Playing Space Invaders with Deep Reinforcement Learning

Pablo López Santori

June 7th, 2019

Domain Background

Reinforcement Learning works by maximizing some value function where the agent chooses actions that maximize the expected reward. In the past few years, reinforcement learning has seen great progress thanks to advancements in Deep Learning, in areas such as computer vision. These breakthroughs have allowed reinforcement learning algorithms to successfully use neural networks as function approximators, which provide better results than previous techniques when working with continuous spaces.

Atari games have been used as a benchmark for Reinforcement Learning since the introduction of the Deep Q-Learning algorithm (Minh et al. 2013). The motivation for this project is to recreate the algorithm for the Space Invaders game and surpass the performance of a human player. To accomplish this I'll implement some improvements that have appeared in the recent years after the original paper, such as prioritized experience replay (Schaul et al. 2015) and duelling networks (Wang et al. 2015).

Problem Statement

This problem can be framed into a Makarov Decision Process (Sutton and Barto, 2018), where the agent is the learner making the decisions, and the environment is the Atari simulator. At each time step t the agent will select an action from a discrete space of actions $A = \{1, ..., K\}$. One time step later, as a consequence of that action, the agent will receive a new state $s_t \in S$ from the emulator as an image, as well as a reward R_{t+1} based on the current score after taking that action. This finite MDP process gets repeated rising to a sequence trajectory like $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$.

The goal of the agent is to select actions that maximize the expected return, thus, we assume that the future rewards are discounted by a factor of γ per time step. The expected return can be defined as $\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.

Datasets and Inputs

Rather than building my own Space Invaders' clone I'll use an OpenAI Gym environment to run the simulations. Gym is a toolkit for developing and comparing reinforcement learning algorithms.

The agent will only receive raw pixel data as input, the same as a human would see on a screen. With only these raw images as inputs, the agent will learn to play Space Invaders from scratch, based on its previous experiences and rewards. In order to give the agent

access to a sequence of frames, M images will be stacked together resulting in a final state space size of $n \times n \times M$.

Solution Statement

The goal of the agent is to select actions which will maximize the expected return. The agent chooses actions based on a policy π, which is a mapping from a state s to an action a. The action-value function can be defined as a function that yields the expected return if the agent starts in state s, takes action a, and then follows the policy for all future time steps.

Bellman's expectation equation $Q^*(s,a) = \max_{\pi} E\left[G_t \mid S_t = s,\ A_t = a,\ \pi\right]$ gives us a way to express the action-value of any state in terms of the values of the states that could potentially follow. In a temporal difference problem, this equation $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t))$ can be used to create an update rule for every time step and find the optimal action-value function. In order to keep a balance between exploration and exploitation, I'll use a ϵ-greedy policy.

I'll use a Convolutional Neural Network as the function approximator which will have a better performance in a continuous state space. The update rule will need to be changed so that the neural network can compute the loss function and update w using gradient descent. Same as before, I can use the estimated return in the next time step as a target to compute the loss function and create our new update rule: $\Delta w = \alpha(R_{t+1} + \gamma max_a Q(S_{t+1}, a, w^-) - Q(S_t, A_t, w))\, \nabla_w Q(S_t, A_t, w)$, where α is the importance of newer memories and w^- are the fixed function parameters that don't change during the learning step.

Benchmark Model

The idea behind reinforcement learning is to find a way for computers to learn from interacting in an environment in a similar way that humans do. For this reason I think it is a good idea to compare the results from this project to the results obtained from a professional human games tester (Minh et al. 2015).

Evaluation Metrics

In a supervised learning scenario, one can easily separate the dataset into train, test and validation. In reinforcement learning, evaluating the model's performance is not as straightforward. The evaluation metric I'll be using is the performance of the agent over time, by observing the total reward obtained by the agent in each episode during the training.

