University of Waterloo
Faculty of Engineering
Department of Mechanical and Mechatronics Engineering

Deep Reinforcement Learning: Q-Learning on Atari

Self Study

Prepared by
Rae Chan Jeong
ID 20524561
userid rcjeong
4A Mechatronics Engineering
13 September 2017

Unit 5 647 Albert St
Waterloo, Ontario, Canada
N2L 3V5

13 September 2017

William Melek, Director
Mechatronics Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear William Melek:

This report, entitled ‘‘Deep Reinforcement Learning: Q-Learning on Atari’’ was prepared as my 4A Work Report for the University of Waterloo. This report is in fulfillment of the course WKRPT 400. The purpose of this report is to show the self study in the field of Deep Reinforcement Learning.

Deep Reinforcement Learning algorithm is used to create a computer program that is capable of playing games from Atari 2600 gaming console.

This work is heavily dependent on a course from University of California, Berkeley, CS294: Deep Reinforcement Learning. I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Rae Chan Jeong
ID 20524561

Contributions

This is a self-study work-term report not based on the experienced gained at my previous co-op job.

Summary

The main purpose of the report is to show a simple implementation of Deep Q Network (DQN), introduced by DeepMind Technologies, as well as it's robustness with respect to its hyper-parameters.

The major points covered in this report are the performance of the DQN agent on Atari games and the ability of a single algorithm adapt to different games.

The major conclusions in this report are that the DQN algorithm can successfully learn to play simple Atari games, given enough training time. Some of the network's hyper-parameters must be chosen carefully and this may be dependent significantly on the specific game that the agent is attempting to learn.

Conclusion in this report are that with recent advances in open source tools for Deep Learning and Machine Learning in general has allowed many people to implement advanced algorithms without much trouble. It is encouraged that for anyone that is interested in this field to try and implement their own version of this algorithm.

Table of Contents

Contributions . . . iii
Summary . . . iv
List of Figures . . . vi
List of Tables . . . vii
1 Introduction . . . 1
1.1 Deep Learning . . . 1
1.1.1 Fully Connected Layer . . . 1
1.1.2 Convolutional Layer . . . 2
1.2 Reinforcement Learning . . . 3
1.2.1 Makov Decision Process . . . 3
1.2.2 Q-Learning . . . 3
1.3 Deep Reinforcement Learning . . . 4
2 Background . . . 5
3 Deep Q-Learning . . . 5
4 Pong . . . 6
4.1 Hyper Parameter Search for Pong . . . 8
5 Analysis . . . 8
5.1 Batch Normalization . . . 10
5.2 Deeper Network . . . 10
Conclusions . . . 12
Recommendations . . . 13
References . . . 14

List of Figures

Figure 1-1 A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections between neurons across layers, but not within a layer [4]. 2
Figure 1-2 A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations [4]. 2
Figure 1-3 The agentenvironment interaction in reinforcement learning [7]. 3
Figure 3-1 Schematic illustration of the convolutional neural network [5]. 6
Figure 3-2 Deep Q-Learning with Experience Replay [6]. 6
Figure 4-1 DQN agent learning Atari 2600 Pong [6]. 7
Figure 4-2 Progress of DQN agent learning Atari 2600 Pong at start, 1 Million Time Steps, 5 Million Time Steps, from left to right 7
Figure 4-3 Comparison of the DQN agent with the best reinforcement learning methods in the literature [5]. 9
Figure 4-4 Hyper Parameter Search for Pong [6]. 10

List of Tables

Table 4-1 Network Architectures 8

1 Introduction

Deep Learning is without a doubt one of the most interesting and popular areas of Artificial Intelligence and Machine Learning. Advances in deep learning can extract high-level features from high-dimensional raw sensory readings with deep neural networks and this has led to breakthroughs in computer vision [3].

On the other hand, classical reinforcement learning techniques had had great success with learning algorithms but has relied on hand-crafted and domain specific features with linear value function and/or policies to produce respectable results.

In 2013, DeepMind Technologies had successfully combined these techniques to develop the first deep learning model to successfully learn control polices directly from high-dimensional sensory input and achieved superhuman performances on some of the Atari 2600 games tested [6].

