Towards integrating model dynamics for sample efficient reinforcement learning

Aashi Manglik
Robotics Institute
Carnegie Mellon University
amanglik@andrew.cmu.edu

Shashank Tripathi
Robotics Institute
Carnegie Mellon University
shashant@andrew.cmu.edu

Abstract

Despite outperforming average human performance on Atari games, model-free reinforcement learning requires millions of training examples restricting their application to environments in simulation. Humans, on the contrary, are much more efficient in modeling environmental priors which allows faster learning of optimal control policies. This work attempts to develop a principled approach to solve the sample inefficiency problems associated while deploying model-free approaches in real environments. Specifically, we wish to learn the dynamics model using episodes generated in the real world, and use the learned model to augment real-world episodes as the training progresses. We do not assume the availability of a simulator, which necessitates that the agent learns the dynamics model with relatively fewer real-world examples. We find that augmenting real world data using an approximation of the dynamics model can be more sample-efficient than naive model-free reinforcement learning.

1 Introduction

In this work, we explore the task of increasing the sample efficiency of model-free reinforcement learning methods such as DQN[8] and REINFORCE[16]. As access to the real environment is usually limited, we wish to limit the number of interactions required with the real environment for finding the optimal policy. One way to solve this problem is to learn the dynamics function of the real environment, which can then be used to generate fake episodes for training the reinforcement learning agent. The model dynamics can be approximated using a neural network whose parameters are trained on transitions sampled from the real environment.

Given access to the transition function, it is also possible to deploy model-based reinforcement learning methods such as Value Iteration, Monte Carlo Tree Search (MCTS) [13] or Model Predictive Control (MPC) [9]. However, when the state and action space is large, a model-based approach struggles to generate a solution in real-time. Many problems in computer vision and robotics, such as robot navigation, requires optimal action to be computed within a few milliseconds. Thus, doing a Monte Carlo Tree Search (MCTS) using the transition function is not feasible within the desired time budget. In contrast, in model-free reinforcement learning, a forward pass on a trained network could output the optimal action in dynamic environments much faster. Even though model-free methods are fast at test time, training a deep model-free network requires millions of interactions with the real environment. To increase sample efficiency, we propose to learn the dynamics model on the fly. Augmenting real world episodes with episodes generated by the learned model could reduce the number of episodes to be collected in real world.

For experimentation, we test our approach on two environments: CartPole and Maze. Cartpole is an easy environment with low-dimensional state-space and frequent rewards. We also create a custom Maze environment that simulates a robot navigating in a crowd. The robot has to reach a defined goal location and avoid any collision with fellow pedestrians. In this case, states are 16×16 RGB images and rewards are sparse, only occurring when the agent reaches the goal or collides with pedestrian.

1.1 Related Work

The wide popularity of Reinforcement Learning has been largely fueled by model-free deep reinforcement learning algorithms. Be it the task of playing video-games like an expert [8], defeating the top ranking human in Go[12] or learning complex gaits for locomotion [3], model-free approaches like Deep Q networks [8] and policy gradients [14] have been shown to be successful in a wide array of tasks. However, such methods typically require millions of samples to achieve a reasonable performance[2] restricting their application to simulated environments. In many real-world applications, either a simulator is not available, or even if it is available, optimal policies learned in simulation do not port well to the real-world. This inherent issue with model-free approaches obviates their deployment in solving real world environments.

On the other end of the spectrum, model-based reinforcement learning approaches assume a transition function that can predict future states given current state and actions. Having an accurate model of the world can be thought of as having our own simulator which we can use to do multi-step lookahead for planning. It is for this reason that model-based reinforcement learning algorithms are widely considered to be more sample efficient in comparison to their model-free counterparts [2]. However, in complex environments, when the observation space is large, building accurate world-models is often a very challenging task, almost tantamount to predicting the future [11, 5, 1]. This problem is further aggravated in stochastic environments. For the task of model learning in image-based environments, we take inspiration from the vast body of work in video frame prediction. Walker et al[15], propose a variational autoencoder based approach to predict trajectories of pixels in image sequences. Along the same lines, Lotter et al[7] present a neural network architecture called "PredNet" for predicting future frames. These methods however do not condition video prediction on agent/user actions as is common in robotic applications and Atari like environments.

