MODEL-BASED PLANNING UNDER ALEATORIC AND EPISTEMIC UNCERTAINTY

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-based reinforcement learning has the potential to reduce the number of environment interactions an agent needs to learn an effective policy. However, there exist settings, such as autonomous driving, where even a single suboptimal action executed in a real environment can have unacceptable consequences. At the same time, there exist many demonstrations of expert behavior which can be used for learning.

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 games (Mnih et al., 2015; 2016) to learning complex locomotion tasks (Lillicrap 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. Although it may be possible to build a simulator where the agent can safely try out policies without facing real consequences, this requires human engineering effort which increases with the complexity of the environment being modeled.

Model-based reinforcement approaches try to learn a model of the environment dynamics, and then use this model to plan actions or train a parameterized policy (Nguyen & Widrow, 1990; Schmidhuber, 1990; Jordan & Rumelhart, 1992). A common setting is where an agent alternates between collecting experience by executing actions produced by 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, there are many settings where observational data of an environment is abundant. This suggests a need for algorithms which can perform the majority of their learning from observational data, and achieve good empirical performance right away. 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 from purely observational data is challenging because the data may only cover a small region of the space it lives in. If the observational data contains expert trajectories, one option is to use imitation learning (Pomerleau, 1991). However, passive imitation learning can suffer from a mismatch between input states seen at training and execution time, and may need to be augmented by interacting with an expert (**?**). Another option is to learn a dynamics model from observational data. However, it may make arbitrary predictions outside the domain it was trained on, which may wrongly be associated with high rewards (or low cost). A planner or 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

added to the training set, which will correct the environment model in those areas. However, this does not solve the problem if the dataset of environment interactions which the model is trained on is fixed.

In this work, we propose an end-to-end, model-based approach to learning driving policies from purely observational data. We first introduce a large-scale dataset of real-world driving trajectories, which we also adapt into a interactive 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. To handle this, we use recent advances in stochastic video prediction to learn an action-conditional stochastic forward model which can then be used for planning. We furthermore propose adding a differentiable term in the cost function used at planning time, which represents the epistemic uncertainty of the forward model. This 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 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 Dataset and Planning Environment

To begin with, we summarize the dataset and planning environment which we used, with full details provided in Appendix A. 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 aleatoric uncertainty. 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, represented as sequences of coordinates, which are then split into training (80%), validation (10%) and testing sets (10%).

We then applied additional preprocessing to obtain suitable 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 6.

We also define a cost function with two terms: 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. Both costs are computed using masks in pixel space, which are pointwise multiplied with the green channel for the proximity cost and the red channel for the lane cost. These operations are differentiable so that we can backpropagate gradients with respect to both costs through images predicted by a forward 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

