Deep Reinforcement Learning for Long Term Strategy Games

Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala

(akhilay, ashwini1, meghaj) @stanford.edu
Department of Computer Science
Stanford University

Introduction

Deep reinforcement learning has been increasingly successful in a wide variety of tasks. Deep Mind's Deep Q-network (DQN) was able to surpass the overall performance of a professional human player in most Atari games. However, games like Seaquest, Private Eye, Montezuma's Revenge which demand a more temporally extended planning strategy still constitute a major challenge for all existing agents.

The primary difficulty for learning goal-directed behaviour in such games is that the rewards received from the environments are sparse and the state space is relatively large to explore. Thus if the agent receives some additional environment specific information to help aid its exploration, it might be able to learn robust value functions. The primary objective of our project is to explore different heuristics which can be provided to the agent along with the architectural changes that can be made to help the agent learn a long term strategy in sparse reward games.

We decided to explore sparse reward setup with respect to two environments:

1. Atari "Montezuma's Revenge"
2. A complex discrete stochastic markov decision process (MDP)

We have trained our agent using the Open AI Gym environment for Montezuma's Revenge using Asynchronous Advantage Actor Critic - Context Tree Switching (A3C - CTS) and a hierarchical Deep Q-network (DQN) architecture. Additionally we also tested the hierarchical DQN architecture on a complex discrete stochastic MDP environment.

Related Work

Most of the recent breakthroughs in Deep Reinforcement Learning were spearheaded through the seminal paper by Mnih et. al. [7] where they proposed Deep Q Network

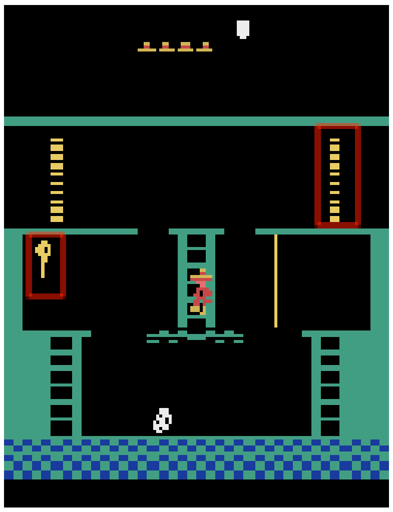


Figure 1: Room 1 of Montezuma's Revenge with only 2 sources of extrinsic reward

(DQN). Further details are present in the section describing the Baseline Methods.

In Asynchronous Advantage Actor Critic by Mnih et. al. [6] multiple parallel actor learners learn in parallel. Further details are present in the Baseline Methods .

One of the earliest works of hierarchical reinforcement learning is by Dayan and Hinton (1993) where they proposed Feudal Reinforcement learning (FRL) [2]. In FRL there are multiple levels of hierarchy within an agent. A level in the hierarchy communicates to the level below it what must be achieved, but does not specify how to do so.

A more recent work by Vezhnevets et. al. [9] introduces Feudal Networks (FuN) where each of the individual Workers are actor-learners, that learn the policy for goal provided by Manager. They propose a novel RNN design for the Manager – a dilated LSTM – which extends the longevity of the recurrent state memories and allows gradients to flow through large hops in time, enabling effective back-propagation through hundreds of steps.

There has also been a significant progress in nonhierarchical deep RL methods by using auxiliary losses and rewards. Houthooft et. al. [3] proposed Variational Information Maximizing Exploration (VIME), an exploration strategy based on maximization of information gain about the agent's belief of environment dynamics. Bellemare et al. [1] have significantly advanced the state-of-the-art on Montezuma's Revenge by using pseudo-count based auxiliary rewards for exploration, which stimulate agents to explore new parts of the state space.

Methods

Baselines:

1. **Deep Q-Learning (DQN)**
   A deep Q-network [7] combines Reinforcement Learning with deep neural networks. The goal of the agent is to select actions which maximizes the cumulative future reward. The function Q is parameterized as a convolutional neural network.

