A Thorough Analysis of RL models plus Self-Attention DQN based on Atari's Space Invader

Yu-Chen Lin
UCLA CS Department
UID: 705315195
ericlin8545@cs.ucla.edu

Jo-Chi Chuang
UCLA CS Department
UID: 005350427
ashleychuang@cs.ucla.edu

Adrian Hsu
UCLA CS Department
UID: 405331564
adrianhsu@g.ucla.edu

Po-Chun Yang
UCLA CS Department
UID: 605297984
a29831968@g.ucla.edu

Yu-Chi Lee
UCLA CS Department
UID: 705331666
yuchi.lee0418@gmail.com

Abstract

Outer space is a place full of mystery and romance. Many people are attracted to the exploration of space. Inspired by SpaceX and NASA's successful astronaut launch recently, we would like to do a space game that can help craft automatically explore space with its own artificial life. In our project, we will adopt the classic Atari game space invader, where players use a craft to explore the space and defeat enemies in the space. In this project, **we proposed a new DQN architecture called Self-Attention DQN** that combine the Self-Attention techniques with current DQN model. We also analyze Self-Attention DQN and compare it with DQN, double DQN, and dueling DQN.

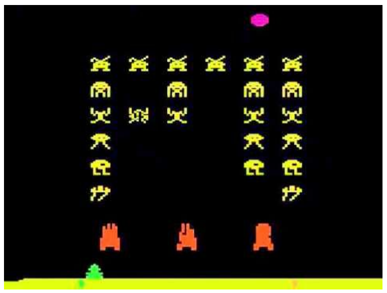
Figure 1: Game interface of Space Invader

1 Introduction

Space is always a charming and mysterious place that people want to explore. Thus, we use a game called space invader which players can control a spaceship to explore the space and defeat enemies as our project base. Space invader is a 1978 arcade game createdby Tomohiro Nishikado, and it was the first fixed shooter and set the template for the shoot 'em up genre. The interface of space invader is like the figure 1. Humans can play this game because we know what a spaceship is and we know what we should do to our enemies. However, if a computer wants to play this game, it's not so easy. The computer might not know what a enemy is and what should it do inside the game to achieve the ultimate goal. Thus, scientists started to find a way for computers to learn how to play the game based on the fact that computers don't posse real-world knowledge and only know the pixels on the screen and the score to go on.

Google Brain proposes a related paper to solve the above situation [2]. The researchers proposes a novel algorithm Deep Q-network (DQN) that use a specific network architecture to tune parameters only with the raw screen pixels, set of available actions and game score as input. In this research, DQN can outperform previous machine techniques in not only Space Invader, but also several games. And it also defeats most of professional human players.

After the DQN is proposed, lots of variants of DQN are conducted and proposed. For example, Double DQN and Dueling DQN modify the architecture of DQN and achieve different results. And in this project, we would like to propose our own architecture.

In this project, we design and implement **Self-Attention DQN**, which adopted the Self-Attention techniques taken from [8] into the DQN architecture and tried to use the characteristics of Self-Attention to gain better performance.

Besides Self-Attention DQN, we also survey and implement DQN, double DQN, and dueling DQN in the experimental section. After the experiments, we also do the analysis based on the metric like Q-value, average reward, and loss value.

The layout of this report is like the following. In the Chapter 2, we introduce the background knowledge of Q-learning, DQN, double DQN, dueling DQN, and Self-Attention. And in the Chapter 3, we explain how we design Self-Attention DQN and our motivation. Chapter 4 shows our experimental settings and we analyze the experimental results in Chapter 5. In the end, we have a brief conclusion on our project.

