Where is the World Headed? Trajectory Prediction for Interacting Agents

Nitin Kamra[1], **Hao Zhu**[2], **Dweep Trivedi**[1], **Ming Zhang**[2], **and Yan Liu**[1]
University of Southern California, CA, USA[1] | Peking University, Beijing, China[2]
{nkamra,dtrivedi,yanliu.cs}@usc.edu[1] | {hzhu1998,mzhang_cs}@pku.edu.cn[2]

Abstract

Trajectory prediction for scenes with multiple agents is an important problem for traffic prediction, pedestrian tracking and path planning. We present a novel relational neural network model to address this problem, which flexibly models interaction between agents by making fuzzy decisions and combining the corresponding responses with a fuzzy operator. Our approach shows significant performance gains over many existing state-of-the-art predictive models in diverse domains such as human crowd trajectories, US freeway traffic and physics datasets.

1 Introduction

Predicting trajectories of agents interacting with each other is a key challenge in many practical domains such as traffic prediction [17, 11], pedestrian tracking [1, 3] and path planning [13]. Modeling interactions is challenging because: (a) interaction is dynamically varying over time and it is hard to infer precisely when two agents are interacting, and (b) changes to trajectories resulting from interactions can be quite fuzzy. Since fuzzy decisions enter routinely into human interactions and are also useful to represent interactions in systems of inanimate entities, we present a novel Fuzzy Query Attention (FQA) mechanism to solve the aforementioned challenges.

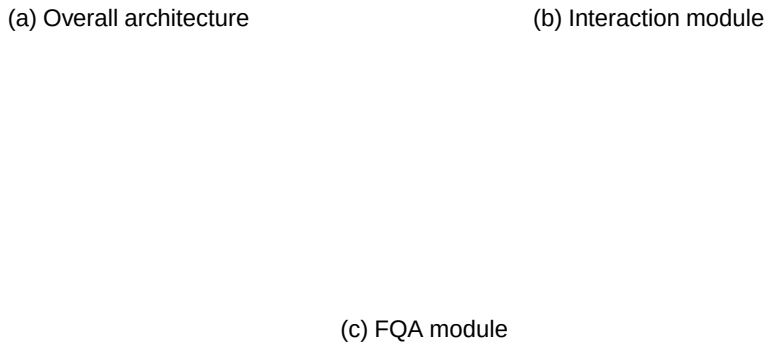
Problem formulation: In all subsequent text, p denotes positions and v denotes velocities. Given a scene comprising of N agents with positions $p_i^t = (x_i^t, y_i^t)\ \forall i \in [N]$ at a sequence of time-steps $t \in [T]$. The task is to observe all agents from time 1 to T_{obs}, infer their motion characteristics and ongoing interactions and predict their positions for time-steps $T_{obs}+1$ to T. We use p^t without a subscript for the positions of all agents at time t, while p_i without a superscript denotes the trajectory of agent i at all time-steps (similarly for velocity).

2 Fuzzy Query Attention model

Basic approach: FQA generates pairwise attention to decide about when two agents are interacting via dot-product based fuzzy decisions. It simultaneously learns how the agent under focus is affected by the influencing agent given the fuzzy decisions. We demonstrate significant gains over existing baselines on human crowds [1, 7], freeway traffic [6] and physics datasets [4, 10], thereby showing that FQA can model diverse kinds of interactions.

FQA architecture: Our architecture (Figure 1a) takes the spatial positions $p_{i=1:N}^t$ of all agents at time t and predicts the next step positions $\hat{p}_{i=1:N}^{t+1}$. An LSTM module whose weights are shared by all agents is used to maintain the hidden state (h_i^{t-1}) containing the past trajectory information for the i^{th} agent. We predict the next position of agent i by predicting a first-order linear velocity offset ($\tilde{v}_i^t$) augmented with a non-linear correction (Δv_i^t): $\hat{p}_i^{t+1} = p_i^t + \tilde{v}_i^t + \Delta v_i^t$. The $\tilde{v}_i^t$ term captures the linear velocity of the agent and the Δv_i^t captures agent's own intentional deviations from linearity plus effects due to interactions with other agents. The intentional deviations are captured by a preliminary update to the LSTM's hidden state: $h_i^{t-1} \rightarrow \tilde{h}_i^t$ (in figure 1a). For modeling interactions,

Preprint. Under review.



