Next city prediction in multi-destinations trips

Clément Martin-Gougenheim
CentraleSupélec
ESCP Business school
February 20th 2021
clement.martin-g@student-cs.fr

Abstract

The goal of the study is to evaluate the best way to build a sequence prediction model, combining both token-level features and context-level features. The task is close to learning a language model, but with the addition of context features. We will therefore evaluate the performance of different types of RNN layers, as well as the best way to combine all of the features we have.

1. Introduction

In this study we focus on task of predicting the next city in a multi-destinations trip. This specific problem has been outlined by Booking.com and they decided to launch a campaign in order to solve it. Booking.com is an online platform and travel agency that helps travellers connect with hotels and transports. The website has over 28 million listings. Many of the travellers go on trips which include more than one destination. For instance, a user from the US could fly to Amsterdam for 5 nights, then spend 2 nights in Brussels, 3 in Paris and 1 in Amsterdam again before heading back home. In this scenario, they try to suggest options for extending the client's trip immediately when they make their booking

The goal of this challenge is to use a dataset based on millions of real anonymized accommodation reservations to come up with a strategy for making the best recommendation for their next destination in real-time.

2. Prior work

This challenge has numerous business applications, especially in the ecosystem of online platforms. Therefore, some solutions have already been proposed.

Booking.com have their own version currently active on their website. They explained this problem on two conferences, available on the web. Pavel Levin from Booking.com talked about Modeling Multi-Destination Trips with Recurrent Neural Networks at DataConf 2018, a casual data science conference by DataHack. A github is available on the slides presented [1].

The problem was then emphasized in 2019 at a PyData Meetup in Tel-Aviv on the subject of Modeling Multi-Destination Trips with RNNs, presented by Sarai Mizrachi. She insisted on the importance of considering and differenciating token-level features and sequence-level features. A video transcription of the conference is available online.

Furthermore, they jointly wrote a paper in 2019 on Combining Context Features in Sequence-Aware Recommender Systems [3], where they describe and evaluate the possibilities of such combinations in a neural network.

A kaggle competition was also organized on a similar topic: predicting the destination of a taxi based on the beginning of its trajectory, represented as a variable-length sequence of GPS points, and diverse associated meta-information, such as the departure time, the driver id and client information. This is indeed quite similar to the task we are presented with. The winners published an interesting article describing their approach using an MLP [4].

3. Data description

Booking.com gives us access to real data retrieved from their website in the year 2016. The training dataset consists of over a million of anonymized hotel reservations, based on real data, with the following features:

user_id - User ID

check-in - Reservation check-in date

checkout - Reservation check-out date

affiliate_id - An anonymized ID of affiliate channels where the booker came from (e.g. direct, some third party referrals, paid search engine, etc.)

device_class - desktop/mobile

booker_country - Country from which the reservation was made (anonymized)

hotel_country - Country of the hotel (anonymized)

city_id - city_id of the hotel's city (anonymized)

utrip_id - Unique identification of user's trip (a group of multi-destinations bookings within the same trip)

Each reservation is a part of a customer's trip (identified by utrip_id) which includes at least 4 consecutive reservations. There are 0 or more days between check-out and check-in dates of two consecutive reservations.

The evaluation dataset is constructed similarly, however the city_id of the final reservation of each trip is concealed and requires a prediction. There are over two hundred thousands different trips in total in the dataset and almost forty thousands different cities. We define N, the number of trips.

4. Problem approach

4.1. Problem characterization

The type of problem we have to solve is a classification task. We will try to predict a category (next city) as an output, with an input of trip information and a sequence of cities. We consider Y, containing all target destinations Y_i for each trip $i \in N$.

The metrics we used is the one proposed by the challenge: the top-4 accuracy. If the correct answer is in the top 4 propositions made by the model, then it is considered correct.

4.2. Data decisions

Modeling cities as token-level features

We choose to consider cities as the essential key element of the sequences of data we have for each trip. Therefore we will consider them separately from the rest of the information we gather from each reservation (time it was made, the country of the booker...).

Additionally, we decide to remove consecutive reservations in the same city in our training data. This is a rather unpredictable behaviour from the clients, and drives us away from the main objective of the study: finding logic in the travel destinations.