Our dynamics learning neural network model has been largely inspired by the work of Oh et al[10], who propose an autoencoder-like deep learning architecture to predict future frames conditioned on actions. While Oh et al only predict the next frame, Leibfried et al[6] extend their architecture to predict future rewards as well. In this work, we explore both these architectures. Additionally, we also study an adversarial loss formulation to improve next frame prediction.

The closest work in this direction is by Nagabandi et al [9] who propose to use a deep neural network dynamics model to initialize a model-free learner. They apply model predictive control on the neural network dynamics approximation and use that as expert trajectories to initialize model-free learning. Our approach is more robust as we sample even sub-optimal trajectories over the course of training.

2 Environments

We analyze the sample efficiency of model-free approaches in two different environments. The first environment is the well-known CartPole available in OpenAI gym.

The second problem addresses robot navigation in crowded environments where the state observations consist of a bird's eye view (RGB image) of the scene as shown in figure 1. Specifically, we want to train an agent navigating in a maze scenario to reach a goal state in the shortest time. The environment is dynamic in nature and is filled with moving obstacles (people) which the agent needs to avoid. Obstacles move along the same direction until they reach the boundary after which they turn around. At the start of each episode, obstacles are placed randomly in the environment and start patrolling in a randomly chosen direction. The stochasticity in initial obstacle placement and direction of motion adds a high-degree of complexity to this task. The agent needs to learn from RGB images, the high-level construct of trading-off between colliding with obstacles and reaching the goal quickly. Collisions with the obstacle state render a -5 reward whereas reaching the goal state accrues +1 reward for every step the agent stays on that state. At each timestep, the agent will choose one of the five possible actions - move left, right, forward, backward or wait.

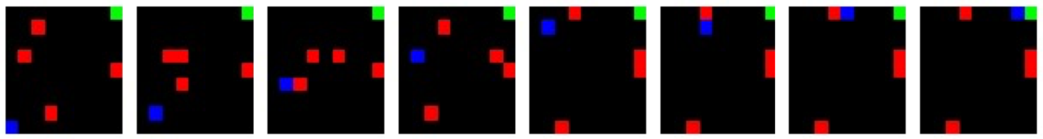


Figure 1: **Maze Environment:** Blue square is the agent, red squares are moving obstacles, and green square is the static goal. Agent receives a negative reward on collision with obstacles and positive reward for reaching the goal. Faster it reaches the goal, more is the positive reward.

3 Methods

3.1 Prediction from physical state: CartPole environment

Cartpole's physical state is represented by a 4-dimensional vector comprising of position of cart, velocity of cart, pole angle and pole's angular velocity. A multilayer perceptron shown in figure 2 is trained to jointly predict the next state s', reward r and probability that the next state is terminal p using the following loss function.

$$L(\theta) = \sum_{i=1}^{N} ||s'_i - trueS_i||^2 + (r_i - trueR_i)^2 - (d \log p + (1-d)\log(1-p)) \tag{1}$$

In above equation, θ represents the parameters of multilayer perceptron. $trueS_i$, $trueR_i$ and d denote the true next state, reward and terminal flag sampled from real environment respectively.
We used two hidden layers of 256 units each and a Stochastic gradient descent (SGD) optimizer with a learning rate 0.01.

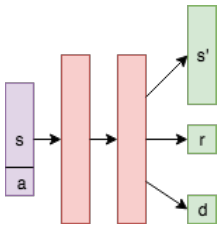


Figure 2: Network Architecture for learning the model dynamics of CartPole environment

3.1.1 Effect of the policy on state distribution

The policy of our agent determines the visitation distribution on states. For example, an optimal policy will only take the agent to good states. To capture the true underlying model dynamics, we trained the dynamics network on diverse policies as described in section 3.1.2. This ensures that the training does not have any bias in the state visitation distribution and the network could generalize well to any unseen state and action pair.

3.1.2 Training

Our method uses two different networks, one for learning the optimal policy and another for learning the environment's dynamics.

