, the attention mechanism allows the Transformer to 'pay attention' to particular relevant prior observations despite their potential early position in the sequence (Vaswani et al., 2017). Theoretically, this should allow the Transformer to adequately process long-term sequences in a more time-efficient manner.

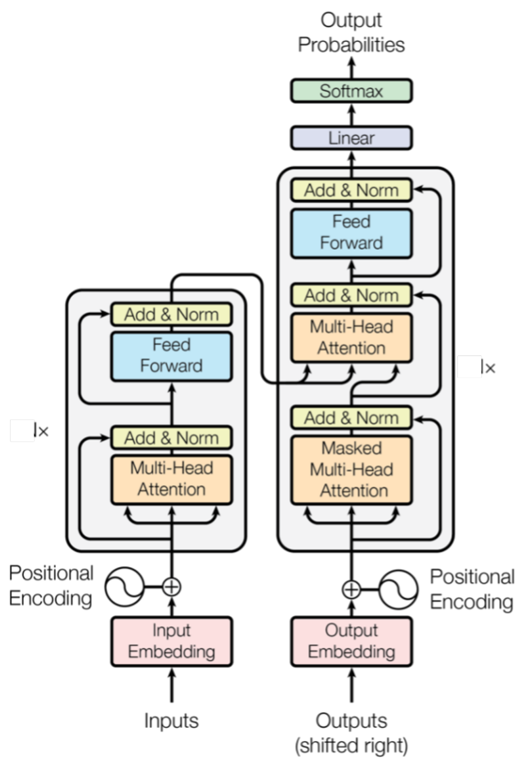


FIGURE 2.4: Overview of the Transformer Neural Network Architecture (Vaswani et al., 2017).

Since its introduction in 2017, many variations and improvements on the original Transformer (as proposed by Vaswani et al. (2017)) have appeared (e.g., Devlin et al., 2018; Dai et al., 2019; Yang et al., 2019; Clark et al., 2020). For example, in 2018, Google developed BERT (Bidirectional Encoder Representations from Transformers), an encoder-only language model that uses a slightly modified encoder architecture of the original transformer as its base (Devlin et al., 2018). BERT is trained through a training objective called masked language modeling (MLM), which will be discussed in Chapter 4.3.2. Several variants of BERT have surfaced since then (e.g., RoBERTa and ALBERT). In 2020, Clark et al. developed ELECTRA, a Transformer architecture that uses Replacement Token Detection (RTD) as training objective, which will also be discussed in Chapter 4.3.2. Considering the outstanding performance of these transformer neural networks architectures in Natural Language Processing, it is interesting to investigate how these architectures perform in a business or marketing related context (i.e., session-based recommendations) in comparison with the state-of-the-art LSTM neural network and the adjusted collaborative filtering recommendation system. In particular, in this study, we investigate how Transformer architectures inspired by BERT (as in Sun et al., 2019) and ELECTRA perform in providing session-based recommendations.

3 Data

In this chapter, the processing of the data is elaborated on.

3.1 Data Source

The data that was used in this study is provided by Instacart, an online grocery delivery service in the United States (Instacart, 2017), and is freely available on Kaggle.com. The original data set contains over 34 million in-cart shopping item observations of approximately 3.3 million shopping baskets of 206,209 unique users. The original in-cart shopping items consist of around 49,000 distinctive products of 134 product categories.

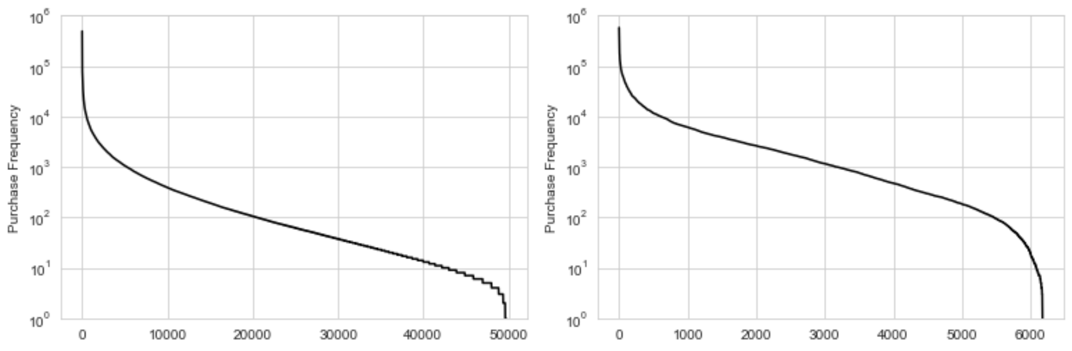
3.2 Data Processing

Considering the fact that machine learning algorithms require a sufficient number of occurrences of each class to adequately learn (Geron, 2019), we decided to aggregate (low-frequency) products in subsets of products to ensure each product class contains enough observations. In particular, we manipulated the data as follows. First, we checked the number of distinctive products each category contained. For each category containing over 50 distinctive products, we selected 25 distinctive products with the highest purchase frequency per category. All other products were aggregated into sub-categories within their main category based on their textual product name. For example, in the category 'Cereal', all remaining cereal products containing (derivatives of) the word 'oat' in their product name were aggregated into the sub-category 'Other Cereal: Oats'. If the product name contained multiple categories (e.g., 'rice' and 'oat'), the product was categorized in the most popular of these subcategories. Subsequently, these sub-categories were treated as distinctive products in the data. This resulted in 6175 distinctive 'products'. In the original data, there was no information about how frequently the same product occurred in a single session. Therefore, if after aggregation an order contained duplicate values of a sub-category, these duplicate values were removed. In Figure 3.1, the log of the purchase frequency of each product is plotted. We notice that, before and after processing, the purchase frequencies are highly imbalanced across products, meaning that there are high differences between products in terms of purchase frequency. However, after processing, the data seems to be slightly more balanced. Second, a minimum and maximum session length filtering process was applied to the data. In particular, all sessions containing only one purchased item were removed as a minimum sessions of two is needed to create input and output for the model. Moreover, as approximately 90 percent of the sessions contained less than 25 items, all sessions with a session length of more than 25 items were also removed from the data to limit the training time while maintaining a sufficient share of information in the data. After applying the filter, the final data set consisted of 3,053,549 sessions.

	Sessions	Mean Length	Min-Max Length	Items
Before	3,346,083	10.107	1-145	49,685
After	3,053,549	9.365	2-25	6,175

TABLE 3.1: Data Statistics Before and After Processing

At last, we created extra variables that might enrich the information in the data. Therefore, we constructed several binary variables based on the products: ’vegan’, ’lactose-free’, ’organic’ and ’gluten-free’. These variables were created as they reflect dietary preferences of customers, which could contribute to the prediction of their next purchase. In order to train the models, we performed a 95-2.5-2.5 random split of the data into a training, test and validation set, respectively.



(A) Before Processing (B) After Processing

FIGURE 3.1: Distribution of Purchase Frequencies (Log scale)

4 Research Methodology

In this chapter, the research methodology is discussed. First, the problem definition, the technical details of the LSTM and the technical details of the Transformer are elaborated on. Next, the training procedures, the performance metrics, the configurations of the models, and the benchmark algorithms are discussed.

4.1 Problem Definition

As mentioned before, the aim of this study is to assess how various neural network architectures provide session-based recommendations. In particular, we evaluate and compare the performance of the Transformer and LSTM architecture in predicting the next product in a shopping cart based on the contextual information in a single session. Considering this problem, each session is defined as the collection of N_i products purchased by a particular anonymous consumer in a single uninterrupted shopping session. The products in each session are represented by their corresponding product IDs and the order of the products in a session are based on the original add-to-cart order. Hence, a session is formally denoted as

$$\mathbf{S}_i = \{p_{j=1}, p_{j=2}, p_{j=3}, ..., p_{j=N_i}\}$$

where p_j is the product ID of the product on the j-th posistion in session i of length N_i.

Hence, for instance, session i that consists of five items with product IDs 751, 1523, 2675, 169 and 3616 would be represented as $\mathbf{S}_i = \{751, 1523, 2675, 169, 3616\}$.

4.2 Neural Networks

4.2.1 Long-Term-Short-Memory Networks

As mentioned before, the LSTMs were invented to solve the short-term memory problem of standard recurrent neural networks in sequence-to-sequence tasks (Hochreiter and Schmidhuber, 1997). In essence, the LSTM network consists of an input layer, a recurrent LSTM layer and an output layer. Within the recurrent LSTM layer, there are two states and multiple gates. The states entail a cell $(\mathbf{c}_t)$ and hidden state $(\mathbf{h}_t)$, which could be described as 'short-term' and 'long-term', respectively (Geron, 2019). The multiple gates entail an input $(\mathbf{i}_t)$, forget $(\mathbf{f}_t)$, cell $(\mathbf{g}_t)$ and output $(\mathbf{o}_t)$ gates. The rationale behind the different gates and states and the corresponding calculations are discussed below.

In the LSTM architecture, the problem is represented as a sequence-to-sequence task, in which an input sequence $(\mathbf{X})$ is mapped to a target sequence $(\mathbf{Y})$. In this study, the target variable is the next product added to the shopping cart after a certain sequence of products in a single session. Logically, this is represented by the last product added in the session. Therefore, the input sequence of a shopping session consist of the products in that shopping session without the last product. In contrast, the target sequence of a shopping session consist of the products in that session without the first product. Considering the aforementioned session i that consisted of product IDs 751, 1523, 2675, 169 and 3616, the input sequence would be denoted as $\mathbf{X} = \{751, 1523, 2675, 169\}$ and the target sequence as $\mathbf{Y} = \{1523, 2675, 169, 3616\}$.

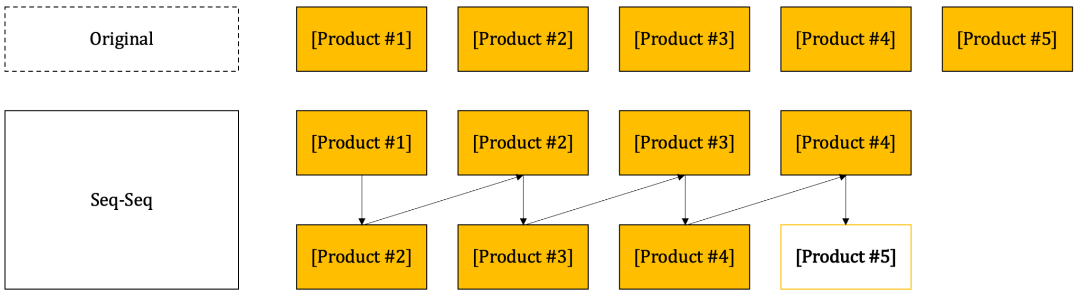


FIGURE 4.1: Training Process of Long-Term-Short-Memory Networks

As seen in Figure 4.2, we have input $\mathbf{x}_t$, which is the input value at time step t. For the first time step ($t = 0$) in our example, this would be the vector (embedding) representing the product with product ID 751. Along with the previous hidden state ($\mathbf{h}_t$), the input vector ($\mathbf{x}_t$) is sent to four separate layers, which compute the input ($\mathbf{i}_t$), forget ($\mathbf{f}_t$), cell ($\mathbf{g}_t$) and output ($\mathbf{o}_t$) gates, respectively (Geron, 2019).