We consider X, containing all sequences of cities X_i for each trip i ∈ N, and $x_{i,j}$ the j^{th} city in trip i of length l_i:

$$X = \{X_i\}_{i \in N}$$

$$X_i = \{x_{i,j}\}_{0 \leq j < l_i}$$

Adding sequence-level information

On the other hand, we consider that the sequential data we have for each reservation doesn't have a sequential logic (The booker country is often the same and is not to be followed by a specific one, same goes for the type of device used...). For this very reason, we consider them as sequence-level features. They do not say something about the timestep from which they were retrieved, but really from the trip in general. This consideration leads us to gather the mode of the booker country for each trip and the month of the last reservation.

We consider X_info, containing all information vectors for each trip $i \in N$

$$X\_info = \{X\_info_i\}_{i \in N}$$

4.3. Layers choices

The logic behind the data that we have is clearly sequential. So we will consider Recurrent neural networks and their gated variants (GRU [5] and LSTM [6]). We are in a many-to-one system situation, where, from an input sequence X_i with timesteps $x_{i,j}$ we try to predict a single city $\hat{y}_i$.

4.3.1 Traditional recurrent Neural Network

Traditional RNN works in the following way. At each timestep t, a hidden state h_t is calculated, based on the previous hidden states and two different weights matrices. One weight matrix, noted W_{xh} affecting the timestep input $x_{i,t}$. And a second one, noted W_{hh} affecting the hidden state at timestep t: h_t.

Each hidden state is calculated the following way:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_{i,t} + b)$$

Where $tanh$ is the activation function, and b a bias.

It is interesting to point out that this is just like a perceptron, or a classical neural network, where $W_{xh}X_t$ is a weighted sum of features X_t and $tanh$ is just an activation function. Also, in this simple rnn, the same weights are shared for all steps of the sequence.

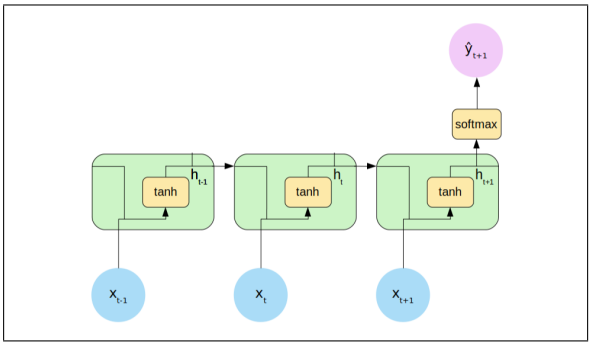

Figure 1. Example of such RNN with a softmax activation as the final activation function

Finally, the output is computed using a third weight matrix: W_{hy}:

$$\hat{y}_{t+1} = softmax(W_{hy}h_t + b)$$

4.3.2 Gated reccurents units

The problem with this classical RNN, is that it has troubles memorizing long term dependencies, and tends to "forget" information in hidden states calculated far before the current timestep. For this reason, another type of structure was created. In order to keep memory of older state, the first idea was to create an update gate usually noted Γ_u. This is a Gated Recurrent Unit (often called GRU)

The idea of this update gate is to have a variable that says how much information from the past step we should keep.

The structure would be the following:

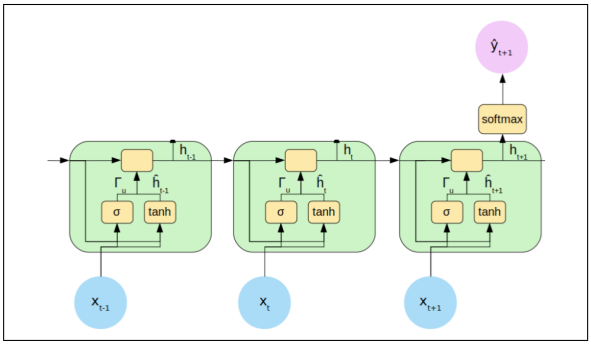

Figure 2. Example of such GRU with a softmax activation as the final activation function

A new set of weights matrices is introduced: W_{hu} and W_{xu} affecting the update gate Γ_u, and W_{hr} and W_{xr} affecting the relevance gate Γ_r.

