Assault (Atari 2600): which is the best agent?

Alessandro Pavesi
Department of Computer Science and Engineering - DISI
Alma Mater Studiorum
Bologna, Italy
alessandro.pavesi2@studio.unibo.it

Abstract

The algorithms introduced in Reinforcement Learning have substantial differences in which application could be used and in how they are applied. To better understand these differences I've used a unique environment to compare different agents, to analyze the different choices of implementations, and to find the best model applied to the Assault games from Atari 2600.

1 Introduction

Reinforcement Learning is one of the most exciting branches of Artificial Intelligence and in particular, in my personal opinion, the most fascinating thing is that the approaches and the applications are substantially different from Machine Learning and Deep Learning techniques. The approach of RL is more similar to how humans learn and the applications to these techniques can help to reach a General AI. The main focus of this project is to discuss how different methods learn to play Assault, a game of the Atari 2600. The neural network used is a Convolutional Neural Network that differs only on the output layers that need to be adapted to the various technologies used. The RL algorithms implied in are the Q-Learning through a Deep Q-Network and the Actor-Critic methods.

2 Background

2.1 OpenAI Gym

The researcher of OpenAI provides a toolkit where everyone can test their algorithms, more important they provide a huge number of environments ready to use with a specific API. The environments vary from control theory problems to algorithmic problems, continuous control problems, and Atari 2600 games. The environment chosen is Assault, a 2D game where the player needs to control *a spaceship fixated at the bottom of the screen, the gameplay involves the player shooting projectiles towards an enemy mothership that deploys smaller ships to attack the player. The player must also prevent enough projectiles from touching the bottom of the screen.*

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

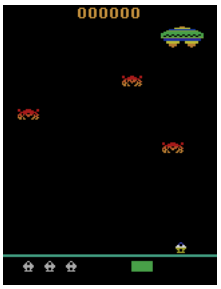

Figure 1: Frame of Assault

The game of Assault could be defined as a Markov Decision Process. An MDP is a discrete-time stochastic control process, in which at each time step, the process is in some state S and the decision-maker may choose any action A that is available in state S. The process responds at the next time step by randomly moving into a new state s_{t+1}, and giving the decision-maker a corresponding reward $R_a(s_t, s_{t+1})$.

2.2 Q-Learning

One of the most important algorithm of RL is Q-Learning, an Off-Policy TD control algorithm. The simplest form is defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

This formula it's used to learn an action-value function Q that approximates the optimal $Q*$ action-value function. In the simplest application, we could build a matrix where each cell represents the value for a pair (State, Action). This approach could be applied to simple problems but for the aim of this project there's a need for something more complex that can handle a large number of inputs and states. I used a CNN to approximate the action-value function using the formula above to update the weights of the neural network. The training of the CNN is based on the data provided by the playing agent, the idea is to learn step by step, after each move, an update for the Q-values.

The policy followed to act in-game is the one usually applied with Q-learning: $\epsilon - greedy$. The $\epsilon - greedy$ policy uses a value ϵ to indicate a probability to explore instead of exploit.

The Q-learning using neural networks are more unstable and the training time could become too much, moreover, the convergence to the optimal value function isn't guaranteed. To use the potential of neural network as a function approximator some modifications need to be done.

Target Network Using the naive Q-learning algorithm it's possible to slip into the *maximization bias* problem. This problem is intrinsic in the algorithm because using the *argmax* the estimated values could reach bias. To solve it I decided to use a second Q-Network, called *target network*; it is equal to the first (called *q-network* but not trained in the same way and it's used during the training of the q-network to estimate the values of the next states. This approach proposed by Mnih *et al.*[1] aligns the target network after a defined number of episodes with the weights of the q-network to reduce the problem of overestimated values.

The final formula becomes:

$$Q(S_t, A_t; \theta) \leftarrow Q(S_t, A_t; \theta) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \omega) - Q(S_t, A_t; \theta)]$$

Experience Replay The training of the agent is trying to approximate a complex, nonlinear function, Q(s, a), with a Neural Network. However, one of the fundamental requirements for the Gradient Descent algorithm is that the training data needs to be independent and identically distributed but the naive Q-Learning algorithm uses to do a step of training using the data of the last episode, thus the sequence of experience tuples are highly correlated.

To prevent action values from oscillating or diverging it's possible to use a large buffer to contain the experience of the agent and sample training data from it randomly. This technique is called *replay buffer or experience buffer*. The replay buffer contains a collection of experience tuples

(S_t, A_t, R_t, S_{t+1}). The tuples are gradually added to the buffer as the agent interacts with the Environment.

