Stochastic Path Forecasting from Expert Demonstrations

Nikhil Mohan
`nikhilm@andrew.cmu.edu`

Devesh Walawalkar
`dwalawal@andrew.cmu.edu`

Abstract

Path Forecasting is an active research problem which has seen multiple different methods applied to it. We attempt to approach this problem using Deep Convolutional Neural networks and recurrent networks like LSTMs to predict probable future locations an entity could take.We focus our attention on stochastic path forecasting for an autonomous vehicle with the entire environment simulated through the comprehensive CARLA simulator. We collect our own data using this simulator for the purpose of sampling expert path locations. We also train our models on the publicly available KITTI odometry dataset.

1 Introduction

For our project we attempt to implement a version of (1) R2p2 paper to forecast future possible paths of a vehicle from a single view frontal image. The algorithm predicts future x and y positions assuming the current position of the car is at the origin. This is an important task as it is needed as a preliminary step in precise path planning systems that can operate safely. Path planning is can be though of as a decision making problem, where the planner picks the best possible path from a set of plausible paths. Hence, we want the forecasting model to have two main characteristics. We want the output to be stochastic, i.e we want our model to generate a distribution over possible paths, and we want the output to be precise, i.e we want the generated distribution to be plausible and realistic as future downstream tasks need a good set to choose from. Due to the nature of the problem and the limited availability of datasets we run experiments on both simulated and some real world data. We collect our own data using the CARLA simulator (2) which helps us simulate different lightning and weather conditions. For real world data purpose we make use of the comprehensive KITTI odometry dataset. This dataset consists of images collected through a sensor mounted on top of a vehicle which is being driven along streets and different topological regions. Considering the varied conditions and drive environments that we can make with the simulator, we propose two different methods - one for the simulated environment and one for the real data. Our objective formulation is motivated by (1).

2 Related Works

Visual Prediction looks at predicting future actions from the current and possibly previous inputs. For many different sequential prediction tasks recurrent neural networks have shown to have state of the art performance. Long Short Term Memory blocks (3) have especially proven to handle long sequences well showing that longer patterns were able to be created without diverging from good outputs. We hence employ these blocks in our models to predict possible future positions where a long rollout is required.

Activity Forecasting is about predicting categorical activities. In (4; 5; 6),future activities are predicted via classification-based approaches. In (7), a first-person camera wearer's future goals are forecasted with Inverse Reinforcement Learning (IRL). IRL has successfully been applied to

Preprint. Work in progress.

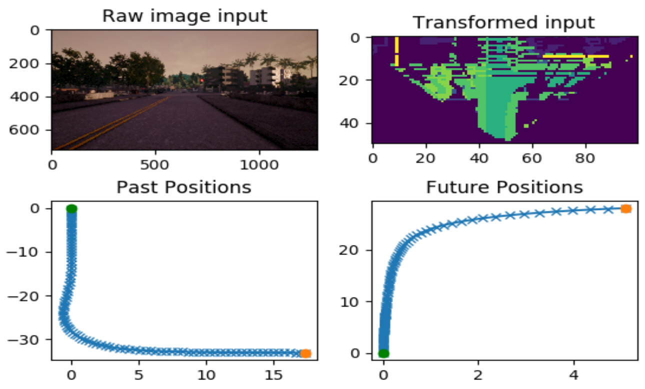


Figure 1: Data collected using the CARLA simulator (2)

predict and control robot, taxi, and pedestrian behavior (8; 9; 10).

Imitation Learning can be used to frame our problem: learn a model to mimic an agent's behavior from a set of demonstrations (11). One subtle difference is that in forecasting, we are not required to actually execute our plans in the real world. IRL is a form of imitation learning in which a reward function is learned to model demonstrated behavior. In the IRL method of (12) a cost map representation is used to plan vehicle trajectories. However, no time-profile is represented in the predictions, preventing use of time-profiled metrics and modeling. GAIL (4; 13) is also a form of IRL, where a GAN framework is combined with an IRL problem.

