

---

# Learning Latent Dynamics for Planning from Pixels

---

Danijar Hafner<sup>1,2</sup> Timothy Lillicrap<sup>3</sup> Ian Fischer<sup>4</sup> Ruben Villegas<sup>1,5</sup>  
 David Ha<sup>1</sup> Honglak Lee<sup>1</sup> James Davidson<sup>1</sup>

## 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 using a latent dynamics model with both deterministic and stochastic transition components. Moreover, we propose a multi-step 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, 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 previously 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. Moreover, we experiment with a novel 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

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

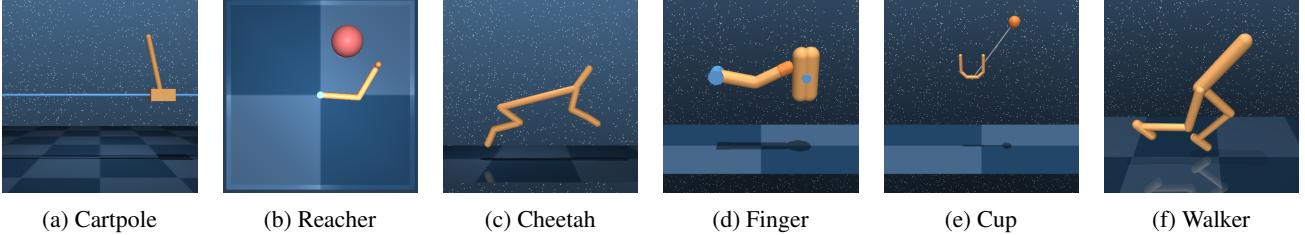


Figure 1: Image-based control domains used in our experiments. The images show agent observations before downscaling to  $64 \times 64 \times 3$  pixels. (a) The cartpole swingup task has a fixed camera so the cart can move out of sight. (b) The reacher task has only a sparse reward. (c) The cheetah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

its latent space. Our agent substantially outperforms the model-free A3C and in some cases D4PG algorithm in final performance, with on average  $200\times$  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 regularizer that can improve 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 Section 3.

**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

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

---

**Algorithm 1:** Deep Planning Network (PlaNet)
 

---

**Input :**

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

```

1 Initialize dataset  $\mathcal{D}$  with  $S$  random seed episodes.
2 Initialize model parameters  $\theta$  randomly.
3 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 3.
        Update model parameters  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$ .
    // Data collection
     $o_1 \leftarrow \text{env.reset}()$ 
    for time step  $t = 1..\lceil \frac{T}{R} \rceil$  do
        Infer belief over current state  $q(s_t | o_{\leq t}, a_{<t})$  from
        the history.
         $a_t \leftarrow \text{planner}(q(s_t | o_{\leq t}, a_{<t}), p)$ , 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
             $r_t^k, o_{t+1}^k \leftarrow \text{env.step}(a_t)$ 
             $r_t, o_{t+1} \leftarrow \sum_{k=1}^R r_t^k, o_{t+1}^k$ 
     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^T\}$ 

```

---

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

**Model-based planning** PlaNet learns a transition model  $p(s_t | s_{t-1}, a_{t-1})$ , observation model  $p(o_t | s_t)$ , and reward model  $p(r_t | s_t)$  from previously experienced episodes (note italic letters for the model compared to upright letters for the true dynamics). The observation model provides a rich training signal but is not used for planning. We also learn an encoder  $q(s_t | o_{\leq t}, a_{<t})$  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 or value 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 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  $a_{t:t+H} \sim \text{Normal}(\mu_{t:t+H}, \sigma_{t:t+H}^2 \mathbb{I})$ , 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,  $\mu_t$ . 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  $t$ , 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  $\{s_t\}_{t=1}^T$ ,

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

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  $q(s_{1:T} | o_{1:T}, a_{1:T}) = \prod_{t=1}^T q(s_t | s_{t-1}, a_{t-1}, o_t)$  to infer approximate state posteriors from past observations and actions, where  $q(s_t | s_{t-1}, a_{t-1}, o_t)$  is a diagonal Gaussian with mean and variance parameterized by a convolutional neural