2.3 Actor-Critic

The Actor-Critic method approximates directly the policy to use in-game and the state-value function. The Actor is the one that estimates the policy and the critic is the one that estimates the state-value function. This temporal difference method updates the weight of the NN after each step, as the DQN, but here we always follow the policy given by the network so it's an On-Policy method. The policy is updated using the formula :

$$\delta_t \leftarrow r_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)$$
$$\theta_{t+1} \leftarrow \theta_t + \alpha \delta_t \nabla \ln \pi(A_t|S_t, \theta)$$

3 Methods

Assault The goal of Assault is to eliminate waves of alien spacecraft moving above the agent using a laser cannon that can fire in horizontal and vertical. The reward process is relatively sparse, a reward of +21 is given by the environment every time the agents hit an alien, no reward are given when the alien hit the ground or when the agent lose a life, thus the reward is usually 0. The OpenAI Gym environment returns the game screen, the same visualized in the Atari 2600 console, as 210 by 160 RGB matrix. It's necessary to pre-process the image to be used with the neural network. The pre-process starts with a crop of the header, which inside have the score, useless for the agent, to have an image of 200 by 160. Then it's applied transformation to make the image in gray-scale. At the end of the process, the final dimensions are 200x160x1.

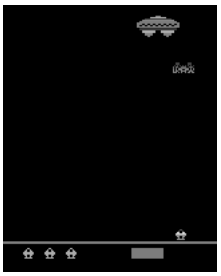

Figure 2: Pre-processed frame

The game has some parts that move during the game, like the spacecrafts or the alien's laser fired to the player that needs to be avoided, those parts require the agent "to see" the movement. To solve this problem the idea comes from Mnih *et al.*[1], suggesting to stack 4 game frames. Using this solution the neural network has the minimum amount of information to try to avoid the laser so as to lose a life.

Deep Q-Network The network developed at the base of the Q-learning method has initially two Convolutional layers with different parameters: the first one has 16 filters of size 8 with stride 4 and padding, the second one has 32 filters with size 4 with stride 2. Both have Rectified Linear Unit as an activation. Then a 256 units Dense layer takes the output of the previous with ReLU as activation and after the last Dense layer with several units the number of actions needed to play. This layer hasn't an activation function. The architecture is similar to van Hasselt *et al.*[2]. In case of Assault the actions are 7 and the means of each is: *['NOOP', 'FIRE', 'UP', 'RIGHT', 'LEFT', 'RIGHTFIRE', 'LEFTFIRE']*. I wanted to let the network focus on the important actions leaving all the actions selectable.

The Deep Q-Network is a simple neural network, the main part of this project is the implementation of the Q-learning algorithm.

Algorithm 1 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $epsilon > 0$
Initialize $Q(s, a; \theta)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
Initialize $Q(s, a; \omega)$ equal to $Q(s, a; \theta)$
foreach *episode* **do**
  Initialize S
  foreach *step of episode* **do**
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S_t, A_t; \theta) \leftarrow Q(S_t, A_t; \theta) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \omega) - Q(S_t, A_t; \theta)]$
    $S \leftarrow S_{t+1}$
  end foreach
end foreach

The policy used is an $\epsilon - greedy$ policy with an ϵ that decreases over time, initially high (0.5) to explore the largest number of moves and lower at the end (0.01) to follow the policy given by the Q-values.

Actor Critic The Actor-Critic method is implemented using a single network to predict the state-value and the other to estimate the policy. The network is nearly the same as the Q-learning method above, based on two Convolutional layers and a single 128 units Dense layer. Then the two head using one Dense layers each, the first one with a single unit that estimates the state-value and one with as units the number of possible action defined by the game (7). The former has no activation function, the latter uses a softmax as activation to outputs probabilities.

Algorithm 2 Actor-Critic for estimating $\pi \approx \pi^*$

Input: differentiable policy parametrisation $\pi(a|s, \theta)$, a differentiable state-value function parametrisation $\hat{v}(s, \mathbf{w})$
Algorithm parameters: step size $\alpha^\theta > 0$ and $\alpha^w > 0$
Initialize policy parameters $\theta \in^d$ and state-value weights W $\in^d$
foreach *episode* **do**
  Initialize S
  foreach *S is not terminal* **do**
    Choose A from S using policy π
    Take action A, observe R, S'
    $\delta \leftarrow R + \hat{v}(S', \mathbf{w}) - \hat{v}(s, \mathbf{w})$ ▷ if S' is terminal then $\hat{v}(s, \mathbf{w})$=0
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(s, \mathbf{w})$
    $\theta \leftarrow \theta + \alpha^\theta \delta \nabla \ln \pi(A|S, \theta)$
  end foreach
end foreach

4 Results

Before comparing directly the two methods it's necessary to let the agents play to reach consistent rewards. Here I analyze the performance of the two methods independently. The training was done using Google Colaboratory and Amazon SageMaker. The evaluation procedure is to let train the agent during the play, then after a defined number of episode evaluate it making him play the game without training at the same time in case of A2C and with a greedy policy for the DQN. To evaluate properly the agents in terms of reward and actions a Random Agent is taken as a baseline, the mean reward is around 80 points, instead the actions have the same probability so every action is used an equal number of times.