As we can see on the figure, the update gate is computed using the $sigmoid$ function, using the same inputs as a regular RNN:

$$\Gamma_u = sigmoid(W_{hu}h_{t-1} + W_{xu}x_{i,t} + bu)$$

The relevance gate is also computed using the $sigmoid$ function:

$$\Gamma_r = sigmoid(W_{hr}h_{t-1} + W_{xr}x_{i,t} + br)$$

An intermediary state $\hat{h}_t$ is also calculated before the hidden state h_t

$$\hat{h}_t = tanh(W_{hu}h_{t-1} + W_{xu}x_{i,t} + bu)$$

$$h_t = \Gamma_u\hat{h}_t + (1 - \Gamma_u)h_{t-1}$$

This is why Γ_u can really be seen as a gate: if $\Gamma_u = 1$, the previous state is completely forgotten and if $\Gamma_u = 0$ the hiddent state is not updated, the previous hidden state is kept.

4.3.3 Long-Short term Memory cell

Historically, before creating GRU, Long Short-Term Memory (LSTM) cells were invented. They are even more complex than GRU. LSTM cells allow to keep information over longer sequences than GRU cells. The drawback is that they have much more parameters, and are thus harder to train: LSTM need a larger training dataset. A GRU cell can be seen a binary update gate: you can either keep all the past information, or rely only on current information. (Even though it is more a threshold that can keep half past and half current information.) A LSTM offers much more liberty: you can either keep all past information and all current information. The structure is the following:

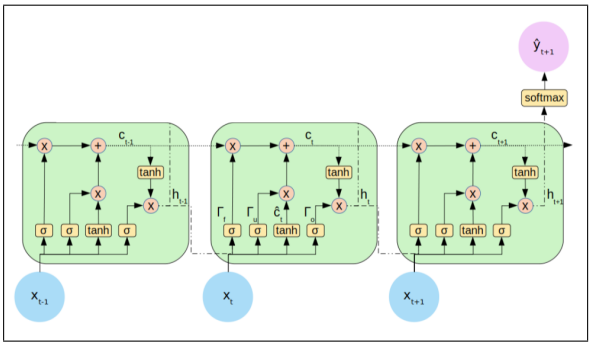

Figure 3. Example of such LSTM with a softmax activation as the final activation function

There is still the update gate Γ_u that keeps working the same. And there is now the forget gate Γ_f that is computed the same way with different weights matrices:

$$\Gamma_f = sigmoid(W_{hf}h_{t-1} + W_{xf}x_{i,t} + bf)$$

And we still compute an intermediary state, called now instead of before, but it is also a similar calculation:

$$\hat{c}_t = tanh(W_{hc}h_{t-1} + W_{xc}x_{i,t} + bc)$$

There is a new cell, the cell state c_t that is computed thanks to the update gate, the forget gate and the current state:

$$c_t = tanh(\Gamma_f c_{t-1} + \Gamma_u\hat{c}_t)$$

And there is one final gate computed like other gates, the output gate Γ_o:

$$\Gamma_o = sigmoid(W_{ho}h_{t-1} + W_{xo}x_{i,t} + bo)$$

This allows us to compute the hidden state $\hat{h}_t = \Gamma_o c_t$

4.4. Model architecture

The architecture we chose is the following:

- An Embedding layer with a dimension size 100, projecting the cities in a different space.
- A RNN layer (simpleRNN, GRU or LSTM), with 100 units as a many-to-many system.
- A second RNN layer (simpleRNN, GRU or LSTM), with 100 units as a many-to-one system.
- An output layer, using softmax activation function.

4.5. Training

We use an Adam optimizer, with a categorical crossentropy loss function. We use the bucketing technique with a custom data generator, loading batches of 256 shuffled same-sized trips. We trained our network over 50 epochs, on a GPU provided by Google colab.

5. Experiments

5.1. Data preprocessing

5.1.1 Encoding and feature engineering

As explained before, we decide to treat differently cities and trips information. Cities will be encoded in a learned embedding space by the first layer of our neural network.

We create additional information for each trip based on the data that we have (season of the reservation, country of the trip organizer, number of days stayed in the last hotel...). For non numerical data, we decide to apply simple label encoding.

5.1.2 Data enrichment

Since our data is sequential, we can create additional subtrips simply by decomposing trips of length n into subtrips of length $3 \leq l \leq n-1$. We decide to choose 3 as the minimum length of our created subtrips, since we consider that anything below that is not something that is modelable by our neural network.

And we also notice that we can retrieve additional subtrips from our validation and test data, without any data leakage. Indeed, using the same idea (and without touching the target Y), we can enrich our data set. Using this method, we have a new number N of trips to consider.