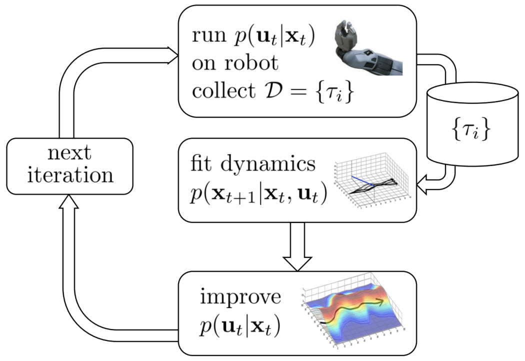
- **Learning the optimal policy**
  To train for optimal policy, we implemented REINFORCE. For CartPole, REINFORCE network is a multilayer perceptron with a single hidden layer of 8 units. It takes the four dimensional state as input and outputs the probability over two actions.
- **Dynamics model**
  The loss function and architecture of dynamics model is described in section 3.1.

Iterative updates in Policy gradients and Dynamics model
The training is divided into two phases.

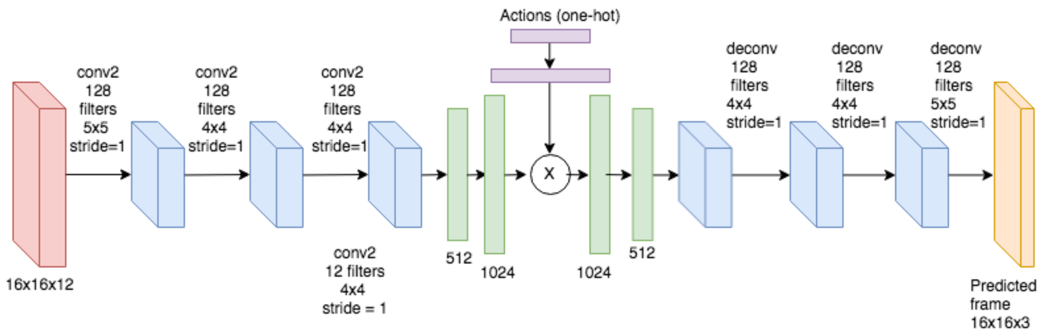
- In the initial phase, we do not generate any fake episodes as the dynamics model is not trained yet. The policy network updates its policy after each episode and the sampled transitions $(s, a, r, s', done)$ are also used by the dynamics network to optimize the supervised loss given by the equation 1.
- After the completion of $N = 100$ episodes, the trained dynamics model is used to generate fake episodes following the training loop illustrated in figure 3.

3.2 Prediction from raw pixels: Maze environment

We use the same training procedure f r the Maze environment. Here, our dynamics model architecture 4 consists of an encoder-decoder module [10]. In order to capture temporal dependencies, the encoder module takes four consecutive RGB frames, concatenated along channels, as input.


run $p(\mathbf{u}_t|\mathbf{x}_t)$
on robot
collect $\mathcal{D} = \{\tau_i\}$
$\{\tau_i\}$
next iteration
fit dynamics
$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$
improve
$p(\mathbf{u}_t|\mathbf{x}_t)$
Figure 3: Iterative training of policy network and the dynamics model

The input frames are passed through a feed-forward encoding module which consist of three convolutional layers followed by a fully connected layer. We hypothesize that the feed-forward features capture high-level semantic information better, which is not explicitly contained in raw images. Since the next state is conditionally dependent on both past states and action, we take hadamard product between action features and feed-forward features. The combined features are then passed though a decoder module which consists of three transpose-convolutional layers. The input vector to the upsampling module can therefore be represented as: $h^{dec} = W^{dec}(W^{enc}h^{enc} \odot W^a a) + b$. The upsampling module is exactly symmetric to the downsampling module. To stabilize learning and ensure better gradient flows, we use Batch Normalization[4] after every layer except for the input layer. Each layer is also followed by a ReLU non-linearity.


Actions (one-hot)
conv2 128 filters 5x5 stride=1
conv2 128 filters 4x4 stride=1
conv2 128 filters 4x4 stride=1
conv2 12 filters 4x4 stride = 1
512
1024
X
1024
512
deconv 128 filters 4x4 stride=1
deconv 128 filters 4x4 stride=1
deconv 128 filters 5x5 stride=1
16x16x12
Predicted frame 16x16x3
Figure 4: Network Architecture for learning the dynamics of maze environment

We tried several loss functions to optimize the dynamics networks:

