Look, Attend, Play: An Attempt to Combine Selective-Attention with Deep Reinforcement Learning Model

Haoyang Pei *hp2173*, Jiajin Liu *jl11523*

I. Introduction

SELECTIVE-ATTENTION is a visual psychological phenomenon that lets us remain focused on important parts of our world without distraction from irrelevant details. Inspired by this phenomenon, we attempt to apply the property of it into the deep reinforcement learning model. It can allow us to constrain the visual input of the deep reinforcement learning model with fewer learnable parameters, while the model performance will be improved since it does not take some irrelevant information to make decisions. More specifically, it can generalize to environments where task-irrelevant elements are modified while conventional methods fail. In this project, we will combine the selective-attention mechanism with deep Q-network to build a attention-based DQN model to eliminate the irrelevant visual input of DQN, thus to improve the performance of DQN. We test our attention-based DQN model on three Atari series games, including breakout, beam rider, and pong.

II. Literature Survey

According to our idea, we found three related pieces of literature and references.

1) Neuroevolution of Self-Interpretable Agents[1]: The author uses input transformation, patch selection, and action generation to train a selective-attention-based RL model and does experiments on CarRacing and DoomTakeCover, which have good performance even using a small LSTM controller. Here is the original experiment: CarRacingExperiment[2].

2) An initial attempt of combining visual selective attention with deep reinforcement learning[3]: The author visualizes and analyzes the feature maps of DQN on a toy problem Catch, and proposes an approach to combine visual selective attention with deep reinforcement learning. Experiments are based on optical flow-based attention and A2C on Atari games, with improvement of sample efficiency on tested games using visual selective attention.

3) Human-level control through deep reinforcement learning[4]: In this paper, the author builds a Deep Q-network to play Atari series game and outperform the human-level score for those games. Our models will base on this paper to do some modifications, including preprocesses the environment.

III. Technical Details

A. Dataset

Our project will use OpenAI gym environment to do experiment and test our model. In each environment, the observation is an RGB image of the screen. The specific environments are shown in Table. I:

Note: Since the DQN is not applicable to the environment with the continuous actions, we only implement the idea of the paper[1] on the 'CarRacing-v0' environment and test DQN and Attention DQN on the 'Breakout-v0', 'Pong-v0', and 'BeamRider-v0' environment.

TABLE I: Environment Summary

No.	Environment Name	Description	Link
1	CarRacing-v0	Easiest continuous control task to learn from pixels, a top-down racing environment, wher point grows with the frames goes by and decreases when the car is out of the designed way. The observation is an array of shape (3, 96, 96). The goal is to train a RL model to control the car on the fixed road to get at least more than 900 scores.	https://gym.openai.com/envs/CarRacing-v0/
2	Breakout-v0	The player gains points by destroying the blocks using the ball and losses the game run when the ball does not pass the paddle. The observation in this environment is an array of shape (210, 160, 3). To train a RL agent to get at least 20 scores in total within 5 continuous runs.	https://gym.openai.com/envs/Breakout-v0/
3	Pong-v0	A simple "tennis like" game with the goal to catch the ball and defeat the opponent by being the one gain more points within continuous 12 game runs. The observation in this environment is an array of shape (210, 160, 3). To train a RL agent to get 20 scores. (always win).	https://gym.openai.com/envs/Pong-v0/
4	BeamRider-v0	Game points are gained when targets are shot with a scrolling shooter and the game run is over when the target reaches the shooter. The observation in this environment is an array of shape (210, 160, 3). To train a RL agent to gain 400 scores with 3 game runs.	https://gym.openai.com/envs/BeamRider-v0/

B. preprocessing the Environment

In order to get better performance and reduce the computation cost, we preprocesses the environment before training. The preprocessing details are as follows, and we use the Openai official wrappers in [5] to do following preprocessing steps.:

- Convert RGB representation to gray-scale and down-sampling it to a 8484 image.[4]
- Clip the reward between [-1,1].we clipped all positive rewards at 1 and all negative rewards at 21, leaving 0 rewards unchanged. Clipping the rewards in this manner limits the scale of the error derivatives and makes it easier to use the same learning rate acrossmultiple games.[4]
- Concatenate most recent 4 frames and stacks them to produce the input to the Q-function. It ensures the network to make decision according to multiple frames but not one single frames.[4]
- Use 'NoFrameskip-v4' for each environment but not the original environment since it is easy to train by using 'NoFrameskip-v4'.

