Model-Based Reinforcement Learning for Playing Atari Games

Justin Fu
justinfu@cs.stanford.edu

Irving Hsu
irvhsu@cs.stanford.edu

Abstract

In this project, we explore the domain of model-based reinforcement learning applied to playing Atari games from images. Recently, search-based methods such as AlphaGo have proven to be effective for complex and long-term planning in games such as Go. In a general setting, we do not have a perfect model of an environment such as in Go, but recent work has also explored building predictive models for Atari games using video frames. This project thus proposes an integrated methodology that employs model-based control in conjunction with a deeper investigation of existing search algorithms. We used the Arcade Learning Environment (ALE) to perform both data collection and evaluation, and implemented both a model-free policy gradient agent and a predictive model based on convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to serve as a model for a planning algorithm based on tree search.

We evaluate our methods against a human benchmark and DQN on two simple games. While we were unable to outperform DQN, we were able to surpass human performance in Pong using the policy gradient method and MCTS. Our results demonstrate that MCTS shows promise and potential for planning in reinforcement learning tasks.

1. Introduction

Within the domain of reinforcement learning (RL), one of the long-standing challenges is learning to control agents from high-dimensional sensory inputs such as vision and speech. Combining modern reinforcement learning and deep learning approaches holds much promise for making advancements in challenging applications involving both rich perception and policy selection. For example, a classic application area of reinforcement learning is in robotics, where deep convolutional networks (CNNs) have been applied to produce joint torque commands given a combination of vision and proprioception [6]. Reinforcement learning also has applications outside of typical agent vs. nature environments - for example, it has also been applied to training neural attention models for image captioning [4].

A popular benchmark for evaluating reinforcement learning progress in recent years has been on the Atari game domain, provided through the Arcade Learning Environment (ALE), an object-oriented framework that allows researchers to easily develop AI agents. Deep-Q Networks (DQNs) [7, 8] were a recent breakthrough in combining model-free RL with deep learning, and use a modified version of Q-Learning incorporating experience replay with a convolutional neural network (CNN) for function approximation, which proved the effectiveness of neural networks in the field of reinforcement learning.

In our project, we wish to explore model-based control for playing Atari games from images. Our motivation is to build a general learning algorithm for Atari games, but model-free reinforcement learning methods such as DQN have trouble with planning over extended time periods (for example, in the game Montezuma's Revenge). We hope that by factoring the problem into two parts – modeling and planning – we can improve performance by using long-horizon planning algorithms such as Monte Carlo Tree Search (MCTS).

2. Related Work

Many current deep reinforcement learning approaches fall in the model-free reinforcement learning paradigm, which contains many approaches ranging from approximate dynamic programming [7], to policy gradients [3], to search with supervised learning [6, 9].

Search methods have recently enjoyed much success in long-term planning tasks. For example, AlphaGo [1] is a MCTS-based method which uses a policy network to select candidate actions and a value network to score states. By using MCTS, and narrowing the set of candidate actions, AlphaGo can efficiently consider 20+ moves in the future even though Go has a large branching factor, and is currently the state-of-the-art for computer Go playing.

In a general setting, we do not have a perfect model of the world as we do in an abstract game such as Go. Our project is inspired from recent work using convolutional neural networks to build visual models of Atari games [2] from sequences of image and actions, which were shown to be very accurate for control on a short timescale, and better than random for as long as 100 timesteps, which show promise for a model-based control approach.

3. Background

In our project, we implemented two methods for controlling an agent in ALE – a model-free policy gradient method, and a model-based tree search method.

3.1. Likelihood-ratio/Policy Gradients

The likelihood ratio method is a gradient-based policy search algorithm which directly optimizes a parametrized policy by computing a gradient estimate from rollouts, and using that to update the policy using standard numerical optimization techniques such as stochastic gradient descent/ascent (SGD).

