Learning Latent Dynamics for Planning from Pixels

Danijar Hafner
Google Brain
mail@danijar.com

Timothy Lillicrap
DeepMind

Ian Fischer
Google Research

Ruben Villegas
Google Brain

David Ha
Google Brain

Honglak Lee
Google Brain

James Davidson
Google Brain

Abstract

Planning has been very successful for control tasks with known environment dynamics. To leverage planning in unknown environments, the agent needs to learn the dynamics from interactions with the world. However, learning dynamics models that are accurate enough for planning has been a long-standing challenge, especially in image-based domains. We propose the Deep Planning Network (PlaNet), a purely model-based agent that learns the environment dynamics from pixels and chooses actions through online planning in latent space. To achieve high performance, the dynamics model must accurately predict the rewards ahead for multiple time steps. We approach this problem using a latent dynamics model with both deterministic and stochastic transition function and a generalized variational inference objective that we name latent overshooting. Using only pixel observations, our agent solves continuous control tasks with contact dynamics, partial observability, and sparse rewards. PlaNet uses significantly fewer episodes and reaches final performance close to and sometimes higher than top model-free algorithms.

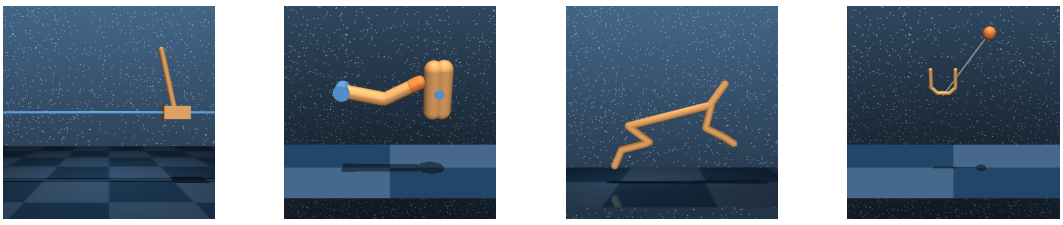
1 Introduction

Planning is a natural and powerful approach to decision making problems with known dynamics, such as game playing and simulated robot control (Tassa et al., 2012; Silver et al., 2017; Moravčík et al., 2017). To plan in unknown environments, the agent needs to learn the dynamics from experience. Learning dynamics models that are accurate enough for planning has been a long-standing challenge. Some of the difficulties include model inaccuracies, accumulating errors of multi-step predictions, capturing multiple plausible futures, and overconfident predictions outside of the training distribution.

Planning using learned models offers several potential benefits over model-free reinforcement learning. First, model-based planning can be more data efficient because it leverages a richer training signal and does not require propagating rewards through Bellman backups. Moreover, planning carries the promise of increasing performance just by increasing the computational budget for searching for actions, as shown by Silver et al. (2017). Finally, learned dynamics can be independent of any specific task and thus have the potential to transfer well to other tasks in the same environment.

Recent work has shown promise in learning the dynamics of simple low-dimensional environments (Gal et al., 2016; Amos et al., 2018; Chua et al., 2018; Henaff et al., 2018). However, these typically assume access to the underlying state of the world and the reward function, which may not be available in practice. In high-dimensional environments, the dynamics can be learned in a compact latent space to enable fast planning. The success of such latent models has been limited to simple tasks such as balancing cartpoles and controlling 2-link arms from dense rewards (Watter et al., 2015; Banijamali et al., 2017).

In this paper, we propose the Deep Planning Network (PlaNet), a model-based agent that learns the environment dynamics from pixels and chooses actions through online planning in latent space. To



(a) Cartpole (b) Finger (c) Cheetah (d) Ball in cup

Figure 1: Image-based control domains considered in our experiments. (a) For the cartpole balance and swingup tasks, the camera is fixed so the cart might move out of sight. (b) The finger spinning task includes contact dynamics between the finger and the object. (c) The cheetah forward and backward running tasks include both contact dynamics and a larger number of joints. (d) The ball-in-cup task has a sparse reward that is only given once the ball is caught.

learn the dynamics, we use a transition model with both stochastic and deterministic components and train it using a generalized variational objective that encourages multi-step predictions. PlaNet solves several continuous control tasks from pixels that are more difficult than those previously solved by planning with learned models.

The key contributions of this paper are:

- **Planning in latent spaces.** We train a latent dynamics model while collecting data online using planning in latent space. Our agent solves several environments with contact dynamics, ground friction, and sparse rewards from the DeepMind control suite (Tassa et al., 2018), as shown in Figure 1. It significantly outperforms the model-free algorithm A3C (Mnih et al., 2016) and in some cases D4PG (Barth-Maron et al., 2018) in terms of final performance, while using $50\times$ less environment interaction and the same amount of computation time. The results show that learning latent dynamics models for planning in image domains is a promising approach.
- **Recurrent state space model.** Purely stochastic dynamics models tend to yield inconsistent trajectories and diverging state sequences due to sampled outliers. On the other hand, stochasticity can be important to capture partial observability, provide robustness during planning, and represent unpredictable parts of the environment. We design a latent dynamics model with both deterministic and stochastic components (Buesing et al., 2018; Chung et al., 2015). Our experiments indicate this to be crucial for high planning performance.
- **Latent overshooting.** For planning, we need the model to accurately predict ahead for multiple time steps. However, the standard variational bound for latent sequence models only trains one-step predictions. For a model of limited capacity and restricted posterior family, the solution to this objective does not necessarily coincide with the best multi-step predictions (Bengio et al., 2015; Talvitie, 2014). We generalize the variational bound to include all multi-step predictions. Using only the terms in latent space results in a fast and effective regularizer that improves long-term predictions and is compatible with any latent sequence model.

2 Latent Space Planning

To solve unknown environments via planning, we need to learn a model of the environment dynamics from experience. Since the agent may not initially visit all parts of the environment, we need to iteratively collect new experience and refine the model. We do this by planning with the current partially trained model. Starting from a few seed episodes collected under random actions, we train the dynamics model and add one additional episode to the data set every C update steps, as outlined in Algorithm 1. In this section, we introduce notation for the environment and describe the planning algorithm.

Since individual image observations generally do not reveal the full state of the environment, we consider a partially observable Markov decision process (POMDP). We define a discrete time step t, hidden states s_t, image observations o_t, continuous action vectors a_t, and scalar rewards r_t, that

Algorithm 1: Deep Planning Network (PlaNet)

Input : Environment, planner, initial state model $p_\theta(s_1)$, transition model $p_\theta(s_t \mid s_{t-1}, a_{t-1})$, observation model $p_\theta(o_t \mid s_t)$, encoder $q_\theta(s_t \mid o_{1:t}, a_{1:t-1})$, reward predictor $r_\theta(s_t)$, exploration noise source $\epsilon \sim p(\epsilon)$, action repeat R, number of seed episodes S, data collection interval C, batch size B, chunk length L.

