Pedestrian Trajectory Prediction using Machine Learning and Deep Learning Algorithms

Ashish Roongta, Bryan Zhao, Pranav Narahari, Yayati Jadhav

{aroongta, bryanz, pnarahar, yayatij}@andrew.cmu.edu

Abstract

Modeling the interactions between autonomous vehicles (AVs) and pedestrians is currently a critical issue for public safety and predictive modeling. Accurate predictions of pedestrian trajectories would allow AVs to safely navigate around pedestrians in crowded scenarios. In this paper, we have used both traditional machine learning algorithms (linear regression, KNN regression) and deep learning frameworks (Vanilla LSTM, GRU) to predict future steps, in the form of spatial trajectories, of various pedestrians based on their previous trajectories. We evaluate the proposed methods on publicly available datasets and then we use Mean Square Error (MSE) loss, final displacement error and average displacement error as the metrics for performance. Finally, we present a study on the influence of observed trajectory length and predicted trajectory length on prediction accuracy.

1. INTRODUCTION

With the advent of autonomous vehicles, the problem of predicting pedestrian path trajectories has a renewed interest in academia and industry. For autonomous vehicles to transform from academic exploration to commercial products, it is imperative for AVs to be able to predict the future positions of the pedestrians based on their past trajectories. This type of prediction is of utmost importance in critical scenarios such as: when a pedestrian is walking on the sidewalk and makes a sudden turn on the road or when the pedestrian suddenly stops. The ability of AVs to recognize this type of pedestrian behaviour will help with the design of intelligent tracking systems and modeling countermeasures for such cases.

Humans have the ability to intuitively "read" situations and generally traverse based on certain trends such as respecting the personal space of others and yielding right of way, which is unique to every pedestrian. Furthermore, the walking behaviour of pedestrians is also largely dependent on the environment, crowd density and obstacles in their path. The problem of trajectory prediction can be viewed as a sequence generation task, where we forecast the future trajectories based on past positions [5].

Model-based methods in the field of pedestrian modeling are largely dependent on hand-crafted factors such as preferred walking speed and spatial locations of the pedestrians, and thus cannot be generalized. Also, it is not as straightforward to combine all trajectory influence factors into a single model. This limits the application of model-based methods for trajectory prediction in crowded scenes [1].

In this project, we present a performance comparison between traditional machine learning algorithms (Linear Regression and KNN with linear regression) and deep learning frameworks (Vanilla LSTM and GRU).

2. LITERATURE REVIEW

C.Song et al. [5] have shown that autonomous mobility is predictable at a city-scale level, which is encouraging for large-scale pedestrian modeling. Kitani et al. [6] in their study have shown that scene schematics are strong cues used to forecast pedestrian's trajectory. J.Wiest et al. [7] in their work used traditional machine learning algorithms such as Gaussian mixture model by fitting data into their models, whereas Asahara et al. [8] have used Mixed Markov-chain models. Additionally, Ellis et al. [9] have used Gaussian process regression in their approach for pedestrian modeling.

Another related method was developed by Helbing et al., which incorporates attractive and repulsive forces, also known as the Social Force model, in their pedestrian motion model [10]. Though this Social Force model has been shown to perform well on pedestrian datasets, it completely relies on intricate hand-crafted functions based on distance-based energy potentials, and is not a data-driven model. In contrast, we then look to models in deep learning (with a benchmark in machine learning) that can learn these pedes-

trian trajectories in a more generalized way that is data-driven.

In recent years, variations of Recurrent Neural Networks (RNNs), which include Gated Recurrent Units (GRUs) [11] and Long Short-Term Memory (LSTM) [12] have proven to be very successful for sequence prediction tasks, such as text recognition, speech recognition, automatic caption generation, image and video classification, and predicting human dynamics to name a few [3]. Various authors have attempted to use multiple networks to learn and capture intricate interactions in crowd settings [1, 3, 4]. Alahi et al. [1] wrote a pioneering work that utilizes a "social pooling" layer that models nearby pedestrians. However, none of these attempts have benchmarked the performance of conventional machine learning algorithms against the aforementioned deep learning frameworks. Moreover, it has been the standard for research papers to predict trajectory lengths of 12 steps [1], but there have been no in-depth studies or explanations on the effects of varying the number of steps.

