Predicting Agent Trajectory using Temporal data

Vaidehi Som
Dept of Robotics (GRASP)
University of Pennsylvania
somv@seas.upenn.edu

Ayush Goel
Dept of Robotics (GRASP)
University of Pennsylvania
aygoel@seas.upenn.edu

Dheeraj Bhurewar
Dept of Mechanical Engineering (MEAM)
University of Pennsylvania
bhurewar@seas.upenn.edu

***Abstract*—For robots and self driving cars to move in human surroundings the robots should be able to predict the trajectory to decide its own path and motion. For this reason the problem of trajectory prediction becomes important. Working on this problem of trajectory prediction of pedestrians for self driving scenario, we have compared different models of LSTM and GRU to show what type of model works best for this problem. We have compared 4 models- Vanilla LSTM, Social LSTM, OLSTM, and GRU to show their comparison for predicting non linear trajectories of pedestrians in different scenes. We demonstrate their performance on publically available datasets. Comparing all these models we show how it is important to take into account the surroundings of the pedestrians to predict with better accuracy. We humans change our trajectory based on the obstacles we encounter, so it is essential that we take the obstacles in the scene as input to our models to predict accurately.**

***Index Terms*—Self Driving Car, Trajectory prediction, Vanilla LSTM, Social LSTM, OLSTM, GRU, Dynamic obstacles**

I. Introduction

An autonomous vehicle needs to perceive, interpret, and predict the behavior of the agents (other cars, pedestrians, etc) in it's surroundings to adjust its path and plan its motion.

In this paper, we are focusing on the trajectory prediction of pedestrians. The ability of modelling the behavior of humans motion is important for many tasks, not just for self driving. When robots are deployed near humans, it is important that they be able to predict their motion and avoid these dynamic obstacles. Intelligent tracking and other such systems will also require such type of prediction.

We have addressed this problem using Recurrent Neural Networks (RNN) which works best for forecasting using temporal data. RNNs have been shown to perform best on such kind of data in comparison to other traditional machine learning approaches like KNNs and Linear Regression. We have specifically used variants of RNNs- LSTM and GRU, and compared their performance when training and predicting different lengths of pedestrian's trajectory

For Self Driving Cars to function we need to know the current and future state (positions and velocities) of the agents (pedestrians, other cars, etc) in the environment. This information is important for us to find the best path for our car to travel on. In our project we want to predict the motion of these agents given their temporal data. This motion prediction of agents can then be used to find the optimal path for our self driving car.

It is very hard to model the non linearity of human motion with a mathematical model. Since neural nets are used to solve similar non linear problems, it is best to use Deep Learning models to solve the complexity of human interaction given the dependency of trajectory on many parameters.

II. Literature review

Recurrent Neural Networks (RNNs) have been used to predict sequence tasks in many works in the application of image/video classification [4], [5], [6] and [7], speech recognition [8], [9], [10], semantic segmentation [11], and predicting handwriting sequence [12]. These works have led people to use RNNs for trajectory prediction which is the same sequence prediction problem.

Trajectory prediction has also been solved using other methods. Pioneering work from Helbing and Molnar [13] presented a pedestrian motion model with attractive and repulsive forces referred to as the Social Force model. This has been shown to achieve competitive results even on modern pedestrian datasets. Similar approaches have been used to model human-human interactions with strong priors for the model. Treuille et. al. [14] use continuum dynamics, Antonini et. al. [15] propose a Discrete Choice framework and Wang et. al. [16], Tay et. al. [17] use Gaussian processes. Such functions have alse been used to study stationary groups [18]. These works target smooth motion paths and do not handle the problems associated with discretization. But most of these models provide hand-crafted energy potentials based on relative distances and rules for specific scenes. Since writing such rules and models is not an easy task and involve lots of human annotations and parameter selections, using deep learning gives us ways to learn such motions using the data instead.