Here we attempt to reproduce this result with the help from open source software libraries as well as the online course offering from University of California, Berkeley, CS294: Deep Reinforcement Learning. Some additional experiments to test out the different hyper-parameters are performed to evaluate the DQN algorithm further.

1.1 Deep Learning

Brief overview of deep learning is necessary in order to understand the work of deep reinforcement learning. Here we give a very brief description of advances in deep learning that is important for the DQN algorithm. Deep learning typically refers to study of neural networks that have many layers. Neural networks consists of different layers that are made up of neurons. These neurons typically have an affine transform followed by some non-linear activation function. Some of the components of the neural networks will be discussed here. For in-depth overview of the field, the text by Yoshua Bengio is recommended [1].

1.1.1 Fully Connected Layer

Most common and simple type of a layers is a fully connected layer in which neurons between two adjacent layers are fully pairwise connected but neurons within a single layer share no connections [4]. Figure 1-1 shows an example of a topology of neural network that consists of fully connected layers.

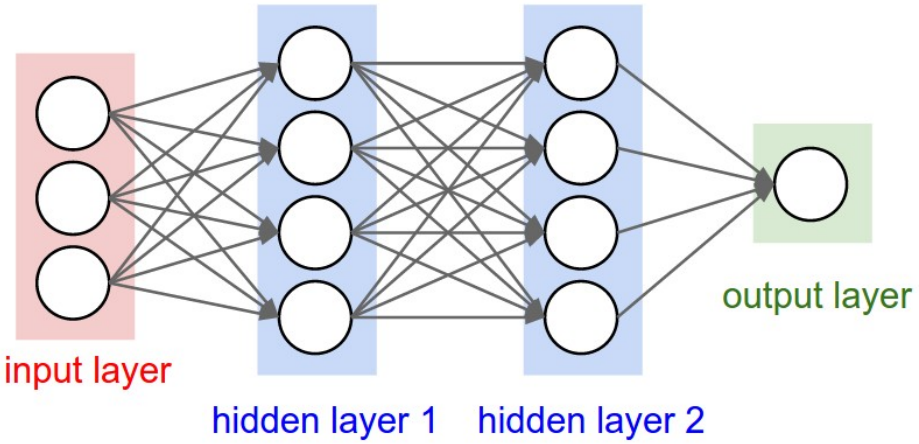

Figure 1-1. A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections between neurons across layers, but not within a layer [4].

1.1.2 Convolutional Layer

One of the major advances in deep learning and computer vision has been the use of convolutional neural networks (ConvNet) which uses convolutional layers. ConvNet architectures make the explicit assumption that the inputs are images, the convolution operation makes an assumption that the values are structured spatially, and have important local meaning, which allows us to encode this properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. Convolutional layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume [4]. Figure 1-2 shows an example of a topology of neural network that consists of convolutional layers.

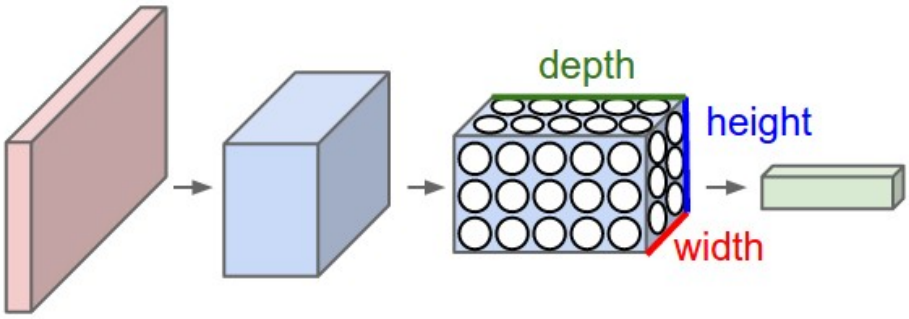

Figure 1-2. A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations [4].

1.2 Reinforcement Learning

Brief overview of reinforcement learning is necessary in order to understand the work of deep reinforcement learning. Here we give a very brief description of reinforcement learning that is important for the DQN algorithm. Reinforcement learning in this context can be understood as a method of solving the Markov Decision Process (MDP), which will be discussed in the following section. In more general sense, it can be understood as a method to maximize rewards over long term and dealing with delayed reward in possibly a stochastic environment. For in-depth overview of the field, the text by Richard Sutton is recommended [7].

