Food Recommender System based on Behavior Sequence Transformers

Arne Klages
aklages@uni-osnabrueck.de

Erik Nickel
ernickel@uni-osnabrueck.de

Jan-Luca Schöder
janlschroede@uni-osnabrueck.de

April 10, 2022

Course: Implementing ANNs with TensorFlow
- Winter term 2022/2023 -
University of Osnabrück

Abstract

In recent years, transformer based architectures became popular in deep learning. This paper presents a Behavior Sequence Transformer based recommender model, that is able to recommend the next best recipe for a user, given some previously cooked recipes. The model uses Multi-Head attention. This model was trained using a rather large dataset. Although our model overfitted on the training data, it still is able to recommend from learned data the best next recipe for the given input recipes. Overall, this approach looks very promising for this application and should be investigated further.

Contents

List of Acronyms

1 Introduction

In our current time, many people often do not have the time, energy or motivation to think about recipes they want to cook after. Additionally, with the huge amounts of culinary possibilities, one can easily get overwhelmed. One possibility to solve this could be to go to one's favorite recipe-website and to then look for a recipe. However, due to the sheer amount of recipes and due to very different tastes between different people, this is often not a plausible option for long term. Because of similar problems like the above-mentioned, personalized recommender systems [10] got an ever-increasing relevance in helping users searching through large amounts of information on certain topics in the last decades. Although several approaches of food recipe recommendation systems were build, many of those approaches used only small datasets for their model (see e.g. [12]). Here comes our approach in play. We wanted to build a personalized recipe recommender-system, which, given an input sequence of the last cooked recipes, recommends the next best recipe to cook. Additionally, we used a rather large dataset for our model, further described in the section Dataset. With the advancements in machine learning Machine Learning (ML), such a personalized recommender system becomes realizable.
Our transformer based model takes a sequence of previously cooked recipes as an input and put this sequence through an encoder framework to predict the next best recipe.

In the following section, we are first going to explain the Background behind our approach, and then we will explain our Model and the Results of our approach.

2 Background

In this section, we are going to look at the background concepts behind our approach to create a common ground.

2.1 Recommender Systems

Recommender systems are algorithms, which goal is to recommend users items relevant for them, based on knowledge about the user. A recommender system works with a history of user data to predict the relevant item(s). The first of such a recommender systems was published by Goldberg et al. [6]. They are able to assist humans in decision or choice based situations. A distinction between three bigger categories of recommender systems here is possible. The first ones are collaborative filtering based methods, the second ones are content based methods and the third knowledge based methods [11]. However, as each of these three approaches has its strengths and drawbacks, there exist also hybrid methods between these approaches.

2.1.1 Collaborative filtering based methods

The idea behind collaborative filtering based methods can be described as the model tries to find users in a group of users that share interests and tastes in order to predict the next item. A collaborative filtering based method takes items and ratings from users of these items as an input. The model then compares the given input of a user to the data of other users, in order to find the user which is the most similar to the current user.

2.1.2 Content based methods

The approach of content based methods focuses on a single user itself and compares the content of the previously rated items of a user to find similarities for the prediction. It works with profiles of users, where it stores information and the taste based on the given ratings of the user [2]. Additionally, it requires the content of the items to be well-structured. The models analyze details, like descriptions of items, in order to find interesting items for the current user [9]. Therefore, these models act as classifiers.

2.1.3 Knowledge based methods

It uses knowledge about the user and items in order to give a recommendation. The model requires active input of the requirements of the user to find a solution for the search [4].

2.1.4 Hybrid approaches

In order to get better results, to accumulate the advantages of the approaches and to avoid the limitations, often hybrid approaches are used. Such a hybrid approach can be utilized in different ways. These can be for example [1]:

- Using a unified system, that takes several approaches into account
- Separate, side by side implementations, where the results are then joint
- Using some rules of one approach in another approach

2.2 The new age of Recommender system: Transformer

Now you have read about traditional Recommender systems. However, nowadays, primarily transformers are being used for recommendation systems, as they are a lot more versatile, performant and produce better results. In this section, we will give a short introduction into the topic.
Transformer were introduced by Vaswani et al. [14]. One important addition of the transformer architecture in comparison to previous approaches is, that it uses Attention in its architecture. It typically consists of an Encoder and a Decoder (see Figure 3). The encoder typically maps the input sequence to a sequence of the continuous representations, while the decoder generates an output sequence based on the output of the encoder in combination with the output of the decoder at the time step from the previous iteration.

