Vehicle Trajectory Forecasting: A classification approach

TDDE19 Advanced Project Course - AI and Machine Learning

January 9, 2021

Contents

1 Introduction

During the last decade, a lot of research has been made in support of the development of self driving cars. In traffic environments, multiple different actors continuously interact with each other. As the development of self driving cars continues to progress, the need for models that can forecast these interactions increase. There are many insights needed to support the progression of self driving cars and one is the ability to forecast different actors' trajectories in traffic environments. Trajectory forecasting is the process of predicting the future location of an object based on historical locations. This can be applied in processes such as pedestrian prediction and traffic prediction which is especially useful for applications such as self driving cars.

This paper evaluates the state of the art Transformer Network(TN) in trajectory forecasting instead of it's traditional application in Natural language processing (NLP) tasks. We further evaluate this model by comparing the results to a LSTM (long short term memory) network. Transformers introduce the concept of attention which plays an essential role in the performance when compared to LSTM.

1.1 Background

Trajectory forecasting is a research area which has been thoroughly researched throughout the years. The research can be applied to many application areas, where one is in different traffic scenarios. Some of the research has been performed to forecast the trajectory of pedestrians while other work investigates the possibility of predicting motion patterns of vehicles.

In the paper by Alahi et al [1] the authors propose a model for predicting human trajectories in crowded spaces using LSTM networks. In their work they address the problem of agents not taking into consideration interactions between humans in crowded environments by introducing an "social" pooling layer. Each human in a scene is represented by an individual LSTM network. The hidden layers of the different LSTM networks are shared between closely related networks, i.e networks representing humans which are close to each other. The sharing of the hidden layers is performed through the social pooling layer.

Within the area of vehicle trajectory prediction, some work is based on data from the drivers point of view, while other work is based on predicting trajectories from a top down point of view. Yichuan et al[6] propose in their work a probabilistic framework where latent variables are learned, without the need of explicitly labeling the variables. The framework uses multiple RNNs, one for each agent in a scene. All the RNNs learn a latent variable each, and share their weights with each other.

In this paper the focus will be on the task of forecasting vehicle trajectories in urban traffic environments using the TN. As mentioned earlier, the TN is traditionally used for sequence modelling in NLP tasks. However, in the paper by Francesco et al [3], a new application area of forecasting pedestrian interactions and movements is proposed. The authors compare the TN with a Gaussian LSTM and achieve state of the art results.

2 Method

In this section, we will first discuss the different architectures of the neural networks that we use in this study. The purpose of this overview is to highlight the fundamental features that separate the methods we implement and ultimately decide which one is appropriate for our problem. The following sections presents the methodology used when evaluating the models. Firstly, the dataset used to compare the models is described; including its characteristics and properties. Secondly, an overview of how the models were implemented is described. Lastly, the evaluation metrics are presented.

2.1 Recurrent Neural Networks

What mainly separates the RNN from a feed-forward network is that it contains loops, disqualifying it as a directed acyclic graph by definition. However, note that they can be represented in a purely feed-forward manner, although we don't address that in this text and define them as strictly recurrent. The purpose of these loops stems from the sequential data that recurrent networks are specialized for processing. Essentially, the task is to learn shared meaning in contrast to a feed-forward network that treats the input data independently.