3. DATASET

Publicly available pedestrian data-sets from ETH [4], UCY [2] and Stanford [13] have been used for this study. The data-sets contain the trajectory of pedestrians in the form of spatial coordinates of pedestrians at each frame-interval of 0.4 seconds. These spatial trajectories the pedestrians were obtained from manually recorded and annotated crowd videos through an aerial-view camera. The highly

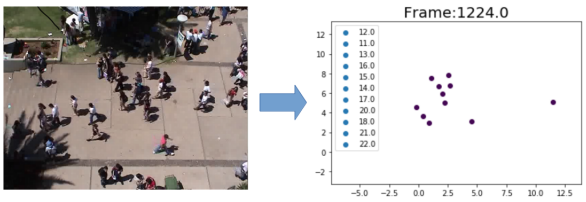


Figure 1. Conversion of aerial camera view to spatial coordinates

stochastic nature of pedestrian trajectories warrants extensive data preprocessing. Sequence length, trajectory curve as well as the number of pedestrians in each frame was found to be highly variable. Fig.1 shows variance of the number of pedestrians with the frames.

3.1. Data Preprocessing for Linear and KNN Regression

The raw data consists of pedestrians' ID, 2-D spatial coordinates and frame numbers. The number of features or weights are equal to the observed sequence of the pedestrians. For each pedestrian, the trajectory is split into blocks, where:

$$BlockSize = Features + Predicted$$

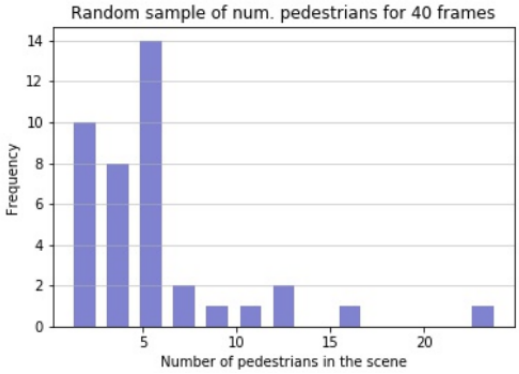


Figure 2. Histogram demonstrating the stochastic nature of pedestrian trajectories

$Features$:= Number of features for the model
$Predicted$:= 1 predicted step, i.e. the output

The trajectory of $Features$ are added to the observed set and the last spatial coordinate is added to the ground truth set. Pedestrians with number of steps less than the required block size are ignored. Also, the indivisible part of the trajectory of a person is truncated. For instance, if a pedestrian has 14 steps in total and for an observed sequence of 5 steps, 2 blocks of 6 steps will be formed with the first 5 steps of each block added to the training set and the last step to the ground truth set.

3.2. Data Preprocessing for Deep Learning algorithms

Since the dataset is highly stochastic in terms of the number of pedestrians in each frame and trajectory length, the data was processed by eliminating the pedestrians with sequence length lesser than the total trajectory length (L_{total}), where:

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

$L_{observed}$: observed trajectory length
$L_{predicted}$: predicted trajectory length

Additionally, the trajectories with length greater than L_{total} were truncated.

4. METHODOLOGY

In this study, we benchmarked the performance of traditional machine learning algorithms against deep learning frameworks. The following methods were implemented:

1. Linear Regression
2. KNN Regression
3. LSTM
4. GRU

4.1. Linear Regression and KNN

Linear regression was first used to set a baseline and determine how closely we can predict the pedestrians trajectories using a simple machine learning algorithm. The model is directly trained on data from 7 different datasets and tested on an 8^{th} unseen data set.

