Predicting Human Trajectories in Multi-Class Settings

Karthik Raju (kraju444), Max Chang (mchang4)

December 17, 2016

1 Introduction

In recent years, autonomous navigation has become of increasing prominence with companies like Uber, Google, and Tesla building their state-of-the-art self-driving cars. There are other opportunities, however, for self-navigating vehicles in non-road settings, including social robots that can navigate crowded spaces. Examples include driving through malls, walking dogs through populated parks, or helping blind pedestrians navigate around others.

In navigating crowded areas, autonomous agents must adopt a set of common sense rules and social conventions in interacting with other objects in the same proximity. For example, when humans are deciding how to walk on a crowded sidewalk, they plan where to move next by considering their immediate neighbors' personal spaces as well as understanding who has the right-of-way. Similarly, autonomous robots should also adopt a model of movement that fluidly circumvents humans by predicting their trajectories and roughly complying with the same common sense rules as those they share space with. Additionally, a good model should also capture that interactions between a biker and a cart-driver are different than between a biker and a pedestrian or another biker. Hence, the model should account for different "classes" of scene occupants.

In this paper, we build a GRU, a recently explored RNN-variant that has been shown to be more computationally efficient and produce models comparable to that of LSTMs [10]. The GRU uses a gated architecture to solve problems that traditionally plague normal RNNs (namely the exploding and vanishing gradient problems). In the past decade, gated RNN-variants have championed sequence-learning problems and provide great promise for our approach to trajectory-learning. Our input to the neural network is a 25-frame sequence of an object's positions and it's neighbors' hidden states. The output is the object's position at the next frame. In our case, the hidden state of the GRU is equivalent to the object's past trajectory, hence the network's prediction is some function of the past trajectory.

2 Related Work

Predominantly, the approach has been to develop highly engineered settings that constrain movement (and hence capture social conventions). The Social Forces (SF) model by D. Hellbing and P. Molnar [3], where individuals respond to energy potentials of other individuals in a shared space, has been most prominent. However, models directly based on this, use a single set of parameters for all navigation models, even when multiple classes of targets are presented. It is easy to imagine how improving these models to capture more complex social conventions can be cumbersome.

More recently, works including Antonini et. Al's "A discrete choice pedestrian behavior model for pedestrian detection in visual tracking systems" [4] demonstrate a Discrete Choice Model for evaluating human trajectories in crowded spaces, while works like Wang et. Al's "Gaussian process dynamical models for human motion. Pattern Analysis and Machine Intelligence," [6] show Gaussian methods to predict such trajectories. These works, however, assume that the end location is previously known, which cannot work for autonomous trajectory prediction (for our intents, it is practically useless to predict a person's trajectory once traversed).

Most applicable to this project are Alahi et. al.'s "Human Trajectory Prediction in Crowded Spaces" [1] and Robicquet et. al's "Learning Social Etiquette: Human Trajectory Understanding in Crowded Scenes" [2]. The former uses LSTMs to model each target along with a social pooling layer, but only models pedestrian-pedestrian interaction. The latter paper, while accounting for different classes of objects, uses the older prediction methods of the aforementioned Social forces Model. Alahi et al's paper, however, has demonstrated that LSTM's are an effective way to model human interaction by capitalizing on the advantages of deep learning.

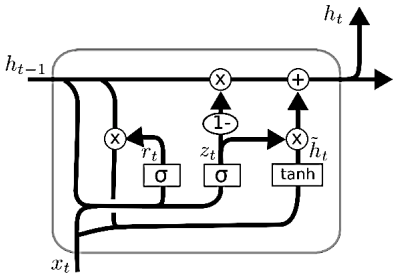

Figure 1: GRU diagram showing cell architecture with update and reset gate using tanh activator.[7]

3 Data Set

For our data, we use the Stanford Drone Dataset, taken and pre-annotated by Stanford's Computer Vision and Graphics Lab[12]. This data set captures 22 minutes of drone footage, observing the trajectories of 1548 bicyclists, 917 pedestrians, 107 skaters, and 132 carts. There are 4 classes: "Pedestrians", "Bikers", "Skaters", and "Carts". In this set, there are on average, 60 occupants (of all classes) per scene. All footage was taken above a roundabout, notably called the "Circle of Death" for the high level of traffic (see figure 2).