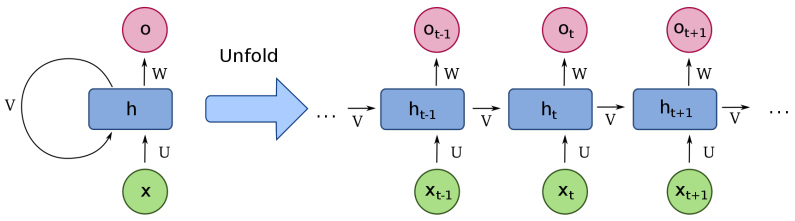

Figure 1: RNN and unfolded RNN

Figure 1 demonstrates how the context of past data affects the output of current and future outputs. On the left part of the figure, the recurrence is represented in the hidden state with the arrow pointing to itself. To further illustrate how past input affects future outputs the network is presented unfolded. In this form, we can clearly see that the output produced at any point t, will take all previous states into account when performing the computation. This mechanism of passing forward the internal computed state works as a memory for the RNN and, ultimately, allows for previous data to play a role in the output of the following data. However, although the RNN memory is unlimited in theory, in practice, as the length of the sequence grows, past inputs affect on current outputs begin to fade. This issue is known as vanishing gradients and is described well by Sepp Hochreiter [5]. The problem of vanishing gradients arises when training the network and is a consequence of backpropagation. The network calculates the gradient of the current layer with respect to previous layers, and as the gradient gets smaller in previous layers, the current gradient will be even less. This will ultimately lead to the gradient of early elements of the sequence becoming minuscule, causing the change to be almost zero, signifying that it will fail to learn. This inadequate short-term memory of the RNN is what gives rise to our next topic, a flavor of the RNN called long short-term memory (LSTM).

2.2 LSTM

The LSTM is a response to the inherent short-term memory of the RNN. The internal structure of the LSTM cell functions to separate valuable information in the sequence from irrelevant information, in the sense that the latter will not contribute significantly to the decision process of the output. Intuitively, the LSTM addresses this issue by keeping gates in its cells. These gates learn during training what to remember and what to forget, allowing early elements of a given input sequence to play a role in calculations at later stages.

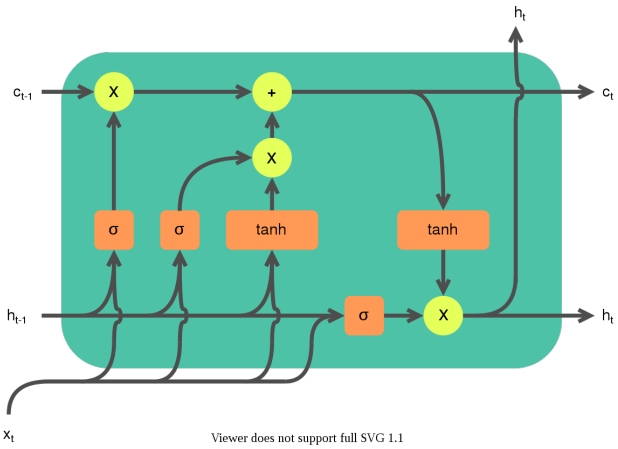

Figure 2: LSTM cell

The structure of the LSTM cell is illustrated in figure 2, with lines separating the different gates. Moving from left to right we start with the forget gate. As the name suggests, the network decides given the input sequence and the previous hidden state what to forget. The sigmoid function provides the functionality of this procedure, firstly, it is defined between [0,1], setting the importance of the value respectively. A value close to 0 is considered not important and can be forgotten, whilst values closer to 1 are important. Next, we move on to the input gate, whose task is to update the cell state. Essentially it will pass the input sequence and the previous hidden state as the forget gate, however, this is to determine what values are important to be updated in the next step. The same input combination will be passed through a tanh function, to regulate the network, this output will finally be multiplied with the input passed through the sigmoid function, and finally, a cell state is computed. The final gate, the output gate, forwards the cell state to the next cell together with the new hidden state. The hidden state is calculated by first passing the input sequence and previous hidden state through a sigmoid function and multiplying that with the current cell state that has been passed through a tanh function.

Together with the increased memory the LSTM provides comes increased computation [4]. Its worth noting that unless there is a demand for the LSTM as in long sequences, the RNN should be considered in its original state.

2.3 Transformers

The transformer is a neural network developed at google and published in 2018 [7]. The transformer is simple in its architecture, in terms of complexity, when compared what at the time was the state of the art, the RNNs, and convolutional neural networks (CNNs). What mainly separates it from the previous memory oriented networks, is that it utilizes an attention mechanism to learn the relationships between the elements

in the input embeddings. Additionally, it is not entirely dependent on sequential input throughout the network, given the feed-forward neural networks. This allows for parallelization during training which significantly reduces running time.