Since the distribution of pedestrians is very stochastic, it is hard to separate the entire dataset with a single hyperplane. We implemented KNN to solve this problem. KNN is a non-parametric method which is used for classification and regression. The algorithm predicts the trajectory based on similarity measurement. In this project, we picked the K nearest neighbours of a test point by finding the distance between the point and the training data to perform linear regression.

4.2. Deep Learning Algorithms

A pedestrian's trajectory is dependent on a variety of features such as his/her previous trajectory, the scene layout, social interaction with other neighbouring pedestrians, intent of motion (e.g. gaze direction), age and group size, to name a few. Intuitively, the trajectories of pedestrians can be considered as time sequence data, therefore sequence generation algorithms such as Recurrent Neural Networks (RNN) seemed most suitable to the problem statement. In this study, we have used Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). These are types of RNN algorithms in which cells can handle long term memory, since input runs down the entire chain with some interactions at each state. Each LSTM cell contains three gates where non-linearity is applied to determine how much of prior cell states will be forwarded to the next cell. "Forget Gate" decides what information to delete from the previous cell state,"Input Gate" decides what new information will be stored in the current cell state and "Output Gate" decides what information will be output to by the current cell. GRU is a variation of LSTM which combines "input gate" and "forget gate" inside an LSTM, thus making it run faster.

Network Architecture In our deep learning architectures, we utilize an embedding dimension of 64 for the spatial coordinate data before feeding them into the LSTM and GRU networks (individually). We first input the coordinates to a dense layer, which maps the data to a higher dimensionality. Then, we use a hidden state dimension of 128 for the LSTM and GRU cells. Furthermore, dropout is applied to the dense layer, and the embedding layer uses ReLU non-linearity before the features are fed into the RNN cells. These hyperparameters were selected based on cross-validation with the publicly available datasets [1] from ETH, UCY and Stanford, and additional experimentation.

Finally, we used a learning rate of 0.0007 and the Adam optimizer for training the model. The deep learning models were implemented in Pytorch.

4.3. Comparison Metrics

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

4.3.1 Mean Square Error (MSE)

$$L(s_p, s_{gt}) = (s_p - s_{gt})^2 = (\frac{1}{m})\Sigma||\Delta s^i||_2^2$$

where $\Delta s^i = \begin{pmatrix} x_p \\ y_p \end{pmatrix} - \begin{pmatrix} x_{gt} \\ y_{gt} \end{pmatrix}$, $\begin{pmatrix} x_p \\ y_p \end{pmatrix}$ is the predicted next position of the pedestrian and $\begin{pmatrix} x_{gt} \\ y_{gt} \end{pmatrix}$ is the ground truth. We calculated the second norm of the difference between two vectors to evaluate the prediction of each pedestrian.

4.3.2 Average Displacement Error (meters)

The average displacement error was calculated by computing the euclidean distance between each step of the predicted trajectory and ground truth, further dividing it by the trajectory length.

$$D_{avg} = \frac{1}{l}\Sigma_i^l\sqrt{(x_p^i - x_{gt}^i)^2 - (y_p^i - y_{gt}^i)^2}$$

where l is the trajectory length, $(x_p, y_p)^i$ is the i^{th} predicted step and $(x_{gt}, y_{gt})^i$ is the i^{th} ground truth step of the pedestrian's trajectory.

4.3.3 Final Displacement Error(metres)

The final displacement error was calculated by computing the euclidean distance between the final predicted position and ground truth position of each pedestrian's trajectory.

$$D_{final} = \sqrt{(x_p - x_{gt})^2 - (y_p - y_{gt})^2}$$

where (x_p, y_p) and (x_{gt}, y_{gt}) are the predicted and ground truth final spatial positions of each pedestrian's trajectory.