1.2.1 Makov Decision Process

In reinforcement learning the learner and decision maker is called the agent and what the agent interacts with is considered the environment. At each time step, the agent is in a state which is a representation of the environment at that time step and the agent can take an action in that state. Following the agent's action, the agent receives a numerical value reward and transitions into a new state. The state transition and the reward is only indirectly influenced by the agent through its actions. This frame work is shown in Figure 1-3 Markov property is true when the state of the environment is sufficient to solving for the optimal policy. Which means, the states that the agent observes represents that environment well enough to perform optimally. A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP [7].

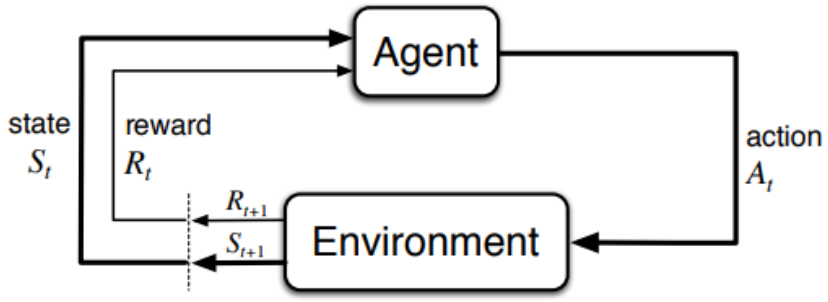

Figure 1-3. The agentenvironment interaction in reinforcement learning [7].

1.2.2 Q-Learning

Q-Learning is one of reinforcement learning algorithms that learns the Q-function. The Q-function is often referred to as action-value function because it maps from state and action pair to its value. Simply, its a function that represents a value of taking a certain action in a certain state. Q-learning is also quite flexible in the sense that it is off-policy, which means that the action-value function is

directly estimating the optimal action-values regardless of the policy used to explore the environment, also known as behaviour policy [7]. While there are different formulations of the Q-function, here we will look at discounted return in a stochastic environment. More specifically, the *return* at time t is defined as,

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

where T is the time-step at which the episode terminates. We define the optimal action-value function to be maximizing the expected value of the *return* over policy π.

$$Q^*(s,a) = max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{2}$$

Since the optimal action-value function obeys the *Bellman equation*[7], Equation (2) can be rewritten as,

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim \varepsilon} [r + \gamma max_{a'} Q^*(s',a') | s,a] \tag{3}$$

The basic idea behind many reinforcement learning algorithms is to estimate the action-value function, by using the Bellman equation as an iterative update [6].

1.3 Deep Reinforcement Learning

While deep reinforcement learning is a field on its own that is rapidly growing, here we will only discuss in the context of the DQN algorithm. The ”deep” part of deep reinforcement learning is about representations and features. As in, that the neural networks are excellent for extracting features from images and being used as a non-linear function approximation. The reinforcement learning part is about the learning algorithm. As in, that the reinforcement learning algorithms can formulate the problem in a way that the objective is clear on what to optimize. Convolutional neural networks are excellent for extracting features from raw image data and by combining this nonlinear function approximator with reinforcement learning algorithm, many games on Atari 2600 can be learned to super human performance.

2 Background

In the recent years, the field of deep learning has disrupted many other fields and the field of game playing and decision making is no exception. Another big driver for this movement can be credited to the development of open source tools as well as free online education resources. Here with the help of the course CS294 Deep Reinforcement Learning, the objective of this report is set. The first objective is to implement the DQN algorithm using open source tools. Next objective is to experiment with some of the hyper parameters of the network to see if it can improve upon its performance as well as its robustness. Open AI gym is used to simulate the Atari 2600 gaming environment. Tensorflow is used as the automatic differentiation tool. Numpy and python is used to program our DQN algorithm. Some helper functions and classes are taken from CS294.

3 Deep Q-Learning

Deep Q-Learning uses a ConvNet to approximate the action-value function and uses stochastic gradient updates to do a Bellman updates on the action-value function. In addition to this, *experience replay* [6] is used to decouple to correlations between consecutive samples. In other words, when the samples are taken in a time consecutive manner, the updates will be heavily biased in that state and it's surrounding successor states. By using *experience replay*, the samples are taken as a mini batch with an uniformly random sampling from the queue D. This also allows the the samples to be used more than once, making it more data efficient.

