Continues Control

Shreyas Chandra Sekhar

Abstract—Unity Reactor Environment was considered to study and establish the complete control over 20 agents in the environment. The double-joined armed were taught to move to a target location and stay there for as many time steps as possible. DDBP an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces was used to train the agents. A mean score of 30+ was attained for the last 100 consecutive episodes.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

Goal of artificial intelligence is to solve complex high dimensional problems with accuracy and with minimal learning curve. The Deep Q Network(DQN) is capable of human level performance on Aatari games which are of low dimensions. However, DQN can only perform discrete action on high spaces. They cannot be applied directly on continues spaces as depends on actions that can maximize the rewards of action value function. To avoid this short fall a continues action space could be though as a solution. However, this lead to an exponential increase of action space with degree of freedom. DQN seems not so efficient under these circumstance. The discretion of DQN throws away information about the structure of the action domain which can be used to increasing the efficiency of convergence.

This project was based on model-free, off-policy actor-critic algorithm that can learn high dimensional continues action-space. This uses deterministic policy gradient. Deep DQN uses both value and action based approaches to handle continues space environment and converges at a faster rate.

2 Background

If a Deep Reinforcement Learning agent uses a deep neural network to approximate a value function, the agent is said to be value based. It can be used to approximate to learn about state value function Vpi(s), the action value function Qpi(s,a), the advantage function Api(s,a) and the optimal versions of these. A policy is derived from it. If a Deep Reinforcement Learning agent uses a deep neural network to approximate a policy, the agent is said to be policy based. The agents optimize its policy and learns from it directly. The policy is usually sarcastic pi(a/s) but it can also be used to learn deterministic pi(s).

Actor-Critic method uses both Value-base and Policy-based methods. It uses the base line to reduce the variance of policy based agent. Value function can be used as a baseline. Train a neural network to approximate a value function and use it as a baseline to create a better baseline. This better baseline would further reduce the variance of policy based method

Actor-Critic falls in-between Value-base and policy base methods. They use value based techniques to reduce the variance of policy based methods.

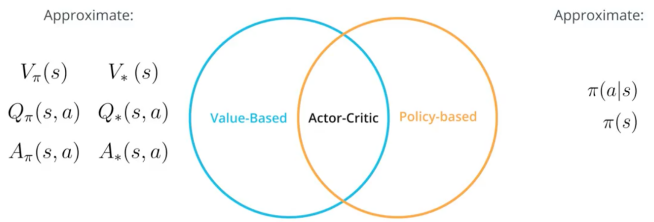

Fig. 1. Actor-Critic

[1]

2.1 Bias Vs Variance

In machine learning, there is always a trade off between bias and variance. We would like to have low bias and low variance to have a predictable model. When a function try to estimate the value functions or policies from returns, the return is calculated using single trajectory. However, the value function which we are trying to estimate are calculated using the expected returns. As we are limited to sampling the environment we can only estimate the expected returns.

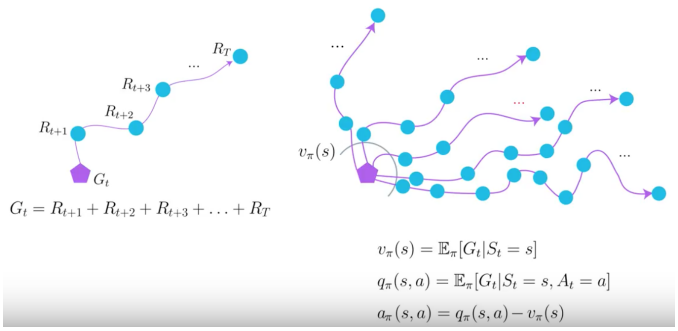

Fig. 2. Bias Vs Variance

[1]

2.1.1 Monte-Carlo Estimates(MC)

Monte-Carlo estimate consist of rolling out an episode and calculating the discounted total reward from the reward sequence. It adds all the rewards for t steps. We get the collection of episodes (A, B, C and D). Some of the episodes will go through the same state and give different Monte-Carlo estimate for the same value function as they are sarcastic. The estimates are averages to calculate the value function. The more value function better would be the estimate.