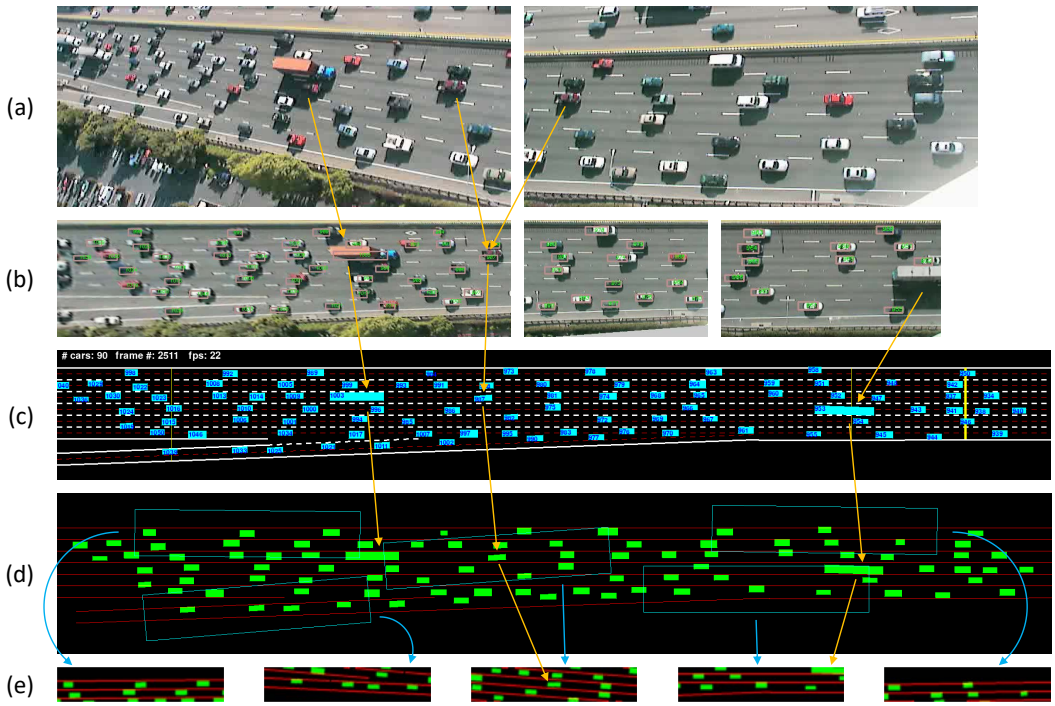


Figure 1: Preprocessing pipeline for the NGSIM-I80 data set. Orange arrows show same vehicles across stages. Blue arrows show corresponding extracted context state. (a) Snapshots from two of the seven cameras. (b) View point transformation, car localisation and tracking. (c) Every vehicle in the simulator is initialised with starting position, initial velocity, dimensions, and ID number. (d) Context states are extracted from rectangular regions surrounding each vehicle. (e) Five examples of context states extracted at the previous stage. Notice how vehicles oriented slightly to the left (2nd and 3rd examples) have lane markings rotated to the right.

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 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.

3 Stochastic Forward Model

3.1 Stochastic Forward Model

For our dynamics model, we adapted recent models for stochastic video prediction (Babaeizadeh et al., 2017; Denton & Fergus, 2018) based on Variational Autoencoders (Kingma & Welling, 2013). Our prediction model $f_\theta(s_{1:t}, a_t, z_t)$ takes as input a sequence of past states $s_{1:t}$, an action a_t, and 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. 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. Specifically, the updates are given by:

$$
\begin{aligned}
(\mu_\phi, \sigma_\phi) &= q_\phi(s_{1:t}, s_{t+1}) && (1)\\
\epsilon &\sim \mathcal{N}(0, I) && (2)\\
z_t &= \mu_\phi + \sigma_\phi \cdot \epsilon && (3)\\
\tilde{s}_{t+1} = (\tilde{i}_{t+1}, \tilde{u}_{t+1}) &= f_\theta(s_{1:t}, a_t, z_t) && (4)
\end{aligned}
$$

The per-sample loss is given by:

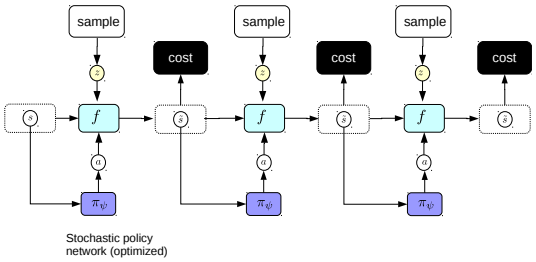
$$
\ell(s_{1:t}, s_{t+1}, a_t) = \|\tilde{i}_t - i_t\|_2^2 + \|\tilde{u}_t - u_t\|_2^2 + \beta D_{KL}(\mathcal{N}(\mu_\phi, \sigma_\phi)||p(z)) \tag{5}
$$

After training, different future predictions for a given sequence of frames can be generated by sampling different latent variables from the prior distribution. Details of the prediction model and inference network architectures are given in Appendix B.

3.2 Training a Policy Network

Once a stochastic forward model is trained, we can use it to train a parameterized policy network π_ψ. We do this by unrolling the forward model for T steps, at each step sampling a latent variable from $p(z)$. The actions input to the forward model are produced by the policy network. 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.

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



(a) Stochastic Value Gradients (SVG)

Figure 2: 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.

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.

4 Differentiable Epistemic Uncertainty Cost

We propose to augment the proximity and lane costs described in Section A with an additional term which penalizes the uncertainty of the forward model, calculated using dropout, and use derivatives of this cost for planning. Dropout (Hinton et al., 2012; Srivastava et al., 2014) consists of randomly setting hidden units in a neural network to zero with some probability, and has been shown to substantially reduce overfitting. 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 epistemic uncertainty can be obtained by calculating the variance of an output given the same input over multiple dropout masks. More precisely, let $f_{\theta_1}, ..., f_{\theta_K}$ be the same model f_θ with K different dropout masks, and for some input x, let $y_k = f_{\theta_k}(x)$. The uncertainty of the original model about a given prediction $y = f_\theta(x)$ is given by the covariance matrix of the outputs under different dropout masks (up to some constants):

$$\text{Cov}[y_1, ..., y_K] = \frac{1}{K}\Big[\sum_{k=1}^{K} y_k y_k^\top\Big] - \Big[\frac{1}{K}\sum_{k=1}^{K} y_k\Big]\Big[\frac{1}{K}\sum_{k=1}^{K} y_k)\Big]^\top \tag{6}$$

We note that this uncertainty estimate is the composition of differentiable functions: each of the models with different dropout masks 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 provides a scalar estimate of uncertainty which is also differentiable with respect to the inputs. This allows us to obtain gradients of the inputs with respect to this measure of epistemic uncertainty, which gives us directions in input space which would make the model more certain.