5. EXPERIMENTATION

In order to chose optimum hyperparameters for the traditional machine learning algorithms and deep learning framework, selection curves were plotted for each hyperparameters with respect to cost. Hyperparameters for KNN regression include observed trajectory length (i.e. number of features) and K nearest neighbours. Hyperparameters for LSTM and GRU include the learning rate, embedding size, rnn size, dropout, observed and predicted trajectory lengths.

5.1. Hyperparameter selection for KNN

Before gathering results from the actual experiments, we executed hyperparameter selection for KNN by running different combinations of observed trajectory length and number of neighbours.

In order to choose appropriate features(i.e number of observed trajectory length) for input data we plotted number of features v/s train loss.

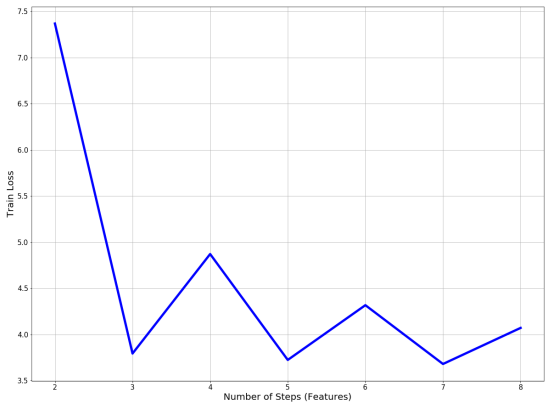


Figure 3. Feature selection for observed trajectory length

Based on observations from Fig 3., we opted for a trajectory length of 5 over 7 because the number of blocks formed for an observed trajectory length of 5 were found to be more than the number of blocks formed for an observed trajectory length of 7.

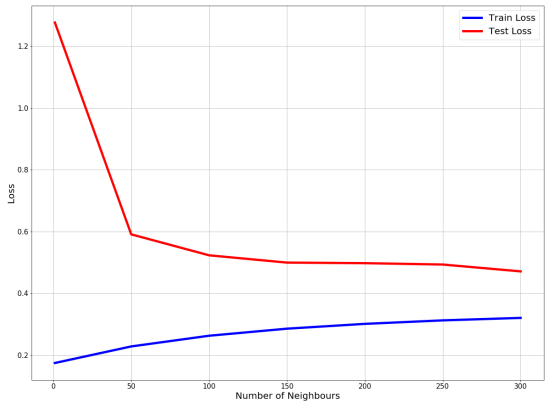


Figure 4. Loss v/s Number of Neighbours (K)

To find the optimum value of K we plotted a loss v/s number of neighbours graph to see for which value of K the train loss and test loss was minimum. With increase in the number of neighbours we observed that the training loss increases and the test loss keeps decreasing. By observing Fig.4 we found that the optimum value of K was 50.

5.1.1 Hyperparameter selection of RNN

Optimal values of all the hyperparameters of the LSTM and the GRU architectures were computed through cross-validation and training/testing on different architectures. The LSTM architecture was trained/tested on different learning rates with observed trajectory length of 3.2 seconds and predicted trajectory length of 4.8 seconds as shown in the Fig. Based on the results, an optimum learning rate of 0.0007 was chosen. It should be noted that a coarse-grain search was initially performed for searching for critical hyperparameters, such as learning rate and network architecture parameters. The results below highlight some of the narrowed-down hyperparameter search.

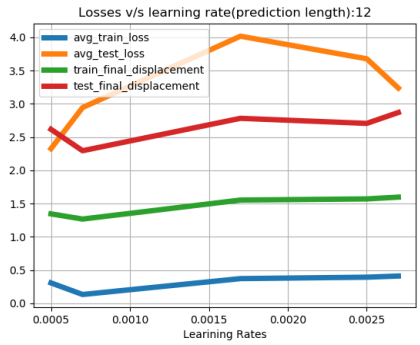


Figure 5. Fig. Errors/losses vs Learning Rate

The hidden state dimensions and rnn size were chosen using cross validation. Since, GRU is a modification of the LSTM cell, it shares the same optimum value for all hyperparameters. Recurrent Linear Unit (ReLU) non linearity activation on the input embedding layer along with a dropout of 0.5 gave better results. Additionally, different lengths for observed and predicted trajectories were experimented to gain a better understanding of dependence of these parameters on the prediction accuracy of the model.