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize model parameters θ randomly.
while *not converged* **do**
  `// Learning`
  for *update step* $s = 1..C$ **do**
    Randomly draw sequence chunks $\{(o_t, a_t, r_t)_{t=k}^{L+k}\}_{i=1}^{B} \sim \mathcal{D}$ from the dataset.
    Compute loss $\mathcal{L}(\theta)$ using Equation 6.
    Update model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$.
  `// Collection`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..\frac{T}{R}$ **do**
    Infer belief over current state $q(s_t \mid o_{1:t}, a_{1:t-1})$ from history.
    $a_t \leftarrow$ `planner(`$q(s_t \mid o_{1:t}, a_{1:t-1}), p_\theta, r_\theta$`)`, see Algorithm 2.
    Add exploration noise $\epsilon \sim p(\epsilon)$ to action.
    for *action repeat* $k = 1..R$ **do**
      $r_t^k, o_{t+1}^k \leftarrow$ `env.step(`a_t`)`
    $r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_{t+1}^R$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$

follow the stochastic dynamics

$$s_1 \sim \mathrm{p}(s_1) \quad o_t \sim \mathrm{p}(o_t \mid s_t) \quad r_t = \mathrm{r}(s_t) \quad a_t \sim \mathrm{p}(a_t \mid o_{\leq t}, a_{<t}) \quad s_t \sim \mathrm{p}(s_t \mid s_{t-1}, a_{t-1}). \quad (1)$$

The goal is to implement a policy $p(a_t \mid o_{\leq t}, a_{<t})$ that maximizes the expected sum of rewards $\mathrm{E_p}\big[\sum_{\tau=t+1}^{T} \mathrm{r}(s_\tau)\big]$. We implement the policy as a model-predictive control (MPC) planner (Richards, 2005) that repeatedly searches for the best action sequence under the learned dynamics model. It then executes the first action of the found sequence in the environment and replans. This allows the agent to adapt its plan based on the new observation. In contrast to model-free and hybrid reinforcement learning algorithms, there is no learned function that directly predicts actions.

To evaluate a candidate action sequence under the learned model, we sample one or more future state trajectories, starting from the current state belief $q(s_t \mid o_{\leq t}, a_{<t})$. Section 3 discusses how this belief is inferred from the history. For each state trajectory, we compute the sum of predicted rewards, and then average them. Because the reward is modeled as function of the latent state, the planner can operate purely in latent space without generating images, which allows for fast evaluation of large batches of action sequences.

As outlined in Algorithm 2, we use the cross entropy method (CEM) (Rubinstein, 1997; Chua et al., 2018) to search for the best action sequence under the model. CEM is a population-based optimization algorithm that infers a distribution over action sequences that maximize the expected sum of rewards. For this, we define a time-dependent diagonal Gaussian belief over action sequences $a_{t:t+H} \sim \mathrm{Normal}(\mu_{t:t+H}, \sigma^2_{t:t+H}\mathbb{I})$, where t is the current time step of the agent and H is the length of the planning horizon. Starting from zero mean and unit variance, we repeatedly sample J candidate action sequences, evaluate them under the model, and re-fit the belief to the top K action sequences. After I iterations, the planner returns the mean of the belief for the first time step μ_t. Since we are using a population-based optimizer, we found it sufficient to predict a single future per action sequence.

During evaluation, we directly execute the action that the planner returns in the environment. When collecting new episodes for the data set, we add small Gaussian exploration noise to the action. After executing the action, the agent receives the next observation and repeats the planning procedure. Importantly, the belief over action sequences starts from zero mean and unit variance again to avoid local optima. To reduce the planning horizon and support model learning, we repeat each action R

Algorithm 2: Cross entropy method planner

Input : Current state belief $q_\theta(s_t \mid o_{1:t}, a_{1:t-1})$, transition model $p_\theta(s_t \mid s_{t-1}, a_{t-1})$, reward predictor $r_\theta(s_t)$, planning horizon distance H, optimization iterations I, candidates per iteration J, number of candidates to re-fit K.

Initialize factorized belief over action sequences $q(a_{t:t+H}) \leftarrow \mathrm{Normal}(0, \mathbb{I})$.
for *optimization iteration* $i = 1..I$ **do**
  `// Evaluate` J `action sequences from the current belief.`
  for *candidate action sequence* $j = 1..J$ **do**
    $a^{(j)}_{t:t+H} \sim q(a_{t:t+H})$
    $s^{(j)}_{t:t+H+1} \sim q_\theta(s_t \mid o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p_\theta(s_\tau \mid s_{\tau-1}, a^{(j)}_{\tau-1})$
    $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} r_\theta(s^{(j)}_\tau)$
  `// Re-fit belief to the` K `best action sequences.`
  $\mathcal{K} \leftarrow \mathrm{argsort}(\{R^{(j)}\}_{j=1}^{J})_{1:K}$
  $\mu_{t:t+H} = \frac{1}{K} \sum_{k \in \mathcal{K}} a^{(k)}_{t:t+H}, \quad \sigma_{t:t+H} = \frac{1}{K-1} \sum_{k \in \mathcal{K}} |a^{(k)}_{t:t+H} - \mu_{t:t+H}|.$
  $q(a_{t:t+H}) \leftarrow \mathrm{Normal}(\mu_{t:t+H}, \sigma^2_{t:t+H}\, \mathbb{I})$
return *first action mean* μ_t.

times, as is common in reinforcement learning (Mnih et al., 2015; 2016). The next section introduces the latent dynamics model that our planner operates on.

3 Recurrent State Space Model

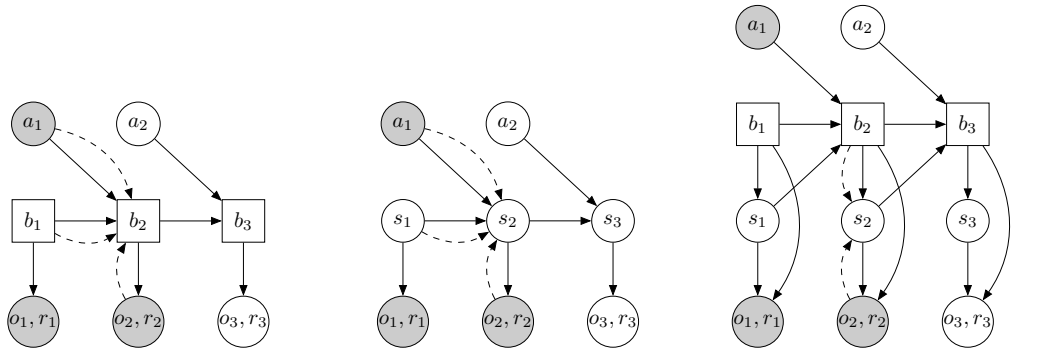
For planning, we need to evaluate thousands of action sequences at every time step of the agent. We use a recurrent state-space model (RSSM) for this that can predict forward purely in latent space, similar to recently proposed models (Karl et al., 2016; Buesing et al., 2018; Doerr et al., 2018). Instead of an extensive comparison to prior architectures, we highlight two findings that can guide future model designs: our experiments show that both stochastic and deterministic paths in the transition function are crucial for successful planning. In this section, we introduce notation, remind the reader of latent state-space models, and then describe our dynamics model.