Another technique to improve the stability is to use a target network in the Equation (4). Target network is copy of the action-value network, just that it is copied over after many updates. This allows the target network to be more stable and allows the update to avoid chasing a moving target. The Q-Learning update at iteration i uses the foll lowing loss function:

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim U(D)}[(r + \gamma max_{a'} Q(s', a', \theta_i')) - Q(s, a, \theta_i))^2] \tag{4}$$

where γ is the discount factor, θ_i is the the parameters of the action-value function and θ_i' is the parameters of the target network [5].

In contrast to typical Q function in reinforcement learning setting where the action is fed into the function as an input, this network outputs all of the action values at that state. This allows the network to be just ran forward once rather than multiple times to find the greedy action. The action-value network architecture is shown in Figure 3-1. The deep q-learning algorithm with experience replay is shown in Figure 3-2

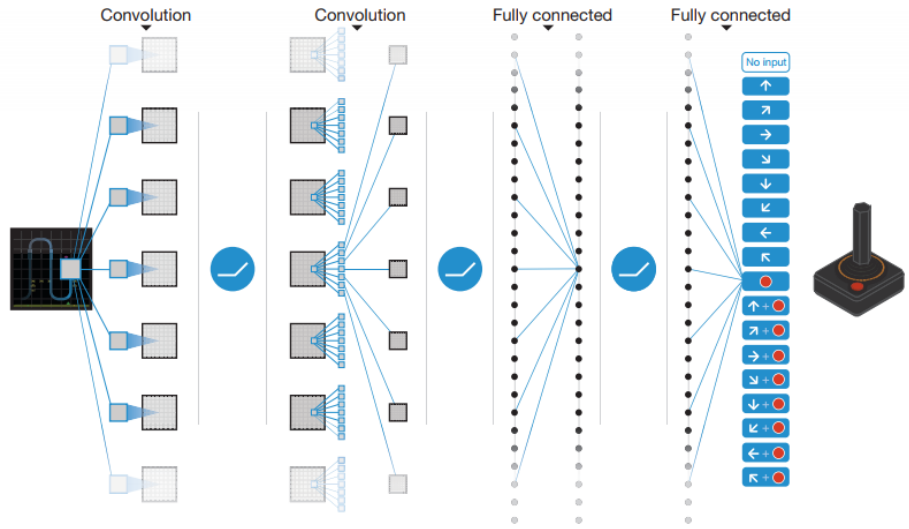

Figure 3-1. Schematic illustration of the convolutional neural network [5].

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 3-2. Deep Q-Learning with Experience Replay [6].

4 Pong

Pong is one of the most popular and intuitive classic arcade game. It also serves as a great test bed for reinforcement learning algorithms. When the ball passes through the player's side the score immediately goes down and allows for easy learning. But to score on the opponent's side, the action that influences the score is taken much earlier than when the agent receives the reward. In other words, once the ball leaves the paddle, all the consecutive action do not influence the trajectory of the ball. The delayed reward from the environment makes this a great environment to be modeled

as a Markov Decision Process. Also the game screen is rich enough to fully observe the game state and make optimal decision, making it easier to learn from.

Here we look at the performance of the DQN algorithm in Pong. DQN algorithm successfully learns to play Pong around four million time steps. The reward after four million time steps is around 20, which is approximately the maximum score that one can get in a game of Pong. The exploration is done through epsilon greedy policy and the value of epsilon is reduced linearly over the episodes. ADAM optimizer is used to train the action-value network. Figure 4-1 shows the mean reward over 100 time steps growing quite consistently while the agent learns to play Pong. This model is trained in roughly 5 hours with GeForce GTX 1070 GPU.