2 Background

Reinforcement Learning Machine learning techniques can be separated into 3 different categories - supervised learning, unsupervised learning, and reinforcement learning. For reinforcement learning, the learning models will continuous adjusted itself by interacting with the environment to achieve a goal or simply learning from reward and punishments. Generally, the components of reinforcement learning include *environment*, *action*, *state*, reward, *policy*, and *value*. *Environment* means the physical environment where the agents operates. *Action* is all the possible moves that the agent can make. *State* is the current situation returned by the environment or the state/situation where an agent is operating now. *Reward* is feedback from the environment on the work done, it can be either positive or negative. *Policy* is agents' strategy to determine the next action based on the current state. *Value* is the expected future reward that an agent would receive by taking action in a particular state/states. In reinforcement learning, agents normally used a Markov decision process (MDP) to make the decision. According to Wikipedia, the definition of MDP is a discrete-time stochastic control process that provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. Based on MDP, we can have the following equation:

$$V(s) = \max_a (R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s')) \tag{1}$$

$V(s)$ means the value in the state s, a means the current action, $R(s,a)$ means the reward when takes action a in the state s, s' means the potential future state, and γ means the discount factor. $P(s,a,s')$ means the probability of moving from room s to room s' with action a.

2.1 Q-learning

Q-learning is a basic form of Reinforcement Learning which uses Q-values to iteratively improve the behavior of the learning agent. It is based on Q-function, which measures the expected return or discounted sum of rewards obtained from currant state s by taking action a. In Q-learning [7], we start to consider the quality of an action that is taken to move to a state, and thus we will have a new equation like the following equation 2:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha(R(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a)) \tag{2}$$

where $Q_t(s, a)$ is the new Q-value and $Q_{t-1}(s, a)$ is the old Q-value. $\alpha$is the learning rate of Q-learning. The meaning of Q-value is an estimation of how good is it to take the action, it is also called action value.

2.2 Deep Q Network (DQN)

Reinforcement learning can be suitable to the environment where the all achievable states can be managed and stored in RAM memory. Nevertheless, in the environment where the number of states outnumber the capacity of contemporary computers such as Atari games, the standard Reinforcement Learning approach is not very applicable. Besides, the Agent needs to deal with continuous states, continuous variables and continuous control problems in real environment.

Thus, we don't want to keep the huge table and would like to replace it with Deep Neural Network [1]. In the deep neural network, it maps environment states to Agent actions, and learning is conducted during training phrase of this neural network. In short, the goal of DQN is to approximate a the nonlinear function Q(s, a) with a deep neural network.

During the training process, the agent interacts with the environment and receives data, and this data is used to learn the Q network. In the beginning, the agent would randomly choose the actions to explore the environments and build a complete picture of transitions and action outcomes. After a while, the agent will start

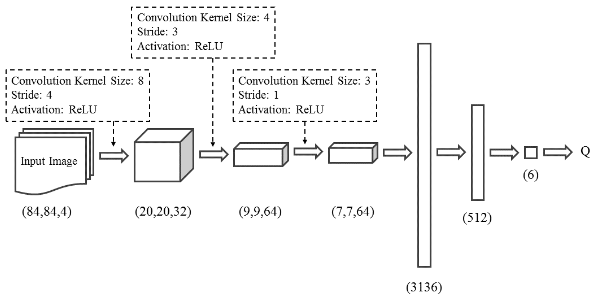

Figure 2: The architecture of a DQN model.

to look on Q-network to decide its action. And this method is called epsilon-greedy method that means to switch between random and Q policy with the use of probability hyper parameter epsilon. In DQN, the loss function is set as the squared difference between the target and predicted value. By updating the weights, we can try to minimize the loss. During the learning, we have two separate Q networks to calculate predicted value and target value respectively. For the target network, it's updated once for several time steps by copying the weights from the actual Q-network. In this way, the training process can be more stable.

In DQN, the Q-learning use the following loss function to update

$$Loss = (\gamma + \gamma max_{a'} Q(s', a'; \theta')) \tag{3}$$

where theta means the weight in the network.

2.3 Double DQN

In previous section, we talked about the main components in DQN, which is the combination of Q-learning and deep neural network. However, Q-learning is known for its overoptimism on value estimation, which were first investigated by Thrun and Schwartz[3]. In Q-learning and DQN, the *max* operator always picks the largest value, and uses the same value to both select and evaluate an action. That is, if some action values are overestimated at a current state, the *max* operator will take the largest one, which is also an overestimated value, as Q-value. Therefore, we can conclude that Q-value in Q-learning is overoptimistic on value estimation since it tends to select the action that is overestimated.