   Additionally an experience replay memory is used to sample random experiences during Q-learning updates so that there is no correlation between the experiences, thereby reducing the variance of the updates.

   $$L(\theta_i) = E_{(s,a,r,s')\sim D}[(r + \gamma max_{a'} Q(s', a', \theta_i^-) - Q(s, a, \theta_i))^2]$$

   where D is the experience replay memory, θ_i are the parameters of Q network at iteration i, θ_i^- are the network parameters used to compute the target at iteration i.

2. **Asynchronous Advantage Actor Critic (A3C)**
   Consider a function J(π) as a discounted reward that a policy π can gain, averaged over all possible starting states s_0.
   $$J(\pi) = E_{\rho^{s_0}}[V(s_0)]$$
   where p^{s_0} is a distribution of starting states.
   Then the gradient of J w.r.t. w is

   $$\nabla_w J(\pi) = E_{s\sim\rho^\pi, a\sim\pi(s)}[A(s,a) \cdot \nabla_w log\pi(a|s)]$$

   where:
   a) $\nabla_w log\pi(a|s)$ is the actor: since it points in which direction the probability of taking action a in state s increases
   b) $A(s,a)$ is the critic: it tells us the advantage of taking this action in this context compared the average. The advantage function is defined as
   $A(s,a) = Q(s,a)$ - $V(s)$ = r + γ V(s') - V(s).

   This forms the **Actor-Critic** [4] model which maintains a policy π and an estimate of the value function V. It operates in a "forward view" and uses mix of n-steps returns to update both the policy and the value function which gives the model more stability and faster training. **Asynchronous Advantage Actor-Critic (A3C)** [6] is a variant of the actor-critic model where multiple actor-learners are running in parallel and are exploring different parts of the environment.

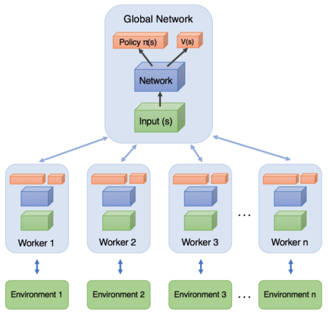


Figure 2: High level architecture of A3C

Our Approaches:

1. **Context Tree Switching (CTS)**
   CTS [8] is a Bayesian variable order Markov Model that takes a 2D image and assigns to it a probability according to the product of location-dependent L-shaped filters. The CTS density model ρ is then used to calculate pseudo-counts, $\hat{N}_n(x)$. $N(\hat{x}) = \rho(x)\hat{n}(x)$, where $\hat{n}(x)$ can be thought of as a total pseudo-count computed from the model's recoding probability $\rho'(x)$, the probability of x computed immediately after training on x.

   Pseudo-counts provided by the density model can then be converted to exploration bonuses ($R^+ n(s,a)$), which incentivizes the agent to re-visit less frequently visited states and so it reduces it's uncertainty about the environment in the absence of any direct environmental reward.

   $(R^+ n(s,a)) = \frac{\beta}{\hat{N}_n(s)+0.01}$

2. **Asynchronous Advantage Actor Critic with Context Tree Switching (A3C-CTS)**
   Actor-critic methods have explicit separation of policy and Q-function parameters, which leads to richer behavioral space. This very separation however, often leads to deficient exploration. In games with sparse rewards, it might be useful for the agent to explore the state-space in a better way by using intrinsic motivation, which provides the agent with an exploration bonus and guides it to explore novel states. In A3C-CTS algorithm [1], pseudo-count derived from CTS density model is used to generate an exploration bonus.

In A3C-CTS, during training the density model is updated with the states generated by following a policy. During the policy gradient step, the intrinsic rewards are computed by querying the density model and adding it to extrinsic rewards before clipping them in range [-1, 1].

3. **Deep Q-Network with Context Tree Switching (DQN CTS)**
   In games with sparse rewards, DQN can benefit by exploring the state-space in a guided manner by revisiting novel states, and hence reducing it's uncertainty about the environment. Similar to A3C, DQN-CTS involves use of the CTS density model during training to compute intrinsic rewards, which are then added to extrinsic rewards.