LSTM was first introduced in [19], and GRU in [20]. These two models have been used in trajectory prediction problems. But the problem with these architectures are that they do not take into account the surrounding of the agent. We aim to compare these models with the work of [1][] which introduces Social LSTM and OLSTM, that take as input the trajectory of neighboring agents.

III. Dataset

We have tested our work on the publically available datasets of UCY [3] and ETH [2]. The dataset has been recorded from bird eye view and annotated manually by the authors as shown in Fig. 2. Since the datasets have been collected in real world scenarios, they are stochastic with respect to number of pedestrians in each frame and the length of steps each

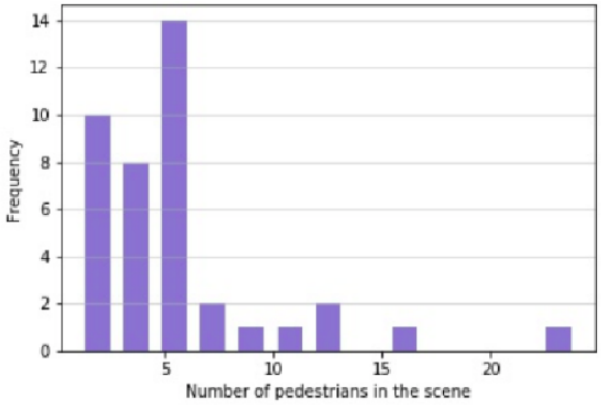

Fig. 1: Histogram showing variation of number of pedestrians with frames

pedestrian takes. The scenes are crowded and show pedestrians in different settings, like people walking as a couple or group. The trajectories are highly stochastic and non linear. The data is in the form of txt files with each txt file having 4 values- frame number, pedestrian id, x value, y value.

Both UCY and ETH datasets contains 2 scenes with 786 and 750 different pedestrians respectively in each scene. UCY is divided into 3 sets (UCY, ZARA-01, ZARA-02) and ETH into 2 (Hotel, ETH). So we have trained and tested our models by combining all 5 sets. We trained on 4 datasets, tested on 5th and repeat this for all datasets. Also the scenes in datasets are very different, with different paths which the pedestrians can travel on. This ensures that we have diverse set on dataset and will not overfit due to the dataset being very similar. Fig. 1 shows the variation of number of pedestrians with random 40 frames.

For data preprocessing, we eliminated pedestrians based on their total trajectory length, i.e. if $L_{pedes} < L_{total}$ where:

$$L_{total} = L_{observed} + L_{predicted}$$

Trajectories with length greater than L_{total} were truncated. After the preprocessing our dataset comes about the length of 40000 datapoints with 20 datapoints per pedestrian.

We divide the length of pedestrians trajectory into observed and future(which are same in number as predicted) steps. Then we train on observed number of steps, predict the predicted number of steps and compare our predicted steps with the actual future steps of pedestrian.

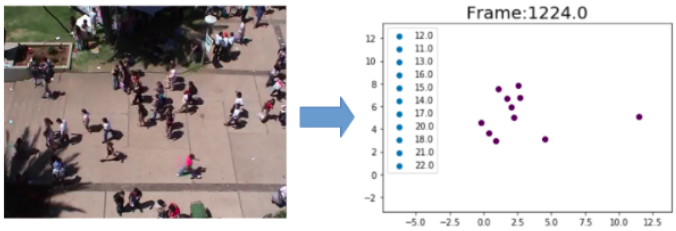

Fig. 2: ETH dataset showing conversion from bird eye view to spatial coordiantes

IV. Problem Formulation

We take as input the spatial coordinates (x,y) of pedestrians from time 1 to $T_{observed}$ and predict the spatial coordinated from time $T_{observed} + 1$ to $T_{observed} + T_{predicted}$. This is an example of sequence generation problem. The input sequence is the observed trajectory of pedestrian and our output is the future trajectory during different time steps of that person.

