LEARNING POLICIES FROM OBSERVATIONAL DATA BY PENALIZING COVARIATE SHIFT

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning a policy using only observational data is challenging because the distribution of states induced by a learned policy may differ from the distribution observed during training. In this work, we propose to address this covariate shift by explicitly minimizing a measure of the mismatch between these two distributions. We obtain estimates of the distribution induced by the policy through a learned dynamics model, and propose a general, differentiable measure of the distribution mismatch using uncertainty estimates of the dynamics model's predictions, calculated using dropout. Our framework also naturally yields a model-based imitation learning algorithm when the observational data consists of expert trajectories. We evaluate both approaches using a large-scale observational dataset of driving behavior recorded from traffic cameras, and show that we are able to learn effective driving policies from purely observational data, with no environment interaction or additional labeling by an expert.

1 INTRODUCTION

In recent years, model-free reinforcement learning methods using deep neural network controllers have proven effective on a wide range of tasks, from playing video or text-based games (Mnih et al., 2015; 2016; Narasimhan et al., 2015) to learning algorithms (Zaremba et al., 2015) and complex locomotion tasks (Lillicrap et al., 2015; Zhang et al., 2015). However, these methods often require a large number of interactions with the environment in order to learn. While this is not a problem if the environment is simulated, it can limit the application of these methods in realistic environments where interactions with the environment are slow, expensive or potentially dangerous. Building a simulator where the agent can safely try out policies without facing real consequences can mitigate this problem, but requires human engineering effort which increases with the complexity of the environment being modeled.

Model-based reinforcement learning approaches try to learn a model of the environment dynamics, and then use this model to plan actions or train a parameterized policy. A common setting is where an agent alternates between collecting experience by executing actions using its current policy or dynamics model, and using these experiences to improve its dynamics model. This approach has been shown empirically to significantly reduce the required number of environment interactions (Atkeson & Santamaria, 1997; Deisenroth & Rasmussen, 2011; Nagabandi et al., 2017; Chua et al., 2018). Theoretical results have also shown the model-based approach to offer improved sample complexity in simple settings (Tu & Recht, 2017; Recht, 2018).

Despite these improvements, there exist settings where even a *single* poor action executed by an agent in a real environment can have consequences which are not acceptable. At the same time, with data collection becoming increasingly inexpensive, there are many settings where observational data of an environment is abundant. This suggests a need for algorithms which can learn policies primarily from observational data, which can then perform well in a real environment. Autonomous driving is an example of such a setting: on one hand, trajectories of human drivers can be easily collected using traffic cameras (Halkias & Colyar, 2006), resulting in an abundance of observational data; on the other hand, learning through interaction with the real environment is not a viable solution.

However, learning policies from purely observational data is challenging because the data may only cover a small region of the space over which it is defined. If the observational data contains state-action pairs produced by experts, one option is to use imitation learning (Pomerleau, 1991). How-

ever, passive imitation learning is well-known to suffer from a mismatch between input states seen at training and execution time, and may need to be augmented by interacting with an expert (Ross et al., 2011).

Another option is to learn a dynamics model from observational data, and then use it to train a policy. However, the dynamics model may make arbitrary predictions outside the domain it was trained on, which may wrongly be associated with low cost (or high reward) as shown in Figure 1. The policy network may then exploit these errors in the dynamics model and produce actions which lead to these wrongly optimistic states. In the interactive setting, this problem is naturally self-correcting, since states where the model predictions are wrongly optimistic will be more likely to be experienced, and thus will correct the dynamics model. However, this does not solve the problem if the dataset of environment interactions which the model is trained on is fixed.

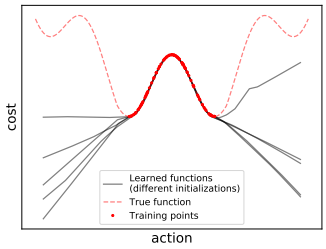


Figure 1: Different models fitted on training points which cover a limited region the function's domain. Models make arbitrary predictions outside of this region.

TODO: rewrite this part In this work, we propose to explicitly penalize the mismatch between the distribution over states induced by the policy network and the one reflected in the training data. The key element of our approach is a differentiable term in the cost function used at planning time, representing the uncertainty of the forward model about its own predictions, which is calculated using dropout. Minimizing this term encourages a planner or policy network to choose action sequences which keep it on the data manifold which the forward model was trained on. We first introduce a large-scale dataset of real-world driving trajectories, which we also adapt into an environment for testing learned policies and planning methods; both dataset and environment will be made public. This dataset is highly stochastic due to the unpredictable nature of human driving, and we train both deterministic and stochastic action-conditional forward models which can then be used for planning. We show in our experiments that model-based control using this additional regularizer substantially outperforms unregularized control, as well as both standard and model-based imitation learners, and enables learning good driving policies using only observational data with no environment interaction.

2 Approach

We assume we are given a dataset of observational data which consists of state-action pairs $\mathcal{D} = \{(s_t, a_t)\}_t$. Our approach consists of two steps: learning an action-conditional dynamics model using the collected observational data, and then using this model to train a fast, feedforward policy network which minimizes both a policy cost and an uncertainty cost.

2.1 Action-conditional Forward Model

We consider both deterministic and stochastic dynamics models. Our deterministic model $f_\theta(s_{1:t}, a_t)$ takes as input a sequence of observed states $s_{1:t}$ and an action a_t, and produces a prediction of the next state $\hat{s}_{t+1}$. The per-sample loss which it minimizes is given by $\ell(s_{t+1}, \hat{s}_{t+1})$, where ℓ is a loss function appropriate for the task at hand. The stochastic model additionally takes as input a latent variable z_t which represents the information about the next state s_{t+1} which is not a deterministic function of the input. In this work, we consider recent approaches for stochastic prediction based on Variational Autoencoders (Kingma & Welling, 2013; Babaeizadeh et al., 2017; Denton & Fergus, 2018), but other approaches could be used as well. During training, latent variables are sampled from a posterior network $q_\phi(s_{1:t}, s_{t+1})$ which outputs the parameters (μ_ϕ, σ_ϕ) of a diagonal Gaussian distribution over latent variables conditioned on the past inputs and true targets. This network is trained jointly with the prediction model using the reparameterization trick, and a term is included in the loss to minimize the KL divergence between the posterior distribution and a fixed prior $p(z)$, which in our case is an isotropic Gaussian. Details of the stochastic model updates are given in Appendix B. After training, different future predictions for a given sequence of frames can be generated by sampling different latent variables from the prior distribution.

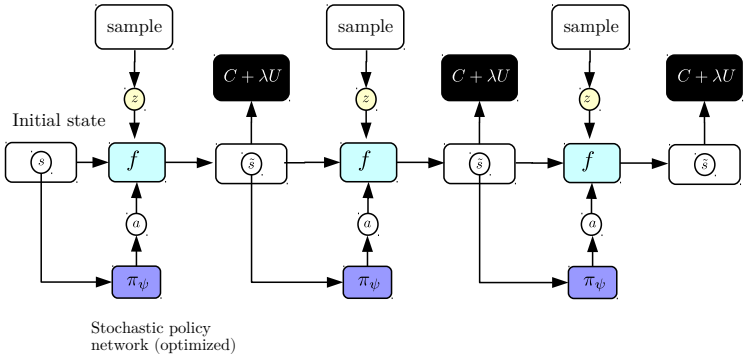


Figure 2: Training the policy network using the stochastic forward model. Gradients with respect to costs associated with predicted states are passed through the unrolled forward model into a policy network.