We consider sequences $\{o_t, a_t, r_t\}_{t=1}^{T}$ with discrete time step t, high-dimensional image observations o_t, continuous action vectors a_t, and scalar rewards r_t. A typical latent state-space model as shown in Figure 2b resembles the structure of a partially observable Markov decision process. It defines the generative process of the observations and rewards using a state sequence $\{s_t\}_{t=1}^{T}$,

$$s_1 \sim p(s_1) \qquad s_t \sim p(s_t \mid s_{t-1}, a_{t-1}) \qquad o_t \sim p(o_t \mid s_t) \qquad r_t = r(s_t), \tag{2}$$

where the initial state distribution $p(s_1)$ is a standard multivariate Gaussian, the transition function $p(s_t \mid s_{t-1}, a_{t-1})$ is Gaussian with mean and variance parameterized by a feed-forward neural network, the decoder $p(o_t \mid s_t)$ is Gaussian with mean parameterized by a deconvolutional neural network and identity covariance, and the reward predictor $r(s_t)$ is a feed-forward neural network trained via mean squared error.

Since the states are unknown, we cannot learn this model directly by maximizing the data likelihood. Instead, we use a convolutional encoder $q(s_t \mid s_{t-1}, a_{t-1}, o_t)$ to construct a variational bound on the data log-likelihood. Note that we use the filtering posterior here since we are ultimately interested in planning using the model, but one may also use the full smoothing posterior during training (Babaeizadeh et al., 2017). For simplicity, we omit first time steps and write losses for predicting only the observations — the reward losses follow by analogy. The variational bound obtained using



(a) Deterministic model (RNN) (b) Stochastic model (SSM) (c) Recurrent state-space model (RSSM)

Figure 2: Latent dynamics model designs. In this example, the model observes the first two time steps and predicts the third. Circles represent stochastic variables and squares deterministic variables. Solid lines denote the generative process and dashed lines the inference network. (a) Transitions in a normal recurrent neural network are fully deterministic. This prevents the model from capturing multiple futures and makes it prone to exploitation by the planner. (b) Transitions in a normal state-space model are fully stochastic. This can cause open-loop predictions to diverge due to latent outliers. (c) Our model combines both stochastic and deterministic paths in the transition function, allowing the model to robustly generate multiple futures.

Jensen's inequality is

$$\begin{aligned}
\ln p(o_{1:T} \mid a_{1:T}) &= \ln \int \prod_{t=1}^{T} p(o_t \mid s_t) p(s_t \mid s_{t-1}, a_{t-1}) \frac{q(s_t \mid o_{\leq t}, a_{<t})}{q(s_t \mid o_{\leq t}, a_{<t})} \, ds_{1:T} \\
&\geq \int \sum_{t=1}^{T} q(s_t \mid o_{\leq t}, a_{<t}) \ln \left(p(o_t \mid s_t) p(s_t \mid s_{t-1}, a_{t-1}) \frac{1}{q(s_t \mid o_{\leq t}, a_{<t})} \right) ds_{1:T} \\
&= \sum_{t=1}^{T} \Big(\underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t}, a_{<t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}[q(s_t \mid o_{\leq t}, a_{<t}) \parallel q(s_t \mid o_{\leq t-1}, a_{<t})]}_{\text{complexity}} \Big),
\end{aligned} \tag{3}$$

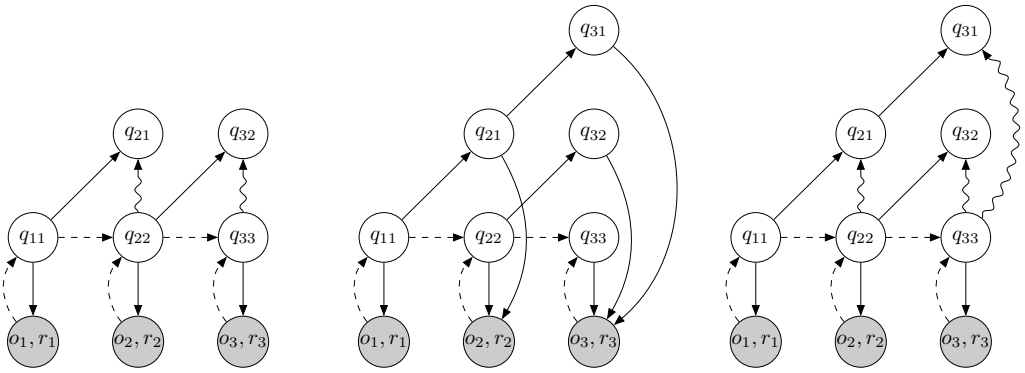
where $q(s_t \mid o_{\leq t-1}, a_{<t}) = \int p(s_t \mid s_{t-1}, a_{t-1}) q(s_{t-1} \mid o_{\leq t-1}, a_{<t-1}) \, ds_{t-1}$ is the temporal one-step prior. Estimating the expectation using a single reparameterized sample from the approximate posterior yields an efficient objective for inference and learning in non-linear latent variable models that can be optimized using gradient ascent (Kingma and Welling, 2013; Rezende et al., 2014).

Despite its generality, a practical limitation of this model is that when autoregressively sampling a state sequence an outlier may cause the sequence to diverge because the transition function was not trained on such states. This motivates including a deterministic sequence $\{b_t\}_{t=1}^{T}$ that allows the model to remember information not just about the last state but all previous states (Chung et al., 2015; Buesing et al., 2018). We use such a model shown in Figure 2c that we name recurrent state-space model (RSSM),

$$s_1 \sim p(s_1 \mid b_1) \quad b_t = b(b_{t-1}, s_{t-1}, a_{t-1}) \quad s_t \sim p(s_t \mid b_t) \quad o_t \sim p(o_t \mid b_t, s_t) \quad r_t = r(b_t, s_t), \tag{4}$$

where $b(b_{t-1}, s_{t-1}, a_{t-1})$ is implemented as a recurrent neural network (RNN). We use a convolutional encoder $q(s_t \mid b_t, o_t)$ to predict the approximate state posteriors. For simplicity, we choose to only consider the past images but not the past rewards when inferring the state belief.

The model can be trained using the same loss function (Equation 3). In addition, we add a fixed global prior to prevent the posteriors from collapsing in near-deterministic environments. This adds additional KL-divergence loss terms from each posterior to a fixed standard Normal distribution. Another interpretation of this is to define the prior at each time step as product of the learned temporal prior and the global fixed prior. In the next section, we identify a limitation of the standard variational bound for sequence models and propose a generalization of it that leads to improved long-term predictions.