In the context of our stochastic forward model, we define an epistemic uncertainty estimate as follows:

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

where D is the dimensionality of the output. Minimizing this quantity with respect to actions encourages a planning algorithm or policy network to produce actions which, when plugged into the forward model, will produce predictions which the model is confident about. To compensate for differences in baseline uncertainty across different modalities or rollout lengths, for each modality 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 epistemic uncertainty cost as follows:

$$C_U(s_{1:t}, a_t, z_t) = \Big[\frac{\Omega(s_{1:t}, a_t, z_t) - \mu_\Omega^t}{\sigma_\Omega^t}\Big]_+ \tag{7}$$

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.

5 Related Work

In recent years, several works have explored prediction of complex time series such as video in the deterministic setting (Kalchbrenner et al., 2016; Srivastava et al., 2015; Denton & Birodkar, 2017). Others have included latent variables as a means to model aleatoric uncertainty, using the framework of Generative Adversarial Networks (Mathieu et al., 2015) or Variational Autoencoders (Villegas et al., 2017; Babaeizadeh et al., 2017; Denton & Fergus, 2018). These works have focused on predicting future frames rather than planning. Other works have learned action-conditional forward models which are then used for planning (Oh et al., 2015; Finn et al., 2016; Agrawal et al., 2016; Kalchbrenner et al., 2016; Srinivas et al., 2018). These have primarily focused on deterministic environments and do not account for model uncertainty when planning.

Our approach to learning policies which minimize predicted cost fits within the conceptual framework of Stochastic Value Gradients (SVG) (Heess et al., 2015), and extends it to a setting with high-dimensional state representations. This requires us to use more sophisticated stochastic models than the ones in the original work, which used additive Gaussian noise whose parameters were learned using the reparamaterization trick. They also considered an online setting where the agent continues to collect experience, which reduces the need for the epistemic uncertainty penalty, whereas we found this to be essential in our setting where we learn purely from observational data.

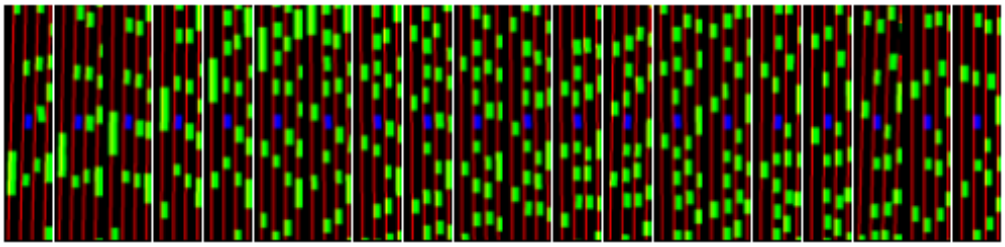
The works of (McAllister & Rasmussen, 2016; Chua et al., 2018) also used dropout masks to account for epistemic uncertainty in the context of model-based reinforcement learning, but did so during the forward prediction step. Namely, they used different dropout masks to simulate different state trajectories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

Our epistemic 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 stochastic forward model at every time step, and backpropagate gradients through the unrolled forward model to either optimize actions or train a policy network by gradient descent.

