Policy Gradient Methods In Deep Reinforcement Learning

MS Project Defense

Sebastián Gracia Guzmán

School of Mathematical and Statistical Sciences
Clemson University

April 9, 2021

Overview

Machine Learning

Machine learning has three main fields that share overlaps. All fall under the scope of Artificial Intelligence which share the same goal of creating computer program which mimic intelligence found in natural life.

Machine Learning

Machine learning has three main fields that share overlaps. All fall under the scope of Artificial Intelligence which share the same goal of creating computer program which mimic intelligence found in natural life.

1. **Supervised Learning** - Statistical Learning, *(Image classification, forecasting)*

Machine Learning

Machine learning has three main fields that share overlaps. All fall under the scope of Artificial Intelligence which share the same goal of creating computer program which mimic intelligence found in natural life.

1. **Supervised Learning** - Statistical Learning, *(Image classification, forecasting)*
2. **Unsupervised Learning** - Identifying structures or patterns within unlabeled data. *(Netflix recommendations, medical imaging)*

Machine Learning

Machine learning has three main fields that share overlaps. All fall under the scope of Artificial Intelligence which share the same goal of creating computer program which mimic intelligence found in natural life.

1. **Supervised Learning** - Statistical Learning, *(Image classification, forecasting)*
2. **Unsupervised Learning** - Identifying structures or patterns within unlabeled data. *(Netflix recommendations, medical imaging)*
3. **Reinforcement Learning** - Interact with environment and learn to act optimally through trial and error. *(Autonomous driving, robotics, animation, much more)*

Machine Learning

Machine learning has three main fields that share overlaps. All fall under the scope of Artificial Intelligence which share the same goal of creating computer program which mimic intelligence found in natural life.

3. **Reinforcement Learning** - Interact with environment and learn to act optimally through trial and error. *(Autonomous driving, robotics, animation, much more)*

Agent Environment Interaction

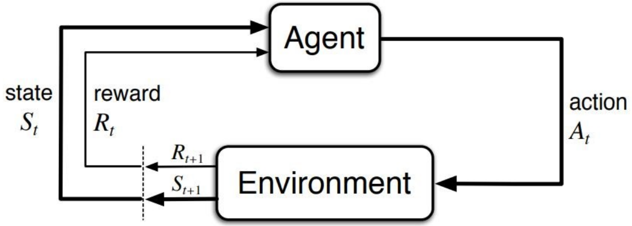


Figure: Agent environment interaction given by Sutton & Barto, 2017.

Agent Environment Interaction

1. Agent receives a state, s, from environment.

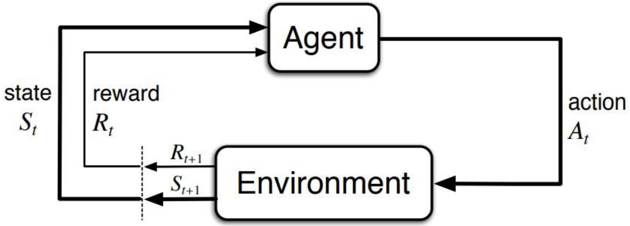


Figure: Agent environment interaction given by Sutton & Barto, 2017.

Agent Environment Interaction

1. Agent receives a state, s, from environment.
2. Agent takes an action or "step" in the world with action a.

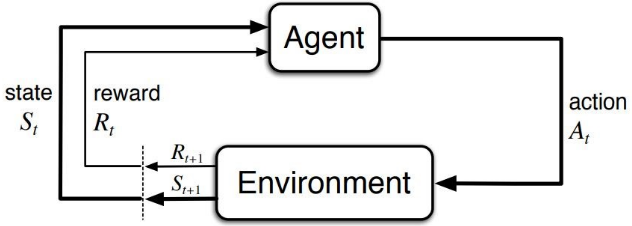


Figure: Agent environment interaction given by Sutton & Barto, 2017.

Agent Environment Interaction

1. Agent receives a state, s, from environment.
2. Agent takes an action or "step" in the world with action a.
3. Agent receives reward, r, from environment and new state, s'.

