Learning Latent Dynamics for Planning from Pixels

Danijar Hafner [1 2] **Timothy Lillicrap** [3] **Ian Fischer** [4] **Ruben Villegas** [1 5]
David Ha [1] **Honglak Lee** [1 5] **James Davidson** [1]

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 images and chooses actions through fast 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 components and a multi-step variational inference objective that we call latent overshooting. Using only pixel observations, our agent solves continuous control tasks with contact dynamics, partial observability, and sparse rewards, which exceed the difficulty of tasks that were previously solved by planning with learned models. PlaNet uses substantially fewer episodes and reaches final performance close to and sometimes higher than strong 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. Key difficulties include model inaccuracies, accumulating errors of multi-step predictions, failure to capture multiple possible futures, and overconfident predictions outside of the training distribution.

Planning using learned models offers several 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 environment.

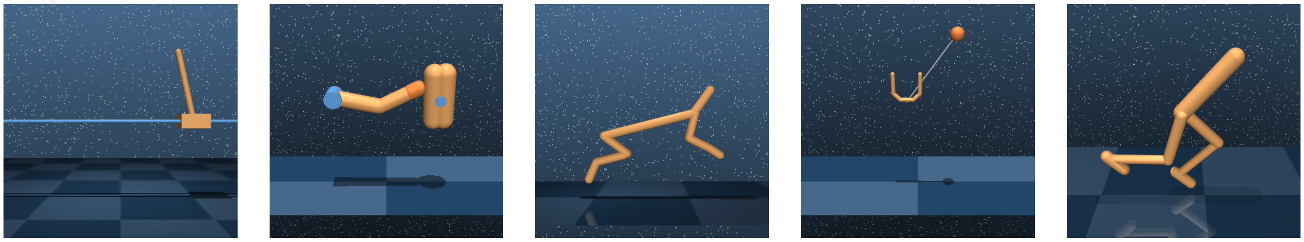
Recent work has shown promise in learning the dynamics of simple low-dimensional environments (Deisenroth & Rasmussen, 2011; Gal et al., 2016; Amos et al., 2018; Chua et al., 2018; Henaff et al., 2018). However, these approaches typically assume access to the underlying state of the world and the reward function, which may not be available in practice. In high-dimensional environments, we would like to learn the dynamics 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 a compact latent space. To 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 continuous control tasks from pixels that are more difficult than those previously solved by planning with learned models.

Key contributions of this work are summarized as follows:

- **Planning in latent spaces** We solve a variety of tasks from the DeepMind control suite, shown in Figure 1, by learning a dynamics model and efficiently planning in its latent space. Our agent substantially outperforms the model-free A3C and in some cases D4PG algorithm in fi-

[1] Google Brain [2] University of Toronto [3] DeepMind [4] Google Research [5] University of Michigan. Correspondence to: Danijar Hafner <mail@danijar.com>.



(a) Cartpole (b) Finger (c) Cheetah (d) Cup (e) Walker

Figure 1: Image-based control domains used in our experiments. The figure shows agent observations before downscaling them to $64 \times 64 \times 3$ pixels. (a) For the cartpole balance and swingup tasks, the camera is fixed so the cart can move out of sight. (b) The finger spinning task includes contacts between the finger and the object. (c) The cheetah running task includes both contacts and a larger number of joints. (d) The cup task has a sparse reward that is only given once the ball is caught. (e) The walker task requires balance and predicting difficult interactions with the ground when the robot is lying down.

nal performance, with on average $50\times$ less environment interaction and similar computation time.

- **Recurrent state space model** We design a latent dynamics model with both deterministic and stochastic components (Buesing et al., 2018; Chung et al., 2015). Our experiments indicate having both components to be crucial for high planning performance.
- **Latent overshooting** We generalize the standard variational bound to include multi-step predictions. Using only 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 model the environment dynamics from experience. PlaNet does so by iteratively collecting data using planning and training the dynamics model on the gathered data. In this section, we introduce notation for the environment and describe the general implementation of our model-based agent. In this section, we assume access to a learned dynamics model. Our design and training objective for this model are detailed in Sections 3 and 4, respectively.

Problem setup 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 follow the stochastic dynamics

$$\begin{aligned}
&\text{Transition function:} && s_t \sim \mathrm{p}(s_t \mid s_{t-1}, a_{t-1}) \\
&\text{Observation function:} && o_t \sim \mathrm{p}(o_t \mid s_t) \\
&\text{Reward function:} && r_t \sim \mathrm{p}(r_t \mid s_t) \\
&\text{Policy:} && a_t \sim \mathrm{p}(a_t \mid o_{\leq t}, a_{<t}),
\end{aligned} \tag{1}$$

where we assume a fixed initial state s_0 without loss of generality. The goal is to implement a policy $\mathrm{p}(a_t \mid o_{\leq t}, a_{<t})$ that maximizes the expected sum of rewards $\mathrm{E}_{\mathrm{p}}\big[\sum_{\tau=t+1}^{T} \mathrm{p}(r_\tau \mid s_\tau)\big]$, where the expectation is over the distributions of the environment and the policy.

Algorithm 1: Deep Planning Network (PlaNet)

```
Input :
$R$ Action repeat          $p(s_t \mid s_{t-1}, a_{t-1})$  Transition model
$S$ Seed episodes          $p(o_t \mid s_t)$               Observation model
$C$ Collect interval       $p(r_t \mid s_t)$               Reward model
$B$ Batch size             $q(s_t \mid o_{\leq t}, a_{<t})$ Encoder
$L$ Chunk length           $p(\epsilon)$                   Exploration noise
$\alpha$ Learning rate

Initialize dataset $\mathcal{D}$ with $S$ random seed episodes.
Initialize model parameters $\theta$ randomly.
while not converged do
    // Model fitting
    for update step $s = 1..C$ do
        Draw sequence chunks $\{(o_t, a_t, r_t)_{t=k}^{L+k}\}_{i=1}^{B} \sim \mathcal{D}$
          uniformly at random from the dataset.
        Compute loss $\mathcal{L}(\theta)$ from Equation 7.
        Update model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$.
    // Data collection
    $o_1 \leftarrow$ env.reset()
    for time step $t = 1..\lceil \frac{T}{R} \rceil$ do
        Infer belief over current state $q(s_t \mid o_{\leq t}, a_{<t})$ from
          the history.
        $a_t \leftarrow$ planner($q(s_t \mid o_{\leq t}, a_{<t}), p$), see
          Algorithm 2 in the appendix for details.
        Add exploration noise $\epsilon \sim p(\epsilon)$ to the action.
        for action repeat $k = 1..R$ do
            $r_t^k, o_{t+1}^k \leftarrow$ env.step($a_t$)
        $r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_{t+1}^R$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$
```

Model-based planning PlaNet learns a transition model $p(s_t \mid s_{t-1}, a_{t-1})$, observation model $p(o_t \mid s_t)$, and reward model $p(r_t \mid s_t)$ from previously experienced episodes (note italic letters for the model compared to upright letters for the true dynamics). The observation model provides a training signal but is not used for planning. We also learn an encoder $q(s_t \mid o_{\leq t}, a_{<t})$ to infer an approximate belief over the current hidden state from the history using filtering. Given these components, we implement the policy as a planning algorithm that searches for the best sequence of future actions. We use model-predictive control (MPC; Richards, 2005) to allow the agent to adapt its plan based on new observations, meaning we replan at each step. In contrast to model-free and hybrid reinforcement learning algorithms, we do not use a policy network.

