Advanced Machine Learning Final Project: Trajectory Forecasting with DPP

Instructor: Dr.Professor Fabio Galasso

Leonardo Placidi,
Negin Amininodoushan

A.Y. 2020 - 2021

Abstract

Our work introduces Determinantal Point Processes(DPP) for the first time in Transformers for trajectory forecasting. Our model is a modification of the Quantized Transformer developed by Prof.Galasso and his research group[1], Transformer in which we introduce a new formulation of DPP loss to retrain the model and modify the last part of the architecture to support natually multimodal futures. The loss is inspired by the paper Diverse Trajectory Forecasting with Determinantal Point Processes by Yuan and Kitani[2] and we used the there mentioned metrics ADE, FDE, plus a new metric we defined and called AD: the first two assess the probability integrity of the results, the latter the diversity. Results are presented in comparison with the multimodal results from the Quantized Transformer above mentioned and they sadly confirm our doubts about probabilistic consistency of the DPP adaptation. The method does not improve the ADE and FDE, and in some cases just sliglty reaches a higher score in the AD. In brief the main reasons are that we had to adapt a method not yet formulated for transformers and not really compatible, training with the loss itself in a transformers leads to losing forecasting' quality and finally the partially higher diversity induced is drawn back from the lack of good predictions. We discuss further in the report a possible second approach, quite more challenging but that could promise better results. As we were quite unsatisfied with our model, we tried to read more about DPP and we found a way of sampling with DPP directly from the probabilities in output from the Transformer. This leads us to a surprising improvement over the standard Quantized Transformer, almost doubling in the AD score, improving in the ADE and FDE scores, **that assess we got likely and diverse paths, and beating the benchmark** consisting of the scores on the original Quantized Transformer.

Introduction

Trajectory forecasting is a growing field that is receiving a great attention by scientists, in particular in the last year we observed an introduction of NLP methods for computer vision, the branch where Trajectory Forecasting is mainly studied. A good example of those models can be found in the mentioned paper 'Transformer Networks for Trajectory Forecasting' by our professor Galasso. This field aims at predicting the position of moving subjects given a brief observation of their past and in some cases at generating the 'multifutures' that can happen for every subject(all data are 2D).

Our work consists in a modified Quantized Transformer and a method that perform DPP sampling after the ouput of the Transformer at each step. In both the attempts we tried to achieve a model that based on Transformers would generate very diverse paths given a moving subject.

Here we introduce briefly the two models.

[1]Transformer Networks for Trajectory Forecasting by Giuliari, Hasan, Cristani, Galasso
[2]Diverse Trajectory Forecasting with Determinantal Point Processes by Yuan, Kitani

Quantized_TF with DPP loss: LeNeg

Since the paper for DPP sampling on which we inspired our work dealt mostly with VAE and cVAE, models that are not compatible with Transformers, we did adapt their algorithms that aim at training a sampling function, called DSF(γ), to a Transformer friendly code. We point out that the γ above mentioned, after some discussion with Prof.Galasso and Dr.Franco, we fix them as the Transformers parameters. So all the codes we wrote aim at modify the Quantized Transformer that would now give as output many possible positions given and input and will start the training from the optimized Quantized Transformer of Prof. Galasso (we selected the model trained at epoch 19). After the model is trained we wrote a recursive path creator, we hand coded the significant metrics from the DPP paper (ADE, FDE) and we created a new metric AverageDiversity(AD) that scores the diversity of the paths. The model, as we feared, present a loss in the probabilistic integrity which increases the ADE and FDE, and provides a not enough better result in terms of AD: the method is then unsuccessful. We will cast more light into a possible other approach, quite more challenging, but that would be able to mantain the probabilistic integrity. We called this attempt LeNeg, since is right to give names also to things that disappoint us but made us learn a lot.