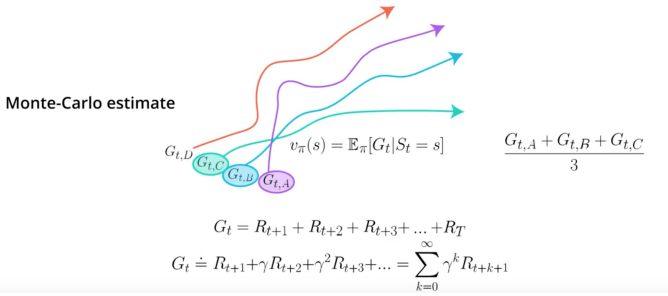

Fig. 3. Bias Vs Variance

[1]

2.1.2 Temporal Difference estimate(TD)

Say we are estimating state value function, estimating the value of the current state uses a single reward sample and the estimated total return, the agent will obtain from the next state on wards. So that's estimating with the estimate. This done by boot strapping using dynamic programming. It means that you can leverage the estimate you currently have for the next state in order to calculate the new estimate for the value function of the current state. The estimate of the next state will be off early on but this gets better and better as the agent see more data. This makes the all the dependent values better. This will make the value function better after many iterations. [1]

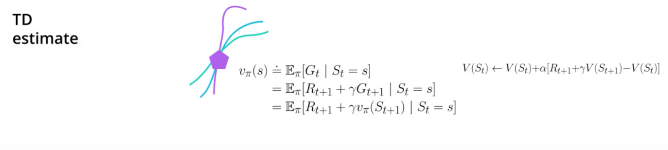

Fig. 4. Temporal Difference estimate

2.1.3 MC Vs (TD)

Monte-Carlo agents will have high variance because estimates for the state can vary greatly across episodes. This is because it compounds lots of random events that happens with course of single episode. However, Monte-Carlo agents are unbiased as it is not estimate using estimates. It only uses try rewards that are obtained. So given lot of data, the estimates would be accurate.

TD estimates are low variance as it compounds single step of randomness instead of full roll out. As its boot stopped on next state estimate which are not true values it adds bias. The agent will learn faster but will have problems converging. This algorithm involves three techniques.

TABLE 1
MC vs TD

Agent	Bias	Variance
Monte Carlo	low	high
Temporal Difference	high	low

In reinforcement learning is calculated as total discounted return. This way is same way as in MC approach which has high variance. A base line is used to reduce the variance of the reinforce algorithm. However, this baseline was also calculated by using MC approach. If we use deep learning to learn this baseline, function approximation can be used to gain generalization. This means when there is a new state, deep neural network will come up with better estimates as its been trained using similar data.

TD estimates are used to train baseline and this acts as a critic. This introduces bias, but reducing variance and improving convergence problem and speeding up learning.

2.1.4 Actor - Critic Approach)

Actor-Critic methods is to reduce high variance in policy based agents. By using TD critic instead of MC baseline, the variance is further reduce the variance of policy base methods. This leads to faster learning by using just policy based agents alone and see better consistent convergence then value based agents alone. In a policy based approach, the agent is leaning to act. In a value based approach the agent is learning to estimate.

2.1.5 Actor - Critic Agent)

An Actor - Critic method uses function approximation uses a policy and a value function. It uses two neural networks. The Critic will learn to evaluate the State-Value function Vpi using TD estimate. Using critical Advantage function is calculate and train the actor using this value.

Actor takes in a state outputs distribution of actions. The other network Critic, takes in a state and outputs a state value function of policy Vpi. Input the current state into the actor and get the action to take in that state. Observe next state and reward to get experience double (s,a, r, s). Then using TD estimate, which is reward r+ critics estimate for s (r + gamma* V(s; thetav) train the critic.

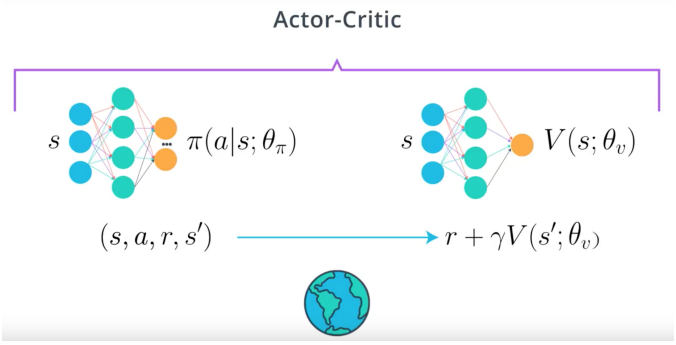

Fig. 5. Temporal Difference estimate

[1]

To calculate Advantage function, we also use the critic. We train the Actor using the calculated Advantage as a baseline. [1]