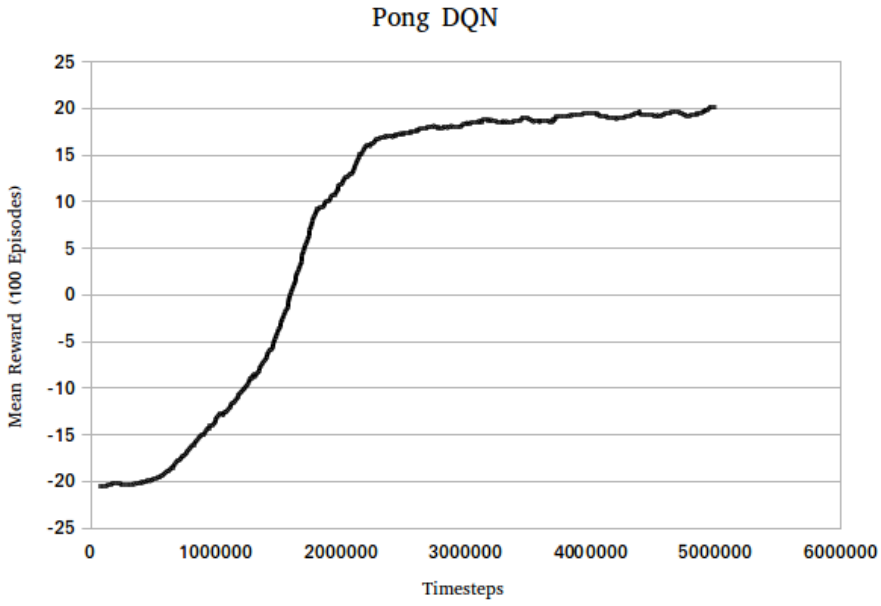

Figure 4-1. DQN agent learning Atari 2600 Pong [6].

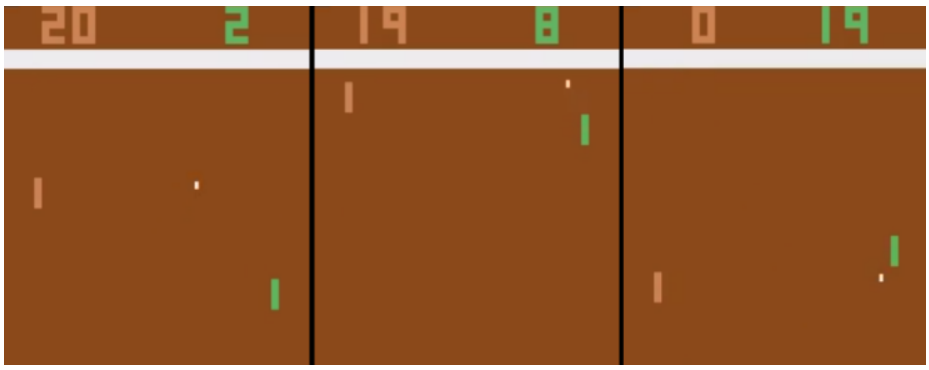

Figure 4-2. Progress of DQN agent learning Atari 2600 Pong at start, 1 Million Time Steps, 5 Million Time Steps, from left to right

This is incredibly impressive result considering that this is above the human expert players as well

as it outperforms the best linear reinforcement learning algorithms [5]. The performance of 49 games with DQN algorithm is well illustrated from [5]. It is clear from [5] that the performance of DQN agent varies significantly over games. This can be attributed to the fact that even the state of the art AI algorithms have a hard time with environments that requires long term planning. This in general can be argued as more of a difficult task because it requires more domain knowledge or other methods to combat the highly delayed reward. The combination of highly delayed reward as well as the sparsity of the reward signal can make it difficult for the DQN algorithm to learn.

4.1 Hyper Parameter Search for Pong

Tuning hyper parameters is a critical step in making the network work. Wrong hyper parameters can significantly hurt the convergence rate of the network as well as making the network diverge. Here we explore to see the effect of slightly different architectures. By tuning the kernel size, stride and output depth of the convolutional layers, the overall parameters can be reduced, which could allow faster convergence of the action-value function. More advance methods such as Batch normalization layers can be used as well to help with regularization and robustness.

First, we try the network with addition of batch normalization to see if that would improve the learning. Second we try a slightly different network that is not as wide but a bit more deeper to see what kind of result we might see. The networks are only trained for 1 million time steps due to computation constraints. The numbers for the convolutional layers are output depth, kernel size, stride and type of activation function respectively.