2.2 Training a Policy Network

Once the forward model is trained, we can use it to train a parameterized policy network π_ψ, which we assume to be stochastic. In the case of the deterministic forward model, we first sample an initial state sequence $s_{1:t}$ from the training set, unroll the forward model over T time steps, and backpropagate gradients of a differentiable objective function with respect to the parameters of the policy network. During this process the weights of the forward model are fixed, and only the weights of the policy network are optimized. This objective function contains two terms: a *policy cost* C, which reflects the underlying objective the policy is trying to learn, and an *uncertainty cost* U, which reflects how close the predicted state induced by the policy network is to the manifold which the data $\mathcal{D}$ is drawn from:

$$\underset{\psi}{\text{argmin}} \Big[\sum_{i=1}^{T} C(\hat{s}_{t+i}) + \lambda U(\hat{s}_{t+1}) \Big], \text{ such that: } \begin{cases} \hat{a}_{t+i} \sim \pi_\psi(s_{t+i-1}) \\ \hat{s}_{t+i} = f(\hat{s}_{t+i-1}, \hat{a}_{t+i}) \end{cases}$$

In the case of the stochastic forward model, we additionally sample latent variables from the prior for each step and input them into the forward model. The problem them becomes:

$$\underset{\psi}{\text{argmin}} \Big[\sum_{i=1}^{T} C(\hat{s}_{t+i}) + \lambda U(\hat{s}_{t+1}) \Big], \text{ such that: } \begin{cases} z_{t+i} \sim p(z) \\ \hat{a}_{t+i} \sim \pi_\psi(s_{t+i-1}) \\ \hat{s}_{t+i} = f(\hat{s}_{t+i-1}, \hat{a}_{t+i}, z_{t+i}) \end{cases}$$

The uncertainty cost U is applied to states predicted by the forward model which reflect the distribution induced by the policy network. This cost could reflect any prior knowledge the user has about the distribution of states. We propose here a general form based on the uncertainty of the dynamics model, calculated using dropout. Intuitively, if the dynamics model is given a state-action pair from the same distribution as $\mathcal{D}$, it will have low uncertainty about its prediction. If it is input a state-action pair which is outside this distribution, it will have high uncertainty.

Dropout (Hinton et al., 2012; Srivastava et al., 2014) is a regularization technique which consists of randomly setting hidden units in a neural network to zero with some probability. The work of (Gal & Ghahramani, 2016) showed that a neural network trained with dropout is mathematically equivalent to an approximation of a probabilistic deep Gaussian Process model, which includes uncertainty estimates. A key result of this is that estimates of the neural network's model uncertainty for a given input can be obtained by calculating the covariance of an outputs taken over multiple dropout masks. More precisely, let $f_{\theta_1}, ..., f_{\theta_K}$ denote our prediction model with K different dropout masks applied to it. Then for some set of inputs $(s_{1:t}, a_t, z_t)$, the uncertainty of the forward model is given by the covariance matrix of the outputs under the different dropout masks (up to some constants).

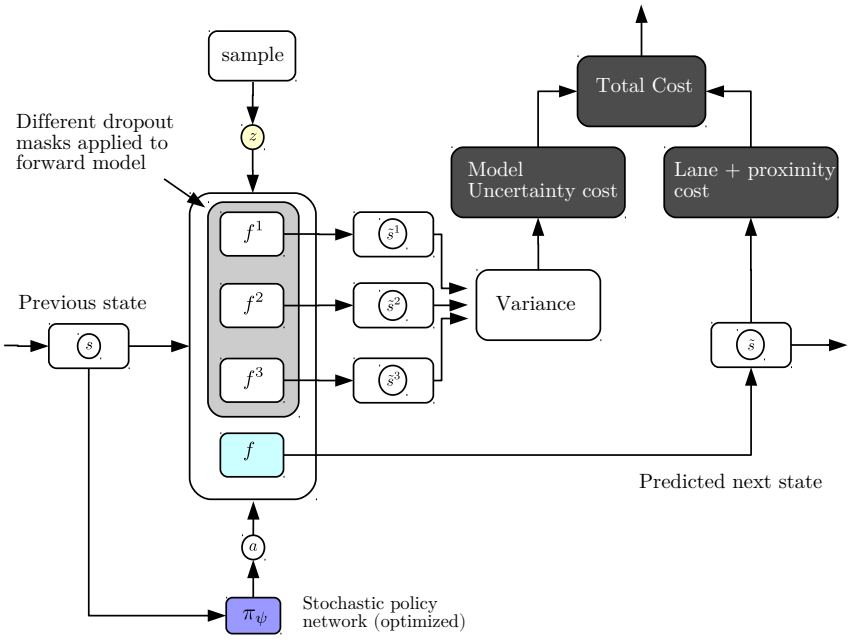


Figure 3: Training the policy network using the differentiable uncertainty cost, calculated using dropout.

We note that this uncertainty estimate is the composition of differentiable functions: each of the models induced by applying a different dropout mask is differentiable, as is the covariance operator. Furthermore, we can summarize the covariance matrix by taking its trace (which is equal to the sum of its eigenvalues, or equivalently the sum of the variances of the outputs across each dimension), which is also a differentiable operation. This provides a scalar estimate of uncertainty Ω which is differentiable with respect to the inputs:

$$\begin{aligned}\Omega(s_{1:t}, a_t, z_t) &= \text{tr}\Big[\text{Cov}[\{f_{\theta_k}(s_{1:t}, a_t, z_t)\}_{k=1}^K]\Big] \\ &= \sum_{j=1}^{d} \text{Var}(\{f_{\theta_k}(s_{1:t}, a_t, z_t)_j\}_{k=1}^K)\end{aligned}$$

where d is the dimensionality of the output. Minimizing this quantity with respect to actions encourages the policy network to produce actions which, when plugged into the forward model, will produce predictions which the forward model is confident about. To compensate for differences in baseline uncertainty across different modalities or rollout lengths, for each modality (i.e. images i_t and vectors u_t) we estimate the empirical mean and variance of Ω for every rollout length t of the forward model over the training set, to obtain μ_Ω^t and σ_Ω^t. We then define our uncertainty cost as follows:

$$U(\hat{s}_{t+1}) = U(s_{1:t}, a_t, z_t) = \Big[\frac{\Omega(s_{1:t}, a_t, z_t) - \mu_\Omega^t}{\sigma_\Omega^t}\Big]_+ \tag{1}$$

If the uncertainty estimate is lower than the mean uncertainty estimate on the training set for this rollout length, this loss will be zero. These are cases where the model prediction is within normal uncertainty ranges. If the uncertainty estimate is higher, this loss exerts a pull to change the action so that the future state will be predicted with higher confidence by the forward model.