2.3.1 Attention

The transformer network uses an encoder-decoder structure. The attention layer is embedded in both blocks. Intuitively, the network captures important contextual information for each word and its relation to each other and encodes the information in a vector. From the input vector, the network will apply the attention function (1), which takes three arguments, the query (Q), key (K), and value (V). These arguments are computed by taking the input vector and taking its dot product with the respective matrix for each argument. Their respective matrices are all randomly initialized by the network and modified during training from the data. Additionally, the network uses multi-headed attention, with a default value of six attention heads, essentially this allows for the network to place attention on multiple parts of the input sequence.

$$Attention(Q, K, V) = softmax(QK^T/\sqrt{d_k})V \tag{1}$$

2.3.2 Parallelization

One of the big advantages the transformer network has over other models is the feed-forward neural networks that reside in the layers. Since the input of the feed-forward network is entirely independent of each other, These networks allow for parallelization, increasing computational speed. However, how can we take sequential information, that is dependent on each other, and process it independently? The answer lies in equations (2) and (3). In the first step of the Transformer network, all vectors are added a time vector, that will encode its position into its embedding space, in relation to all other vectors in the input sequence.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{2}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{3}$$

2.4 Data

The decision to look at the transformer model for trajectory forecasting was based on a paper [3], which applied the model on pedestrian movement data. In this project the task was looked at from another viewpoint, where the task now instead was to predict vehicle movement in urban environments.

Figure 3: Vehicle data mapped to an aerial map. The red lines corresponds to the training data, and the blue line corresponds to the testing data.

The dataset chosen for this project was a dataset based on collected GPS coordinates of a vehicle moving around in an urban city environment in the city of Sidney in Australia[1]. The dataset contains different traffic settings such as turns and roundabouts, which can be seen in figure 3. The data was split into a training set and a testing set. In figure 3, the red line is the training data which corresponds to 70% of the dataset. The blue line is the testing data, which corresponds to 30% of the dataset.

The task of predicting trajectories was modelled as a classification problem. The original format of the dataset was in x and y coordinates. Since the transformer network is a sequence modelling algorithm, the data needed to be transformed into another format. The new way to format the dataset was to split up all the data into unique, one dimensional bins. The dataset was first split up into multiple smaller driving sessions, to form a trajectory. These trajectories were transformed into relative coordinates, where the first coordinate in the trajectory starts from the origin. The relative coordinates are then mapped into the bins. The mapping to bins was performed by first scaling all the x and y relative coordinates to a value between 0 and *max*, where *max* is the square root of the total number of bins. The scaling was performed using a min-max scalar. For example, to scale the x-axis the function in (4) was used. Note, in this setting the min(x) will always be equal to zero.

$$x_{scaled} = (x - min(x))/(max - min(x)) \tag{4}$$

The transformation from the relative coordinates to bins was performed using the following function:

$$bin_i = y * max + x \tag{5}$$

where bin_i is the index of the bin to map the coordinates to. The x and y values were rounded to the nearest integer.

An example of the mapping can be seen in figure 4. The example is of an fictive trajectory which is mapped into bins, shown as black squares where $max = 10$ and total number of bins = 100. To make the model more general, the data was diversified by randomly rotating some of the trajectories.

[1] http://its.acfr.usyd.edu.au/datasets/naturalistic-intersection-driving-dataset/

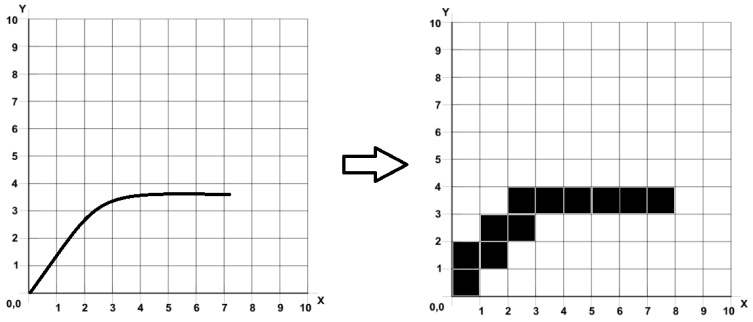

Figure 4: Fictive example of trajectories mapped into bins

2.5 Model Input and Output

For each vehicle trajectory, the models take the first N bin positions as input and outputs M predicted future bin positions. These bin positions are represented by their bin index. We test using two different dimensions of bins; namely (30 x 30) bins and (100 x 100) bins, where a bin in the former dimension configuration spans approximately 0.5 meters (both in longitude and latitude directions) and a bin in the latter configuration spans 0.15 meters. These configurations are adopted in order to capture sufficiently small changes in position. Using this setup, the vehicle trajectory problem is therefore modeled as a sequential multi-class classification problem with N_{bins} classes, where each class corresponds to a bin position and a class is predicted at each time step.