DPP sampling

After not getting much success with the previous method, we decided to change our approach and get diverse samples with DPP after getting the probabilities from the transformer. We used a DPP sampler from a github repository [3] which is based on a Matlab code and has been converted in a semi-automated process using SMOP package. In the first time step, we feed the DPP sampler with 100 most likely trajectories from the transformer and and get the 21 most diverse ones and for each of them, the transformer predicts the next time steps. We also experimented more with using DPP sampling in both first and second time stamp but the diversity measure got lower in this case.

Related work

Our work is based on the papers **Transformer Networks for Trajectory Forecasting** and **Diverse Trajectory Forecasting with Determinantal Point Processes**. In Prof.Galasso paper we see an innovative application of Transformers for Trajectory forecasting, an adaptation of a NLP method that represents a SOA model. Dr.Kitani paper instead proposes a series of experiments where the introduction of a DPP loss to train a sampling function DSF(γ) that results in a visible increased variety in the outputs. Those experiments mainly deal with **diverse** trajectory forecasting of cars in motion and pose estimation. As discussed all the work there is based on VAE and cVAE and at start was inconsistent with any application for a Transformer. In particular our adaptation is based on considering what the paper calls 'latent codes' as the probabilities associated to the clusters relative to the sampled positions (from our modified Quantized Transformer) at each time in the future.

[3]https://github.com/chappers/dpp

Proposed method

First of all, we started to code from the Transformers repository by Francesco Giuliari [4], and you can find our work in the repository by Gruntrexpewrus[5]. The files that we produce are the following, all but the last are about the Quantized Transformer with DPP, the last is about the Transformer coupled with the DPP sampling after output(that was the best method):

- **Quantized_TFsamples.py**: this originally was the main class of Prof.Galasso Transformer, we introduced here, after the decoder, a sampling of from the ouput. The class is able to return both the cluster chosen and the relative position.
- **LossLeNeg.py**: Here is implemented the custom Loss following the algorithm for the DSF(γ) training of Kitani's paper, but adapted by us for the Transformer. More detail will come below.
- **Sampler_LeNeg.py**: This is the module where, using torch.multinomial, we sample at each step some clusters from the 1000 dimensional vector returned by the standard quantized transformer.
- **TESTDPP.py**: Here we coded all the metrics, defined a recursive function to create the mutlifutures and tested our results.
- **test_quantizedTF.py**: this is the code for the test using Prof.Galasso Transformer, we implemented also here our metrics and functions to compare the results later with our new method.
- **DPP_sampler.py**: this is the code for the test using Prof.Galasso Transformer and implement DPP sampler on the result of it for the first time inteval to add diversity. The result of it is called PP sampling after TF v1 in table 1.
- **DPP_sampler2.py**: this the code for test using Prof.Galasso Transformer and implement DPP sampler on the result of it for the first and second time interval. The result of it is called PP sampling after TF v2 in table 1.

Let's now talk a little more in detail about our implementation of a Transformer trained with the DPP loss.

The paper by Kitani was not directly adaptable to transformers and didn't provide codes to check our results, then we had to work on it and extract our adaptation. In the paper all the method is based on training a sampling function that they say is a neural network with parameters γ. This for us was not possible to achieve because that DSF sampled from the output of the encoder of a VAE or cVAE. In those model usually we have a representation in form of a normal distribution and we never had such things in our Transformers. So we had to invent a way to adapt the DSF and after some thinking and discussion with Prof.Galasso and Dr.Franco, we decided to consider the entire parameters of the Transformer as γ to be trained, casting sure some normal doubts about the quality of the predicted paths (that indeed resulted the principal cause of failure

[4]https://github.com/FGiuliari/Trajectory-Transformer/
[5]https://github.com/Gruntrexpewrus/TrajectoryFor-and-DPP

in the scores), but seemed a good way of mixing the two methods and as student we must do and learn from our mistakes.

As in the introduction we already dealt with the idea of the work, we will now go in detail in the implementation of the DPP loss. The modification of the Quantized Transformer can been understood from the list of files above and for any doubt the codes are fully commented about implementation and choices.