Let $\pi_\theta(a|s)$ be a stochastic policy (it represents a distribution over actions) that is parametrized by θ. We wish to maximize the expected sum of total rewards:

$$R_{tot} = E[R_0 + \gamma R_1 + \gamma^2 R_2 + ...]$$

Evaluating the gradient of this directly can be intractable, but note that we can push a gradient inside an expectation with the "REINFORCE trick" [5] (in this example, X is a random variable and $f(X)$ is a function of X):

$$\begin{aligned}\frac{\partial}{\partial\theta}E[f(X)] &= \frac{\partial}{\partial\theta}\int f(x)p_\theta(x)dx \\ &= \int f(x)\frac{\frac{\partial}{\partial\theta}p_\theta(x)}{p_\theta(x)}p_\theta(x)dx \\ &= \int f(x)\frac{\partial}{\partial\theta}\log p_\theta(x)p_\theta(x)dx \\ &= E[f(X)\frac{\partial}{\partial\theta}\log p_\theta(x)\end{aligned}$$

By replacing $f(X)$ with the sum of rewards, $p_\theta(x)$ with our policy, and simplifying some terms, we can derive the policy gradient estimator

$$\frac{\partial}{\partial\theta}R_{tot} = E[\sum_{t=0}^{T-1}[\frac{\partial}{\partial\theta}\log\pi_\theta(a|s)\sum_{t'=t}^{T-1}\gamma^{t'}R_{t'}]]$$

Now that the expectation is outside of the gradient, we can use Monte Carlo estimation to approximate the expectation by simply averaging across samples. Applying this method simply requires that the policy $\pi(a|s)$ is differentiable, so we can use a function approximator such as a neural network or a linear approximator.

3.2. Monte Carlo Tree Search

Given a model, the planning agent determines how to use the model to pick an action to take. Since the action space is discrete, we used search-based methods.

Monte Carlo Tree Search (MCTS) is a randomized tree search method, and is akin to performing open-loop control. Rather than exhaustively explore a tree using method such as depth-first search, we sample trajectories through the tree by sampling children uniformly at each decision point. In the context of MDPs, each node in the tree is a state, each edge is an action, and we are trying to maximize the total reward accrued during a path. After sampling paths and recording rewards obtained, we pick the action with the best average reward.

Naive MCTS can spend much time exploring low-reward sections of the search tree – to force the search to bias towards more fruitful actions, we can weight the sampling by how good the model thinks that action is. As a heuristic, we can estimate the Q-function $\hat{Q_\pi}(s,a)$ and sample according to a softmax:

$$p(a_i \mid s) \propto e^{\hat{Q_\pi}(s,a_i)}$$

This approach is similar to using expert knowledge – if $\hat{Q}_\pi$ is close to Q^{opt}, then we can expect to explore regions with high reward.

4. Dataset and Evaluation

4.1. Dataset

We used the Arcade Learning Environment (ALE) as a simulator in which we performed both data collection and evaluation. The simulator provides interfaces to retrieve the internal RAM of the game as well as RGB images.

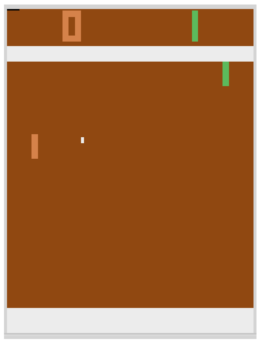


Figure 1: An example screenshot from Pong. The white dot represents the ball, and the green and light brown bars represent the paddles.

To collect data to train our model-based method, we needed trajectories sampled from a policy with high expected reward. Thus, we first trained a model-free method using policy gradients from the RAM of the game (which is much smaller than the image and easier to train on), then collected 1600 200-timestep trajectories of images, rewards, sum of discounted future rewards, and actions using this policy. This served as supervised data for training our models. We downsampled images to 64×64 and flattened them to grayscale to reduce the number of parameters in our model.

4.2. Evaluation

The standard evaluation metric for Atari game playing is the score returned by the game itself. Human performance provides an interpretable baseline, as done in [7, 8].

For our project, we evaluated on two games: Pong and Breakout. Both of these games are reflex-based games that do not require long-term planning, but we were not able to get results better than random on more difficult games such as Seaquest, Qbert, or Montezuma's Revenge.

5. Technical Approach

Our project consisted of implementing a model-free method and a model-based method with two components – modeling and planning. Each component is described in detail below.