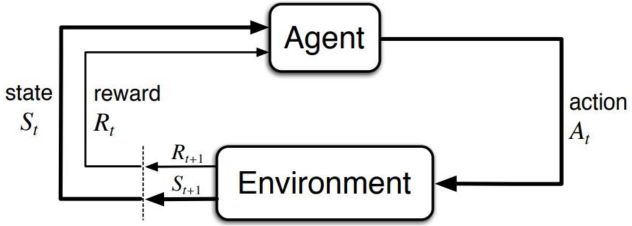


Figure: Agent environment interaction given by Sutton & Barto, 2017.

Agent Environment Interaction

1. Agent receives a state, s, from environment.
2. Agent takes an action or "step" in the world with action a.
3. Agent receives reward, r, from environment and new state, s'.
4. Repeats until horizon T. *Horizon may be infinity.*

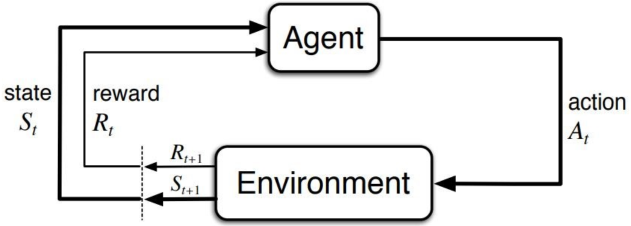


Figure: Agent environment interaction given by Sutton & Barto, 2017.

Agent Environment Interaction

1. Agent receives a state, s, from environment.
2. Agent takes an action or "step" in the world with action a.
3. Agent receives reward, r, from environment and new state, s'.
4. Repeats until horizon T. *Horizon may be infinity.*

Goal: Maximize total expectation of discounted, cumulative rewards obtained.

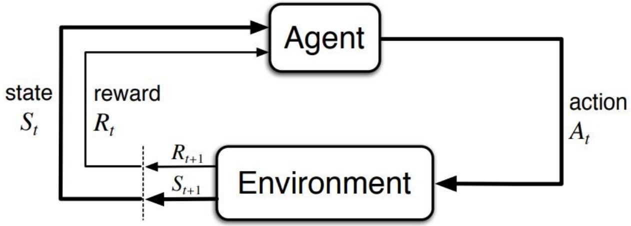


Figure: Agent environment interaction given by Sutton & Barto, 2017.

Markov Decision Process (MDP)

Agent environment interaction can be modeled using a Markov Decision Process.

Markov Process

Sequence of random states with the Markov property.
A series of states $s_1, s_2, \ldots$ is Markov if and only if $\mathbb{P}[s_{t+1}|s_t] = \mathbb{P}[s_{t+1}|s_1, \ldots, s_t]$

MDP

An MDP can be described by a state space S, an action space A, a transition function, and a reward function $R(s, a)$.

Transition Function

$p(s'|s, a) = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a]$

Policy Function

A *policy* is a way for an agent to determine actions based on a given state.

$$\begin{aligned} \pi(s) &= a && \text{(Deterministic Policy)} \\ \pi(a|s) &= \mathbb{P}[a_t = a | s_t = s] && \text{(Stochastic Policy)} \end{aligned}$$

Goal: To obtain optimal policy. Knowing and taking which actions will incur the greatest expectation of reward.

State Value Function

Need to criteria to decide when one policy is better than another policy. Introduce the state value function. Function tells us how valuable a state is by giving the expected discounted rewards incurred from state s onward:

$$v(s) = \mathbb{E}\left[G_t | S_t = s\right].$$

Where $\gamma \in [0, 1)$ is a discount rate parameter and discounted rewards are:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^t R_{t+k+1}.$$

State Value Function Cont.