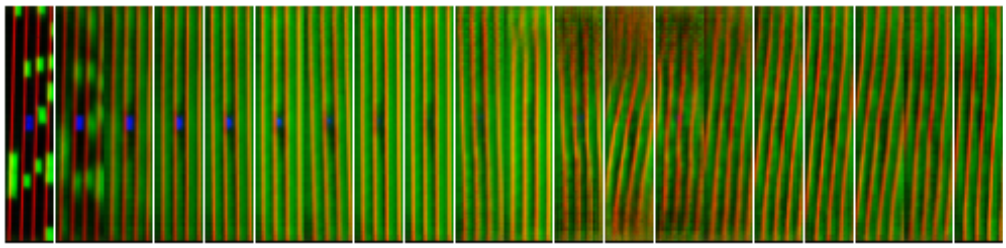
6 Experiments

6.1 Prediction Results

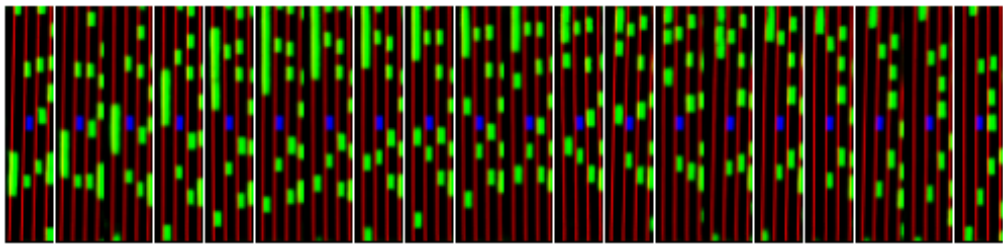
Training Details: We trained our model in deterministic mode ($p = 0$) for 200,000 updates, followed by another 200,000 updates in stochastic mode with probability of masking the z vector set to $p = 0.5$ and dimensionality of z set to 32. We save the model after training in deterministic mode and treat 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



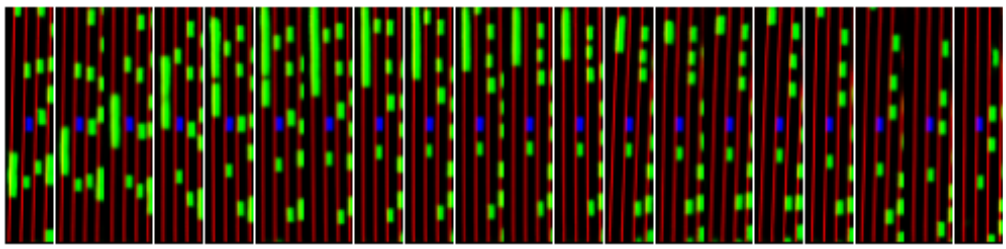
(a) Ground truth sequence



(b) Deterministic Model



(c) Stochastic Model, sample 1



(d) Stochastic Model, sample 2

Figure 3: 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.

($p_{dropout} = 0.1$) at every layer, which was necessary for computing the epistemic uncertainty cost during planning.

Figure 6.1 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`.

Method	Mean Distance	Success Rate (%)
Human	1358.5	100.0
No action	56.6	16.2
Rule-Based Policy	-	-
A3C	-	-
A3C + IL warmstart	-	-
DDPG	-	-
DDPG + warmstart buffer	-	-
1-step IL	322.2	1.4
SMBIL (1 step)	T	T
SMBIL (3 step)	T	T
SMBIL (5 step)	T	T
SMBIL (10 step)	T	T
SMBIL (20 step)	T	T
SVG(∞)	103.0	0.0
SVG(∞) + Epistemic Cost	525.8	14.5
SVG-sim(∞)	88.7	0.0
SVG-sim(∞) + Epistemic Cost	974.6	60.3

Table 1: Planning performance on NGSIM dataset, measured in mean episode length. An episode ends when the controlled car collides with another car, drives off the road or reaches the end of the road segment.

6.2 Planning Results

We now give results for planning, with performance measured in terms of episode length. An episode terminates when a car collides with another car, drives off the road or reaches the end of the road segment. Lower episode lengths indicate more collisions. All cars are initialized at the beginning of the road segment with the initial speed they were driving at in the dataset, and then controlled by the policy being measured. Since there is considerable variation in episode length, we report both the median and the mean. Table 1 compares performance for the different methods described in Section C.

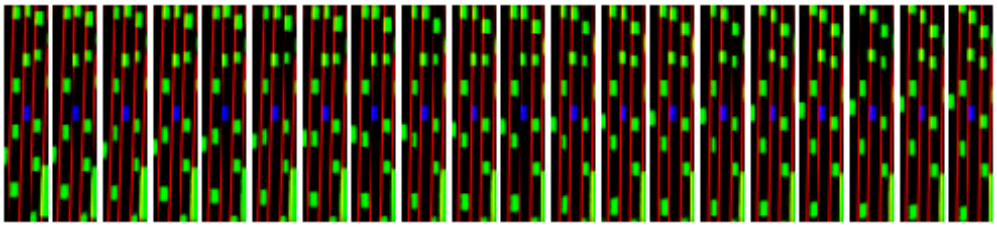
Both the SMPC and SVG methods require optimizing a differentiable cost function. We used the following:

$$C = C_{proximity} + 0.1 \cdot C_{lane} + \lambda_U \cdot C_U \tag{8}$$

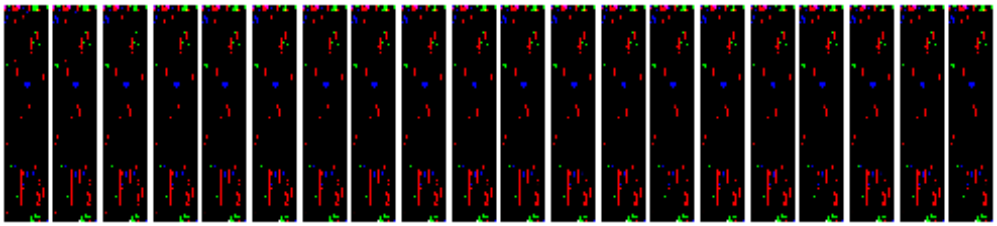
where $C_{proximity}$ and C_{lane} are the proximity and lane costs described in Section A, and C_U is the differentiable epistemic uncertainty cost described in Section 5.5. We compare results with $\lambda_U = 0$ and $\lambda_U > 0$ to show its effect.