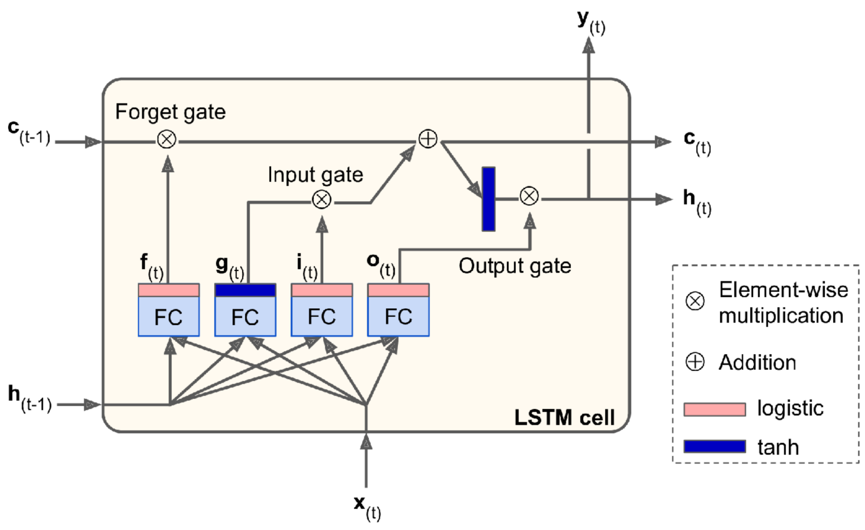


FIGURE 4.2: Long-Term-Short-Memory Cell (Geron, 2019)

At the left-hand side of Figure 4.2, we notice information flowing in from the previous cell state ($\mathbf{c}_{t-1}$). The cell state ($\mathbf{c}_t$) moves the information from previous state ($t-1$) to the current one (t), enabling the model to transfer knowledge from previous states (i.e., long-term knowledge). Logically, at time step 0, the information from the previous cell state ($\mathbf{c}_{t-1}$) will be zero. The cell state is formally notated as

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$

where $\odot$ is the element-wise product, and $\mathbf{f}_t$, $\mathbf{i}_t$ and $\mathbf{g}_t$ are the outputs of the forget gate, input gate and cell state at time step t, respectively.

With the help of the forget gate ($\mathbf{f}_t$), the cell state is able to memorize or forget information (Hochreiter and Schmidhuber, 1997). For example, assume the first product (ID: 751) was an organic vegan chicken substitute. From this first step of the sequence, the model would 'learn' that this customer only buys vegan and organic products, giving non-organic animal products a low probability. However, the second product (ID: 1523) is organic eggs. Based on the current input of the second product, the model should adjust its assumptions on the customer preferences to 'not only vegan, but still organic'. At first, the model should 'forget' the 'vegan only' preferences of the customer. In practice, this 'forgetting' of information is done by element-wise multiplication of the forget gate ($\mathbf{f}_t$) and the previous cell state ($\mathbf{c}_{t-1}$). The forget gate ($\mathbf{f}_t$) is essentially calculated through a neural net layer with a logistic function (σ), which takes the weights of the current input ($\mathbf{x}_t$) and the previous hidden state ($\mathbf{h}_{t-1}$) as input. As output, this logistic layer computes numbers between 0 and 1 for all the numbers in the previous cell state ($\mathbf{c}_{t-1}$), where 0 indicates information that will be totally forgotten and 1 indicates information that will be totally remembered (Geron, 2019). The forget gate is formally notated as

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$$

where $\mathbf{b}_f$ is the bias of the forget gate, and $\mathbf{W}_{xf}$ and $\mathbf{W}_{hf}$ are the matrices of the weight coefficients associated with the input and previous hidden state at time step t, respectively.

Once a part of the information is 'forgotten' by the cell state, we may also need to 'add' new information to it. In our example, the purchase of organic eggs could inform the model that the customer also has a preference for 'organic', 'protein rich' and 'vegetarian' products. As seen in the formula of the cell state, the addition of the information to the cell state happens by adding the values of element-wise multiplication of the cell gate and input gate to the element-wise multiplication of the forget gate and previous cell state (Geron, 2019). The cell gate is a vector containing new potential values that might be incorporated into the current cell state (Geron, 2019). In other words, this gate holds the new information that could be added to the model. It is calculated through a neural net layer with a tanh function, which takes the information of the input vector and previous hidden state as input and is formally notated as

$$\mathbf{g}_t = tanh(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g)$$

where $\mathbf{b}_g$ is the bias of the cell gate, and $\mathbf{W}_{xg}$ and $\mathbf{W}_{hg}$ are the matrices of the weight coefficients associated with the input and previous hidden state at time step t, respectively.

The decision on which part of the new information of the cell gate will be stored in the cell state is controlled by the input gate (Geron, 2019). Similar to the forget gate, the input gate ($\mathbf{f}_t$) is calculated through a neural net layer with a logistic function (σ), which takes the weights of the current input ($\mathbf{x}_t$) and the previous hidden state ($\mathbf{h}_{t-1}$) as input and is formally notated as

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\,\mathbf{x}_t + \mathbf{W}_{hi}\,\mathbf{h}_{t-1} + \mathbf{b}_i)$$

where $\mathbf{b}_h$ is the bias of the input gate, and $\mathbf{W}_{xi}$ and $\mathbf{W}_{hi}$ are the matrices of the weight coefficients associated with the input and previous hidden state at time step t, respectively.

Next, the output gate $(\mathbf{o}_t)$ controls what part of information of cell state will be used as both the hidden state $(\mathbf{h}_t)$ and output $(\mathbf{y}_t)$ of the cell at time step t (Geron, 2019). It is calculated through a neural net layer with a logistic function (σ), which takes the weights of the current input $(\mathbf{x}_t)$ and the previous hidden state $(\mathbf{h}_{t-1})$ as input and is formally notated as

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\,\mathbf{x}_t \; + \mathbf{W}_{ho}\,\mathbf{h}_{t-1} \; + \mathbf{b}_o\,)$$

where $\mathbf{b}_o$ is the bias of the input gate, and $\mathbf{W}_{xo}$ and $\mathbf{W}_{ho}$ are the matrices of the weight coefficients associated with the input and previous hidden state at time step t, respectively.

At last, the new hidden state $(\mathbf{h}_t)$ and output $(\mathbf{y}_t)$ are calculated through the element-wise multiplication of the output gate $(\mathbf{o}_t)$ and the tanh function of the cell state $(\mathbf{c}_t)$ (Geron, 2019), which is formally notated as

$$\mathbf{y}_t = \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

where $\odot$ is the element-wise product.

4.2.2 Transformer Networks

In the paper 'Attention Is All You Need' (Vaswani et al., 2017), the original Transformer architecture consists of an Encoder and Decoder block. However, depending on the task, the employment of solely the Encoder block of the Transformer could be sufficient (Devlin et al., 2018). Considering next purchase prediction as a generative task, we argue that the implementation of the Decoder is not necessary, and an adjusted version of the Encoder is adequate (see Figure 4.3) (Devlin et al., 2018). Therefore, in this section, we only focus on the methodology of the Encoder.

The encoder itself consists of l identical layers stacked on top of each other. As seen in Figure 4.3, the Encoder consists of two main elements: a Multi-Head Attention and a Feed Forward Layer (Vaswani et al., 2017). Consider the aforementioned example of session i consisting of product IDs 751, 1523, 2675, 169 and 3616. In contrast to the LSTM, the Transformer is capable to process the entire sequence at once. Therefore, the input $(\mathbf{S}_i)$ consists of all the product IDs in the session, resulting in $\mathbf{S}_i = \{751, 1523, 2675, 169, 3616\}$. The target depends on the objective used to train the model. In this study, we use two variations of Masked Language Modeling (MLM), which will be discussed in detail in Chapter 4.3.1. Fundamentally, the rationale behind Masked Language Modeling is that a certain percentage of the products in a shopping session is replaced with a special token, which is frequently referred to as the MASK token and has its own embedding in the model (Devlin et al., 2018). Subsequently, the model is asked to predict which products belong on the positions of the replaced token. As a result, the model learns to recognize patterns, relationships and co-occurrences between products (Devlin et al., 2018).

First, similar to the LSTM, the input is converted into input embeddings, so that each product (in the session) has its own vector representation (Geron, 2019). Considering our example, we would have the embeddings $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$ and $\mathbf{x}_5$ representing product IDs 751, 1523, 2675, 169 and 3616, respectively.

As mentioned before, the implementation of positional encoding allows the Transformer to simulate 'order of interactions' in the session. Considering the nature of our problem, one should question if the order of interactions or selected products in a session entails more information on customer preferences than if the order was random. In other words, there can be cases in which the customer's order of interactions or selected products does not contain any additional information on their preferences. For example, if the website of an online supermarket has the vegetable department as starting page, a customer might first add products from this department. In this case, the order of the products in the session contains more information about the website design than about the purchase intentions of the customers. Although this information could contribute to the prediction at a certain time step (e.g., if the fruit department comes after the vegetable department, there is a high probability that often fruits is bought after vegetables), it still does not provide the model with greater knowledge of the customers true preferences. Hence, it would be interesting to see to what extent the performance differs between a Transformer model with and without positional embedding. Based on the model configuration (i.e., positional encoding or not), we add positional encoding to the input embeddings before transferring into the encoder block. In this study, we use sinusoidal positional embeddings as seen in Vaswani et al. (2017).

As mentioned in Chapter 2, self-attention allows the model to consider other relevant items in the input to improve the encoding of the current item (Geron, 2019). To calculate self-attention, we use three inputs, namely the Query, Keys and Values (Vaswani et al., 2017). For each product in the session, we create a query $(\mathbf{q}_j)$, key $(\mathbf{k}_j)$ and value $(\mathbf{v}_j)$ vector by multiplying the input vectors by three trained matrices $(\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v)$. Normally, this would be formally notated as

$$\begin{aligned} \mathbf{q}_j &= \mathbf{x}_j \mathbf{W}_q \\ \mathbf{k}_j &= \mathbf{x}_j \mathbf{W}_k \\ \mathbf{v}_j &= \mathbf{x}_j \mathbf{W}_v \end{aligned}$$

where j is the position of the item in the session.

Next, considering the current item of interest, the score of each item in the session in reference to the item of interest is calculated by taking the dot product of the query vector of the current item and the key vector of the other items (Vaswani et al., 2017). In our example, if we consider the first item in the session (ID: 751) as the current item of interest $(\mathbf{q}_1)$, the score of the each item would be calculated as

$$score_j = \mathbf{q}_1 \mathbf{k}_j^T$$

where j is the position of the item in the session.

In order to stabilize the gradients in the model, the scores are divided by the square root of the dimension of the key vectors and, subsequently, normalized through a softmax operation (Vaswani et al., 2017), which is formally notated as

$$softmaxscore_j = softmax(\frac{\mathbf{q}_1\mathbf{k}_j^T}{\sqrt{d_k}})$$

where j is the position of the item in the session and d_k is the dimension of the key vectors.