When acting under a specific policy, π, we arrive at the *state value function for policy* π:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right].$$

Here, $v_\pi(s)$ gives expected rewards when taking actions according to policy, π.
Use this to define a partial ordering over policies.

$$\pi_1 \succcurlyeq \pi_2 \iff v_{\pi_1}(s) \geq v_{\pi_2}(s) \quad \forall s \in S.$$

Optimal Policy and State Value Function

Formulate the optimal policy and value function.

$$v^*(s) = \max_{\pi} v_\pi(s) = v_{\pi^*}(s) \quad \forall s \in S.$$

$$\pi^* \succcurlyeq \pi \quad \forall \pi.$$

State Action Value Function

Similar to the value function, the state action value function or q-function, gives a the expected future rewards after taking action a at state s.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Can easily formulate a deterministic policy from the q-function.

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, q(s, a)$$

Using this strategy to obtain the optimal policy is called q-learning. There exists families of algorithms relying upon this methodology called q-learning methods.

Bellman Optimality Equations

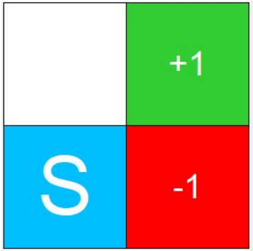
In the case in which the agent does have full access to the MDP transition probability function, the environment can be solved with an optimal solution. We can rewrite the optimal value function with the transition probability as:

$$\begin{aligned}
v^*(s) &= \max_a q_{\pi^*}(s,a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]
\end{aligned}$$

GridWorld Example

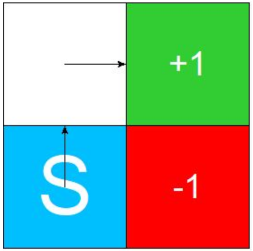
State space:
$S = \{(1,1),(1,2),(2,1),(2,2)\}$
Action space:
$A = \{north, south, east, west\}$

Goal: Solve the environment using Bellman Optimiality Equations



Solving Bellman Optimalitity Equations

$$v^*(2,2) = 1$$
$$v^*(2,1) = -1$$
$$v^*(1,2) = \max\{0.9v^*(1,2), 0.9v^*(1,1), 0.9v^*(2,2), 0.9v^*(1,2)\} = .9$$
$$v^*(1,1) = \max\{0.9v^*(1,2), 0.9v^*(1,1), -0.9v^*(2,1), 0.9v^*(1,1)\} = .81$$



Noisy GridWorld

- Consider a new environment in which there is 0.3 noise in the environment.
- If an agent selects an action there is a 0.3 probability that any adjacent action is taken instead.
- Ex. If an agent selects action north there is 0.3 probability of taking actions east or west.

How will stochasticity change the system of equations?

Stochasticity Increases Complexity

$$v^*(2,2) = 1$$

$$v^*(2,1) = -1$$

$$v^*(1,2) = \max \left\{ \begin{array}{l} .85 * .9v^*(1,2) + .15 * .9v^*(2,2), \\ .7 * .9v^*(1,1) + .15 * .9v^*(2,2) + .15 * .9v^*(1,2), \\ \color{red}{.85 * .9v^*(2,2) + .15 * .9v^*(1,1)}, \\ .85 * .9v^*(1,2) + .15 * .9v^*(1,1) \end{array} \right\} \approx .8$$

$$v^*(1,1) = \max \left\{ \begin{array}{l} .7 * .9v^*(1,2) + .15 * .9v^*(2,1) + .15 * .9v^*(1,1), \\ .85 * .9v^*(1,1) + .15 * .9v^*(2,1), \\ .85 * .9v^*(2,1) + .15 * .9v^*(1,1), \\ \color{red}{.85 * .9v^*(1,1) + .15 * .9v^*(1,2)} \end{array} \right\} \approx .45$$

Stochasticity Increases Complexity

Solving the system numerically we obtain the solved environment.

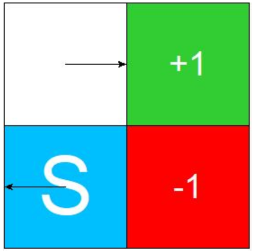


Figure: Introducing noise to the environment changes the optimal actions.