2.2.1 Attention

As mentioned before, one important aspect of The new age of Recommender system: Transformer is the concept of attention. Attention is a term typically found in the fields of cognitive psychology. There, attention can be understood as a mechanism that selectively enhances some stimuli and suppresses others. Inspired by this, attention is a concept also used in machine learning and artificial neural networks. One of the first models using attention was introduced by Bahdanau, Cho, and Bengio [3]. In that paper, the researchers were concerned with neural machine translation and used an attention mechanism in the translation model. Generally, artificial attention can be understood as "a form of iterative re-weighting" [7, p. 9]. Attention can be more formally described as "mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key" [14, p. 3], There are several types of artificial attention used in different transformers. Two of those are described below.

Scaled Dot-Product Attention

In Scaled Dot-Product Attention the dot products of the input get divided by the root of the key of the input dimension, which works as a scaling factor, followed by a softmax function [14, p. 4] (see Figure 1).

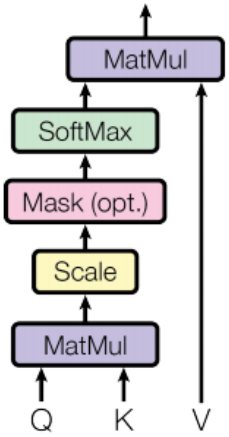

Figure 1: Scaled Dot-Product Attention [13]

Multi Head Attention

Multi-head attention runs in parallel several times through an attention mechanism. After this, the outputs of these runs are then concatenated and, using linear transformation, transformed into the expected dimension [16] (see Figure 2). Multi-head attention has the advantage that the algorithm can apply shorter or longer-term dependencies more variable to parts of the input sequence. It allows the model to look at different points in different representations and positions at the same time [14].

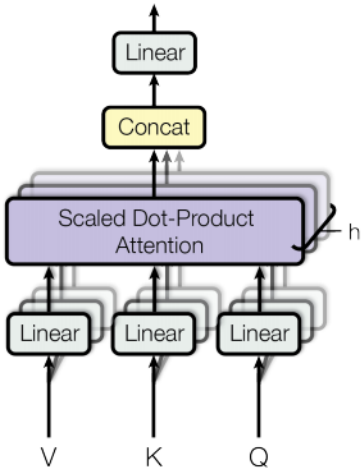

Figure 2: Multi Head Attention [13]

2.2.2 Behavior Sequence Transformer

A new type of transformer for recommender systems was introduced by Chen et al. [5]. This so-called Behavior Sequence Transformer Behavior Sequence Transformer (BST) typically consists of an embedding layer, a transformer layer and a Multi Layer Perceptron (MLP) at the end to generate an output [15]. In a

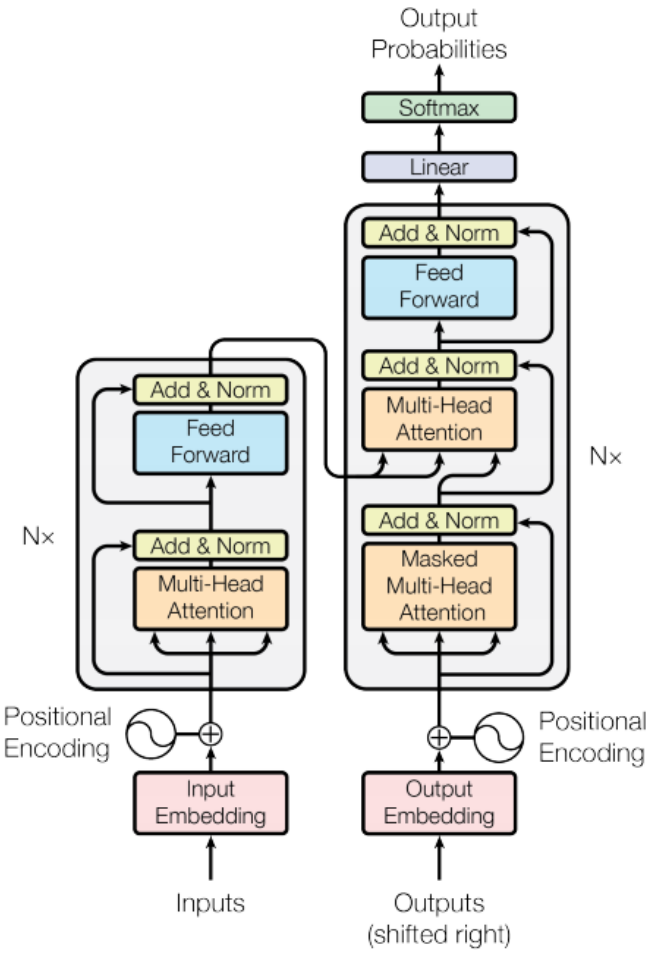

Figure 3: Transformer [13]