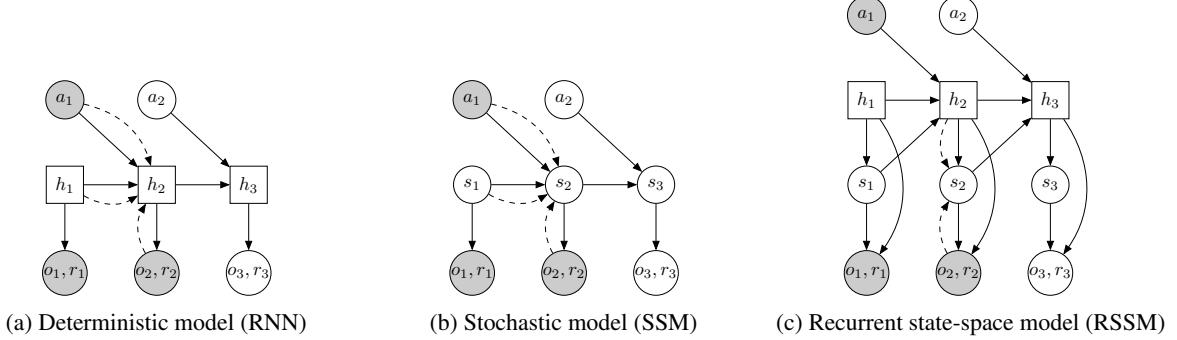


Figure 2: Latent dynamics model designs. In this example, the model observes the first two time steps and predicts the third. Circles represent stochastic variables and squares deterministic variables. Solid lines denote the generative process and dashed lines the inference model. (a) Transitions in 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.

network followed by a feed-forward neural network. We use the filtering posterior that conditions on past observations 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; Gregor & Besse, 2018).

**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

$$\begin{aligned} \ln p(o_{1:T} | a_{1:T}) &\triangleq \ln \int \prod_t p(s_t | s_{t-1}, a_{t-1}) p(o_t | s_t) ds_{1:T} \\ &\geq \sum_{t=1}^T \left( \underbrace{\mathbb{E}_{\substack{q(s_t | o_{\leq t}, a_{<t}) \\ \text{reconstruction}}} [\ln p(o_t | s_t)] - \mathbb{E}_{\substack{q(s_{t-1} | o_{\leq t-1}, a_{<t-1}) \\ \text{complexity}}} [\text{KL}[q(s_t | o_{\leq t}, a_{<t}) \| p(s_t | s_{t-1}, a_{t-1})]] \right). \end{aligned} \quad (3)$$

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),

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

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  $q(s_{1:T} | o_{1:T}, a_{1:T}) = \prod_{t=1}^T q(s_t | h_t, o_t)$  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.

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 (Equation 3). As shown 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 stochastic path of the transition function  $p(s_t | s_{t-1}, a_{t-1})$  is only trained via the KL-divergence regularizers for one-step predictions: the gradient flows through  $p(s_t | s_{t-1}, a_{t-1})$  directly into  $q(s_{t-1})$  but never traverses a chain of multiple  $p(s_t | s_{t-1}, a_{t-1})$ . In this section, we generalize this variational bound to *latent overshooting*, which trains all multi-step predictions in latent space. We found that several dynamics models benefit

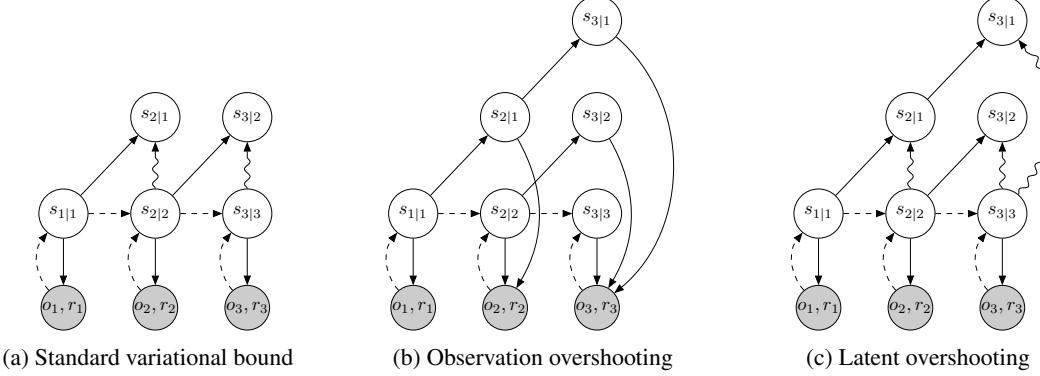


Figure 3: Unrolling schemes. The labels  $s_{i|j}$  are short for the state at time  $i$  conditioned on observations up to time  $j$ . Arrows pointing at shaded circles indicate log-likelihood loss terms. Wavy arrows indicate KL-divergence loss terms. (a) The standard variational objectives decodes 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.

from latent overshooting, although our final agent using the RSSM model does not require it (see Appendix D).