Table 4-1. Network Architectures

Network 0	**Network 1**	**Network 2**
CONV-32-8-4-RELU	CONV-32-8-4-RELU	CONV-32-4-2-RELU
CONV-64-4-2-RELU	BATCH NORM	BATCH NORM
CONV-64-3-1-RELU	CONV-64-4-2-RELU	CONV-32-4-2-RELU
FC-512	BATCH NORM	BATCH NORM
FC	CONV-64-3-1-RELU	CONV-64-3-1-RELU
	FC-512	BATCH NORM
	FC	CONV-64-3-1-RELU
		FC-256
		FC

5 Analysis

Looking at the graph, it is clear that there are no big differences in the performance of the networks in the first 600000 time steps. This is expected since the reinforcement learning algorithm had to

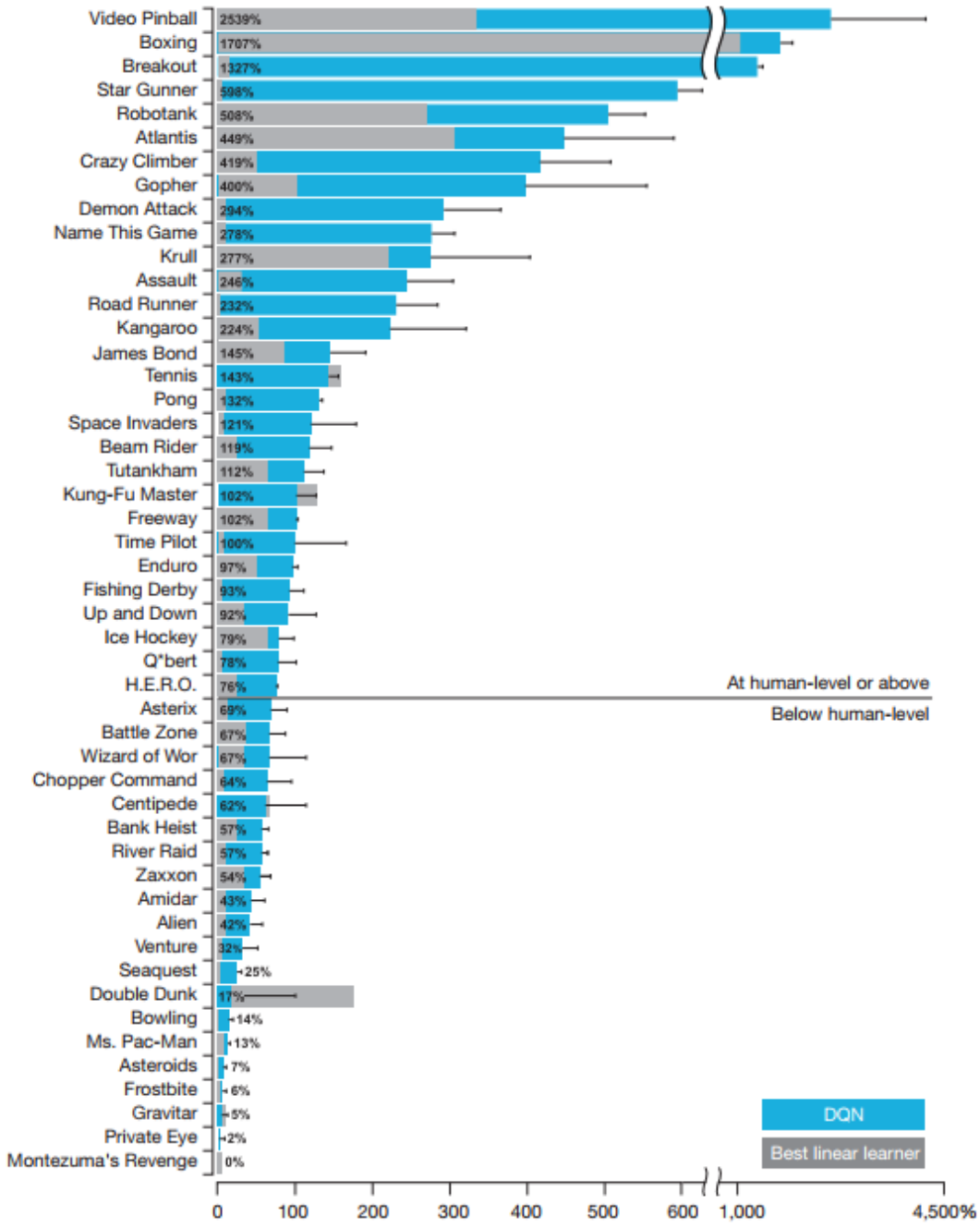

Figure 4-3. Comparison of the DQN agent with the best reinforcement learning methods in the literature [5].