(a) Overall architecture (b) Interaction module

(c) FQA module

Figure 1: FQA prediction architecture

our interaction module (Figure 1b) generates edges between agents and (E) uses the FQA module to generate attentions (a) for each agent, which is used to update the LSTM state (h) and predict the velocity corrections (v_i) for each agent. The FQA module (figure 1c) generates yes-no responses ($V_{y;sr}$; $V_{n;sr}$) from sender-receiver features between agent pairs, combines them with a fuzzy if-else according to fuzzy decisions (D_{sr}) to model interactions between agents s and r, and aggregates the combinations to generate the required attention per agent.

3 Experiments

We report Root Mean Square Error between ground truth and predictions over all time steps on test sets of diverse benchmark datasets. FQA with 8 decisions outperforms state-of-the-art baselines by significant margins and accurately models diverse kinds of multi-agent interactions for trajectory prediction. Please refer to the appendix for detailed descriptions of FQA, all baselines, the datasets, experimental setup and more comprehensive results.

Table 1: Prediction RMSE for FQA, Social LSTM (SLSTM) and Graph Attention Networks (GAT)

Model	ETH-UCY		Collisions		NGsim		Charges	
SLSTM	0.690	0.013	0.211	0.002	6.453	0.153	0.485	0.005
GAT	0.575	0.007	0.237	0.001	6.100	0.063	0.524	0.004
FQA (ours)	0.540	0.006	0.176	0.004	5.071	0.186	0.409	0.019

4 Conclusion

We have presented a novel Fuzzy Query Attention (FQA) mechanism which models pairwise attention between agents by learning to make fuzzy decisions. While motivated from multi-head self-attention introduced in [15], the novelty of FQA lies in generalizing self-attention which applied to a single entity, to pairwise-attention which attends to an ordered pair (sender-receiver) of entities and captures the interaction effects of the sender on the receiver. Further our simpler row-wise dot-product structure fits easily on a single GPU (with 12GB memory), while still retaining the strong performance of the dot-product attention structure. We demonstrate significant gains over existing state-of-the-art baselines in diverse domains thereby demonstrating the potential of FQA.

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In IEEE Conference on Computer Vision and Pattern Recognition, pages 961–971, 2016.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261, 2018.

[3] Stefan Becker, Ronny Hug, Wolfgang Hübner, and Michael Arens. An evaluation of trajectory prediction approaches and notes on the trajnet benchmark. arXiv preprint arXiv:1805.07663, 2018.

[4] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In International Conference on Learning Representations, 2017.

[5] Benjamin Coifman and Lizhe Li. A critical evaluation of the next generation simulation (ngsim) vehicle trajectory dataset. Transportation Research Part B: Methodological, 105(C):362–377, 2017.

[6] Nachiket Deo and Mohan M Trivedi. Convolutional social pooling for vehicle trajectory prediction. In IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 1468–1476, 2018.

[7] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2255–2264, 2018.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems, pages 1024–1034, 2017.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[10] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In International Conference on Machine Learning, pages 2693–2702, 2018.

[11] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In IEEE Conference on Computer Vision and Pattern Recognition, pages 336–345, 2017.

[12] Yaguang Li, Chuizheng Meng, Cyrus Shahabi, and Yan Liu. Structure-informed graph auto-encoder for relational inference and simulation. In ICML Workshop on Learning and Reasoning with Graph-Structured Representations, 2019.

[13] Christoph Rösmann, Malte Oeljeklaus, Frank Hoffmann, and Torsten Bertram. Online trajectory prediction and planning for social robot navigation. In 2017 IEEE International Conference on Advanced Intelligent Mechatronics, pages 1255–1260. IEEE, 2017.

[14] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. In International Conference on Learning Representations, 2019.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, pages 5998–6008, 2017.

[16] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In International Conference on Learning Representations, 2018.