The following three comparison metrics were used for comparing the prediction accuracy between different machine models and datasets:

1) Mean Square Error (MSE)- We calculated the second norm of the difference between two vectors to evaluate the prediction of each pedestrian.
2) Average Displacement Error - Euclidean distance between each step of the predicted trajectory and ground truth, further dividing it by the trajectory length.
3) Final Displacement Error - Euclidean distance between the final predicted position and ground truth position of each pedestrian's trajectory.

This performance matrix works best for our problem because we are predicting a sequence. We cannot show accuracy/AUC as a performace measure as we are predicting more than 1 time step into the future and we need to quantify the uncertainty on each time step in meters so that our robot is able to use this error measurement as a safety constraint around the predicted path.

V. Methods

In our work, we have baselined the performance of Vanilla LSTM and shown its comparison to Social LSTM and OLSTM (LSTM with occupancy maps). We have also shown the comparison of LSTM against GRU in different scenario, but the GRU used has same architecture to that of Vanilla LSTM. Since GRU is shown to have similar performance as LSTM, it has been compared only to show its computational advantage over Vanilla LSTM.

A. Vanilla LSTM

This model has the architecure shown in Fig 3. It has an embedding dimension of 64 which is fed in to LSTM cell. We first input the coordiantes to a dense layer which maps the data to a higher dimensionality. Hidden state dimension of 128 is used for the LSTM cell. The hidden layer in LSTM captures

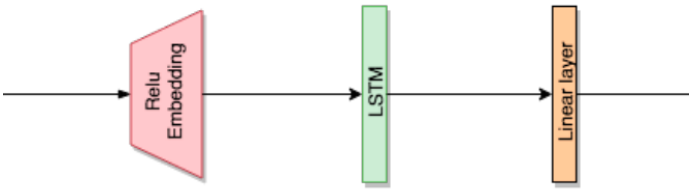

Fig. 3: Vanilla LSTM architecture

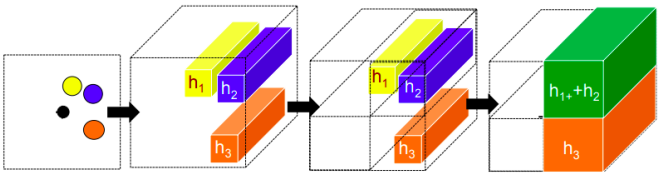

Fig. 4: Taken from [1]. The Social pooling for the person is represented by a black-dot. We then pool the hidden states of the neighbors (shown in yellow, blue and orange) within a certain spatial distance. The pooling partially preserves the spatial information of neighbors as shown in the last two steps

the latent unformation of that agent. Dropout of 0.5 is applied to dense layer. The embedding layer then uses ReLU non linearity before the features are fed into RNN cells. Learning rate has been taken as 0.0003. Optimizer used is Adam.

B. GRU

GRU has exactly the same architecture as Vanilla LSTM with same hyperparameters. We just use GRU cell from PyTorch in-place of LSTM cell.

C. OLSTM

The Occupancy Map LSTM (OSTM) uses an additional matrix to that of Vanilla LSTM to input the information of the occupancy grids which are occupied by other agents in the agent's scene. It is introduced in [1], and the architecture is taken from this paper. To compute the map, we discretize the frame into MXN with our agent in the middle. OLSTM is also similar to Vanilla LSTM except that we also input the occupancy grid with our agent's observed trajectory to our model. This model is therefore able to learn information from its surrounding.

D. Social LSTM

We have used social LSTMs to overcome the limitation of Vanilla LSTM. Vanilla LSTM does not take as input the motion of neighboring pedestrians to predict the motion of an agent. When people are moving, they adjust their motion according to the motion of people around them. Therefore it is important that we model this parameter to gain better results. To combine all the information from the complete scene, Social LSTMs have been introduced in [1]. The architecture of social LSTM is shown in Fig 5 which is taken from the same paper. Social LSTMs handle the scenery information by using social pooling layer as shown in Fig 4. At every time step, the LSTM cell receives the information from its surrounding scene with these pooling layers. The information from hidden layer for an agent is shared with the LSTM cell of the other agent using this social pooling layer. The spatial pooling size is taken as 32 with pooling window to be 8x8. Unlike OLSTM, we back propagate across all trajectories during training.