explore the state space well enough to make any progress. We begin to see deviation around 900000 time steps when the network with just batch normalization does perform worse.

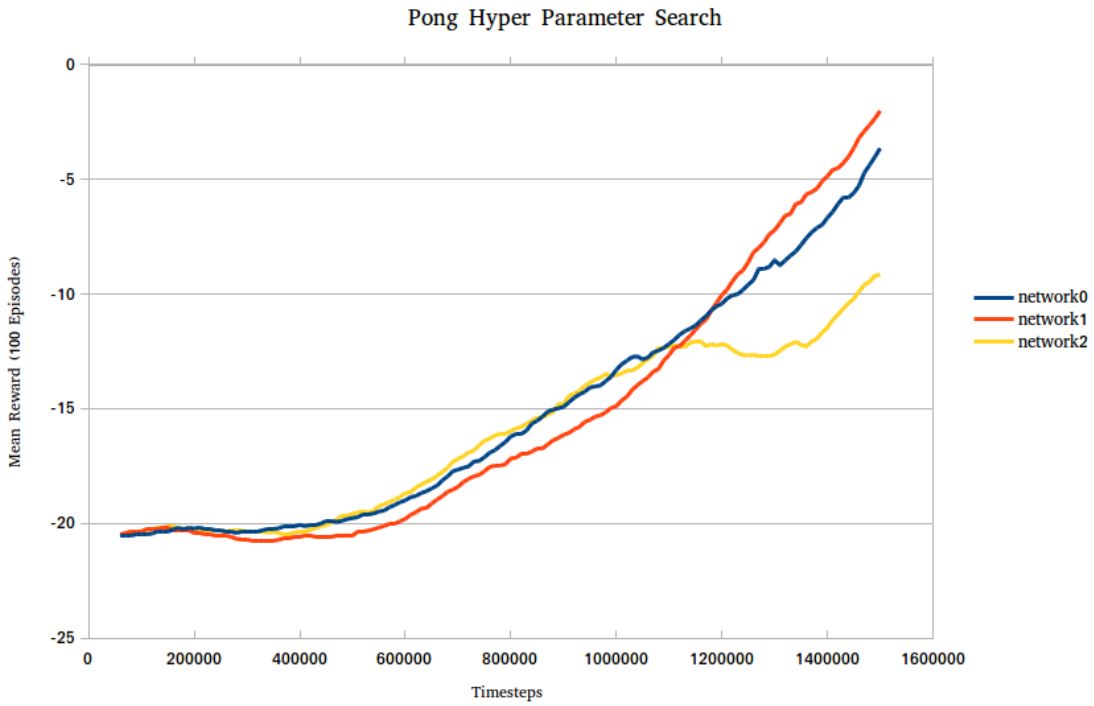

Figure 4-4. Hyper Parameter Search for Pong [6].

5.1 Batch Normalization

As expected the batch normalization network, the network 1, does better by 1.5 million time steps. The batch normalization allows the network to control the distribution of the input that it sees and helps the network learn faster, while improving robustness. But we don't see any significant improvements here. Given that batch normalization is mainly a method used to help deeper networks learn, it is likely that this network did not experience the same issues. More specifically, batch normalization is used to help solve the internal covariance shift problem, where during training the variance and mean are not consistent between each back propagation and in effect make learning more difficult due to a changing environment. If this was dealing with more complex images and required deeper convolutional networks to solve the task, it is likely that the batch normalization would help significantly.

5.2 Deeper Network

Deeper but shallower network is used in attempt to reduce the number of parameters. Also, now that we have implemented the batch normalization without a problem, it makes sense to try a deeper network with batch normalization. But since the original DQN agent is able to solve the pong completely, it's not the final performance that we would like to improve but the convergence rate.