Project Design

Preprocessing. Atari games are displayed at a resolution of 210 by 160 pixels, with 128 different possible colors for each pixel. To reduce this complexity a number of preprocessing steps will be performed to the original raw images. First, the images will be

converted to grayscale, as the colours don't provide the agent with any useful information. Then, the images will be converted to a square of $n \times n$ pixels. Square images allow to perform more optimized neural network operations on GPUs. Finally, in order to give the agent access to a sequence of frames, M images will be stacked together for every state input, creating a state space of $n \times n \times M$. This preprocessing and stacking operation is denoted by Φ.

Model Architecture. On the output side, unlike the traditional reinforcement learning setup where only one Q-value is produced at a time, the deep Q-Network will produce a Q-value for every possible action in a single forward pass. By doing it this way the agent can choose the action with the maximum value without having to run the network one time per possible action.

The first layers of the network will be convolutional layers, which will be in charge of finding spatial relationships. Also, since M frames are stacked to create a single input, the convolutional layers will be able to extract some temporal properties across these frames. These layers will be followed by a hidden fully connected layer with RELU activation function and an output layers with as many outputs as possible actions.

Training the neural network requires a lot of data. There are situations in which the network weights can diverge due to the high correlation between actions and states, which can result in a very unstable and inefficient policy. To overcome these challenges two techniques are used: Experience Replay and Fixed Q-Targets.

Experience Replay consists on saving the state, reward, done tuples in a replay buffer and re-use them later to train the agent. These memories can be sampled into batches and feed them to the network at once. Experience Replay can help by recalling some unusual states as well as preventing the agent to learn from unwanted sequences by shuffling the batches.

Fixed Q-Targets consists of fixing the function parameters used to generate the target . It can be implemented by copying w into w^-, and use it to generate targets while changing w for a certain number of learning steps. Then, the value of w^- gets updated to the value of w, the agent learns for a number of steps, and so on. This technique helps decouple the parameters.

Algorithm. There are two main processes that are interleaved in this algorithm. The first one samples the environment by performing actions and store away the experienced tuples in a replay memory. The other one is where a small batch of tuples is selected randomly from the memory and uses the gradient descent step to learn from that batch. These two processes are not directly dependent on each other.

The rest of the algorithm is built to support the two steps. The algorithm starts by initializing a finite replay memory D with capacity N. Then, the parameters w for the action-value function $\hat{q}$ are initialized. To use the fixed Q-targets technique, w^- will be initialized as w, and will be periodically updated later on. Finally the necessary preprocessing for the input images will be implemented, followed by a stacking of M frames together to create a sequence.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Algorithm developed by Minh et al. (2013)

Improvements. Ever since the introduction of the Deep Q-Learning algorithm (Minh et al. 2013) a number of improvements to the algorithm have been introduced. The two of them that I'll try to implement are Prioritized Experience Replay (Schaul et al. 2015) which prioritizes rare memories in the memory buffer; and Dueling Networks (Wang et al. 2015) where the network uses 2 streams: one to estimate the state values, and one estimate the advantage values. Both streams get combined at the end to get the desired Q-values.

References

Mnih, V., Kavukcuoglu, K., Silver, D., A. Rusu, A., Veness, J., G. Bellemare, M., Graves, A., Riedmiller, M., K. Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, [online] (518). *Available at: https://www.nature.com/articles/nature14236* [Accessed 6 Jun. 2019].

Minh, V., Kavukcuoglu, K., Silver D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller M. (2013). Playing Atari with deep reinforcement learning. *arXiv*. [online] Available at: https://arxiv.org/pdf/1312.5602v1.pdf [Accessed 5 Jun. 2019].

Schaul, T. Quan, J., Antonoglou, I. and Silver, D. (2015). Prioritized Experience Replay. *arXiv*. [online] Available at: https://arxiv.org/abs/1511.05952 [Accessed 5 Jun. 2019].

Sutton, R. and Barto, A. (2018). *Reinforcement learning*. 2nd ed. Cambridge, Massachusetts: The MIT Press, pp.47-68.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2015). Dueling Network Architectures for Deep Reinforcement Learning. *arXiv*. [online] Available at: https://arxiv.org/abs/1511.06581 [Accessed 5 Jun. 2019].