BST, only the encoder part of the transformer architecture, described in section 2.2, is used. Importantly, a BST uses self-attention to combine the inputs from the previous time steps. The embedding layer here reshapes the features of the input data into a vector that gets passed on to the transformer layer. Further, in the MLP layer, fully connected layers can be used to learn from the interactions between embeddings of possible non-primary features [15].

In addition to the item sequence, other features can be incorporated into the model. These features are, for example, user features such as age group or other contextual features. Other features are also embedded, but they are not passed into the transformer layer, instead they are concatenated with the output sequence of the transformer layer. [5]

The complete architecture of a BST is shown in Figure 4. This is the transformer architecture our approach is based on. See the section Model for further details.

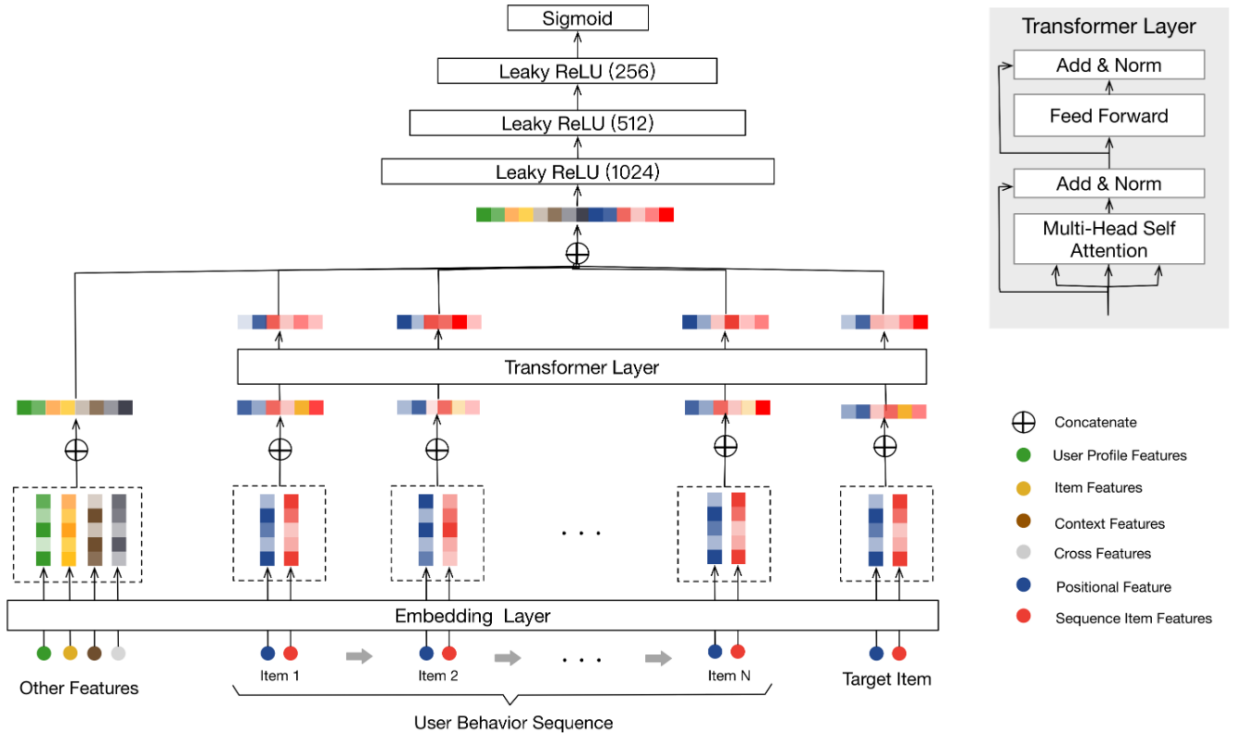

Figure 4: Behavior Sequence Transformer [5]

3 Dataset

We used the "Food.com Recipes and Interactions" which can be found at https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions for our approach. The data provided is a mixture of raw and pre-processed data. We used one of the pre-processed data set but mainly the raw data sets and did our own pre-processing. The two raw data sets contained the recipe data in one and the interaction data in another file.