0.1 Implementation of the DPP loss

The paper introduces this algorithm to train a DSF(γ) with DPP, the procedure must be repeated for every trajectory until convergence.

- Generate latent codes $\{z_1, ..., z_n\}$ with the DSF(γ).
- Compute the ground set $Y = x_1, ...x_n$ with the decoder from the z_i.
- Compute the similarity matrix S and quality vector r, the equation will be shown below.
- Compute the DPP kernel $L(\gamma) = Diag(r) * S * Diag(r)$[6]
- Compute diversity Loss $L_{diverse} = trace(I - (L(\gamma) + I)^{-1})$
- Update γ with the gradient of the loss, note that the loss is likely the opposite of the opposite of the opposite sign of a probability, therefore is a negative loss.

Our choices for the z_i where the probabilities associated from the clsuters of the sampled positions, the x_i where the 2D positions that you obtained from the sampled clusters. The generation of the latent code was done mainly in Sampler-LeNeg.py where there was a sampling torch.multinomial(). So to be more explicative the dimensions of our set of z_i for one batch was $[num\_el\_in\_batch, num\_samples]$ where the elements are probabilities, the x_i where of dimension $[num\_el\_in\_batch, num\_samples, 2]$, where every input had a tot of possible future positions.

Here to be complete the formula to compute the similarity matrix S and the quality vector r.

The similarity matrix (we chose k = 0.1):

$$S_{ij} = exp(-k||x_i - x_j||^2) \tag{1}$$

The quality vector (R was such that the sphere included 90% of the trajectories, ω has been set to 0.1):

$$r_i = \begin{cases} \omega, & \text{if } ||z_i|| \leq R \\ \omega exp(-z_i^T z_i + R^2), & \text{otherwise} \end{cases}$$

The above mentioned parameters come from a brief parameter tuning where we found out that values between 0 and 1 for k and ω were better.

Dataset and benchmark

Dataset

We used zara1 and zara2 datasets from ETH/UCY for our experiments.

[6] dot product

Metrics and benchmarks

The metrics we used in the project are explained below. Two of them were mentioned in diverse trajectory forecasting paper and the other one is invented by us.

- **ADE (Average Displacement Error)**: for calculating ADE, we had to create lists of noisy ground truths (the number of noisy ground truth that we created for each element is equal to number of samples-1) and then we got the min squared norm between all noisy ground truths and samples for each time step and then calculated the scaled average of that within 12 time intervals. The formula for this metric is:

$$ADE = \frac{1}{T \cdot |\chi^i|} \sum_{x \in \chi^i} min_{\tilde{x} \in y_f} ||\tilde{x} - x||^2$$

  where χ^i is set of noisy ground truth and y_f is the set of forecasted trajectories.

- **FDE (Final Displacement Error)**: for getting FDE, we did the same procedure as ADE but with the difference that we only considered the last time interval. The formula for this metric is:

$$FDE = \frac{1}{|\chi^i|} \sum_{x \in \chi^i} min_{\tilde{x} \in y_f} ||\tilde{x}^T - x^T||^2$$

- **AD (Average Diversity)**: We created this metric for measuring the diversity within our samples. The average diversity is calculated by averaging over min euclidean norm between all the trajectories; so if the AD is higher, we actually have more diversity in average in our samples.

$$AD = \frac{1}{|Trajectories|} \sum_{x_j} min_{x_i, i \neq j} ||x_j - x_i||$$

  where x_j the entire trajectories.

We also implemented ADE and FDE measures in transformer trajectory paper codes. We compared the results of our 2 methods to the results of both prof.Galasso Transformer for multimodality. The result of our experiment is shown in the next section, then the benchmark that we used has been the scores computed on the Transformer Network from Prof.Galasso paper.

Experimental results

For choosing the right transformer model we trained the quantized transformer model and choose the number of epoch that gives us a good train and validation loss and does not have overfit; we decided to choose the model with 19 epoch for zara1 and 18 epochs for zara2.