3 Proposed Idea

We model our objective as symmetric cross entropy between the samples seen from the expert and the generated samples from our model. This is shown in equation 1 where Ψ is the input to our system.

$$min_\pi - (E_{x\sim p(.|\psi)} log(q_\pi(x|\psi)) +_{x\sim q_\pi(.|\psi)} log(\hat{p}(x|\psi))) \tag{1}$$

The first term corresponds to the forward cross-entropy and encourages the model to cover the expert demonstrations as it will penalize points on the expert trajectory that are have low probability under our models output. The second term is the backward cross entropy and penalizes our model for generating trajectories that are highly unlikely given $\hat{p}$ which is a simple approximation of p as we do not have access to the true expert distribution.

$$s_t = \mu_t + \sigma_t z_t \tag{2}$$

$$min_\pi - (E_{x\sim p(.|\psi)} \frac{q_0(g_\pi^{-1}(x|\psi))}{|det(J_{g_\pi}(g_\pi^{-1}(x|\psi))|} + E_{z\sim q_0(.|\Psi)} \hat{p}(g_\pi(z|\psi))) \tag{3}$$

There are two problems that arise immediately from the given loss function, first we need to be able to evaluate q_π at every position x and the second term is not-differentiable as sampling is a non-differentiable operator. Due to these problems, the reparameterization trick is employed. We create our output by sampling from a noise distribution which we know the underlying distribution, to create possible future paths as shown in equation 2. This can be thought of as a push-forward distribution, which allows us to move the predicted points to future time-step predictions. Specifically for each rollout our model predicts a mean and standard deviation.

Figure 2 illustrates how learning takes place in the following algorithm. When the learning generates a given path the likelihood under the expert demonstration is computed. This likelihood is then changed to match the expert behaviour such that better paths are made more likely and bad paths are made less likely.

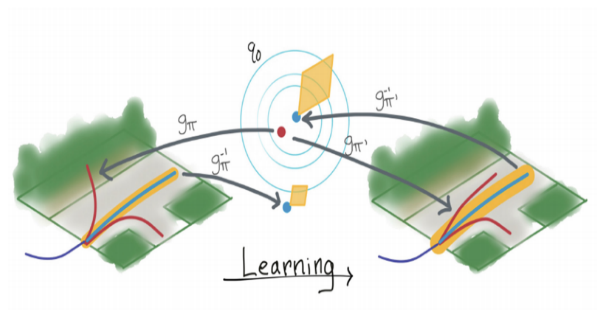


Figure 2: Illustration of how probability of paths are changed. (1)

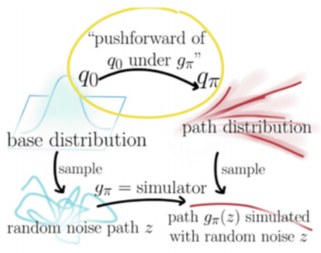


Figure 3: Illustration of bijection between noise distribution and possible trajectories taken from (1)

For our application we use $z \sim q_0 = N(0, I)$. This is illustrated in Figure 3 (1) which maps random simulated noise to a possible future path. We can hence rewrite our objective function as shown in equation 3. This makes the model differentiable with respect to the objective and can hence be trained end to end. From the re-written loss function we can see that all the terms can be easily computed given our learnt bijection as we know the true underlying distribution q_0. Equation 4 and 5 show how the numerator and denominator term can be evaluated for the forward cross-entropy loss. For our implementation at each roll-out position t the log likelihood loss is computed according to equation 4. We call this point z_t^e. Intuitively this translates to how likely we were to cover the expert point given our predicted mean and variance. We then regularize the σ values to make sure the model does not simply try to take this value to ∞ such that the loss decreases to 0. Since we need to compute the inverse of our σ matrix, we need to make sure it is not singular. One option is to have the model output 4 values for σ and take the matrix exponential of these values. We chose to ensure invertibility by simply predicting 2 values and interpret them as being elements of a diagonal matrix. This way computing the inverse is also a lot more computationally efficient.