Training Details: 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. We also provide a comparison without the buffer in the next section.

The 1-step imitation learner performs similarly to the SMBIL policy with a single step prediction. Training SMBIL policies with longer rollouts improves performance up to a point, but still does not beat the simple baseline of performing no action. Both SMPC and SVG without the uncertainty



(a) $\lambda_U = 0.01$



(b) $\lambda_U = 0$

Method	Ep. Length	Predicted cost	C_U
SVG-TEN, $\lambda_U = 0.01$	29.6	0.0830	1.451
SVG-TEN, $\lambda_U = 0$	2.6	0.0005	8717

Figure 4: 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.

cost also give poor performance. However, adding the uncertainty cost improves performance dramatically, outperforming all other methods by a wide margin. Planning with the SMPC was very computationally demanding and the results we report are only over the first 100 trajectories (out of 278) in the test set, which still took over 36 hours wallclock time. In contrast, SVG with the uncertainty penalty is both fast and effective. 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 forward model for an action sequence produced by SVG policy networks trained with and without the epistemic uncertainty cost, shown in Figure 4. 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 4 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.

We additionally performed several ablation experiments to understand the effects of different modeling choices, shown in table 2. Using the action buffer has a non-negligible effect on the SMPC planning, as we see a performance drop when starting from newly initialized actions every time planning is required rather than initializing with a previously optimized action sequence. It is possible that with larger numbers of gradient steps this difference in performance would decrease, but the current setup is already very expensive. We also compared our TEN model to a VAE model with the

Method	Ep. Length	Planning Time
SMPC-TEN	28.6	2000
SMPC-VAE	28.4	2000
SMPC-TEN, no action buffer	22.6	2000
SVG-TEN	29.6	1
SVG-TEN, no z masking	27.5	1

Table 2: Ablation experiments. SMPC-VAE represents the SMPC method using a VAE instead of a TEN as a forward model.

same architecture and hyperparameters, except for the β hyperparameter representing the weight of the KL term in the loss, which we set to 10^{-6} (we optimized over the range $\{1, 10^{-1}, ..., 10^{-6}\}$ and found that values higher than 10^{-5} produced blurry predictions similar to the deterministic model. Performance is very similar for both models.

Finally, we compare SVG performance with and without masking the z variables during training. Not including the masking causes a performance drop due to the forward model becoming less sensitive to the actions.

7 Conclusion

In this chapter 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 a stochastic forward model, and using it for planning together with a new uncertainty regularizer which solves the domain mismatch problem. There are several directions for future work. First, 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.

8 General formatting instructions

The text must be confined within a rectangle 5.5 inches (33 picas) wide and 9 inches (54 picas) long. The left margin is 1.5 inch (9 picas). Use 10 point type with a vertical spacing of 11 points. Times New Roman is the preferred typeface throughout. Paragraphs are separated by 1/2 line space, with no indentation.

Paper title is 17 point, in small caps and left-aligned. All pages should start at 1 inch (6 picas) from the top of the page.

Authors' names are set in boldface, and each name is placed above its corresponding address. The lead author's name is to be listed first, and the co-authors' names are set to follow. Authors sharing the same address can be on the same line.

Please pay special attention to the instructions in section 10 regarding figures, tables, acknowledgments, and references.

9 Headings: first level

First level headings are in small caps, flush left and in point size 12. One line space before the first level heading and 1/2 line space after the first level heading.

9.1 HEADINGS: SECOND LEVEL

Second level headings are in small caps, flush left and in point size 10. One line space before the second level heading and 1/2 line space after the second level heading.

9.1.1 HEADINGS: THIRD LEVEL

Third level headings are in small caps, flush left and in point size 10. One line space before the third level heading and 1/2 line space after the third level heading.

10 CITATIONS, FIGURES, TABLES, REFERENCES

These instructions apply to everyone, regardless of the formatter being used.

10.1 CITATIONS WITHIN THE TEXT

Citations within the text should be based on the `natbib` package and include the authors' last names and year (with the "et al." construct for more than two authors). When the authors or the publication are included in the sentence, the citation should not be in parenthesis (as in "See **?** for more information."). Otherwise, the citation should be in parenthesis (as in "Deep learning shows promise to make progress towards AI (**?**).").