In order to mitigate the problem, we can use two Q networks for selection and evaluation separately, in order to prevent overoptimistic value estimates. This is the idea behind Double DQN[4]. The first network is used for selecting the best action with maximum Q-value. The other network is used for calculating the estimated Q-value with action selected above.

The update to the target network is the same as for DQN. The algorithm details is shown as below,

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \max_a Q(S_{t+1}, a; \theta_t), \theta_t^-) \tag{4}$$

These two Q networks share the same structure but different values of parameters, which is denoted as θ_t and θ_t^-. θ_t is the weights of the selection network, and θ_t^- is the weights of the target network. Double DQN can be used at scale to successfully

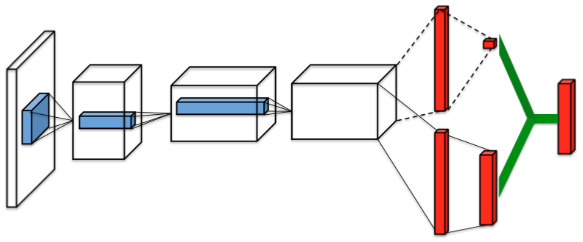

Figure 3: Dueling Architecture. Figure is from the paper[6]

reduce this overoptimism. If the selection network overestimate a certain action, as long as the target network does not overestimate on the same action, it will give it a proper value. On the other hand, even if the target network overestimate a value of an action, eventually Q-value should be fine as long as the selection network does not pick that action.

2.4 Dueling DQN

There is another algorithm called Dueling DQN, which only changes the structure of the network from DQN. This new architecture is named *dueling architecture*, proposed by Wang et al.[6]. The key motivation behind this new architecture is that, it is unnecessary to know the value of each action at every timestamp as DQN.

The normal DQN consists of convolutional layers followed by a single sequence of fully connected layers. In DQN, the output of the last layer of fully connected layer is the final Q-value. However, in dueling architecture, as shown in figure 3, they split the network into two streams, one for estimating the state-value and the other for estimating state-dependent action advantages. Finally, these two streams are combined to produce a single output function:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{5}$$

V is a scalar, which denote the state-value. A is state-dependent action advantages, denoted as a vector. θ denotes the parameters of the convolutional layers. We can look at the equation 5 that both V and A have the same θ, which means they share the same convoluational layers exactly as shown in the figure 3. α and β are the parameters of the two streams of fully connected layers.

The main benefit of dueling DQN is the effectiveness to update Q function by adjusting V function, rather than updating the whole A function. However, this method has some downsides. One is that, the naive sum of the two can be unidentifiable. For instance, if V turns out be zero for all actions, then final Q is simply A vector, which does not take any advantage of dueling DQN. Therefore, the last module of the neural network implements forward mapping shown below:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a' \in |A|} A(s,a';\theta,\alpha)) \tag{6}$$

Alternatively, we use the following module for implementations, which has the same effect as equation 7:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\Sigma_{a'} A(s,a';\theta,\alpha)) \tag{7}$$

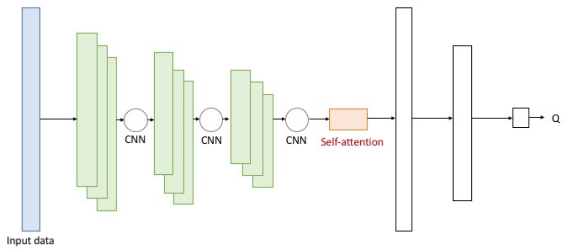

Figure 4: Architecture of Self-Attention DQN

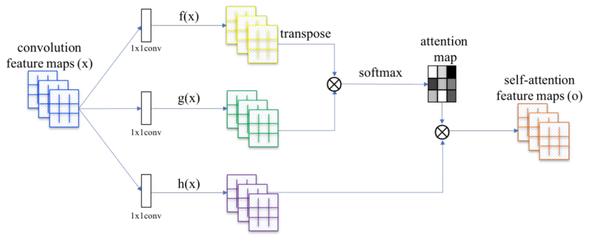

Figure 5: Self-attention module for the DQN, taken from [8]

2.5 Self-Attention

Self-Attention is simply a vector, often the outputs of dense layer using softmax function [5]. But with Self-Attention, the machine can look over all the information in the input data, and generate the output based on current input data and the context. Using the concept of Self-Attention, we can conduct Self-Attention. Self-Attention is compression of Self-Attentions toward itself. Compared to RNN, Self-Attention has the advantage of parallel computing. When comparing wit CNN, Self-Attention doesn't need to use deep network. Also Self-Attention is mainly applied in Natural Language Processing (NLP) field, we would like to try it in our project.