C. Model

There are 3 kinds of model we implemented.

1) The Model in the Paper [1]: The model first resizes the input image which shape is $96 \times 96 \times 3$, and then segments it into 529 patches with the sliding window size of 7 and the stride size of 4, each of which is a $7 \times 7 \times 3$ patch. Then, put the patches into the selective-attention network to process. In the selective-attention network, it first projects the $7 \times 7 \times 3$=147 pixels of each patch into 4 values, and make a multiplication operation with its transpose matrix, which is just like the attention mechanism discussed in the lecture. However, after softmax on columns and sum on rows, this model will sort the value of these 529 rows(patches) in increasing order, and select the top 5 to do further process. It maps the rank index with the corresponding position and flatten it into a LSTM network to predict the action. The details of the model are shown in Fig.1 and Fig.2.

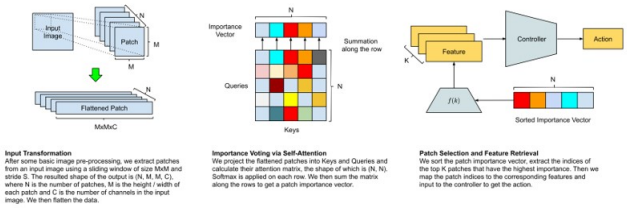

Fig. 1: The overall step of the model paper[1] mentioned

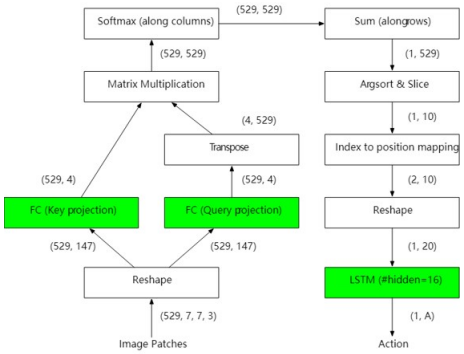

Fig. 2: The architecture of the model paper[1] mentioned

2) DQN Model: For Atari series games, the input is the $4 \times 84 \times 84$ (4 continuous frames, Height=84 , Width=84)image of the game after preprocessing. The output is the Q-values of each actions and we choose the optimal action according Q-values. The Deep Q-learning algorithm is shown in Fig.3 and the architecture of the Q-Network is shown in Fig.4.

3) Attention DQN: We propose 2 tentative Attention DQN model and test them on the same games.

a) Method 1: We first get patch-based flatten images to compute the attention map and do importance voting to get the relative importance of each patches. Then, we reshape the importance vector to 27×27. Finally, we feed the reshaped result to the decision layer to get the Q-Values of each action. The details are shown in Fig.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 $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Fig. 3: The algorithm of the Deep Q-learning

Like DQN, we will also use Q-target network and Experience Replay

Convolutional 2D, 4, 32, 8x8 kernels, stride 4, ReLU activation
Convolutional 2D, 32, 64, 4x4 kernels, stride 2, ReLU activation
Convolutional 2D, 64, 64, 3x1 kernels, stride 1, ReLU activation
Flattening
Dense, N_flatten*512 , ReLU activation
Output: Dense, 512*N_actions

Fig. 4: The architecture of the Q-Network

b) Method 2: We first get patch-based flatten images to compute the attention map and do importance voting to get the relative importance of each patches. Then, we reshape the importance vector to 27×27. Then, we use the transpose convolution layer to upsample the reshaped importance vector to the size of the input frame and multiply it with the input frames(element-wise). Finally, we feed the result to the decision layer to get the Q-Values of each action. The details are shown in Fig.6.