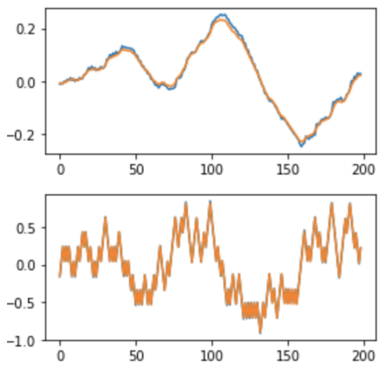
1. L1 loss: $\mathcal{L}_i = |\hat{x_i} - x_i|$
2. L2 loss: $\mathcal{L}_i = \frac{1}{2}||\hat{x_i} - x_i||^2$
3. Multi-label Binary Cross Entropy loss:
$$-\sum_i y[i] * \log((1 + \exp(-x[i]))^{-1}) + (1 - y[i]) * \log\left(\frac{\exp(-x[i])}{(1 + \exp(-x[i]))}\right)$$

However, since we are dealing with stochastic environments, optimization using $\mathcal{L}_1$ and $\mathcal{L}_2$ loss resulted in averaging over the possible locations. This led to blurry results. Multi-label binary cross-entropy loss with a threshold of 0.2 resulted in much better recall on agents, obstacles and goal states.

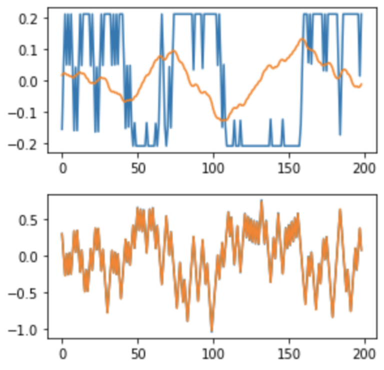
4 Results

4.1 Experiment 1: Predictions and sample efficiency for CartPole

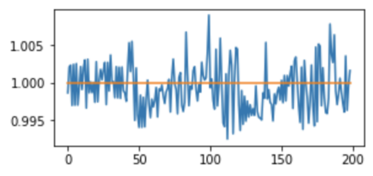
The predictions of the trained dynamics model 2 are compared with the ground truth in Fig 5. The blue curve shows the states and rewards from rolling out one fake episode while the orange curve represents the real environment from OpenAI. The network accurately predicts the position, linear velocity and pole's angular velocity. However, it struggled to make correct predictions for pole angle. We rolled out the fake episodes using these predictions to train policy gradients alternately with real episodes.



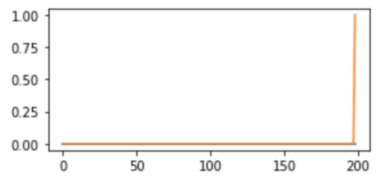
(a) Cart's position (above) and velocity (below)



(b) Pole angle (above) and its angular velocity (below)

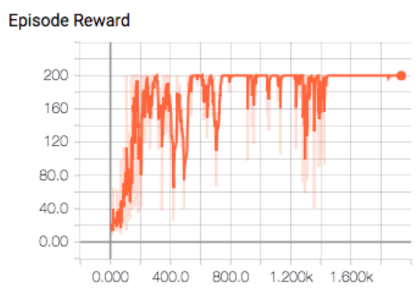


(c) Reward prediction

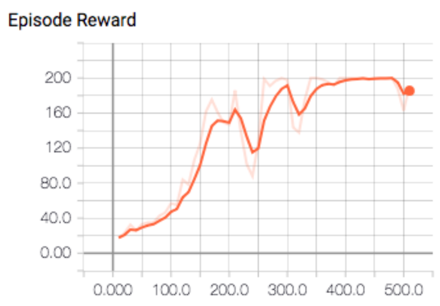


(d) Terminal state prediction

Figure 5: Predictions from model dynamics approximation



(a) REINFORCE



(b) REINFORCE with dynamics model

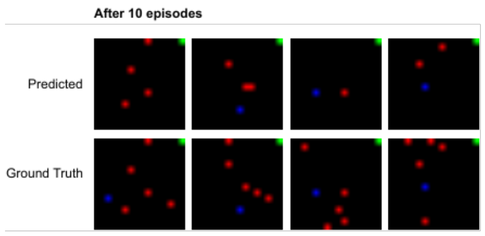
Figure 6: Average training reward vs number of real episodes

With an addition of fake episodes, the policy gradients required lesser number of interactions with real environment as shown in learning curves 6(a) and 6(b). Averaging over 10 runs, REINFORCE alone solved the environment in 650 episodes whereas model-free with model dynamics solved it in 400.