A simple way to define U given an initial sequence of states $s_{1:t}$ from $\mathcal{D}$ would be to set $U(\hat{s}_{t+k}) = \|\hat{s}_{t+k} - s_{t+k}\|$. This would encourage the policy network to output actions which lead to a similar

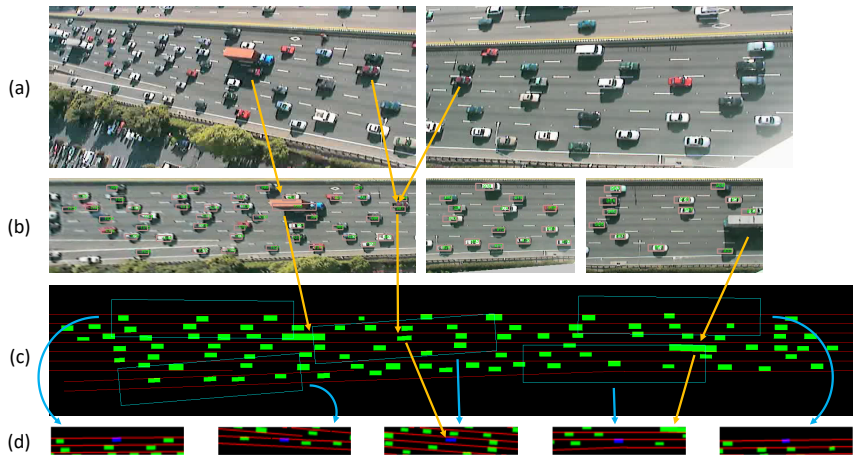


Figure 4: Preprocessing pipeline for the NGSIM-I80 data set. Orange arrows show same vehicles across stages. Blue arrows show corresponding extracted context state. (a) Snapshots from two of the seven cameras. (b) View point transformation, car localisation and tracking. (c) Context states are extracted from rectangular regions surrounding each vehicle. (d) Five examples of context states extracted at the previous stage.

trajectory as the one observed in the dataset. Although this leads to a set of states the model is presumably confident about, it may not be a trajectory which also satisfies the policy cost C. The advantage of using the more general cost above is that there are presumably multiple trajectories which the forward model is confident about, and the policy network can then choose one which also satisfies the cost it is trying to optimize. However, if the dataset $\mathcal{D}$ consists of *expert* trajectories, then we can set $C(\hat{s}_{t+i}) = U(\hat{s}_{t+i}) = \frac{1}{2}\|\hat{s}_{t+k} - s_{t+k}\|$.

3 Dataset and Planning Environment

We apply our approach to learn driving policies using a large-scale dataset of driving videos taken from traffic cameras. The Next Generation Simulation program's Interstate 80 (NGSIM I-80) dataset (Halkias & Colyar, 2006) consists of 45 minutes of recordings from traffic cameras made of a stretch of highway. The driver behavior is complex and includes sudden accelerations, lane changes and merges which are difficult to predict; as such the dataset has high enviroment stochasticity. After recording, a viewpoint transformation is applied to rectify the perspective, and vehicles are identified and tracked throughout the video. This yields a total 5596 car trajectories, which are then split into training (80%), validation (10%) and testing sets (10%).

We then applied additional preprocessing to obtain a state and action representations (s_t, a_t) for each car, suitable for learning an action-conditional predictive model. Our state representation s_t consists of two components: an image i_t representing the neighborhood of the car, and a vector u_t representing its current position and velocity. The images i_t are centered around the ego car and encode both the lane emplacements and the locations of other cars. Each image has 3 channels: the first (red) encodes the lane markings, the second (green) encodes the locations of neighboring cars, which are represented as rectangles reflecting the dimensions of each car, and the third channel (blue) represents the ego car, also scaled to the correct dimensions. This is summarized in Figure 4. We also define two cost functions to be minimized at planning time: a proximity cost which reflects how close the ego car is to neighboring cars, and a lane cost which reflects how much the ego car overlaps with lane markings. These are represented as a cost vector at each timestep, $c_t = (C_{proximity}(s_t), C_{lane}(s_t))$.

We also adapted this dataset to be used as an environment to evaluate learned policies, with the same interface as OpenAI Gym (Brockman et al., 2016). Choosing a policy for neighboring cars is challenging due to a cold-start problem: to accurately evaluate a learned policy, the other cars would need to follow human-like policies which would realistically react to the controlled car, which are not available. We take the approach of letting all the other cars in the environment follow their

trajectories from the dataset, while a single car is controlled by the policy we seek to evaluate. This approach avoids hand-designing a policy for the neighboring cars which would likely not reflect the diverse nature of human driving. The limitation is that the neighboring cars do not react to the controlled car, which likely makes the problem more difficult.

4 Related Work

A number of authors have explored the use of learned, action-conditional forward models which are then used for planning, starting with classic works in the 90's (Nguyen & Widrow, 1990; Schmidhuber, 1990; Jordan & Rumelhart, 1992), and more recently in the context of video games (Oh et al., 2015; Pascanu et al., 2017; Weber et al., 2017), robotics and continous control (Finn et al., 2016; Agrawal et al., 2016; Nagabandi et al., 2017; Srinivas et al., 2018). Our approach to learning policies by backpropagating through a learned forward model is related to the classic work of (Nguyen & Widrow, 1989) in the deterministic case, and the SVG framework of (Heess et al., 2015) in the stochastic case. However, neither of these approaches incorporates a term penalizing the uncertainty of the forward model when training the policy network.

The works of (McAllister & Rasmussen, 2016; Chua et al., 2018) also used model uncertainty estimates calculated using dropout in the context of model-based reinforcement learning. They did so during the forward prediction step. Namely, they used different dropout masks to simulate different state trajectories which were then averaged to produce a cost estimate used to select an action.

Our model uncertainty penalty is related to the cost used in (Kahn et al., 2017), who used dropout and model ensembling to compute uncertainty estimates for a binary action-conditional collision detector for a flying drone. These estimates were then used to select actions out of a predefined set which yielded a good tradeoff between speed, predicted chance of collision and uncertainty about the prediction. In our work, we apply uncertainty estimates to the predicted states of a forward model at every time step, and backpropagate gradients through the unrolled forward model to then train a policy network by gradient descent.

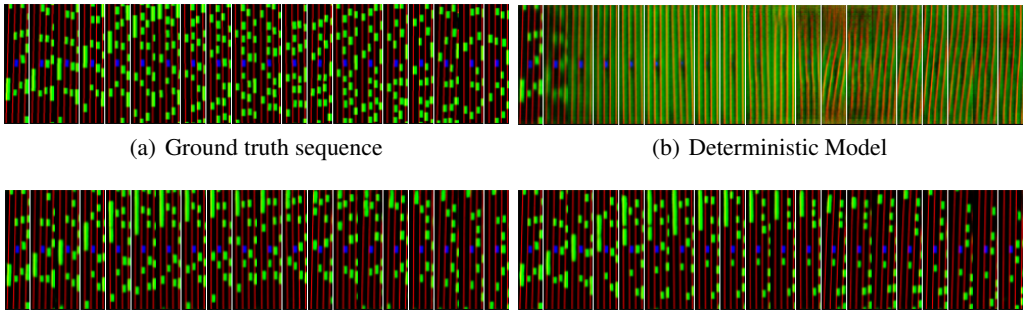
The problem of covariate shift when executing a policy learned from observational data has been well-recognized in imitation learning. It was first noted in the early work of (Pomerleau, 1991), and was shown in (Ross & Bagnell, 2010) to cause a regret bound which grows quadratically in the time horizon of the task. The work of (Ross et al., 2011) proposed DAGGER to efficiently use expert feedback if available, which has been applied in the context of autonomous driving (Zhang & Cho, 2016). Our approach also addresses covariate shift, but does so without querying an expert.