Experience collection Since the agent may not initially visit all parts of the environment, we need to iteratively collect new experience and refine the dynamics model. We do so by planning with the partially trained model, as shown in Algorithm 1. Starting from a small amount of S seed episodes collected under random actions, we train the model and add one additional episode to the data set every C update steps. When collecting episodes for the data set, we add small Gaussian exploration noise to the action. To reduce the planning horizon and provide a clearer learning signal to the model, we repeat each action R times, as is common in reinforcement learning (Mnih et al., 2015; 2016).

Planning algorithm We use the cross entropy method (CEM; Rubinstein, 1997; Chua et al., 2018) to search for the best action sequence under the model, as outlined in Algorithm 2. We decided on this algorithm because of its robustness and because it solved all considered tasks when given the true dynamics for planning. CEM is a population-based optimization algorithm that infers a distribution over action sequences that maximize the objective. As detailed in Algorithm 2 in the appendix, we initialize a time-dependent diagonal Gaussian belief over optimal 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 current time step, μ_t. Importantly, after receiving the next observation, the belief over action sequences starts from zero mean and unit variance again to avoid local optima.

To evaluate a candidate action sequence under the learned model, we sample a state trajectory starting from the current state belief, and sum the mean rewards predicted along the sequence. Since we use a population-based optimizer, we found it sufficient to consider a single trajectory per action sequence and thus focus the computational budget on evaluating a larger number of different sequences. Because the reward is modeled as a 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. The next section introduces the latent dynamics model that the planner uses.

3. Recurrent State Space Model

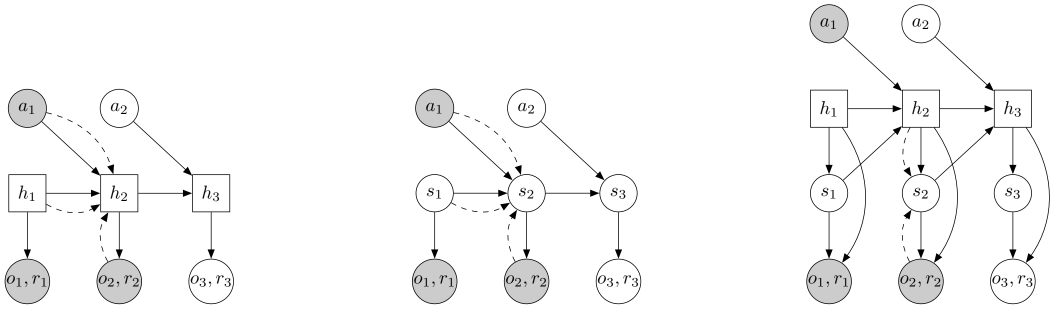
For planning, we need to evaluate thousands of action sequences at every time step of the agent. Therefore, we use a recurrent state-space model (RSSM) that can predict forward purely in latent space, similar to recently proposed models (Karl et al., 2016; Buesing et al., 2018; Doerr et al., 2018). This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

Latent dynamics 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 is shown in Figure 2b and resembles the structure of a partially observable Markov decision process. It defines the generative process of the images and rewards using a hidden state sequence $\{s_t\}_{t=1}^T$,

$$\begin{aligned}
&\text{Transition model:} && s_t \sim p(s_t \mid s_{t-1}, a_{t-1}) \\
&\text{Observation model:} && o_t \sim p(o_t \mid s_t) \\
&\text{Reward model:} && r_t \sim p(r_t \mid s_t),
\end{aligned} \tag{2}$$

where we assume a fixed initial state s_0 without loss of generality. The transition model is Gaussian with mean and variance parameterized by a feed-forward neural network, the observation model is Gaussian with mean parameterized by a deconvolutional neural network and identity covariance, and the reward model is a scalar Gaussian with mean parameterized by a feed-forward neural network and unit variance. Note that the log-likelihood under a Gaussian distribution with unit variance equals the mean squared error up to a constant.

Variational encoder Since the model is non-linear, we cannot directly compute the state posteriors that are needed for parameter learning. Instead, we use an encoder $q(s_{1:T} \mid o_{1:T}, a_{1:T}) = \prod_{t=1}^T q(s_t \mid s_{t-1}, a_{t-1}, o_t)$ to infer approximate state posteriors from past observations and actions, where $q(s_t \mid s_{t-1}, a_{t-1}, o_t)$ is a diagonal Gaussian with mean and variance parameterized by a convolutional neural network followed by a feed-forward neural network. We use the filtering posterior that conditions on past observations



(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 model. (a) Transitions in a recurrent neural network are purely deterministic. This prevents the model from capturing multiple futures and makes it easy for the planner to exploit inaccuracies. (b) Transitions in a state-space model are purely stochastic. This makes it difficult to remember information over multiple time steps. (c) We split the state into stochastic and deterministic parts, allowing the model to robustly learn to predict multiple futures.

since we are ultimately interested in using the model for planning, but one may also use the full smoothing posterior during training (Babaeizadeh et al., 2017).

Training objective Using the encoder, we construct a variational bound on the data log-likelihood. For simplicity, we write losses for predicting only the observations — the reward losses follow by analogy. The variational bound obtained using Jensen's inequality is

$$\begin{aligned}\ln p(o_{1:T} \mid a_{1:T}) &\triangleq \ln \int \prod_t p(s_t \mid s_{t-1}, a_{t-1})\, p(o_t \mid s_t)\, ds_{1:T} \\ &\geq \sum_{t=1}^{T} \Bigg(\underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t}, a_{<t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} \hookleftarrow \\ &\quad \underbrace{-\, \mathop{\mathrm{E}}_{q(s_{t-1} \mid o_{\leq t-1}, a_{<t-1})} \big[\mathrm{KL}[q(s_t \mid o_{\leq t}, a_{<t}) \parallel p(s_t \mid s_{t-1}, a_{t-1})]\big]}_{\text{complexity}} \Bigg).\end{aligned} \tag{3}$$

For the derivation, please see Equation 8 in the appendix. Estimating the outer expectations using a single reparameterized sample yields an efficient objective for inference and learning in non-linear latent variable models that can be optimized using gradient ascent (Kingma & Welling, 2013; Rezende et al., 2014; Krishnan et al., 2017).

