EAComp
Encontro Acadêmico de Computação

VI Encontro Acadêmico de Computação - UFMA
 ISSN: 2594-486X

Deep Learning for Games Directly from Pixels

Ivan de Jesus Pereira Pinto[1], **Gabriel Garcez Barros Sousa**[1]

[1]Universidade Federal do Maranhão (UFMA)
Av. dos Portugueses, 1966 – 65065-545 – Vila Bacanga – MA – Brazil

navi1921@gmail.com,gabrielgarcezbsousa@gmail.com

Abstract. *The creation of new algorithms in hopes of developing a General Artificial Intelligence through Reinforcement Learning has become a popular topic in recent times. This is mainly due to the field being fueled by the recent advances on Deep Learning methods. In this paper we study and experiment with the new class of algorithms, called Deep Reinforcement Learning, for training a neural network agent to play games of varying complexity.*

Resumo. *A criação de novos algoritmos na esperança de eventualmente desenvolver uma Inteligência Artificial Geral por Aprendizagem por Reforço se tornou um assunto popular recentemente. Isso é principalmente devido ao campo de pesquisa ter passado por grandes avanços em métodos de Aprendizagem Profunda. Nesse artigo, nós estudamos e experimentamos com a nova classe de algoritmos, chamados Aprendizagem pro Reforço Profundo, para o treino de uma rede neural aplicada a jogos de complexidades diferentes.*

1. Introduction

The development of Artificial Intelligence (AI) through Reinforcement Learning (RL) (Sutton and Barto 1998) has become quite popular in spite of recent major advances. RL is a method based on psychological studies done on animal behaviour, which has an agent learning how to optimally control their environment. This learning strategy has many benefits but its largest hindrance is how to represent complex environments such as the real world.

Many different methodologies for RL have been developed around the world; In 2014 a paper was published detailing the creation of a new Deep Learning strategy that had trained an AI that could play several different Atari 2600 games with a single algorithm(Mnih et al. 2013). Months later, the startup company that published the paper was acquired by Google and they remain at the forefront of Deep Reinforcement Learning, developing many new techniques in hopes of reaching a completely General Artificial Intelligence.

This paper will study different techniques found in literature on how to train an AI through Reinforcement Learning to play video games directly from the game screen's pixels. Video games are a naturally great testbed for AIs since they provide environments of varying complexities that are relatively simple to validate.

2. Theoretical Background

The method used to formalize the Reinforcement Learning problem that this paper tackles is called the Markov Decision Process. This process is widely used in robotics and

development for robotics and AI around the world. An MDP is a 5-tuple represented by $< S, A, P_a(s, s'), R_a(s, s'), \gamma >$ where

- S and A are finite sets of possible states and actions
- $P_a(s, s')$ is the probability of executing action a at time t will result in transitioning from state s to s'
- $R_a(s, s')$ is the reward received from transitioning to s' from s after carrying out action a
- γ is a percentage that represents the importance between future rewards from present ones

The state for Deep Reinforce methods is a frame from the game's screen, that may or may not have been through some form of pre-processing. The actions are game inputs such as up or down for Pong's case. The rules that decide on how an agent will select an action are called policy. Finding this policy is the core problem for MDPs.

3. Reinforcement Learning

A Markov Decision Process problem can be solved by different approaches, the most common ones being dynamic programming and reinforcement learning(RL). The latter is a branch of machine learning, where unlike the popular supervised learning method it does not need a set of labeled batches for training, using directly the scalar reward for each action input given by the MDP environment.

In RL we have the return $R_t = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{t-1} r_t$ called the total discounted reward. The goal is then to maximize the expected reward from each state s_t. The value of the state s_t under a policy π is defined as $V^t(s) = E[R_t | s_t = s]$ and is simply the expected return for following policy π from state s. The action value can be easily derived from it, with $Q^t(a, s) = E[R_t | s_t = s, a]$ being the expected return selecting action a in state s and following policy π. Finally the goal of RL is to find an optimal action value function $Q^*(a, s) = max_\pi Q^\pi(a, s)$ that tell us the best action for every possible state. The next sections show two popular distinct approaches to reach this goal.

3.1. Value Based

Value-based methods in RL attempt to learn the policy by iteratively learning a value function. A common representation of a value function is a lookup table that fits all possible states of the MDP. With larger MDPs the number of states grows exponentially, rendering it infeasible to use a table to store them all. We then require the use function approximators to represent the value function, such as a neural network. The new value function is $Q(a, s; \theta)$ with θ as the parameters of the function approximator. The parameters can be updated by a series of methods, the most popular one being Q-Learning, that directly approximates the optimal action value function. The learning process occurs when the parameters of θ are learned iteratively by minimizing a series of loss functions as given in the equation:

$$L_i(\theta_i) = E(r + \gamma max_{a'} Q(a', s'; \theta_{i-1}) - Q(a, s; \theta_i))^2 \quad (1)$$

where s' is the next state and L_i is the ith loss function.