6. RESULTS

After determining the optimum hyperparameters we gathered the following results:

6.1. Comparative Results

We compare the performance of the following methods at different observed and predicted trajectory lengths:

Table 1. Cross-validation results for GRU during hyperparameter search.

GRU: 8 obs. frames, 8 pred. frames			
Learning rate	0.0005	0.0007	0.0017
Avg train loss	0.1639	0.1491	0.1759
Avg test loss	1.7303	2.1536	2.8133
Avg train disp err.	0.5853	0.5529	0.5307
Final train disp err.	0.9363	0.8921	0.8583
Avg test disp err.	1.2648	1.3957	1.5666
Final test disp err.	2.1357	2.3529	2.5491

Table 2. Cross-validation results for Vanilla LSTM during hyperparameter search.

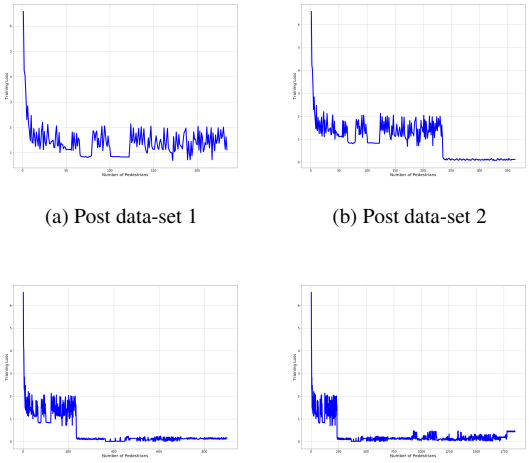
V-LSTM: 8 obs. frames, 8 pred. frames			
Learning rate	0.0005	0.0007	0.0017
Avg train loss	0.1699	0.1395	0.1582
Avg test loss	1.5720	2.0212	2.432
Avg train disp err.	0.5617	0.5424	0.5127
Final train disp err.	0.9363	0.9171	0.8201
Avg test disp err.	0.9489	1.1846	1.4291
Final test disp err.	2.0806	2.0178	2.5521

Machine Learning Algorithm	Number of Steps Predicted	Test Loss (MSE)	Final test displacement error (meters)	Average test displacement error (meters)
Linear Regression	1	8.7331	3.4498	–
KNN Regression	1	0.4182	0.7319	–
LSTM	8	2.0212	2.0806	1.1846
GRU	8	2.1536	2.3529	1.3957

- Linear and KNN Regression: Both the models predicted 1 step with an input of trajectory length 5.
- LSTM: This is the basic vanilla LSTM-based trajectory prediction method. This architecture was used to predict trajectories of length ranging from 2 to 12. Observed trajectories of lengths 2-12 was used for this architecture.
- GRU: This is another variation of RNN in which each cell stores only the cell state and not the hidden state. The architecture used for GRU was similar to the architecture used for LSTM. Observed trajectories of lengths 2-12 was used for this architecture.

6.2. KNN Regression-Results

Form Fig.6 it can be observed that training loss converges after 250 pedestrians. It can be inferred that this model is fully trained after it encounters data for 250 pedestrians.



(a) Post data-set 1 (b) Post data-set 2

(c) Post data-set 4 (d) Post data-set 7

Figure 6. Loss v/s Number of Pedestrians

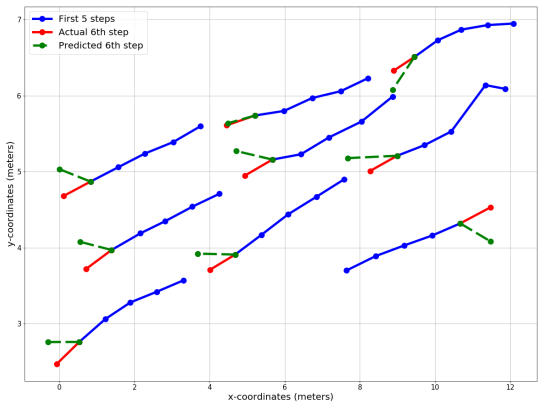


Figure 7. Actual and Predicted path of pedestrians

In Fig.7 we can see how the model predicts the pedestrian path.