We use the first 17 minutes of footage for the training set, and the remaining 5 minutes for the testing set. Training data includes 1307 Bikes, 831 Pedestrians, 91 Skaters, and 111 Carts.

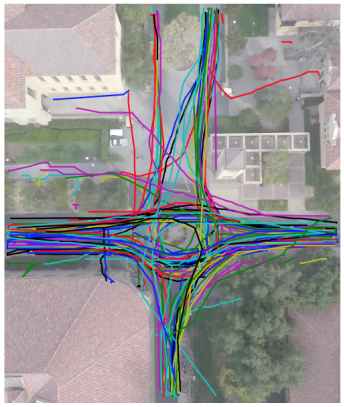

Figure 2: Scene trajectories superimposed onto the Circle of Death

4 Method

In our pooling GRU Model, each occupant trajectory is considered a sequence. Model Parameters are shared among all members of a class and are different for members of different classes. If the parameters are thought of as the learned navigation rules, this translates to members of a class sharing a set of navigation rules and are different from class to class. We define two baseline models for comparison:

Linear Model: For our initial baseline, we use a simple linear model and assume constant acceleration to find the next few points, that is:

$$d = v_i(t) + \frac{1}{2}at^2$$

d is the distance moved, v_i is the initial velocity, a is the acceleration and t is the time passed.

Naive GRU: As a baseline for our choice of the final model and justification for our use of GRUs, we start off initially with a GRU that predicts future trajectory by only considering the past trajectory. The Naive GRU approach is fundamentally a classic sequence generation. Because it only considers it's own trajectory and treats the path independent of all other objects in the scene, the model is dubbed "naive." The Naive GRU is trained with trajectories of a single class, producing weights that capture the rules for that particular class. We observe positions from T_0 to some T_{obs} and predict the next time step, $T_{obs} + 1$.

The GRU forward propagation is defined as follows:

(1) Embedding Layer with ReLU non-linearity for position, x_{t-1}, from dim(2) to dim(hidden).

$$x_e = W_e x_{t-1}$$

(2) Find update gate, z, and reset gate, r, given x_e and past hidden state, h_{t-1}.

$$z = U_z x_e W_z h_{t-1}$$

$$r = U_r x_e W_r h_{t-1}$$

(3) Calculate hidden state, s_t with the ReLU activator. [$\odot$ is the Hadamard Product].

$$h_t = ReLU(U_h x_e + W_h(h_{t-1} \odot r)$$

(4) Calculate the position prediction, x_t.

$$x_t = (1 - z) \odot h_t + z \odot h_{t-1}$$

Intuitively, the weighted equations above determine how to use the previous trajectory to calculate

the next time step (reset gate) and how much the next time step depends on the previous trajectory (update gate) [7, 10].

We assume a bivariate Gaussian distribution for the coordinates x and y [1, 3]. Hence, our model is parametrized by the means (μ_x and μ_y), standard deviations (σ_x and σ_y), and covariance factor (ρ).

$$(x, y)_t^i \sim \mathcal{N}(\mu_x, \mu_y, \sigma_x, \sigma_y, \rho)$$

The distribution parameters are determined by the past hidden state for each trajectory and time step. More concretely, for trajectory i and time step t, our model parameters are some linear transformation of the last hidden state to $\mathbb{R}^5$ by some parameter matrix W_p:

$$\begin{bmatrix} \mu_x \\ \mu_y \\ \sigma_x \\ \sigma_y \\ \rho \end{bmatrix} = W_p h_i^{t-1}$$

In training, the GRU weights are updated by maximizing the log liklihood of the predicted position (x_t^i, y_t^i) for some time step, t and trajectory, i. In other terms, we are minimizing the following expression (our loss term) [1].

$$-\sum_{t=0}^{T} logP((x, y)_i^t | \mu_{x_t}^i, \mu_{y_t}^i, \sigma_{x_t}^i, \sigma_{y_t}^i, \rho_t^i)$$

This choice for the objective is rationalized by considering the prediction. Since the future prediction is a function of the hidden state, the distribution of the future prediction will also be some function of the hidden state.