The corresponding references are to be listed in alphabetical order of authors, in the REFERENCES section. As to the format of the references themselves, any style is acceptable as long as it is used consistently.

10.2 FOOTNOTES

Indicate footnotes with a number[1] in the text. Place the footnotes at the bottom of the page on which they appear. Precede the footnote with a horizontal rule of 2 inches (12 picas).[2]

10.3 FIGURES

All artwork must be neat, clean, and legible. Lines should be dark enough for purposes of reproduction; art work should not be hand-drawn. The figure number and caption always appear after the figure. Place one line space before the figure caption, and one line space after the figure. The figure caption is lower case (except for first word and proper nouns); figures are numbered consecutively.

Make sure the figure caption does not get separated from the figure. Leave sufficient space to avoid splitting the figure and figure caption.

You may use color figures. However, it is best for the figure captions and the paper body to make sense if the paper is printed either in black/white or in color.

10.4 TABLES

All tables must be centered, neat, clean and legible. Do not use hand-drawn tables. The table number and title always appear before the table. See Table 3.

Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lower case (except for first word and proper nouns); tables are numbered consecutively.

11 DEFAULT NOTATION

In an attempt to encourage standardized notation, we have included the notation file from the textbook, *Deep Learning* **?** available at `https://github.com/goodfeli/dlbook_`

[1] Sample of the first footnote

[2] Sample of the second footnote

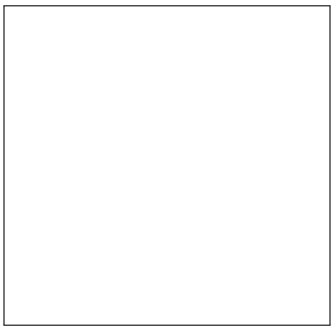


Figure 5: Sample figure caption.

Table 3: Sample table title

PART	DESCRIPTION
Dendrite	Input terminal
Axon	Output terminal
Soma	Cell body (contains cell nucleus)

notation/. Use of this style is not required and can be disabled by commenting out math_commands.tex.

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor A
$\mathsf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{X}} y$	Tensor containing derivatives of y with respect to X
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x}\sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P\|Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\mathrm{condition}}$	is 1 if the condition is true, 0 otherwise

12 FINAL INSTRUCTIONS

Do not change any aspects of the formatting parameters in the style files. In particular, do not modify the width or length of the rectangle the text should fit into, and do not change font sizes (except perhaps in the REFERENCES section; see below). Please note that pages should be numbered.

13 PREPARING POSTSCRIPT OR PDF FILES

Please prepare PostScript or PDF files with paper size "US Letter", and not, for example, "A4". The -t letter option on dvips will produce US Letter files.

Consider directly generating PDF files using `pdflatex` (especially if you are a MiKTeX user). PDF figures must be substituted for EPS figures, however.

Otherwise, please generate your PostScript and PDF files with the following commands:

```
dvips mypaper.dvi -t letter -Ppdf -G0 -o mypaper.ps
ps2pdf mypaper.ps mypaper.pdf
```

13.1 MARGINS IN LATEX

Most of the margin problems come from figures positioned by hand using `\special` or other commands. We suggest using the command `\includegraphics` from the graphicx package.

Always specify the figure width as a multiple of the line width as in the example below using .eps graphics

```
\usepackage[dvips]{graphicx} ...
\includegraphics[width=0.8\linewidth]{myfile.eps}
```

or

```
\usepackage[pdftex]{graphicx} ...
\includegraphics[width=0.8\linewidth]{myfile.pdf}
```

for .pdf graphics. See section 4.4 in the graphics bundle documentation (`http://www.ctan.org/tex-archive/macros/latex/required/graphics/grfguide.ps`)

A number of width problems arise when LaTeX cannot properly hyphenate a line. Please give LaTeX hyphenation hints using the `\-` command.

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.

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 Vighnesh Birodkar. Unsupervised learning of disentangled representations from video. *CoRR*, abs/1705.10915, 2017. URL http://arxiv.org/abs/1705.10915.

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

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.

Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *CoRR*, abs/1610.00527, 2016. URL http://arxiv.org/abs/1610.00527.

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.

Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015. URL http://arxiv.org/abs/1511.05440.

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.

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.

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.

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.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR*, abs/1502.04681, 2015. URL http://arxiv.org/abs/1502.04681.

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.

Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. Learning to generate long-term future via hierarchical prediction. *CoRR*, abs/1704.05831, 2017. URL http://arxiv.org/abs/1704.05831.