Next, the softmax score of each product is multiplied by each value vector in order to only 'focus' on the important products as the unimportant products will be multiplied by low values (e.g., 0.01) and important products by higher values (e.g., 0.90). Logically, the highest attention would be on the product of interest itself. However, other relevant items will most likely also receive a high attention score. The use of the softmax operation is slightly comparable to the forget gate in the LSTM architecture, in which softmax operations are used to 'forget' information. At last, the outcome of multiplications are summed up to calculate the output of the self-attention for the current product of interest $(\mathbf{q}_1)$ (Vaswani et al., 2017). The complete operation is formally notated as

$$selfattention_1 = \sum_{j=1}^{N_i} softmax(\frac{\mathbf{q}_1\mathbf{k}_j^T}{\sqrt{d_k}})\mathbf{v}_j$$

where j is the position of the item in the session, N_i is the number of items in the session and d_k is the dimension of the key vectors.

Following Vaswani et al. (2017), we decrease computation time by using matrix calculation of self-attention. More specifically, we transform the input embeddings of the products in our session into one matrix $(\mathbf{X}_i)$, in which every row represents a certain product of the session. Subsequently, this matrix $(\mathbf{X}_i)$ is multiplied by the three trained weight matrices $(\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V)$. This results in matrices for Query $(\mathbf{Q})$, Keys $(\mathbf{K})$ and Values $(\mathbf{V})$. To compute the self-attention matrix, we use the following formula:

$$Attention(\mathbf{Q},\ \mathbf{K},\ \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}$$

where d_k is the dimension of the key vectors (Vaswani et al., 2017).

Moreover, we implemented Multi-Headed Attention. Vaswani et al. (2017) argue two major benefits of using Multi-Headed Attention. First, it improves the model's capacity to concentrate on various positions with 'infinite' memory and, second, it allows the attention layer to consist of various 'representation subspaces'. In Multi-Headed Attention, there are multiple sets of the aforementioned Query, Keys and Values weight matrices. Each set is initialized at random. According to Vaswani et al. (2017), these sets allow the attention layer to consist of various 'representation subspaces'. In other words, as the multiplication of the input embeddings with the different weight matrices in each set results in h different Scaled Dot-Product Attention matrices, all of these matrices could potentially 'focus' on different characteristic of the products. Consider our session i that consists of 'an organic vegan chicken substitute' and 'organic eggs' and assume that the other products in the session are 'vegan pasta sauce', 'organic pasta' and 'organic toilet paper'. In the LSTM, the preferences of the customer were updated with ’organic’, ’protein rich’ and ’vegetarian’ products after the second time step. In contrast, the Transformer uses the various matrices to focus on various aspects of the customer’s preferences (e.g., ’organic’, ’protein rich’ and ’vegetarian’) in parallel. For example, when encoding the first product (organic vegan chicken substitute), the subspace that entails ’organic’ would give the highest attention to ’organic vegan chicken substitute’, ’organic toilet paper’, ’organic eggs’ and ’organic pasta’. In contrast, the subspace that entails ’vegan’ would most likely have a high attention score for only ’organic vegan chicken substitute’ and ’vegan pasta sauce’.

Due to the fact that the Feed-Forward Layer is not capable of processing multiple matrices, we concatenate the multiple matrices into one matrix and multiply with another weight matrix $(\mathbf{W}_O)$ (see right-hand side of Figure 4.3) (Vaswani et al., 2017). Before feeding the output of the Multi-Head Attention into the Feed-Forward Layer of the encoder, we add the ‘original’ input embeddings to the output of the Multi-Head Attention and apply normalization (Vaswani et al., 2017). After passing the data through the Feed-Forward Layer, we add the normalized output from the previous step to the output of the Feed-Forward Layer and, again, apply normalization (Vaswani et al., 2017). As our model only consists of an encoder block (see left-hand side of Figure 4.3), we pass the output of the Feed-Forward Layer directly through a linear layer and, subsequently, apply softmax to derive the output probabilities.

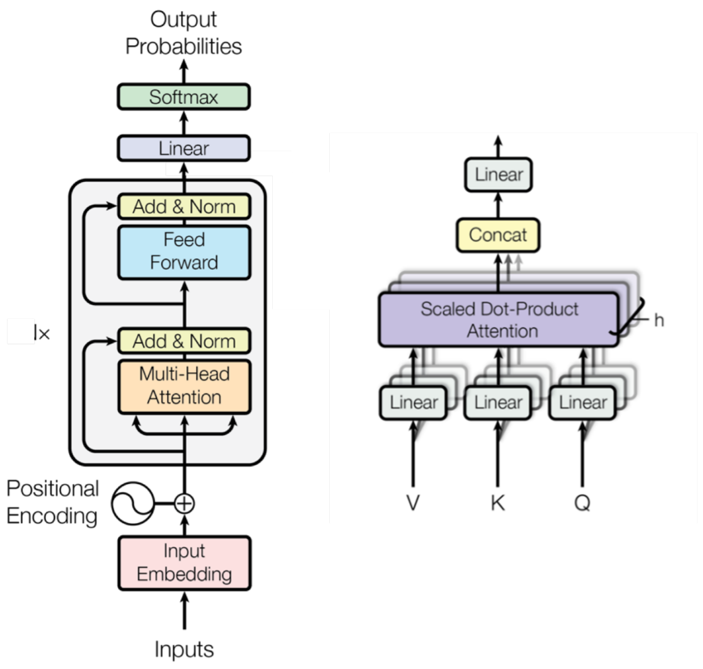


FIGURE 4.3: Adjusted Encoder Architecture (Left) and Multi-Head Attention (Vaswani et al., 2017) (Right)

4.3 Implementation and Training

In this subsection, we discuss the training processes, implementations and performance metrics of the models. Also, the benchmark algorithms are presented.

4.3.1 LSTM Training Objective

As mentioned before, the target variable for the LSTM during training is the next product added to the shopping cart after a certain sequence of products in a single session. Logically, this is represented by the last product added in the session. For each session, we eventually calculate a vector of probabilities $(\hat{\mathbf{y}})$ containing the probabilities of each unique product being the last product in the sequence. As the problem is presented as a multi-class classification problem, Cross Entropy Loss is used as loss function and is formally defined as

$$L(\hat{y}, y) = -\sum_{j=1}^{N} y_{i,j} \log(\hat{y}_{i,j})$$

where N is the number of unique classes (products), log is the natural log, y is a binary value indicating if the predicted product j is the target product in session i and $\hat{y}$ is the predicted probability of product j being the target product in session i (Geron, 2019).

4.3.2 Transformer Training Objectives

In the Transformer architecture, we used two different training objectives, creating two separate Transformer models (see Figure 4.4).

Masked Language Modeling

In the first model, we represent the problem as a Masked Language Model (MLM) objective as used in the BERT paper (Devlin et al., 2018). In order to train the model in recognizing patterns, relationships and co-occurrences between products, we use Masked Language Modeling, which entails that a certain percentage of the products in a shopping session is replaced with a special token. The token is frequently referred to as the MASK token and has its own embedding in the model (Devlin et al., 2018). In contrast with the original BERT paper, which masked 15% of the words in each sentence, we follow the best practices of Bianchi et al. (2020) and Sun et al. (2019) that recommended masking a higher portion of the products when dealing with relatively shorter sequences. Considering that shopping sessions in our data had an average length of 10, we choose to mask 25% of the products in each session (as in Bianchi et al., 2020). Subsequently, the model is asked to predict which product originally was replaced by the MASK token. During this task, the model has access to all non-masked products in the session as input. Hence, the input sequence consists of the products in a session with 25% of these products replaced by the MASK token (see Figure 4.4) and the target is to predict the original products that were replaced by the MASK token. In this case, the self-attention mechanism would only focus on the current product of interest and the other non-masked products. In other words, masked products other than the product of interest would not be considered for self-attention. For each MASK token in a session, we calculate a vector of probabilities $(\hat{\mathbf{y}})$ containing the probabilities of each unique product being the originally masked product. For the MLM Transformer, as the problem is presented as a multi-class classification problem, Cross Entropy Loss is used as loss function and is formally defined as

$$L(\hat{y}, y) = -\frac{1}{M_i} \sum_{m=1}^{M_i} \sum_{j=1}^{N} y_{i,j,m} \log(\hat{y}_{i,j,m})$$

where N is the number of unique classes (products), M_i is the number of MASK tokens in session i, log is the natural log, y is a binary value indicating if the predicted product j is the m^{th} originally masked product in session i and $\hat{y}$ is the predicted probability of product j being the m^{th} originally masked product in session i (Geron, 2019).

Replacement Token Detection

In the second model, we represent the problem as a Replacement Token Detection (RTD) objective, inspired by the ELECTRA paper (Clark et al., 2020). In the ELECTRA paper, RTD consists of two parts: a (MLM) generator and a discriminator (Clark et al., 2020). First, a certain percentage of the tokens or words in a session or sequence is replaced by a MASK token. Next, the generator (an MLM model) is asked to predict which tokens or words originally were replaced by the MASK token. Subsequently, the MASK tokens in the session or sequence are replaced with the predicted tokens of the generator (Clark et al., 2020). Afterwards, the 'updated' sequence (i.e., containing the predicted tokens of the generator) is used as input for the discriminator, which is asked to indicate for each token if it has been replaced or not, making it a binary classification task. As the discriminator model calculates loss over all tokens, Clark et al. (2020) argue that it allows the model to learn from each token in the sequence instead of only the masked ones.

By using an MLM model as generator to generate the input sequence for the discriminator, Clark et al. (2020) hoped to provide more plausible replacement words to the discriminator, potentially helping its learning process. However, one could argue that in language modeling, due to the grammatical rules and structure, the plausibility of words might be easier to predict than the plausibility of products in (grocery) purchase sessions (Sun et al., 2019). This, in turn, could limit the benefits of using an MLM generator in an online grocery shopping context. Therefore, in contrast to the original ELECTRA paper (Clark et al., 2020), we have chosen to simplify the model and shorten training time by only using the discriminator part of the original ELECTRA implementation and replace the generator's predictions with random products. In others words, instead of training an MLM model to generate the input sequence of the discriminator, the input is generated by replacing 25% of the products in an input shopping session with random products. Subsequently, the model (discriminator) is asked to indicate which of the products in the session were replaced. During this task, the discriminator has access to all other products in the session. In essence, although simplified, this approach still allows the model to learn from all products in the session, securing the main benefit from the RTD objective (Clark et al., 2020). Hence, the input sequence consists of the products in a session with 25% of these products replaced by random products (see Figure 4.4) and the target is to predict, for each product, if it is the original product or a corrupted product. In this case, the self-attention mechanism would focus on the current product of interest and all other products as it does not know which ones are corrupted or not. For the RTD Transformer, as the problem is presented as a binary classification problem, Binary Cross Entropy Loss is used as loss function and is formally defined as

$$L(\hat{y}, y) = -\frac{1}{N_i} \sum_{j=1}^{N_i} y_{i,j} \log(\hat{y}_{i,j}) + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j})$$

where N_i is the number of products in session i, log is the natural log, y is a binary value indicating if product j in session i is replaced and $\hat{y}$ is the predicted probability of product j in session i being replaced (Geron, 2019).