**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 (Krishnan et al., 2015; Lamb et al., 2016; Chiappa et al., 2017), 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,

$$\begin{aligned} p(s_t | s_{t-d}) &\triangleq \int \prod_{\tau=t-d+1}^t p(s_\tau | s_{\tau-1}) ds_{t-d+1:t-1} \\ &= E_{p(s_{t-1}|s_{t-d})}[p(s_t | s_{t-1})]. \end{aligned} \quad (5)$$

The case  $d = 1$  recovers the one-step transitions used in the original model. Given this definition of a multi-step predic-

tion, we generalize Equation 3 to the variational bound on the multi-step predictive distribution  $p_d$ ,

$$\begin{aligned} \ln p_d(o_{1:T}) &\triangleq \ln \int \prod_{t=1}^T p(s_t | s_{t-d}) p(o_t | s_t) ds_{1:T} \\ &\geq \sum_{t=1}^T \left( \underbrace{\mathbb{E}_{q(s_t | o_{\leq t})}[\ln p(o_t | s_t)]}_{\text{reconstruction}} - \underbrace{\mathbb{E}_{\substack{p(s_t | s_{t-d}) q(s_{t-d} | o_{\leq t-d}) \\ p(s_{t-1} | s_{t-d}) q(s_{t-d} | o_{\leq t-d})}}[\text{KL}[q(s_t | o_{\leq t}) \| p(s_t | s_{t-1})]] \right). \end{aligned} \quad (6)$$

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  $I(s_t; s_{t-d}) \leq I(s_t; s_{t-1})$  and thus  $E[\ln p_d(o_{1:T})] \leq E[\ln p(o_{1:T})]$ . 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 predic-

tions of all distances  $1 \leq d \leq D$ ,

$$\begin{aligned} \frac{1}{D} \sum_{d=1}^D \ln p_d(o_{1:T}) &\geq \sum_{t=1}^T \left( \underbrace{\mathbb{E}_{q(s_t|o_{\leq t})}[\ln p(o_t|s_t)]}_{\text{reconstruction}} \right. \\ &\quad \left. - \frac{1}{D} \sum_{d=1}^D \beta_d \mathbb{E}_{p(s_{t-1}|s_{t-d})q(s_{t-d}|o_{\leq t-d})} [\text{KL}[q(s_t|o_{\leq t}) \| p(s_t|s_{t-1})]] \right). \end{aligned} \quad (7)$$

latent overshooting

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  $\beta$ -VAE (Higgins et al., 2016). While we set all  $\beta_d$  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.

## 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, iterative planning, and online experience collection. We refer to the appendix for hyper parameters (Appendix A) and additional experiments (Appendices C to E). Besides the action repeat, we use the same hyper parameters for all tasks. Within less than one hundredth 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 10 to 20 hours (depending on the task) on a single Nvidia V100 GPU compares favorably to that of A3C and 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, reacher has a sparse reward given when the hand and goal area overlap, finger spinning includes contact dynamics between the finger and the object, cheetah exhibits 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 100 episodes, PlaNet outperforms the policy-gradient method A3C trained from proprioceptive states for 100,000 episodes, on all tasks. After 500 episodes, it achieves performance similar to D4PG, trained from images for 100,000 episodes, except for the finger task. PlaNet surpasses the final performance of D4PG with a relative improvement of 26% on the cheetah running task. 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 deterministic part allows the model to remember information over many time steps. The stochastic component is even more important – the agent does not learn without it. This could be 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.

**Agent designs** Figure 5 compares PlaNet, a version collecting episodes under random actions rather than by planning, and a version that at each environment step selects the best action out of 1000 sequences rather than iteratively refining plans via CEM. We observe that online data collection helps for all tasks and is necessary for the cartpole, finger, and walker tasks. Iterative search for action sequences using CEM improves performance on all 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  $C$  update steps. We use the same hyper parameters as for the main experiments above. The agent solves all tasks while learning slower compared to individually trained agents. This indicates that the model can learn to predict multiple domains, 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;