5.2. Experimental protocol

The protocol we used for our experiments is the following:

1. Randomly split $D_1 = [X_i, Y_i]_{i \in N}$ into Train, Validation and Test data (respectively T, V and Te)
2. Train three different neural networks (N_1, N_2 and N_3 respectively with simpleRNN, GRU or LSTM layer) on T
3. Evaluate them on V and choose the best performing one
4. Retrain and adjust hyperparameters using T and V
5. Evaluate generalization performance using Te
6. Split $D_2 = [X_i, X\_info_i, Y_i]_{i \in N}$ into Train, Validation and Test data (respectively T_2, V_2 and Te) using same randomness (to obtain same Te)
7. Train our best model on T_2 and evaluate on V_2
8. Evaluate generalization performance using Te
9. Compare the results and conclude on the best approach

5.3. First results: The best type of layer for the task

After performing steps 1, 2 and 3, we obtain the following performances on train and validation:

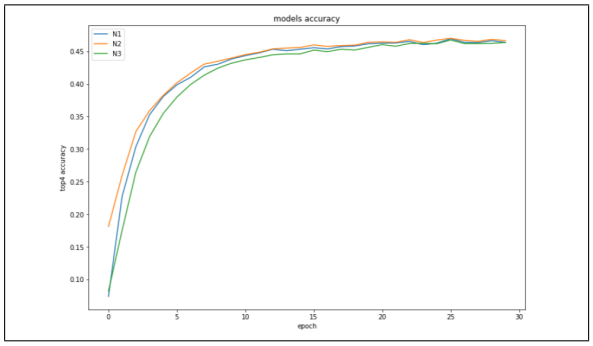

Figure 4. Top4 accuracy scores on validation set V for our 3 baselines neural networks over 30 epochs

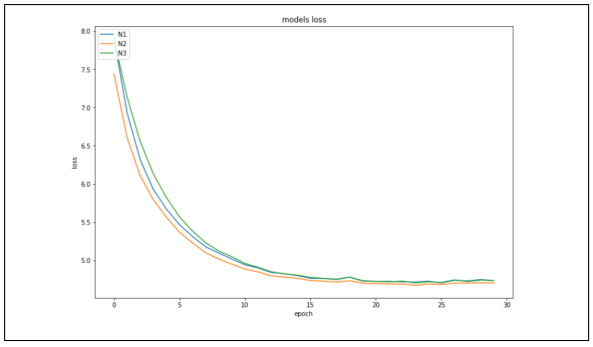

Figure 5. Loss scores on validation set V for our 3 baselines neural networks over 30 epochs

Our first three models seem to perform similarly. The N_2, incorporating GRU layers does have the lowest cross

entropy after 30 epochs on the validation set V, and also achieved the best top4 accuracy score with 46.76% on V.

Following our experimental protocol, we now keep N_2 as our main model. After adjusting hyperparameters on V, we chose an embedding dimension of 300 and 100 units (memory cells) for each of our first two GRU layers.

Testing our configuration for our neural network on Te, we obtain a top4 accuracy of 47.31%. Our model does indeed generalize well, since the performance on the validation set is comparable to the performance on the test set.

5.4. Adding trip information vector

We now obtained a model that generalizes reasonably well. To get better performances, we will now try to combine our sequence-level features (trip information) with our token-level features (cities at each timestep).

We add a concatenation layer to our model N_2', that add nodes (with a size equal to the vector of trip information) next to the output of our second GRU layer. This should help our model classify by giving a little more context to each trip. Let's see how it performs. After training it on V_2, we obtain the following results (Figure 6).

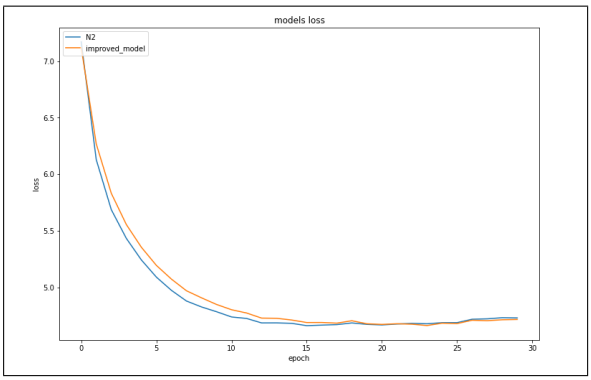

Figure 6. Loss comparison between N2 and N2' over 30 epochs

Adding the information vector for each trip to our model input as a sequence-level information seemed to improve our performance. We achieved a significant lower cross entropy, meaning our model successfully predicts better results on our validation set.