Optimality Equations Don't Scale

Even with a trivial environment, solving the system of equations can become complex. Consider the following Gridworld environment.

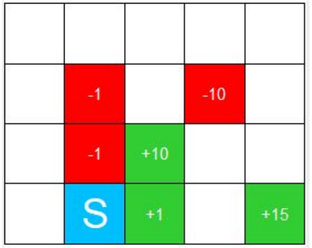
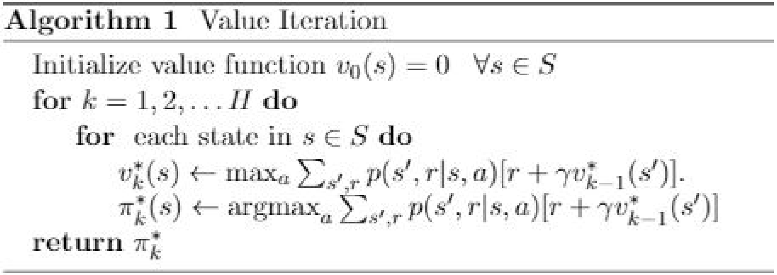


Figure: With 14 non-terminal states the complexity of the system increases substantially.

No closed form solution exists for the Bellman Optimality Equations.

Value Iteration Algorithm

$v_k^*(s)$ is total expected rewards starting from s, acting optimally for k steps.
$v_0^*(s) = 0 \;\; \forall s \in S$
$v_1^*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_0^*(s')]$



```
Algorithm 1 Value Iteration
Initialize value function $v_0(s) = 0 \;\; \forall s \in S$
for $k = 1, 2, \ldots H$ do
    for each state in $s \in S$ do
        $v_k^*(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_{k-1}^*(s')]$.
        $\pi_k^*(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_{k-1}^*(s')]$
return $\pi_k^*$
```

Figure: This algorithm has guaranteed convergence to $v^*(s)$ as $H \longrightarrow \infty$. In practice algorithm will break after specified tolerance is met.

Value Iteration Algorithm

Problems With Exact Methods

- Transition probability not always known.
- Even with transition probabilities, exact methods don't scale.
- With large state and actions spaces, tabular methods become very computationally expensive.
- Can't handle environment with continuous action or state spaces.

Need a way to approximate π which can handle large/ continuous state and actions spaces.

Deep Learning

Neural Networks

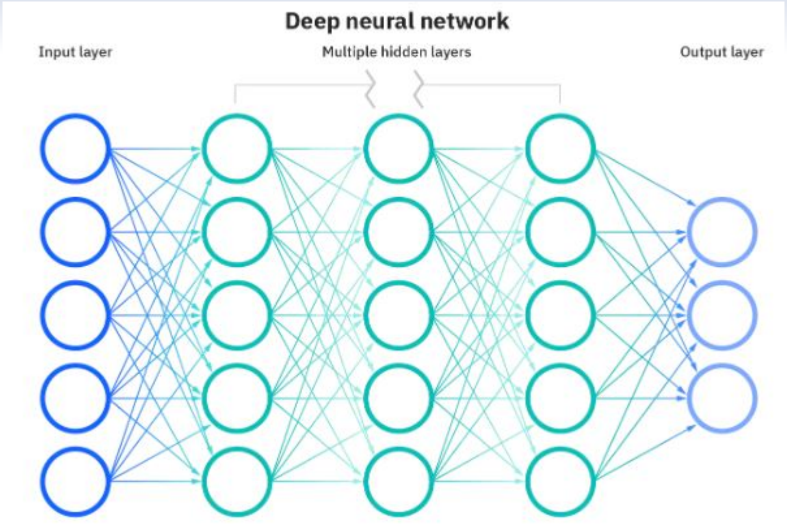


Figure: *Image from IBM Cloud.*

Neural Networks

- Neural networks are very flexible and can approximate complex nonlinear functions.
- Take in features (inputs) and apply functions between multiple layers.
- Number of outputs can be adjusted. Outputs can also be set to lie within specified values.
- Can calculate the gradient through back-propagation.
- Can handle large amounts of data as input.