(a) Standard variational bound (b) Observation overshooting (c) Latent overshooting

Figure 3: Unrolling schemes. The labels are short for multi-step priors $q_{ij} = q(s_i \mid o_{\leq j}, a_{<i})$. Arrows pointing to filled circles indicate log-likelihood loss terms. Wavy lines indicate KL-divergence loss terms. (a) The standard variational objectives decodes the posterior at every step to compute the reconstruction loss. It also places a KL on the prior and posterior at every step, which trains the transition function for one-step predictions. (b) Observation overshooting decodes all multi-step states to apply additional reconstruction losses. This is very expensive for image-based models. (c) Latent overshooting predicts all multi-step priors. These state beliefs are trained towards their corresponding posteriors in latent space to encourage accurate multi-step predictions.

4 Latent Overshooting

In the previous section, we derived the typical variational bound for learning and inference in latent sequence models (Equation 3). As show in Figure 3a, this objective function contains reconstruction terms for the observations and KL-divergence regularizers for the approximate posteriors. A limitation of this objective is that the transition function $p(s_t \mid s_{t-1}, a_{t-1})$ is only trained via the KL-divergence regularizers for one-step predictions. In this section, we generalize this variational bound to *variational overshooting* to include all multi-step predictions, and then develop *latent overshooting*, a similar objective that is fast to compute.

If we could train our model to make perfect one-step predictions, it would also make perfect multi-step predictions, so this would not be a problem. However, when using a model with limited capacity and restricted distributional family, training the model only on one-step predictions until convergence does in general not coincide with the model that is best at multi-step predictions. For successful planning, we need accurate multi-step predictions. Therefore, we take inspiration from Amos et al. (2018) and earlier related ideas (Chiappa et al., 2017; Villegas et al., 2017; Lamb et al., 2016), and train all multi-step predictions of the model. We develop this idea for latent sequence models, we name variational overshooting.

Variational overshooting is an objective function for latent sequence models that generalizes the standard variational bound (Equation 3) to train the model on all possible multi-step predictions,

$$\ln p(o_{1:T} \mid a_{1:T}) \geq$$
$$\frac{1}{D} \sum_{t=1}^{T} \sum_{d=1}^{D} \Big(\underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t-d+1}, a_{<t})}[\ln p(o_t \mid s_t)]}_{\text{observation overshooting}} - \underbrace{D_{\mathrm{KL}}[q(s_t \mid o_{\leq t}, a_{<t}) \parallel q(s_t \mid o_{\leq t-d}, a_{<t})]}_{\text{latent overshooting}} \Big), \quad (5)$$

where $q(s_t \mid o_{\leq t-d}, a_{<t})$ is the temporal prior for the current state after seeing only observations up to and including step $t - d$. This multi-step prior is computed from the approximate posterior $q(s_{t-d} \mid o_{\leq t-d}, a_{<t-d})$ by repeatedly applying the transition function to a state sample. Variational overshooting trains multi-step image predictions of the model and adds the corresponding KL regularizers between closed loop posterior and open loop temporal prior. Since the objective is an average over variational bounds with different priors, it is itself a lower bound.

Evaluating the image prediction terms can be expensive in practice. We therefore separate the image prediction terms that we name *observation overshooting* from the KL-regularizer terms that we name *latent overshooting*. Latent overshooting can be interpreted as a regularizer in latent space that

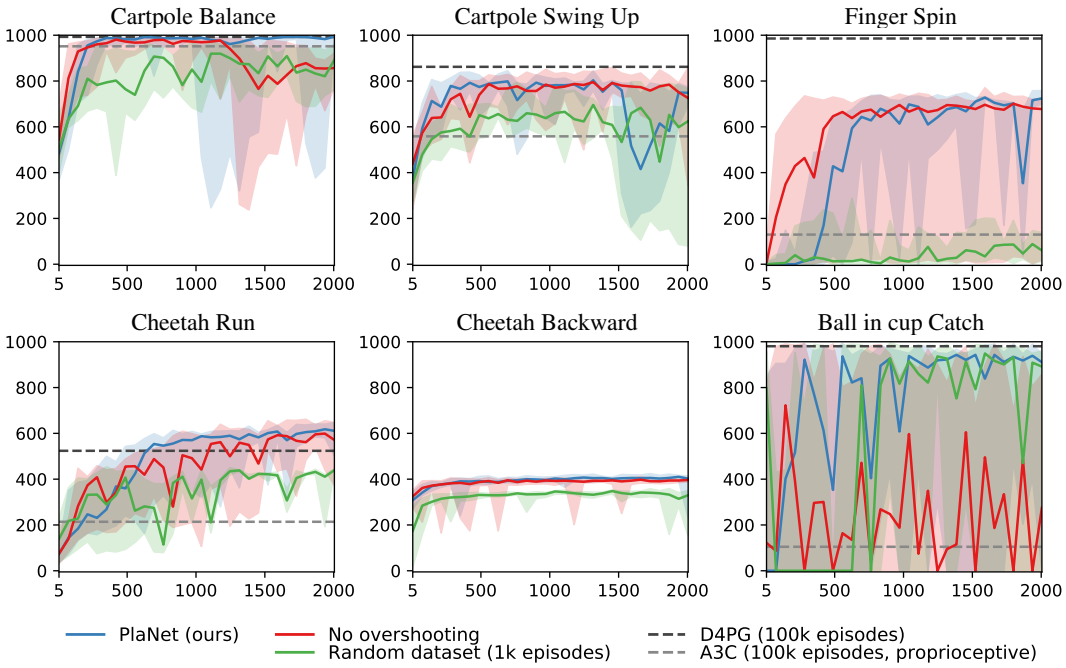


Figure 4: Comparison of agent designs to model-free algorithms. Plots show test performance for the number of collected episodes. We compare PlaNet using latent overshooting (Equation 6), a version with standard variational objective (Equation 3), and a version that trains from a random data set of 1000 episodes rather than collecting experience during training. The lines show medians and the areas show percentiles 5 to 95 over 4 seeds and 10 rollouts.

encourages consistency between closed and open loop predictions,

$$\ln p(o_{1:T} \mid a_{1:T}) \geq \sum_{t=1}^{T} \Bigg(\underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t}, a_{<t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} - \underbrace{\frac{1}{D} \sum_{d=1}^{D} \beta_d D_{\mathrm{KL}}[q(s_t \mid o_{\leq t}, a_{<t}) \parallel q(s_t \mid o_{\leq t-d}, a_{<t})]}_{\text{latent overshooting}} \Bigg), \quad (6)$$

where we include weighting factors $\{\beta_d\}_{d=1}^{D}$ analogously to the β-VAE by Higgins et al. (2016). These can all be set to the same value, or chosen to let the model focus more on long-term or short-term predictions.

Since latent overshooting can be seen as Equation 3 with a mixture of multiple temporal priors, it again remains a lower bound on the observation log-likelihood. In practice, we stop gradients of the posterior distributions for overshooting distances $d > 1$, so that the open loop predictions are trained towards closed loop predictions, but not the other way around. For model-based planning, we observe the effect of latent overshooting in the form of lower variance, sometimes slower initial training, and often higher final task performance. We leave schedules of the β_d weights that could accelerate initial learning for future work.