6.3. Variation of Errors/Losses with Predicted Length

LSTM and GRU models were trained and tested for different predicted trajectory lengths keeping a constant observed observed trajectory length of 8 steps (3.2 seconds). As shown in the figure, the average train and test losses as well as average and final displacement errors continuously increase with increase in the predicted trajectory length.

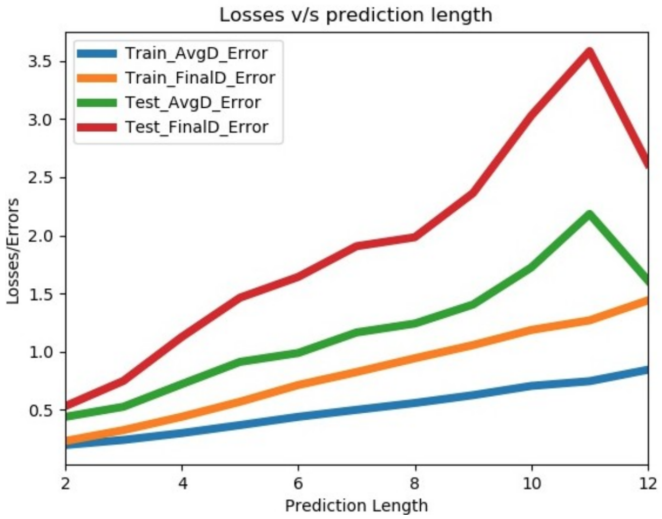


Figure 8. Errors/losses vs observed length

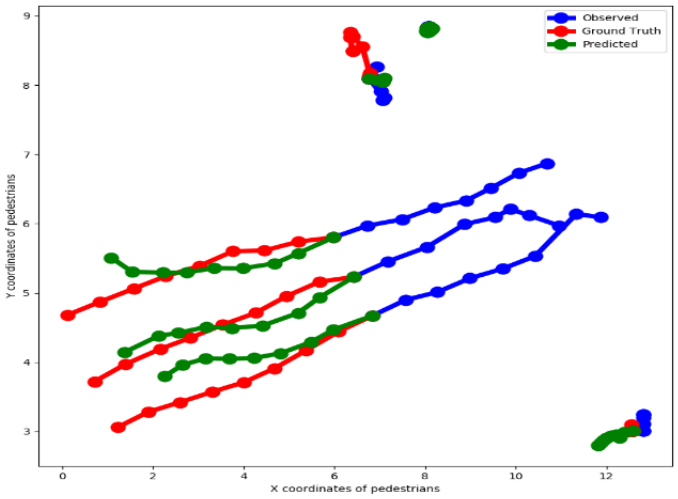


Figure 9. LSTM: Visualization of actual, predicted and ground truth path

6.4. LSTM and GRU visualization

The visualization of trajectories predicted by LSTM and GRU have been shown in Figure 9 and Figure 10 respectively. The observed trajectory is shown in blue, ground truth in red and predicted trajectory in green. As observable in the figures clearly, the models are capable of successfully predicting even the rare scenarios such as when the pedestrian is relatively stationary or traversing at a slower speed.

6.5. Variation of Errors/Losses with Observed Length

LSTM and GRU models were trained and tested on different observed trajectory lengths, fixing the prediction length to study its dependence on prediction accuracy. Fig.11 (a),(c),(e),(g) (i) show the variation of average test and train loss with observed trajectory length. Fig.11 (b),(d),(f),(h) (j) show the variation of train average displacement error, train final displacement error, test average displacement error and test final displacement error with the observed trajectory length.