Moreover, to minimize the loss, 'Adam' (as proposed by Kingma and Ba (2014)) was used as optimization algorithm in all models.

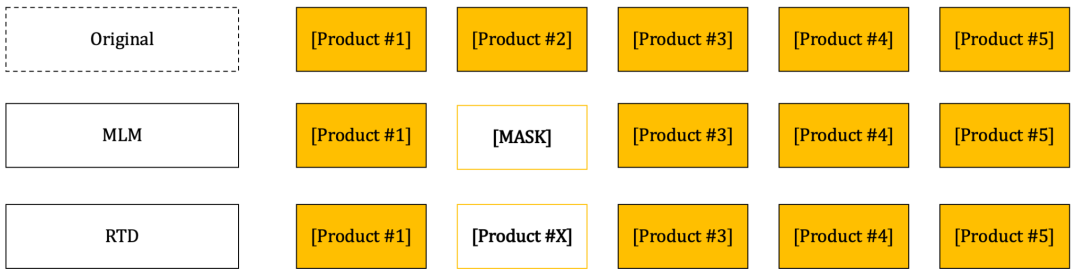


FIGURE 4.4: Masking Procedures of Transformer Networks

4.3.3 Test Objective and Performance Metrics

To make a good comparison between LSTM and the Transformers, the target variable in the test phase is the next item added to the shopping session after a certain sequence (i.e., the last product added to the session). For the MLM Transformer, this entails that the MASK token will be placed on the last item in the session. For the RTD Transformer, since the underlying Transformer architecture of the discriminator is similar to the architecture used in the MLM Transformer, it is possible to use this model in a similar fashion as an MLM model by changing the prediction task. Hence, in the test phase, we change the prediction task of the RTD Transformer to an MLM objective and place the MASK token on the last item in the session. In this sense, for each session, the models predict the probability of each distinctive product being the next product in the session. Subsequently, the products with the highest predicted probabilities will be presented as recommendations in a recommendation list, which will be used for evaluation as discussed in detail below.

Following previous literature (e.g., Ludewig and Jannach, 2018; Sun et al., 2019), the evaluation metrics used in this study are hit rate (HR), coverage (COV) and average rank. HR@n indicates the percentage of observations in which the correct target product is among the top n predicted products of the recommendation list. For example, a HR@20 of 0.10 indicates that the correct target product is within the top 20 products of the recommendation list for 10% of the observed sessions. COV@n indicates to what extent the model recommends various products in the top n of the recommendation list. This is important as it shows the models capacity to provide diverse recommendations (Ludewig and Jannach, 2018). It is calculated by diving the number of distinct products in all recommendation lists of a model by the total number of distinct products in the data. As the selection of the cutoff point is dependent on the context in which the recommendations are given, we use several values for n when assessing the models. First, following common practice in recommendation system literature, we use $n = 10$ and $n = 20$. Moreover, we also use 1% and 2.5% of the total number of unique products in the data, resulting in $n = 62$ and $n = 155$, respectively. Moreover, for a more refined evaluation, we also use average rank as an evaluation metric. In contrast with hit rate, average rank considers the rank of the target product in the entire recommendation list ($n = 6175$) and essentially indicates the average quality of recommendations. Also, to determine the stability of the model performance between sessions, we report the standard deviation of the rank.

4.3.4 Enriched Information and Positional Encoding

Inspired by Mizrachi and Levin (2019), we trained an RTD Transformer and an MLM Transformer with enriched information, creating 'RTD+' and 'MLM+', respectively. As mentioned before, this enriched information entailed the product category and the several binary variables (vegan, lactose-free, organic and gluten-free). This information was added to the model through concatenation of the embeddings of product, product category, and the binary variables. Subsequently, to reduce the size of the model, the concatenated embeddings were projected onto the same dimension size as the initial product embeddings through a linear layer. For equal comparison between the LSTM and Transformer, we also added enriched information to the LSTM in a similar manner, creating 'LSTM+' (Mizrachi and Levin, 2019). For Transformers, it is important to note that the addition of extra information happens before the positional embedding is added. Moreover, as discussed in section 4.2.2, we also implemented two versions of the Transformer without positional embedding (NP-RTD and NP-MLM). As we aim to compare the isolated effects of positional encoding and the addition of extra information, the hyper-parameters of LSTM+, RTD+, MLM+, NP-RTD and NP-MLM are identical to the optimized hyper-parameter settings of their original counterparts. In others words, the hyper-parameters of these adjusted models are not separately tuned from their original implementation.

4.3.5 Models Configurations

We have trained and evaluated multiple configurations of the models. We trained all models for 10 epochs with a batch size of 128. For all models, we experimented with various sizes of the embedding dimension and hidden dimension, both ranging from $d = 64$ to $d = 512$. Moreover, we tried multiple values for the number of layers, ranging from $l = 1$ to $l = 8$. For the Transformer models, we also tried several number of attention heads, ranging from $h = 2$ to $h = 8$. Based on the performance on the validation set, we present the best hyper-parameter configurations of each model in Table 4.1. Please note that the configurations of the hyper-parameters of the LSTM+, RTD+, MLM+, NP-RTD and NP-MLM Transformers are identical to the optimized hyper-parameter settings of their original counterparts (LSTM, RTD and MLM).

4.3.6 Benchmark Algorithms

In this study, a simpler algorithm is used as benchmark to assess and put the performance of the neural models into perspective. In particular, we use the aforementioned adjusted session-based collaborative filtering algorithm (see Chapter 2), which uses the currently selected products or interactions in a session as the variables to calculate similarity to other

Model	MLM	RTD	LSTM
Input Dimension Size	256	256	256
Hidden Feed-Forward Size	1024	1024	-
Hidden State Size	-	-	256
h	8	8	-
l	2	2	2
Dropout Rate	0.05	0.05	0.05

TABLE 4.1: Model Configurations

purchase sessions. In practice, similar to previous literature (Bonnin and Jannach, 2014), each session is portrayed as a binary vector with the size of the number of unique products N. Subsequently, we used a k-Nearest Neighbours algorithm, in which the similarity between sessions is measured by the Cosine similarity between these two binary vectors corresponding to the sessions in question. The Cosine similarity between two sessions is calculated as shown in following formula:

$$CosineSimilarity(\mathbf{a},\mathbf{b}) = \frac{\sum_{j=1}^{n} \mathbf{a}_j \mathbf{b}_j}{\sqrt{\sum_{j=1}^{n} \mathbf{a}_j^2}\sqrt{\sum_{j=1}^{n} \mathbf{b}_j^2}}$$

where a_j and b_j are the binary values for product j in the two separate vectors (sessions), and N is the number of unique products in the data.

This way, similar sessions (i.e., sessions with similar products) result in greater Cosine similarity. As recommendations, the algorithm provides items with the highest frequency in the k most similar sessions. In this study, we evaluated several values of k on the validation set ranging from 50 to 5000.

In the validation set, we noticed that the performance of the k-NN algorithm in the multiple metrics is highly dependent on k. Considering the practical relevance of the metrics, we used HR@20 as the decision criteria as a recommendation list of 20 items is more common and seems more practical than a list of 62 or 155 items (Jannach and Ludewig, 2017). We found the optimal value for HR@20 at $k = 200$.

5 Results

In this chapter, the results of the tested recommendation models are presented. First, based on the introduced metrics, we discuss the performance of the models on the complete test set. Moreover, to investigate the robustness of the models, we explore how the models perform in different contexts by splitting the test set on characteristics of the sessions. In particular, we split the test set based on popularity and session length.

5.1 Model Performance

The results of the models on the test set are provided in Table 5.1. The best model is displayed in bold and the second best model is underlined. In general, based on the performance of the popularity benchmark, we can conclude that all models have an added predictive power over recommending based on product popularity. Moreover, the use of the k-NN algorithm as a benchmark enables us to put the performance of the neural models into perspective.

To gain insight in the quality of the models, we use multiple performance metrics. In terms of Hit Rate, we notice that the k-NN algorithm performs best at the cutoff points 10 (HR@10) and 20 (HR@20) with 0.172 and 0.239, respectively. However, when increasing the cutoff point to 62(HR@62) and 155 (HR@155), the enriched MLM Transformer (MLM+) seems to perform best with 0.369 and 0.525, respectively. In terms of Transformer models, we observe that the Transformers suffer a decline in Hit Rate performance when positional encoding is not implemented. Moreover, considering the overall better performance of the other Transformer models compared to the LSTM, it is interesting that the Transformers without positional encoding (NP-MLM and NP-RTD) score lower than the LSTM for HR@10 and that NP-RTD even scores lower for HR@20. Furthermore, we only observe a positive effect of enriched information for both Transformers (MLM+ and RTD+), although this effect seems to be slightly greater for RTD Transformers than for MLM Transformers with an increase in HR@10 of 0.009 and 0.004, respectively.

In terms of coverage, the popularity benchmark scores the lowest at all cutoff points as it only recommends the most popular products, resulting in non-diverse recommendations. The greater coverage scores of the other models suggest that these models do not solely recommend based on product popularity. The k-NN algorithm has highest coverage at all cutoff points, followed by the enriched MLM Transformer. When comparing coverage of the neural models, there are some interesting observations. First, it is notable that, compared to the other neural models, the Transformer models without positional encoding (NP-MLM and NP-RTD) seem to have a relatively low coverage at all cutoff points with, for example, a COV@10 of 0.314 and 0.213, respectively. This is a decrease in COV@10 of 0.111 for NP-MLM and 0.100 for NP-RTD compared to their positional encoded counterparts (MLM and RTD). Second, we notice that all MLM Transformers (MLM, MLM+ or NP-MLM) have a higher coverage than their RTD counterpart (RTD, RTD+ or NP-RTD). At last, we observe that, compared to the original RTD and MLM, the addition of enriched information in RTD+

and MLM+ has a greater positive effect for the RTD Transformer than for the MLM Transformer. For example, the increase in COV@10 for MLM+ (compared to MLM) and for RTD+ (compared to RTD) are 0.053 and 0.100, respectively.

In terms of rank, the k-NN algorithm also has the highest average rank and the highest standard deviation of the rank. Taking into account the performance of the k-NN algorithm for HR@10 and HR@20, the relatively high average and standard deviation of rank seems to suggest that this algorithm is either very confident or totally clueless about which items to recommend. This, in turn, could be an explanation of the high coverage of the k-NN as it will recommend items from non-similar sessions if there are not enough similar sessions. Since non-similar sessions have a relatively high probability of containing items that have weak co-occurrence with the products in the session of interest, the absence of sufficient similar sessions might introduce a level of randomness in the recommendations, resulting in a 'clueless' recommendation. In contrast, all neural models, especially the Transformer models, have a relatively low average rank and standard deviation of the rank compared to the k-NN and the popularity benchmark. In particular, the best performing models in terms of rank are the enriched Transformers (MLM+ and RTD+) with an average rank of 416 and 422, respectively. Similar to hit rate and coverage, the positive effect of enriched information on rank seems to be greater for RTD Transformers than for MLM Transformers. However, the negative effect of the absence of positional encoding on rank seems to be equal between the two Transformers.