GRU with Pooling: We improve on the naive GRU model by allowing the GRUs to share their hidden states. We define the 200 X 200 pixels square centered upon a scene occupant as that occupant's immediate neighborhood. We assume trajectories to be independent of occupants that are too far away to have an influence. We divide the 200 X 200 neighborhood into 400 pooling-windows of size 10 X 10 pixels. If there are occupants in the same pooling window, then their hidden states are summed. This accounts for places of high-density such as couples or larger groups. We then construct a 20 x 20 x dim(hidden) pooling tensor, H_t^i, given the neighboring hidden states, to summarize the trajectories of an occupant's immediate neighbors.

$$H_t^i(m, n, :) = \sum_{j \in N^i} 1[(x, y)_t^j]_{mn} h_{t-1}^j$$

where N^i is the set of immediate neighbors for occupant i and $1[]_{mn}$ is an indicator variable that checks if a given coordinate is within pooling window (m, n) of the 20 x 20 pooling grid. The pooling tensor is flattened and embedded in a vector, $H_e \in \mathbb{R}^{dim(hidden)}$.

$$H_e = W_d H_t^i$$

The embedded pooled tensor vector is then concatenated to the embedded position vector and fed into the GRU.

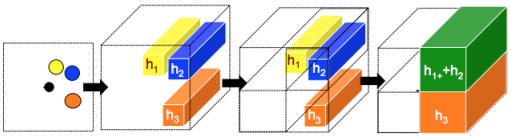

Figure 3: The top diagram exhibits pooling among a smaller example neighborhood. The hidden states are extracted, summed if applicable, and pooled to the black occupant [1].

5 Implementation

The GRU models has a hidden state dimension 128, input dimension 128/256 (Naive/Pooling) and output dimension 2. The GRUs use an ReLU activator. The use of a non-saturating activator such as ReLU has multiple benefits, one of which is decreasing the influence of vanishing gradients. We stacked two GRU layers to capture higher level complexities. The occupant neighborhood was set to 200 X 200 and used 10 X 10 pooling windows. Parameters were updated using Stochastic Gradient Descent with RMS-prop optimization. For regularization, we implemented dropout with value 0.5. Dropout can be thought of as a dramatic example of the ensemble learning technique, bagging. A visualization of the dropout implementation is provided in Figure 4. The initial learning rate was set to 0.003 and annealed upon increased loss. The model trained for 100 epochs. We use a custom Theano implementation.

To train the model, we allowed the GRU to observe 25 frames of a trajectory and predict the next time step. We utilized the entire trajectory.

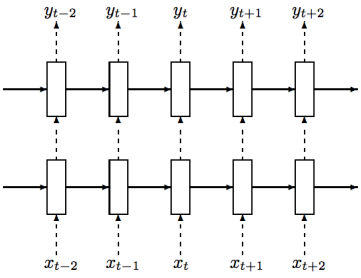

Figure 4: Each box represents a GRU layer. Solid lines (recurrent connections) are unaffected by dropout. Dashed lines are. Each element in vectors traversing the dashed lines have are set to 0 with probability 0.5. This allows the GRU to generalize and improve performance on unseen data.

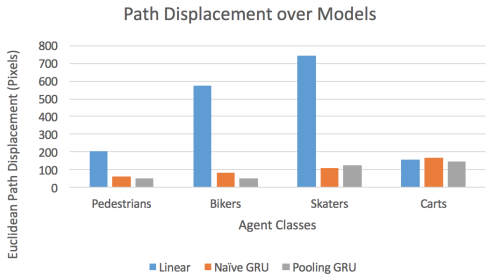

Figure 5: Path Displacement with respect to Agent Classes and Model Types

6 Evaluation

For testing, we employ holdout and evaluate the model on the last 5 minutes of the data set. From a quantitative perspective, we compare the models using two main metrics: path displacement and final displacement. We average the metrics over all trajectories in the test set.