$$g_\pi^{-1}(x_t|\psi) = \sigma_t^{-1}(x_t - \mu_t) = z_t^e \tag{4}$$

$$det(J_{g_\pi}(g_\pi^{-1}(x|\psi)) = \sum_{t=1}^{T} |det(\sigma_t)| \tag{5}$$

Figure 1 shows how we transform our input in the simulated CARLA environment. The top left image corresponds to the raw image input. In the simulator we also have access to the depth and segmentation data. We transform the segmentation input using depth data to create an overhead semantic map as shown in the top right image. The past position input to our model can be seen in the bottom right image where the current position is given in green, and the starting position is in orange. The bottom right image corresponds to the ground truth path our expert demonstrator took. We normalize all trajectories using the orientation information provided to us by the simulator. We found that while the idea of using an overhead map was appealing as it made intuitive sense, in the real world this data is often very noisy and is often missing. We found that this actually deteriorated performance and hence we went back to using just the raw image as input. We think it is because of the sparseness of the data. Without the presence of lidar data input, it is difficult to create an overhead map that contains a sufficient number of data points. Hence, we would like to further experiment about how we can get this to actually work on the simulator.

Using just the raw image also allows us to test our models on unseen data sets as well to check the generalization capability across many different datasets that contain images. Due to this we also create two models, one that takes as input both the past state positions and an image, whereas the other only takes as input an image.
We would like to explore the idea of using an overhead map in the future as extending our model to this new input is trivial with new methods such as (14) which we have implemented for the purposes of this project, in case we would like to switch to this method in the future in the presence of lidar input.

