CSE 573

Spring 2022

Paper Report

Playing Atari with Deep Reinforcement Learning

1. **Problem/Question:**
    a. Most successful deep learning methods require large amounts of hand-labelled training data. This isn't realistic in most cases, especially in the realm of Atari games. The games typically have correlated states that aren't independent, making it more difficult to apply ML methods on these types of games. The paper attempts to answer the question of how we can overcome these challenges using CNNs from video input data in complex RL environments. They want to master control policies for Atari games using raw pixel input and provide online learning that updates stochastic values through Q-learning and trained deep networks for RL.

Solution/Approach:

b. Their approach starts by establishing the setting of the Atari environment. They use ε as a method of representing images from the emulator rather than its internal state found within the game. In lecture, we often compared the difference between offline (MDPs) and online (RL) learning. In this case, the idea that ε is just a set of observed images implies the model will be an online learning scenario, but since the emulator is set to terminate in a finite number of steps we can have a large but finite MDP. They use the idea of efficiently training deep neural networks using updated values of stochastic gradient descent from RGB images rather than internal training sets. They reference Tesauro's TD-Gammon, an RL program that was similar to Q-learning where the agent approximated values using an MLP with a hidden layer. They alter the setup a bit by storing an agent's experience into a time step, $e_t = s_t, a_t r_t s_{t+1} a_{t+1}$. However, their work revolves more around NFQ, an optimization of loss functions using RPROP to update the Q-network. Obviously, their input revolves around RGB images rather than internal data sets. Their algorithm begins by initializing the necessary values of the game, and iterating through each episode with each time step maximizing the Q* and obtaining its reward and action through the inputted RGB image. It then stores the transition and determines whether the transition is terminal or non-terminal, allowing it to calculate its gradient descent step. The algorithm repeats this method until it can find the most optimal action for the agent as the game progresses. In figure 1.1, it depicts their suggested algorithm for implementation.

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

Figure 1.1: Depicts the paper curated Deep Q-Learning algorithm.

Contributions:

c. This paper contributes to both the online Q-learning and RL games a whole. Q-learning with non-linear function approximators could cause Q-networks to diverge, and this has been addressed by using gradient temporal-difference methods. In the case of this paper, they have used an optimization of loss functions using RPROP to update the Q-network and their approach applies RL directly from visual inputs, allowing for learned features that are directly relevant to discriminating actions. Q-learning typically pulls from internal state values rather than visual inputs. Their smooth predicted Q's during their training on games such as Breakout and Seaquest meant there weren't any divergence issues. This contributes a solution to the Q-learning issue of divergence on non-linear function approximators.

This paper has also set a precedent for deep Q-networks being implemented on different games and optimized in different ways. An example of this is independent developers such as Andrew Grebenisan creating double-deep Q networks to optimally beat Super Mario Bros on the NES [1]. Not only has this article contributed to branching out algorithms such as Q-learning and TD-Gammon, it has also been a pioneering step to optimizing more complex RGB inputted agents. In fact, it is one of the most widely sourced deep learning models in the realm of control policies from sensory inputted reinforcement learnings. People are already optimizing the structure of their Q-learning algorithms by implementing biases that don't over-estimate complex structures such as a Mario Brothers obstacles [1].

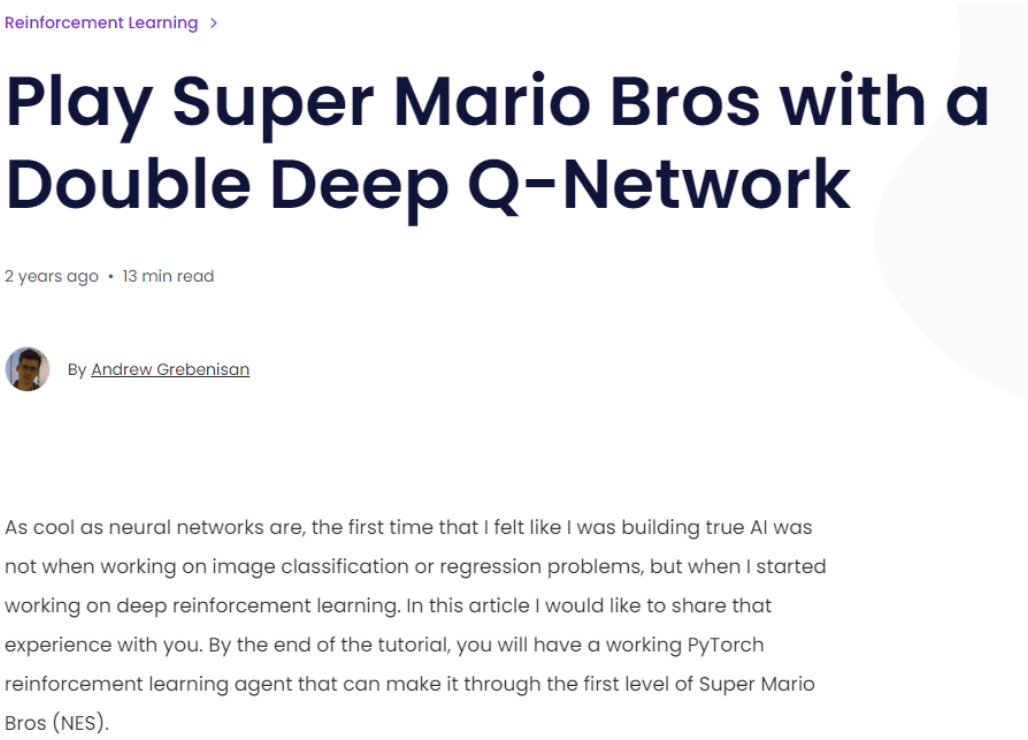

Figure 1.2: Depicts an article showing an optimal Super Mario Bros agent from an independent developer.