2.6 Transformer

The Transformer implementation uses 6 layers and 8 attention heads, with a dimensionality of 256. These parameters are chosen to expand the model's ability to focus on different positions and relationships in the sequences. Because of time constrains and the heavy computing power needed to explore all possible parameter combinations, we did not have the opportunity to optimize these parameters. However, the original paper that introduces transformers [7] also use 6 layers and 8 attention heads. The output of the final linear layer in the transformer network acts as a classifier with an output size as large as the number of possible bins. The output of the classifier then goes through a softmax layer, which produces probability scores for each bin. Therefore, the cross-entropy loss is used for the loss metric. Furthermore, Stochastic Gradient Descent is used together with a decaying learning rate and a dropout value of 0.2. The Transformer model was implemented in *Python 3.6* using *Pytorch*.

2.7 LSTM

The LSTM model used was implemented in Tensorflow using the Keras API [2] as an encoder-decoder architecture. The input sequence of bins was first embedded into a dimensionality of 256 to match the transformer. These embedding vectors were passed into the encoder containing 256 LSTM cells. The encoded vector is used as input into the decoder which also contains 256 LSTM cells. Because this implementation is modeled as a classification problem, the decoded result is passed through a dense layer with as many units as there are bins. This creates a vector of probabilities over the bin space using a softmax activation function.

To train the LSTM networks stochastic gradient descent with momentum was used. A dropout value of 0.20 was used in all the LSTM layers and categorical cross entropy was used as the loss function.

2.8 Experiment Setup

The models were trained and evaluated on four dataset configurations using three different combinations of input and output length: *(30, 70)*, *(100, 100)* and *(11, 22)*, as well as two configurations of N_{bins}: *900*, *10000* as shown in figure 6. The input length denotes the number of bin positions that the models take as input, and the output length denotes the length of the prediction horizon.

Datasets			
Dataset	Input sequence length	Output sequence length	Number of bins
1	30	70	10,000
2	30	70	900
3	100	100	900
4	11	22	900

Figure 5: Dataset configurations

These input and output lengths are chosen to test the models' performance on both long and short sequences. The different configurations of N_{bins} are chosen to examine how the models' performance change when increasing the number of possible classes it can predict. The dataset that has an input length of 11 and output length of 22, is slightly different from the other datasets; only every third coordinate in the trajectory have been sampled. Therefore, there are greater distances between each bin position.

The models are compared using four different evaluation metrics: average displacement error, final displacement error, bin accuracy, and percentage of perfect predictions.

- Average Displacement Error (ADE): The average of the root mean squared error (in meters) between the ground truth and the predicted trajectory position at every time step.
- Final Displacement Error (FDE): The root mean squared error (in meters) between the ground truth and the predicted trajectory position at the last time step.
- Bin Accuracy Percentage (BAP): The percentage of correct predicted bin indices.
- Percentage of Perfect Predictions (PPP): The percentage of trajectories where all the predicted bin indices are correct.

The ADE and FDE evaluation metrics are often used when performing trajectory prediction. To calculate the displacement errors, the bin indices were first converted back into their relative coordinates. These relative coordinates were then converted into UTM coordinates that were used to calculate the error in meters rather than some arbitrary distance. This also gives a more understandable metric when trying to understand how well the model predicted the trajectory of the vehicle. However, information is lost when converting real coordinates to bin positions and then, in turn, converting bin positions back to real coordinates. Furthermore, a bin position

covers several real coordinates. Therefore, the *BAP* and *PPP* metric was introduced to give more context to the displacement errors.

3 Results

In this section the main results are presented. The most relevant experiments are presented in a table followed by a visualization of trajectories that are shown on a satellite image.

3.1 Comparison Results

Figure 6 shows the experimental results. In this table the colors represent different datasets that the LSTM and transformer model were evaluated on. The train and validation loss is reported as the log-loss from categorical cross entropy. The rest of the columns are explained above in section 3.5.