4. **Hierarchical Deep Q-Network (h-DQN)**
   A hierarchical-DQN model (Figure 3) takes decisions over two levels of hierarchy [5]:

   a) The *meta-controller* that looks at the raw states and produces a policy over goals by estimating the value function $Q_1(s_t, g_t; \theta_1)$ by maximizing expected future extrinsic reward
   $$L_1(\theta_i) = E_{(s,g,f,s')\sim D_1}[(r + \gamma max_{g'} Q_1(s', g'; \theta_i^-) - Q_1(s, g; \theta_i))^2]$$
   where D_1 stores the experiences (s, g, f, s'); f is the extrinsic reward from the environment, θ_i are the parameters of Q network at iteration i, θ_i^- are the network parameters used to compute the target at iteration i.

   b) The *actor-controller* takes in states and the current goal, and produces a policy over actions by estimating the value function $Q_2(s_t, a_t; \theta_1, g_t)$ to solve the predicted goal. The internal critic checks if goal is reached and provides an appropriate intrinsic reward to the controller. The controller terminates either when the episode ends or when g is accomplished. The meta-controller then chooses a new g and the process (a - b) repeats.
   $$L_2(\theta_i) = E_{(s,a,g,r,s')\sim D_2}[(r + \gamma max_{a'} Q_2(s', a'; \theta_i^-, g) - Q_2(s, a; \theta_i, g))^2]$$
   where D_2 stores the experiences (s, a, g, r, s'); r is the intrinsic reward from the critic.

Results and Analysis

Baseline Results

1. **Deep Q-Network (DQN)**
   - *Input to CNN in the DQN:* 4 consecutive 84 X 84 frames (after downscaling and grayscale conversion).
   - *Hyperparameters:* RMSProp with minibatch size 64, $\epsilon - greedy$ with ϵ annealed linearly from 1 to 0.1 over 1 million frames, replay memory size: 1M

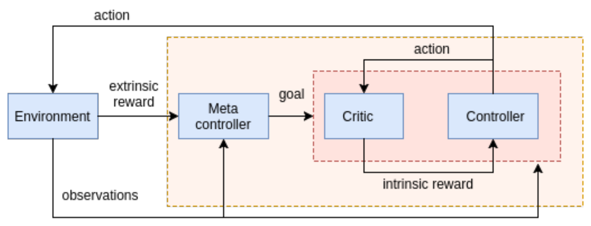


Figure 3: Hierarchical framework architecture

   - The agent failed to learn an optimal policy and received an extrinsic reward of 0.
   - The agent would take actions that would cause it to die immediately. We therefore decided to explore other methods that involve exploration of a larger state space.

2. **Asynchronous Advantage Actor Critic (A3C)**
   - 16 parallel actor learners and a global learner.
   - Parameters of actor-learners were synchronized with the global learner after every 120K steps.
   - *Hyperparameters:* RMSProp with minibatch size 32.

   Although there were 16 parallel actor learners, the worker did not learn a strategy to reach the key. Hence it did not receive any reward.

Our Approaches' Results

1. **Deep Q-Network - Context Tree Switching (DQN-CTS)**
   - In addition to the existing set up of DQN, CTS density heuristic was used to aid the agent with exploration bonuses to explore novel states.
   - *Hyperparameters for CTS:* $\beta = 0.05$, $\eta = 0.9$, images are downsampled to 42X42
   - Agent obtained a maximum reward of +100 after 5 hours

   Since A3C is a stronger baseline compared to DQN, we decided to utilize our limited compute resources instead to train an agent using A3C-CTS [6] on Montezuma's Revenge.

2. **Asynchronous Advantage Actor Critic - Context Tree Switching (A3C-CTS)**
   - In addition to the existing set up of A3C, CTS density model heuristic was used to aid the agent with exploration bonuses to explore novel states.
   - *Hyperparameters for CTS:* $\beta = 0.05$, $\eta = 0.9$, images are downsampled to 42X42
   - All the workers successfully learnt to cross the first room, and few of them learnt to cross the second room as well, resulting in a maximum score of 1400.

