Mastering Atari using Deep Reinforcement Learning

CS419: Course Project

Anupam Nayak, Darin Jeff, Ishan Kapnadak, Sheel Shah

May 14, 2021

Motivation

Reinforcement learning is one of the three key paradigms of machine learning, alongside supervised and unsupervised learning. The main aim of reinforcement learning is to design intelligent agents that can learn how to optimise their strategies to maximise the notion of a cumulative reward. One of the long-standing challenges in this field has been to learn to control agents directly from high-dimensional sensory input data such as images, for vision tasks, and audio, for speech tasks. Until recently, most solutions to this problem relied on hand-crafted feature representations of these high-dimensional inputs. Of course, the quality of performance of such reinforcement learning systems also depended on the quality of feature representation used.
However, this problem was tackled recently by Mnih et al. 2013 who proposed the first deep learning model that successfully learned to control policies directly from high-dimensional sensory inputs using reinforcement learning. This led to the birth of the field of *deep reinforcement learning*. The model proposed by Mnih et al. was a convolutional neural network, trained with a variant of Q-learning, which takes raw pixels as input and whose output is a value function estimating future rewards. This sort of a model has since come to be known as a *deep Q-network* (DQN). The goal of this project is to successfully implement a few variants of the DQN and to explore future improvements and advancements in the field of reinforcement learning in general.

Theory

In this project, we explore two variants of the DQN, namely,

1. the Double DQN, and
2. the Dueling DQN.

We now discuss the fundamental ideas behind Deep Q-Networks, and explain how each variant improves upon the plain, vanilla DQN.

Deep Q-Networks

The notion of reward in a deep Q-network is represented as the Q value, which is the optimal sum of discounted rewards associated with a state-action pair.

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \dots$$

where s_t and a_t are the state of the environment and the action taken by the agent respectively at time step t. r_{t+1} is the reward the agent receives after taking the action a_t. γ is the discount factor. We can incrementally learn the Q-function by iteratively solving the following equation

$$Q(s_t, a_t) = R_{t+1} + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a)$$

In order for this iterative process to converge to the true Q function, we need:

1. $\gamma < 1$: This is easily met by choosing $\gamma \in (0, 1)$.
2. **All state, action, next-state tuples are reached under the conditioned policy**: In order to meet this condition, we use an epsilon-greedy exploration strategy. We introduce an exponentially decaying (with iteration) parameter, ϵ and for each iteration, we choose that action which maximizes the expected sum of discounted reward with probability $1-\epsilon$ (exploitation) or a random action from the set of possible actions with probability ϵ (exploration). Hence we iteratively solve

$$Q(s_t, a_t) = \begin{cases} R_{t+1} + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) & \text{w.p. } (1-\epsilon) \\ R_{t+1} + \gamma \cdot \text{Unif}_{a \in A} Q(s_{t+1}, a) & \text{w.p. } \epsilon \end{cases}$$

Double Deep Q-networks

The changing Q-values for each iteration in the above approach make the convergence to the true Q values slow. Moreover, since we pick the maximum of $Q(s_{t+1}, a)$ among all actions, this leads to overestimation of the Q-value. This is overcome in *Double* Deep Q Networks by maintaining two copies of the same Q-network - Q_{local}, which is updated every iteration, and Q_{target}, which is updated with the values from the local network every few episodes. Hence, for each iteration, we solve

$$Q_{\text{local}}(s_t, a_t) = R_{t+1} + \gamma \cdot \max_{a \in A} Q_{\text{target}}(s_{t+1}, a)$$

Dueling Deep Q-networks

A dueling DQN has a model such that the output Q function is decomposed into two components, as follows.

$$Q(s, a) = V(s) + A(s, a)$$

Here, $V(s)$ represents the *value* of the state s, which is how good it is to be in state s, and $A(s, a)$ represents the *advantage* of performing action a while in state s, which is how much better it is to take action a while in state s, as opposed to all other actions. A dueling

DQN model can learn and estimate both $V(s)$ and $A(s, a)$ separately. By decoupling this estimation, our dueling DQN model can learn which states are valuable *without* learning the effect of each action at each state. This is particularly useful when the agent is in a "bad" state and it doesn't matter much what action the agent takes.
The dueling architecture consists of two streams that represent the value and advantage functions, which share a common convolutional feature learning module. The two streams are combined via a special aggregating layer to produce an estimate of Q-function. Figure 1 shows the difference in architecture between a DQN and a dueling DQN.