5 Model-based RL Experiments

We evaluate PlaNet on six continuous control tasks from pixels. We explore multiple design axes of the agent: online experience collection, the latent overshooting objective, and stochastic and deterministic paths in the dynamics model. In this section, we focus on planning performance. We refer to the appendix for video predictions and hyper parameters. We use the same hyper parameters for all tasks, except for the action repeat. Within $50\times$ fewer episodes, PlaNet outperforms A3C (Mnih et al., 2016) and achieves similar performance to the top model-free algorithm D4PG (Barth-Maron et al., 2018). The training time of 24 hours on a single Nvidia V100 GPU matches that of D4PG. Our implementation uses TensorFlow Probability (Dillon et al., 2017) and will be open sourced. Please visit `https://danijar.com/planet` for videos of the trained agents.

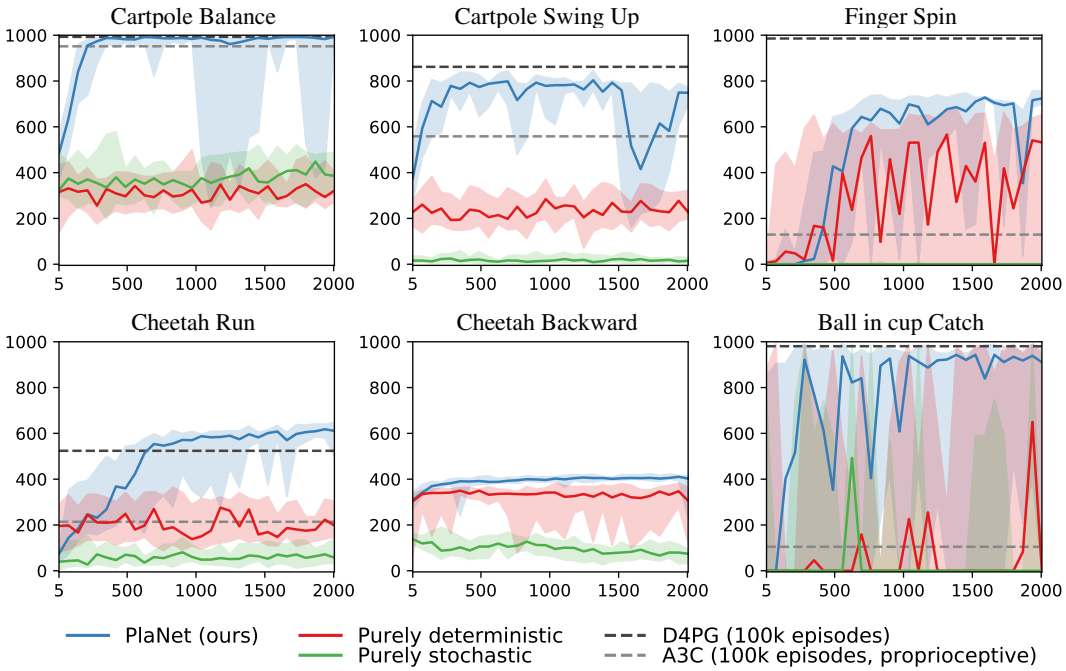


Figure 5: Comparison of model designs. Plots show test performance for the number of collected episodes. We compare PlaNet using our RSSM (Section 3) to purely deterministic (RNN) and purely stochastic models (SSM). The RNN does not use latent overshooting, as it does not have stochastic latents. The lines show medians and the areas show percentiles 5 to 95 over 4 seeds and 10 rollouts.

For our evaluation, we consider six image-based continuous control tasks of the DeepMind control suite (Tassa et al., 2018), shown in Figure 1. These environments provide qualitatively different challenges. The cartpole swingup task requires a long planning horizon and to memorize the cart when it is out of view, the finger spinning task includes contact dynamics between the finger and the object, the cheetah tasks exhibit larger state and action spaces, and the ball-in-cup task only has a sparse reward for when the ball is caught. For the cheetah backward running task, which is not included in the control suite, we simply negate the velocity before passing it into the reward function of the forward running task.

Figure 4 compares the performance of PlaNet to the model-free algorithms reported by Tassa et al. (2018). Within 400 episodes, PlaNet outperforms the policy-gradient method A3C, trained from proprioceptive states for 100,000 episodes, on all tasks. After 2,000 episodes, it achieves similar performance to D4PG, trained from images for 100,000 episodes, except for the finger task. On the cheetah running task, PlaNet surpasses the final performance of D4PG with a relative improvement of 19%. We refer to Table 1 for numerical results, which also includes the performance of CEM planning with the true dynamics of the simulator.

Figure 4 additionally compares PlaNet with latent overshooting to versions with standard variational objective (Equation 3), and with a random data set rather than collecting experience online. We observe that online data collection helps all tasks and is necessary for the finger task. Latent overshooting reduces variance in the performance and is necessary for successful planning on the sparse-reward ball-in-cup task. It also slows down initial learning on some of the tasks, but increases final performance on the cartpole balance and cheetah tasks.

Figure 5 compares design choices of the dynamics model. For this, we train PlaNet using our recurrent state-space model (RSSM), as well as versions with purely deterministic GRU (Cho et al., 2014) (RNN), and purely stochastic state-space model (SSM). We observe the importance of both stochastic and deterministic elements in the transition function on all tasks. The stochastic component might help because all tasks are stochastic from the agent's perspective due to partial observability of the start state. The noise might also add a margin to the planning objective that results in more robust action sequences. The deterministic part is even more important — the agent does not train without it.

6 Related Work

Previous work in model-based reinforcement learning has focused on planning in low-dimensional state spaces (Gal et al., 2016; Higuera et al., 2018; Henaff et al., 2018; Chua et al., 2018), combining the benefits of model-based and model-free approaches (Kalweit and Boedecker, 2017; Nagabandi et al., 2017; Weber et al., 2017; Kurutach et al., 2018; Buckman et al., 2018; Ha and Schmidhuber, 2018; Wayne et al., 2018; Igl et al., 2018; Srinivas et al., 2018), and pure video prediction without planning (Oh et al., 2015; Krishnan et al., 2015; Karl et al., 2016; Chiappa et al., 2017; Babaeizadeh et al., 2017; Gemici et al., 2017; Denton and Fergus, 2018; Buesing et al., 2018; Doerr et al., 2018; Gregor and Besse, 2018). Appendix B reviews these orthogonal research directions in more detail.

Relatively few works have demonstrated successful planning from pixels using learned dynamics models. The robotics community focuses on video prediction models for planning (Agrawal et al., 2016; Finn and Levine, 2017; Ebert et al., 2017) that deal with visual complexity and solve simple tasks, such as grasping or pushing objects. In comparison, we focus on simulated environments, where we scale to larger state and action spaces, as well as sparse reward functions. E2C (Watter et al., 2015) and RCE (Banijamali et al., 2017) embed images into a latent space, where they learn local-linear latent transitions and plan for actions using LQR. These methods balance simulated cartpoles and control 2-link arms from images, but have been difficult to scale up. We lift the Markov assumption of these models, making our method applicable under partial observability, and present results on more challenging environments that include longer planning horizons, contact dynamics, and sparse rewards.