This dataset contains approximately 180,000 recipes and over 700,000 reviews of these recipes. Further, the dataset contains recipes consisting of approximately 13,000 ingredients combined over all recipes. Each recipe has at least 4 ingredients and maximally 20 ingredients. This data was collected on the website https://www.food.com/ over the span of 18 years (2002 - 2018). The dataset was collected and used by

Majumder et al. [8].

index	recipe_id
0	41141
1	127816
2	100687
...	...
216799	134610
216800	148713

Table 1: An example of the (training-) dataset we used.

4 Model

Our model is based on the BST architecture introduced in the paper [5]. The Model takes a sequence of recipes as an input and returns a recipe recommendation. The recipes are represented by a continuous integer named recipe_id. The input sequence represents the nine most recent recipes that have been positively rated by a user in order of the rating. Because of the use of attention in Transformers, we had to choose a fixed sequence length for the input of the model. In our case, this length is nine. Our model architecture consists of multiple components that are described in the following sections.

4.1 Embedding

The model's input sequence is first passed through an embedding layer called `RecipeEmbedding`. This layer is responsible for the embedding of the single recipe ids and for the addition of a positional encoding. Recipe embedding is achieved by passing the sequence of recipe ids into a TensorFlow `Embedding` layer with an embedding size of 64. The positional encoding is realized by passing a range from 0 to the sequence length into an `Embedding` layer with the same embedding size as the recipe embedding. The output of the `RecipeEmbedding` is generated by an `Add` layer that adds the positional encoding to the recipe encoding.

4.2 Encoder and Transformer Layer

A core component of the model is the Transformer Layer. As shown by Chen et al. [5], the BST only uses the encoder part of a transformer architecture. Such an encoder consists of one or multiple stacked transformer layers. In this model, we use one transformer layer that performed best in [5]. Our transformer layer consists of several components. The first is a `MultiHeadAttention` with four attention heads. This attention layer is used for self-attention by passing the embedded sequence as the query as well as the value into the attention layer. The outputs of the `MultiHeadAttention` are passed through a `Dropout` layer. All `Dropout` layers in our model have a dropout rate of 0.2. Residual (skip) connections are used in the Transformer Layer to tackle vanishing and exploding gradients. A residual connection is established after the dropout layer. In this case, the output after the `dropout` layer is added by an `Add` layer to the input of the transformer layer. The resulting Values are then passed through a `LayerNormalization` with an epsilon of 1e-6. After normalization, the output is passed into two `Dense` layers with a leaky ReLU activation function. The first layer has a size of 256 and the second a size corresponding to the recipe embedding (64). After these layers, a second dropout and a residual connection is used, followed by a `LayerNormalization` to create the output of the transformer layer.

4.3 Output

The output section consists of multiple layers. At first, the sequential output from the transformer layer is flattened by a `Flatten` layer. Then the output is passed through three `Dense` layers with sizes 512, 256 and 128. Each of these layers has a leaky ReLU as an activation function. Finally, comes a `Dense` layer with a size corresponding to the number of recipes (178265). This output layer does not have an activation function. The default activation function for such a layer would be `softmax`. How to deal with linear outputs that should correspond to a probability distribution is explained in section 6.

4.4 Model size

Our Model has a total of 34,965,081 trainable parameters. Most Parameters come from the `RecipeEmbedding` layer and the output layer, see table 2.

layer	**parameters**
recipe embedding	11,409,536
encoder	99,712
dense 1	295,424
dense 2	131,328
dense 3	32,896
Output	22,996,185

Table 2: Components of the model with their amount of trainable parameters

5 Data Preprocessing

To get a good understanding of the data which originates from different source files, the data was inspected in a Jupyter notebook. Three different files were loaded as pandas dataframes and merge along the `recipe_id` which is present in all files. In the Jupyter notebook, we established a general datastream. The data set of RAW_interactions, RAW_recipes and PP_recipes were loaded and merged along the recipe-id. Our approach and earlier research base our work on the assumption that we can infer the next interaction from the past ones. Thus, the data from RAW_interactions is the baseline for us, since it includes the sequences in which a user interacted with the recipes.

