Human Trajectory Prediction Team: Final Report

Sokhna Seye, Saniya Srivastava, Anwitha Kollipara, Sunidhi Dhawan, Abhinav Vishnuvajhala, Steve Shi, Smit Shete, Shalin Jain, Ikhyun An

***Abstract*— An overview of all of the different weighting functions that the Human Trajectory Prediction team employed to improve the existing Egonet model.**

I. INTRODUCTION

This semester the Human Trajectory Prediction Team focused their efforts on improving the results of the existing EgoNet model. The central hypothesis driving our efforts was that weighting pedestrians by distance, angle, or a combination thereof would enable the model to focus attention on the most relevant agents, in turn improving prediction accuracy. To test this, each member of the team conducted in-depth investigations into a specific mathematically-grounded weighting scheme. The results of the experimentation conducted throughout this semester are summarized in this report, with each member highlighting their experience in perfecting one weighting function for the EgoNet model. By evaluating performance gains on key metrics over the baseline EgoNet, we can identify the most impactful weighting for reducing errors. In this report, we will analyze the outcome of each weighting function, provide comprehensive comparisons, and ultimately identify the best performing function.

II. DISTANCE WEIGHTING

Here we introduce the various distance based weighting function the Human Trajectory Prediction Team tested in order to improve the model.

A. Cosine Similarity

The Cosine Similarity weighting function is a function that makes use of the euclidean distances from the agent to apply weight to the encoder. The euclidean distance is simply the formula $\sqrt{x^2 + y^2}$ applied on the set of (x,y) pairs resulting in dist. We can find the max within the set, call this d_{max}. Then we can proceed to normalize each distance by dividing it by d_{max}. After that we apply cosine on the set of results. The final formula is below where dist is the current euclidean distance within the set.

$$val = cos(dist/d_{max})$$

The resultant value from the equation will provide higher weights to the vectors that are closer together and lower weight to those farther apart. The current formula will have a minimum weight of cos(1) since one of the values will be d_{max} in the numerator. In order to make this easier to contrast and provide consistent weighting among the past_sums we need a range of 0.00000001 to 1. A simple linear transformation via the formula below can be applied to val which is the result of the cosine function.

$$0.00000001 + 0.99 * ((val - cos(1))/0.46)$$

From the results, we can see that is does not perform better compared to regular trajectory weighting.

B. Exponential

The exponential distribution function utilizes an average calculated rate to make predictions about events that could occur. We used this distribution model to weigh each pedestrian based on the function's output given the pedestrian's distance. The equation for this distribution is given by:

$$\begin{aligned} f(x;\lambda) &= e^{-x}, x \geq 0 \\ &= 0, x < 0 \end{aligned}$$

Here lambda (λ) is the rate parameter which is a constant that represents the average rate of change in our pedestrian distances, where the reciprocal of λ is the mean or expected value:

$$E(x) = \frac{1}{\lambda}$$

We used the above property to calculate the rate parameter for the pedestrian inputs. By first calculating the euclidean distance for all pedestrians and then finding the mean we are left with the expected value for our input. By then taking the reciprocal of the mean we find our rate parameter. Going back to the original formula:

$$f(x;\lambda) = e^{-x}, x \geq 0$$

Utilizing our rate parameter as a constant we calculate weights based on this equation, where x is the euclidean distance for each pedestrian. We then multiply the weight into the input of the pedestrian trajectory encoder. This method of weighing aims to make the closer pedestrians have a higher weight, and those further away have a smaller weight. We can disregard the case where $x < 0$ as our euclidean distance will never be negative.

After running this weighing function on the Zara dataset we see that the minimum ADE and FDE are 3.291 and 6.224 respectively. Comparing this with the average ADE/FDE from the base model (3.545 and 6.709) we see that the exponential distribution function has shown an improvement in reducing error.