Policy Approximation

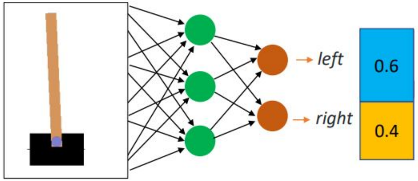


Figure: Neural Network taking in an image of the Cart-pole environment and mapping to an action probability distribution *(Figure by I. Karamouzas.)*

- The weights and biases of each component of the neural network are encompassed in θ. A policy parameterized by θ is denoted by either $\pi_\theta(a|s)$ or $\pi(a|s;\theta)$.

Objective function

Need a way to evaluate how well a parameter θ is performing. Introduce an objective function that will judge .

$$J(\theta) \doteq v_{\pi_\theta}(s_0).$$

Our goal becomes to maximize the objective function and to solve the optimization problem:

$$\max_\theta J(\theta).$$

Policy Gradient Theorem

Theorem

Assume that the policy $\pi_\theta(s, a)$ is differentiable. Then for objective function $J(\theta)$, the gradient is proportional to the state value function and the gradient of the policy:

$$\nabla J(\theta) \quad \propto \quad \sum_s \mu(s) \sum_a \nabla \pi_\theta(a|s) q(s, a).$$

Where $\mu(s)$ is the state distribution under π of an episode. In a continuous MDP we can simplify since $\sum_s \mu(s) = 1$.

$$
\begin{aligned}
\nabla J(\theta) &= \sum_a q_\pi(s,a)\nabla\pi(a|s;\theta) \\
&= \sum_a q_\pi(s,a)\pi(a|s;\theta)\frac{\nabla\pi(a|s;\theta)}{\pi(s|a;\theta)} && \text{Multiply and divide by } \pi. \\
&= \sum_a \pi(s|a;\theta) q_\pi(s,a)\nabla\log\pi(a|s;\theta) && \nabla log(f) = \frac{1}{f}\nabla f. \\
&= \boxed{\mathbb{E}_\pi\left[q_\pi(s,a)\nabla\log\pi(a|s;\theta)\right]}.
\end{aligned}
$$

Gradient Ascent

Having the gradient of the objective function allows us to implement the gradient ascent algorithm:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla J(\theta_t).$$

- Using gradient ascent we know that the policy will converge efficiently to a critical point of modest accuracy.
- Evaluating $\sum_{a \in A} \nabla_\theta \pi(a|s;\theta) q(s,a)$ is still expensive however. For this reason stochastic gradients need to be used.
- Different algorithms implement different formulations of the gradient but all rely on the Policy Gradient Theorem.

Algorithms

REINFORCE

- Follows directly from Policy Gradient Theorem.
- Based on Monte Carlo episode sampling. (Stochastic Gradient method)
- Collect episode, then calculate gradient and update after episode has terminated.
- Expectation of sample gradient is unbiased estimator of true gradient.

$$\begin{aligned}
\nabla J(\theta) &= \mathbb{E}[q(s,a)\nabla \ln \pi(a|s)] \\
&= \mathbb{E}[\mathbb{E}[G_t|s_t,a_t]\nabla \ln \pi(a_t|s_t)] && \text{Definition of q-function.} \\
&= \mathbb{E}[G_t \nabla \ln \pi_\theta(a_t|s_t)] && \text{Law of total Expectation.}
\end{aligned}$$

REINFORCE

Algorithm 2 REINFORCE

```
Initialize policy parameter $\theta$
Initialize learning rate and discount factor $\alpha, \gamma$
loop forever:
Generate an episode following policy $\pi_\theta : s_1, a_1, r_2, s_2, \ldots S_T$
for step in episode $t = 1, \ldots, T-1$ do
    $G_t \leftarrow$ expected return at step $t$.
    $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \log \pi(a_t, |s_t; \theta)$.
```

REINFORCE

Algorithm 2 REINFORCE

```
Initialize policy parameter $\theta$
Initialize learning rate and discount factor $\alpha, \gamma$
loop forever:
Generate an episode following policy $\pi_\theta : s_1, a_1, r_2, s_2, \ldots S_T$
for step in episode $t = 1, \ldots, T-1$ do
    $G_t \leftarrow$ expected return at step $t$.
    $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \log \pi(a_t, | s_t; \theta)$.
```

- High variance and very sample inefficient!