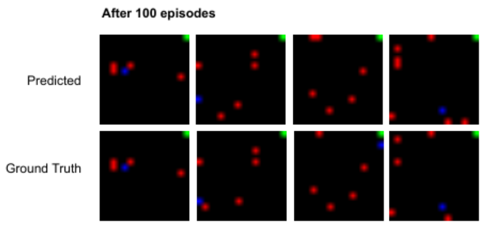
4.2 Experiment 2: Predictions and sample efficiency for Maze

The predictions of model 4 are compared with the ground truth in figures 7(a), 7(b), and 7(c) after 10, 100 and 300 episodes respectively. As can be qualitatively observed, our model learns to make accurate predictions on agent and goal states after 10 episodes. Obstacle trajectories are also predicted accurately after 100 episodes of training. At this stage, the average F1 score on obstacle predictions is 0.83.

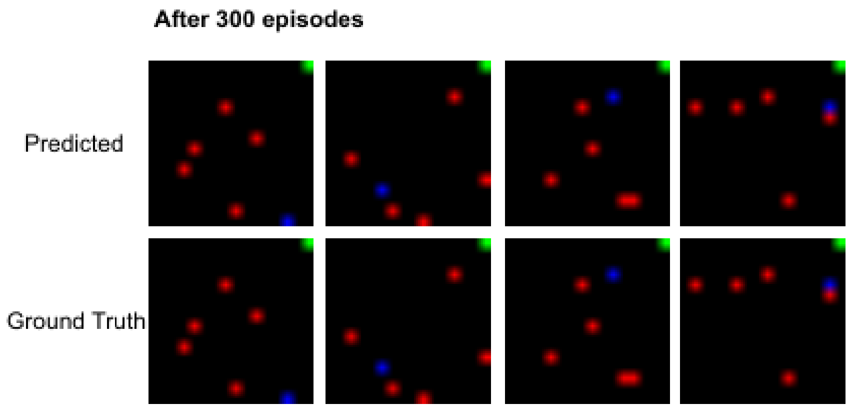
Training episodes	F1 score : Agent	F1 score : Obstacles	F1 score : Goal
After 10 episodes	0.83	0.22	0.97
After 100 episodes	0.92	0.83	0.99
After 300 episodes	0.98	0.93	1



(a) Next state predictions



(b) Next state predictions after 100 episodes

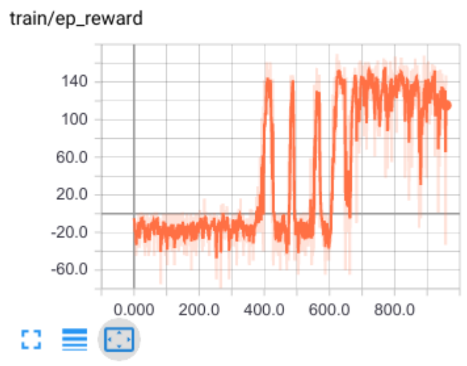


(c) Next state predictions after 300 episodes

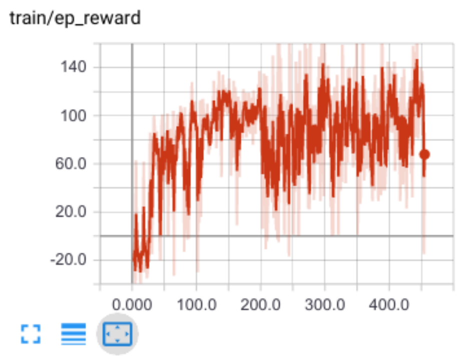
Figure 7: Progress of the trained dynamics model

With an addition of fake episodes, DQN required lesser number of interactions with real environment. 8(a) and 8(b). Averaging over 10 runs, DQN alone solved the environment in 700 episodes whereas on augmentation with model dynamics solved it in 250. We can see some interesting policies that emerge over training in the video: `https://github.com/sha2nkt/drl_project_final`. Since rewards from collision are more frequent, the agent first learns to maneuver in the environment without colliding. Eventually, the agent learns to wait until the obstacle has passed before proceeding to the goal state. Finally, it learns to quickly reach the goal while avoiding collision. This behaviour is reminiscent of how humans behave in crowded environments.