5.1. Model-Free Method

As mentioned previously, to generate good training data, we needed an automatic way of collecting trajectories with high reward. To this end, we trained a two-layer network (with 16 hidden units and a tanh nonlinearity) to predict from the 128-dimensional RAM to the probabilities of actions (of which there are 3 possible). The action probability distribution is the softmax of the final layer.

$$x_{hid} = \tanh(W_1 x_{RAM} + b_1)$$

$$y = W_2 x_{hid} + b_2$$

$$\pi_\theta(a_i \mid s) \propto e^{y_i}$$

For Pong, training this network took approximately 5 hours of training time to achieve an average reward of +7.2. Breakout took approximately 4 hours to achieve an average reward of +10.5.

We optimized our policy using SGD with a momentum of 0.9, starting with a learning rate of 5.0 and dropping by 1/2 every 100 iterations. Gradients were obtained using the likelihood ratio method described previously.

5.2. Modeling

The first half of our model-based method was to construct a model of the game, which given some current state of the game, needs to predicts the next state and reward.

We experimented with two different architectures. Our networks are all implemented in Tensorflow, and the code is included in the supplemental files. We optimized our models using Tensorflow's AdagradOptimizer using a learning rate of 0.001. All nonlinearities used were ReLU nonlinearities ($\text{ReLU}(x) = \max(0, x)$), chosen for ease of training. Since we are performing regression, all losses are squared losses

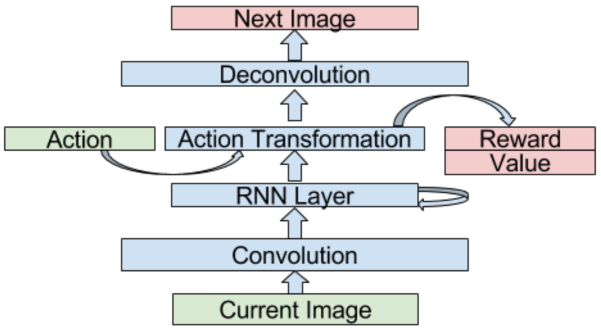


Figure 2: A high-level diagram of our video prediction model architecture. The green boxes represent inputs, and the red boxes represent outputs.

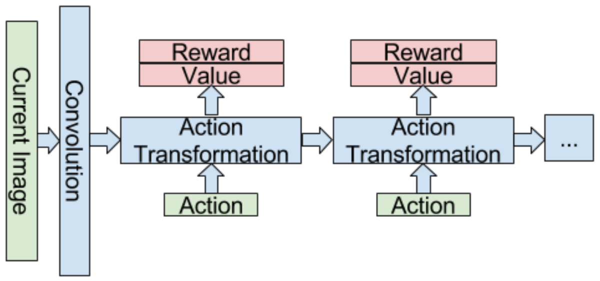


Figure 3: A high-level diagram of our videoless architecture. Here, we only use the convolutional layers once to initialize the hidden state.

with respect to the supervised targets (which can be images, or rewards).

5.2.1 Video Prediction Architecture

Our video prediction network was modeled after the work done in [2], and consists of a series of convolutions, a recurrent layer, and a series of deconvolutions to reconstruct the image. The model is shown in Figure 2. The input to the model is the images and actions, and the supervised target is the next image, rewards, and expected future reward (the value).

Our architecture consists of two convolutional layers ($7 \times 7 \times 3$ and $5 \times 5 \times 3$), a 128-dimensional RNN layer, and two deconvolutional layers ($5 \times 5 \times 3$ and $7 \times 7 \times 3$). To incorporate the action, we embedded each action as a vector, and performed pointwise multiplication with the output of the RNN, similar to what was done in [2]. This is the action-conditional transformation, which saves parameters compared to having a separate weight matrix for each action, given by:

$$\mathbf{h}_t^{transf} = W_{dec}(W_{enc}\mathbf{h}_t^{RNN} \odot a_t^{embed}) + b$$

Because we were not able to get good performance out of this model (the errors compounded too quickly), we instead resorted to a simpler model which did not attempt to reconstruct the images. An example of the (blurry) images we obtained are shown in Figure 4b.