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	0.077	0.113	0.210	0.342	0.002	0.003	0.010	0.025	805	1037
k-NN	***0.172***	***0.239***	0.365	0.473	***0.717***	***0.836***	***0.968***	***0.995***	1424	1914
MLM	0.151	0.216	0.362	0.519	0.425	0.533	0.688	0.792	427	672
RTD	0.144	0.207	0.354	0.509	0.339	0.445	0.617	0.736	441	689
LSTM	0.130	0.186	0.312	0.451	0.386	0.483	0.634	0.735	590	874
MLM+	0.155	0.221	***0.369***	***0.525***	0.478	0.585	0.726	0.821	***416***	***661***
RTD+	0.153	0.218	0.365	0.522	0.439	0.547	0.693	0.796	422	670
LSTM+	0.130	0.185	0.310	0.450	0.368	0.468	0.619	0.729	591	875
NP-MLM	0.127	0.187	0.330	0.491	0.314	0.416	0.603	0.734	449	682
NP-RTD	0.123	0.181	0.325	0.484	0.239	0.331	0.508	0.659	462	697

TABLE 5.1: Model performance on test set (best model in bold and second-best model is underlined)

5.2 Robustness of Models

In order to investigate how the models perform in different scenarios, we looked at the results from different angles by splitting the test set based on different criteria. Robustness is measured as the percentage-wise change in model performance between different sets. For completeness, the absolute performance of each model in each set is also assessed.

5.2.1 Popularity

As mentioned before, one could argue that in certain contexts the value of recommending popular items is lower than recommending mid-popular or long-tail items. For example, if the 300 most popular items of an online retailer account for more than 50 percent of the purchases, it is likely that customers will find these items on their own. Therefore, the value of recommending these items is lower and it would be more valuable to recommend long-tail products. However, in a repetitive purchase context (e.g., grocery shopping), if a company uses the real-time predictions to provide real-time offers to nudge the customer into purchase, it might still be valuable to 'remind' them of popular compatible products. This is already partially measured by coverage as it shows the diversity in the recommendation list. However, coverage does not indicate the quality of 'long tail' recommendations. Given that popular products are more likely to be rated highly by recommendation systems, we did a popularity split on the test set based on the popularity level of the target product in the session. The popularity level of a product was determined based on the frequency of each product in the training set. By determining the split based on the 25th percentile, 50th percentile and 75th percentile of cumulative purchases, four levels of product popularity were created. To clarify, this means that popularity level 1 consists of products with the highest purchase frequencies that entail 25 percent of the total number of purchases. In contrast, popularity level 4 consists of products with the lowest purchase frequencies that entail 25 percent of the total number of purchases. Logically, this results in popularity level 1 containing fewer products than popularity level 4. Subsequently, based on the target product in the test set, the different sessions were divided into the subsets, resulting in a separate test set for each popularity level with level 1 being the most popular items. In this way, we are able to identify the performance of each model for different popularity levels. Please note that the results of POP will not be included in the comparison given that it is self evident that its performance is highly dependent on the popularity of the set. The results of the models on the popularity-based sets is presented in Table 5.2.

For sessions with popularity level 1, the best performing model in HR@10 is k-NN with only a slight advantage over RTD+. In contrast, MLM+ has the best performance in HR@20 and HR@62. Furthermore, the LSTM achieves the highest HR@155 and performs the best in rank with an average rank of 38. For the sessions with popularity level 2, we find results similar to the complete test with the highest HR@10 and HR@20 for k-NN and the highest HR@62 and HR@155 for MLM+. Moreover, among the models, MLM+ has the best average rank. For the sessions with popularity level 3, the k-NN scores the highest HR@10, HR@20 and HR@62. Again, MLM+ yields the best performance in HR@155 and the average rank. For the least popular items (popularity level 4), the k-NN outperforms all other models at all cutoff points of HR. However, in terms of rank, the regular MLM scores the best average and MLM+ has the most stable ranking. Again, the neural models seem to strongly outperform the k-NN in the average rank with MLM+ as the best model. This pattern is similar to the aforementioned findings on the complete test set, in which the k-NN performance deviates highly between sessions.

Besides the absolute performance in each set, it is interesting to investigate the changes in model performance between sets. Overall, we observe that performance in hit rate and rank of all models is proportional to the popularity of products (i.e., recommendations are more accurate when dealing with popular items). A reason for this could be that the models are biased to popular products as it sees these products more frequently in training, making the prediction of popular items easier while making the prediction of less popular items more difficult (Geron, 2019). Moreover, this change in performance is greatest for HR@10 and attenuates as the cutoff point (n) increases. For model comparison of robustness, we look at the percentage decrease in HR@10 and HR@20 from popularity level 1 to level 4. The model that is least affected by the change in product popularity is the k-NN with a percentage decrease in performance of 85.5% and 82.2%, respectively. From the neural models, the least affected model is LSTM+ with a decrease of 88.3% in both HR@10 and HR@20. The least robust models (i.e., with the highest decline in performance) are the two Transformer without positional encoding. In particular, NP-MLM decreases 96.2% and 94.1% in performance, and NP-RTD decreases 96.4% and 95.2% in HR@10 and HR@20, respectively. Furthermore, it is noteworthy that the relative difference in performance between the Transformers with and without positional encoding increases as popularity decreases. For example, for popularity level 1, the percentage-wise decline in HR@10 and HR@20 from MLM to NP-MLM was 1.2% and 1.1%. In contrast, the decline was 57.9% and 48.4% for popularity level 4. As the general difficulty of the prediction task seems to increase as popularity decreases, a potential explanation for the increasing relative difference could be that the extra information provided by positional encoding is relatively more valuable when predicting less popular items compared to predicting popular items. Hence, the absence of positional encoding has a higher impact on performance when predicting less popular items.

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	-	-	-	-	-	-	-	-	-	-
k-NN	***0.433***	0.551	0.707	0.802	***0.502***	***0.659***	***0.880***	***0.968***	456	1252
MLM	0.423	0.570	0.818	0.950	0.313	0.418	0.593	0.718	41	85
RTD	0.420	0.567	0.822	***0.951***	0.252	0.346	0.527	0.668	41	83
LSTM	0.358	0.520	0.811	0.967	0.321	0.418	0.569	0.687	***38***	***59***
MLM+	0.430	***0.576***	0.823	***0.951***	0.346	0.459	0.629	0.742	40	85
RTD+	0.432	***0.576***	0.820	0.949	0.319	0.427	0.603	0.718	41	88
LSTM+	0.359	0.520	0.806	0.965	0.313	0.407	0.563	0.681	***38***	***59***
NP-MLM	0.418	0.564	0.815	0.944	0.226	0.320	0.505	0.660	43	94
NP-RTD	0.413	0.564	***0.824***	0.950	0.171	0.254	0.427	0.594	41	89

(A) Popularity level 1

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	-	-	-	-	-	-	-	-	-	-
k-NN	***0.153***	***0.229***	0.393	0.537	***0.552***	***0.698***	***0.904***	***0.979***	1266	1902
MLM	0.130	0.197	0.411	0.680	0.346	0.451	0.625	0.742	157	218
RTD	0.124	0.190	0.403	0.676	0.279	0.375	0.555	0.689	160	221
LSTM	0.106	0.148	0.308	0.591	0.341	0.436	0.594	0.704	179	***207***
MLM+	0.134	0.205	***0.418***	***0.688***	0.385	0.494	0.660	0.767	***155***	218
RTD+	0.131	0.200	0.416	0.683	0.355	0.462	0.625	0.741	156	220
LSTM+	0.104	0.147	0.307	0.592	0.334	0.429	0.583	0.702	179	208
NP-MLM	0.095	0.163	0.379	0.661	0.251	0.349	0.535	0.682	169	229
NP-RTD	0.088	0.154	0.377	0.662	0.197	0.282	0.455	0.617	168	225

(B) Popularity level 2

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	-	-	-	-	-	-	-	-	-	-
k-NN	***0.097***	***0.150***	***0.266***	0.374	***0.559***	***0.707***	***0.916***	***0.981***	1643	1968
MLM	0.074	0.113	0.211	0.368	0.352	0.461	0.636	0.753	384	428
RTD	0.065	0.102	0.199	0.353	0.285	0.386	0.567	0.700	394	432
LSTM	0.066	0.091	0.147	0.240	0.344	0.440	0.598	0.705	506	471
MLM+	0.077	0.119	0.217	***0.376***	0.395	0.505	0.670	0.758	***375***	***424***
RTD+	0.074	0.117	0.215	0.373	0.365	0.474	0.641	0.754	378	425
LSTM+	0.065	0.089	0.145	0.239	0.333	0.432	0.589	0.703	507	473
NP-MLM	0.044	0.075	0.168	0.334	0.258	0.356	0.545	0.695	405	432
NP-RTD	0.040	0.067	0.154	0.317	0.201	0.281	0.462	0.618	412	434

(C) Popularity level 3

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	-	-	-	-	-	-	-	-	-	-
k-NN	***0.063***	***0.098***	***0.178***	***0.266***	***0.619***	***0.747***	***0.929***	***0.987***	2075	1969
MLM	0.038	0.064	0.121	0.198	0.384	0.491	0.662	0.774	***938***	942
RTD	0.030	0.052	0.106	0.179	0.304	0.411	0.589	0.717	1021	***926***
LSTM	0.042	0.061	0.098	0.139	0.366	0.466	0.612	0.720	1429	1165
MLM+	0.041	0.067	0.128	0.206	0.428	0.540	0.698	0.802	955	931
RTD+	0.038	0.063	0.121	0.202	0.396	0.506	0.666	0.777	971	943
LSTM+	0.043	0.061	0.097	0.139	0.354	0.453	0.603	0.717	1432	1163
NP-MLM	0.016	0.033	0.078	0.155	0.275	0.376	0.568	0.710	1030	939
NP-RTD	0.015	0.027	0.068	0.135	0.215	0.301	0.478	0.637	1071	951

(D) Popularity level 4

TABLE 5.2: Model performance on test sets per popularity level

5.2.2 Session Length

As previous research shows that there are differences between models in the capability of providing recommendations for different lengths of sessions (Tan et al., 2016; Sun et al., 2019), it is interesting to explore how the models perform in sessions with different lengths. To facilitate this process, we split the data in four separate sets. The 'small' set consists of sessions with 2 to 5 items, the 'low-medium' set of sessions with 6 to 10 items, the 'high-medium' set of sessions with 11 to 15 items, the 'large' set of sessions with 16 to 20 items and the 'extra large' set of session with 21 to 25 items. The results of the models on the length-based sets is presented in Table 5.3.

