Machine Learning for Games Directly from Pixels

Ivan Pereira[1], **Gabriel Sousa**[1]

[1]Instituto de Informática – Universidade Federal do Maranhão (UFMA)
Av. dos Portugueses, 1966 – 65085-580 – Vila Bacanga – MA – Brazil

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 recents advances on Deep Learning methods. In this paper we study and experiment with the new class of algorithms, called Deep Reinforcement Learning, for training an neural network agent to play games of varying complexity.*

Resumo. *A criação de novos algoritmos na esperança de eventualmente desenvolver uma Inteligeência 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 agente para jogar 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 is a finite set of possible states
- A is a finite set of possible 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

This 5-Tuple can be used to represent a stochastic environment containing an agent, with S representing the possible configurations of the environment and A the actions that the agent can take. With this, if an agent is set to complete some task in the environment, then they would simply need to analyze and traverse the MDP in an efficient manner. 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 state grows exponentially, rendering it infeasible to use a table to store all states. We then require 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 approximate 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:

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

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 update of θ are usually made by performing approximate gradient ascent on the expected return $E[R_t]$. In practice we use a 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; since representing these in simple tables is completely infeasible due to their complexity. This is exactly why Deep Reinforcement Learning was conceived. This form of learning makes use of deep neural networks to approximate the RL Problem's value function by extracting features from raw sensory data. Deep neural networks are layers of artificial neural networks, where each layer of the network creates a more abstract form of representation of the input. The use of these networks has become commonly used in object and speech recognition and even in recommendation systems.

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. This form of network mimics the functions of how animals perceive and represent visual information and is mainly applied in object recognition. This type of network is perfectly suited for Reinforcement Learning and the next sections will 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 by a startup company called DeepMind in 2014 by having an AI learn how to play classic Atari games such as Pong and Breakout[Mnih et al. 2013]. Their algorithm was so successful that the paper is often hailed as the first large step to general artificial intelligence, which is an AI that can freely adapt to any given situation.

The Deep Q 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 the input state. Despite this, the network isn't actually trained by 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 training by avoiding using the similar-looking transitions consecutively.

Tabela 1. Model Architecture

Layer	Input	Filter Size	Stride	Activation Func	Output
Convolutional 1	84x84x4	8x8	4	ReLU	20x20x32
Convolutional 2	20x20x30	4x4	2	ReLU	9x9x64
Convolutional 3	9x9x64	3x3	1	ReLU	7x7x64
Fully Conn 1	7x7x64			ReLU	512
Fully Conn 2	512			Linear	18

A common strategy used by many RL algorithms, including Deep Q-Learning, is the selection of random actions during training. This is done so the AI is encouraged to explore a wide variety of actions and not overfit and exploit a singular basic strategy. This is called the ϵ-greedy exploration, where ϵ is a percentage that dictates the chance of the agent performing a random action at that given timestep. This percentage can be gradually lowered during training until it reaches a specified lower value. Algorithm 1 shows how Deep Q-Learning can be implemented in pseudocode.

```
initialize replay memory $D$ with capacity $N$
initialize action-value function $Q$ with random weights
observe initial state $s$
for episode=1, M do
    with random probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = argmax_a Q(s_t, a)$
    execute $a_t$
    observe reward $r_t$ and new state $s_{t+1}$
    store experience $< s_t, a_t, r_t, s_{t+1} >$ in $D$
    sample a batch of random transitions $< s_j, a_j, r_j, s_{j+1} >$ from $D$
    calculate target for each minibatch transition
    if $s_{j+1}$ is a terminal state then
        $t_j = r_j$
    else
        $t_j = r_j + \gamma max_{a'} Q(s_{j+1}, a_j)$
    end
    train the Q network using $(t_j - Q(s_j, a_j))^2)$ as loss function
    update $s_t = s_{t+1}$
end
```

Algorithm 1: Deep Q-Learning with Experience Replay and ϵ-greedy exploration

4.1.1. Model Architecture

The architecture used for the model is described in table 1 and is directly based on the one in DeepMind's paper[Volodymyr Mnih 2015]. As 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 new update rule is show together with the reinforce pseudo code on Algorithm 2:

```
Initialize $\theta$ arbitrarily
**for** *episodes* $s_1, a_1, r_2...s_{t-1}, a_{t-1}, r_t \sim \pi_\theta$ **do**
    **for** *t = 1 to T-1* **do**
        $\theta = \theta + \alpha\nabla_\theta log\pi_\theta(a_t, s_t)v_t$
    **end**
**end**
```