2. **Novelty, significance, and empirical results:**
   The paper written by DeepMind Technologies is the first deep learning model to learn control policies from RGB input using reinforcement learning. This paper is significant because it was an advancement of RL during its time.

   The algorithm is model-free and off-policy. It takes samples from the emulator ε without constructing an estimate of it and learns about a greedy value while following behavioral distribution for exploration of the state space. All the logic needs to be within the neural network itself. However, there is significance in the way the paper orchestrates its functions. Something very notable is their reference to the differentiation to the loss function with respect to the weights. Modern developers now no longer directly implement this type of loss function; they typically just run a built in TensorFlow function to directly estimate the values.

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right].$$

Figure 1.3: Depicts the gradient loss function.

The input RGB to the neural network is a small 84x84x4 image. The neural network is small, consisting of two convolutional layers with 16 fairly large 8x8 filter.

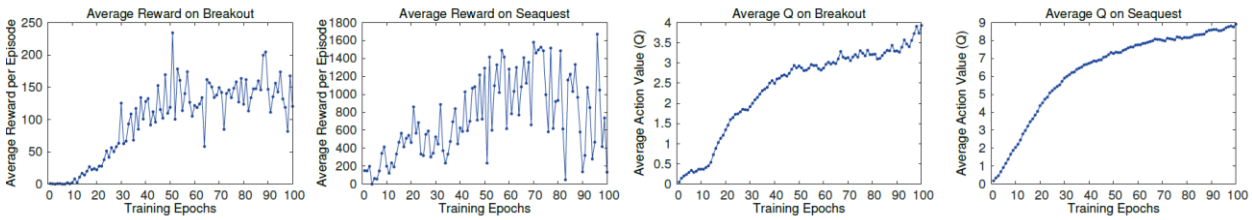

Figure 1.4: Depicts the Epoch results for Seaquest and Breakout.

The empirical results show plots of the rewards being a bit noisy at the start but then improving over time. The average Q continuously goes up during training meaning it is a successful method. Something interesting is the Q-value will spike when an enemy appears on screen, regardless of whether they are a threat or not. We see that in the figure of Seaquest with "Frame #" in the x-axis.

In the table where the paper compares the results of the implementations of Sarsa, Contingency, Human, and DQN (Their own method), it shows the DQN doing better than all methods except for some games with humans having a greater average total reward. We can see in Q*bert, Seaquest, and Space invaders, that the human was doing significantly better than the DQN which could be a result of false reward analysis or an error in estimation due to the algorithm needing to plan further ahead.

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Figure 1.5: Depicts the average rewards for different methods.

3. **Strengths of the Paper**

   The paper does a very good job of describing their procedures and how they are distinguishing themselves from previous methods. With background knowledge on RL, it is clear to the reader their approach in conquering Atari games using RGB inputs for neural network analysis in RL. The pseudocode for their experience replay algorithm was very clear and easy to understand. They gave good introduction to the algorithms referenced and explained why their method was different and what problems they solved in comparison to the old methods. The description of their neural network was clear and concise and their experiments offered supporting evidence for their claims. Their training reward graphs allowed for a clear understanding of why their method was running well, and their table scores allowed for a neat comparison of how other algorithms compared to theirs. In essence, the paper was a fun and an interesting read and I thoroughly enjoyed how they articulated their procedure and the results that came with it.

4. **Weaknesses of the Paper**

A small critique that I found throughout the paper was this idea of an expert human. I'm a bit confused on what they considered an expert human player to be. The scores that they received in Figure 1.5 were quite arbitrary and had no reasonable scale for what an average, weak, and good human would score. The DQN scores were also somewhat arbitrary because we were never told how many times these samples were ran, thus never understanding the variance and fluctuations of each methods' rewards.

Their explanation of RL was very dense and carried lots of information that could have easily been missed if the reader had little knowledge of RL or if they had never taken CSE 573 before. I found myself referencing the lecture slides to ensure that I was following along properly and understanding their Q*, discount, and other functions. I understand that their focus was showcasing their improvements to a new method based on fundamental RL, but it still could have displayed more explanation on how these standardized RL equations came about [2].

References

[1]. Grebenisan, Andrew. “Building a Deep Q-Network to Play Super Mario Bros.” *Paperspace Blog*, Paperspace Blog, 9 Apr. 2021, https://blog.paperspace.com/building-double-deep-q-network-super-mario-bros/.

[2]. Mnih, Volodymyr, et al. “Playing Atari with Deep Reinforcement Learning.” *ArXiv.org*, 19 Dec. 2013, https://arxiv.org/abs/1312.5602.