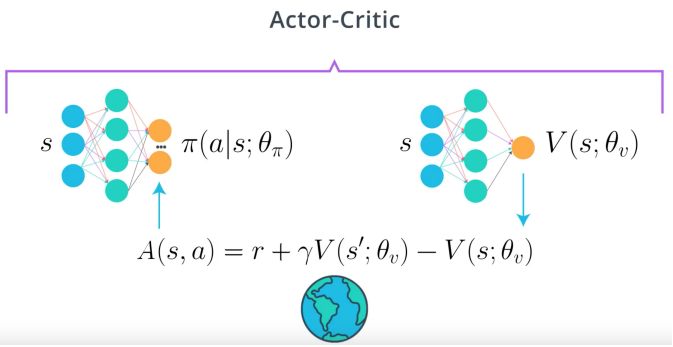

Fig. 6. Temporal Difference estimate

2.1.6 DDPG Internals)

The Critic in DDPG is used to approximate the maximizer over the Q values of next state and not a learned baseline. One of the limitation of a DQN agent is that it is not straight forward to use in continues action spaces. Finding the Max value in a discrete action space can be simple even if we have many actions. However this this not possible for continues space.

DDPG solve this problem. DDPG uses two neural networks, Actor and Critic. Actor is used to approximate the optimal policy deterministically. This is output just the best believed action for any given state. The actor is learning the argmaxaQ(s, a) which is the best action. The Critic learns to evaluate optimal action value function by using the actors best believed action. Actor is used, which is an approximate maximizer to calculate the new target value for training the action value function.

2.1.7 DDPG Algorithm)

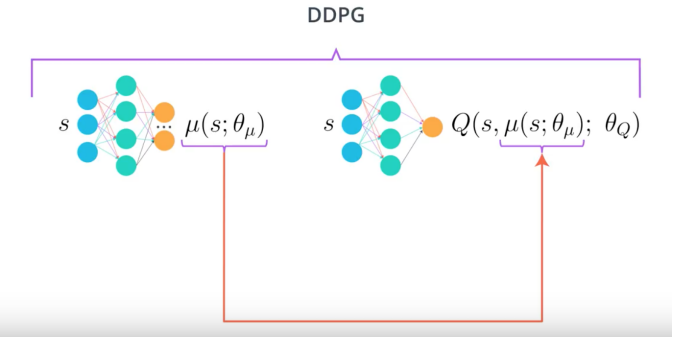

Fig. 7. DDPG Algorithm

[1]

DDPG uses reply buffer and soft updates to target networks. In the DQN we have two copies of the network weights, the Regular and Target network. Both Actor and Critic has Regular and Target networks. Target network are updated using soft update strategy. It consists of slowly blending the regular weights to target network every step. This makes the target network is 99.99 percent of this own just a 0.01 percent of regular network weight. This slowly mixes the regular network weight to target. Regular network is the most up to date network that is being trained, while the target network is the one used for prediction to stabilize. This makes the convergence faster.

2.1.8 DDPG Algorithm)

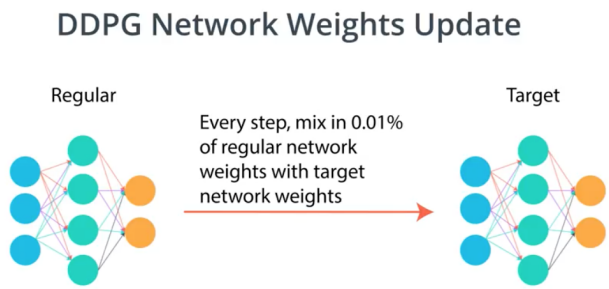

Fig. 8. DDPG Algorithm

[1]

2.1.9 DDPG Algorithm)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Fig. 9. DDPG Algorithm

[1]

2.1.10 Environment)

Unity Reactor environment was used for this experiment.

- Set-up: Double-jointed arm which can move to target locations.
- Goal: The agents must move it's hand to the goal location, and keep it there.
- Agents: The environment contains 20 agent linked to a single Brain.
- Agent Reward Function (independent): +0.1 Each step agent's hand is in goal location.
- Brains: One Brain with the following observation or action space.
- Vector Observation space: 26 variables corresponding to position, rotation, velocity, and angular velocities of the two arm Rigidbodies.