Deterministic path Despite its generality, the purely stochastic transitions make it difficult for the transition model to reliably remember information for multiple time steps. In theory, this model could learn to set the variance to zero for some state components, but the optimization procedure may not find this solution. This motivates including a deterministic sequence of activation vectors $\{h_t\}_{t=1}^T$ that allow the model to access not just the last state but all previous states deterministically (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),

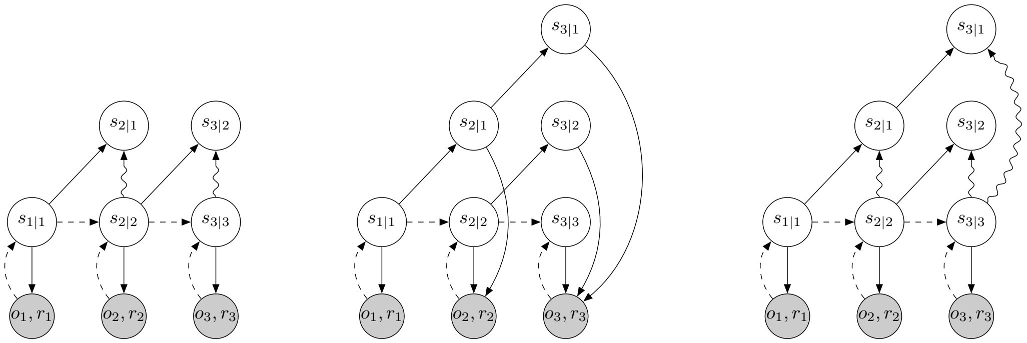
$$\begin{aligned} &\text{Deterministic state model:} && h_t = f(h_{t-1}, s_{t-1}, a_{t-1}) \\ &\text{Stochastic state model:} && s_t \sim p(s_t \mid h_t) \\ &\text{Observation model:} && o_t \sim p(o_t \mid h_t, s_t) \\ &\text{Reward model:} && r_t \sim p(r_t \mid h_t, s_t), \end{aligned} \tag{4}$$

where $f(h_{t-1}, s_{t-1}, a_{t-1})$ is implemented as a recurrent neural network (RNN). Intuitively, we can understand this model as splitting the state into a stochastic part s_t and a deterministic part h_t, which depend on the stochastic and deterministic parts at the previous time step through the RNN. We use the encoder $q(s_{1:T} \mid o_{1:T}, a_{1:T}) = \prod_{t=1}^T q(s_t \mid h_t, o_t)$ to parameterize the approximate state posteriors. Importantly, all information about the observations must pass through the sampling step of the encoder to avoid a deterministic shortcut from inputs to reconstructions.

Global prior 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 alleviates overfitting to the initially small training data set and grounds the state beliefs (since posteriors and temporal priors are both learned, they could drift in latent space). The global prior adds additional KL-divergence loss terms from each posterior to a standard Gaussian. 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 objective for latent sequence models and propose a generalization of it that improves long-term predictions.

4. Latent Overshooting

In the previous section, we derived the typical variational bound for learning and inference in latent sequence models



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

Figure 3: Unrolling schemes. The labels $s_{i|j}$ are short for the state at time i conditioned on observations up to time j. Arrows pointing at shaded circles indicate log-likelihood loss terms. Wavy arrows 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 (Amos et al., 2018) decodes all multi-step predictions to apply additional reconstruction losses. This is typically too expensive in image domains. (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.

(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: the gradient flows through $p(s_t \mid s_{t-1}, a_{t-1})$ directly into $q(s_{t-1})$ but never traverses a chain of multiple $p(s_t \mid s_{t-1}, a_{t-1})$. In this section, we generalize this variational bound to *latent overshooting*, which trains all multi-step predictions in latent space.

Limited capacity 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 the model on multi-step predictions of all distances. We develop this idea for latent sequence models, showing that multi-step predictions can be improved by a loss in latent space, without having to generate additional images.

Multi-step prediction We start by generalizing the standard variational bound (Equation 3) from training one-step predictions to training multi-step predictions of a fixed distance d. For ease of notation, we omit actions in the conditioning set here; every distribution over s_t is conditioned upon $a_{<t}$. We first define multi-step predictions, which are computed by repeatedly applying the transition model and integrating out the intermediate states,

$$\begin{aligned} p(s_t \mid s_{t-d}) &\triangleq \int \prod_{\tau=t-d+1}^{t} p(s_\tau \mid s_{\tau-1})\, ds_{t-d+1:t-1} \\ &= \mathrm{E}_{p(s_{t-1} \mid s_{t-d})}[p(s_t \mid s_{t-1})]. \end{aligned} \tag{5}$$

The case $d = 1$ recovers the one-step transitions used in the original model. Given this definition of a multi-step prediction, we generalize Equation 3 to the variational bound on the multi-step predictive distribution p_d,

$$\begin{aligned} &\ln p_d(o_{1:T}) \triangleq \ln \int \prod_{t=1}^{T} p(s_t \mid s_{t-d}) p(o_t \mid s_t)\, ds_{1:T} \\ &\geq \sum_{t=1}^{T} \Big(\underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} \hookleftarrow \\ &\quad \underbrace{- \mathrm{E}_{p(s_{t-1} \mid s_{t-d}) q(s_{t-d} \mid o_{\leq t-d})} \big[\mathrm{KL}[q(s_t \mid o_{\leq t}) \parallel p(s_t \mid s_{t-1})]\big]}_{\text{multi-step prediction}} \Big). \end{aligned} \tag{6}$$

For the derivation, please see Equation 9 in the appendix. Maximizing this objective trains the multi-step predictive distribution. This reflects the fact that during planning, the model makes predictions without having access to all the preceding observations.

We conjecture that Equation 6 is also a lower bound on $\ln p(o_{1:T})$ based on the data processing inequality. Since the latent state sequence is Markovian, for $d \geq 1$ we have $\mathrm{I}(s_t; s_{t-d}) \leq \mathrm{I}(s_t; s_{t-1})$ and thus $\mathrm{E}[\ln p_d(o_{1:T})] \leq \mathrm{E}[\ln p(o_{1:T})]$. Hence, every bound on the multi-step predictive distribution is also a bound on the one-step predictive distribution in expectation over the data set. For details, please see Equation 10 in the appendix. In the next paragraph, we alleviate the limitation that a particular p_d only trains predictions of one distance and arrive at our final objective.

Latent overshooting We introduced a bound on predictions of a given distance d. However, for planning we need accurate predictions not just for a fixed distance but for all distances up to the planning horizon. We introduce latent overshooting for this, an objective function for latent sequence models that generalizes the standard variational bound (Equation 3) to train the model on multi-step predictions of all distances $1 \leq d \leq D$,

$$\frac{1}{D}\sum_{d=1}^{D} \ln p_d(o_{1:T}) \geq \sum_{t=1}^{T} \Bigg(\underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} \hookleftarrow \\ \underbrace{- \frac{1}{D}\sum_{d=1}^{D} \beta_d \, \mathop{\mathrm{E}}_{p(s_{t-1} \mid s_{t-d}) q(s_{t-d} \mid o_{\leq t-d})} \big[\mathrm{KL}[q(s_t \mid o_{\leq t}) \parallel p(s_t \mid s_{t-1})] \big]}_{\text{latent overshooting}} \Bigg). \quad (7)$$

Latent overshooting can be interpreted as a regularizer in latent space that encourages consistency between one-step and multi-step predictions, which we know should be equivalent in expectation over the data set. We include weighting factors $\{\beta_d\}_{d=1}^{D}$ analogously to the β-VAE (Higgins et al., 2016). While we set all $\beta_{>1}$ to the same value for simplicity, they could be chosen to let the model focus more on long-term or short-term predictions. In practice, we stop gradients of the posterior distributions for overshooting distances $d > 1$, so that the multi-step predictions are trained towards the informed posteriors, but not the other way around. Equation 7 is the final objective function that we use to train the dynamics model of our agent.