For choosing a good model of LeNeg, we did similarly and the results of train and validation loss for LeNeg are shown in table 3. For zara1 dataset, we choose epoch 15 since it has a good validation loss and after it the validatoin loss gets

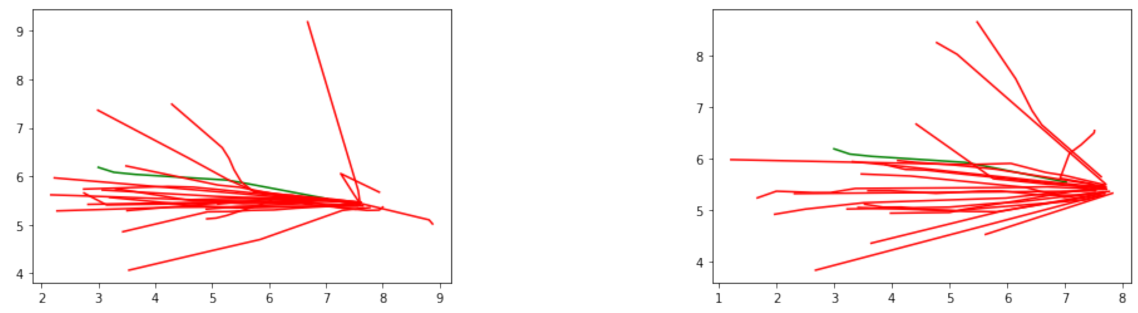

Figure 2: On left result from transformer quantized, on right result of model with DPP sampling v1

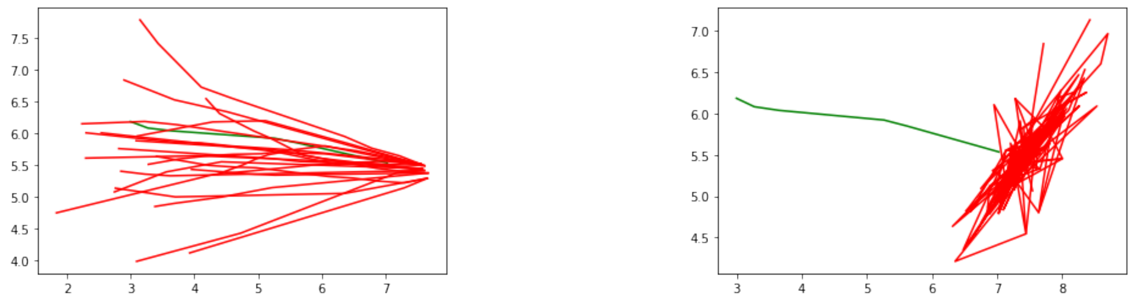

Figure 4: On the left result of model with DPP sampling v1, on the right result of LeNeg model

worse though the training loss gets better so it has overfitting after epoch 15. For the same reasoning, we choose LeNeg with 5 epochs for zara2
As we can see in the table 1 and table 2, the LeNeg model could not predict trajectories well and ADE and FDE scores are worse than the base quantized but it could diversify samples a bit more. On the other hand, the DPP sampling on the results of transformers not only diversify a lot the samples but could also improve the FDE and ADE with respect to quantized transformer.
So in other end the DPP sampling beat the Quantized transformer on every metric, therefore we **beat the benchmark**.

zara1	**ADE**	**FDE**	**AD**
Quantized TF	2.84690	2.55312	1.76516
LeNeg TF	10.14295	20.00560	1.96567
DPP sampling after TF v1	2.29275	1.68296	2.72366
DPP sampling after TF v2	2.2628	1.6205	2.6738

Table 1: Comparison of metrics for Zara1 on different models, DPP sampling beat the benchmark

zara2	**ADE**	**FDE**	**AD**
Quantized TF	7.4053	7.0980	1.3427
LeNeg TF	6.3881	11.6196	1.9034
DPP sampling after TF v1	6.0470	4.8251	2.5626
DPP sampling after TF v2	6.1066	4.8685	2.3016