It is suggested by the BST paper that additional features can be beneficial during training Chen et al. [5]. We would get ingredient ids from the PP_recipes file. The ids were already preprocessed by giving multiple instances of the same ingredient the same id. From the RAW_recipes file we got the number of ingredient, the number of steps and the minutes per recipe. Since each recipe has a (varying) number of ingredients, we used multi hot encoding. After having some major issues with the performance of our model, we decided to exclude the earlier mentioned additional feature and ended up training only on the recipe-id. We excluded every user which had made less than 10 reviews, since we regard this as the minimum sequence length our approach would be feasible on. To receive the sequence of interactions per user we sorted the data by user-id and date, these columns were dropped later on. Afterwards, we shuffled again while grouping on user-id. So in the end, we have our data randomized in regard to the order of the users, but we have every user ordered in itself. For prototyping, we implemented the option to take only a part of the data to continue from here. After the randomization, we split the data into training data and validation data. At this point, we stored the data into CSV files again. Saving to CSV and reading from it in the next step gave us better performance

than loading the data directly from the pandas dataframes. We used the generator to implement a sliding window. The generator yielded a sequence of nine interaction and giving the tenth interaction as the target. The sliding window goes through all interaction of the given user with a step-size of one. This means that our model possibly trains on some data points multiple times during one epoch. This allows us to get more information from one data point since it is in different places in multiple sequences. This helps us to represent sequences that are longer than our sliding window size.
We generated a TensorFlow dataset from the generator, this dataset was then fed into a preprocessing pipeline. Here, the dataset was batched with a batch size of 256 and then shuffled and prefetch.

6 Training

We trained the model for 20 epochs using the Adam optimizer and an initial learning rate of 0.001. A batch size of 256 was used for the training. The data set was split into validation and training data, with 2/3 as training and 1/3 as validation data. The training targets are provided as integer indices, so `SparseCategoricalCrossentropy` is used as the loss function. The output layer of the model uses a linear output instead of a softmax activation function. Therefore, the flag from_logits=True has to be activated.

7 Results

During training, the accuracy and loss were tracked for each epoch. The training loss decreased steadily over the 20 epoch, from 11.75 after the first down to 0.18 at the 20th epoch. In contrast, the validation loss increased steadily from 11.63 at the first epoch up to 43.43 at the 10th epoch. After the 10th epoch, the validation loss began to oscillate, with a maximum loss of 47.1 at the 12th epoch and 44.95 at the last. These results are visualized in Figure 5. The Training accuracy increased from 0% at the first epoch up to 95% at the 20th epoch, and the validation accuracy stagnated at 0% for all 20 epochs (see figure 6).

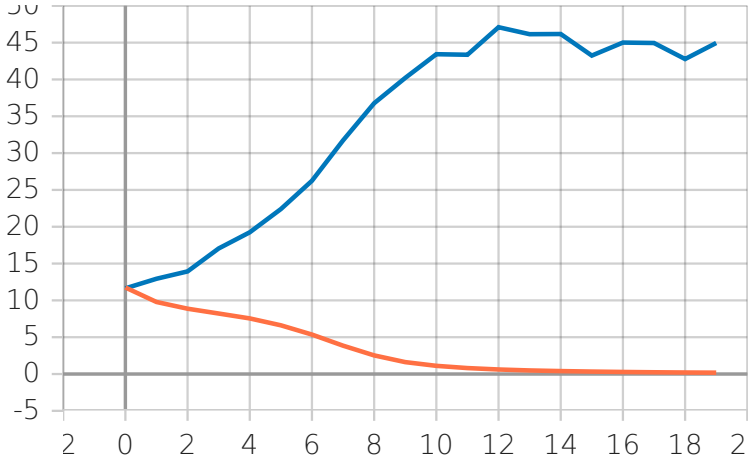

Figure 5: Training and validation loss across 20 epochs (validation data: blue, training data: orange, epochs on the x-axis, loss on the y-axis)

As shown, the training loss decreases during training, in contrast, the validation loss and training accuracy increase, while validation accuracy plateaus at 0. This indicates that the model only memorizes the training data while training, leading to overfitting.