REINFORCE

Algorithm 2 REINFORCE

```
Initialize policy parameter $\theta$
Initialize learning rate and discount factor $\alpha, \gamma$
*loop forever*:
Generate an episode following policy $\pi_\theta : s_1, a_1, r_2, s_2, \dots S_T$
**for** step in episode $t = 1, \dots, T-1$ **do**
    $G_t \leftarrow$ expected return at step $t$.
    $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \log \pi(a_t, |s_t; \theta)$.
```

- High variance and very sample inefficient!
- **Goal:** Lower variance and increase efficiency.

Actor-Critic Methods

- To reduce variance a *critic* can be introduced. The critic can be based on the computation of estimated state values $v(s)$ or action state values $q(s, a)$ depending on the algorithm.
- One other component, the *actor*, updates the policy, π_θ based on feedback from the critic.

Actor-critic algorithms follow different variants of an approximate policy gradient given by:

$$\nabla J(\theta) \approx \mathbb{E}[\nabla \ln \pi_\theta(a|s) q_\phi(s, a)]$$

A3C

Asynchronous Advantage Actor Critic (A3C)

- Advantage function: $A(s, a) = q(s, a) - v(s) \approx G_t - v_\phi(s_t)$

A3C

Asynchronous Advantage Actor Critic (A3C)

- Advantage function: $A(s, a) = q(s, a) - v(s) \approx G_t - v_\phi(s_t)$
- Actor-Critic: $\nabla_\theta J(\theta) \approx \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s; \theta) A(s, a)]$

A3C

Asynchronous Advantage Actor Critic (A3C)

- Advantage function: $A(s, a) = q(s, a) - v(s) \approx G_t - v_\phi(s_t)$
- Actor-Critic: $\nabla_\theta J(\theta) \approx \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s; \theta) A(s, a)]$
- Asynchronous: Multiple agents working in parallel with each collecting experiences and updating parameters. This lowers correlation within samples used to learn.

A3C

Asynchronous Advantage Actor Critic (A3C)

- Advantage function: $A(s, a) = q(s, a) - v(s) \approx G_t - v_\phi(s_t)$
- Actor-Critic: $\nabla_\theta J(\theta) \approx \mathbb{E}_\pi[\nabla_\theta \ln \pi_\theta(a|s; \theta) A(s, a)]$
- Asynchronous: Multiple agents working in parallel with each collecting experiences and updating parameters. This lowers correlation within samples used to learn.

Asynchronous updates not needed. Better convergence and less variance with synchronous updates.

A2C

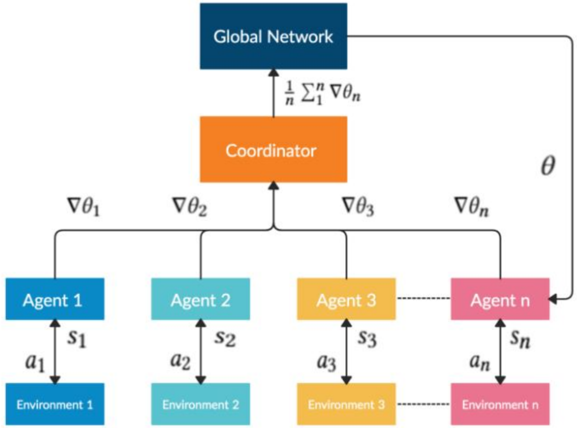
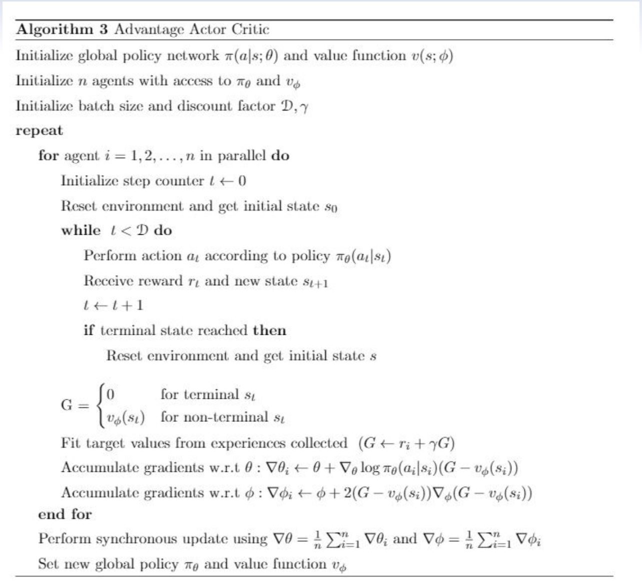