Our model-based imitation learning algorithm is related to the work of (Englert et al., 2013), who also performed imitation learning at the level of trajectories rather than individual actions. They did so in low-dimensional settings using Gaussian Processes, whereas our method uses an unrolled neural network representing the environment dynamics which can be applies to high-dimensional state representations.

Several works have applied deep learning methods in the context of autonomous driving. The works of (Pomerleau, 1991; Bojarski et al., 2016; Pan et al., 2017) used neural networks to control policies trained by imitation learning, while (Williams et al., 2017) learned models of the vehicle dynamics. These works focused on lane following in visually rich environments and did not focus on settings with dense traffic, which we focus on in this work. The work of (Sadigh et al., 2016) developed a model of the interactions between the two drivers which was then used to plan actions in simple settings, using symbolic state representations. In our work, we consider the problem of learning driving policies in dense traffic, using high-dimensional state representations which reflect the neighborhood of the ego car.

5 Experiments

We now report experimental results. All training details can be found in Appendix D.



(a) Ground truth sequence (b) Deterministic Model

(c) Stochastic Model, sample 1 (d) Stochastic Model, sample 2

Figure 5: Video prediction results using a deterministic and stochastic model. Two different future predictions are generated by the stochastic model by sampling two different sequences of latent variables. The deterministic model averages over possible futures, producing blurred predictions.

5.1 Prediction Results

Figure 5.1 shows predictions made by our stochastic model, as well as the deterministic model which does not use latent variables. The deterministic model produces predictions which become increasingly blurry further into the future, illustrating the phenomenon of averaging over different futures described in the introduction. Our stochastic model produces predictions which stay sharp far into the future. By sampling different sequences of latent variables, different future scenarios are generated. Note that the two sequences generated by the stochastic model are different from the ground truth future which occurs in the dataset. This is normal as the future observed in the dataset is only one of many possible ones. Additional video generations can be viewed at the following URL: `https://youtu.be/wRrQEvLq3dA`.

5.2 Planning Results

We evaluate different policies using two measures: whether the controlled car reaches the end of the road segment without colliding into another car or driving off the road, and the distance travelled before the episode ends (policies which collide quickly will travel shorter distances). All cars are initialized at the beginning of the road segment with the initial speed they were driving at in the dataset, and then are controlled by the policy being measured.

We compared our approach against several baselines which also learn from observational data, which we briefly describe below. Full details of each of them are contained in Appendix C. All policy networks have the same architecture, and all are fed the concatenation of the 20 previous states as input. They all output the parameters of a 2D diagonal Gaussian over action space, from which the next action is sampled.

Human: This is the performance of the actual human trajectories observed in the testing set, which are all collision-free.

No action: A simple policy which outputs an action of zero, i.e. maintains its current speed and direction.

1-step Imitation Learner: A policy network trained to minimize the negative log-likelihood of the next human action observed in the dataset under the parameters of its output distribution.

SVG(∞): A policy network trained with stochastic value gradients, using our stochastic forward model. Here the latent variables are inferred using the posterior network. This is the same setup as (Heess et al., 2015), with the difference that the agent does not interact with the environment and learns from a fixed observational dataset.

SVG-P(∞): Same as SVG, but with latent variables sampled from the prior rather than inferred with the posterior network.

Method	Mean Distance	Success Rate (%)
Human	1358	100.0
No action	566	16.2
Rule-Based Policy	-	-
1-step IL	322	1.4
MBIL (1 step)	327	6.4
MBIL (3 step)	473	16.2
MBIL (5 step)	830	37.5
MBIL (10 step)	943	67.1
MBIL (20 step)	1024	66.6
MBIL (30 step)	1028	67.7
Stochastic MBIL (10 step)	284	0.7
Stochastic MBIL (20 step)	333	1.2
Stochastic MBIL (30 step)	349	0.7
Stochastic MBIL (40 step)	386	0.8
VG(∞)	97	0.0
SVG(∞)	88	0.0
VG(∞) + Epistemic Cost	963	56.4
SVG(∞) + Epistemic Cost	**976**	**59.1**

Table 1: Planning performance on NGSIM dataset.

VG(∞): This is the same setup as SVG(∞), but using the deterministic forward model. This is the same setup as (**?**).

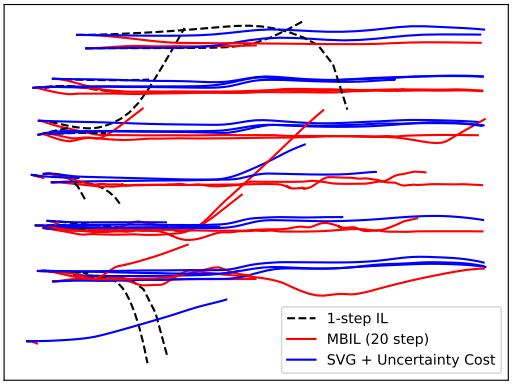
Table 1 compares performance for the different methods. The cost function which we minimize for SVG is given by:

$$C = C_{proximity} + 0.2 \cdot C_{lane} + \lambda_U \cdot C_U \tag{2}$$

where $C_{proximity}$ and C_{lane} are the proximity and lane costs described in Section 3, and C_U is the model uncertainty cost described in Section **??**. We compare results with $\lambda_U = 0$ and $\lambda_U > 0$ to show its effect.

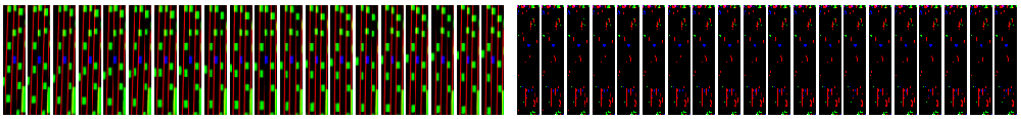
The 1-step imitation learner performs poorly. Training MBIL policies with longer rollouts improves performance up to a point, but still does not beat the simple baseline of performing no action. Both SVG and VG without the uncertainty cost also give poor performance. However, adding the uncertainty cost improves performance dramatically, outperforming all other methods by a wide margin. Despite the difference in prediction quality observed in Figure 5.1, using the stochastic rather than deterministic forward model offers a relatively modest improvement in terms of the performance of the policy network it is used to train. Videos of the learned policies driving in the environment can be found at `https://youtu.be/ApjHjhyAMw0`. The policy learns effective behaviors such as braking, accelerating and turning to avoid other cars.