Datasets								
Dataset	#Bins	Model	Train loss	Val loss	ADE (m)	FDE (m)	BAP (%)	PPP (%)
1	10,000	TF	1.06	1.01	0.13	0.32	68.3	13.8
2	900	TF	0.12	0.09	0.02	0.07	98.7	88.2
3	900	TF	0.17	0.48	0.13	0.54	91.6	43.9
4	900	TF	0.16	0.13	0.01	0.02	99.4	95.3
1	10,000	LSTM	1.75	2.20	1.28	1.93	2.4	~ 0
2	900	LSTM	0.81	1.18	0.59	0.99	44.4	3.1
3	900	LSTM	1.12	2.08	1.85	2.56	33.1	7.5
4	900	LSTM	0.91	1.32	0.65	0.92	38.6	2.3

Figure 6: Experiment Results

The transformer model outperformed our LSTM model in every single metric that we tested. LSTM performed best using dataset 2 with 900 bins resulting in a bin accuracy of 44 percent. An interesting result is that increasing our bin count to 10 000 reduced the LSTM's bin accuracy to about 2 percent for the same dataset. On the other hand, when the same bin increase was tested on the transformer, it still performed well and got around 68 percent bin accuracy. Our findings show that the transformer model can learn and predict on this dataset very well with accuracy above 91 percent on all the experiments using 900 bins. LSTM struggled to learn long sequences and started to overfit the training data which lead to early stopping at high loss values and resulted in poor performance when compared to the transformer.

3.2 Visualization of Trajectories

Figure 7 shows a satellite image over a city road in Sydney, Australia and contains predictions from the LSTM and transformer model. The sequences are created by converting the predicted bins back into relative coordinates, then the relative coordinates into latitude/longitude. Another important detail is that this visualization is a concatenation of many individual predictions from the models, each prediction is about ten dots on on the map. This result clearly shows that the LSTM performs poorly in the turns of the vehicle path, whereas the transformer performed almost

perfectly. These results were consistent in the majority of intersections throughout the test data.

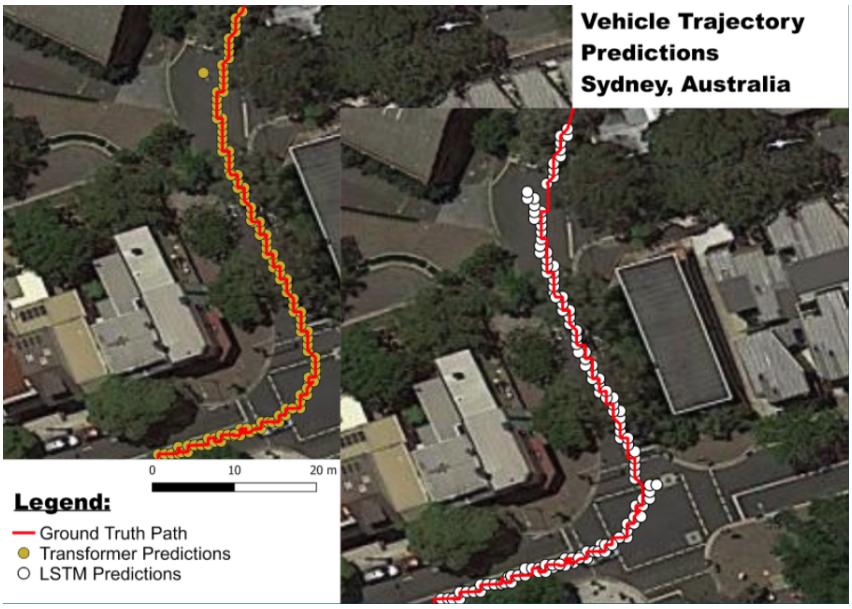

Figure 7: Satellite image of real world coordinate predictions

4 Discussion

In this section, the results will be discussed together with potential sources of errors in conjunction with the dataset and the implemented models. Then further improvements and additions will be discussed in future work.

4.1 Results and Differences

In this study, we have presented a method to predict vehicle trajectories using a multi-class classification approach. Instead of using cardinal coordinates as input, the models operate on bin indices, where each bin index contains a range of relative coordinates from the origin of the trajectory. We compared and evaluated two encoder-decoder models using a dataset consisting of urban driving. We varied the length of the model input length and prediction horizon, as well as the number of bins. The results show that the transformer network outperforms the LSTM model in all evaluation metrics.