5. Experiments

We evaluate PlaNet on six continuous control tasks from pixels. We explore multiple design axes of the agent: the stochastic and deterministic paths in the dynamics model, the latent overshooting objective, and online experience collection. We refer to the appendix for hyper parameters. Besides the action repeat, we use the same hyper parameters for all tasks. Within one fiftieth the 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 1 day on a single Nvidia V100 GPU is comparable to that of D4PG. Our implementation uses TensorFlow Probability (Dillon et al., 2017). Please visit `https://danijar.com/planet` for access to the code and videos of the trained agent.

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, the cup task only has a sparse reward for when the ball is caught, and the walker is challenging because the robot first has to stand up and then walk, resulting in collisions with the ground that are difficult to predict. In all tasks, the only observations are third-person camera images of size $64 \times 64 \times 3$ pixels.

Comparison to model-free methods Figure 4 compares the performance of PlaNet to the model-free algorithms reported by Tassa et al. (2018). Within 500 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.

Model designs Figure 4 additionally compares design choices of the dynamics model. We train PlaNet using our recurrent state-space model (RSSM), as well as versions with purely deterministic GRU (Cho et al., 2014), 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 the tasks are stochastic from the agent's perspective due to partial observability of the initial states. The noise might also add a safety margin to the planning objective that results in more robust action sequences. The deterministic part allows the model to remember information over many time steps and is even more important — the agent does not learn without it.

Agent designs Figure 5 compares PlaNet with latent overshooting to versions with standard variational objective, and with a fixed random data set rather than collecting experience online. We observe that online data collection helps all tasks and is necessary for the finger and walker tasks. Latent overshooting is necessary for successful planning on the walker and cup tasks; the sparse reward in the cup task demands accurate predictions for many time steps. It also slows down initial learning for the finger task, but increases final performance on the cartpole balance and cheetah tasks.

One agent all tasks Figure 7 in the appendix shows the performance of a single agent trained on all six tasks. The agent is not told which task it is facing; it needs to infer this from the image observations. We pad the action spaces with unused elements to make them compatible and adapt Algorithm 1 to collect one episode of each task every $6\,C$ update steps. We use the same hyper parameters as for the main experiments above. The agent reaches the same average performance over tasks as individually trained agents. While learning is slowed down for the cup task and the easier cart-

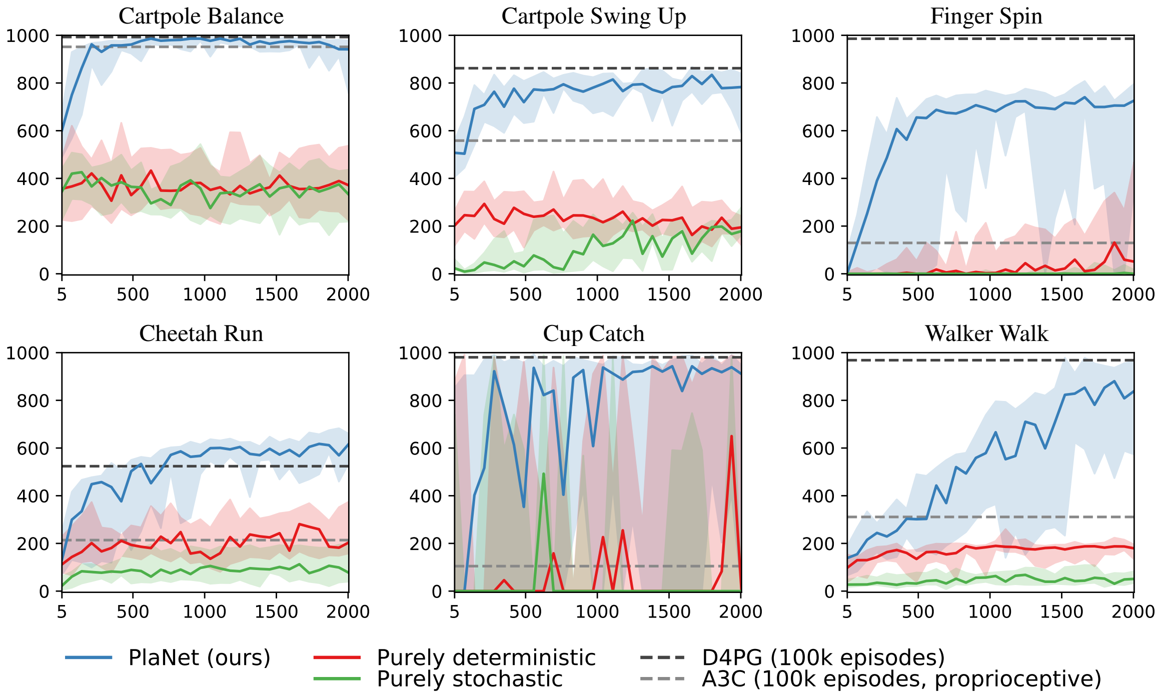


Figure 4: Comparison of PlaNet to model-free algorithms and other model designs. Plots show test performance for the number of collected episodes. We compare PlaNet using our RSSM (Section 3) to purely deterministic (RNN) and purely stochastic models (SSM). The RNN does not use latent overshooting, as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

pole tasks, it is substantially improved for the difficult task of walker. This indicates that positive transfer between these tasks might be possible using model-based reinforcement learning, regardless of the conceptually different visuals.

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 & Boedecker, 2017; Nagabandi et al., 2017; Weber et al., 2017; Kurutach et al., 2018; Buckman et al., 2018; Ha & 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 & Fergus, 2018; Buesing et al., 2018; Doerr et al., 2018; Gregor & Besse, 2018). Appendix E 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 & Levine, 2017; Ebert et al., 2018) that deal with the visual complexity of the real world and solve tasks with a simple gripper, such as grasping or pushing objects. In comparison, we focus on simulated environments, where we leverage latent planning to scale to larger state and action spaces, longer planning horizons, as well as sparse reward tasks. 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 present PlaNet, a model-based agent that learns a latent dynamics model from image observations and chooses actions by fast planning in latent space. To enable accurate long-term predictions, we design 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 $50\times$