And indeed, when we evaluate their respective generalization performances on Te, N_2' seems to get better results (Table 1).

Model	Top4 accuracy score on Te
N_2	47.31
N_2'	48.28

Table 1. Results comparison. Our second adapted model performs better on Te

We will now present the final model F, greatly inspired by N_2' architecture, but trained a little bit differently to maximize our results.

6. Final Model

6.1. Adding an attention layer

After some research, it appears that the attention mechanism is a very effective method in sequence modelling that allows the model to focus on some specific timesteps when perfoming a prediction task. We believe this method can have an interesting impact on our modelization.

When the model tries to predict which is the next target cities in the sequence, it computes an attention score. After computing a score for each input token, we take a softmax and obtain a probability distribution (Figure 7).

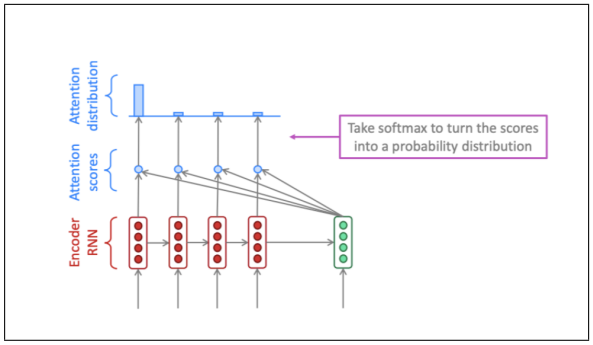

Figure 7. The attention mechanism

We then use this probablity distribution from the softmax to compute a weighted sum of the encoder hidden states and obtain the Attention output for the output timestamp t with the following formula:

$$a_t = \sum_i \alpha_i^t h_i$$

where α_i^t is the attention score for the output timestamp t and input timestamp i.

This should emphazise the most important cities to consider within a trip, when predicting the next city destination.

6.2. Model architecture

The architecture we chose is the following:

- An Embedding layer with a dimension size 300, projecting the cities in a different space.
- A GRU layer, with 100 units as a many-to-many system.
- A second GRU layer, with 100 units as a many-to-many system.
- An self made Attention mechanism.

- A concatenation layer, adding the trip information input to the Attention output.
- An output layer, using softmax activation function.

6.3. Data generator

As already mentioned, we created a custom data generator, inspired by the bucketing technique. We also add, inside the data generator, a specific function that randomly skip timesteps within the trip, depending on its length.

First, the data is loaded, then it is shuffled. Then, if it is in train mode, we randomly skip timesteps in a certain percentage of the trips. We then group the trips by their length. And we finally feed a batch at a time to the model.

6.4. Results

6.4.1 Interpreting metrics

Our new model F performs better than before. The training graphs can be seen on Figure 8 and Figure 9.

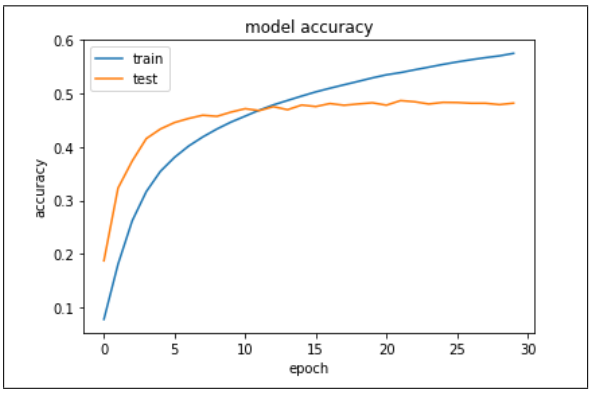

Figure 8. F top4 accuracy scores on train and validation set over 30 epochs

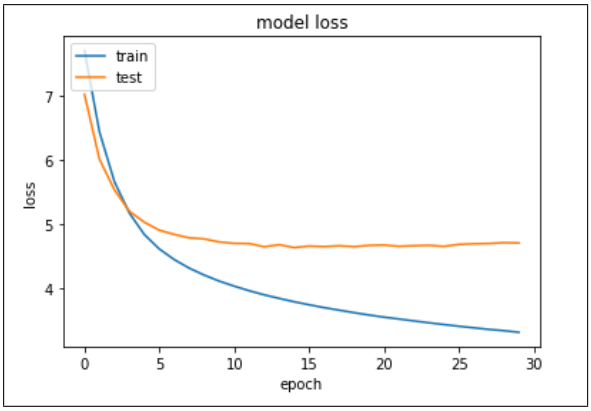

Figure 9. F cross entropy on train and validation set over 30 epochs

At around 30 epochs, we can see that the model slowly starts to overfit. The loss is still decreasing on the validation set, but it starts to increase for our train set. Using early stopping techniques, we retrieve the best version of F on this validation data.