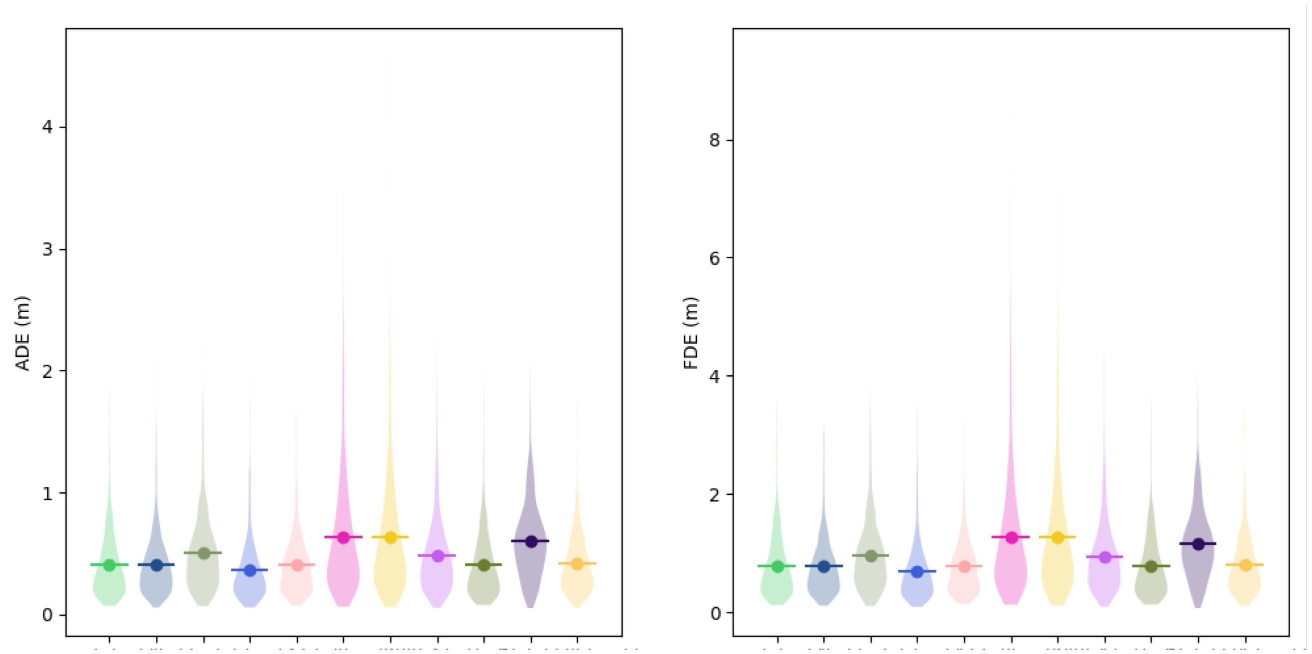

Fig. 1: Violin graphs of the various ADE and FDEs of Distance based Weighting functions, compared to an unweighted model (rightmost). The distance weighting functions from left to right are Cosine Similarity, Exponential, Inverse Distance, Gaussian, Unknown, Poisson, Sigmoidal, Augmented Cosine, Angular Similarity, and Field of View.

C. Gaussian

This weighting function prioritizes pedestrians that are closer to the egonet. Distance is determined by the Euclidean distance formula given by:

$$dist = \sqrt{x^2 + y^2}$$

Weights are assigned based on the following equation:

$$W_{gaussian} = e^{-(\frac{dist}{\sigma})^2}$$

Where $dist$ is the Euclidean distance and σ is the standard deviation of the gaussian curve. This weighting function is based on the probability density function of the gaussian distribution, given by the equation:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5(\frac{x-\mu}{\sigma})^2}$$

Where μ is the mean of the gaussian curve, σ is the standard deviation, and x is the point in question. In the weighting function, our x is the distance, our distribution is centered at zero, and we multiply the function by $\sigma\sqrt{2\pi}$. This is done so that a pedestrian with a distance of 0 from the egonet has the highest weight of 1, and a pedestrian far away will have a distance closer to, but not equaling, 0.

The value of σ is chosen through trial and error. After training on the Zara dataset with 4 different sigma values (shown in Figure 2), it was determined that $\sigma = 2$ produces the best results.

This distance weighting performed better than its unweighted counterpart when trained on the Zara dataset, with a mean ADE of 0.407 and a mean FDE of 0.781.

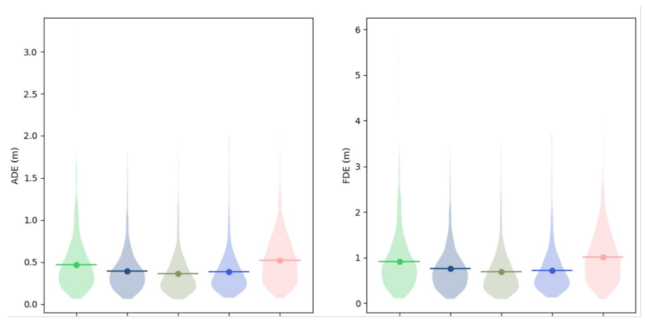

Fig. 2: Violin graph of The ADE and FDE of several trained gaussian weighting functions with different sigma values. From left to right sigma is equal to 1, 1.5, 2, 2.5. The last violin graph is non-weighted.

D. Inverse Distance

The inverse distance function weights the features from the neighboring pedestrian trajectories based on their immediate distance from the ego agent. The weight is inversely proportional to the immediate distance, resulting in the features extracted from closer pedestrians having a higher weighting with the intuition that the ego agent's future trajectory will be conditioned more strongly on the pedestrians closer to it. Given that the ego agent's position at the current timestep is (0,0) the weight is calculated as follows,

$$W_{pedestrian} = \frac{1}{1 + (x_0^2 + y_0^2)}$$

where x_0, y_0 is the position of the pedestrian at the current timestep.

E. Poisson

The Poisson distribution function was chosen as another weighting function because it is known to be very applicable in real-world situations. The Poisson distribution has only one parameter, λ, that defines both the mean and variance of the distribution. This makes the distribution very convenient to work with mathematically. The formula for the Poisson distribution function is represented mathematically by:

$$W_{poisson} = \frac{\lambda^x}{x!} * e^{-\lambda}$$

The x value is derived from the distances, while λ is chosen through trial and error. Distance values are obtained by taking the square root of the horizontal and vertical distances of a point from the origin squared. With further experimentation, perhaps a better lambda value, one that would produce results with less error, could be identified. However, with the best checked lambda value of 1, the error was quite similar to that of no applied distribution. It can therefore be concluded that the Poisson distribution is not the best distribution to apply to normalize the points in this dataset.