When comparing all models, we have taken same hyper-parameters. We took these hyperparameters from the implementation of social LSTM in [1], and used the same hyperparameters for all other models too. To summarize,

1) Input size and output size is 2, as they are spatial coordiantes.

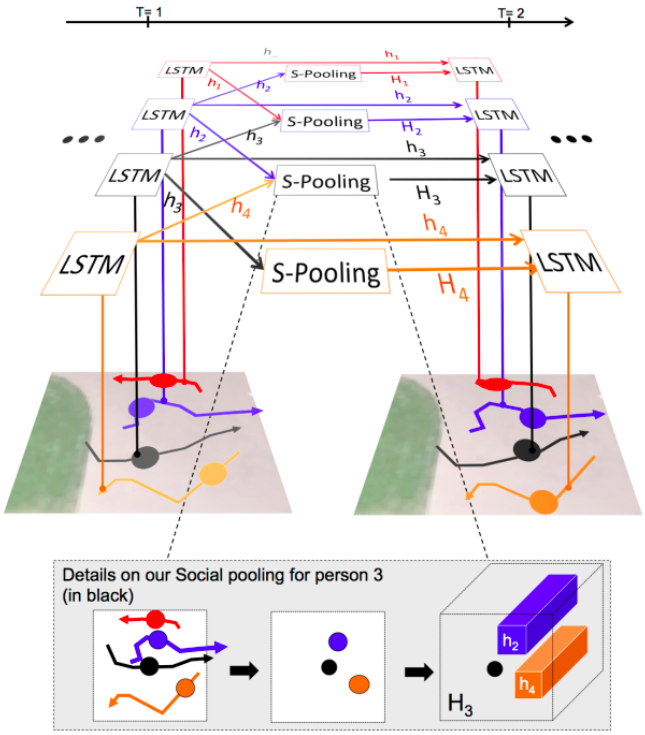

Fig. 5: Taken from [1]. . Overview of Social-LSTM method. Separate LSTM network is used for each trajectory in a scene. The LSTMs are then connected to each other through a Social pooling (S-pooling) layer. Unlike the traditional LSTM, this pooling layer allows spatially proximal LSTMs to share information with each other. The bottom row shows the S-pooling for one person in the scene. The hidden-states of all LSTMs within a certain radius are pooled together and used as an input at the next time-step.

2) Dropout value is 0.5
3) For all spatial coordinates, social tensors, and occupancy map, the embedding size has been taken as 64
4) Hidden state size in LSTM cell is 128
5) Spatial pooling size use in social LSTM is taken to be 32 with pooling size window 8x8
6) All models are trained with GPU
7) Grid size is taken as 16x16 for both Social LSTM and OLSTM

The loss function used by Vanilla LSTM and GRU is MSE loss, and Social LSTM and OLSTM are minimizing negative log-Likelihood loss. We have used different loss functions to check the variation of performance of different versions of LSTMs under different loss functions.

VI. Experiments and Results

Fig 6 - Fig 8 shows the observed, predicted, and ground truth trajectory of pedestrians in a scene. As observable in the figure clearly, the model is capable of successfully predicting even the rare scenarios such as when the pedestrian is relatively stationary or traversing at a slower speed.

Fig 6 shows results for steps 4, Fig 7 for 6 steps and Fig 8 for 8 steps. We can see how for steps 4, Vanilla LSTM predicts perfectly the future steps but the prediction for 8 time steps is not very good. The performance of models depend a lot on the predicted length, as it becomes harder to predict long time into the future. Fig 9 shows the train and test final and average errors for Vanilla LSMT with varying prediction lengths.