Figure: A2C Architecture (Figure by FINRL.)

A2C



Algorithm 3 Advantage Actor Critic

```
Initialize global policy network $\pi(a|s;\theta)$ and value function $v(s;\phi)$
Initialize $n$ agents with access to $\pi_\theta$ and $v_\phi$
Initialize batch size and discount factor $\mathcal{D}, \gamma$
repeat
    for agent $i = 1, 2, \ldots, n$ in parallel do
        Initialize step counter $t \leftarrow 0$
        Reset environment and get initial state $s_0$
        while $t < \mathcal{D}$ do
            Perform action $a_t$ according to policy $\pi_\theta(a_t|s_t)$
            Receive reward $r_t$ and new state $s_{t+1}$
            $t \leftarrow t + 1$
            if terminal state reached then
                Reset environment and get initial state $s$

        $G = \begin{cases} 0 & \text{for terminal } s_t \\ v_\phi(s_t) & \text{for non-terminal } s_t \end{cases}$
        Fit target values from experiences collected $(G \leftarrow r_i + \gamma G)$
        Accumulate gradients w.r.t $\theta : \nabla\theta_i \leftarrow \theta + \nabla_\theta \log \pi_\theta(a_i|s_i)(G - v_\phi(s_i))$
        Accumulate gradients w.r.t $\phi : \nabla\phi_i \leftarrow \phi + 2(G - v_\phi(s_i))\nabla_\phi(G - v_\phi(s_i))$
    end for
    Perform synchronous update using $\nabla\theta = \frac{1}{n}\sum_{i=1}^{n} \nabla\theta_i$ and $\nabla\phi = \frac{1}{n}\sum_{i=1}^{n} \nabla\phi_i$
    Set new global policy $\pi_\theta$ and value function $v_\phi$
```

Implementation

Implementation

- Python was used to implement for its extensive machine learning libraries. The package Pytorch is a Machine Learning framework with built in capabilities that include neural networks, optimizer functions, loss functions, and parallel processing.
- StableBaseLines 3 (SB3) is a set of implementations of RL algorithms along with image preprocessing and neural network architecture.
- Training was done on both the Palmetto Cluster and Google Colab.

PyTorch

Pong

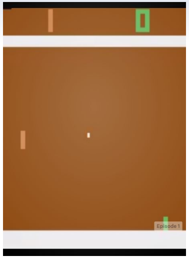


Figure: Atari 2600 Pong given by OpenAI gym.

- Inputs are images of the current game state.
- Agent has the actions *move up*, *move down*, *no action*.
- The agent receives a $+1$ reward for scoring against the computer controlled opponent and -1 reward for the computer scoring against the agent. All other rewards are zero.

Results

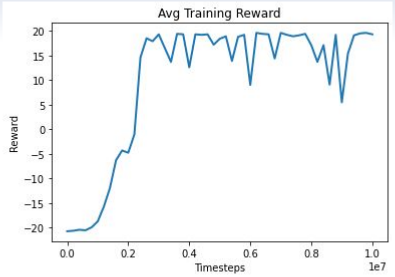


Figure: Pong A2C Learning curve with 16 parallel workers.