3.2. Policy Based

Policy based methods differ from value ones by directly parametrizing the policy $\pi(a|s;\theta)$ instead of the value function. The θ updates are usually made by performing approximate gradient ascent on the expected return $E[R_t]$. In practice we use an estimate of $\nabla_\theta E[R_t]$ given as $\nabla_\theta log\pi_\theta(a,s)$ that is called a score function. The score function can take many forms, and the most commonly used is the softmax policy. The complete policy gradient equation is:

$$\nabla_\theta J(\theta) = E_{\pi\theta}[\nabla_\theta log\pi_\theta(a,s)Q^{\pi\theta}(a,s)] \tag{2}$$

The advantage of this approach is the convergence to local optimum, the effectiveness on high dimensional action spaces, and the stochastic policy learned, which is very useful for games like "rock, paper, scissors" that need random behaviours to not be predictable.

4. Deep Reinforcement Learning

As has been mentioned by the past sections, the greatest problem encountered by regular Reinforcement Learning techniques is how to represent and work with high-level interpretations of an environment, such as picture or sound. Deep Reinforcement Learning was conceived to facilitate these representations by using deep neural networks to approximate a RL Problem's value function by extracting features from raw sensory data.

The class of deep neural networks applied in this paper is of the deep convolutional network, which applies the use of layers of tiled convolutional filters to extract and learn features from images similarly to how animals perceive and represent visual information. This type of network is perfectly suited for Reinforcement Learning and the next sections will be over how it can be applied.

4.1. Deep Q-Learning

Deep Q-Learning is essentially the Q-Learning method that makes use of a deep neural network as its approximator function. It was developed and applied in 2014 by having an AI learn how to play classic Atari games such as Pong and Breakout(Mnih et al. 2013). The algorithm was so successful that it is often hailed as the first large step to general artificial intelligence.

The DeepQ Network functions similarly to the previously described value tables, but instead of having a table that contains states and the expected rewards for taking an action in that state, a deep convolutional network is used to receive the state's current configuration, and output the $Q^*(a,s)$ values of every possible action from that state. Despite this, the network isn't actually trained by the most recent action and state configuration, but by random minibatches of saved states in memory. This training technique is called experience replay and is used to quicken learning by avoiding consecutively training on similar-looking transitions.

Another strategy adopted by Deep Q-Learning is the selection of random actions during training. This makes it so the AI can explore several different options and not simply converge to the first basic strategy it encounters. This is called ϵ-greedy exploration, with ϵ being a percentage that dictates the chance of the agent performing a random action at that given state. This percentage can be gradually lowered during training until it reaches a specified lower value.

The architecture used for the model is directly based on the one in DeepMind's paper(Volodymyr Mnih 2015), so it contains 3 convolutional layers with varying filter sizes and two more fully connected layers. As its input, the model receives a state in the shape of 4 preprocessed 84x84 frames of game screen. Preprocessing involved resizing to 84x84 and converting the images into grayscale so the computational complexity of learning with high complexity images could be assuaged.

4.2. REINFORCE

The reinforce algorithm was proposed by (Williams 1992), and is a Monte Carlo approach to the policy gradient defined on equation (2). It uses the return v_t as an unbiased sample of $Q^{\pi\theta}(a, s)$. The θ update rule for each state and action on the episode is given as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta log \pi_\theta(a_t, s_t) v_t \tag{3}$$

A network of 1 hidden layer with 200 hidden units is used as the parametric function θ, with RMS for optimization and standard backpropagation for training. It has a input size of 7056, which is an 84x84 pixel representation a game's screen used in the experiments. The output layer is connected to the softmax function generating a probability distribution for all actions, easing the need for a exploration-exploitation policy.

4.3. Asynchronous Advantage Actor-Critic

The Asynchronous Advantage Actor-Critic(A3C)(Mnih et al. 2016) is an extension of the reinforce inspired actor-critic. The actor-critic too, directly parameterizes the policy, but it learns an estimate value function $V^\pi(s_t)$ or *baseline*. The *baseline*, known as the critic, is used to lower the variance estimate of the policy, called actor. We call *advantage* the equation: $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$. The Advantage Actor-Critic (AAC) has its θ update changed to:

$$\theta \leftarrow \theta + \alpha \nabla_\theta log \pi_\theta(a_t, s_t) A^w(a_t, s_t). \tag{4}$$

A3C improves AAC by accumulating the updates with parallel actor-learners, and using convolutional neural networks as the function approximator. The actor-learners, that are a copy of the main learner, return their variate experience, improving the training stability of the main learner. Our A3C uses a total of 30 actor-learners.

5. Experiments

Experiments were done in Python by training the neural network to play Pong for the Atari 2600 and Mortal Kombat for the Super Nintendo Entertainment System. The Arcade Learning Environment(Bellemare et al. 2013) and Retro Learning Environment(Bhonker et al. 2016) were used to emulate, extract visual data and reward data for training for each respective game. Figure 1 shows images of both games, note how much more complex and noise-filled Mortal Kombat is.