4 Model Architecture

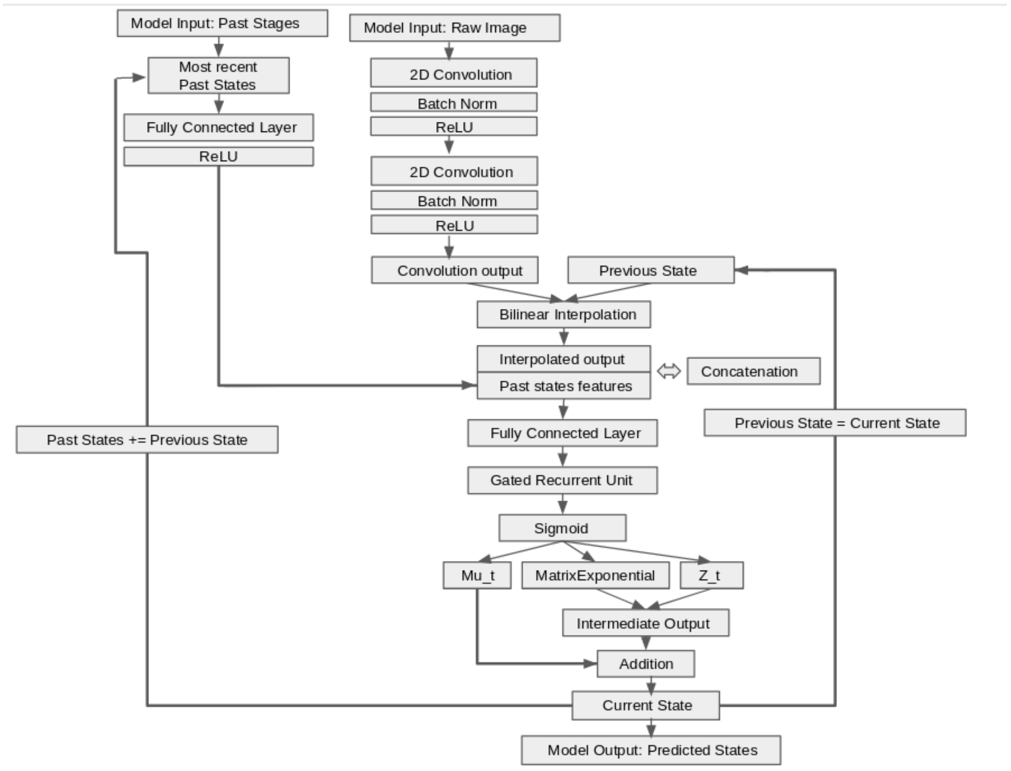


Figure 4: Model Architecture for CARLA dataset

In figure 4 we illustrate our first model which uses both past positions and the current image as input. We run the input image through a two layer convolutional neural network. We then interpolate the output feature maps with the previous state to get a mixed information output. The previous state in this case is the predicted state from the previous time step. Ideally for the first timestep we use the initial start location of the trajectory as the past predicted state. In Parallel, we pass the input past states from N timesteps through a fully connected layer. We concatenate this output with the bilinear interpolation output and pass them as hidden states of a GRU decoder. This recurrent network gives us the sigma and mean of predicted future path normal distribution. We sample a point randomly from this distribution and consider that as the next predicted path point. We then compare this predicted point with the expert path point and compute the log likelihood as our loss for that time step. We use this model for the generated CARLA dataset.

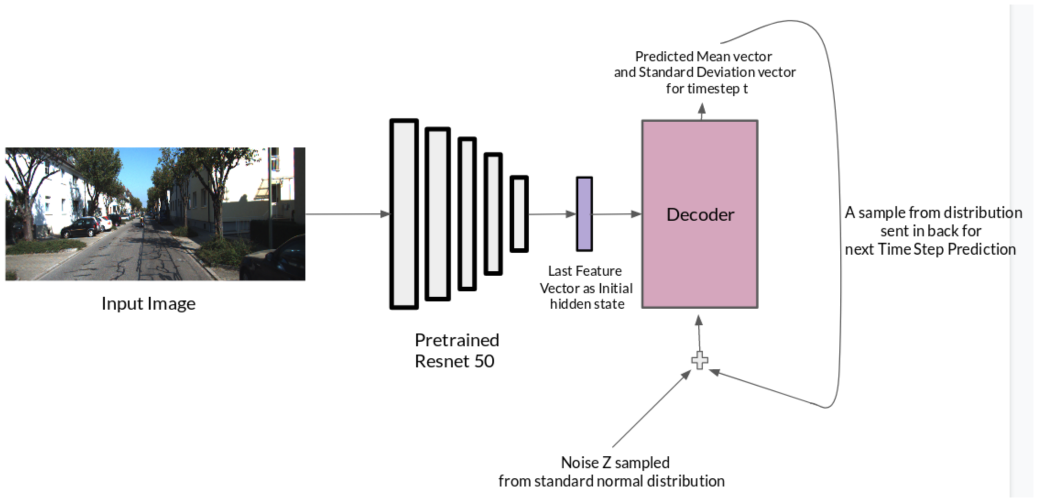


Figure 5: Model Architecture for KITTI dataset

In figure 5 we illustrate our second model which uses only the current image as input. We pass this image through a pre-trained Resnet50 architecture. We use the last feature vector embedding as the initial hidden state for the decoder. The decoder consists of a two layer GRU with a fully connected layer converting the GRU outputs to two dimensional mean and standard deviation vectors which define the normal distribution for the next predicted time step. We combine the previous time step predicted mean and standard deviation vectors with noise Z sampled from a normal distribution. We feed this as the input to the decoder for a particular time step. We repeat this for N time steps. We sample a point from the predicted distribution and compare it with the expert path point to compute the log likelihood as our loss for that time step. We use this model for the KITTI odometry dataset.