A network with 1 hidden layer with 200 hidden units is used as the parametric function θ, with RMS for optimization and standard backpropagation for training. The output layer is connected to the softmax function generating a probability distribution for all actions

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 and reward data for training for the respective games. Figure 1 shows images of both games, note how much more complex and noise-filled Mortal Kombat is.

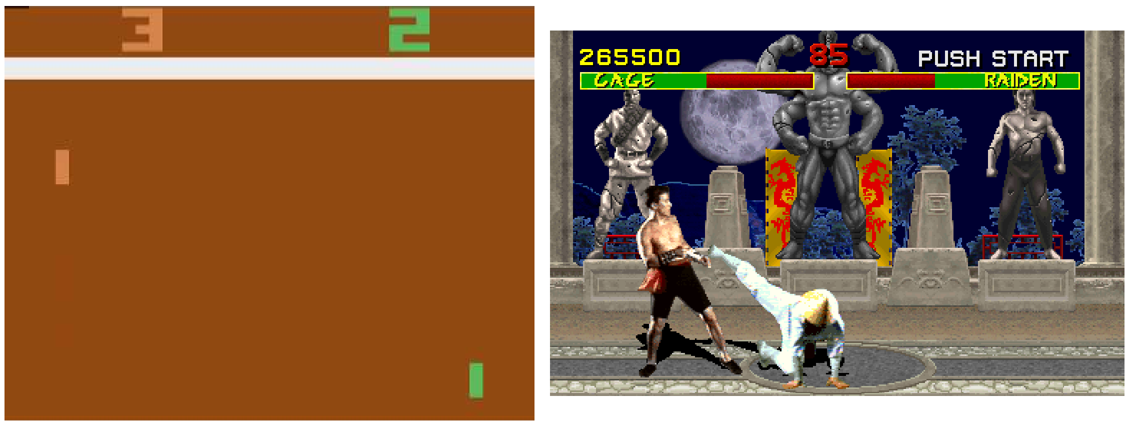

Figura 1. Left: Pong;Right: Mortal Kombat

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 received 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, after around 20 hours of training and over 2000 games of pong, the Deep Q Network started to consistently defeat the opposing computer player.