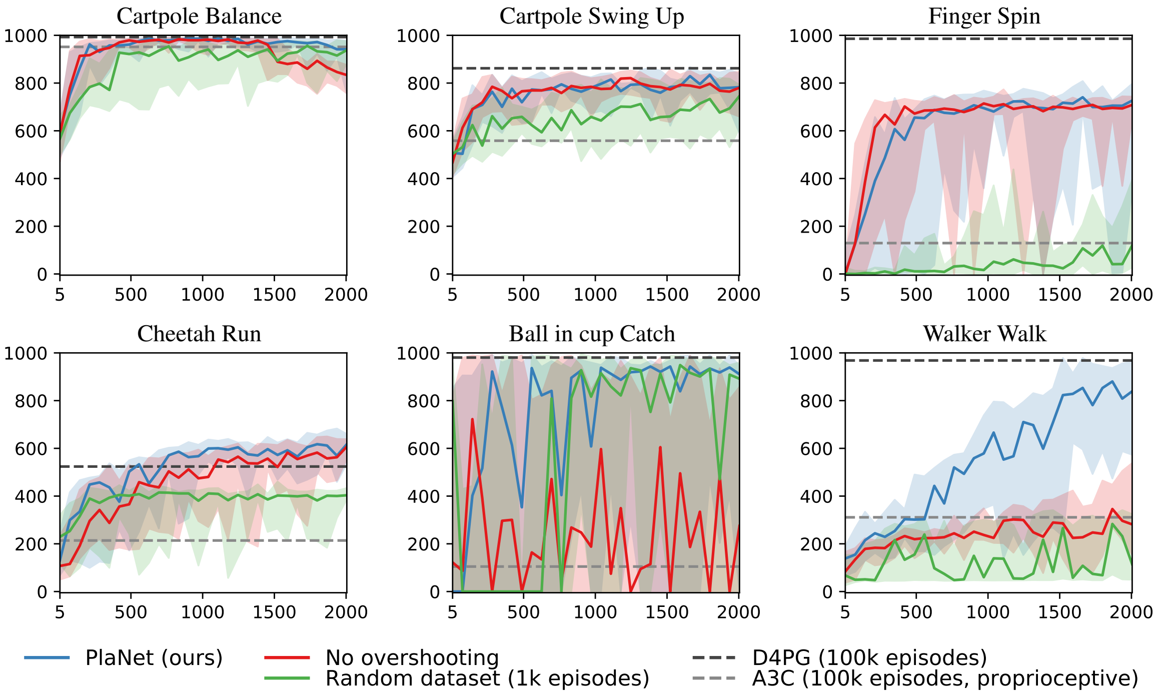


Figure 5: Comparison of agent designs. Plots show test performance for the number of collected episodes. We compare PlaNet using latent overshooting (Equation 7), a version with standard variational objective (Equation 3), and a version that trains from a random data set of 1000 episodes rather than collecting experience during training. The lines show medians and the areas show percentiles 5 to 95 over 4 seeds and 10 rollouts.

Table 1: Comparison of PlaNet to the model-free algorithms A3C and D4PG reported by Tassa et al. (2018). The training curves for these are shown as orange lines in Figure 4 and as solid green lines in Figure 6 in their paper. From these, we estimate the number of episodes that D4PG takes to achieve the final performance of PlaNet to estimate the data efficiency gain. We further 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. Numbers indicate mean final performance over 4 seeds.

Method	**Modality**	**Episodes**	**Cartpole Balance**	**Cartpole Swingup**	**Finger Spin**	**Cheetah Run**	**Ball in cup Catch**	**Walker Walk**
A3C	proprioceptive	100,000	952	558	129	214	105	311
D4PG	pixels	100,000	993	862	985	524	980	968
PlaNet (ours)	pixels	2,000	986	831	744	650	914	890
CEM + true simulator	simulator state	0	998	850	825	656	993	994
Data efficiency gain PlaNet over D4PG (factor)			100	180	16	50+	20	11

fewer episodes and similar training time. The results show that learning latent dynamics models for planning in image domains is a promising approach.

Directions for future work include learning temporal abstraction instead of using a fixed action repeat, possibly through hierarchical models. To further improve final performance, one could learn a value function to approximate the sum of rewards beyond the planning horizon. Moreover, exploring gradient-based planners could increase computational efficiency of the agent. Our work provides a starting point for multi-task control by sharing the dynamics model.

Acknowledgements We thank Jacob Buckman, Nicolas Heess, John Schulman, Rishabh Agarwal, Silviu Pitis, Mohammad Norouzi, George Tucker, David Duvenaud, Shane Gu, Chelsea Finn, Steven Bohez, Jimmy Ba, Stephanie Chan, and Jenny Liu for helpful discussions.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Krishnan, R. G., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. In *AAAI*, pp. 2101–2109, 2017.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A. Hyper Parameters

We use the convolutional and deconvolutional networks from Ha & Schmidhuber (2018), a GRU (Cho et al., 2014) with 200 units as RNN in the dynamics model, and implement all other functions as two fully connected layers of size 200 with ReLU activations (Nair & Hinton, 2010). Distributions in latent space are 30-dimensional diagional Gaussians with predicted mean and standard deviation.

We pre-process images by reducing the bit depth to 5 bits as in Kingma & Dhariwal (2018). The model is trained using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 10^{-3}, $\epsilon = 10^{-4}$, and gradient clipping norm of 1000 on batches of $B = 50$ sequence chunks of length $L = 50$. For latent overshooting, we use $D = 50$ and set $\beta_1 = D$ and $\beta_{>1} = 1$. We allow 2 free nats to the model, meaning we clip the divergence loss below this value. 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 $C = 100$ update steps under $\epsilon \sim \mathrm{Normal}(0, 0.3)$ action noise. The action repeat differs between domains: cartpole ($R = 8$), finger ($R = 2$), cheetah ($R = 4$), cup ($R = 6$), walker ($R = 2$). We find important hyper parameters to be the action repeat, the KL-divergence scales β, and the learning rate.

B. Planning Algorithm

Algorithm 2: Latent planning with CEM

```
Input : $H$ Planning horizon distance          $q(s_t \mid o_{\leq t}, a_{<t})$        Current state belief
        $I$ Optimization iterations            $p(s_t \mid s_{t-1}, a_{t-1})$    Transition model
        $J$ Candidates per iteration           $p(r_t \mid s_t)$                 Reward model
        $K$ Number of top candidates to fit

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
        $a^{(j)}_{t:t+H} \sim q(a_{t:t+H})$
        $s^{(j)}_{t:t+H+1} \sim q(s_t \mid o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p(s_\tau \mid s_{\tau-1}, a^{(j)}_{\tau-1})$
        $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} \mathrm{E}[p(r_\tau \mid s^{(j)}_\tau)]$
    // 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 $\mu_t$.
```

C. Multi-Task Learning

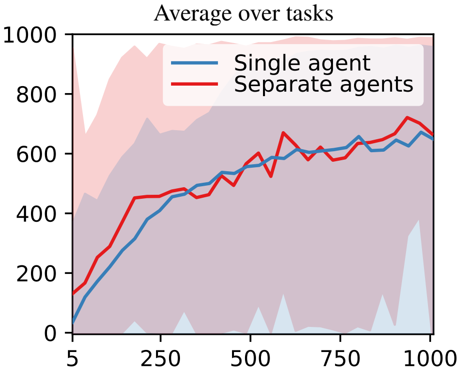


Figure 6: We compare a single PlaNet agent trained on all tasks to individual PlaNet agents. The plot shows test performance for the number of episodes collected for each task. The single agent achieves the same performance as the individual agents. The hyper parameters are the same as for the main experiments in the paper. The multi-task agent collects one episode of each task every $C = 6000$ to match the rate at which the single-task agent collects experience. The lines show medians and the areas show percentiles 5 to 95 over 6 tasks, 4 seeds, and 10 rollouts.