[17] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In IEEE Conference on Computer Vision and Pattern Recognition, pages 1345–1352. IEEE, 2011.

A Full description of FQA

Following previous work [3], we learn to predict offsets from the current position of each agent. However, we break the offset into a first-order linear velocity estimate and a correction term (see eq 1). This reflects the inductive bias that most inanimate objects and agents tend to travel with nearly constant velocity in the absence of external influence, since only the velocity correction term accounts for non-linearities and inter-agent interactions.

$$p_i^{t+1} = p_i^t + v_i^t + v_i^t; \quad \forall i \in 1:N \tag{1}$$

$$\text{(Linear velocity):} \quad v_i^t = p_i^t - p_i^{t-1}; \quad \forall i \in 1:N \tag{2}$$

$$\text{(Agent's Intentions):} \quad \hat{h}_i^t = \text{LSTM}(p_i^t; h_i^{t-1}); \quad \forall i \in 1:N \tag{3}$$

$$\text{(Interactions):} \quad h^t; \; v^t = \text{InteractionModule}(p^t; \hat{h}^t) \tag{4}$$

The first-order linear estimate of velocity v_i^t is made by a direct difference of agents' positions from consecutive time steps (eq 2). To compute the correction term v_i^t, a preliminary update is first made to the LSTM's hidden state using the incoming observation for each agent. This preliminary update captures the deviations from linearity due to an agent's own intentional acceleration or retardation (eq 3). The intermediate hidden state $\hat{h}_i^t$ and the current positions of all agents are further used to infer the ongoing interactions between agents, aggregate their effects and update the hidden state of each agent to h_i^t while also computing the correction term for the agent's velocity via an interaction module (eq 4). Since computation in all sub-modules happens at time t, we drop the superscript t from here on.

Interaction module: The interaction module (Figure 1b) first creates a graph by generating directed edges between all pairs of agents (ignoring self-edges). The edge E, the positions and the states of all agents are used to compute an attention vector a_i for each agent aggregating all its interactions with other agents via the Fuzzy Query Attention (FQA) module (eq 5). This aggregated attention along with each agent's current position and intermediate hidden state is processed by subsequent fully-connected layers to generate the updated state h_i (which is fed back into the LSTM) and the velocity correction v_i for each agent (eqs 6 and 7).

$$a = \text{FQA}(p; \hat{h}; E) \tag{5}$$

$$h_i = FC_2(\text{ReLU}(FC_1(p_i; h_i; a_i))); \quad \forall i \in 1:N \tag{6}$$

$$v_i = FC_4(\text{ReLU}(FC_3(h_i))); \quad \forall i \in 1:N \tag{7}$$

Fuzzy Query Attention: The FQA module treats each directed edge as a sender-receiver (s, r) pair of agents (Figure 1c). It generates independent features p_s, p_r, h_s and h_r for the senders and receivers by replicating p and h along each edge and also relative features $p_{sr} = p_s - p_r$ (relative displacement), $h_{sr} = h_s - h_r$ (relative state), $\hat{p}_{sr} = p_{sr} / \|p_{sr}\|$ (unit-vector along p_{sr}) and $\hat{h}_{sr} = h_{sr} / \|h_{sr}\|$ (unit-vector along h_{sr}). These features $f_{sr} = \{p_s; p_r; p_{sr}; \hat{p}_{sr}; h_s; h_r; h_{sr}; \hat{h}_{sr}\}$ are combined by single fully-connected layers to generate keys $K_{sr} \in \mathbb{R}^{n \times d}$ and queries $Q_{sr} \in \mathbb{R}^{n \times d}$ of dimension d each for every $s \neq r$ pair (eqs 8 and 9), which are then combined via a variant of dot-product attention to generate fuzzy decisions $D_{sr} \in \mathbb{R}^n$ (eq 10):

$$K_{sr} = FC_5(f_{sr}^{?}); \quad \forall (s,r) \in 1:N, s \neq r \tag{8}$$

$$Q_{sr} = FC_6(f_{sr}^{?}); \quad \forall (s,r) \in 1:N, s \neq r \tag{9}$$

$$D_{sr} = (K_{sr} \; ? \; Q_{sr} + B) = \left(\sum_{dim=1} K_{sr} \; Q_{sr} + B \right); \quad \forall (s,r) \in 1:N, s \neq r \tag{10}$$

where $\;$ represents element-wise product, $B, \in \mathbb{R}^n$ is a learnable bias parameter, σ stands for the sigmoid activation function and $\perp$ stands for the detach operator[1]. As a consequence of this formulation, $D_{sr} \in [0,1]^n$ can be interpreted as a set of fuzzy decisions capturing the interaction

