Reforcement Learning Alogorithms' Comparison

First Author
ZitianChen
153071301 91@fudan.edu.cn

Abstract

Deep Reinforcement Learning has yielded proficient controllers for complex tasks.Many algorithm was proposed and their concepts enlightened us to discover more efficient methods.We introduce the popular algorithms (DQN,DDQN,Dueling Network,A3C) and compare their efficiencies ,advantages and disadvantages.

1. Introduction

Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL).Based on the abstract concept coming from the Monte-Carlo, TD-LearningDynamic Programming,Convolutional network and Asynchronous method, new methods outperform the state-of-the-art on the Atari 2600 domain continuously.

1.1. DQN(Deep Q-Networks)

DQN is the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning,whose input is raw pixels and whose output is a value function estimating future rewards.Instead of manually picked features,DQN have high efficiency benifited from it's deep network.

$$L_i(\theta_i) = E_{s,a,r,s'}[(y_i^{DQN} - Q(s,a;\theta_i))^2] \tag{1}$$

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s',a';\theta^-) \tag{2}$$

1.2. DDQN(Double Deep Q-networks)

In the Double Q-learning algorithm, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, θ and θ' . For each update, one set of weights is used to determine the greedy policy and the other to determine its value.

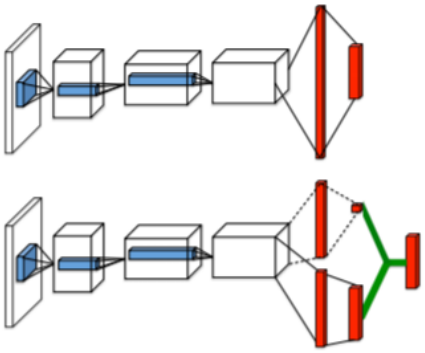

Figure 1. Dueling network's architecture

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1},a;\theta_t)); \theta_t'). \tag{3}$$

This approch help reduce the DQN's overestimation issue.

1.3. Dueling network

Dueling network represents two separate estimators: one for the state value function and one for the state-dependent action advantage function. This architecture leads to better policy evaluation in the presence of many similar-valued actions.The key insight is it is unnecessary to estimate the value of each action choice in many states.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)) \tag{4}$$

1.4. Bootstrapped DQN

Bootstrapped DQN use k heads to produce k Q-value function:$Q_k(s,a;\theta)$. We randomly choose a Q-value function and follow the policy according to this Q-value function and generate replay experience. Then each Q-value function $Q_k(s,a;\theta)$is trained against its own tar-

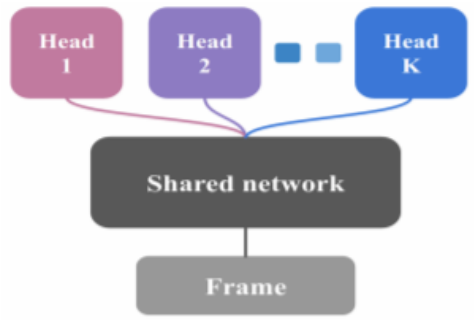

Figure 2. Bootstrapped neural nets

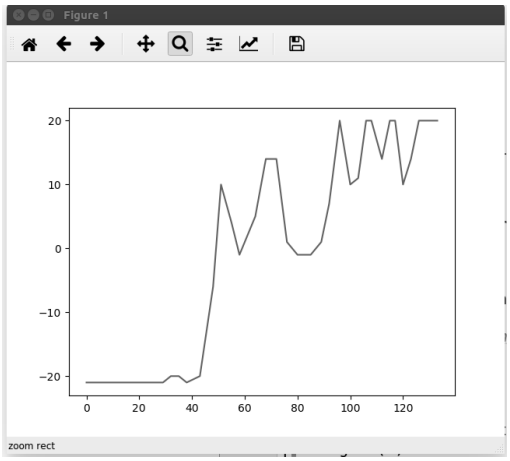

Figure 3. A3C's performance on Pong

get network $Q_k(s, a; \theta_-)$. By randomly initialize the network,Bootstrapped DQN implement a very deep exploration algorithm.

My intuition behind this method is imaging a 3-D world with many bowl-shape holes.DQN can easily get stuck in local minimum.But bootstrapped DQN have k start points and when a Q-value function take a little step to it's local minimum,all Q-value function take a tiny step toward it's direction.So other Q-value functions is able to went out their local minimum by the exploration took by one Q-value function.

2. A3C's Implement

I use pytorch to implement Asynchronous Methods for deep reinforcement learning on Atari 2600. This method can train deep neural network policies reliably and without large resource requirements. First I just write the code according to the pseudocode. Then I find that it doesn't work.The reason is the loss function which I will talk later. After changing the loss function, it took my 4 cores CPU laptop only 2 hours to reach the high scores on Pong and also behave well on Breakout.

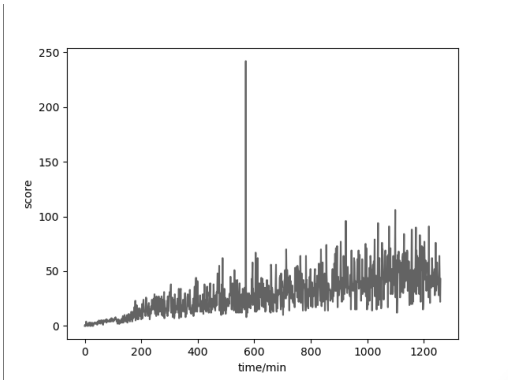

Figure 4. A3C's performance on Breakout

2.1. Architechure

Firstly,we change the input RGB channels image to gray channels and resize it by 42*42. So the input of the neural network is 1*42*42. Then the input forward pass 4 convolutional layers with 3*3 kernel,stride 2,padding 1 and 32 output channels. So then the data compress to 3*32*32. After that we feed it into the LSTM network with 128 hidden neurons. Finally use two independent fully-connected layer we got the value estimation and the action value estimation.