D. Evalution Metrics

We would like to use 4 metrics to evaluate our models.

1) Reward Per Episode/Life: The average reward the agent get per life.

2) Loss: The huberloss between the Q-target and Q-eval.

3) Initial State Q Value: The first maximum Q-values when one episode/life of the game begins.

4) Gradient Norm: The gradient norm after clipping. We are able to judge the gradient exploding or varnish during training by observing the Gradient Norm.

E. Hyperparameters Used

a) Model in the Paper[1]: The details are shown in Table.II

b) DQN, Attention DQN with Method 1, and Attention DQN with Method 2: The details are shown in Table.III. For some games, it does not need 3e6 total steps to train, therefore we choose a dynamic total steps to save the training time.

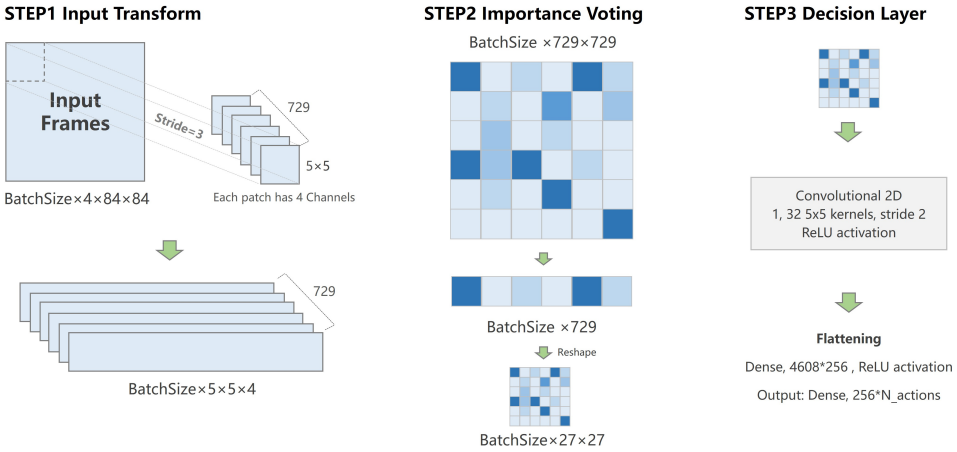

Fig. 5: The architecture of the Attention DQN with method 1

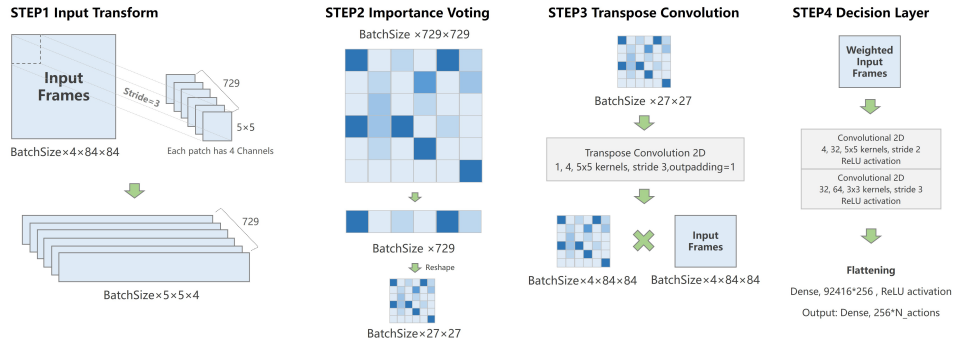

Fig. 6: The architecture of the Attention DQN with method 2

TABLE II: Hyperparameters for Model in the Paper[1]