5 Dataset Implementation

5.1 CARLA Dataset generation

We configure the CARLA simulator (2) to run for 20 episodes in total, by capturing certain sensor values at every 10 milliseconds. We capture data from RGB sensor located at the front of the car. We capture the current global location coordinates of the car for each time step. Thus for each time step we are able to produce a list of N previous locations and N future location which we label as the expert future locations. The car is in fully autonomous mode implemented by CARLA simulator using certain search algorithms over all possible future path trajectories. We model different weather conditions (cloudy,rainy,sunny etc.) and different daytime conditions (morning,evening,afternoon and night) to create a good real life like generalized dataset. We also consider different types of vehicles (car,truck,bikes etc.) to generalize the speed, acceleration,turning profiles of every individual vehicle in our data capture. Trajectories involving collisions with other vehicles, stop signs,pedestrians etc. were labelled as bad examples and removed from the final dataset version. We ran into troubles getting our simulated dataset to be uniform, and free from stopping as the car would inherently stop at red lights. We also ran into data capture issues where the simulator position was out of sync and did not get captured for durations of time when the GPU load was too high.

5.2 KITTI Dataset

For our second dataset, we make use of the KITTI ODOMETRY dataset (15) publicly available under the KITTI Vision Benchmark Suite. We samples the images from its different provided trajectories and simultaneously store the next T state position as expert path locations. We perform standard image pre processing procedure to make the data suitable for the pre-trained Resnet 50 architecture.

6 Experiments and Results

6.1 CARLA Dataset

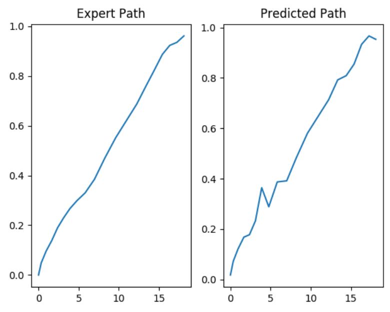
For the Carla dataset we have access to the current x and y location of the agent in world co-ordinates and also the current angle of the car with respect to the z-axis. We use these values to transform the future path such that the current location is at the origin and the current orientation of the car is normalized to have 0° rotation. For the purposes of this experiment we train for 10 epochs with a learning rate of 10^{-3} with a decay rate of 10^{-1} every 2 epochs. We also found that gradient clipping was an effective way to regulate training as past research has shown that gradients can explode when training recurrent networks on long sequences. We used a clip value of 0.75. We found that our algorithm found it difficult to predict the correct path in a majority of the situations and need to further investigate why our algorithm converges to a bad local minimum.

6.2 KITTI Odometry

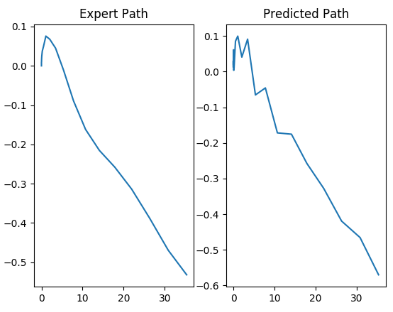
For the KITTI odometry dataset we computed our expert trajectories by using the given camera matrix for each frame. We also computed the next position in an interval of every 4 frames so that our points were sufficiently spaced apart. To compute the translation between successive frames given the camera matrices M_1 and M_2 we used the following method -

$$[R \quad t] = M_1^{-1} * M_2$$

We then extract t from the last column of the resulting matrix. We follow this procedure keeping M_1 the same and grabbing future positions of M_2 to get a future path. We also use an pretrained Resnet-50 architecture and use the second to last Fully Connected layer as our initial image representation. We do this to take advantage of the large scale datasets created for supervised image classification.For the purposes of this experiment we train for 10 epochs with a learning rate of 10^{-3} with a decay rate of 10^{-1} every 2 epochs. We again found that our algorithm converged to a bad local minimum often predicting paths that our not plausible.