So to reduce the parameter space, we reduce the convolutional layers as well as the fully connected layers. Unfortunately, this network was much worse in it's convergence rate. Up until 1.2 million time steps, the network showed very similar performance to the original DQN and sometimes out performing it slightly. After 1.2 million time steps, the network seems to be stuck at some local minima and the performance saturates. Fortunately, the network does recover its convergence rate around 1.4 million time steps. This is most likely due to the fact that the network is deeper with more non-linearity and some gradients may have been lost due to diminishing gradient problem. Also the increase in complexity of the network most likely didn't help with the convergence. Since the images for the pong is relatively simple, deeper network and its ability to have more complicated feature extractors may have not been needed. Overall, the deeper network has worse convergence rate and should not be used, at least not with this architecture and the training techniques presented here.

Conclusions

The first conclusion is that it has become very accessible for people to implement and test out some advanced artificial intelligence algorithms. This is very exciting since, all the software that is used in this work is open source or free to use. The computing power that is required to train these models are also not outside the normal computer's ability, making it feasible for people to train the networks. Another critical advances that needs mentioning is the widely available academic resources. Courses such as CS294 from Berkeley and CS231n from Stanford, allows people to get a formal introduction to the field and accelerates learning greatly. Combining the advances in computing, the freely available software for deep learning, and the open academic resources, it is possible for anyone to learn and try out some amazing algorithms in the exciting field of deep learning and deep reinforcement learning.

The second conclusion is regrading the hyper parameter search, more specifically trying out different network architectures. Through the results, it is clear that the batch normalization does not hurt the performance and it shows small improvement on the convergence rate. For more difficult tasks that require interpreting complex images, the batch normalization would most likely improve the performance significantly due to the reasons mentioned earlier.

The third conclusion is that the deeper network tried out in this work does not improve the convergence rate and is not recommended to be used. The original architecture is expressive enough, at least for the game of pong to give it high performance as well as respectable convergence rate with the training techniques use in this work

Recommendations

Based on the analysis and conclusions in this report, it is recommended that to increase the convergence rate of the network, smarter sampling from the experience replay buffer is probably required as mentioned in [6]. Prioritized replay that is used in the reinforcement learning literature is good choice and should make the more used of the data and hence learn faster. Another easy improvement may be using Double Q learning, which combats the problem of positive reward bias in Q learning algorithms [2]. For Atari games that require lone term planning, DQN may not be well suited. In that case, mix of model free and model based reinforcement algorithms are recommended with a mix of search algorithms such as Monte Carlo Tree Search method to utilize the model. As different architectures are explored, it would be smart to test the algorithm on different Atari games to make sure that the algorithm is still general purpose.

Note, while one of the models were trained on a GTX 1070, the rest of the networks were trained on a laptop with an old GPU, the Quadro K1000M. This GPU took five days to train the Pong DQN entirely and made development much more difficult. It may be an obvious point but if one can afford it, an investment to GTX 1080 TI or some other high end GPU is recommended. But the hope is that the rough numbers on the training time gives the readers an idea of the computing and cost trade off that needs to be made.

References

[1] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. ‘‘Deep Learning’’. Book in preparation for MIT Press. 2015. URL: http://www.iro.umontreal.ca/~bengioy/dlbook.

[2] Hado van Hasselt, Arthur Guez, and David Silver. ‘‘Deep Reinforcement Learning with Double Q-learning’’. In: *CoRR* abs/1509.06461 (2015). URL: http://arxiv.org/abs/1509.06461.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ‘‘ImageNet Classification with Deep Convolutional Neural Networks’’. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[4] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. ‘‘CS231n: Convolutional Neural Networks for Visual Recognition 2016’’. In: (). URL: http://cs231n.stanford.edu/.

[5] Volodymyr Mnih et al. ‘‘Human-level control through deep reinforcement learning’’. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. URL: http://dx.doi.org/10.1038/nature14236.

[6] Volodymyr Mnih et al. ‘‘Playing Atari with Deep Reinforcement Learning’’. In: *CoRR* abs/1312.5602 (2013). URL: http://arxiv.org/abs/1312.5602.

[7] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.