To illustrate the effect of the epistemic uncertainty cost, we plot predictions made by the stochastic forward model for an action sequence produced by SVG policy networks trained with and without the epistemic uncertainty cost, shown in Figure 7. The policy network trained with the uncertainty cost produces actions which, when plugged into the forward model, yield predictions which remain on the data manifold. The policy network trained without the uncertainty cost, however, produces actions which produce predictions not resembling any of the training examples. Notice however that these predictions yield low proximity cost, since the green channel representing cars is almost zero. Also included in Figure 7 are the average predicted proximity cost by the forward model for both policies, as well as average epistemic uncertainty cost C_U. The model trained without the uncertainty penalty produces actions for which the forward model predicts very low cost, but its uncertainty about its predictions is very high. The model trained with the penalty produces actions for which the forward model predicts higher cost, but with much less uncertainty. Including the uncertainty cost forces the policy network to produce actions which still produce reasonable predictions under the forward model, and give much better results when executed in the environment.



(a) Ground truth sequence

Figure 6: Video prediction results using a deterministic and stochastic model. Two different future predictions are generated by the stochastic model by sampling two different sequences of latent variables. The deterministic model averages over possible futures, producing blurred predictions.



(a) with uncertainty cost (b) without uncertainty cost

Method	Mean Distance	Success Rate	Total Predicted Cost	C_U
SVG(∞)	88.7	0.0	0.03	4087.4
SVG(∞) + Epistemic Cost	976.4	59.1	0.22	1.1

Figure 7: Predictions made by the same forward model for an action sequence produced by a policy trained *with* (top) and *without* (bottom) the epistemic uncertainty cost. Without the uncertainty cost, the policy network learns to output pathological action sequences which produce invalid predictions which nevertheless have low cost.

Method	$\|\partial i_{t+1}/\partial a_t\|_2$	$\|\partial u_{t+1}/\partial a_t\|_2$
SVG(∞)	-	-
SVG(∞)	-	-

There is a significant difference in performance between SVG(∞) and SVG-sim(∞). The main difference between these two methods is that SVG(∞) uses latent variables which are inferred from true trajectories in the training set using the posterior network, whereas SVG-sim(∞) uses latent variables which are sampled from the prior distribution $p(z)$. To better understand the effect of these two approaches, we compared the predictions made by the forward model using inferred and sampled latent variables for different sequences of actions sampled from the training set. Examples are shown in Figure **??**. The predicted sequences using the inferred latent variables resemble the ground truth sequence, even when changing the sequence of actions input to the forward model. In contrast, when using latent variables sampled from the prior, changing the action sequence results in different sequences being predicted by the forward model. This suggests that using inferred latent variables reduces the sensitivity of the forward model to the actions. To quantify this, we measured the magnitude of the partial derivatives of the output images with respect to actions, averaged over the training set, and found that these derivative were much smaller when the latent variables were inferred rather than sampled.

One explanation is that the forward model encodes some factors of variation of the output due to the actions in its latent variables. The sequence latent variables sampled from the prior are independent, which may cause these effects to cancel each other over the sequence. However, the sequence of

inferred latent variables are highly dependent, and together they may explain away the effects of the actions.

6 CONCLUSION

TODO: improve

In this work we have presented an end-to-end approach for learning driving policies from observational data, which includes preparing a dataset of real-world driving trajectories, adapting it to become a planning environment, training action-conditional forward models, and using them to train policies using with a new uncertainty regularizer which addresses the domain mismatch problem. There are several directions for future work. Although we have applied it here in the context of learning driving policies, the approach is general and could be used in other domains. Furthermore, our current approach does not capture dependencies which are longer than 20 time steps into the future, which corresponds to 2 seconds. We tried two different approaches to capture longer-term dependencies: learning a value function using temporal differences (which in theory can capture arbitrary length dependencies) and unrolling for more time steps, but these did not yield any improvements and sometimes hurt performance. This requires more investigation. Second, it would be interesting to optimize actions or policies to produce more complex and useful behaviors, such as changing to a specified lane while avoiding other cars. In the current setup, the policies are optimized only to avoid collisions and stay within lanes when possible, whereas in a real-world scenario we would want a policy which can safely navigate among traffic to different locations.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL http://arxiv.org/abs/1606.07419.

C. G. Atkeson and J. C. Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pp. 3557–3564 vol.4, April 1997. doi: 10.1109/ROBOT.1997.606886.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *CoRR*, abs/1710.11252, 2017. URL http://arxiv.org/abs/1710.11252.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018. URL http://arxiv.org/abs/1805.12114.

Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *In Proceedings of the International Conference on Machine Learning*, 2011.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *CoRR*, abs/1802.07687, 2018. URL http://arxiv.org/abs/1802.07687.

P. Englert, A. Paraschos, J. Peters, and M. P. Deisenroth. Model-based imitation learning by probabilistic trajectory matching. In *2013 IEEE International Conference on Robotics and Automation*, pp. 1922–1927, May 2013. doi: 10.1109/ICRA.2013.6630832.

Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *CoRR*, abs/1605.07157, 2016. URL http://arxiv.org/abs/1605.07157.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

John Halkias and James Colyar. NGSIM interstate 80 freeway dataset. *US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA*, 2006.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2944–2952. Curran Associates, Inc., 2015.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307 – 354, 1992. ISSN 0364-0213. doi: http://dx.doi.org/10.1016/0364-0213(92)90036-T. URL http://www.sciencedirect.com/science/article/pii/036402139290036T.

Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. 02 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013. URL http://dblp.uni-trier.de/db/journals/corr/corr1312.html#KingmaW13.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

Rowan McAllister and Carl E. Rasmussen. Improving pilco with bayesian neural network dynamics models. 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 0028-0836. doi: 10.1038/nature14236. URL http://dx.doi.org/10.1038/nature14236.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017. URL http://arxiv.org/abs/1708.02596.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *CoRR*, abs/1506.08941, 2015. URL http://arxiv.org/abs/1506.08941.

Nguyen and Widrow. The truck backer-upper: an example of self-learning in neural networks. In *International 1989 Joint Conference on Neural Networks*, pp. 357–363 vol.2, 1989. doi: 10.1109/IJCNN.1989.118723.

Derrick Nguyen and Bernard Widrow. Neural networks for control. chapter The Truck Backer-upper: An Example of Self-learning in Neural Networks, pp. 287–299. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-262-13261-3. URL http://dl.acm.org/citation.cfm?id=104204.104216.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. Action-conditional video prediction using deep networks in atari games. *CoRR*, abs/1507.08750, 2015. URL http://arxiv.org/abs/1507.08750.

Toshiyuki Ohtsuka. A continuation/gmres method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563–574, 2004. URL http://dblp.uni-trier.de/db/journals/automatica/automatica40.html#Ohtsuka04.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *CoRR*, abs/1709.07174, 2017. URL http://arxiv.org/abs/1709.07174.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sébastien Racanière, David P. Reichert, Theophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *CoRR*, abs/1707.06170, 2017.

Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:97, 1991.

Benjamin Recht. A tour of reinforcement learning: the view from continuous control. *CoRR*, abs/1806.09460, 2018. URL https://arxiv.org/abs/1806.09460.