7 Discussion

We presented PlaNet, a model-based agent that learns a latent dynamics model from image observations and chooses actions by planning in latent space. To enable accurate long-term predictions, we chose a model with both stochastic and deterministic paths and train it using our proposed latent overshooting objective. We show that our agent is successful at several continuous control tasks from image observations, reaching performance that is comparable to the best model-free algorithms while using $50\times$ fewer episodes.

Directions for future work include learning temporal abstraction instead of using a fixed action repeat, possibly through hierarchical models. Moreover, our work provides a starting point for multi-task control by sharing the dynamics model and learning multiple reward predictors. While the results are encouraging for the field of model-based reinforcement learning, further innovations are necessary to catch up with model-free algorithms on control tasks of higher difficulty.

Acknowledgements We thank Jacob Buckman, Nicolas Heess, Shane Gu, and Chelsea Finn for helpful discussions.

Table 1: Comparison of PlaNet to model-free algorithms. We use the results for A3C and D4PG reported by Tassa et al. (2018). We include CEM planning ($H = 12, I = 10, J = 1000, K = 100$) with the true simulator instead of learned dynamics as an estimated upper bound on performance.

Method	**Modality**	**Episodes**	**Cartpole Balance**	**Cartpole Swingup**	**Finger Spin**	**Cheetah Run**	**Cheetah Backward**	**Ball in cup Catch**
A3C	proprioceptive	100,000	952	558	129	214	—	105
D4PG	pixels	100,000	993	862	985	524	—	980
PlaNet (ours)	pixels	2,000	990	780	558	633	408	914
CEM + true simulator	simulator state	0	998	850	825	656	656	993

References

P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

B. Amos, L. Dinh, S. Cabi, T. Rothörl, A. Muldal, T. Erez, Y. Tassa, N. de Freitas, and M. Denil. Learning awareness models. In *International Conference on Learning Representations*, 2018.

M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell, and S. Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

E. Banijamali, R. Shu, M. Ghavamzadeh, H. Bui, and A. Ghodsi. Robust locally-linear controllable embedding. *arXiv preprint arXiv:1710.05373*, 2017.

G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

L. Buesing, T. Weber, S. Racaniere, S. Eslami, D. Rezende, D. P. Reichert, F. Viola, F. Besse, K. Gregor, D. Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

E. Denton and R. Fergus. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018.

J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. Probabilistic recurrent state-space models. *arXiv preprint arXiv:1801.10395*, 2018.

F. Ebert, C. Finn, A. X. Lee, and S. Levine. Self-supervised visual planning with temporal skip connections. *arXiv preprint arXiv:1710.05268*, 2017.

C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

Y. Gal, R. McAllister, and C. E. Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

M. Gemici, C.-C. Hung, A. Santoro, G. Wayne, S. Mohamed, D. J. Rezende, D. Amos, and T. Lillicrap. Generative temporal models with memory. *arXiv preprint arXiv:1702.04649*, 2017.

K. Gregor and F. Besse. Temporal difference variational auto-encoder. *arXiv preprint arXiv:1806.03107*, 2018.

D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

M. Henaff, W. F. Whitney, and Y. LeCun. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*, 2018.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2016.

J. C. G. Higuera, D. Meger, and G. Dudek. Synthesizing neural network controllers with probabilistic model based reinforcement learning. *arXiv preprint arXiv:1803.02291*, 2018.

M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for pomdps. *arXiv preprint arXiv:1806.02426*, 2018.

N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

G. Kalweit and J. Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206, 2017.

M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

A. M. Lamb, A. G. A. P. GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609, 2016.

M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

T. M. Moerland, J. Broekens, and C. M. Jonker. Learning multimodal transition dynamics for model-based reinforcement learning. *arXiv preprint arXiv:1705.00470*, 2017.

M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

A. G. Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354, 2017.

A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

E. Talvitie. Model regularization for stable sample rollouts. In *UAI*, pages 780–789, 2014.

Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE, 2012.

Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

A. van den Oord, O. Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6309–6318, 2017.

A. Venkatraman, M. Hebert, and J. A. Bagnell. Improving multi-step prediction of learned time series models. In *AAAI*, pages 3024–3030, 2015.

R. Villegas, J. Yang, Y. Zou, S. Sohn, X. Lin, and H. Lee. Learning to generate long-term future via hierarchical prediction. *arXiv preprint arXiv:1704.05831*, 2017.

C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, 2016.

M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

G. Wayne, C.-C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, J. Rae, P. Mirowski, J. Z. Leibo, A. Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

A Hyper Parameters

The dynamics model is trained using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 2×10^{-3} on batches of $B = 50$ sequence chunks of length $L = 50$. We pre-process images by reducing the bit depth to 5 bits as in (Kingma and Dhariwal, 2018). We use the convolutional and deconvolutional networks from Ha and Schmidhuber (2018), a GRU (Cho et al., 2014) with 200 units in the dynamics model, and implement all other functions as two dense layers of size 200 with ReLU activations (Nair and Hinton, 2010). Distributions in latent space are 30-dimensional diagional Gaussians with predicted mean and standard deviation. For latent overshooting, we use $D = 50$ and set $\beta_1 = 0.5$ and $\beta_{>1} = 0.1$. The divergence terms to the fixed global prior are scaled by 0.1.

For planning, we use CEM with horizon length $H = 12$, optimization iterations $I = 10$, candidate samples $J = 1000$, and refitting to the best $K = 100$. We start from $S = 5$ seed episodes with random actions and collect another episode every $C = 50$ update steps with $\epsilon \sim \mathrm{Normal}(0, 0.3)$ action noise. The action repeat differs between domains: cartpole ($R = 8$), finger ($R = 2$), cheetah ($R = 4$), ball-in-cup ($R = 6$). We find important hyper parameters to be the learning rate, the KL-divergence scales, and the action repeat.

B Additional Related Work

Planning in state space When low-dimensional states of the environment are available to the agent, it is possible to learn the dynamics directly in state space. In the regime of control tasks with only a few state variables, such as the cart pole and mountain car tasks, PILCO (Deisenroth and Rasmussen, 2011) achieves remarkable sample efficiency using Gaussian processes to model the dynamics. Similar approaches using neural networks dynamics models can solve two-link balancing problems (Gal et al., 2016; Higuera et al., 2018) and implement planning via gradients (Henaff et al., 2018). Chua et al. (2018) use ensembles of neural networks, scaling up to the cheetah running task. The limitation of these methods is that they access the low-dimensional Markovian state of the underlying system and sometimes the reward function. Amos et al. (2018) train a deterministic dynamics model using observation overshooting for active exploration with a robotics hand. We move beyond low-dimensional proprioceptive representations and use a latent dynamics model to solve control tasks from images.