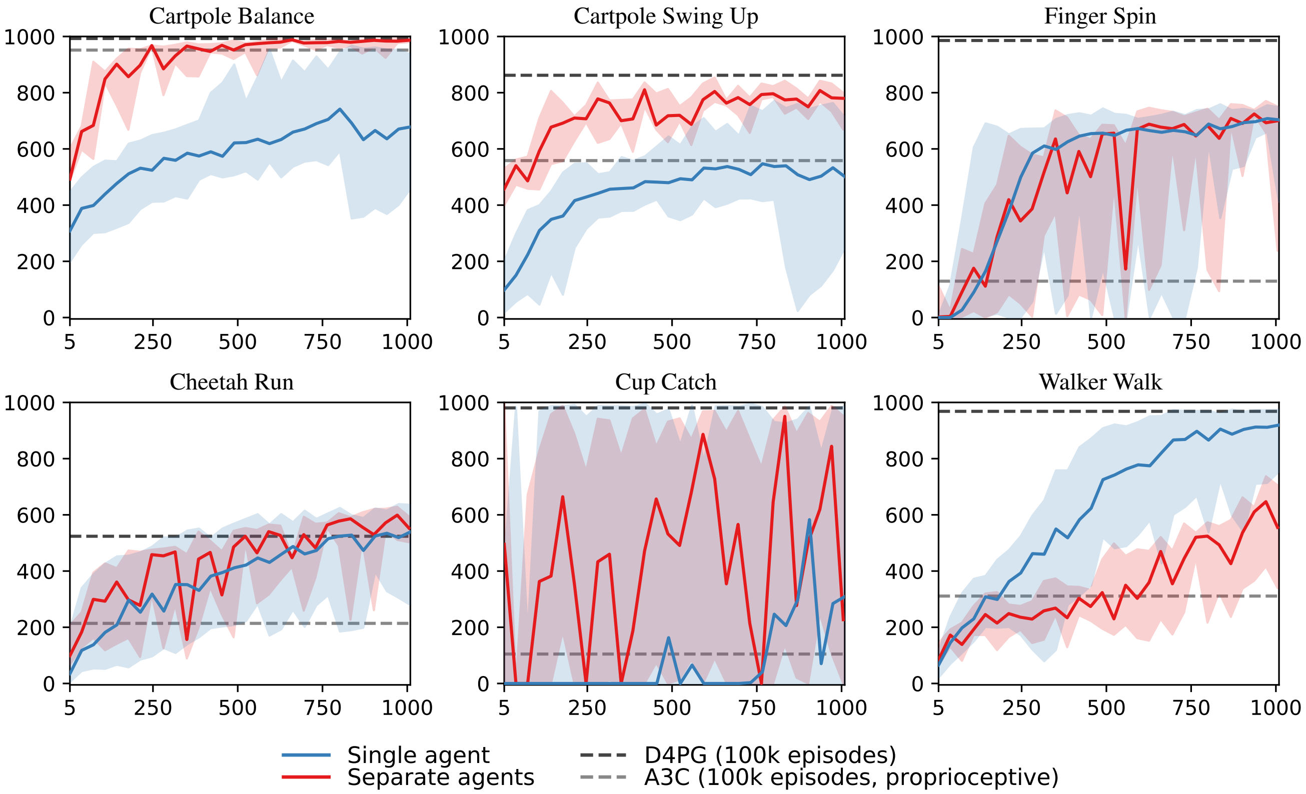


Figure 7: Per-task performance of a single PlaNet agent trained on the six tasks. Plots show test performance for the number of episodes collected per task. The agent is not told which task it is solving and it needs to infer this from the image observations. While learning is slowed down for the easier cartpole tasks, it is substantially improved for the difficult task of walker. These experiments suggest suggest the potential of planning with learned models for building agents that are more generally applicable.

D. Bound Derivations

One-step predictive distribution The variational bound for latent dynamics models $p(o_{1:T}, s_{1:T} \mid a_{1:T}) = \prod_t p(s_t \mid s_{t-1}, a_{t-1}) p(o_t \mid s_t)$ and a variational posterior $q(s_{1:T} \mid o_{1:T}, a_{1:T}) = \prod_t q(s_t \mid o_{\leq t}, a_{<t})$ follows from importance weighting and Jensen's inequality as shown,

$$\begin{aligned}
\ln p(o_{1:T} \mid a_{1:T}) &\triangleq \ln \mathrm{E}_{p(s_{1:T}|a_{1:T})}\left[\prod_{t=1}^{T} p(o_t \mid s_t)\right] \\
&= \ln \mathrm{E}_{q(s_{1:T}|o_{1:T},a_{1:T})}\left[\prod_{t=1}^{T} p(o_t \mid s_t) p(s_t \mid s_{t-1}, a_{t-1}) / q(s_t \mid o_{\leq t}, a_{<t})\right] \\
&\geq \mathrm{E}_{q(s_{1:T}|o_{1:T},a_{1:T})}\left[\sum_{t=1}^{T} \ln p(o_t \mid s_t) + \ln p(s_t \mid s_{t-1}, a_{t-1}) - \ln q(s_t \mid o_{\leq t}, a_{<t})\right] \\
&= \sum_{t=1}^{T} \Big(\underbrace{\underset{q(s_t|o_{\leq t},a_{<t})}{\mathrm{E}}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} - \underbrace{\underset{q(s_{t-1|o_{\leq t-1},a_{<a-1}})}{\mathrm{E}}\big[\mathrm{KL}[q(s_t \mid o_{\leq t}, a_{<t}) \parallel p(s_t \mid s_{t-1}, a_{t-1})]\big]}_{\text{complexity}} \Big).
\end{aligned} \tag{8}$$

Multi-step predictive distribution The variational bound on the d-step predictive distribution $p_d(o_{1:T}, s_{1:T} \mid a_{1:T}) = \prod_t p(s_t \mid s_{t-d}, a_{t-1}) p(o_t \mid s_t)$ and a variational posterior $q(s_{1:T} \mid o_{1:T}, a_{1:T}) = \prod_t q(s_t \mid o_{\leq t}, a_{<t})$ follows analogously. The second bound comes from moving the log inside the multi-step priors, which satisfy the recursion $p(s_t \mid s_{t-d}, a_{t-d-1:t-1}) = \mathrm{E}_{p(s_{t-1}|s_{t-d},a_{t-d-1:t-2})}[p(s_t \mid s_{t-1}, a_{t-1})]$.

$$\begin{aligned}
\ln p_d(o_{1:T} \mid a_{1:T}) &\triangleq \ln \mathrm{E}_{p_d(s_{1:T}|a_{1:T})}\left[\prod_{t=1}^{T} p(o_t \mid s_t)\right] \\
&= \ln \mathrm{E}_{q(s_{1:T}|o_{1:T},a_{1:T})}\left[\prod_{t=1}^{T} p(o_t \mid s_t) p(s_t \mid s_{t-d}, a_{t-d-1:t-1}) / q(s_t \mid o_{\leq t}, a_{<t})\right] \\
&\geq \mathrm{E}_{q(s_{1:T}|o_{1:T},a_{1:T})}\left[\sum_{t=1}^{T} \ln p(o_t \mid s_t) + \ln p(s_t \mid s_{t-d}, a_{t-d-1:t-1}) - \ln q(s_t \mid o_{\leq t}, a_{<t})\right] \\
&\geq \mathrm{E}_{q(s_{1:T}|o_{1:T},a_{1:T})}\left[\sum_{t=1}^{T} \ln p(o_t \mid s_t) + \underset{p(s_{t-1}|s_{t-d},a_{t-d-1:t-2})}{\mathrm{E}}[\ln p(s_t \mid s_{t-1}, a_{t-1})] - \ln q(s_t \mid o_{\leq t}, a_{<t})\right] \\
&= \sum_{t=1}^{T} \Big(\underbrace{\mathrm{E}_{q(s_t|o_{\leq t},a_{<t})}[\ln p(o_t \mid s_t)]}_{\text{reconstruction}} - \underbrace{\underset{p(s_{t-1}|s_{t-d},a_{t-d-1:t-2})q(s_{t-d}|o_{\leq t-d},a_{<t-d})}{\mathrm{E}}\big[\mathrm{KL}[q(s_t \mid o_{\leq t}, a_{<t}) \parallel p(s_t \mid s_{t-1}, a_{t-1})]\big]}_{\text{multi-step prediction}} \Big).
\end{aligned} \tag{9}$$

Since all expectations are on the outside of the objective, we can easily obtain an unbiased estimator of this bound by changing expectations to sample averages.