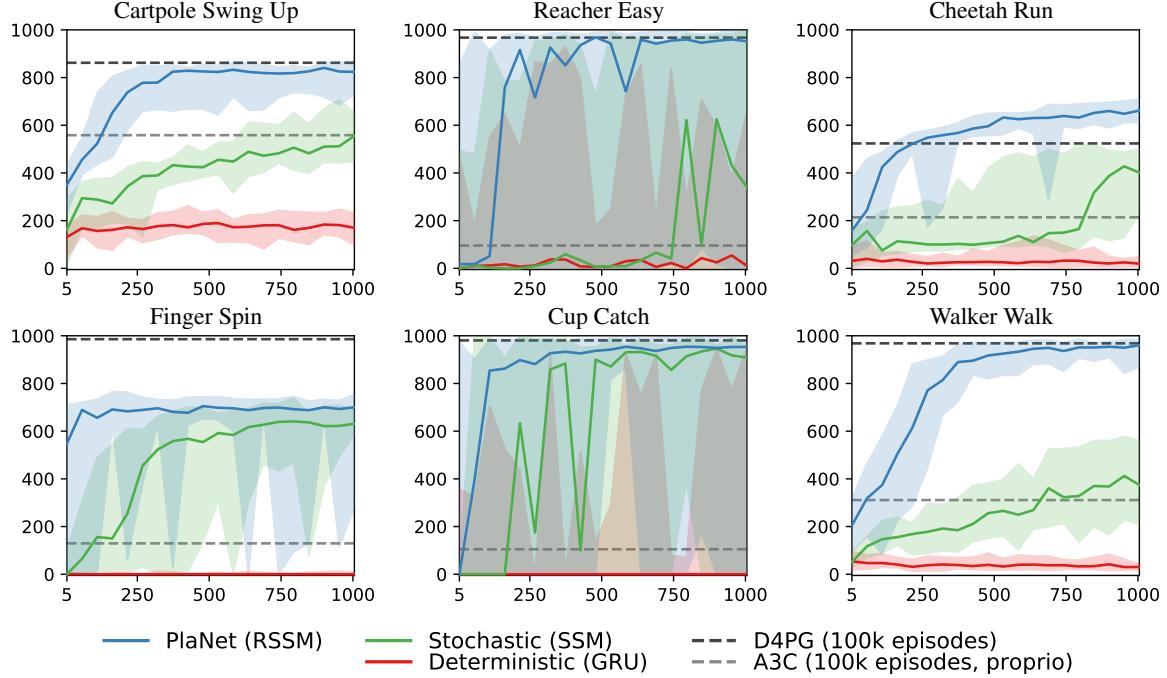


Figure 4: Comparison of PlaNet to model-free algorithms and other model designs. Plots show test performance over the number of collected episodes. We compare PlaNet using our RSSM (Section 3) to purely deterministic (GRU) and purely stochastic models (SSM). The RNN does not use latent overshooting,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

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 5 seeds and 10 trajectories.

Method	Modality	Episodes	Cartpole Swing Up	Reacher Easy	Cheetah Run	Finger Spin	Cup Catch	Walker Walk
A3C	proprioceptive	100,000	558	285	214	129	105	311
D4PG	pixels	100,000	862	967	524	985	980	968
PlaNet (ours)	pixels	1,000	821	832	662	700	930	951
CEM + true simulator	simulator state	0	850	964	656	825	993	994
Data efficiency gain PlaNet over D4PG (factor)			250	40	500+	300	100	90

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; Gre-

gor & Besse, 2018). Appendix G 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; Zhang et al., 2018) that deal with the visual complexity of the real world and solve tasks with