Stéphane Ross and J. Andrew Bagnell. Efficient reductions for imitation learning. In *AISTATS*, 2010.

Stphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudk (eds.), *AISTATS*, volume 15 of *JMLR Proceedings*, pp. 627–635. JMLR.org, 2011.

Dorsa Sadigh, Shankar Sastry, Sanjit A. Seshia, and Anca D. Dragan. Planning for autonomous cars that leverage effects on human actions, 06 2016.

Jurgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 1990.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *CoRR*, abs/1804.00645, 2018. URL http://arxiv.org/abs/1804.00645.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Stephen Tu and Benjamin Recht. Least-squares temporal difference learning for the linear quadratic regulator. *CoRR*, abs/1712.08642, 2017. URL http://arxiv.org/abs/1712.08642.

Theophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.

G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, May 2017. doi: 10.1109/ICRA.2017.7989202.

Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. *CoRR*, abs/1511.07275, 2015. URL http://arxiv.org/abs/1511.07275.

Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter I. Corke. Towards vision-based deep reinforcement learning for robotic motion control. *CoRR*, abs/1511.03791, 2015. URL http://arxiv.org/abs/1511.03791.

Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *CoRR*, abs/1605.06450, 2016. URL http://arxiv.org/abs/1605.06450.

A Dataset and Planning Environment

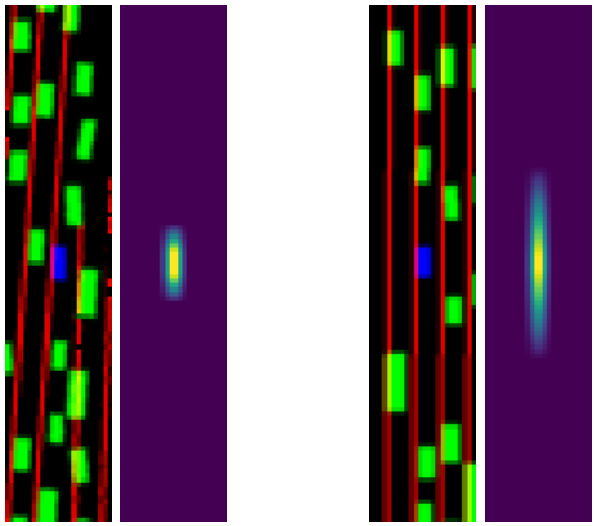
To begin with, we describe the details and preparation of the dataset and planning environment which we used, which are summarized in Figure 4. The Next Generation Simulation program's Interstate 80 (NGSIM I-80) dataset (Halkias & Colyar, 2006) consists of 45 minutes of recordings made of a stretch of highway in the San Francisco Bay Area by cameras mounted on a 30-story building overlooking the highway. The recorded area includes six freeway lanes (including a high-occupancy vehicle lane) and an onramp. The driver behavior is complex and includes sudden accelerations, lane changes and merges which are difficult to predict; as such the dataset has high aleatoric uncertainty. There are three time segments, each of 15 minutes, taken at different times of day which capture the transition between uncongested and congested peak period conditions. After recording, a viewpoint transformation is applied to rectify the perspective, and vehicles are identified and tracked throughout the video; additionally, their size is inferred. This yields a total 5596 car trajectories, represented as sequences of coordinates $\{x_t, y_t\}$. We split these trajectories into training (80%), validation (10%) and testing sets (10%).

We then applied additional preprocessing to obtain suitable representations for learning a predictive model. Specifically, we extracted the following: i) a state representation for each car at each time step s_t, which encodes the necessary information to choose an action to take, ii) an action a_t which represents the action of the driver, and iii) a cost c_t, which associates a quality measure to each state. We describe each of these below.

State representation: Our state representation consists of two components: an image representing the neighborhood of the car, and a vector representing its current position and velocity. For the images, we rendered images centered around each car which encoded both the lane emplacements and the locations of other cars. Each image has 3 channels: the first (red) encodes the lane markings, the second (green) encodes the locations of neighboring cars, which are represented as rectangles reflecting the dimensions of each car, and the third channel (blue) represents the ego car, also scaled to the correct dimensions. All images have dimensions $3 \times 117 \times 24$, and are denoted by i_t. [1] Two examples are shown in Figure 8. We also computed vectors $u_t = (p_t, \Delta p_t)$, where $p_t = (x_t, y_t)$ is the position at time t and $\Delta p_t = (x_{t+1} - x_t, y_{t+1} - y_t)$ is the velocity.

Action representation: Each action vector a_t consists of two components: an acceleration (which can be positive or negative) which reflects the change in speed, and a change in angle. The acceleration at a given time step is computed by taking the difference between two consecutive speeds, while the change in angle is computed by projecting the change in speed along its orthogonal direction:

[1] Another possibility would have been to construct feature vectors directly containing the exact coordinates of neighboring cars, however this presents several difficulties. First, cars can enter and exit the neighborhood, and so the feature vector representing the neighboring cars would either have to be dynamically resized or padded with placeholder values. Second, this representation would not be permutation-invariant, and it is unclear where to place a new car entering the frame. Third, encoding the lane information in vector form would require a parametric representation of the lanes, which is more complicated. Using images representations naturally avoids all of these difficulties.



(a) 19.8 km/h (b) 50.3 km/h

Figure 8: Image state representations and proximity cost masks for cars going at different speeds. The higher the speed, the longer the safety distance required to maintain low cost.

$$\begin{aligned} \Delta\text{speed} &= \|\Delta p_{t+1}\|_2 - \|\Delta p_t\|_2 \\ \Delta\text{angle} &= (\Delta p_{t+1} - \Delta p_t)^\top (\Delta p_t)_\perp / \|\Delta p_t\|_2 \\ a_t &= (\Delta\text{speed}, \Delta\text{angle}) \end{aligned}$$

Cost: Our cost function has two terms: a proximity cost and a lane cost. The proximity cost reflects how close the ego car is to neighboring cars, and is computed using a mask in pixel space whose width is equal to the width of a lane and whose height depends on the speed of the car. Two examples are shown in Figure 8. This mask is pointwise multiplied with the green channel, and the maximum value is taken to produce a scalar cost. The lane cost uses a similar mask fixed to the size of the car, and is similarly multiplied with the red channel, thus measuring the car's overlap with the lane. Both of these operations are differentiable so that we can backpropagate gradients with respect to these costs through images predicted by a forward model.

This preprocessing yields a set of state-action pairs (s_t, a_t) (with $s_t = (i_t, u_t)$) for each car, which constitute the dataset we used for training our prediction model. We then use the cost function to optimize action sequences at planning time, using different methods which we describe in Section C.

We now describe how we adapted this dataset to be used as an environment to evaluate planning methods. Building an environment for evaluating policies for autonomous driving is not obvious as it suffers from a cold-start problem. Precisely measuring the performance of a given driving policy would require it to be evaluated in an environment where all other cars follow policies which accurately reflect human behavior. This involves reacting appropriately both to other cars in the environment as well as the car being controlled by the policy being evaluated. However, constructing such an environment is not possible as it would require us to already have access to a policy which drives as humans do, which in some sense is our goal in the first place. One could hand-code a driving policy to control the other cars in the environment, however is it not clear how to do so in a way which accurately reflects the diverse and often unpredictable nature of human driving.