Similar to the main results, we notice that, for each session length, the neural models consistently outperform the benchmarks (k-NN and POP) in terms of average rank with MLM+ as the best model. Among the neural models, MLM+ also seems to be the best performing model in terms of hit rate at every session length. However, it does not always outperform the strongest benchmark (k-NN). In particular, MLM+ has a 6.9% and 15% advantage over the k-NN benchmark in HR@62 and HR@155 in small sessions. In contrast, in HR@10 and HR@20, it seems to be at a 10.2% and 3.8% disadvantage compared to the k-NN. We observe similar results in the low-medium set with MLM+ as the best performing model in HR@62 (+0.6%) and HR@155 (+11.9%), and k-NN as the best performing model in HR@10 (-12.2%) and HR@20 (-10.0%). In the high-medium and large sets, the MLM+ only outperforms the k-NN in HR@155 (+8.7% | +3.8%) and gets outperformed by the k-NN in HR@10 (-10.0% | -4.2%), HR@20 (-10.9% | -6.1%) and HR@62 (-3.0% | -5.1%). For sessions with a length greater than 21, the MLM+ outperforms the k-NN for HR@10 (+5.3%), HR@20 (+1.5%) and HR@155 (+5.3%) and is only slightly outperformed in HR@62 (-0.9%).

Overall, similar to popularity level, we observe that the session length is negatively correlated with performance (i.e., most models have higher accuracy in shorter sessions). This seems counter-intuitive as generally greater length means more information, which should decrease the difficulty of the prediction task (Geron, 2019). However, considering the context of online grocery shopping, one could argue that smaller sessions might consist of more targeted purchases, resulting in more logical product co-occurrences (e.g., only buying pasta and pasta sauce). In contrast, in longer sessions, customers might do their weekly grocery shopping, resulting in less logical product co-occurrences (e.g., toilet paper and bread). In this case, the extra products in the session might introduce noise instead of adding useful information (Sun et al., 2019). Moreover, we observe that the magnitude of the proportional decline in performance is not equal for all models. For example, the LSTM seems to relatively gain competitive performance as the session length increases. In others words, while the LSTM is the least performing neural model in the small set, it starts to minimize the performance differences with the other models, including the k-NN benchmark, as session length increases. Moreover, it even gains advantage over the Transformer without positional encoding in HR@10 and HR@20 from session length greater than 6 and in HR@62 in session length greater than 11. This is also evident from the robustness analysis across different session lengths of each model. When comparing the performance in the smallest and largest sets, the LSTM only suffers a slight decrease of 15.8% and 14.1% in HR@10 and HR@20, respectively. In contrast, the models with the greatest decline in performance are NP-MLM with 47.9% and 43.9%, and NP-RTD with 49.4% and 43.5%. Furthermore, we observe that the models with enriched information are more robust against changes in length than their original counterparts. For example, HR@10 and HR@20 of MLM drop with 23.8% and 25.1%, while the same metrics of MLM+ decrease with 20.5% and 19.6%. Interestingly, these differences seem even greater for the RTD Transformers with a drop of 34.5% and 31.1% performance of RTD in HR@10 and HR@20, and only a 24.0% and 22.6% decline in the same metrics of RTD+. Lastly, in the k-NN benchmark, we notice a decline of 32.1% and 23.8% for HR@10 and HR@20.

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	0.087	0.123	0.214	0.351	0.002	0.003	0.010	0.025	802	1052
k-NN	***0.196***	***0.260***	0.378	0.486	***0.537***	***0.681***	***0.914***	***0.982***	1419	1914
MLM	0.172	0.247	0.400	0.556	0.313	0.420	0.597	0.719	412	699
RTD	0.168	0.241	0.395	0.548	0.253	0.356	0.534	0.672	420	709
LSTM	0.146	0.206	0.336	0.477	0.343	0.444	0.595	0.709	559	863
MLM+	0.176	0.250	***0.404***	***0.559***	0.354	0.470	0.644	0.754	***406***	***697***
RTD+	0.175	0.248	0.402	0.557	0.325	0.437	0.609	0.725	409	700
LSTM+	0.144	0.204	0.334	0.474	0.329	0.426	0.584	0.703	561	866
NP-MLM	0.165	0.237	0.387	0.547	0.256	0.356	0.540	0.680	421	707
NP-RTD	0.160	0.230	0.383	0.541	0.200	0.281	0.457	0.611	428	713

(A) Small (2-5)

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	0.080	0.118	0.219	0.351	0.002	0.003	0.010	0.025	801	1045
k-NN	***0.172***	***0.240***	0.361	0.464	***0.589***	***0.730***	***0.921***	***0.984***	1427	1907
MLM	0.147	0.211	0.357	0.512	0.351	0.462	0.633	0.751	434	680
RTD	0.142	0.202	0.350	0.505	0.289	0.391	0.568	0.704	449	701
LSTM	0.130	0.187	0.310	0.448	0.355	0.455	0.609	0.715	597	884
MLM+	0.151	0.216	***0.363***	***0.519***	0.402	0.516	0.675	0.784	***423***	***670***
RTD+	0.150	0.214	0.359	0.516	0.366	0.472	0.641	0.754	429	679
LSTM+	0.130	0.185	0.307	0.449	0.341	0.441	0.597	0.709	600	886
NP-MLM	0.125	0.186	0.330	0.488	0.270	0.370	0.558	0.702	452	687
NP-RTD	0.121	0.180	0.326	0.480	0.209	0.296	0.476	0.633	467	705

(B) Low-medium (6-10)

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	0.072	0.106	0.209	0.338	0.002	0.003	0.010	0.025	804	1014
k-NN	***0.160***	***0.229***	***0.361***	0.471	***0.542***	***0.690***	***0.895***	***0.971***	1446	1930
MLM	0.140	0.197	0.341	0.503	0.348	0.450	0.624	0.749	426	635
RTD	0.130	0.189	0.332	0.493	0.267	0.365	0.548	0.684	443	655
LSTM	0.125	0.179	0.300	0.443	0.336	0.428	0.587	0.697	582	843
MLM+	0.144	0.204	0.350	***0.512***	0.380	0.492	0.655	0.771	***412***	***618***
RTD+	0.141	0.199	0.345	0.506	0.351	0.462	0.626	0.747	419	629
LSTM+	0.125	0.176	0.298	0.444	0.324	0.421	0.579	0.696	583	842
NP-MLM	0.105	0.158	0.299	0.464	0.218	0.315	0.501	0.657	459	652
NP-RTD	0.099	0.152	0.292	0.455	0.168	0.244	0.422	0.587	475	668

(C) High-medium (11-15)

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	0.062	0.095	0.185	0.313	0.002	0.003	0.010	0.025	824	1020
k-NN	***0.144***	***0.214***	***0.353***	0.472	***0.457***	***0.612***	***0.842***	***0.958***	1408	1907
MLM	0.132	0.194	0.323	0.483	0.307	0.409	0.589	0.721	440	645
RTD	0.126	0.183	0.312	0.464	0.245	0.338	0.520	0.666	457	665
LSTM	0.120	0.172	0.298	0.431	0.299	0.392	0.548	0.668	597	861
MLM+	0.138	0.201	0.335	***0.490***	0.347	0.452	0.626	0.752	***423***	***623***
RTD+	0.134	0.199	0.329	0.485	0.322	0.431	0.598	0.728	434	640
LSTM+	0.119	0.172	0.294	0.433	0.287	0.386	0.544	0.673	599	861
NP-MLM	0.091	0.134	0.263	0.433	0.175	0.255	0.438	0.607	482	662
NP-RTD	0.088	0.130	0.257	0.424	0.133	0.204	0.362	0.538	496	680

(D) Large (16-20)

Metric	Hit Rate				Coverage				Rank	
n	10	20	62	155	10	20	62	155	Mean	Std
POP	0.058	0.092	0.184	0.310	0.002	0.003	0.010	0.025	823	1014
k-NN	0.133	0.198	***0.346***	0.473	***0.344***	***0.489***	***0.744***	***0.910***	1379	1915
MLM	0.131	0.185	0.338	0.483	0.287	0.391	0.573	0.710	437	645
RTD	0.110	0.166	0.314	0.461	0.213	0.302	0.485	0.638	470	671
LSTM	0.123	0.177	0.303	0.441	0.265	0.356	0.515	0.642	598	868
MLM+	***0.140***	***0.201***	0.343	***0.498***	0.306	0.417	0.597	0.728	***416***	***618***
RTD+	0.133	0.192	0.340	0.490	0.291	0.398	0.569	0.709	429	639
LSTM+	0.121	0.174	0.303	0.445	0.255	0.347	0.510	0.645	597	869
NP-MLM	0.086	0.133	0.264	0.420	0.137	0.201	0.368	0.541	483	650
NP-RTD	0.081	0.130	0.253	0.409	0.102	0.155	0.296	0.470	506	678

(E) Extra Large (21-25)

TABLE 5.3: Model performance on test sets per session length

6 Discussion

In this chapter, the interpretation of the results, the academical and managerial implications of the findings and the limitations of this study are discussed. Subsequently, we provide recommendations for future research.

6.1 Interpretation of Results

6.1.1 Transformer and Recurrent Neural Networks

Inspired by the enhanced performance of Transformer models in Natural Language Processing (NLP), the aim of this study was to compare the predictive performance of Transformers in providing session-based recommendations to the performance of recurrent neural networks (i.e., LSTM). In general, we found that, similar to the developments in NLP, Transformer models outperform the LSTM in session-based recommendations. A potential explanation for this could be that, as the Transformer uses bidirectional context, it is able to learn more combinations of co-occurrences between products, increasing its understanding of the meaning of the products. In contrast, the LSTM approaches the task as unidirectional (left to right) and is therefore restricted to the order of previous items when updating its hidden representations of the current item (Sun et al., 2019). Following this, as the LSTM is originally a sequential model, it also assumes a natural or logical order in the sequence (e.g., time series) (Sun et al., 2019). This assumption is, nonetheless, not always valid in purchase sessions, making the LSTM relatively less suitable for this session-based recommendations than bidirectional Transformers. Moreover, we also benchmark both models to a k-NN session-based recommendation model, which is inspired by the commonly used collaborative filtering approach in non-session based recommendation systems in general, and a popularity-based recommendation model (POP). As seen in Chapter 5, we found that the Transformers and LSTM consistently outperform POP and k-NN in terms of the average rank of the correct item in the recommendation list. This entails that, on average, the neural models are better capable of distinguishing relevant items from irrelevant items than the k-NN and the popularity-based recommender. Nonetheless, from a practical point of view, it should be noted that these ranks are relatively high (i.e., a customer will not scan through a recommendation list of hundreds of products). Therefore, it is also important to take the hit rates at several cutoff points into consideration as this indicates the percentage of sessions, in which the correct product was among the top recommendations. In this study, we looked at hit rates in the top 10, top 20, top 62 and top 155 recommendations. In general, the popularity-based recommender was outperformed by all other models, indicating that there is added value in personalizing recommendations. Moreover, LSTM was found to yield the lowest hit rates at all cutoff points compared to the k-NN and the other neural models. The best Transformer (MLM+) seems to outperform the k-NN algorithm in hit rates in the top 62 and top 155 recommendations and gets outperformed by the k-NN algorithm in hit rates in the top 10 and top 20 recommendations. It seems that although the k-NN provides more relevant recommendations in a recommendation list with length 10 to 20, it does not hold that proportion of relevance in a list of length 62 to 155. Considering the relatively high average rank of the correct item in k-NN recommendations, this suggests that the k-NN algorithm is either very confident or totally clueless in recommending. In contrast, although the Transformer has more relevant recommendations in general, it has more difficulties in ranking the most relevant items higher up in the top 10 and top 20 of the list compared to the k-NN. Therefore, none of the models consistently outperforms its strongest alternative as the list length of recommendations seems crucial in deciding which algorithm is better. The findings in our study are in line with previous studies (Latifi et al., 2021; Dacrema et al., 2019; Ludewig et al., 2021, 2020), in which k-NN methods are able to outperform neural networks in providing session-based recommendations on certain data sets. In this sense, k-NN still seems to be a difficult benchmark to consistently improve upon in recommendations as previously argued by Bianchi et al. (2020) and Latifi et al. (2021).