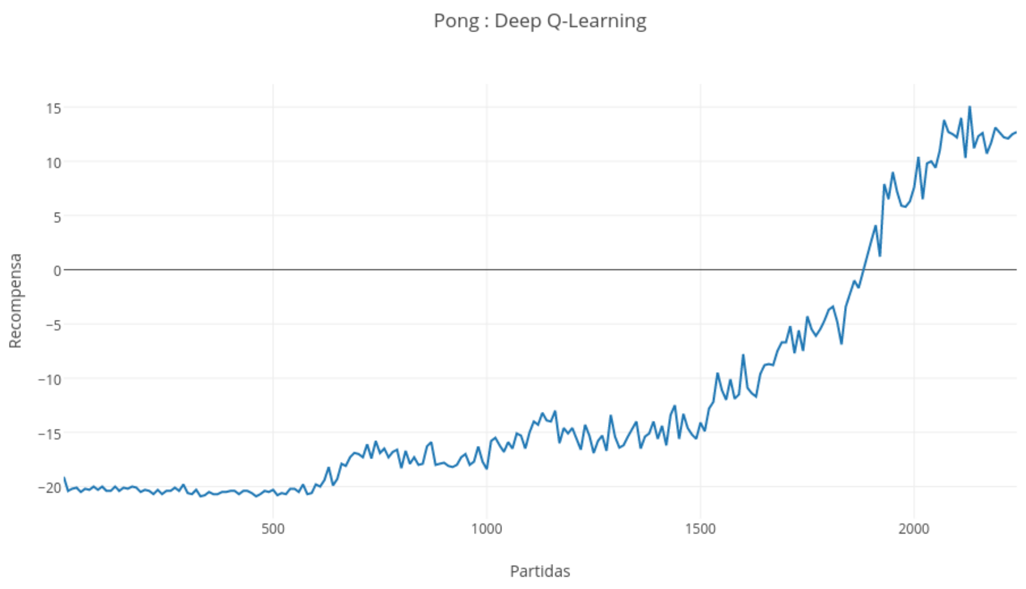

Figura 2. Deep Q-Learning with Pong

Figure 3 details how well training an AI with the reinforce algorithm. The AI started to provide positive results after around 3500 games, which is clearly much slower than Deep Q. However, this algorithm has the benefit of being less computationally-demanding as it doesn't make use of Experience Replay which consumes a high amount of resources.

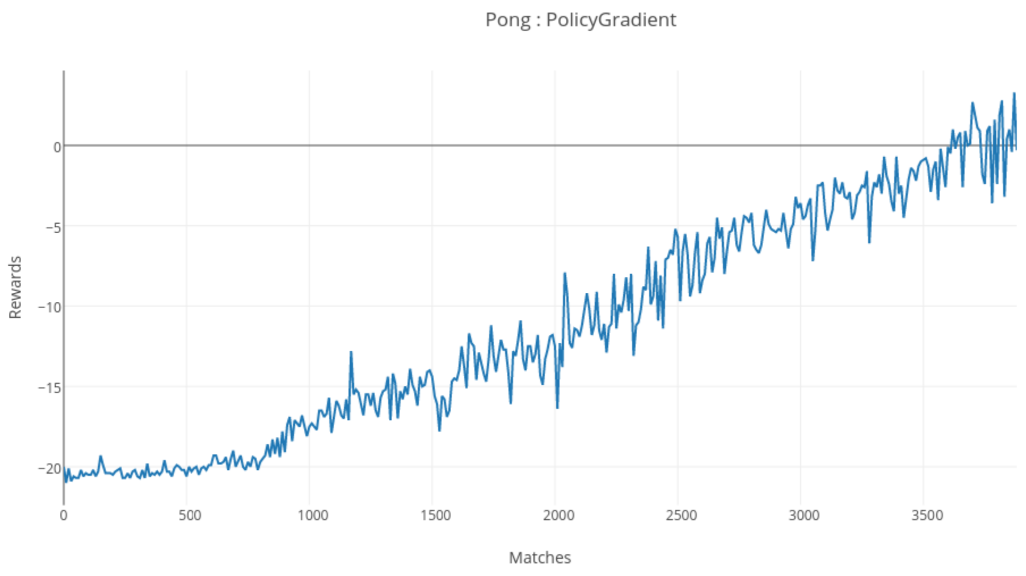

Figura 3. Reinforce with Pong

For Mortal Kombat's case, the rewards were given based on if the neural network received or did damage, and won or lost a match. When it did damage to the opposing player or won a match, a positive reward varying on how much damage and on how well it performed in that match was given. The analogous happened whenever it received damage or lost a match but with a negative value. After several tests, it was noticed that these values would confuse the network since most times when it attacked the opposing player, they would counterattack with a stronger blow, resulting in a net sum of negative reward. This made it so the network opted to always dodge every attack and end the game in a tie. In an attempt to solve this, the positive rewards were clipped to 1 while the ones to -1.

Figure 4 shows the resulting rewards for Deep Q-Learning. It's possible to see that the AI never learned a dominant strategy and never started to consistently defeat the opponent in the time trained. This is most likely due to the hyperparameters used not being well tuned for Mortal Kombat.

Figure 5 shows the network trained with reinforce, and its performance clearly is not much different from Deep Q-Learning. This give us hints that maybe a better configuration of the network can boost both algorithms performance.