[1] The detach operator acts as identity for the forward-pass but prevents any gradients from propagating back through its operand. This allows us to learn feature representations only using responses while the keys and queries make useful decisions from the learnt features.

between agents s and r. These can now be used to fuzzily select the receiving agent's response to the current state of the sending agent. For this, the sender-receiver features are parallelly parsed by two-layer neural networks (with the first layer having a ReLU activation) to generate yes-no responses $V_{y;sr}; V_{n;sr} \in \mathbb{R}^{n \times d_v}$ corresponding to D_{sr} being 1 (yes) or 0 (no) respectively (eqs 11 and 12). Though all these r features can be used here, our preliminary experiments showed that including only a subset of features (h_s and p_{sr}) gave comparable results and led to considerable saving in the number of parameters, so we only use this subset of features to generate the yes-no responses. These responses are then combined using fuzzy if-else according to decisions D_{sr} to generate the final responses $V_{sr} \in \mathbb{R}^{n \times d_v}$ (eq 13):

$$V_{y;sr} = FC_8(\text{ReLU}(FC_7(p_{sr}; h_s))); \quad \forall (s,r) \in 1:N, s \neq r \tag{11}$$

$$V_{n;sr} = FC_{10}(\text{ReLU}(FC_9(p_{sr}; h_s))); \quad \forall (s,r) \in 1:N, s \neq r \tag{12}$$

$$\text{(Fuzzy if-else): } V_{sr} = D_{sr} V_{y;sr} + (1 - D_{sr}) V_{n;sr}; \quad \forall (s,r) \in 1:N, s \neq r \tag{13}$$

Then final responses generated per agent pair ($\in \mathbb{R}^{n \times d_v}$) are then concatenated ($\in \mathbb{R}^{nd_v}$) and final responses from all senders are aggregated on the respected receivers by dimension-wise max-pooling to accumulate effect of all interactions on the receiver agents (eqs 14 and 15). Since max-pooling loses information while aggregating by selecting only the maximum feature values amongst all senders, we pre-process the final responses to increase the dimensions and retain more information followed by subsequent post-processing after aggregation to reduce the number of dimensions again (eqs 14 and 16):

$$V_{proc;sr} = FC_{11}(\text{concat}(V_{sr})) \tag{14}$$

$$V_{proc;r} = \text{maxpool}_{s:(s,r) \in E} V_{proc;sr} \tag{15}$$

$$a_r = FC_{12}(V_{proc;r}); \quad \forall r \in 1:N. \tag{16}$$

Strengths of FQA: While derived from multi-head self-attention introduced in [15], the novelty of FQA lies in generalizing self-attention which applied to a single entity, to pairwise-attention which attends to an ordered pair (sender-receiver) of entities and captures the interaction effects of the sender on the receiver. We simultaneously add a learnable bias to improve modeling power (explained below). Further, though the original matrix-dot-product structure of self-attention requires a large memory to fit even for regular batch sizes e.g. 32, our simpler row-wise dot-product structure fits easily on a single GPU (~12GB memory) for all datasets, while still retaining the strong performance of the dot-product attention structure.