- Vector Action space: (Continuous) Size of 4, corresponding to torque applicable to two joints.
- Visual Observations: None.
- Reset Parameters: Two, corresponding to goal size, and goal movement speed.
- Benchmark Mean Reward: 30

3 Results

There are many tuning parameters in the algorithms. There were many attempts made for tuning and below are results

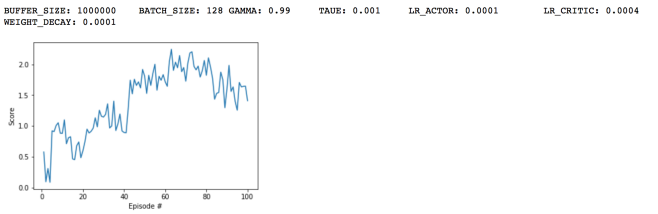

Fig. 10. Trail

[1]

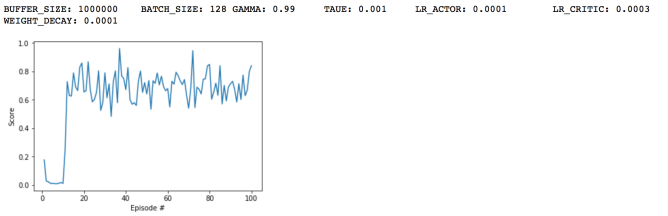

Fig. 11. Trail

[1] [1] [1]

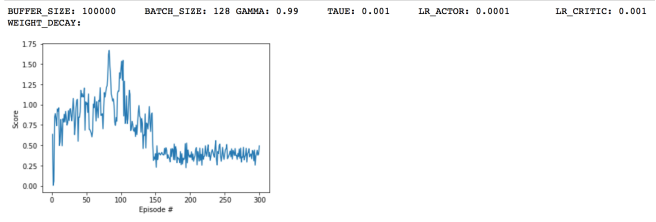

Fig. 12. Trail

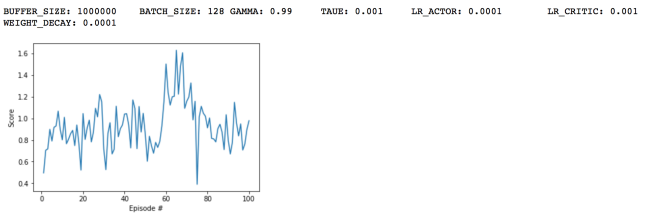

Fig. 13. Trail

[1]
[1]

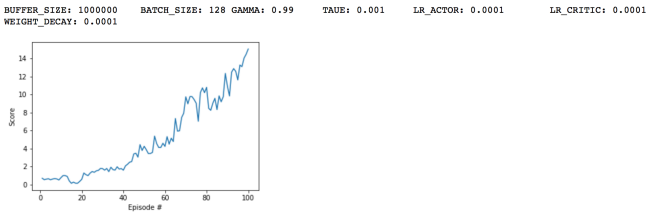

Fig. 14. Trail

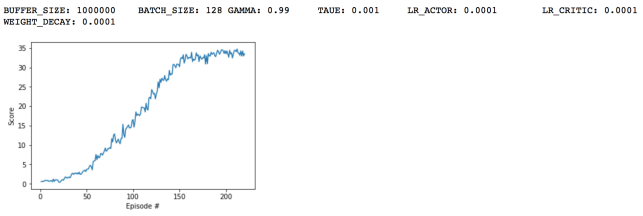

Fig. 15. Trail

4 Discussion

There are many hyper parameters to tune.

- BUFFER_SIZE = int(1e5) replay buffer size
- BATCH_SIZE = 128 minibatch size
- GAMMA = 0.99 discount factor
- TAU = 1e-3 for soft update of target parameters
- LR_ACTOR = 1e-4 learning rate of the actor
- LR_CRITIC = variable learning rate of the critic
- WEIGHT_DECAY = 0.0001 L2 weight decay

Most of these values here were taken from the reference paper. There were many trials with different tuning values.

Buffer size was always kept at the mentioned value. Many trials of Batch size were tried. This did not show major impact on the result. Gamma was always kept at 0.99. TAU was taken from the reference paper and so Weight Decay. The learning rate of Actor and Critic showed huge impact on the results. Having them below 0.0001 or above 0.0002 showed very slow learning. The sweet shot was in between 0.0001 to 0.0002. There were many trials between these numbers finally the required score of 30+ could be achieved with learning rate of 0.0001 for both actor and critic after 190 trails.

5 Conclusion / Future work

All the 20 independent double armed agent were able to be tough with an average scope of 30+ for the last 100 episodes using DDPG algorithm. It took many hours to experiment and finally come to this tuning parameters. The results were very sensitive to learning rate.

There are many other tuning parameters to be explored which includes batch size, Actor learning rate, decay rate. This would be explored and considered as part of future work.

rtab database file is located in result folder of the repository

REFERENCES

[1] A. P. N. H. T. E. Y. T. D. S. . D. W. Timothy P. Lillicrap, Jonathan J. Hunt, *LATEX: CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING*. Google Deepmind, 2016.