Following are the results obtained on running A3C-CTS:

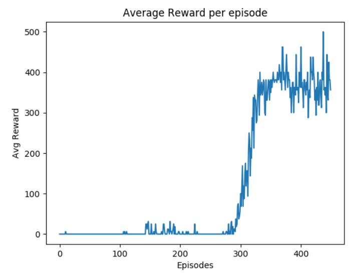


Figure 4: Average Reward over episodes for A3C-CTS

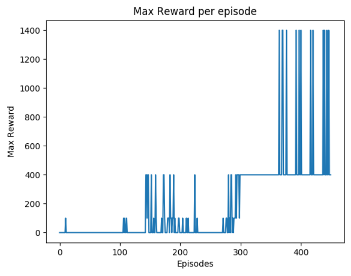


Figure 5: Maximum Reward over episodes for A3C-CTS

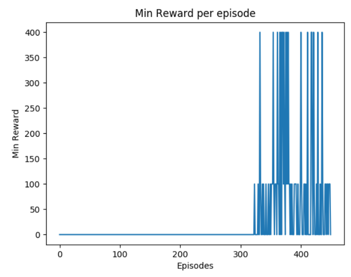


Figure 6: Minimum Reward over episodes for A3C-CTS

3. **Hierarchical framework**
We implemented the hierarchical framework with respect to two environments:
a) A discrete stochastic markov decision process
b) Atari game "Montezuma's Revenge"

Stochastic MDP

- **Game Setup**
  - 6 possible states s_1 to s_6 as shown in Figure 7.
  - Start state: s_2, Terminal state: s_1.
  - Moves *left* deterministically when it chooses left action; action right only succeeds 50% of the time
  - Reward = 1 when it first visits s_6 and then s_1, Reward = 0.01 for reaching s_1 without visiting s_6

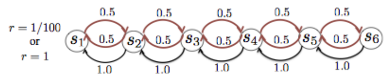


Figure 7: Stochastic MDP where the reward at the terminal state s_1 depends on whether s_6 is visited or not

- **Results**
  - *Hyperparameters:* ϵ greedy strategy with the ϵ being annealed from 1 to 0.1 in 12,000 steps, learning rate $\alpha = 0.00025$, Loss: Huber loss (Smooth L1 Loss), RMSProp with minibatch size 128
  - Agent learnt to choose goals s_4, s_5 or s_6, which statistically lead the agent to visit s_6 before going back to s_1.

  All the figures below illustrate that the model learns to choose goals in a way so that it reaches s_6 more often, thus increasing the rewards and the length of an episode with time.

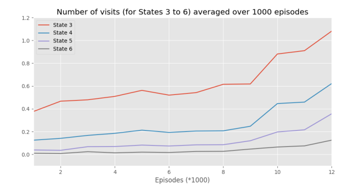


Figure 8: Number of visits (for states s_3 to s_6). The initial state is s_2 and the terminal state is s_1

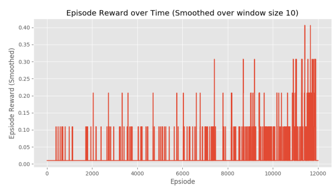


Figure 9: Increase in episode rewards over time

Atari Montezuma's Revenge

- **Game Setup**
  - We considered the first room of Montezuma's revenge. Here "sub goals" are entities in the game like the middle ladder, the bottom left ladder etc.
  - We used a finite state machine (FSM) as the meta controller. Given the current state, the FSM outputs the next sub goal to be reached.

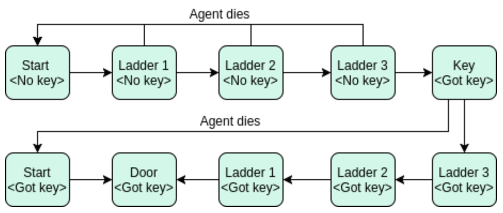


Figure 10: Finite State Machine for Montezuma's Revenge

– Environment: Open AI Gym
– Actor-controller is a convolutional neural network (Figure 11).
– *Input to CNN:* 4 consecutive 90 X 80 frames (after downscaling and grayscale conversion) along with a binary mask of the next sub goal.
– Actor controller learns a policy over the 18 possible actions to reach the sub goal from a given state