What kinds of decisions can FQA learn? Since keys and queries are linear in the senders' and receivers' states and positions, the decision space of FQA contains many intuitive decisions for trajectory prediction, e.g.:

1. Proximity: FQA can potentially learn a key-query pair to be p_{sr} each and the corresponding bias as $-d_{th}^2$, then the decision $D = \sigma(p_{sr}^T p_{sr} - d_{th}^2)$ going to zero reflects if agents s and r are closer than distance d_{th}. Note that such decisions would not be possible without the learnable bias parameter B_r, hence having the bias makes FQA more flexible.
2. Approach: Since a part of the state h_t can learn to model velocity of agent s internally, FQA can potentially learn a key-query pair of the form $K_{sr} = v_{sr}; Q_{sr} = \hat{p}_{sr}; B = 0$ to model $D = \sigma(v_{sr}^T \hat{p}_{sr} + 0)$ which tends to 0 when the agents are directly approaching each other. While we do not force FQA to learn such human-interpretable decisions, our experiments show that the fuzzy decisions learnt by FQA are highly predictive of interactions between agents (section B.3).

Training: FQA and all our other baselines detailed in the next section are trained to minimize the mean-square error in predicting next time-step positions of all agents. Since many datasets involve agents entering and exiting the scene freely between frames, we input binary masks to all models for each agent to determine the presence of that agent in the current frame and control updates for that agents accordingly (masks not shown in Figures 1 and 1c to avoid clutter). All models are trained with the Adam optimizer [9] with a batch size of 32 and with an initial learning rate of 0.001

decaying multiplicatively by a factor $= 0:8$ every 5 epochs. All models are allowed to train for at least 50 epochs after which early stopping is enabled with a max patience of 10 epochs on validation set mean-square error and training is terminated at a maximum of 100 epochs. Since our goal is to test the models by observing T_{obs} (which was kept at $\frac{2T}{5}$ for all datasets) time-steps and make predictions until the remaining time T, we followed a dynamic schedule allowing all models to see the real observations for T_{temp} time-steps followed by $T - T_{temp}$ of its own last time-step predictions. During training, T_{temp} is initialized at T and linearly decayed by 1 every epoch until it becomes equal to T_{obs}. We found this dynamic burn-in schedule employed during training to improve the prediction performance for all models.

B Experiments

We perform multi-agent trajectory prediction on different datasets used previously in the literature with a diverse variety of interaction characteristics. For datasets with no provided splits, we follow a $70 : 15 : 15$ split for training, validation and test set scenes.

1. ETH-UCY [3]: A human crowds dataset with medium interaction density. We sampled about 3400 scenes at random from the dataset and set $T = 20$ following prior work [1, 7].
2. Collisions: Synthetic physics data with balls moving on a friction-less 2D plane, fixed circular landmarks and boundary walls. The collisions between balls preserve momentum and energy, while collisions of agents with walls or immobile landmarks only preserve energy but not momentum of moving agents. Contains about 94500 scenes with $T = 25$.
3. NGsim [5]: US-101 and i-80 freeway traffic data with fast moving vehicles. Since this dataset features very high agent density per scene (ranging in several thousands), we chunked the freeways with horizontal and vertical lines into sub-sections to restrict the number of vehicles in a sub-scene to less than 15. We sampled about 3500 sub-scenes from the resulting chunks and set $T = 20$.
4. Charges [10]: Physics data with positive and negative charges moving under other charges' electric fields and colliding with bounding walls. Contains 36000 scenes with $T = 25$ involving dense attractive and repulsive interactions.

We compare our FQA architecture with state-of-the-art baselines (see section C for architecture details and unique hyperparameters of all methods):

1. Vanilla LSTM [VLSTM]: An LSTM preceeded and followed by fully-connected neural network layers is used to predict the offset without considering interactions.
2. Social LSTM [SLSTM] [1]: Recurrent architecture which models interactions by discretizing space around each agent and aggregating neighbors' latent states via a social pooling mechanism.
3. GraphSAGE [GSAGE] [8]: Graph neural networks with node features to model interactions between agents. We use feature-wise max-pooling for aggregating the messages passed along the edges.
4. Graph Networks [GN] [2, 14]: Graph neural networks with node features, edge features and global features to model interactions between agents. We adapt the EncoderRecurrentGNN Decoder architecture from [14].
5. Neural Relational Inference [NRI] [10]: Uses graph neural networks to model interactions between agents and additionally infers edges between agents using variational inference.
6. Graph Attention Networks [GAT] [16]: Follows an aggregation style similar to GraphSAGE, but weighs messages passed from all sender agents via a learnt attention mechanism.