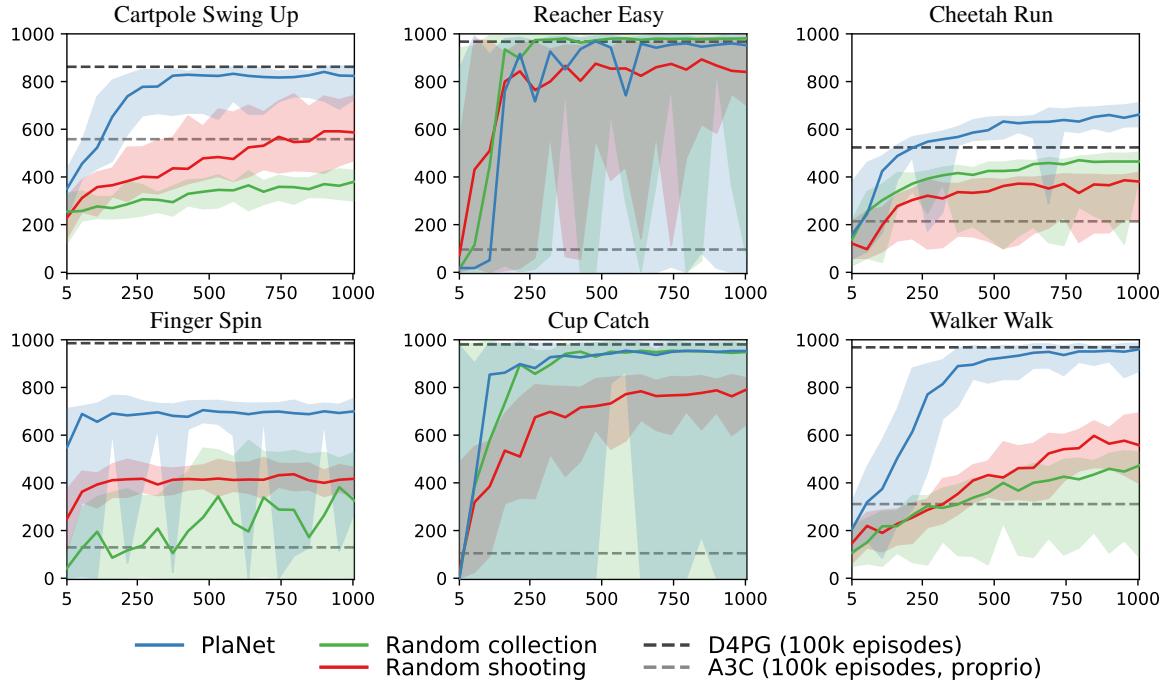


Figure 5: Comparison of agent designs. Plots show test performance over the number of collected episodes. We compare PlaNet, a version that collects data under random actions (random collection), and a version that chooses the best action out of 1000 sequences at each environment step (random shooting) without iteratively refining plans via CEM. The lines show medians and the areas show percentiles 5 to 95 over 5 seeds and 10 trajectories.

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. We show that our agent succeeds at several continuous control tasks from image observations, reaching performance that is comparable to the best model-free algorithms while using  $200\times$  fewer episodes and similar or less computation 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, gradient-based planning could increase the computational efficiency of the agent and learning representations without reconstruction could help to solve tasks with higher visual diversity. 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.

## A. Hyper Parameters

We use the convolutional and deconvolutional networks from Ha & Schmidhuber (2018), a GRU (Cho et al., 2014) with 200 units as deterministic path 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 diagonal 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  $L = 50$ . We do not scale the KL divergence terms relatively to the reconstruction terms but grant the model 3 free nats by clipping the divergence loss below this value. In a previous version of the agent, we used latent overshooting and an additional fixed global prior, but we found this to not be necessary.

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  $\epsilon \sim \text{Normal}(0, 0.3)$  action noise. The action repeat differs between domains: cartpole ( $R = 8$ ), reacher ( $R = 4$ ), cheetah ( $R = 4$ ), finger ( $R = 2$ ), cup ( $R = 4$ ), walker ( $R = 2$ ). We found important hyper parameters to be the action repeat, the KL-divergence scales  $\beta$ , and the learning rate.

## B. Planning Algorithm

**Algorithm 2:** Latent planning with CEM

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

```

1 Initialize factorized belief over action sequences  $q(a_{t:t+H}) \leftarrow \text{Normal}(0, \mathbb{I})$ .
2 for optimization iteration  $i = 1..I$  do
3   // Evaluate  $J$  action sequences from the current belief.
4   for candidate action sequence  $j = 1..J$  do
5      $a_{t:t+H}^{(j)} \sim q(a_{t:t+H})$ 
6      $s_{t:t+H+1}^{(j)} \sim q(s_t | o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p(s_\tau | s_{\tau-1}, a_{\tau-1}^{(j)})$ 
7      $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} \mathbb{E}[p(r_\tau | s_\tau^{(j)})]$ 
8     // Re-fit belief to the  $K$  best action sequences.
9      $\mathcal{K} \leftarrow \text{argsort}(\{R^{(j)}\}_{j=1}^J)_{1:K}$ 
10     $\mu_{t:t+H} = \frac{1}{K} \sum_{k \in \mathcal{K}} a_{t:t+H}^{(k)}$ ,  $\sigma_{t:t+H} = \sqrt{\frac{1}{K-1} \sum_{k \in \mathcal{K}} |a_{t:t+H}^{(k)} - \mu_{t:t+H}|^2}$ .
11     $q(a_{t:t+H}) \leftarrow \text{Normal}(\mu_{t:t+H}, \sigma_{t:t+H}^2 \mathbb{I})$ 
10 return first action mean  $\mu_t$ .
```