Table 2: Comparison of metrics for Zara2 on different models, DPP sampling beat the benchmark

Epochs	**Train loss zara1**	**Val loss zara1**	**Train loss zara2**	**Vall loss zara2**
0	-0.1955	-0.1973	-0.1969	-0.1944
1	-0.1860	-0.1872	-0.1875	-0.1807
2	-0.1756	-0.1751	-0.1759	-0.1765
3	-0.1774	-0.1788	-0.1766	-0.1743
4	-0.1996	-0.1776	-0.1786	-0.1704
5	-0.1715	-0.1945	-0.1705	-0.1704
6	-0.1714	-0.1885	-0.1709	-0.1706
7	-0.1718	-0.1720	-0.1713	-0.1708
8	-0.1722	-0.1735	-0.1810	-0.1710
9	-0.1739	-0.1767	-0.1710	-0.1707
10	-0.2004	-0.2168	-0.1713	-0.1708
11	-0.1723	-0.3258	-0.1716	-0.1711
12	-0.1728	-0.1871	-0.1724	-0.1715
13	-0.1757	-0.1715	-0.1839	-0.2092
14	-0.2251	-0.1718	-0.2156	-0.1817
15	-0.1721	-0.1720	-0.2178	-0.2285
16	-0.1807	-0.1722	-0.2105	-0.1872
17	0.3009	-0.1730	-0.2723	-0.2142
18	-0.4135	-0.1782	-0.2803	-0.2489
19	-0.4775	-0.2806	-0.3313	-0.2335

Table 3: Train and validation loss through different epochs, on zara1 and zara2 for LeNeg

Conclusion and future works

In conclusion our approach of modifying the Transformer from the decoder has been unsuccessful and presented the probabilistic inconsistencies we feared without much gain, while on the other side the DPP sampling after the Transformer resulted a much better method that preserved the probabilistic integrity of the results and almost doubled the diversity score of the possible trajectories, **giving then exactly what we wanted and beating our benchmark**. From what said then we agree that the paper from Dr.Kitani is not exaclty the best inspiration to modify the decoder of a Transformer, because as you saw above the loss does not take into account the real trajectories, losing then a lot in the prediction quality. For future work we believe there is a better adaptation, quite above our scope here, that is to develop a new Transformer network, let's call it Variational Transformer, which would have a cVAE at the end of the encoder. This cVAE will be trained to generate what now are the encoder outputs of a Transformer. This will lead to a direct adaptation of the algorithms of the DPP paper, because we could define their same DSF(γ) and use the DPP loss where z_i are normal distributions and x_i are the output of the decoder of the Transformer. At this point we could summarize this method saying that would produce a transformer that would not generate 1 encoder output for every input, but $num\_samples$ encoder outputs that would then all pass through a trained decoder, generating very likely results. This method we believe could improve dramatically the performance diversity of sampled paths since it will just vary slightly the representation that the attention layers provide in the encoder, and the training of the DSF would not impact on the forecasting ability of the transformer. We believe this model would be as interesting as difficult and as agreed with Prof.Galasso was not in the scope of this final project.

Finally we are happy to have learned so much in this new field and even if in the end our main model resulted in a failure we enjoyed our research experience and we are grateful to our Professor Fabio Galasso for his suggestion of working on this challengin topic: it has been exciting to touch state of the art models and try to modify them!

1 Bibliography

- 1) 'Diverse Trajectory Forecasting with Determinantal Point Processes', ICLR'20, Ye Yuan-Kris M.Kitani.
- 2) 'Transformer Networks for Trajectory Forecasting', ICPR'20, Giuliari-Hasan-Cristani-Galasso.
- 3) 'Determinantal point processes for machine learning', Alex Kulesza-Ben Taskar.