Hybrid agents The challenges of model-based RL have motivated the research community to develop hybrid agents that accelerate policy learning by training on imagined experience (Kalweit and Boedecker, 2017; Nagabandi et al., 2017; Kurutach et al., 2018; Buckman et al., 2018; Ha and Schmidhuber, 2018), improving feature representations (Wayne et al., 2018; Igl et al., 2018), or leveraging the information content of the model directly (Weber et al., 2017). Srinivas et al. (2018) learn a policy network with integrated planning computation using reinforcement learning and without prediction loss, yet require expert demonstrations for training.

Multi-step predictions Training sequence models on multi-step predictions has been explored for several years. Scheduled sampling (Bengio et al., 2015) changes the rollout distance of the sequence model over the course of training. Hallucinated replay (Talvitie, 2014) mixes predictions into the data set to indirectly train multi-step predictions. Venkatraman et al. (2015) take an imitation learning approach. Recently, Amos et al. (2018) train a dynamics model on all multi-step predictions at once. We generalize this idea to latent sequence models trained via variational inference.

Latent sequence models Classic work has explored models for non-Markovian observation sequences, including recurrent neural networks (RNNs) with deterministic hidden state and probabilistic state-space models (SSMs). The ideas behind variational autoencoders (Kingma and Welling, 2013; Rezende et al., 2014) have enabled non-linear SSMs that are trained via variational inference (Krishnan et al., 2015). The VRNN (Chung et al., 2015) combines RNNs and SSMs and is trained via variational inference. In contrast to our RSSM, it feeds generated observations back into the model which makes forward predictions expensive. Karl et al. (2016) address mode collapse to a single future by restricting the transition function, (Moerland et al., 2017) focus on multi-modal transitions, and Doerr et al. (2018) stabilize training of purely stochastic models. Buesing et al. (2018) propose a model similar to ours but use in a hybrid agent instead for explicit planning.

Video prediction Video prediction is an active area of research in deep learning. Oh et al. (2015) and Chiappa et al. (2017) achieve visually plausible predictions on Atari games using deterministic models. Kalchbrenner et al. (2016) introduce an autoregressive video prediction model using gated CNNs and LSTMs. Recent approaches introduce stochasticity to the model to capture multiple futures (Babaeizadeh et al., 2017; Denton and Fergus, 2018). To obtain realistic predictions, Mathieu et al. (2015) and Vondrick et al. (2016) use adversarial losses. In simulated environments, Gemici et al. (2017) augment dynamics models with an external memory to remember long-time contexts. van den Oord et al. (2017) propose a variational model that avoids sampling using a nearest neighbor look-up, yielding high fidelity image predictions. These models are complimentary to our approach.

C Video Predictions

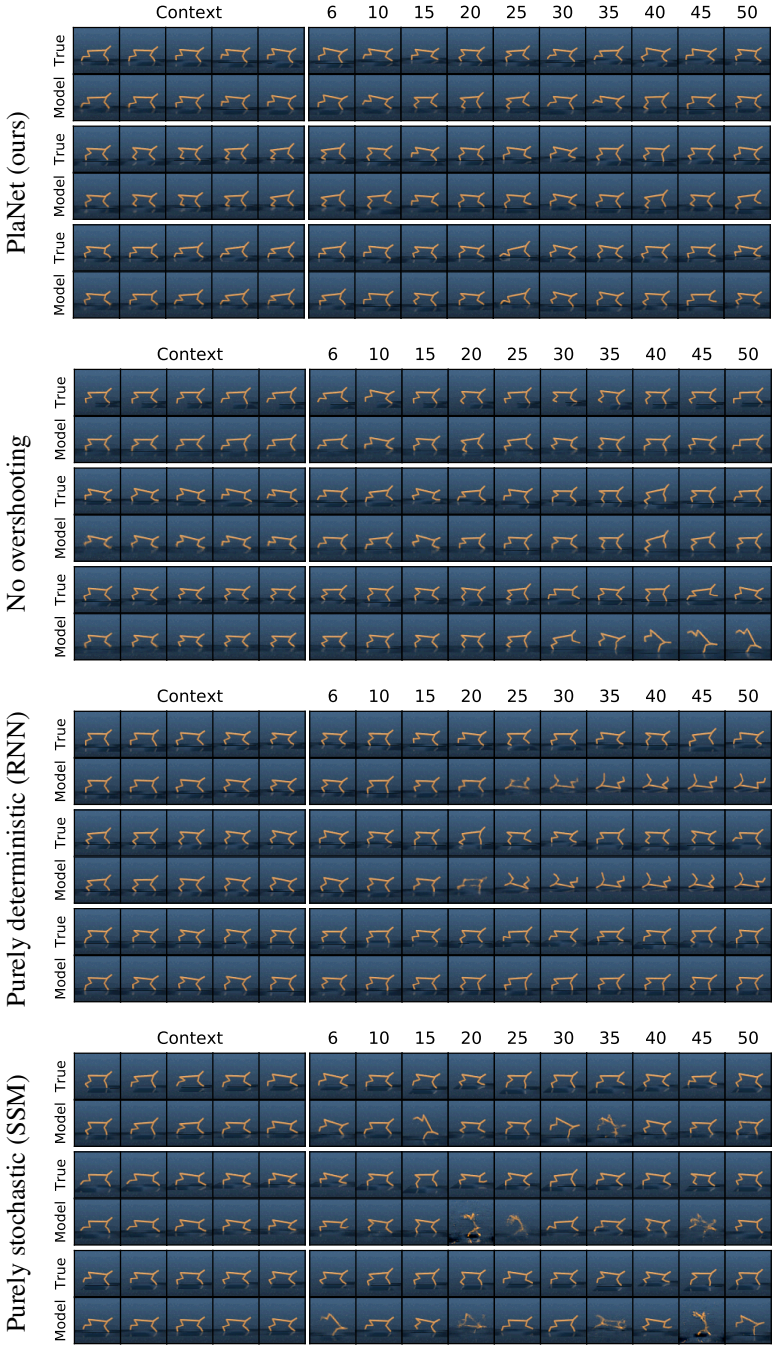


Figure 6: Open-loop video predictions for test episodes. The columns 1–5 show reconstructed context frames and the remaining images are generated open-loop. Latent overshooting enables pixel-accurate predictions for 50 steps into the future in the cheetah environment. We randomly selected action sequences from test episodes collected with action noise alongside the training episodes.