Image size	96×96
Patch Size	7×7
Patch Stride	5
Selective Top K patches	5
Data Dim	3
Query Dim	4
Number of Hidden for LSTM	[16,]
Output Dim	3
Training Algorithm	CMA-ES with Population size=256
Max Iteration	2000

TABLE III: Hyperparameters for DQN, Attention DQN with Method 1, and Attention DQN with Method 2

Batch Size	32
Total step	1e6 - 3e6
Decay Slope	1e6
Init Epsilon	1
Final Epsilon	0.1
Loss Frequency	50
Target Update Frequency	5000
Evaluation Frequency	5000
Max Grad Norm	50
Loss Function	Huber Loss
Training Algorithm	Adam
Learning Rate	1e-4

F. Training

- Due to the hardware resource limitation, we are not able to finish training for the model mentioned in the paper[1]. CMA-ES needs long time and ram to train and their model is trained on the clusters.
- DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models are trained on the New York University High Performance Computing Greene using RTX8000 GPU, 8 cores CPU, and 32GB RAM. It costs roughly 20-30 hours for each model. After training, we compare the performance of different models in terms of different metrics. The results are shown in the Part.IV.

IV. Results

In this section, we only show the results of DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models on three Atari games. The results of the model mentioned in the paper[1] are not shown here. If you are interested in, please see it in the paper[1].

A. The curves of metrics

a) Breakout: The mean reward curve is shown in the Fig.7. The Loss curve is shown in the Fig.8. The Initial Q Value curve is shown in the Fig.9. The grad norm curve is shown in the Fig.10. We see that the general DQN model has the best performance and the attention DQN with method 2 also get a better performance, especially in the beginning of training. The loss of attention DQN with method 2 is lower than others. After training, the initial Q value of different model are very close. Attention DQN with method 1 has a larger gradient norm.

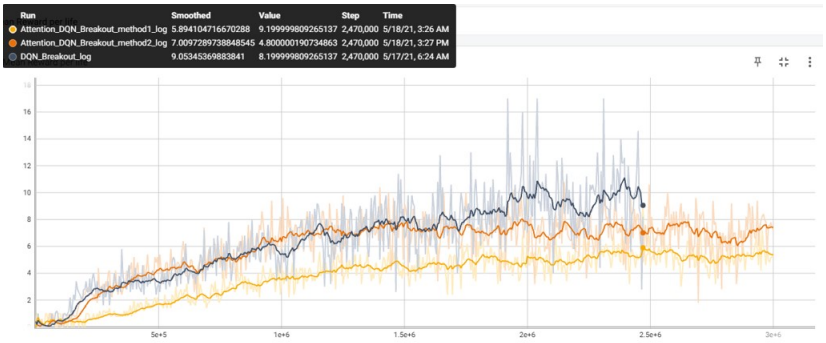

Fig. 7: The Mean Reward per Life for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