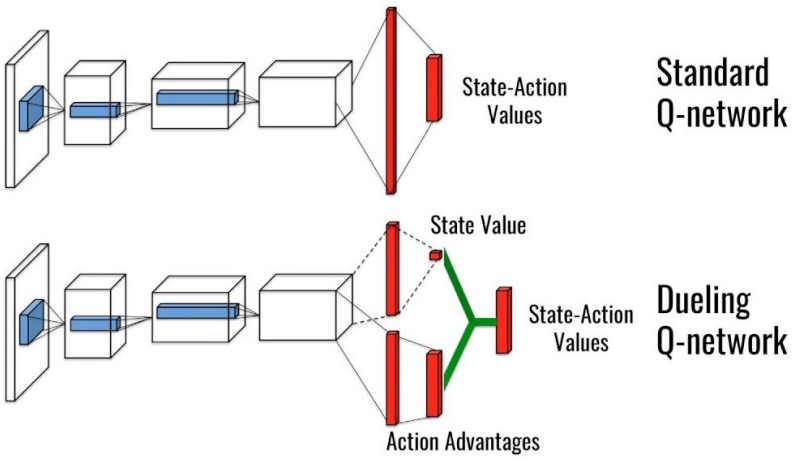


Figure 1: DQN v/s Dueling DQN

Experiment Design

As mentioned earlier, two variants of deep Q networks have been explored, namely,

1. the Double DQN, and
2. the Dueling DQN.

The models (and algorithms) were implemented in PyTorch which can use tensorization and GPU based parallel processing to speed the training process significantly. We found the PyTorch DQN tutorial (Paszke 2021) to be very useful for our project. We trained and tested these models on the popular Atari game, Pong. OpenAI's Gym library was used to simulate the environment of the game, and subsequently allow our model to interact with the game to learn how to play it.
In both the variants, double DQN and dueling DQN, we implement a deep convolutional network that processes the game screen provided by the game environment. Next, we implement a fully-connected neural network layer that uses the features processed by the convolutional network to learn the Q-function. The dueling DQN implements two parallel fully-connected networks - one network for the value function and one network for the advantage function.
After having written and tested the code for these algorithms, we finally implemented the corresponding classes and let our algorithms train on the Google Colaboratory platform, in

order to use the GPU for fast training. The results obtained are discussed in the following section.

Results

Most of the training was done on the OpenAI gym Pong environment with a No Frame Skip setting due to a lack of computational resources. The Pong environment takes about 200 thousand frames to converge, that is, give a considerably high stable mean reward compared to other environments such as Breakout, Space Invaders, Freeway etc. which take 1 to 2 million frames to converge. Figure 2 shows how a typical Pong screen looks like.

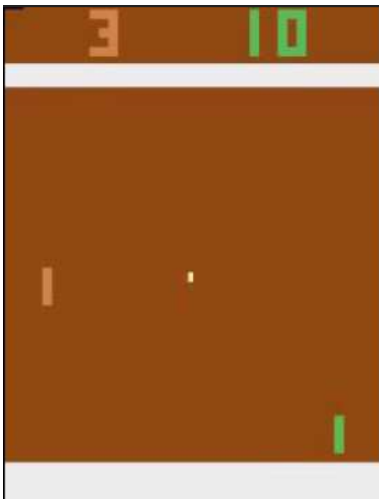
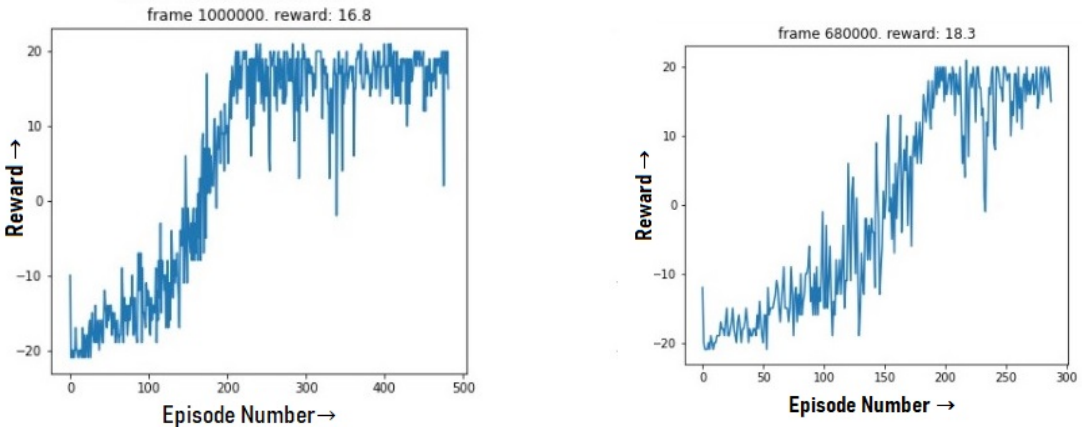