We adopt a different approach where we let all other cars in the environment follow their trajectories in the dataset, while controlling one car with the policy we seek to evaluate. The trajectory of the controlled car is updated as a function of the actions output by the policy, while the trajectories of the other cars remain fixed. If the controlled car collides with another car, this is recorded and

```
observation = env.reset()
while not done:
    action = policy(observation)
    observation, reward, done, info = env.step(action)
    env.render()
```

Figure 9: NGSIM planning environment.

the episode ends. This approach has the advantage that all other cars in the environment maintain behavior which is close to human-like. The one difference with true human behavior is that the other cars do not react to the car being controlled or try to avoid it, which may cause crashes which would not occur in real life. The driving task is thus possibly made more challenging than in a true environment, which we believe is preferable to using a hand-coded policy. The interface is set up the same way as environments in OpenAI Gym (Brockman et al., 2016), and can be accessed with a few lines of Python code, as shown in Figure 9.

B Model Details

The architecture of our forward model consists of four neural networks: a state encoder f_{enc}, an action encoder f_{act}, a decoder f_{dec}, and the posterior network f_ϕ. At every time step, the state encoder takes as input the concatenation of 20 previous states, each of which consists of a context image i_t and a 4-dimensional vector u_t encoding the car's position and velocity. The images $i_{t-20}, ..., i_t$ are run through a 3-layer convolutional network with 64-128-256 feature maps, and the vectors $u_{t-20}, ..., u_t$ are run through a 2-layer fully connected network with 256 hidden units, whose final layers contain the same number of hidden units as the number of elements in the output of the convolutional network (we will call this number n_H). The posterior network takes the same input as the encoder network, as well as the the ground truth state s_{t+1}, and maps them to a distribution over latent variables, from which one sample z_t is drawn. This is then passed through an expansion layer which maps it to a representation of size n_H. The action encoder, which is a 2-layer fully-connected network, takes as input a 2-dimensional action a_t encoding the car's acceleration and change in steering angle, and also maps it to a representation of size n_H. The representations of the input states, latent variable, and action, which are all now the same size, are combined via addition. The result is then run through a deconvolutional network with 256-128-64 feature maps, which produces a prediction for the next image i_{t+1}, and a 2-layer fully-connected network (with 256 hidden units) which produces a prediction for the next state vector u_{t+1}. These are illustrated in Figure B.

The specific updates of the stochastic forward model are given by:

$$(\mu_\phi, \sigma_\phi) = q_\phi(s_{1:t}, s_{t+1}) \tag{3}$$

$$\epsilon \sim \mathcal{N}(0, I) \tag{4}$$

$$z_t = \mu_\phi + \sigma_\phi \cdot \epsilon \tag{5}$$

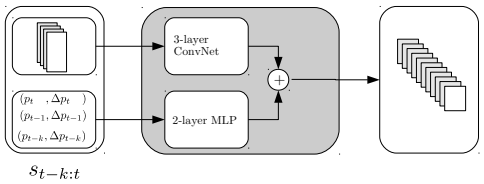
$$\hat{s}_{t+1} = (\tilde{i}_{t+1}, \tilde{u}_{t+1}) = f_\theta(s_{1:t}, a_t, z_t) \tag{6}$$

The per-sample loss is given by:

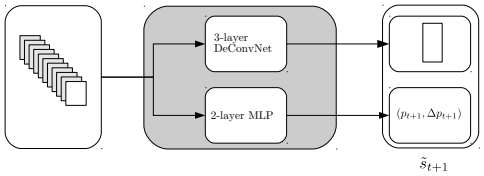
$$\ell(s_{1:t}, s_{t+1}) = \|\tilde{i}_t - i_t\|_2^2 + \|\tilde{u}_t - u_t\|_2^2 + \beta D_{KL}(\mathcal{N}(\mu_\phi, \sigma_\phi)||p(z)) \tag{7}$$

C Planning Approaches

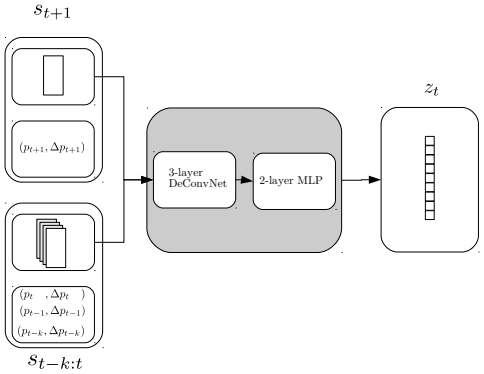
We now describe different planning approaches using our stochastic forward model, with diagrams shown in Figure C. All methods except model-predictive control make use of a policy network which has the same architecture. It consists of an encoder with an identical architecture as f_{enc} used in the forward model (which also takes 20 consecutive states as input), followed by a 3-layer



(a) f_{enc}



(b) f_{dec}



(c) f_ϕ

Figure 10: Individual components of the stochastic prediction model.

fully connected network which outputs a 2-D mean and variance (μ, σ) of a diagonal Gaussian over actions.

Two of the planning methods optimize a cost function C, which is a combination of the proximity and lane costs we described previously, as well as an epistemic uncertainty cost which we describe in the next section. For now, we can treat the cost as a scalar-valued differentiable function of a predicted state.

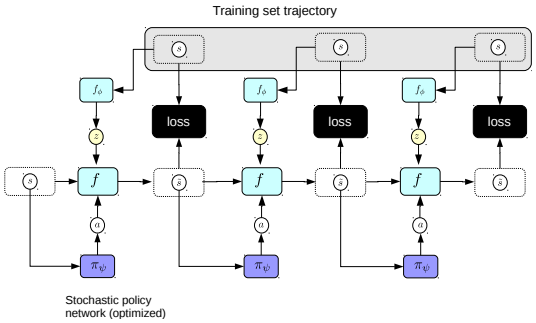
C.1 Single-Step Imitation Learning

The simplest approach to learning a policy from observational data is imitation learning (Pomerleau, 1991), where a network is trained to predict expert actions from states. Here, we give the network a concatenation of 20 states $s_{1:t}$ as input and train it to minimize the negative log-likelihood of the true action observed in the dataset under the parameters of the distribution output by the model:

$$\underset{\psi}{\text{argmin}} \Big[- \log \mathcal{N}(a_{t+1}|\mu, \sigma) \Big], \text{ such that: } (\mu, \sigma) = \pi_\psi(s_{1:t})$$

C.2 Stochastic Model-Based Imitation Learning (SMBIL)

We also experimented with a variant of imitation learning using the learned stochastic model, which we found performed better than the standard version. One issue with imitation learning is that errors can accumulate quickly, leading to divergence from the states seen during training. One cause



(a) Stochastic Model-Based Imitation Learning (SMBIL)

Figure 11: Planning Methods. SMBIL minimizes the distance between training set trajectories and trajectories predicted by the forward model under the current policy network, using latent variables inferred from the training trajectory. The other methods optimize actions or a policy network to minimize the cost predicted by the forward model, using randomly sampled sequences of latent variables.

may be that the model is simply minimizing the ℓ_2 loss in action space, which may not correspond to minimizing distances in trajectory space. Consider the following example where an agent is walking exactly along the side of a cliff, and must output an action which represents its angle. To the left is a drop, and to the right is solid ground. Say the expert action is to go straight, i.e. $a_{t+1} = 0$. Now consider two possible actions predicted by the network, $\hat{a}_{t+1} = -\epsilon$ (slight left) and $\hat{a}_{t+1} = +\epsilon$ (slight right). Both of these actions incur the same ℓ_2 cost of ϵ, but have very different consequences. If the agent moves slightly away from the expert action on the left side, they fall off the cliff, which causes a large deviation of their subsequent states from the expert states. If they move slightly to the right however, they stay on solid ground and their subsequent states remain close to the expert states.