5.2.2 Videoless Architecture

Our second architecture lacks the video prediction element of the previous architecture. The input to the model is the images and actions, and the supervised target is the rewards, and expected future reward (the value). During our rollouts, we embed the initial image (described below) to use as an initial hidden state, and use an RNN-style function to predict the rewards R, values V, and hidden states $\mathbf{h}$ for the rest of the timesteps.

$$\mathbf{h}_0 = \text{CNN}(x_{image})$$

$$\mathbf{h}_{t+1} = ReLU(W_{dec}(W_{enc}\mathbf{h}_t \odot a_t^{embed}) + b)$$

$$R_t = W_{rew}\mathbf{h}_t + b_{rew}$$

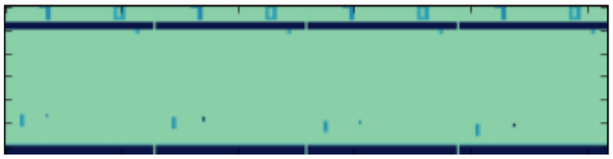
$$V_t = W_{val}\mathbf{h}_t + b_{val}$$

It begins by embedding the image into a 128-dimensional vector with a two-layer CNN, which is used as the initial hidden state of an 128-dimensional RNN. The output of the RNN is then transformed with the actions as done in the video prediction architecture, and used to predict the reward and value of the next state via a single layer. The hidden state is then fed back into the RNN for the next timestep. This architecture is shown in Figure 3.

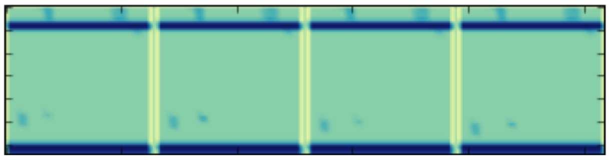
5.3. Planning & Control

We implemented three search methods – Exhaustive tree search (depth-first search), MCTS, and weighted MCTS. Pong has 6 possible actions, so our tree search had a branching factor of 6.

- For depth-first search, we used a depth of 4 (any depth beyond 3-4 was too slow).



(a) Ground Truth



(b) Predicted

Figure 4: Ground truth and predicted images (4 timesteps) from the video prediction network for Pong. The images are grayscale with a yellow-to-blue colorscheme.

- For MCTS, we uniformly sample actions up to a depth of 10. We take between 50-150 samples for each decision.
- For weighted MCTS, we used the same parameters as MCTS.

6. Results

We evaluated MCTS, weighted MCTS, and the policy gradient method on two games: Pong and Breakout. Our baselines are drawn from DQN, and include human performance as well as the results of DQN itself. Results for Pong are presented in Table 1 and for Breakout in Table 2.

Planner	**Model**	**Score**
Random Policy [7]	N/A	-20.4
DQN [7]	N/A	20.0
Human [7]	N/A	-3.0
Policy Gradients	N/A	7.2
MCTS Depth 10, 50 Samples	Videoless	-12.1
MCTS Depth 10, 150 Samples	Videoless	-8.1
MCTS Depth 10, 150 Samples	Video	-19.8
Wt. MCTS Depth 10, 150 Samp.	Videoless	2.9

Table 1: Scores achieved by various methods for the game Pong.

6.1. Discussion

Overall, we were able to surpass human performance on Pong, but were unable to come close to the results of DQN using any of our methods. We see a clear improvement in performance using the weighted MCTS method, but it was still unable to surpass the model-free method the model was trained from, and unable to surpass human performance for Breakout.

Since naive/uniform sampling MCTS wastes much of its computation on exploring bad states and actions (i.e., moving away from the ball in Pong and Break-

Planner	**Model**	**Score**
Random Policy [7]	N/A	1.2
DQN [7]	N/A	168
Human [7]	N/A	31
Policy Gradients	N/A	10.5
MCTS Depth 10, 50 Samples	Videoless	2.7
MCTS Depth 10, 150 Samples	Videoless	3.0
Wt. MCTS Depth 10, 150 Samp.	Videoless	6.1

Table 2: Scores achieved by various methods for the game Breakout.