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 6. 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

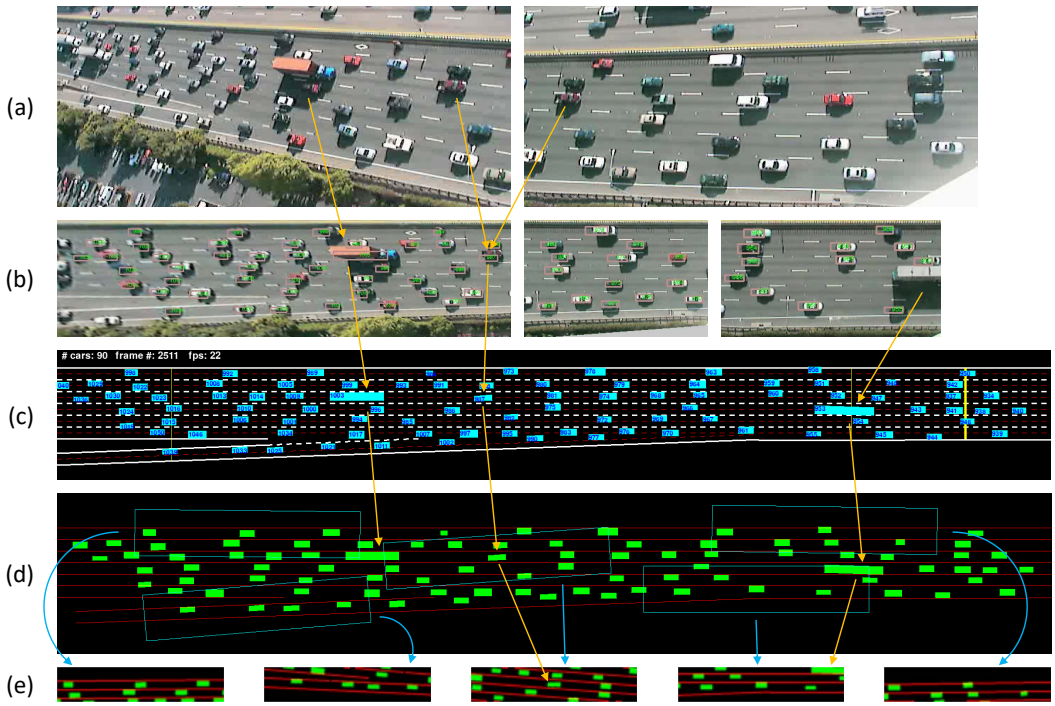
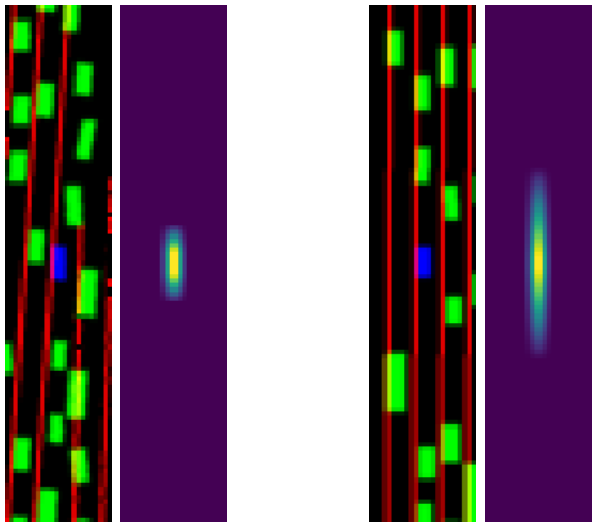


Figure 6: Preprocessing pipeline for the NGSIM-I80 data set. Orange arrows show same vehicles across stages. Blue arrows show corresponding extracted context state. (a) Snapshots from two of the seven cameras. (b) View point transformation, car localisation and tracking. (c) Every vehicle in the simulator is initialised with starting position, initial velocity, dimensions, and ID number. (d) Context states are extracted from rectangular regions surrounding each vehicle. (e) Five examples of context states extracted at the previous stage. Notice how vehicles oriented slightly to the left (2nd and 3rd examples) have lane markings rotated to the right.



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

Figure 7: 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.

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. [3] Two examples are shown in Figure 7. 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:

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

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 7. 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.

[3] 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.

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

Figure 8: NGSIM planning environment.

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

Figure 9: NGSIM planning environment.

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.

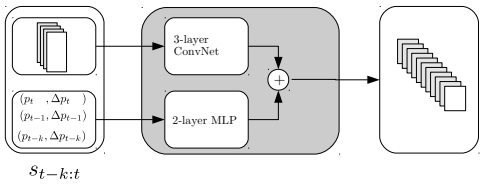
We now describe how we adapted this dataset to be used as an environment to evaluate planning methods. Building an environment for evaluating policies for autonomous driving is not obvious as it suffers from a cold-start problem. Precisely measuring the performance of a given driving policy would require 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 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.