6.1.2 Effect of Training Objective, Positional Encoding and Enrichment

In this study, we also explored the effects of different training objectives, positional encoding and enrichment on the predictive performance and robustness of the models in different contexts.

In particular, we found that a Transformer trained with an MLM objective slightly outperforms an identical Transformer trained with an RTD objective in providing session-based recommendations. Nevertheless, it should be noted that the difference in performance is small. An explanation for the difference might be that the MLM objective is closer to the next item prediction task than the RTD objective. Moreover, during training, we observed that the RTD Transformer converged more moderately than the MLM objective, which in turn converged more rapidly. This is, nonetheless, in contrast with the findings in NLP, in which ELECTRA was found to outperform BERT in shorter training times (Clark et al., 2020). An explanation for this could be that, in the original ELECTRA implementation, the input for the discriminator in the RTD task is generated by a small MLM model. According to Clark et al. (2020), the MLM generator provides 'plausible' replacement words as input for the discriminator. As mentioned in Chapter 4, we, nevertheless, chose to replace the corrupted tokens with random products, which could have resulted in making the learning process potentially prone to the 'randomness' of the replacement products. However, considering that the differences in performance are small, it can still be argued that this version of RTD is a solid alternative for MLM in this context.

Moreover, we investigated the isolated effect of positional encoding on the performance of Transformer recommendation systems and if this effect differs between training objectives. Contradictory to the differences between the two objectives in the 'original' implementation, there seems to be no significant differences in the effect of removing positional encoding between the Transformers. In particular, we found that not implementing positional encoding decreases the performance of both MLM and RTD Transformers almost equally. This suggests that, in general, the positional encoding is an important part in the Transformer architecture, despite of the training objective. However, although the importance of positional encoding does not differ between training objectives, it does seem to differ in different contexts. The Transformers without positional encoding proved to be the least robust against changes in product popularity and session length. As mentioned before, an explanation for the proneness against changes in product popularity could be that the added information of positional encoding is relatively more valuable when predicting less popular items, resulting in greater impact on performance. In terms of session length, consistent with the findings of Sun et al. (2019), we observed that the absence of positional encoding does not lead to a great decline in performance for smaller session lengths. On the contrary, the decline in performance was more present in sessions with greater length, suggesting that the importance of positional encoding is proportional to the length of a sequence (Sun et al., 2019). One explanation of the increased importance of positional encoding in longer sessions is that it enables the attention layers in the model to put more emphasis on the products in the proximity of the target. For example, in our study, we observed that the k-NN algorithm was often outperformed by the Transformer models (e.g., RTD+ and MLM+) in sessions with length greater than 20. Moreover, the previously observed differences in performance between k-NN and LSTM became smaller as the session length increased. In addition, in terms of hit rate, the LSTM primarily outperformed the Transformers without positional encoding in sessions consisting of 16 products or more. Considering that both the LSTM and the Transformer with positional encoding benefit from information of the sequential order of interaction in the session, one could argue that the positional information enables these models to recognize shifts in preferences during the session (Jannach and Ludewig, 2017; Jannach et al., 2020). The occurrence of these preference shifts are more likely in longer sequences. Another reason could be that the positional encoding potentially helps the model in the training process. Without positional encoding to identify the positions of the products in the session, the model might experience greater difficulties in learning from simultaneous masked items as it has no proxy indicator to distinguish between the items (Sun et al., 2019). Subsequently, the model will treat the masked products as one identical product, increasing the complexity of the prediction task.

Besides these explanations, it could also be that, as mentioned in Chapter 4, the importance of positional encoding potentially originates from the possibility that the layout of the company's website is reflected in the order of interactions or purchases of the customer. In this case, the model might improve in predictive performance by learning the underlying website layout instead of improving its understanding of the actual customer preferences. From a practical marketing point of view in the context of session-based recommendations, one might argue that the benefits of improved performance outweighs the potential lack of increased knowledge on customer preferences. However, it is important to realize that our test objective is not identical to a real-life recommendation task. In practice, the main goal of recommendations is to remind the customer of a product that is a potential great addition or fit to the current items in their session. As we actually aimed to predict the last product in a sequence, there might be a bias towards products that, due to the layout of the online store, are often bought last. This decreases the importance of recommending a 'great fit' for prediction performance and increases the importance of recommending items that are frequently bought last. For instance, in practice, when a customer 'forgets' an item during a shopping session, the item should not necessarily have a higher chance of being a product that is often bought last. Nevertheless, our model might be biased towards these products. In this sense, we can argue that although positional encoding improved our predictive performance for our test objective, this might not be the case when implemented in practice. Another caveat of over-reliance on positional encoding is that, if the additional model performance is truly derived from and dependent on the layout of the website, a change in website layout would most likely result in a decline of performance.

At last, we also attempted to further improve performance by enriching our embeddings with new contextual information about the products, which was added through concatenation of the embeddings of product, product category, and the binary variables (Mizrachi and Levin, 2019). Although the LSTM did not benefit from the extra information, all the Transformer models did, resulting in a small increase in performance. The small increase suggests that, although most information is already learned through solely the product embeddings, the Transformers can still benefit from the extra contextual data. The importance of enriched information seems to be proportional to session length, meaning that in longer sequences the effect of enriched information becomes more evident, especially for RTD Transformers. Therefore, since it improves the overall model performance, it is encouraged to use of additional contextual data in session-based recommendation systems.

6.2 Implications

In this section, the academical and managerial implications of the findings are discussed.

6.2.1 Academical Implications

With the findings of this study, we contribute to the existing literature in (session-based) recommendation systems. First, we showed that Transformer neural networks are more effective in providing session-based recommendation than recurrent neural networks, more specifically LSTMs. Next, we investigated the use of Replacement Token Detection as a training objective in session-based recommendation and bench-marked it against the more commonly used Masked Language Modeling. From this, we found that although, contradictory to Clark et al. (2020), the RTD Transformer did not outperform the MLM Transformer in terms of computational time and performance, these two training objectives are both capable of providing session-based recommendation, making RTD a solid alternative for MLM in this context. Moreover, in accordance with previous research (e.g., Jannach et al., 2020), neural networks did not consistently outperform the k-Nearest Neighbour approach in session-based recommendations. Moreover, by experimenting with Transformer without positional encoding, we add to the work of Sun et al. (2019) that in the absence of positional encoding, besides the decline in performance of the Transformer trained on an MLM objective, we also noticed a decline in performance of the Transformer trained on an RTD objective, especially in longer sequences.

6.2.2 Managerial Implications

In this study, we emphasized the importance of session-based recommendations for online marketing and online businesses in general. We showed that, even in a highly repetitive and noisy data landscape, it is possible to provide useful recommendations based on solely session data. Moreover, we found that there is currently no State-of-the-Art session-based recommendation model, as all models perform differently per context and goal of the recommendation. With this knowledge, companies can adjust or maintain their current recommendation systems. Considering the financial and computational costs of training deep learning models and the small increase in performance limited to specific circumstances, it would be advisable for most companies to refer to cheaper alternatives, such as a k-Nearest Neighbour based algorithms, at this stage in the developments in neural based recommendation systems. Nonetheless, based on the findings for longer sessions in this study, it is advised to companies working with longer session sequences with sequential preference shifts during sessions to explore neural based recommendation systems, as k-NN approaches often have difficulties in recognizing these shifts and patterns (Jannach and Ludewig, 2017). Moreover, considering the increase in performance due to the extra session-based information, companies are encouraged to investigate what session-based information they can utilize to enrich the information currently available for their algorithms.

6.3 Limitations and Future Research

We identify several (potential) limitations of this study. First, there is no certainty on how the aggregation of data in subcategories has impacted the training process and, subsequently, the performances of the models. Second, in this study, we did not benchmark the performance of the models on multiple data sets. Therefore, it is uncertain if the findings of the study are data dependent and reproducible on different data sets. Third, we discovered that not all product classes were represented in training. This is likely due to random splitting the sessions in train, validation and test sets, in which the loss of training information could not be prevented as some products only occur in one or two sessions. Fourth, it is unknown if the website layout of Instacart has changed over the course of the data collection process. Therefore, it is uncertain if and how this impacted the results of the Transformer. Lastly, as mentioned before, we evaluated the models in an offline setting instead of in a practical in-field context (i.e., real recommender intervention). Consequently, our test objective is not identical to a real-life recommendation task, potentially resulting in different findings in practice.

For future research, it is recommended to reproduce this study on other data sets to validate if the findings are robust across different data sets. Furthermore, it is advised to reproduce and possibly improve the results of this study with other Transformer architectures (e.g., TransformerXL, XLNet or RoBERTa). Moreover, we observed a decline in performance when dealing with less popular items due to class imbalance in the training data. Considering the importance of recommending 'long-tail' items, it seems interesting to investigate the potential impact of implementing different weighting schemes in either the loss function or as a substitute for masking or corrupting probabilities. The introduction of different weighting schemes could potentially battle the problem of class imbalance. In line with this, it seems valuable to compare these weighted neural approaches to simpler algorithms, such as k-Nearest Neighbours, with implemented weighting schemes. Given the current state and developments of neural based recommendation systems and their limited advantages and advancements in model performance, it might also be valuable to develop a hybrid approach that applies the 'wisdom of the crowd' principle. In essence, such a model would, for example, utilize the recommendations from both approaches (k-NN and Transformer) by means of a trained weighted average to provide improved recommendations. For RNN-based recommendations, this combination has shown promising results (Jannach and Ludewig, 2017). Furthermore, we encourage the exploration of the use of RTD as training objective in (session-based) recommendation systems and how this differs from MLM-based Transformer in different contexts (e.g., longer sequences or higher density of data). Also, considering the advantages of ELECTRA over BERT in both computational time, model size and performance (Clark et al., 2020), it would be interesting to explore how smaller models of both training objectives perform in a session-based recommendation context. Moreover, it is interesting to investigate how the implementation of extra contextual information changes the behaviour of the attention layers in the multi-headed attention of the Transformer. The acquisition of in-depth knowledge of this process enables us to identify the value and potential of additional session-based data, which can improve the deep learning approaches in session-based recommendations. Furtermore, due to limited computational resources, we limited our study to sessions with a maximum length of 25 to improve training speed. It was, however, noticeable that differences in performances between the LSTM and k-NN decreased as the session length increased. It would be interesting to investigate if this trend continues in session lengths greater than 25 and if this could eventually result in the LSTM outperforming the k-NN in longer sessions. Lastly, as it is not certain to what extent the added performance of positional encoding is dependent on the underlying layout of the website, a potentially interesting (field) study would be to investigate how a sudden change in layout would impact the performance of a Transformer model trained on data from the previous layout. In line with this, it is advised to perform a field experiment in which the various recommendation models are tested in real recommender interventions (i.e., giving real-time recommendations to real customers).