For showing our comparison between our baseline model and other models, we first show the difference through numbers obtained by evaluation matrix. Our evaluation matrix, as discussed in Section IV, has been divided into test and train for comparison across all models. Fig 11 shows the average displacement error during training against number of iterations. We can see how the Error of Social LSTM converges the fastest out of all three variations. It takes Social LSTM half of the time to give better results than other both. Table I shows the comparison across all of our models. We can see how social LSTM gives us the best results, as expected.

To visually show our results, we also have plotted the predicted trajectory by all our models against linear motion. The results are shown in Fig 10

	Vanilla LSTM	GRU	Social LSTM	OLSTM
Avg train disp err	0.771	0.7029	0.333	0.407
Final train disp err	1.597	1.2921	0.5021	0.7752
Avg test disp err	1.2648	1.3957	1.2176	1.236
Final test disp err	2.755	2.6529	2.543	2.643

TABLE I: Comparison of all models used. All values are in meters

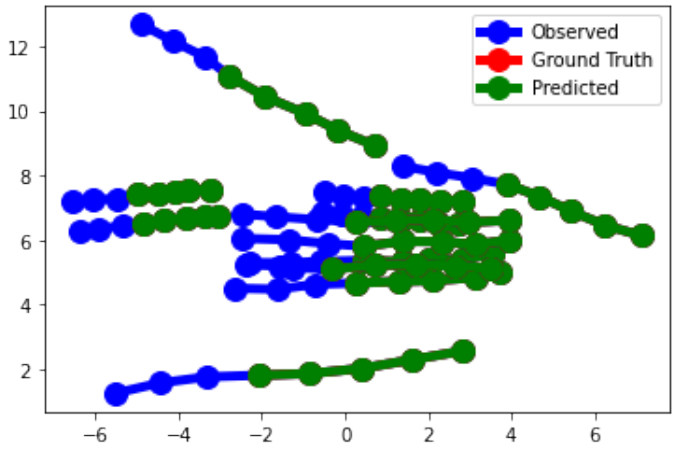

Fig. 6: Predicted trajectory for time step 4 using Vanilla LSTM

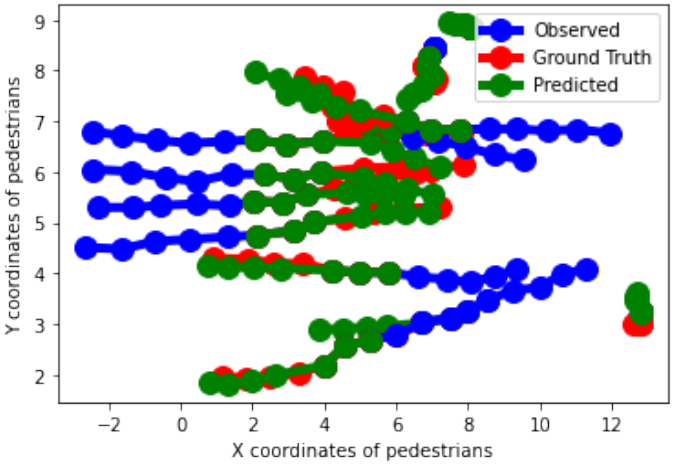

Fig. 7: Predicted trajectory for time step 6 using Vanilla LSTM

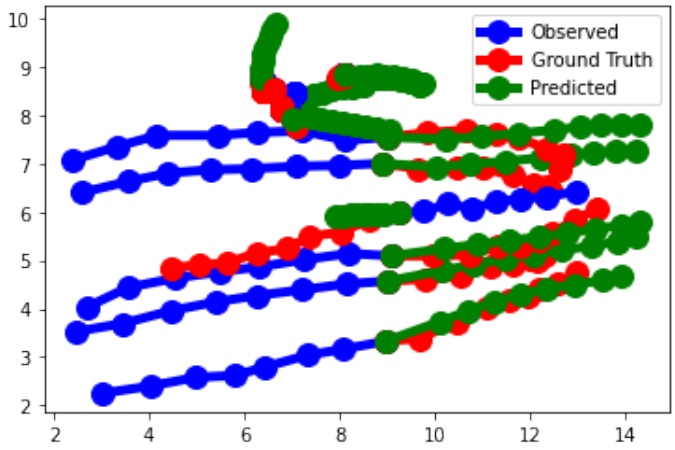

Fig. 8: Predicted trajectory for time step 8 using Vanilla LSTM

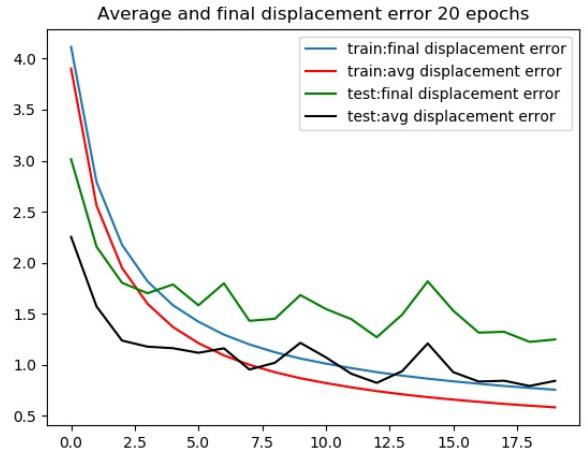

Fig. 9: Evaluation matrix for Vanilla LSTM with varying prediction lengths

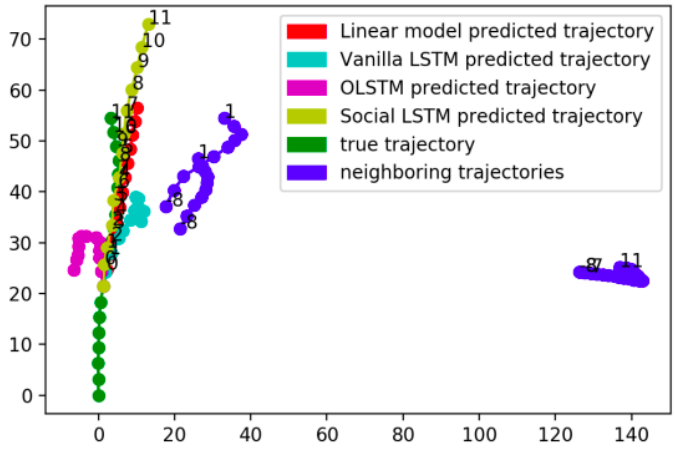

Fig. 10: Evaluation matrix for Vanilla LSTM with varying prediction lengths

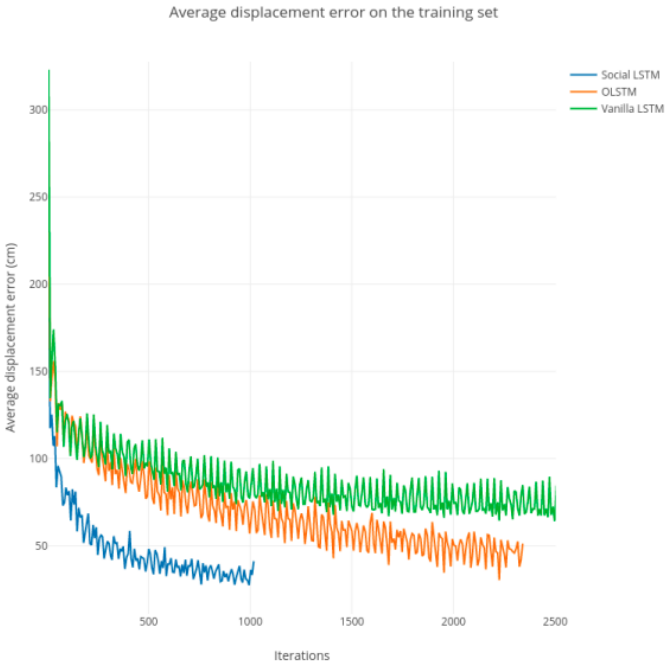

Fig. 11: Average displacement error evolution during training with iterations

VII. Conclusion and Discussion

We have in our work presented different versions of LSTMs that can predict human trajectory. By comparing the results of all these versions, we can see how social LSTM gives the best result. This is expected as the social LSTM takes into account the neighboring trajectories and use the social pooling layer to include the social interaction parameters humans show while walking.