3 Methodology

In out project, **we proposed a new architecture based on DQN called Self-Attention DQN**. The architecture of Self-Attention DQN is like Figure 4. Self-attention calculates the response at a position in a sequence by attending to all positions within the same sequence. [8] In spite of this progress, self-attention has not yet been explored in the context of DQNs. Figure 5 demonstrates our self-attention module for the DQN.

We will add a Self-Attention layer between CNN layer and the fully-connected layer. The reason that we want to add a Self-Attention layer is that we think Self-Attention can help the model to consider the data in the whole environment rather than the local region. Armed with Self-Attention, the generator can draw images in which fine details at every location are carefully coordinated with fine details in distant portions of the image. In this way, it can have broader vision and might have better performance.

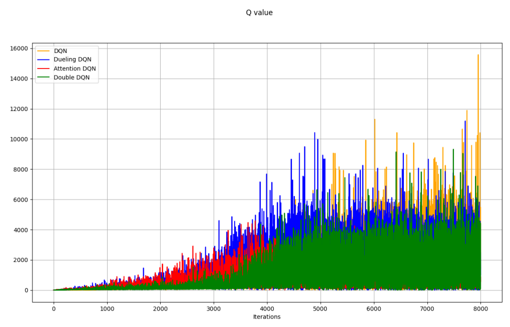

Figure 6: Q values of DQN, Double DQN, Dueling DQN, and Self-Attention DQN.

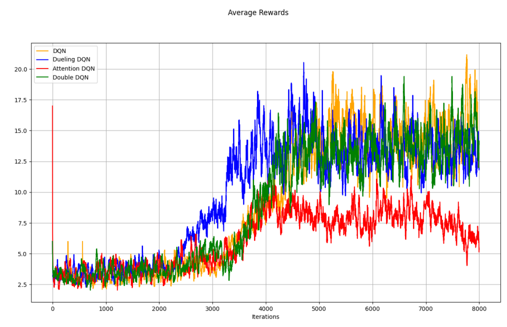

Figure 7: Average rewards of DQN, Double DQN, Dueling DQN, and Self-Attention DQN.

4 Experiments

First, we would like to introduce the environment where we run all of our experiments. In our device, the CPU is AMD Ryzen 7 1700 Eight-Core Processor, the GPU is GeForce GTX 1080 Ti, and the memory of RAM is 16GB.

We implemented DQN, Double DQN, Dueling, and Self-Attention DQN with Tensorflow. Our implementation is based on hsuanchuu's 2019Playing-Space-Invaders-with-Deep-Q-Networks repository on the github and we also created our own structure. The version of Tensorflow that we use is 1.6.0, and the version of Python is 3.6.4.

The DQN model in our experiment is shown in figure 2. The input data is the image from the game, and it is resized to 84x84x4. The model consists of a 5-layer networks, which are three convolution layers with ReLU activation function followed by two fully connected layers. In the first convolution layer, the kernel size is set to 8 with stride = 4. The output dimension of the first layer is 20x20x32. The second convolution layer has kernel size set to 4 and stride to 2, with output dimension 9x9x64. In the last layer, we have kernel size = 3, and stride = 1. The output dimension of the last convolution layer is 7x7x764. In double DQN model, we created two networks to decouple the selection from the evaluation. Basically, the experimental settings are all the same as the DQN model.

As for dueling DQN, instead of a single sequence of fully connected layers after three convolution layers, we add two streams of layers. As shown in figure 3, we add a layer for state-value in one stream, and a layer for state-dependent action advantages in the other stream.

In our training process, we use two Q networks. One is *target network*, the other is *evaluation network*. The *evaluation network* will compute the loss for every action; therefore, its parameters could not update during this whole process. In order to avoid the network falling into the feedback loops between the target and estimated Q values, we use *target network* to keep training and updating the parameters, where the model is set to *trainable*. We assign value to the *target network* every 5000 frames from the *evaluation network*.