Figure 2: A typical Pong screen



(a) Rewards for Double DQN

(b) Rewards for Dueling DQN

Figure 3: Rewards obtained for both variants

Figure 3 above shows the reward obtained on training the Double DQN and Dueling DQN agents on the Pong environment. Each variant took 3 to 4 hours to train on the Google

Colab GPU. The X-axis represents the episode-number, while the Y-axis represents the reward obtained in that episode.
The reward signal for the game of Pong is defined as the agent's score minus the opponent's score. An episode ends when either of the sides scores 21 points. During the initial phase of training we see that the reward that the agents gets is -21 (that is, the agent loses by 21-0), as the model is trained, the expected reward increases and saturates at 21, which is the maximum attainable reward in an episode. Note that the reward obtained at the end of an episode can only take integer values between -21 to 21. This causes the reward to jump around in discrete steps and hence the graph appears jittery. We believe that this is due to the inherent discrete nature of the reward and not due to a higher learning rate.

Future Work

Aside from the two variants we have successfully implemented, we had also attempted to implement two other variants of the DQN algorithm as part of this project. These were the *Rainbow* DQN, and the *Noisy* DQN. However, we were not able to train these models due to a lack of computational resources. We wish to implement these two, and other variants as part of our future work in this field.

There have been a lot of recent advancements in the field of reinforcement learning with a lot of ideas derived from theoretical neuroscience (Hassabis et al. 2017). We wish to explore and implement at least two of these ideas in our future work.

Neural Episodic Control

One of the major drawbacks of traditional DQN variants is that they are slow learners. While a human can interact with the environment for a small number of times and learn the optimal strategy, the number of interactions that a DQN requires to reach the same level of performance as a human is several orders of magnitude higher. One method of overcoming this limitation is to use an agent that can implement *Neural Episodic Control*. "Critically, our agent is able to rapidly latch onto highly successful strategies as soon as they are experienced, instead of waiting for many steps of optimisation (e.g., stochastic gradient descent) as is the case with DQN" (Pritzel et al. 2017). The idea of neural episodic control is inspired by the role of the hippocampus in decision-making. An agent that is capable of implementing neural episodic control possesses several features of episodic memory, such as long-term memory, sequentiality, and context-based lookups. Pritzel et al. 2017 demonstrated that agents employing episodic control show striking gains in performance as compared to deep reinforcement learning networks.

Monte Carlo Tree Search Methods

Another drawback of the DQN model is that it operates in a reactive, or trial-and-error manner, where the agent learns by carrying out actions, observing rewards, and then deciding the optimal strategy. In contrast, humans learn in a simulation-based method. We use predictions generated by an internal mental model of the environment to estimate future

rewards and select the optimal action. Such an internal mental model is learned through experience. The DQN method does not use any model to mimic the environment. It only interacts with the environment by carrying out actions and observing the rewards. Hence, the DQN algorithm belongs to the subfield of reinforcement learning known as "model-free" reinforcement learning. A lot of recent AI research has focused on the subject of simulation-based planning and model-based reinforcement learning. One such class of methods is that of *Monte Carlo Tree Search Methods*. Silver et al. 2016 demonstrated that deep neural networks along with tree search show phenomenal results at the game of Go, and even surpass human experts of the game. Their approach utilises 'value networks' to evaluate board positions and 'policy networks' to select moves. These networks are trained via a combination of supervised learning from human expert games, and reinforcement learning from games of self-play. These games of self-play are simulated using the state-of-the-art Monte Carlo tree search method.

Acknowledgements

We would like to thank our instructor, Prof. Abir De and Ashish Tendulkar, of Google. This project would not have been possible without their guidance throughout the course. We would also like to thank the Teaching Assistants of this course for providing us with much-required assistance. We would like to thank our friends, Adit Akarsh and Sumeet Kumar, for their ingenious inputs and insightful suggestions. Lastly, we would like to thank our parents and family for their constant support and motivation through these trying times.

References

Hassabis, Demis et al. (2017). "Neuroscience-Inspired Artificial Intelligence". In: *Neuron* 95.2, pp. 245–258. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2017.06.011. URL: https://www.sciencedirect.com/science/article/pii/S0896627317305093.

Mnih, Volodymyr et al. (2013). "Playing Atari With Deep Reinforcement Learning". In: *NIPS Deep Learning Workshop*.

Paszke, Adam (2021). *Reinforcement Learning (DQN) Tutorial*. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.

Pritzel, Alexander et al. (June 2017). "Neural Episodic Control". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 2827–2836. URL: http://proceedings.mlr.press/v70/pritzel17a.html.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.