2.2. Loss Function

$$R = \lambda R + r_i \tag{5}$$

For the n'th step:

$$R = V(s_t; \theta'_v) \tag{6}$$

Value loss:

$$Loss_{value} = (R - V(s_i; \theta'_v))^2 \tag{7}$$

For the policy loss,first I try the way in the pseudocode given by DeepMind. Constant X:

$$X = R - V(s_i; \theta'_v) \tag{8}$$

Policy loss:

$$Loss_{policy} = X * log(\pi(a_i|s_i; \theta') \tag{9}$$

This constant X looks like the TD-error which mean it may suffer high various. I figure out that maybe the model is too sensitive that it can't learn anything(In the pong the AI just stay or move to the corner without bouncing the ball). By using a pretrained model,the algorithm make it behave worse with a high variance and changing the learning rate is not help. So I change the constant X:

$$X = \gamma X + (r_i + \gamma V(s_{i+1}; \theta'_v) - V(s_i; \theta'_v)) \tag{10}$$

I think the advantage may be influenced the behavior a few frames ago so I plus γX And $\gamma V(s_{i+1};\theta'_v) - V(s_i;\theta'_v)$ is the based content of Actor-Critic method. Actor critic methods combine the best of both worlds. Employing an actor (policy gradient update) with a good critic (for example the advantage function, Q(s, a) - V(s)). This allows actor critic to be more sample efficient via TD updates at every step. With this loss function the AI use 2h to reach a stable and outstanding performance in Pong.

2.3. Hyper-parameters

I have tried to add the entrophy loss and some weight initialization methods but it does't improve the performance significantly.

I tried to replace the LSTM with a fully connected layer to get a small model which can train fast.But it turns out that this small model learned nothing(stay or keep moving left or right) in 5 hours' training.Also,changed the learning rate of the big model from 0.0001 to 0.001 leads to the same status.This two experiments reflected that the reinforcement learning model is very sensitive.Maybe I change more hyper-parameters then the small model will work.

2.4. Implementing Details

I noticed that sometimes the model will stuck in some local minimum.In the game Pong,when my AI keep losing and luckily reach a statue that AI and the opponents just bouncing the ball in the same trajectory every round.According to my settings,the agent receive a 0:0 score which means the reward is 0 better than -21.Then AI leaned this strategy and just stuck in this statue – changed a little and received -21 then changed back and received 0.It never walked out.So I just write one line of condition code to avoid this situation.

In Breakout,the AI can't figure out start button by it's own so i can't used maximum policy due to it may stuck when it loss one chance.I should choose a stochastic policy.

I used the Adam Optimizer and 8 threads,set the learning rate to 0.0001,γ in loss to 0.99,γ in optimizer to default.I prefer elu to relu due to its smoother curve.

I have tried some batch normalization,weight initialization and some ways to limited the gradient range.But they do not improve the training time obviously so I just delete them for neat.

3. Comparison

A3C is on-policy method and DQN,DDQN,Dueling networks are off-policy method.On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data.So A3C is convenient due to it can train online.But this lead to inefficient

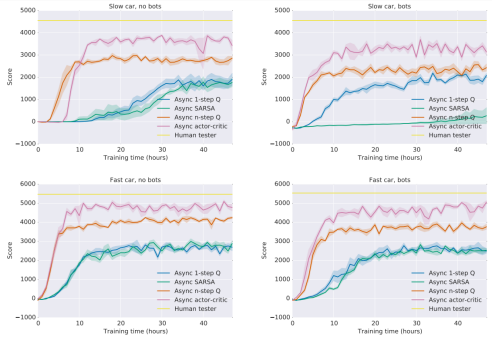

Figure 5. Different algorithn on the TORCS car racing simulator

in some aspect,due to the due to the strong correlations between the samples.So in some situation it only learn a part of the enviourment.With experience replay and prioritized experience replay,off-policy methods allows for greater data efficiency.

Q-learning is valued based method and DQN,DDQN,are action-value based method.Value based methods such as Q learning suffer from poor convergence, as you are working in value space and a slight change in your value estimate can push you around quite substantially in policy space.Policy based methods, both policy search and gradient, work directly in policy space and have much smoother learning curves and performance improvement guarantees with every update the drawback with this method is that it tends to converge to local maximas (in policy based methods you are maximizing the expected reward by searching in the policy space) and suffer from high variance, sample inefficiency.

Actor critic methods combine the best of both worlds. Employing an actor (policy gradient update) with a good critic (for example the advantage function, Q(s, a) - V(s)). This allows actor critic to be more sample efficient via TD updates at every step.

In memory consumption,all algorithms behave similar due to shared deep network.But in computational consumption,Asynchronous Methods used multiple threads limited by CPU so it beat GPU-based methods.

In final performance and training time,A3C defeats all other methods obviously.Dueling networks behave well on some certain game than DDQN and DQN.DDQN decrease the overestimation problem so it behave better than DQN.

In sensitivity on hyper-parameters,I think DQN is the worst due to its overestimation issue.Intuitively,I think A3C and Dueling network is more stable.Because they combine action-value and value.Beside,DQN and DDQN use max to bootstrapped while A3C and Dueling network use average.So they are unlikely suffer from unstable caused by small change on policy.But even A3C is still highly sensitive.I think the reason is they work on high-dimensional space.Little change in any dimension is critical.