(a) Trajectory 1



(b) Trajectory 2

Figure 6: Expert and Predicted Paths for KITTI dataset

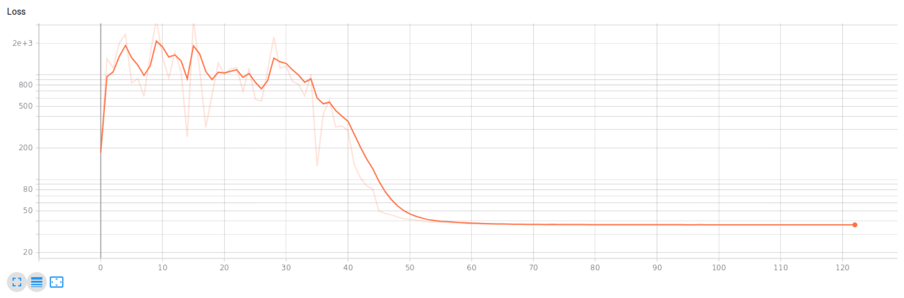


Figure 7: Loss convergence while training on KITTI dataset

7 Discussion

We often found that our paths did not cover the distance of longer trajectories and was also sort of erratic in predicting the points. Our points were generally only covering around 1m in distance. We hypothesize that our model treated these points as outliers as there is a lot of stopping in the KITTI and CARLA datasets leading to compressed paths. We also noticed that the KITTI dataset is sub-optimal for our particular task, we ran experiments on this dataset none the less as it was the best real world dataset we could find. We also found that gradient clipping was an effective way to regulate training as past research has shown that gradients can explode when training recurrent networks on long sequences. We were also restricted by GPU resources as collecting data on AWS servers for CARLA can be time consuming and often difficult to ensure good data. We would like to continue exploring this research area.

8 Conclusion

We found that we needed to spend more time collecting clean data and ensuring correct input to our model. We wanted to implement the model none the less because we wanted to see if it was possible to get plausible paths. We believe that methods that generate a sequence of future positions is more valuable than outputting a single steering angle and velocity as this allows downstream tasks to reason about the safety of the forecasted path and make adjustments when necessary. The stochastic nature of the generations allows the system to change the path when the forecast is deemed to be unsafe. In the next section we discuss areas where we see the most room for improvement along with ways we can select a particular desired path.

9 Future Work

In the path planning implementation we only use the forward cross entropy loss. Some of future steps to get our algorithm to predict more plausible paths would include adding the backward cross entropy term to our loss function as well as formulating $\hat{p}$ in some sort of learn able our reliable way. Our next step is to figure out what is can be done to get the implementation of our algorithm to converge to a more optimal point where it is able to generalize to a larger set of images. We would also like to perform studies on random road images that we can find to see how our model generalizes to different inputs or if there are any inherant biases present in our model. We would also like to study how we can choose a particular trajectory out of the generated distribution given a particular goal state. This involves computing the conditional probability of landing on the given goal state given our distribution and altering the noise input such that this conditional is maximized.

References

[1] N. Rhinehart, K. M. Kitani, and P. Vernaza, "R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 772–788, 2018.

[2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[3] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[4] M. Hoai and F. De la Torre, "Max-margin early event detectors," *International Journal of Computer Vision*, vol. 107, no. 2, pp. 191–202, 2014.

[5] M. Ryoo, T. J. Fuchs, L. Xia, J. K. Aggarwal, and L. Matthies, "Robot-centric activity prediction from first-person videos: What will they do to me?," in *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 295–302, IEEE, 2015.

[6] M. S. Ryoo, "Human activity prediction: Early recognition of ongoing activities from streaming videos," in *2011 International Conference on Computer Vision*, pp. 1036–1043, IEEE, 2011.

[7] N. Rhinehart and K. M. Kitani, "First-person activity forecasting with online inverse reinforcement learning," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3696–3705, 2017.

[8] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *European Conference on Computer Vision*, pp. 201–214, Springer, 2012.

[9] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*, pp. 729–736, ACM, 2006.

[10] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Proceedings of the twenty-third aaai conference on artificial intelligence (2008) maximum entropy inverse reinforcement learning,"

[11] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, p. 1, ACM, 2004.

[12] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[13] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems*, pp. 3812–3822, 2017.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

[15] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.