KNN regression and the deep learning methods far outperformed the baseline, linear regression, for predicting the next 1-2 steps. Intuitively, the trajectories are non-linear and complex, so the poor performance of linear regression was expected. Though the GRU and LSTM networks predicted trajectories for at least 2 steps, they outperformed linear regression and KNN regression significantly. GRUs final displacement error was 0.0863, an order of magnitude better than KNN regression.

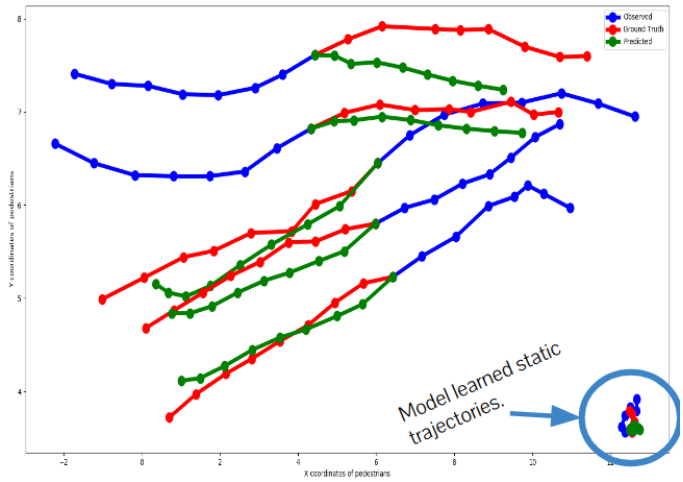
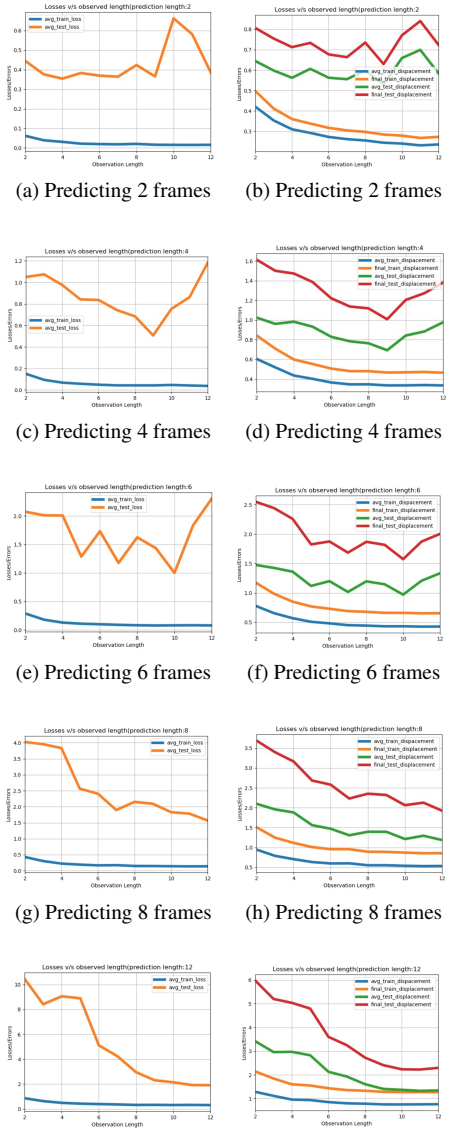


Figure 10. GRU: Visualization of actual, predicted and ground truth path

7. CONCLUSIONS

Unlike previous studies, the models formulated during this research have been generalized to multiple scene layouts and scenarios by training on seven datasets and testing on a new 8^{th} dataset. Both scene specific and generalized models were trained during the study. While the prediction accuracies were much better for the scene specific models, more attention was payed to the generalized models due to higher applicability in real-time prediction.