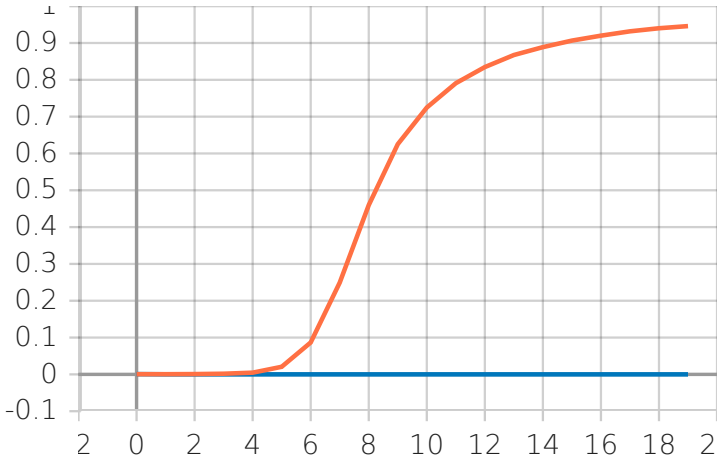

Figure 6: Training and validation accuracy across 20 epochs (validation data: blue, training data: orange, epochs on the x-axis, accuracy on the y-axis)

8 Conclusion

In this paper, we propose a food recipe recommender model, that predicts the next best recipe to cook for a user, given the previously cooked recipes. We accomplished that, using a Behavior Sequence Transformer. On a rather large dataset we showed that our BST-based model, using Multi-Head attention, can learn the structures of the users in the training dataset. However, our model overfitted on the training data, which resulted in lower performance on the validation data. As an outlook for the next steps, we could try to incorporate additional features of the dataset into the embeddings and apply several regularization methods in order to prevent our model from overfitting. Using more features would result in a significant increase in training time and was not feasible for this project with the hardware that we had access to. Nevertheless, our model is able to successfully learn and recommend, if not with the best performance, recipes to the user.

8.1 Limitations

As one could see in our data and our results and conclusion section, although our model learned the training data quite well, it overfitted quite a lot, resulting in poor performance on the validation data. To overcome the overfitting with this data for this task, we recommend using standard methods such as regularization methods. Furthermore, it might be beneficial to use a larger amount of additional features provided with the data to train on. However, this would result in slower training and a careful selection should be made.

Additional Material

The code for our model and the code to create the dataset are available at:
https://github.com/Erik-Nickel/iannwtfael.

References

[1] Gediminas Adomavicius and Alexander Tuzhilin. “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions”. In: *IEEE transactions on knowledge and data engineering* 17.6 (2005), pp. 734–749.

[2] Daniar Asanov et al. “Algorithms and methods in recommender systems”. In: *Berlin Institute of Technology, Berlin, Germany* (2011).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[4] Robin Burke. "Knowledge-based recommender systems". In: *Encyclopedia of library and information systems* 69.Supplement 32 (2000), pp. 175–186.

[5] Qiwei Chen et al. "Behavior sequence transformer for e-commerce recommendation in alibaba". In: *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 2019, pp. 1–4.

[6] David Goldberg et al. "Using collaborative filtering to weave an information tapestry". In: *Communications of the ACM* 35.12 (1992), pp. 61–70.

[7] Grace W Lindsay. "Attention in psychology, neuroscience, and machine learning". In: *Frontiers in computational neuroscience* (2020), p. 29.

[8] Bodhisattwa Prasad Majumder et al. "Generating personalized recipes from historical user preferences". In: *arXiv preprint arXiv:1909.00105* (2019).

[9] Michael J Pazzani and Daniel Billsus. "Content-based recommendation systems". In: *The adaptive web*. Springer, 2007, pp. 325–341.

[10] Al Mamunur Rashid et al. "Getting to know you: learning new user preferences in recommender systems". In: *Proceedings of the 7th international conference on Intelligent user interfaces*. 2002, pp. 127–134.

[11] Baptiste Rocca. *Introduction to recommender systems*. en. June 2019. URL: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada (visited on 04/04/2022).

[12] Chun-Yuen Teng, Yu-Ru Lin, and Lada A Adamic. "Recipe recommendation using ingredient networks". In: *Proceedings of the 4th annual ACM web science conference*. 2012, pp. 298–307.

[13] *Transformer model for language understanding*. URL: https://www.tensorflow.org/text/tutorials/transformer (visited on 04/01/2022).

[14] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[15] Subir Verma. *Behavior Sequence Transformer*. en. Oct. 2021. URL: https://medium.com/mlearning-ai/behavior-sequence-transformer-c9bf7308c463 (visited on 04/04/2022).

[16] Lilian Weng. *Attention? Attention!* en. June 2018. URL: https://lilianweng.github.io/posts/2018-06-24-attention/ (visited on 04/04/2022).