To evaluate and compare all models, we report the Root Mean Square Error (RMSE) between ground truth and our predictions over all predicted time steps for all agents on the test set of every dataset. The standard deviation is computed on the test set RMSE over five independent training runs of a method differing only in their initial random seed.

B.1 Prediction results

We present prediction results on all datasets for each model in Table 2. Our model with $n_d = 8$ decisions outperforms all the state-of-the-art baselines on all benchmark datasets (on many by

Table 2: Prediction error metrics for all methods on all datasets

Model	ETH-UCY		Collisions		NGsim		Charges	
VLSTM	0.576	0.002	0.245	0.001	5.972	0.065	0.533	0.001
SLSTM	0.690	0.013	0.211	0.002	6.453	0.153	0.485	0.005
NRI	0.778	0.027	0.254	0.002	7.491	0.737	0.557	0.008
GN	0.577	0.014	0.234	0.001	5.901	0.238	0.508	0.006
GSAGE	0.590	0.011	0.238	0.001	5.582	0.082	0.522	0.002
GAT	0.575	0.007	0.237	0.001	6.100	0.063	0.524	0.004
FQA (ours)	0.540	0.006	0.176	0.004	5.071	0.186	0.409	0.019

signi cant margins). This shows that FQA can accurately model diverse kinds of interactions. Speci cally, we observe that all models nd it dif cult to model sparse interactions on the Collisions data, while FQA performs signi cantly better with lower errors presumably due to its fuzzy decision making capability letting it decide when two agents are interacting (more detail in section B.3). Further, even though GAT also uses an attention mechanism at the receiver agents to aggregate messages passed from sender agents, FQA outperforms GAT on all datasets showing that FQA might have a better inductive bias towards modeling multi-agent interactions for trajectory prediction.

Further, as a side note, we point out that SLSTM [1] and NRI [10] both of which model interactions are often outperformed by VLSTM which does not model interactions. While surprising at rst, we found that this has also been con rmed for SLSTM by prior works, namely, Social GAN [7] which has common co-authors with SLSTM, and also independently by the TrajNet Benchmark paper [3]. We believe that this is because both methods introduce signi cant noise in the neighborhood of entities: (a) SLSTM does this via its summation of neighbors' hidden states within discretized bins which can potentially lose signi cant direction and motion speci c information, and (b) NRI infers many spurious edges during its variational inference encoding process which was also shown by [12].

B.2 Ablations

To show that it is indeed the fuzzy decisions attention mechanism which lends our model its strength, we present several ablations of our model.

Removing velocity correction: We rst remove the velocity correction term (v_i^t) from our predicted offset to show that this term which models deviations from linear velocity estimate is indeed required for accurate prediction. We call this model FQA_{lin} and Table 3 shows the stark deterioration in performance after the removal of velocity correction.

Removing decision making of FQA: Next, we replaced the keys, queries and decisions in FQA by a sequence of equivalent fully-connected layers so that responses are directly produced from input features without any fuzzy decisions, while retaining the subsequent concatenation and aggregation of responses. We call this variant FQA_{nodec} and show the deterioration in performance from loss of decision making in Table 3. It is clear that removing the decision making reduces FQA to the same or worse level of performance as other baselines on most benchmark datasets, and demonstrates that the strength of FQA comes from its fuzzy decision making process.

Table 3: Prediction error metrics with ablations and augmentations

Model	ETH-UCY		Collisions		NGsim		Charges	
FQA_{lin}	0.576	0.000	0.519	0.000	6.159	0.000	0.778	0.000
FQA_{nodec}	0.539	0.006	0.234	0.001	5.616	0.163	0.505	0.007
GN_{dce}	0.572	0.020	0.227	0.002	5.714	0.155	0.451	0.004
$GSAGE_{dce}$	0.579	0.011	0.231	0.001	5.901	0.099	0.456	0.005
GAT_{dce}	0.571	0.006	0.232	0.001	5.936	0.124	0.460	0.008
FQA_{dce}	0.532	0.002	0.175	0.004	5.814	0.170	0.416	0.001