The results that were obtained in our experiments align with the paper *Transformer Networks for Trajectory Forecasting* by Francesco Giuliari et al [3]. They found that their vanilla LSTM model often predicted straight lines. Meanwhile, the transformer could learn complex trajectories thanks to its ability to process longer sequences compared to the memory of an LSTM model. Our results were similar in the sense that in most trajectories that included 90-degree turns (most intersections) the LSTM would fail to predict the curve, whereas the transformer almost always could.

An important distinction between their approach and ours is the domain of the data. Their models were trained on pedestrian data taken over a large area where pedestrians can move freely. Our models used solely vehicle data.

In addition to the domain of the dataset, the collected data was also limiting to a certain degree. Even though the dataset contained some curves and roundabouts, there where still a predominant share of straight line trajectories within the dataset. As vehicle data is constrained to roads this inherently creates biased data. We introduced data augmentation, where some trajectories where randomly rotated to help address this problem. Adding more diverse data taken from different sources would increase the models ability to generalize and most likely increase prediction accuracy.

Another point to keep in mind is that the data was transformed to relative coordinates. By transforming the data to relative coordinates, the models was forced to learn the patterns of a trajectory curve rather then learning patterns at specific coordinates. Learning general patterns rather than context specific patterns should make the models more robust towards unseen information. But whether or not it is desirable to learn general patterns rather then position specific patterns depends on the problem attempted to solve.

The data was also transformed to fit into a one dimensional space, which gave rise to a trade off between data granularity and problem complexity. When putting multiple coordinates into the same bin, some information is lost during the transformation. How much information that is lost depends on the number of bins that is chosen for the model. As the number of bins increase, the data granularity increases as well as the problem complexity. Since the task was to classify coordinates to bins, the possible output space grows with the number of bins which give rise to the increase in problem complexity.

It is a balancing act when choosing the number of bins to use. A low amount of bins will increase the displacement error, since there is an direct displacement error connected to placing an coordinate into an bin. An large bin, which holds many coordinates will give rise to an larger displacement error inherently. There is also the case of when the model makes an bad prediction. With fewer bins, the distance between the centroid of each bin would increase, which in turn would increase the displacement error as well. We can therefore see that the bin prediction accuracy describes some parts of the displacement error.

It is also problematic to have too many bins. When comparing the results in figure 6, we can see that the model performance decreased while using 10 000 bins compared to 900 bins. This could be due to an increase of the problem complexity, where the output space got larger then what was needed for the problem.

4.2 Future Work

The LSTM's performance is an area that was not fully explored in this paper. The potential to create more complex LSTM models could lessen the gap in performance to the transformer. There exists many LSTM architectures that could have potentially performed differently when compared to our encoder-decoder implementation. In our case, the goal of this paper was not to find the best performing LSTM network and therefore this could be expanded on in future work.

Another aspect that can be further researched is the difference between the classification approach taken in this paper, versus a regression approach. A regression implementation would differ by instead of using one dimensional bins as the model input, X and Y coordinates could be used as two dimensional input. With such a change the problems regarding dimensionality of the bin space will no longer be an issue. This would remove the need of a final dense layer to produce a vector of probabilities over the vector space and instead the models could predict raw coordinates instead. Furthermore, using multivariate data from several signals, such as speed and

acceleration information of the vehicle could potentially increase the accuracy.

A final extension of the work could be to apply the transformer model to higher dimensional data, such as data collected from the drivers point of view. Whether or not it would be feasible to use the same data transformation methods as presented in this paper is something that could be interesting to look further into as well.

References

[1] Alexandre Alahi et al. "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971.

[2] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[3] Francesco Giuliari et al. *Transformer Networks for Trajectory Forecasting*. 2020. arXiv: `2003.08111 [cs.CV]`.

[4] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[5] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[6] Charlie Tang and Russ R Salakhutdinov. "Multiple futures prediction". In: *Advances in Neural Information Processing Systems*. 2019, pp. 15424–15434.

[7] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.