1. Path Displacement: average euclidean distance between the prediction and true value at every time step (measured in pixels) across all test examples

2. Final Displacement: the euclidean distance (measured in pixels) between the last prediction and the corresponding ground truth.

Class Errors for Linear Model (Test Set)				
	Pedestrians	Bikers	Skaters	Carts
Path	203.5	574.2	744.98	**159.3**
Final	302.1	772.8	644.3	**108.6**

Class Errors for Naive GRU Model (Test Set)				
	Pedestrians	Bikers	Skaters	Carts
Path	60.6	82.4	110.4	170.2
Final	82.7	87.9	**179.8**	253.6

Class Errors for Pooling GRU Model (Test Set)				
	Pedestrians	Bikers	Skaters	Carts
Path	**50.5**	**52.5**	**123.2**	203.6
Final	**66.3**	**65.9**	184.8	245.7

Values in **bold** represent the highest path and final displacements per class.

7 Discussion and Analysis

From our results, we see that for Pedestrians and Bikers, there are significant improvements from the Linear Model to Naive GRU to Pooling GRU. We actually see a dip in performance between the Naive GRU and Pooling GRU for Skaters and Carts. Instead, the Linear Model scores substantially higher than that of the Naive GRU or Pooling GRU.

Analysis of the path trajectories suggest to us that carts fit a roughly linear motion, and is therefore exceptionally suited for the linear model. They do not swerve to the same degree as the other classes. This is supported by how carts demonstrate a higher average path error than final displacement error. Furthermore, smaller amount of training data available to carts and skaters must not have been enough to generalize as well as bikers (this will be discussed more later). Hence, these classes produced larger neural network training errors than bikers.

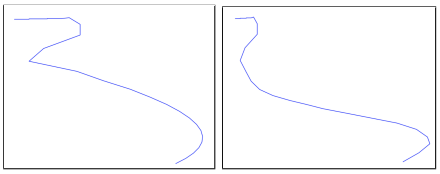

Figure 6: The left represents the Pooling GRU's prediction for the path. The right represents the ground truth for that same path. Plots are trajectories from birds-eye perspective.

From an initial analysis, we verify our hypothesis that firstly, deep learning perform substantially better than linear models. It was able to predict complex non-linear movements (see figure 6), which led to its ability to predict better metrics for pedestrians and bikers. In addition to how closely we are able to predict the trajectories, we wanted to see how closely the approach captures the aforementioned "social norms". After analyzing several trajectories within the same scene, we observed promising

signs of realistic interaction. For instance, we noticed that when a biker and pedestrian were about to collide, the pedestrian stopped for the biker before proceeding along its path. This accurately captures expected behavior, as in the circle of death, it is unusual for the biker to stop to let a pedestrian cross. High speed bikers possess the "right-of-way" and pedestrians must find gaps within of bike traffic.

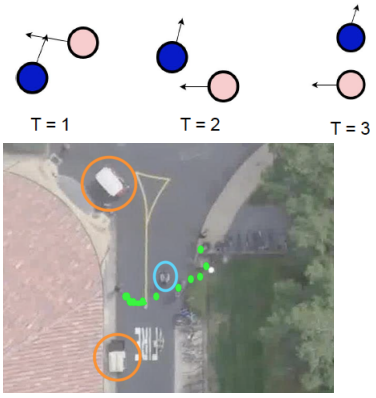

Figure 7: Top: blue represents Biker and red represents Pedestrian. Red allows blue to cross first. Bottom: real predicted trajectory for pedestrian crossing the street. Orange denotes Cart and Blue denotes Biker. Green dots denotes the GRU's predicted trajectory over the next few time frames. Sequential green dots are 16 frames apart.