5 Results

In order to evaluate the results, we use three different metrics for evaluation, which are Q-value, *average rewards*, and *loss*. The models that we have tested are DQN, Double DQN, Dueling DQN,

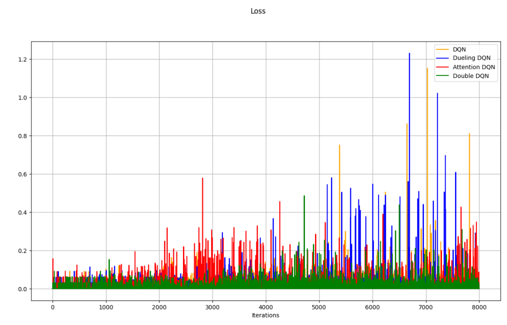

Figure 8: Loss of DQN, Double DQN, Dueling DQN, and Self-Attention DQN.

and Self-Attention DQN. We play game with each of the models for 100 times, and compare the results from different models.

The experiment results are shown in figure 6, 7 , and 8. In figure 6, the yellow line is distribution of Q-value for DQN model, while the green line is the Q-value for Double DQN model. We can see that the Q-value of Double DQN model is generally the lowest, and the value of DQN model tends to be higher. Moreover, the value of DQN is constantly climbing as the iterations increase. This implies that DQN is vastly overoptimistic about the value of the current greedy policy. Overall, this figure shows the DQN's overoptimistic value estimates, and how Double DQN mitigate the problem.

Figure 7 shows the rewarding curve of each of the structures. By comparing the average rewards over DQN and Dueling DQN, we notice that, in the iterations between 2500 and 4000, the curve for Dueling DQN rises quicker than DQN. It implies the dueling architecture's ability to lean the state-value efficiently. This figure shows the benefit of Dueling DQN of being effective to update Q function by adjusting state-value function, rather than updating the whole Q function for every single action as DQN does. Therefore, it proves that by applying dueling architecture, the learning agent can learn faster and better.

Figure 8 illustrates the loss distribution over 8000 iterations for each model. The loss function consists of the mean square of the difference between target Q-value and predicted Q-value. The target Q value is decomposed by the addition of actual reward and the original Q value multiplies decay factors, while the predicted Q value is the prediction of our neural networks. However, the value of the reward in the actual Q varies in each state and actions. To be more specific, in the game of space invader, players score higher when shooting space ships, and score lower when shooting aliens. Therefore, as the games proceeding, the reward for each action fluctuates. But we could notice that, as it iterates from 2000 to 4000 iterations, the actual reward of each action seems to get higher, which implies that the learning agent started to learn from playing the game. The red line denotes the loss for Self-Attention DQN model. Its loss starts to get higher from 2000 iterations, which rises the fastest among all the other models. The indicates that Self-Attention DQN can start to learn from playing the game in the least time.

6 Conclusion

From our experiments, we can see that Self-Attention DQN can learn faster than other models. According to the definition of Self-Attention, the reason might be that the Self-Attention layer can help the Self-Attention DQN model learn from the whole environment rather than local region. Thus it can get more information than other models in a short time. Although the concept of Self-Attention is mostly used in NLP field, more and more researcher would like to apply this concept on the image research. Maybe in the future, we can use Self-Attention to build a image model or even video model that could extract information across different time. And this project is just one of the works inside this trend.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. (2013). http://arxiv.org/abs/1312.5602 cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[3] Sebastian Thrun and Anton Schwartz. 1993. Issues in Using Function Approximation for Reinforcement Learning. In *In Proceedings of the Fourth Connectionist Models Summer School*. Erlbaum.

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

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[6] Ziyu Wang, Nando de Freitas, and Marc Lanctot. 2015. Dueling Network Architectures for Deep Reinforcement Learning. *CoRR* abs/1511.06581 (2015). arXiv:1511.06581 http://arxiv.org/abs/1511.06581

[7] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292. https://doi.org/10.1007/BF00992698

[8] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. 2019. Self-Attention Generative Adversarial Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 7354–7363. http://proceedings.mlr.press/v97/zhang19d.html