B.3 Understanding fuzzy decisions of FQA

Distance-based cutoff for edges To check if FQA can learn fuzzy decisions to reflect proximity between agents, we replaced our edge generator to produce edges with a distance-based cutoff so it outputs a directed edge between agents s and r only if $\|p_s^t - p_r^t\|_2 < d_{thresh}$. The threshold d_{thresh} was found by a crude hyperparameter search and was set to $d_{thresh} = 0.5$ in the normalized coordinates provided to all models. We show prediction errors for FQA and other baselines namely GN, GSAGE and GAT[2] by providing them distance-constrained edges instead of all edges (-dc variants) in Table 3. While dc-variants of baselines show improvement in prediction errors on most datasets, FQA only shows minor improvements on Collisions which has sparse density of interactions, while the performance degrades on the other datasets with dense interactions. This suggests that FQA is indeed able to model proximity between agents even from a fully-connected graph, if the dataset is sufficiently dense in the number of interactions per time-step and does not require aiding heuristics, while other baselines do not necessarily extract this information and hence benefit from the heuristic.

Table 4: Predicting collisions from FQA decisions

	Accuracy	AUROC
1	95.55%	0.854
2	95.48%	0.866
3	95.35%	0.870
Recurrent	95.75%	0.907

(a) Collisions data: FQA models sparse interactions like inter-agent collisions well.

(b) Collsions data: FQA models stationary fixed landmarks well (blue) and predicts sharp collisions with walls.

(c) Charges data: Complex swirling in opposite charges (see pink and orange trajectories) accompanied by high accelerations; No model except FQA is able to predict such complex motion.

Figure 2: Predicted trajectories from all models shown with circles of radii increasing with time. The lighter shades show the observed part until T_{obs} while the darker shades show the predictions till T.

Predicting interactions from decisions To investigate if the fuzzy decisions capture inter-agent interactions well, we present an experiment to predict when a collision happens between two agents on the Collisions dataset[3] from only the 8 agent-pair decisions D_{sr}^t. Since collisions are sparse, we present the prediction accuracy and the area under the ROC curve on a held-out test set in Table 4 for various classifiers trained to predict collisions between agents using different horizon of time-steps ( ) of the input decisions. Note that we do not even use the agents' positions, velocities or the

[2] SLSTM already uses a neighborhood size of 0.5 for discretization, while NRI infers edges internally via variational inference.

[3] This is the only synthetic dataset for which the ground truth of interactions is available.

FQA responses (V_{sr}) as inputs to the predictors. Yet, the decision-trajectories alone are sufficient to predict collisions with a surprisingly high accuracy and AUROC, which strongly indicates that FQA's decisions are accurately capturing inter-agent interactions.

Visualization: Next we visualize the trajectories predicted by FQA and other baselines. Figures 2a and 2b show inter-agent collisions and those between agents and boundaries respectively. Due to agents' small sizes, inter-agent collisions are sparse events and only FQA learns to model them appropriately while the other baselines ignore them. Further FQA models the trajectories of agents faithfully and all collisions sharply while other baselines sometimes predict curved trajectories and premature soft collisions in empty space without any real interaction. We further observe from the pink and orange charges in Figure 2c, that it is hard to model chaotic swirling of nearby opposite charges due to high accelerations resulting from coulombic forces and that FQA comes closest to being an accurate model. More visualization examples are shown in section D.

C Model architectures and hyperparameters

We provide the hyperparameters of our baselines and those of FQA in this section. All our experiments were done on systems with Ubuntu 16.04 and all models trained using either Nvidia Titan X or Nvidia GeForce GTX 1080 Ti GPUs. All code was written in Python 3.6 with neural network architectures defined and trained using PyTorch v1.0.0.

C.1 Vanilla LSTM

The Vanilla LSTM model embeds each p_i^t to a 32-dimensional embedding vector using a fully-connected layer with ReLU activation. This vector is fed along with the previous hidden states to an LSTM with state size 64, whose output is again processed by a fully-connected layer to generate the 2-dimensional offset for next-step prediction.