Figure 1. Left: Pong;Right: Mortal Kombat

Pong's reward system was set up so that whenever the network agent scored a point, it would receive a positive reward of 1. But whenever the opposing player scored a point, it would receive a reward of -1. A game of Pong ends as soon as one side reaches a total of 21 points. This means at the end of a game, if the total sum of rewards ended with a positive value, the trained AI has won the match. As it can be seen from Figure 2, A3C provided the quickest results, converging to a strategy that consistently beat the opponent AI in less than half the amount of games than it was necessary for Deep Q.

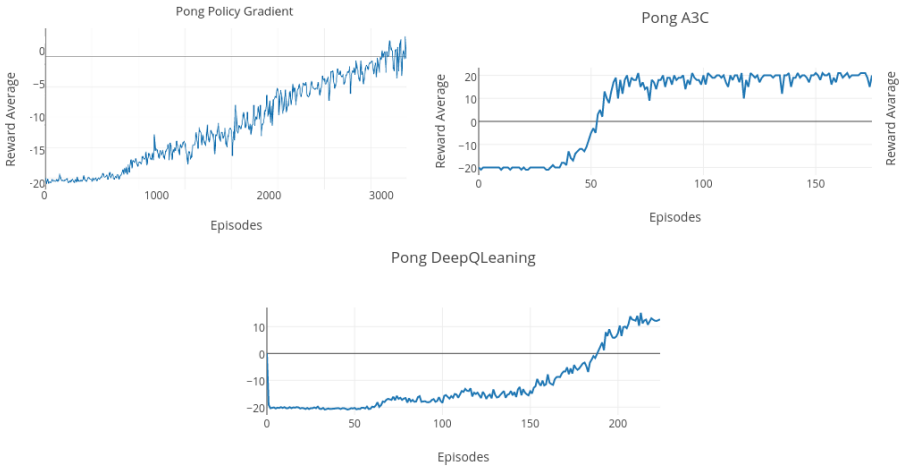

Figure 2. Deep Reinforcement Learning With Pong

For Mortal Kombat's case, the rewards were positive if our player inflicted damage, and negative if it received damage and 0 if neither. Experimentally we found out that the networks learn better with the values clipped to -1 and 1.

The results of training the three methods can be seen in Figure 3. The graphs show us that Policy Gradient wasn't able to increase its learning rate, and Deep Q shows progress, getting positive scores once in a while with its average steadily increasing. A3C was able to learn a policy smart enough to consistently win. In terms of how each technique was acting in-game, some interesting results could be inferred. The Policy Gradient method was not able to derive a strategy for it to follow. Now, Deep Q-Learning managed to devise a strategy that exploited flaws in the game's AI, by jumping around and avoiding the opponent's attack. A3C learned a similar strategy, but in only 1000 epochs versus the 3000 it took Deep Q. From this we speculate that the Deep Q would probably earn more positive rewards if given more time to train. It is important to denote that the A3C has been trained on an 8 core CPU and the Deep Q on a GTX 960M.

Copyright (c) EAComp. ISSN: 2594-486X

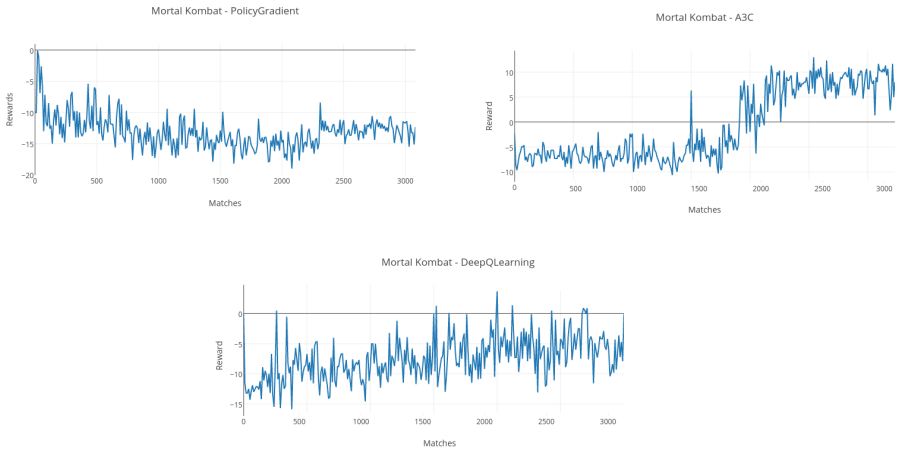

Figure 3. Deep Reinforcement Learning With Mortal Kombat

6. Conclusion

Deep Q-Learning, Reinforce and A3C were proven to be excellent techniques when used with the simple Atari 2600 game. Those methods behaved similarly on the more complex Mortal Kombat, with the DeepQ and A3C being able to exploit some of the opponent weakness. In the future, we plan to experiment to see how these techniques fare against human player, and study with better tuning of hyperparameters or different methods not covered here such as the Double Deep Q-Learning shall be explored.

References

[Bellemare et al. 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[Bhonker et al. 2016] Bhonker, N., Rozenberg, S., and Hubara, I. (2016). Playing snes in the retro learning environment. *arXiv preprint arXiv:1611.02205*.

[Mnih et al. 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

[Mnih et al. 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

[Sutton and Barto 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[Volodymyr Mnih 2015] Volodymyr Mnih, e. a. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–542.

[Williams 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.