Relation between one-step and multi-step predictive distributions We conjecture that the multi-step predictive distribution $p_d(o_{1:T})$ lower bounds the one-step predictive distribution $p(o_{1:T})$ of the same latent sequence model model in expectation over the data set. Since the latent state sequence is Markovian, for $d \geq 1$ we have the data processing inequality

$$\begin{aligned}
\mathrm{I}(s_t; s_{t-d}) &\leq \mathrm{I}(s_t; s_{t-1}) \\
\mathrm{H}(s_t) - \mathrm{H}(s_t|s_{t-d}) &\leq \mathrm{H}(s_t) - \mathrm{H}(s_t|s_{t-1}) \\
\mathrm{E}[\ln p(s_t \mid s_{t-d})] &\leq \mathrm{E}[\ln p(s_t \mid s_{t-1})] \\
\mathrm{E}[\ln p_d(o_{1:T})] &\leq \mathrm{E}[\ln p(o_{1:T})].
\end{aligned} \tag{10}$$

Therefore, any bound on the multi-step predictive distribution, including Equation 9 and Equation 7, is also a bound on the one-step predictive distribution.

E. 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 & 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 model using overshooting in observation space for active exploration with a robotics hand. We move beyond low-dimensional state 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 & Boedecker, 2017; Nagabandi et al., 2017; Kurutach et al., 2018; Buckman et al., 2018; Ha & 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 & 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 & 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.

F. Video Predictions

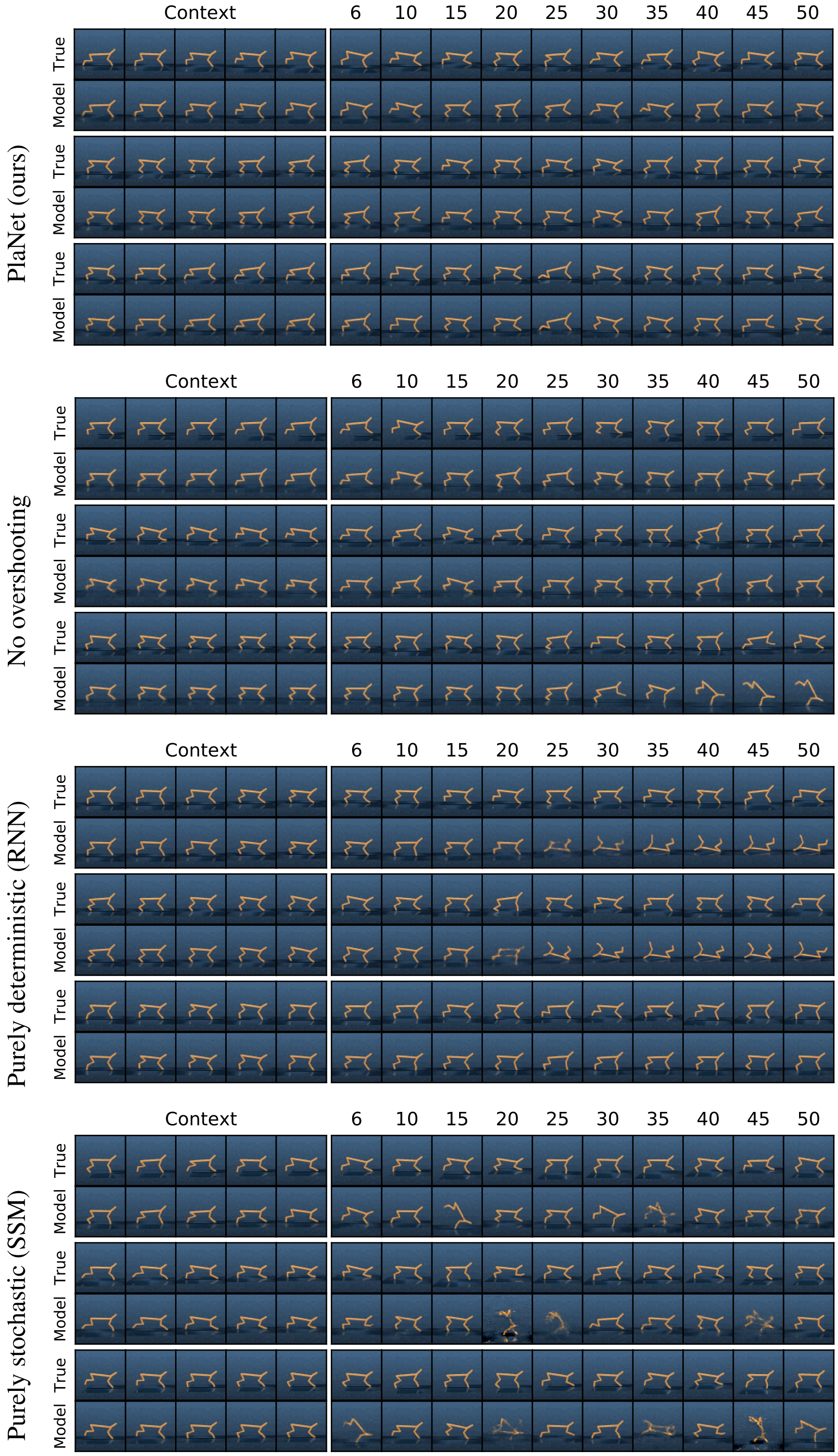


Figure 8: Open-loop video predictions for test episodes. The columns 1–5 show reconstructed context frames and the remaining images are generated open-loop. Latent overshooting enables pixel-accurate predictions for 50 steps into the future in the cheetah environment. We randomly selected action sequences from test episodes collected with action noise alongside the training episodes.