After evaluating it on our test set, we obtain the following score, visible on Table 2.

Model	Top4 accuracy score on Te
N_2	47.31
N_2'	48.28
F	49.14

Table 2. Results comparison. Our final model performs better on Te

F is indeed the model that seem to generalize best.

6.4.2 Embedding analysis

Another interesting analysis we can make, is in regards of the embeddings of the cities. Our model learned to represent cities in a 300 dimension space, through weights optimization and gradient descent. Another way to evaluate our model is to try and see how they are represented in this space.

After using a PCA to reduce the 300 dimension to 2, we decide to plot some of the cities. Each point represents a city.

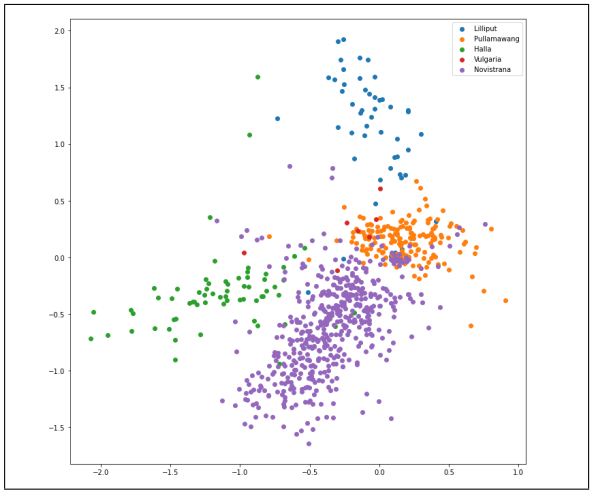

Figure 10. Projection of learned embeddings in a two dimensional space for five random cities

Because the cities are anonymized, we cannot really interpret the results. But our model does seem to have learned to classify each city to its country thanks to the embeddings. Indeed, projected in a $2-dimensional$ space, the countries are almost linearly separable.

7. Conclusion

The results of the study shows the importance of considering the right level of features for a sequence prediction task. Classification task with a very high cardinality (almost forty thousands classes) is possible with the right approach.

The F model we built was submitted, and evaluated on real data by Booking.com. It was ranked $11th$ out of more than 800 participants. Their papers have not yet been submitted, but surely they are worth to look at.

An other approach we didn't consider is the use of autoencoders for modeling cities. Using a latent space, cities can be reduced by getting a lower dimension representation of the features in an encoder. Then a decoder is trained to reconstruct the original data.

Another idea is to use a specific custom loss function (instead of simple cross entropy) to optimize our model. An interesting paper was published regarding this topic [8]. The idea is to take into account the weights of the embeddings inside the loss function. Indeed, most of the learning essentially happens there, when the model learns to differenciate each city. Learning to represent each of the forty thousands city in a $300 - dimensional$ space is a very similar task to learning a words in a language.

References

[1] Pavel Levin and Aviv Rotman *Data Conference Booking.com*. https://github.com/DataHackIL/DataConf/blob/master/DataConf_2018/DataConf_2018_Booking_Pavel_Levin.pdf, 2018.

[2] Sarai Mizrachi. *PyData Conference - Modelling Multi-destinations trips* https://www.youtube.com/watch?v=NtvxDFGbdPQ, 2019.

[3] Pavel Levin and Sarai Mizrachi *Combining Context Features in Sequence-Aware Recommender Systems*, 2019.

[4] Alexandre de Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. *Artificial Neural Networks Applied to Taxi Destination Prediction*, 2015.

[5] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*, In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Doha, Qatar, 1724–1734. https://doi.org/10.3115/v1/D14-1179

[6] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*, 1997. https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. *Attention is all you need*, 2017. https://arxiv.org/pdf/1706.03762.pdf

[8] Hakan Inan, Khashayar Khosravi and Richard Socher. *Tying word vectors and word classifiers: a loss framework for language modelling*, 2017. https://openreview.net/pdf?id=r1aPbsFle