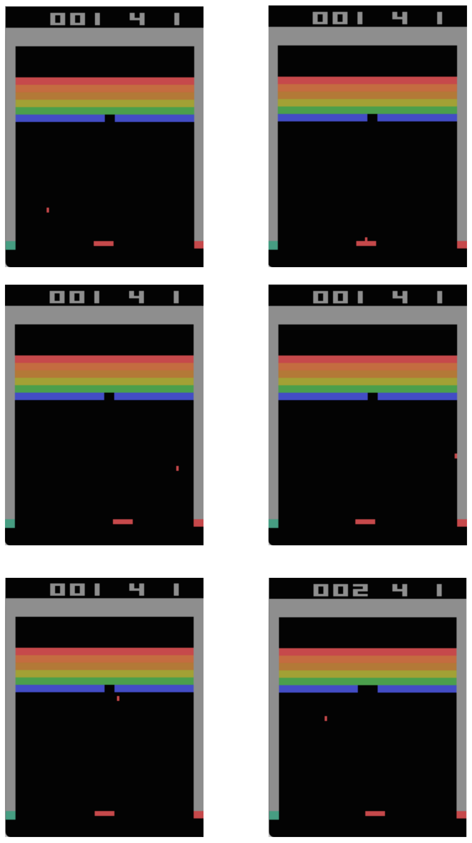


Figure 5: Sample frames from Breakout when playing under Monte Carlo Tree Search.

out), its effective search depth is very limited and it could only plan optimally in the very short term. Qualitatively, we noticed behavior such as correctly hitting the ball back when the paddle was in the vicinity of the ball, but would not even move towards the ball if it was too far away.

The weighted MCTS performed similarly to the model-free policy gradient method, since the model was trained to predict the value function of the policy, so the search was biased towards actions that produced high reward under the model-free policy. Unfortunately, we were not able to surpass the model-free method likely due to compounding errors in the model.

7. Conclusions

In this project, we explored both model-free and model-based approaches for playing simple Atari games from images. In our experiments, we found that weighting MCTS towards high reward regions was the most effective search method out of the three that we implemented (the others being uniform sampling of actions and depth-first search), because it was able to focus its exploration towards states that the model thought was promising.

A major issue with our model-based methods was the problem of compounding errors when doing open-loop control. Normally, we need highly accurate models to perform open-loop control, but our models made errors even after just several timesteps. We encountered this greatly when trying to predict images, and to a lesser degree when only predicting rewards and expected future rewards.

We were also hindered by many engineering and computation issues. Collecting data and training models were incredibly time-consuming, and data collection itself also took many hours of computation. If we had more time to collect data, and more time to train and iterate on our models, we expect much better performance. Due to compounding errors, inaccurate models hurts performance greatly.

8. Future Work

An interesting future topic to explore is hierarchical planning – for example, when humans play a multi-task game such as Montezuma's Revenge (where the agent needs to perform tasks such as finding a key, and using it to open a door), we tend to break the problem up into pieces, and solve the current task with fine-grain detail while having a very rough idea of what should happen next. It's possible that methods such as AlphaGo [1], which ends its search at an evaluation function, has some capability of this.

Another alternative topic to explore is transfer learning. Many games involve moving figures in 2-dimensional space, and there are many phenomena that apply across all games. For example, contiguous objects don't move around the screen randomly and usually have some velocity which controls where the object will end up on the next frame. Once our model is set up, we can explore how well our model generalizes: if we can model moving up/down/left/right in one game, can we predict the same in another similar game? Humans certainly can pick up on cues such as this quickly in our internal model of the world.

9. Software Used

Our project was based in Python used libraries such as Tensorflow (for building models), PIL (for processing images), numpy/scipy, and the Arcade Learning Environment (for simulation).

10. Acknowledgements

The authors would like to thank Irwan Bello and Justin Johnson for their continued guidance and feedback throughout the project.

References

[1] D. Silver, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529:484–489, 2016.

[2] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh. Action-Conditional Video Prediction using Deep Networks in Atari Games. *Advances in Neural Information Processing Systems 28*, 2015.

[3] J. Schulman, S. Levine, P. Moritz, M.I. Jordan, P. Abbeel. Trust Region Policy Optimization. *International Conference on Machine Learning*, 2015.

[4] K. Xu, J. Ba, R. Kiros, K. Cho, A.C. Courville, R. Salakhutdinov, R.S. Zemel, Y. Bengio. Show, Attend and Tell: Neural Image Caption generation with Visual Attention. *International Conference on Machine Learning*, 2015.

[5] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[6] S. Levine, C. Finn, T. Darrell, P. Abbeel. End-to-End Training of Deep Visuomotor Policies. *CoRR*, abs/1504.00702, 2015.

[7] V. Mnih, et al. Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop*, 2013.

[8] V. Mnih, et al. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518:529–533, 2015.

[9] X. Guo, S. Singh, H. Lee, R. Lewis, X. Wang. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. pages 3338–3346, 2014.