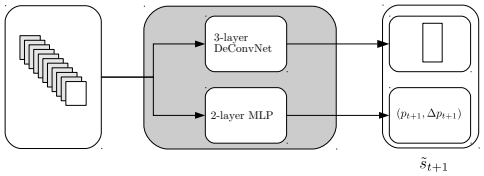
B Model Details

C Planning Approaches

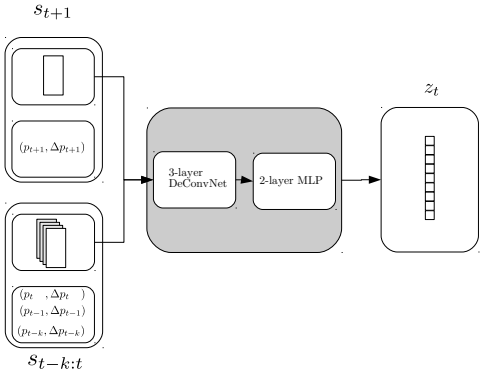
We now describe different planning approaches using our stochastic forward model, with diagrams shown in Figure C. All methods except model-predictive control make use of a policy network which has the same architecture. It consists of an encoder with an identical architecture as f_{enc} used in the forward model (which also takes 20 consecutive states as input), followed by a 3-layer fully connected network which outputs a 2-D mean and variance (μ, σ) of a diagonal Gaussian over actions.



(a) f_{enc}



(b) f_{dec}



(c) f_ϕ

Figure 10: Individual components of the stochastic prediction model.

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.

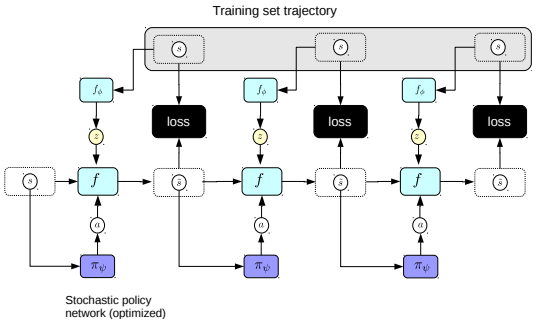
C.1 Single-Step Imitation Learning

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:

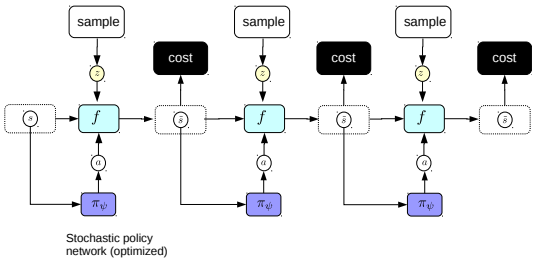
$$\underset{\psi}{\text{argmin}} \Big[- \log \mathcal{N}(a_{t+1}|\mu, \sigma) \Big], \text{ such that: } (\mu, \sigma) = \pi_\psi(s_{1:t})$$

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

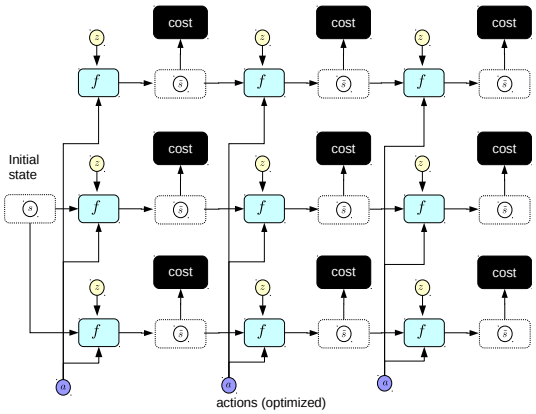
We also experimented with a variant of imitation learning using the learned stochastic model, which we found performed better than the standard version. One issue with imitation learning is that errors can accumulate quickly, leading to divergence from the states seen during training. One cause may be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



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



(b) Stochastic Value Gradients (SVG)



(c) Stochastic Model-Predictive Control (SMPC)

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.

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, $\tilde{a}_{t+1} = -\epsilon$ (slight left) and $\tilde{a}_{t+1} = +\epsilon$ (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.

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

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:

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

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:

$$(a_t, a_{t+1}, a_{t+2}, ..., a_{T-1}, a_T) \leftarrow (a_{t+1}, a_{t+2}, ..., a_{T-1}, a_T, \mathbf{0}) \tag{9}$$

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.

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

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.