(a) Predicting 2 frames (b) Predicting 2 frames

(c) Predicting 4 frames (d) Predicting 4 frames

(e) Predicting 6 frames (f) Predicting 6 frames

(g) Predicting 8 frames (h) Predicting 8 frames

(i) Predicting 12 frames (j) Predicting 12 frames

Figure 11. Losses (left column) and errors (right column) with varying observed lengths of trajectories.

Linear and KNN Regression: Due to the high stochastic nature of individual pedestrian trajectory, it is intuitive that the data cannot be separated by a single hyper-plane. The sole purpose of performing linear regression was to set a baseline for the other algorithms. KNN regression for 50 nearest neighbours outperformed the baseline, linear regression, for predicting the next step with an observed trajectory length of 5.

LSTM and GRU: LSTM and GRU outperformed both the traditional machine learning algorithms for predicting any number of frames with the same or higher observed trajectory length. As show in the results section, the prediction accuracy is highly dependent on the observed trajectory length and gives best results at a particular value of observed trajectory length. 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.

FUTURE WORK

Better prediction accuracy of pedestrian trajectories could be achieved by incorporating additional parameters such as gaze direction to capture pedestrian intention, social interaction between neighboring pedestrians to capture crowd interaction in highly crowded scenarios. The nature of pedestrian trajectory being highly stochastic, more data with different scenarios and scene layouts is needed for better training of a generalized model.

ACKNOWLEDGEMENTS

We would like to thank Jagjeet Singh for his technical advice, especially in regards to the Social-GAN dataloader, implementing backprop for the trajectories, and several other discussions on project ideas. Also, thanks to Professor Farimani for teaching the course.

CONTRIBUTIONS FROM TEAM MEMBERS

Ashish was responsible for data-loader, RNN architecture, hyperparameter tuning, testing and visualization scripts, shell scripts (for running scripts on different parameters). Bryan was responsible for data-loader, RNN architectures, formulation of training scripts (bash scripts) and testing scripts. Pranav was responsible for Linear and KNN Regression, hyperparameter tuning, testing, data-loader and visualization. Yayati was responsible for Linear and KNN Regression, data-loader and visualization. All team members equally contributed for the poster and the final report.

References

[1] Alahi, A., et al.: Social LSTM: Human trajectory prediction in crowded spaces. In: CVPR. (2016)

[2] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. Computer Graphics Forum (Proceedings of Eurographics), 26(3), 2007.

[3] Gupta, A., et al.: Social GAN: Socially acceptable trajectories with generative adversarial networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018).

[4] S. Pellegrini, A. Ess, K. Schindler, L. Van Gool, "You'll Never Walk Alone: Modeling Social Behavior for Multi-Target Tracking", Proc. IEEE Int'l Conf. Computer Vision, 2009.

[5] C. Song, Z. Qu, N. Blumm, and A.-L. Barabasi. Limits of predictability in human mobility. Science, pages 10181021,2010

[6] K. Kitani, B. Ziebart, J. Bagnell, and M. Hebert. Activity forecasting.

[7] J. Wiest, M. Hoffken, U. Kresel and K. Dietmayer, Probabilistic trajectory prediction with Gaussian mixture models, in Proc. IEEE Intelligent Vehicles Symposium (IV), pp. 141-146, 3-7 June 2012.

[8] Akinori Asahara, Kishiko Maruyama , Akiko Sato , Kouichi Seto, Pedestrian-movement prediction based on mixed Markov-chain model, Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, November 01-04, 2011, Chicago, Illinois

[9] D. Ellis, E. Sommerlade, and I. Ried, Modelling pedestrian trajectory patterns with gaussian processes, in IEEE 12th International Conference on Computer Vision (ICCV) Workshops, 2009.

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

[11] 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.

[12] S.Hochreiter and J.Schmidhuber. Long short-term memory. Neural computation, 9(8):17351780, 1997.

[13] Trajectory Forecasting Benchmark. http://trajnet.stanford.edu/data.php?n=1