We can see via Figure 7 that the GRU does not only capture conventions of position, but also speed. Note how initially the dots are clustered, before expanding outwards, indicative of how the pedestrian stopped to allow bikes and carts go by before crossing the street. The network has not only learned how to position occupants amidst oncoming traffic, but also how quickly actions should be taken to best avoid collision. In Figure 6, the network demonstrates that a Pedestrian should allow a biker to pass, but cross quickly and clear the road for other oncoming bikers. In training and testing, sequences contained positions equally spaced by 16 frames. The hidden state must therefore also have captured the distance between subsequent sequence elements. In addition, GRU expects the pedestrian to walk down the road slightly in order to circumvent the bike faster as it passes. However, this could also be influenced by the approaching cart.

From our observations, we note that neither of our GRU Models suggested evidence of overfitting and therefore high variance in our testing set, for bikes and pedestrians, although there is a noticable gap between skaters and carts. More than anything, this suggests to us that expanding our dataset to have more examples will bolster our results. More specifically, we note the following errors when run on our training set:

Training Errors for Naive GRU Model				
	Pedestrians	Bikers	Skaters	Carts
Path	63.4	79.4	95.6	120.3
Final	87.7	85.9	130.5	170.5

Training Errors for Pooling GRU Model				
	Pedestrians	Bikers	Skaters	Carts
Path	48.5	53.1	91.3	140.5
Final	59.7	62.4	113.5	160.3

After more in-depth analysis of the video, we noticed some movements that contradicted our assumptions. Namely, we would sometimes observe two people (who presumably know each other) actually walk towards another and stop to talk. Because of the structure of the pooling layers, our model predicted that their two trajectories would diverge.

Overall, these results met our expectations: linear models cannot capture the constant movement trying to avoid each other that occur in a crowd. Naive GRU's, while able to capture more complex dynamics, could only reason about the future trajectories through its past complex movements, whereas our final model: the Pooling GRU, was able to carry out the understanding of its neighbors and how to avoid them.

8 Conclusion and Future Work

Our results suggest to us that GRU's are an effective way to model trajectory movements, and that pooling layers seem to demonstrate substantial promise over naive GRU's. Because of social factors (like friendship), our current methodology of pooling cannot comprehensively predict trajectories. Given our previous discussion, but the nonetheless provides a stronger framework for understanding dynamics behind crowds. We also further substantiate that a data-driven approach to modeling holds significant advantage to assumed dynamics (i.e. GRU vs Linear).

Our main objective for future work is to first improve the prediction values for the skaters and carts. It would be better to collect a data set with more training examples for those two classes. We would also like to experiment with Reinforcement Learning Algorithms and do a qualitative comparison between how the agent moves about in a crowded space.

References

[1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-fei, and S. Savarese. Social LSTM : Human Trajectory Prediction in Crowded Spaces. Cvpr, 2016.

[2] A. Robicquet, A. Sadeghian, A. Alahi, S. Savarese, Learning Social Etiquette: Human Trajectory Prediction in European Conference on Computer Vision (ECCV), 2016.

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

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

[5] J. D. Hunter, "Matplotlib: A 2D graphics environment," Computing in Science and Engineering, vol. 9, no. 3, pp. 90–95, 2007.

[6] 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.

[7] "Recurrent neural network Tutorial, part 4 – implementing a GRU/LSTM RNN with python and Theano," in WildML, WildML, 2015.

[8] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," Computing in Science & Engineering, vol. 13, no. 2, Mar. 2011.

[9] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," arXiv e-prints, vol. abs/1605.02688, May 2016.

[10] K. Cho, B. Merrienboer, C. Gulcehre, Dzmitry Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, arXiv e-prints, vol. abs/1406.1078v3, Sep 2014.

[11] W. Zaremba, I. Sutskever, O. Vinyals, Recurrent Neural Network Regularization, arXiv e-prints, vol. abs/1409.2329v5, Feb 2015.

[12] A. Robicquet, A. Sadeghian, A. Alahi, S. Savarese, Learning Social Etiquette: Human Trajectory Prediction in European Conference on Computer Vision (ECCV), 2016.