F. Sigmoidal

The sigmoidal distribution is a probability distribution that takes on an "S" shaped curve, with the weights transitioning from low to high around a central midpoint. In this weighting function, a sigmoidal weighting is defined and applied when summing the encoder past trajectories.

This sigmoidal weighting function assigns priority to pedestrians based on their distance from the egonet. Weights are calculated using the sigmoid function:

$$W_{sigmoidal} = \frac{1}{1 + e^{-slope*(dist-midpoint)}}$$

Where dist is the distance, slope controls the steepness of the sigmoid curve, and midpoint shifts where the curve crosses 0.5 on the y-axis. The distance is determined using the Euclidean norm. The Euclidean norm in this function calculates the distance of a point represented by x[i, :] from the origin (0, 0, ..., 0) in a multidimensional space using the square root of the sum of squared elements of the vector.

This function outputs a weight between 0 and 1, with 1 being assigned to a pedestrian at the chosen midpoint distance. As distance increases or decreases from the midpoint, the weight decays towards 0.

The slope and midpoint hyperparameters were tuned through trial and error. After some experimentation during training, a slope of 0.75 and midpoint of 0 was selected. When evaluated on the Zara dataset, this weighting function yielded a mean ADE of 0.406 and a mean FDE of 0.774.

III. Angle Weighting

Here we introduce the various angle based weighting function the Human Trajectory Prediction Team tested in order to improve the model.

A. Augmented Cosine Weighting

This weighting of the cosine of the angle between the robot vector (from robot to goal) and pedestrian vector (current pedestrian trajectory). The pedestrian vector is calculated using locations at past and present timestamps. A value of 1 is added to the denominator of the formula to avoid division by 0. The weighting is high for a low angle (close to 0) since it indicates that the paths of the robot and pedestrian are aligned and likely to collide. The weighting is low for a high angle since it indicates that the robot and pedestrian are headed in different directions. The best results are obtained when this angle weighting is combined with a good distance weighting. This is to account for pedestrians that have a low angle weighting (angle between its trajectory and robot is high) but are very close and in front of the robot since that also indicates high possibility of collision.

$$robotvector \frac{pedestrian}{|robot||pedestrian|} + 1$$

B. Angle Similarity Weighting

This angular weighting model calculates the weights of each pedestrian trajectory by taking both angle and distance into account.

This model iterates over each pedestrian vector, calculating the Euclidean distance from the robot vector, and then applying a Gaussian distribution-based weight to the distances. The standard deviation of the Gaussian is a parameter that affects the model's accuracy, and was tuned to "1" for this particular model. Like other distance models, this one assigns a weight which is inversely proportional to Euclidean distance from the robot.

Next, the model computes angular similarity by calculating the cosine angle (similar to cosine similarity index) between the robot vector and each pedestrian vector. Then, the cosine of the angle is clamped in the range from -1 to 1, and then the angle is re-calculated. After, another Gaussian distribution is applied. Similar to standard deviation of the first Gaussian, the model parameters for the second are the target angle (highest priority) and the range of consideration (what should the model consider). Logically, for this model, the target angle was tuned to 0 degrees, as the robot should prioritize pedestrians most directly in front of its trajectory. Secondly, the range of consideration was tuned to 90 degrees, as this provides a balance between being able to recognize significant deviations from the target angle in front of the robot while rejecting changes in vectors behind the robot.

Ultimately, the final weighting equation is simply taking the average of the two weighting models, as both are normalized to ranges of 0 to 1.

The distance weighting is:

$$W_g = e^{-(dist)^2}$$

where Euclidean distance is simply:

$$dist = \sqrt{x^2 + y^2}$$

The angularity similarity weighting is:

$$W_{diff} = e^{-(a_{diff})^2}$$

where a_{diff} is:

$$robotvector \frac{pedestrian}{|robot||pedestrian|} + 0.001$$

C. Field of View Filtering

We filter out pedestrians by simulating the field of view of the ego agent to select trajectories that the ego agent would realistically be able to reason about. First, we model the current positions of each current pedestrian as cells in an occupancy grid, with the position of each pedestrian being occupied. Next, we define the field of view of the ego agent as a 135 degree angle centered on the ray from the ego agent's current position to the goal position. Finally we filter with the following algorithm,

For each neighboring pedestrian position at the current timestep,

- Test if the line from the ego agent to the pedestrian's position is uninterrupted in the occupancy grid. If the line is interrupted, weight its encoded trajectory features with 0
- If the line is not interrupted, check that the pedestrian position is within the 135 degree field of view. If the position is not in the field of view, weight its encoded trajectory features with 0.
- Otherwise, the pedestrian is in the field of view, weight i's encoded trajectory features with 1.

IV. Conclusion

In conclusion, the Human Trajectory Prediction Team used many methods to improve the performance of the Egonet Model, and successfully found several weighting functions that performed better that it's unweighted counterpart.