Method	Real episodes on CartPole	Real episodes on Maze
Model-free	650	700
Model-free with dynamics model	400	250



(a) DQN



(b) DQN with dynamics model

Figure 8: Average training rewards vs number of real episodes in Maze

4.3 Experiment 3: Predictions for Pong

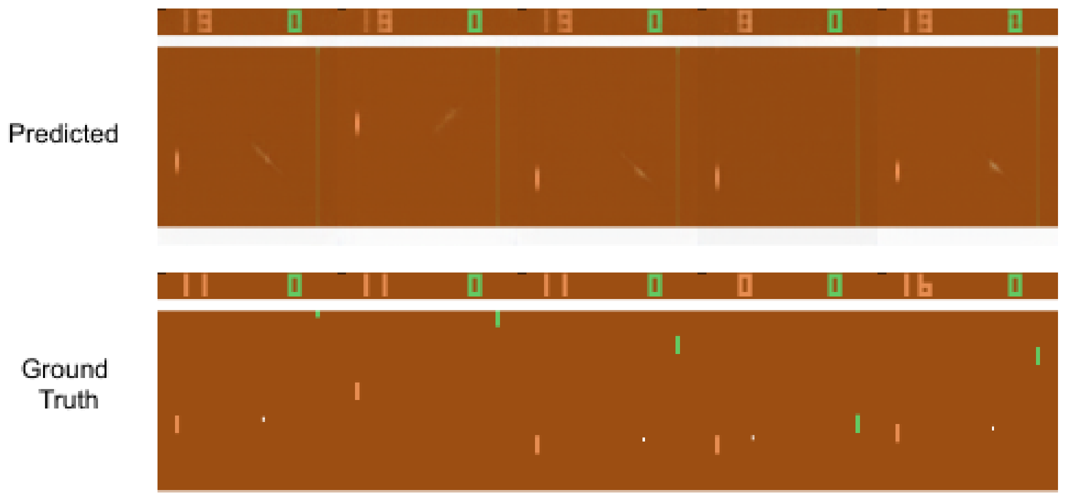


Figure 9: Prediction model on Pong

For dealing with high-dimensional state space in Atari environments, we prefer to work with $\mathcal{L}_2$ loss formulation over multi-label binary cross entropy. We tried training our model on random policy at first. Unfortunately, even with extensive hyperparameter tuning, we couldn't resolve the averaging/blurring effects on the moving ball and paddle locations.

We therefore decided to integrate an adversarial loss formulation for jointly optimizing our model along with the standard $\mathcal{L}_2$ loss.

For the Generative model, we use a similar architecture as in 4. In addition, we compute a binary generative cross-entropy loss using the output features of the upsampling module. The generative model is thus optimized using the following objective.

$$\mathcal{L}_g = \frac{1}{2}||\hat{x}_i - x_i||^2 + \lambda \mathbb{E}_{z \sim p_z(z)}[log(D(G(z)))]$$

where λ is the weighting between $\mathcal{L}_2$ loss and the generative cross-entropy loss.

The discriminator network is optimized using the standard GAN objective:

$$\mathcal{L}_d = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

As can be seen in Fig. 9, the task of learning the model becomes hard when the state-space dimensionality increases. Even though we were able to predict the location of the ball and the opponent paddle, the results are only an approximation. It is worth noting that the agent paddle is blurred out in Fig. 9 as we trained our dynamics model on a random policy. Adding such blurry states confuses the DQN agent, leading to deterioration in sample efficiency. We also notice that the errors compound over noisy steps in an episode.

5 Conclusion

We observe from our experiments that the increase in sample efficiency is correlated with the accuracy of the dynamics model. If the predictions are accurate, fake episodes can be good substitutes for real interactions. From our experience with high dimensional state-space environments like Pong, we claim that the bottleneck in our approach is the model-learner. It can be hard for current generative networks to learn the true underlying dynamics. However, our hope rides on current trends towards speedy progress in generative modeling. Given accurate generative models, we can extend this approach for real-world applicability.

References

[1] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

[2] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[3] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

[6] Felix Leibfried, Nate Kushman, and Katja Hofmann. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*, 2016.

[7] William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[10] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[11] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

[12] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[13] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[14] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[15] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016.

[16] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.