G. State Diagnostics

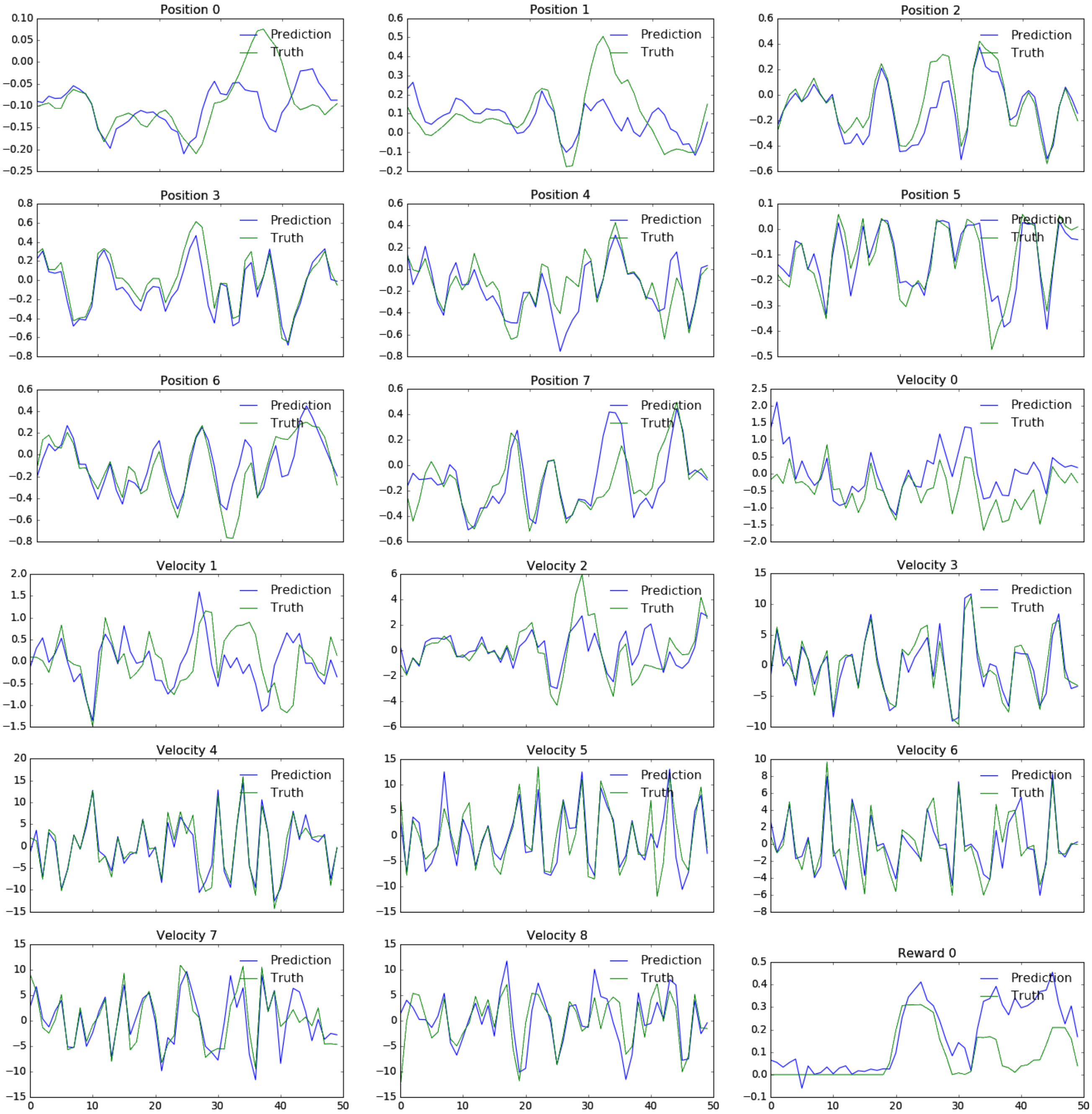


Figure 9: Open-loop state diagnostics. We freeze the dynamics model of a PlaNet agent and learn small neural networks to predict the true positions, velocities, and reward of the simulator. The open-loop predictions of these quantities show that most information about the underlying system is present in the learned latent space and can be accurately predicted forward further than the planning horizons used in this work.

H. Planning Parameters

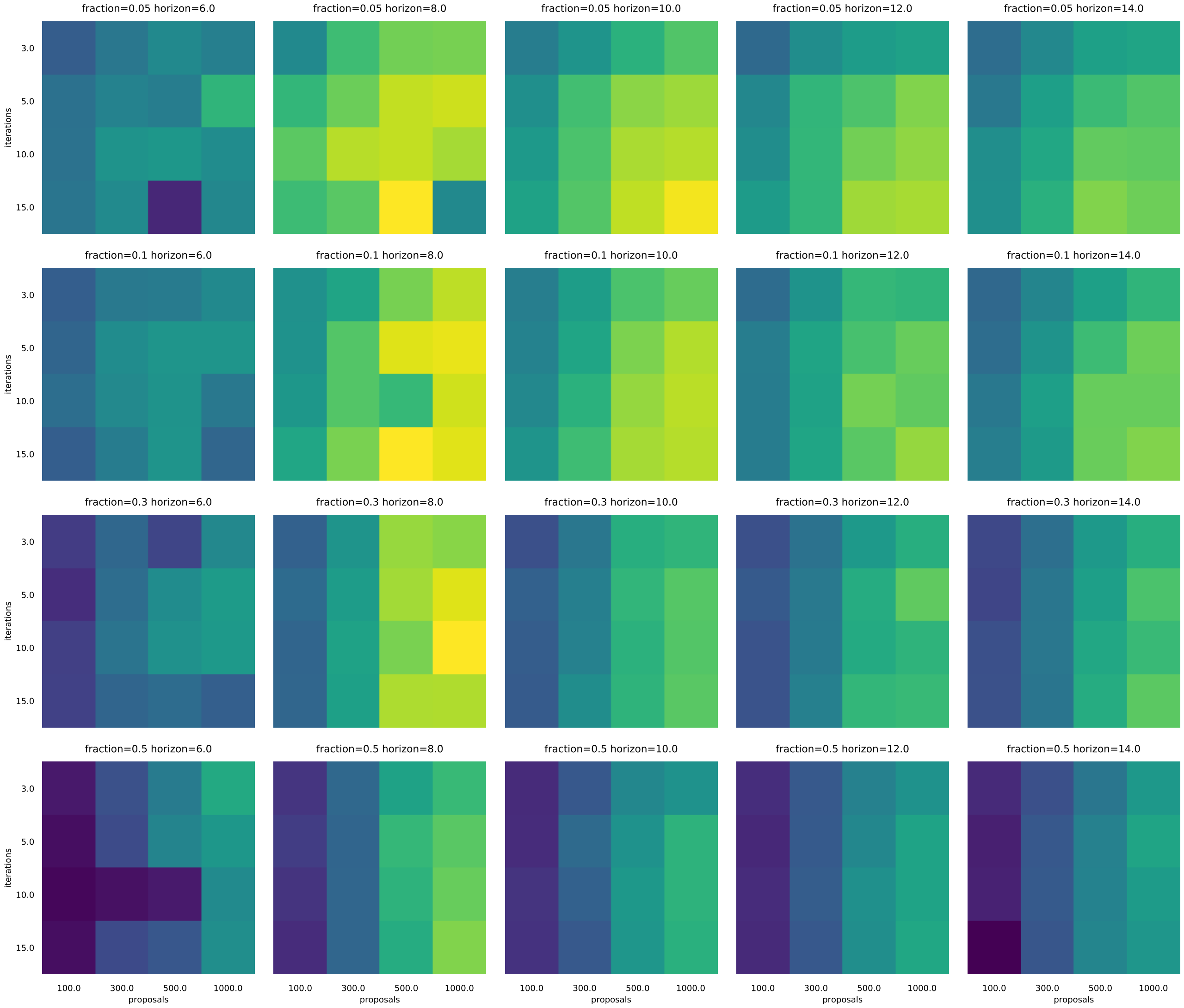


Figure 10: Planning performance on the cheetah running task with the true simulator using different planner settings. Performance ranges from 132 (blue) to 837 (yellow). Evaluating more action sequences, optimizing for more iterations, and re-fitting to fewer of the best proposals tend to improve performance. 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.