4.1 DQN

The training for the DQN involves playing as many episodes as possible, where each game has a range of timestep from 500 to 3500 based on if the agents die or not. I've made a comparison with an

initial agent trained with only 600 games and the best agent with 1300 games played. The statistics are collected after the training using a greedy policy.

To see if the agent learns a strategy for play is possible to compare how much each action is chosen.

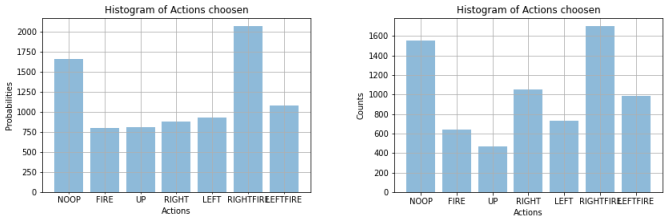

Figure 3: Bar plot of actions chosen in 10 games from different agents

Notably, the differences of strategy, the agent with less experience is more likely to use all the actions with the same probability, instead, the experienced agent uses more the fire and fire-and-move actions. In particular seems that the experienced prefer to use

Another comparison parameter is the episode reward, so the point made by the agent during the play, including all 3 lives.

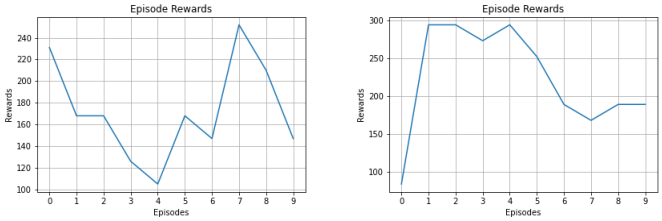

Figure 4: Plot of episode rewards in 100 games from different agents

Also in this case the difference is important, the experienced agent learned how to play and is constantly better than the novice agent.

4.2 A2C

The Actor-Critic learner is more difficult to train, it takes longer also because the training is done during each timestep of the game, and a policy network changes slightly the parameter so to reach consistent results needs more time. The number of training episodes is much less than the DQN training, 300 episodes are made due to the above problems and also for some external problems (time limit of Colab). The first comparison is made as before with the action chosen and the episode rewards:

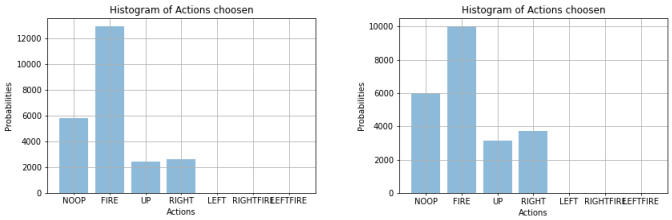

Figure 5: Bar plot of actions chosen in 10 games from different agents

The first figure shows that after only 150 episode the agents learns that the button Fire is the most important, but it doesn't use the buttons for moving and firing at the same time. After 300 episode the actions chosen by the agents are not so different but the

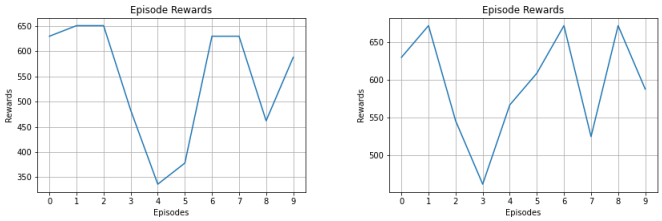

Figure 6: Plot of episode rewards in 10 games from different agents

The rewards after 150 episode is already better than the Random Agent, but the not at the superhuman level reached by [1]. After 300 episode the improvement are significant, the mean rewards are much higher and the time spent in game increases and the strategy learned allows the agent to reach nearly the same reward.

4.3 Compare

After the confirmation that the agents learn to play with both methods, it's necessary to compare directly the two methods implemented. To do this I used the best agent for each algorithm and compare the rewards and actions obtained during 10 games.

Figure 7: Comparison over reward of best agent for each technique

The Actor-Critic win the comparison by far though it was trained with less episodes and it's more difficult to train. Moreover the DQN haven't reached a good quality in strategy.

5 Conclusions

The project aims to implement two Reinforcement Learning methods, let them train a Convolutional Neural Network to play Assault. The results are encouraging even if obtained with small training time, furthermore the results of [1] are way better. The agents seems to learn a good strategy, like that it's better to use always the laser cannon to make points and try to avoid the laser from the aliens. The latter part is lacking a bit but the final results of both the agents could be improved by let them play more time. This situation confirm that the hyper-parameter configuration of both methods could be defined satisfactory and at least I think that now an improvement it's only matter of time.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A Graves, I Antonoglou, D Wierstra, M Riedmiller - Playing Atari with Deep Reinforcement Learning - DeepMind Technologies

[2] H. van Hasselt, A. Guez, D. Silver - Deep Reinforcement Learning with Double Q-learning - Google DeepMind