6.4 Conclusion

In this study, we investigated the effectiveness of Transformers and recurrent neural networks in session-based recommendations. In conclusion, we could state that, similar to findings in Natural Language Processing, both Transformers trained with an MLM objective (e.g., BERT) and an RTD objective (e.g., ELECTRA) are an improvement over recurrent neural networks (i.e., LSTM) in providing session-based recommendations. However, they still appear to experience difficulties in steadily outperforming simpler k-Nearest Neighbour algorithms. In terms of model robustness, we observed that the performance of all models is proportional with the product popularity of the target product (i.e., better prediction performance for popular products) and negatively proportional with session length (i.e., better prediction performance for shorter sessions), although the magnitude of this change is not equal between models. In general, the k-NN has proven to be the most robust against changes in product popularity and the LSTM has proven to be the most robust against changes in session length, increasing in relative competitiveness as session length increases. Furthermore, we showed the effectiveness of positional encoding for the Transformer architecture and the added potential of contextual information for the performance of Transformer session-based recommendation systems. In particular, the Transformers without positional encoding are steadily outperformed by their original counterparts and show to be the least robust against changes in both popularity and session length, emphasizing the importance of positional encoding. In contrast, the enriched Transformers generally outperformed their original counterparts in both robustness and absolute performance, showcasing the potential of extra contextual information.

Bibliography

Adomavicius, G. and Tuzhilin, A. (2005a). Personalization technologies. *Communications of the ACM*, 48(10):83–90.

Adomavicius, G. and Tuzhilin, A. (2005b). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Arora, N., Ensslen, D., Fiedler, L., Liu, W. W., Robinson, K., Stein, E., and Schüler, G. (2021). The value of getting personalization right—or wrong—is multiplying. In *Next in Personalization 2021 Report*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bianchi, F., Yu, B., and Tagliabue, J. (2020). Bert goes shopping: Comparing distributional models for product representations.

Boerman, S. C., Kruikemeier, S., and Borgesius, F. J. Z. (2021). Exploring motivations for online privacy protection behavior: Insights from panel data. *Communication Research*, 48(7):953–977.

Bonnin, G. and Jannach, D. (2014). Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):1–35.

Clark, K., Luong, M., Le, Q. V., and Manning, C. D. (2020). ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198.

Cui, D. and Curry, D. (2005). Prediction in marketing using the support vector machine. *Marketing science (Providence, R.I.)*, 24(4):595–615.

Dacrema, M., Cremonesi, P., and Jannach, D. (2019). Are we really making much progress? a worrying analysis of recent neural recommendation approaches. RecSys '19, pages 101–109. ACM.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860.

Delic, A., Neidhardt, J., Nguyen, T. N., Ricci, F., Rook, L., Werthner, H., and Zanker, M. (2016). Observing group decision making processes. In *Proceedings of the 10th ACM conference on recommender systems*, pages 147–150.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Dhar, V., Geva, T., Oestreicher-Singer, G., and Sundararajan, A. (2014). Prediction in economic networks. *Information systems research*, 25(2):264–284.

Dimoka, A., Hong, Y., and Pavlou, P. A. (2012). On product uncertainty in online markets: Theory and evidence. *MIS quarterly*, 36(2):395–426.

Eliasy, A. and Przychodzen, J. (2020). The role of ai in capital structure to enhance corporate funding strategies. *Array (New York)*, 6:100017.

Ethayarajh, K., Duvenaud, D., and Hirst, G. (2019). Towards understanding linear word analogies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3253–3262, Florence, Italy. Association for Computational Linguistics.

Fausett, L. V. (1994). *Fundamentals of neural networks*. Prentice Hall Internat, Englewood Cliffs, NJ.

Fuentes, I., Nápoles, G., Arco, L., and Vanhoof, K. (2021). Best next preference prediction based on lstm and multi-level interactions. In *Proceedings of SAI Intelligent Systems Conference*, pages 682–699. Springer.

Gabel, S., Guhl, D., and Klapper, D. (2019). P2v-map: Mapping market structures for large retail assortments. *Journal of marketing research*, 56(4):557–580.

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Incorporated, Sebastopol.

Haubl, G. and Trifts, V. (2000). Consumer decision making in online shopping environments: The effects of interactive decision aids. *Marketing science (Providence, R.I.)*, 19(1):4–21.

Hidasi, B. and Karatzoglou, A. (2018). Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 843–852.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

Hidasi, B., Quadrana, M., Karatzoglou, A., and Tikk, D. (2016). Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 241–248.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hurley, N. and Zhang, M. (2011). Novelty and diversity in top-n recommendation–analysis and evaluation. *ACM Transactions on Internet Technology (TOIT)*, 10(4):1–30.

Instacart (2017). The instacart online grocery shopping dataset 2017.

Jannach, D. and Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. RecSys '17, pages 306–310. ACM.

Jannach, D., Mobasher, B., and Berkovsky, S. (2020). Research directions in session-based and sequential recommendation. *User Modeling and User-Adapted Interaction*, 30(4):609–616.

Javadi, M. H. M., Dolatabadi, H. R., Nourbakhsh, M., Poursaeedi, A., and Asadollahi, A. R. (2012). An analysis of factors affecting on online shopping behavior of consumers. *International journal of marketing studies*, 4(5):81.

Kamehkhosh, I., Jannach, D., and Ludewig, M. (2017). A comparison of frequent pattern techniques and a deep learning method for session-based recommendation. In *RecTemp@ RecSys*, pages 50–56.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koren, Y. (2010). Factor in the neighbors: scalable and accurate collaborative filtering. *ACM transactions on knowledge discovery from data*, 4(1):1–24.

Krizanova, A., Lăzăroiu, G., Gajanova, L., Kliestikova, J., Nadanyiova, M., and Moravcikova, D. (2019). The effectiveness of marketing communication and importance of its evaluation in an online environment. *Sustainability (Basel, Switzerland)*, 11(24):7016.

Latifi, S., Mauro, N., and Jannach, D. (2021). Session-aware recommendation: A surprising quest for the state-of-the-art. *Information sciences*, 573:291–315.

Li, Y. and Yang, T. (2017). *Word Embedding for Understanding Natural Language: A Survey*, pages 83–104. Guide to Big Data Applications. Springer International Publishing, Cham.

Lissitsa, S. and Kol, O. (2016). Generation x vs. generation y – a decade of online shopping. *Journal of retailing and consumer services*, 31:304–312.

Lops, P., de Gemmis, M., and Semeraro, G. (2010). *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Recommender Systems Handbook. Springer US, Boston, MA.

Ludewig, M. and Jannach, D. (2018). Evaluation of session-based recommendation algorithms. *User modeling and user-adapted interaction*, 28(4-5):331–390.

Ludewig, M., Mauro, N., Latifi, S., and Jannach, D. (2020). Empirical analysis of session-based recommendation algorithms. *User modeling and user-adapted interaction*, 31(1):149–181.

Ludewig, M., Mauro, N., Latifi, S., and Jannach, D. (2021). Empirical analysis of session-based recommendation algorithms. *User modeling and user-adapted interaction*, 31(1):149–181.

Medsker, L. and Jain, L. C. (1999). *Recurrent Neural Networks: Design Applications*. CRC Press.

Mizrachi, S. and Levin, P. (2019). Combining context features in sequence-aware recommender systems. In *RecSys (Late-Breaking Results)*, pages 11–15.

Olah, C. (2015). Understanding lstm networks. Accessed on Nov 17, 2021 from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Park, S.-H. and Han, S. P. (2013). From accuracy to diversity in product recommendations: Relationship between diversity and customer retention. *International journal of electronic commerce*, 18(2):51–72.

Ricci, F., Rokach, L., and Shapira, B. (2011). *Introduction to Recommender Systems Handbook*, pages 1–35. Recommender Systems Handbook. Springer US, Boston, MA.

Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). *Backpropagation: The basic theory*, pages 1–34. Backpropagation: Theory, architectures and applications. Psychology Press.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Sahoo, N., Singh, P. V., and Mukhopadhyay, T. (2012). A hidden markov model for collaborative filtering. *MIS quarterly*, pages 1329–1356.

Schwarzl, S. and Grabowska, M. (2015). Online marketing strategies: the future is here. *Journal of International Studies*, 8(2).

Shafqat, W. and Byun, Y.-C. (2020). A context-aware location recommendation system for tourists using hierarchical lstm model. *Sustainability*, 12(10).

Shmueli, G. and Koppius, O. R. (2011). Predictive analytics in information systems research. *MIS quarterly*, pages 553–572.

Strycharz, J., Smit, E., Helberger, N., and van Noort, G. (2021). No to cookies: Empowering impact of technical and legal knowledge on rejecting tracking cookies. *Computers in Human Behavior*, 120:106750.

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. (2019). Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450.

Tan, Y. K., Xu, X., and Liu, Y. (2016). Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 17–22.

Thirumalai, S. and Sinha, K. K. (2013). To personalize or not to personalize online purchase interactions: Implications of self-selection by retailers. *Information Systems Research*, 24(3):683–708.

Tran, T. N. T., Atas, M., Felfernig, A., Samer, R., and Stettinger, M. (2018). Investigating serial position effects in sequential group decision making. In *Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization*, pages 239–243.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Weinzierl, S., Stierle, M., Zilker, S., and Matzner, M. (2020). A next click recommender system for web-based service analytics with context-aware lstms. In *Proceedings of the 53rd Hawaii International Conference on System Sciences*, pages 1542–1551.

West, P. M., Brockett, P. L., and Golden, L. L. (1997). A comparative analysis of neural networks and statistical methods for predicting consumer choice. *Marketing Science*, 16(4):370–391.

Wollan, R., Barton, R., Quiring, K., and Ishakawa, M. (2017). Exceed expectations with extraordinary experiences. *Dublin: Accenture Strategy*.

Xiao, B. and Benbasat, I. (2007). E-commerce product recommendation agents: Use, characteristics, and impact. *MIS quarterly*, pages 137–209.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Zhang, T. C., Agarwal, R., and Lucas, H. C. (2011). The value of it-enabled retailer learning: Personalized product recommendations and customer store loyalty in electronic markets. *MIS quarterly*, 35(4):859–881.

Zhu, Y., Li, H., Liao, Y., Wang, B., Guan, Z., Liu, H., and Cai, D. (2017). What to do next: Modeling user behaviors by time-lstm. In *IJCAI*, volume 17, pages 3602–3608.