D State Diagnostics

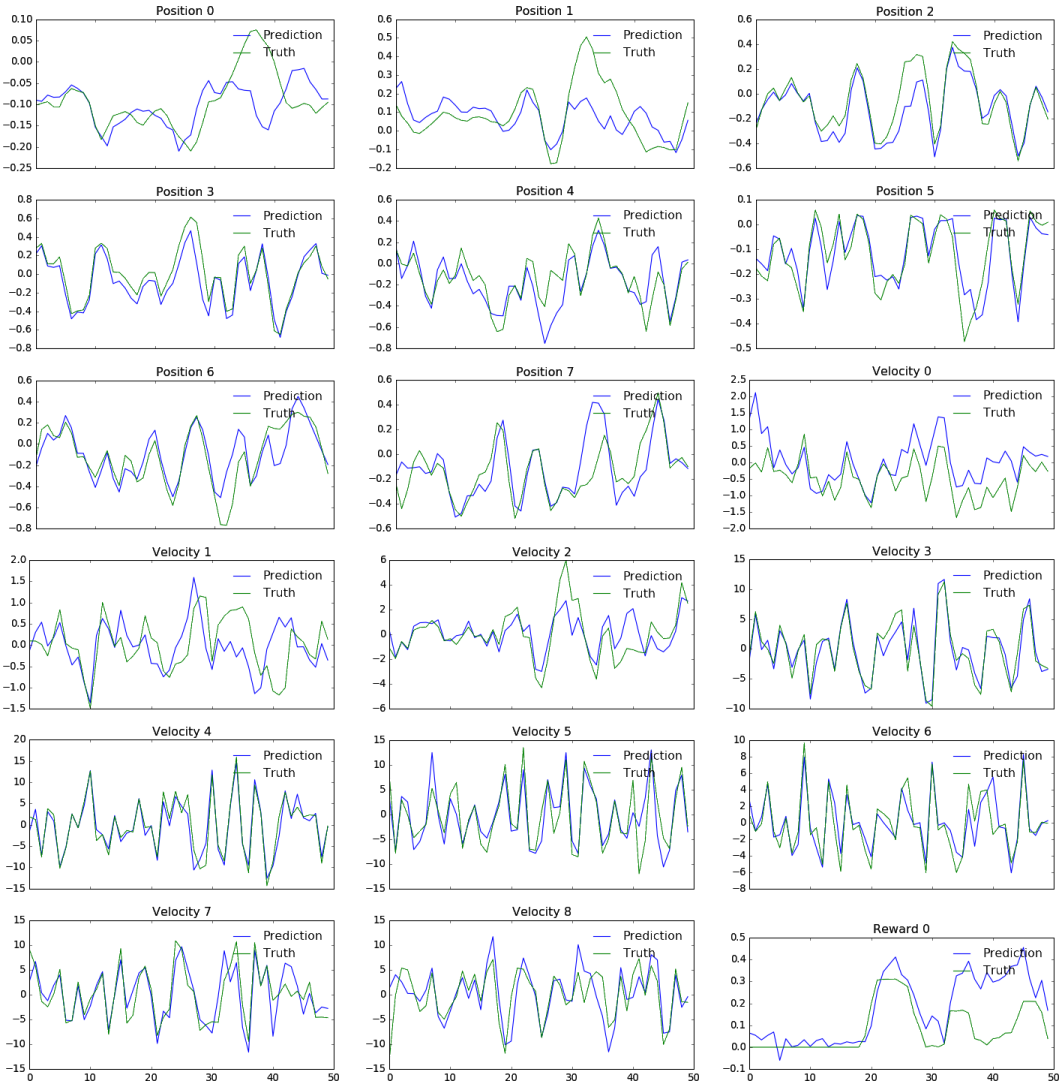


Figure 7: Open-loop state diagnostics. We freeze the dynamics model of a PlaNet agent and learn small neural networks to predict the true positions, velocities, and reward of the simulator. The open-loop predictions of these quantities show that most information about the underlying system is present in the learned latent space and can be accurately predicted forward further than the planning horizons used in this work.

E Planning Parameters

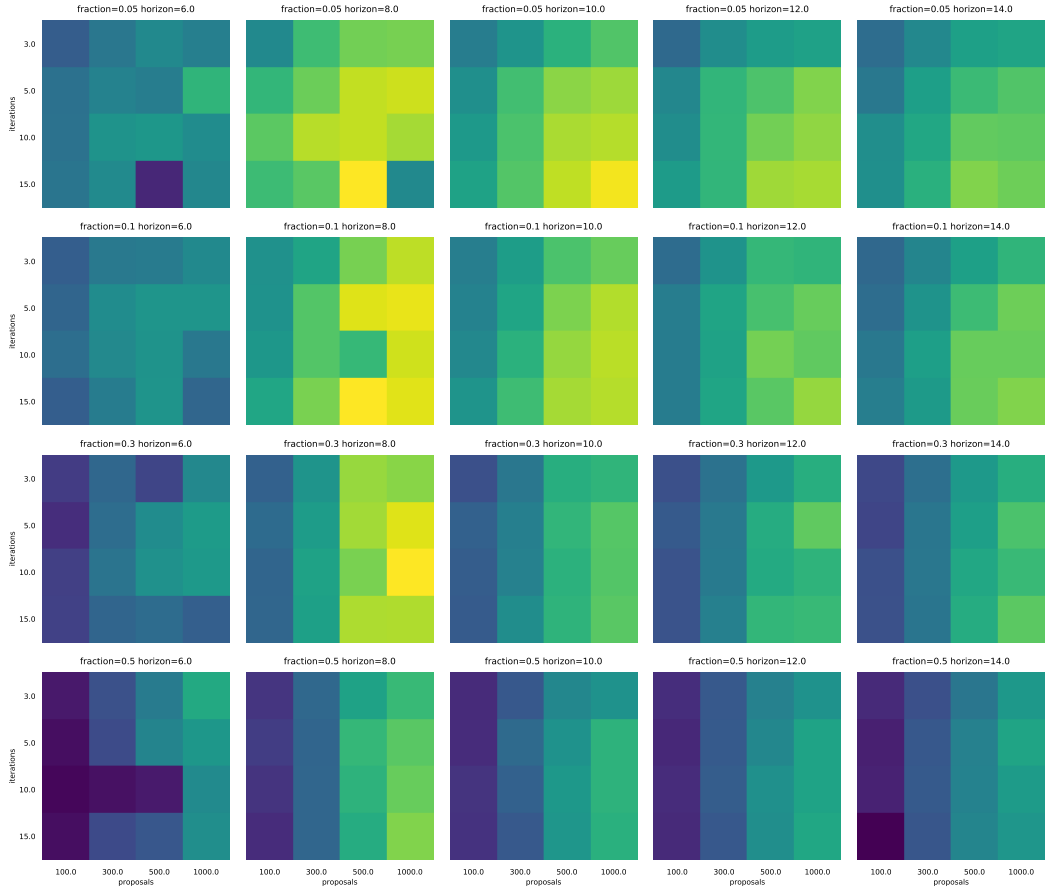


Figure 8: Planning performance on the cheetah running task with the true simulator using different planner settings. Performance ranges from 132 (blue) to 837 (yellow). Evaluating more action sequences, optimizing for more iterations, and re-fitting to fewer of the best proposals tend to improve performance. A planning horizon length of 6 is not sufficient and results in poor performance. Much longer planning horizons hurt performance because of the increased search space. For this environment, best planning horizon length is near 8 steps.