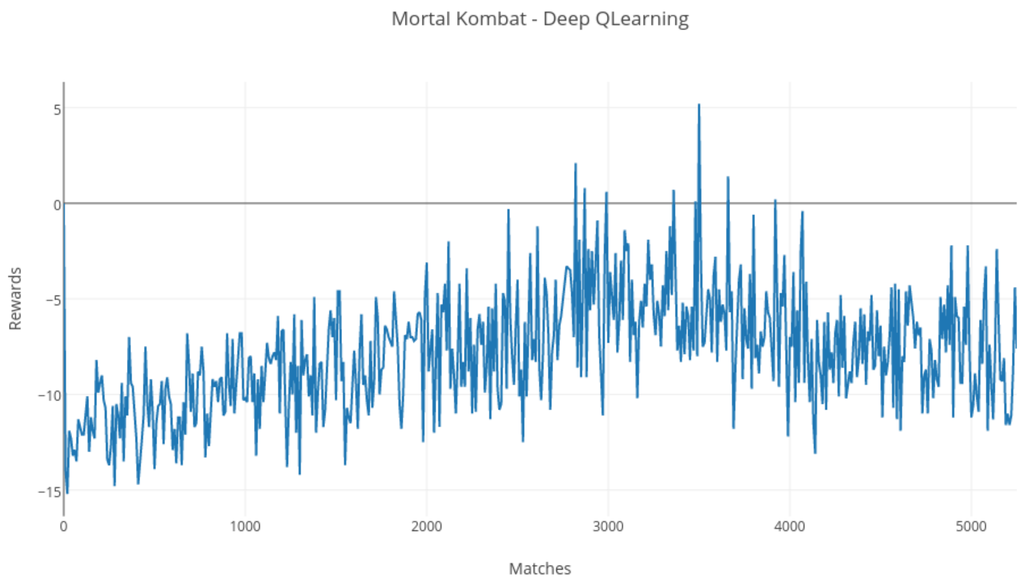

Figura 4. Deep Q-Learning with Mortal Kombat

6. Conclusion

Deep Q-Learning and the Reinforce algorithm were proven to be excellent techniques when used with the simple Atari 2600 game. However, these methods did not seem to work with the more complex Mortal Kombat, since it never converged to a dominant strategy in the time given to train it. In the future, tuning hyperparameters used to train the previous algorithms should be done and different methods not covered here such as the Asynchronous Advantage Actor-Critic[Mnih et al. 2016], Double Deep Q-Learning[van Hasselt et al. 2015] and Dueling Double Deep Q-Learning[Wang et al. 2015] algorithms should be explored.

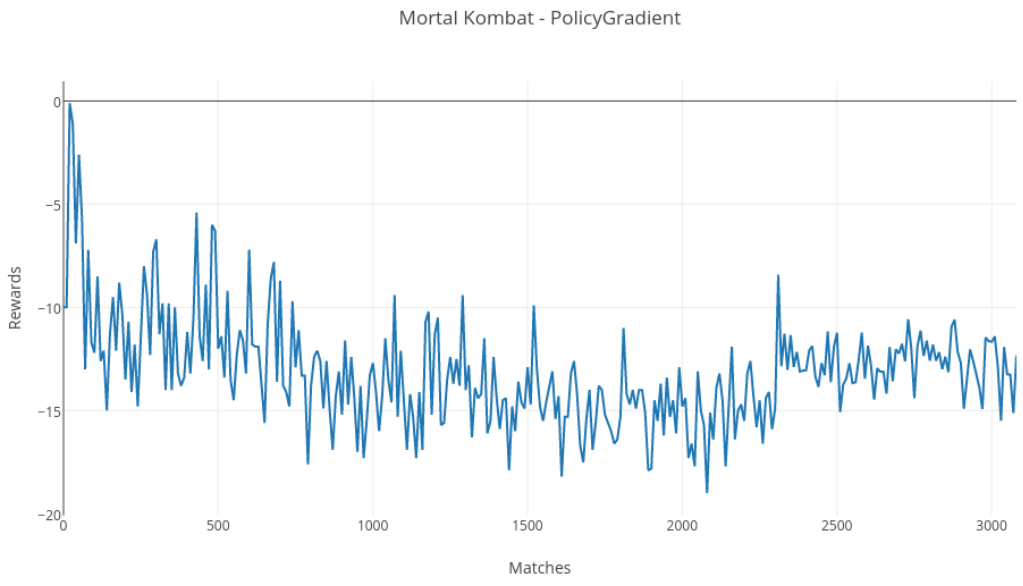

Figura 5. Reinforce with Mortal Kombat

Referências

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

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

van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

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

Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581.

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