As an alternative, we experimented with training a policy to match expert *trajectories*, rather than actions. We do this by unrolling the forward model for multiple timesteps, outputting actions by the policy network using the model predictions, and minimizing the error between the final trajectory output by the forward model and the expert trajectory observed in the dataset. The motivation here is that if the policy network outputs an action which causes small divergence from the target trajectory at the next timestep, but large divergences later on, it will receive gradients from these larger errors backpropagated through the unrolled forward model.

$$\underset{\psi}{\text{argmin}} \Big[\sum_{i=1}^{T} \ell(s_{t+i}, \hat{s}_{t+1}) \Big], \text{ such that: } \begin{cases} \hat{s}_t = s_t \\ z_{t+i} = f_\phi(\hat{s}_{t+i-1}, \hat{s}_{t+i}) \\ \hat{a}_{t+i} = \pi_\psi(\hat{s}_{t+i-1}) \\ \hat{s}_{t+i} = f(\hat{s}_{t+i-1}, \hat{a}_{t+i}, z_{t+i}) \end{cases}$$

C.3 Stochastic Model-Predictive Control (SMPC)

We also evaluated a receding-horizon model-predictive controller (MPC). At each time step, we optimize a sequence of actions over the next T timesteps under the forward model and execute the first one. Since the model is stochastic, we sample K different sequences of latent variables and optimize the same action sequence averaged over all of them. This requires solving the following optimization problem at each time step t:

$$\underset{a_t,\ldots,a_{t+T}}{\text{argmin}} \Big[\frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{T} C(s^k_{t+i}) \Big], \text{ such that: } \begin{cases} z^k_{t+i} \sim p(z) \\ \hat{s}^k_t = s_t \\ \hat{s}^k_{t+i} = f(\hat{s}^k_{t+i-1}, a_{t+i}, z^k_{t+i}) \end{cases}$$

We solve this optimization problem by performing N steps of gradient descent over the sequence of actions $a_t, ..., a_{t+T}$, during which all the sampled sequences of latent variables are held fixed. Intuitively, this optimization problem can be interpreted as follows. We draw K different sequences of latent variables, which represent K different ways in which the future can unfold. We then optimize a single action sequence, to minimize the predicted costs averaged over each of these possible futures. We thus hope to obtain an action sequence which performs well in many scenarios.

This procedure is unfortunately expensive: it requires a total of $K \times N \times T$ model evaluations. Although the model evaluations corresponding to K different futures can be done in parallel on the GPU, evaluations at different time steps and over different optimization iterations must be done sequentially. We found that maintaining a buffer of previously planned actions allowed us to get better performance with a relatively small number of iterations; this was proposed in (Ohtsuka, 2004). Specifically, at every time step we plan for the next T actions, but only execute the first even though the subsequent ones may be reasonable. We therefore initialize the action sequence to be optimized at the next timestep with the result of the action sequence optimized at the previous time step, shifted by one:

$$(a_t, a_{t+1}, a_{t+2}, ..., a_{T-1}, a_T) \leftarrow (a_{t+1}, a_{t+2}, ..., a_{T-1}, a_T, \mathbf{0}) \tag{8}$$

C.4 Stochastic Value Gradients (SVG)

The last approach which we explored was designed to train a policy network using the learned stochastic forward model, using the framework of Stochastic Value Gradients (Heess et al., 2015). We first randomly sample an initial input state s_t from the training set, sample a sequence of latent variables z to represent a future scenario, and then optimize a paramaterized policy network π_ψ to minimize the cost predicted by the forward model conditioned on this sequence of latent variables.

$$\underset{\psi}{\text{argmin}} \Big[\sum_{i=1}^{T} C(s_{t+i}) \Big], \text{ such that: } \begin{cases} z_{t+i} \sim p(z) \\ \hat{a}_{t+i} = \pi_\psi(s_{t+i-1}) \\ \hat{s}_{t+i} = f(\hat{s}_{t+i-1}, \hat{a}_{t+i}, z_{t+i}) \end{cases}$$

Our approach differs somewhat from the setup of (Heess et al., 2015), who used latent variables inferred from ground truth trajectories as a means to compensate for model errors. We did not find this to be a problem, possibly because we trained the forward model to make 20-step predictions, whereas they trained the forward model to make single-step predictions.

D Training Details

D.1 Forward Model

We trained our prediction model in deterministic mode ($p = 0$) for 200,000 updates, followed by another 200,000 updates in stochastic mode. We save the model after training in deterministic mode and treat it as a deterministic baseline. Our model was trained using Adam (Kingma & Ba, 2014) with learning rate 0.0001 and minibatches of size 64, unrolled for 20 time steps, and with dropout ($p_{dropout} = 0.1$) at every layer, which was necessary for computing the epistemic uncertainty cost during planning.

D.2 Policy Models

All policy networks have the same architecture: a 3-layer ConvNet with feature maps of size 64-128-256, followed by 3 fully-connected layers with 256 hidden units each, with the last layer outputting the parameters of a 2D Gaussian distribution over actions. All policy networks are trained with Adam with learning rate 0.0001. The SMBIL and SVG policies are trained by backpropagation through the unrolled forward model using the reparamaterization trick (Kingma & Welling, 2013). The single-step imitation learner is trained to directly minimize the negative log-likelihood of the ground truth action in the dataset under the parameters output by the policy network. Both SMPC

and SVG models unroll the forward model for 20 steps, and we report results for different unrolling lengths for SMBIL policies. The gradient descent procedure for the SMPC planner performs 5 gradient steps using Adam with learning rate 0.1. Actions are initialized to zero for the first time step, after which we used the rotating action buffer described in Section C.3. We sampled 10 sequences of latent variables.

TODO:

- Mikael: (high priority) improve writing to incorporate model-based imitation learning
- Mikael: (high priority) Update figures to reflect that we are learning the cost predictor
- Mikael: (high priority) Quantify the change in action sensitivity when using inferred vs. sampled latent variables
- Mikael: (medium priority) Add baseline without uncertainty regularizer where we just penalize the ℓ_2 norm of the actions.
- Alfredo: (high priority) Add model-free baselines
- Alfredo: (high priority) Get rule-based baseline from Etienne
- Alfredo: (high priority) Update figure 4 (delete middle image and make ego car blue)