We also see how GRU gives similar results as it shares the same architecture but because it is less computationally expensive, it can be used to run on self driving car or any robot in real time to predict trajectories. LSTM had marginally better performance, though a longer run time. GRU performed very similar to LSTM, but much shorter run time since the GRU cell has lesser number of gates.GRU uses less training parameter and therefore uses less memory and executes faster than LSTM whereas LSTM is more accurate on a larger dataset. One can choose LSTM if we are dealing with large sequences and accuracy is concerned, GRU is used when we have less memory consumption and want faster results.

Our models are able to predict non linear trajectories with reasonable accuracy. The results can be made better with more data obtained in varying scenery to make our training more robust. More data will be able to make our predictions for longer trajectories better.

We can interpret from our results that predicting only based on few datapoints will not help us in learning the non linear predictions to far length into the future. We tried to add the penalty in function of the distance to neighboring pedestrian in our loss function believing that it can help our models improve their collision avoidance properties, but we were not able to get improved results, so we used the implementation as presented in [1]. We also learned through our implementations that how important the distribution of our dataset is. We first tried only using a subset of the dataset, only using ETH dataset. That did not give us good results because our models overfit the dataset. We also had to play around with number of epochs to train our models on, as they were not giving very good results with low number of epochs which have been used in some papers.

As possible extensions to our work, we are interested in exploring how the inclusion of different agents in the same scene affects the behavior of our models. We have only trained and tested on scenes which contain only pedestrians. Inclusion of cars, bicycles, skateboards, etc in the scene will make our problem more complex. While considering future trajectories, we should also consider the scenery of our agents. People will respond to the objects in their surroundings that will act like obstacles. This scene information should also be taken as input to our models to predict trajectory in a better way.

References

[1] Alahi, Alexandre, et al. "Social lstm: Human trajectory prediction in crowded spaces." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016

[2] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In Computer Vision, 2009 IEEE 12th International Conference on, pages 261–268. IEEE, 2009.

[3] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. In Computer Graphics Forum, volume 26, pages 655–664. Wiley Online Library, 2007

[4] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu, et al. Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks. ICCV, 2015

[5] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. Draw: A recurrent neural network for image generation. arXiv preprint arXiv:1502.04623, 2015.

[6] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. arXiv preprint arXiv:1411.6447, 2014

[7] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. arXiv preprint arXiv:1503.08909, 2015.

[8] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Proceedings of the 31st International Conference on Machine Learning (ICML14), pages 1764–1772, 2014

[9] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio. Endto-end continuous speech recognition using attention-based recurrent nn: First results. arXiv preprint arXiv:1412.1602, 2014.

[10] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. CoRR, abs/1506.02216, 2015.

[11] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. arXiv preprint arXiv:1502.03240, 2015.

[12] A. Graves. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.

[13] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. Physical review E, 51(5):4282, 1995.

[14] A. Treuille, S. Cooper, and Z. Popovic. Continuum crowds. ´ In ACM Transactions on Graphics (TOG), volume 25, pages 1160–1168. ACM, 2006.

[15] G. Antonini, M. Bierlaire, and M. Weber. Discrete choice models of pedestrian walking behavior. Transportation Research Part B: Methodological, 40(8):667–687, 2006

[16] J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 30(2):283– 298, 2008.

[17] M. K. C. Tay and C. Laugier. Modelling smooth paths using gaussian processes. In Field and Service Robotics, pages 381–390. Springer, 2008

[18] S. Yi, H. Li, and X. Wang. Understanding pedestrian behaviors from stationary crowd groups. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3488–3496, 2015.

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

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.