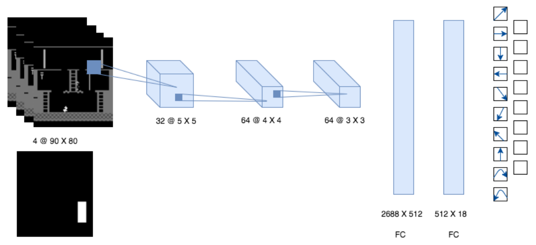


Figure 11: DQN architecture for the actor-controller

- **Results**
  – Learning rate = 0.00025, discount rate $\gamma = 0.99$, ϵ greedy strategy with the ϵ being annealed from 1 to 0.1 in 50,000 steps, Loss: Huber loss (Smooth L1 Loss), RMSProp with minibatch size 30, replay memory size: 50000
  – Agent has successfully learnt to consistently reach the key at around 1200 episodes.
  – We were unable to train our agent for a longer time due to limited compute resources. However due to the consistent decrease in training loss with every update (Figure 12), we feel that with sufficient training time the agent trained using this architecture will reach the door.

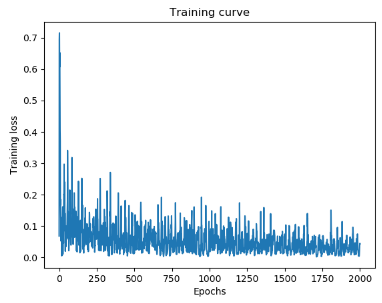


Figure 12: Training loss against epochs

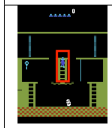
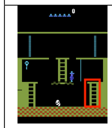
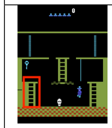
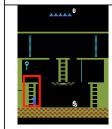
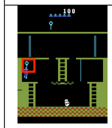
Frame	**State**	**Goal**	**Q values**
	start	ladder1	UP:7.72 RIGHT:8.31 LEFT:9.55 **DOWN:11.34**
	ladder1	ladder2	UPFIRE:10.20 **RIGHTFIRE:11.78** LEFTFIRE: 9.83 DOWNFIRE:0.19
	ladder2	ladder3	UPFIRE:11.08 RIGHTFIRE:8.16 **LEFTFIRE: 11.45** DOWNFIRE:0.07
	ladder2	ladder3	**UP:11.86** RIGHT:4.9 LEFT: 1.14 DOWN:0.56
	ladder3	key	**UPFIRE:11.35** RIGHTFIRE:0.34 LEFTFIRE: 5.08 DOWNFIRE:0.06

Table 1: Q values for different states and actions of the game

Future Work

In this work, we have used a finite state machine to output the sub-goals at each state for Montezuma's revenge. We would like to use a neural network that can learn to output the correct sequence of sub-goals given the current history of the game. We would also like to extend the hierarchical DQN to other Atari games with sparse rewards like Seaquest.

Consolidated Results

	Avg Reward	Std. Dev.	Max Reward	Min Reward	No. Training steps	Training Time
DQN	0	0	0	0	6.2 M	8 hours
A3C	0	0	0	0	0.15 M	6 hours
A3C-CTS	362.5	22	1400	0	6.3 M	8 hours
DQN-CTS	2	28	100	0	7.8 M	5 hours
H-DQN	34.7	6	100	0	1.92 M	20 hours

Contributions

Task	Implementer
Background Survey	Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala
DQN/A3C + CTS Baseline	Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala
A3C-CTS Baseline	Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala
h-DQN implementation + debugging	Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala
Plots/Figures	Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala
Report	Akhila Yerukola, Ashwini Pokle, Megha Jhunjhunwala

References

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Variational information maximizing exploration. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 3675–3683, 2016.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

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.

Joel Veness, Kee Siong Ng, Marcus Hutter, and Michael Bowling. Context tree switching. In *Data Compression Conference (DCC), 2012*, pages 327–336. IEEE, 2012.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.