C.2 Social LSTM

We adapted the code from https://github.com/quancore/social-lstm which directly reproduces the original authors' model from [1]. We kept the initial embedding size as 32, the LSTM's hiddden size as 40, the size of the discretization grid as 4 and the discretization neighborhood size as 0.5[4].

C.3 Neural Relational Inference

We adapted the authors' official repository from https://github.com/ethanfetaya/NRI . The input dimension was kept as 2 for positional coordinates and the number of edge types as 3 (since setting it to 2 gave worse results). The encoder employed the MLP architecture with hidden layers of sizes 32 and no dropout, while the GRU-based RNN architecture was used for the decoder with hidden state of size 32 and no dropout. The variance of the output distribution was set to 5×10^{-5}.

C.4 Graph Networks

While the original repository for Graph Networks is written in TensorFlow (https://github.com/deepmind/graph_nets), we translated the repository into PyTorch and adapted models similar to those employed by [1, 2]. We employed a vertex-level encoder followed by a recurrent Graph Network based on GRU-style recurrence followed by a Graph Net decoder. The vertex-level encoder transforms 2-dimensional positional input at each time step to a 10-dimensional node embedding. An input graph is constructed from these node embeddings having all pairwise edges and dimensions 10, 1 and 1 respectively for the node, edge and global attributes. This input graph along with a previous state graph (with dimensions 45, 8 and 8 for node, edge and global state attributes) was processed using a GRU-style recurrent Graph Network to output the updated state graph of the same dimensions (45, 8 and 8 for node, edge and global state attributes respectively). This new state graph was processed by a feedforward graph-network as prescribed in [2] to output another graph whose node features of dimension 2 were treated as offsets for the next time step prediction. All

[4] This neighborhood size is also the same as the distance cutoff used in section B.3.

update networks both in the encoder and the decoder (for node, edge and global features) used two feedforward layers with the intermediate layer having latent dimension 32 and a ReLU activation. While the original work proposes to use sum as the aggregation operator, we found summing to often cause the training to divergence since different agents have neighborhoods of very diverse sizes ranging from 0 to about 40 at different times in many of our datasets. Hence we used feature-wise mean-pooling for all aggregation operators.

C.5 GraphSAGE, Graph Attention Networks and Fuzzy Query Attention

Since GraphSAGE (GSAGE) [31] and Graph Attention Networks (GAT) [16] were not originally prescribed for a multi-agent trajectory prediction application, we used their update and aggregation styles in our own FQA framework to replace the FQA sub-module in our Interaction module described in Section 2. For all three methods the input size and the output size was 2, while the hidden state dimension of the LSTM shared by all agents was 32. The dimension of the aggregated attention for each agent a_i^t was also set to 32 for all three methods. All the three methods involved the FC_1; FC_2; FC_3 and FC_4 layers described in section 2 and had the output sizes 128, 32, 16 and 2 respectively.

GSAGE: GraphSAGE [8] directly embeds all sender latent vectors h_s^t into 32-dimensional embeddings via two fully-connected layers each with a RELU activation and with the intermediate layer of dimension 32. The output embeddings were aggregated into the receiver nodes via feature-wise max-pooling to generate a_i^t.

GAT: GAT performs a similar embedding of sender hidden states using a similar embedding network as GSAGE but aggregates them via feature-wise max-pooling after weighing the embeddings with attention head coefficients generated as proposed in [16] and finally averages over the 8 aggregations. We used 8 attention heads to match the number of FQA's decisions.

FQA: FQA used 8 query-key pairs for all datasets leading to 8 fuzzy decisions. The dimension for keys and queries was set to 4, while the dimension for yes-no responses was kept as 6. Consequently the dimension of learnt bias vector b_r was also 8 and the sizes of the fully-connected layers FC_5; FC_6; FC_7; FC_8; FC_9; FC_{10}; FC_{11} and FC_{12} described in the main text were 32; 32; 33; 48; 33; 48; 32 and 32 respectively.

D Additional visualizations

Here we show additional visualization from all models on all datasets. The visualizations clearly demonstrate the strong inductive bias of FQA for trajectory prediction.

Figure 3: Predicted trajectory visualization from various models on Charges dataset.