Mean Reward	Standard Deviation
20.00	$\pm$ 0.00

Table: Evaluation of trained A2C agent across 10 episodes

Pybullet

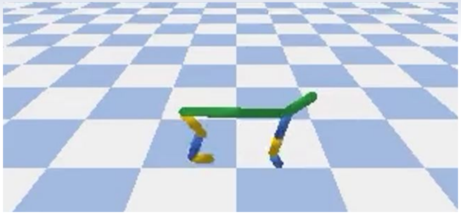


Figure: HalfCheetah pybullet environment.

- Pybullet is a physical simulation environment in which several different dynamical bodies can be used for optimal control tasks.
- The agent receives 26 observations describing the state.
- The action space is continuous since the agent must indicate how much torque to apply to each limb. The expected range for each of the six sub-actions is between [-1,1].

Continuous Action Space

Policy neural networks shown before map states to discrete actions. How do we handle continuous action spaces?

Continuous Action Space

Policy neural networks shown before map states to discrete actions. How do we handle continuous action spaces?

- Neural network can output values in a continuous interval.

Continuous Action Space

Policy neural networks shown before map states to discrete actions. How do we handle continuous action spaces?

- Neural network can output values in a continuous interval.
- This becomes a deterministic policy. We still want a stochastic policy for exploration!

Continuous Action Space

Policy neural networks shown before map states to discrete actions. How do we handle continuous action spaces?

- Neural network can output values in a continuous interval.
- This becomes a deterministic policy. We still want a stochastic policy for exploration!
- Use output of policy network as the mean(s) for a Gaussian distribution. Then sample from distribution. $a_t \sim \mathcal{N}(\mu_t, \sigma^2 \mathcal{I}), \mu \in \mathbb{R}^n, \sigma \in \mathbb{R}^n_+$.

Continuous Action Space

Policy neural networks shown before map states to discrete actions. How do we handle continuous action spaces?

- Neural network can output values in a continuous interval.
- This becomes a deterministic policy. We still want a stochastic policy for exploration!
- Use output of policy network as the mean(s) for a Gaussian distribution. Then sample from distribution. $a_t \sim \mathcal{N}(\mu_t, \sigma^2 \mathcal{I}), \mu \in \mathbb{R}^n, \sigma \in \mathbb{R}_+^n$.
- Sigma can be a fixed, learned, or scheduled hyperparameter.

Results

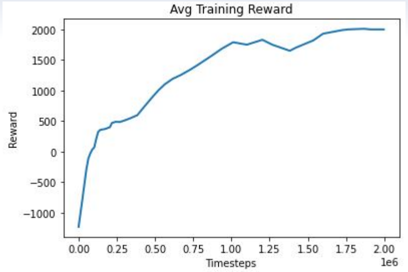


Figure: HalfCheetah A2C learning curve with 4 parallel workers.

Mean Reward	Standard Deviation
2001	$\pm$ 67.3

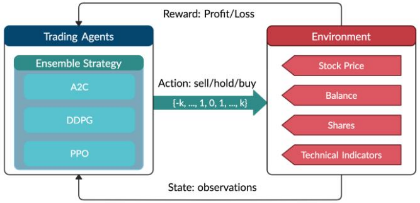
Table: Evaluation of trained A2C agent across 10 episodes

Analysis of Results

- RL requires large number of time steps to learn.
- Environments can be very sensitive to hyperparameters and algorithm choice.
- Improvements to GPU/CPU technology as well as deep learning will mitigate this.

Conclusion

- Through the Policy Gradient Theorem there is a class of method RL focusing on policy optimization involving the gradient.
- Better equipped to develop truly stochastic policies and can incorporate exploration into action selection.
- Current research and applications include multi-agent RL, imitation learning, and much more.



(a) Multi-agent RL stock trading from Yang et. al. 2019.

(b) Deep Mimic Animation

Questions