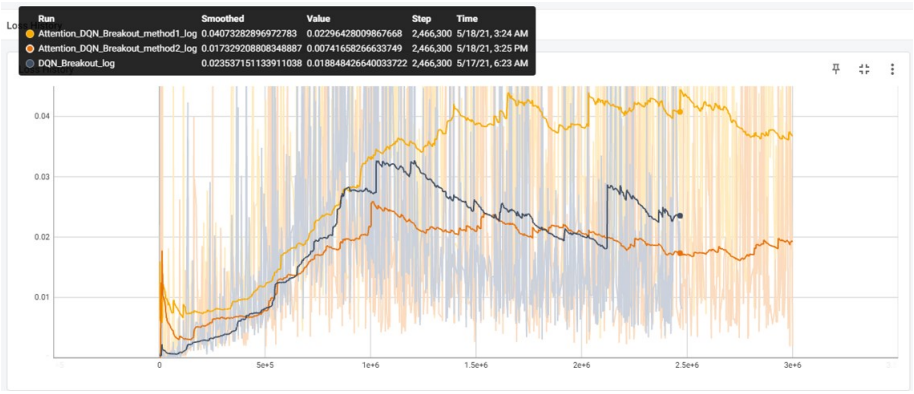

Fig. 8: The Loss for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

b) Beam Rider: The mean reward curve is shown in the Fig.11. The Loss curve is shown in the Fig.12. The Initial Q Value curve is shown in the Fig.13. The grad norm curve is shown in the Fig.14. We see that the attention DQN with method 2 has the best performance and the general DQN model also get a better performance, especially in the beginning of training. The loss of attention DQN with method 1 and 2 are lower than the DQN model. After training, the initial Q value of DQN model is higher than others. DQN model has a larger gradient norm.

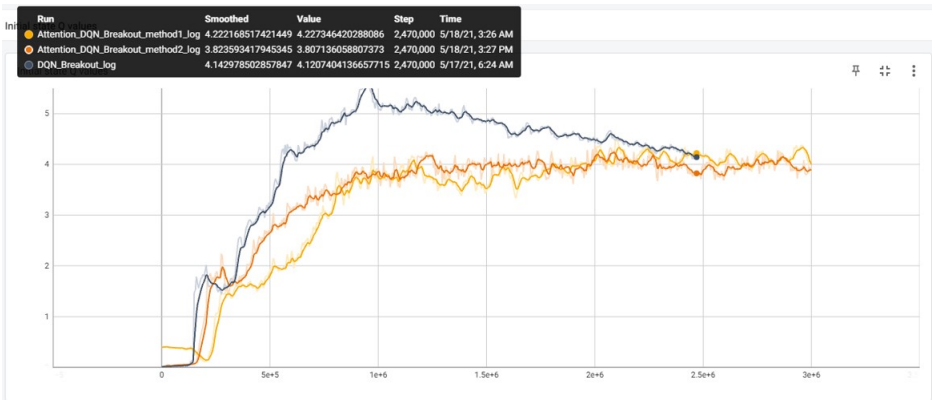

Fig. 9: The Initial Q Value for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

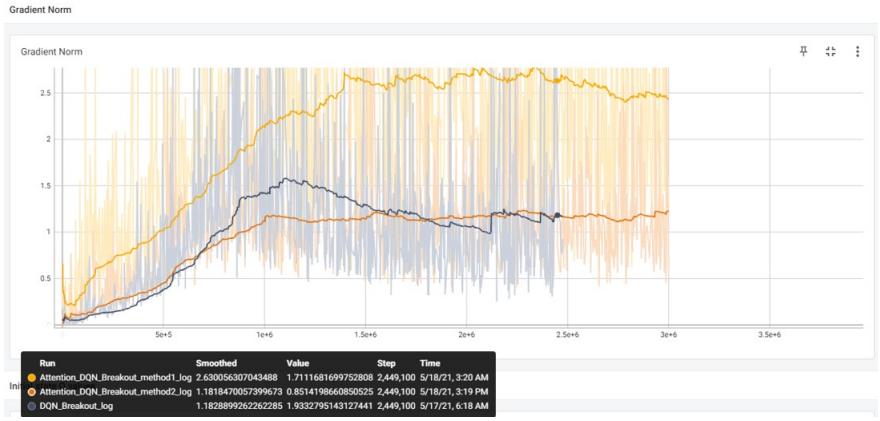

Fig. 10: The Gradient Norm for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

c) Pong: The mean reward curve is shown in the Fig.15. The Loss curve is shown in the Fig.16. The Initial Q Value curve is shown in the Fig.17. The grad norm curve is shown in the Fig.18. We see that the attention DQN with method 2 has the best performance and get the highest reward faster than other models, and the general DQN model also get a better performance. The loss of attention DQN with method 2 and DQN model are lower than attention DQN with method 1 in the begining of training. After training, the initial Q value of DQN model is higher than others. The gradient norm of three models are close.

B. Final Reward

We run each trained model on three game 30 times and compute the average score for three games with different models. The results are shown in the Table. IV.

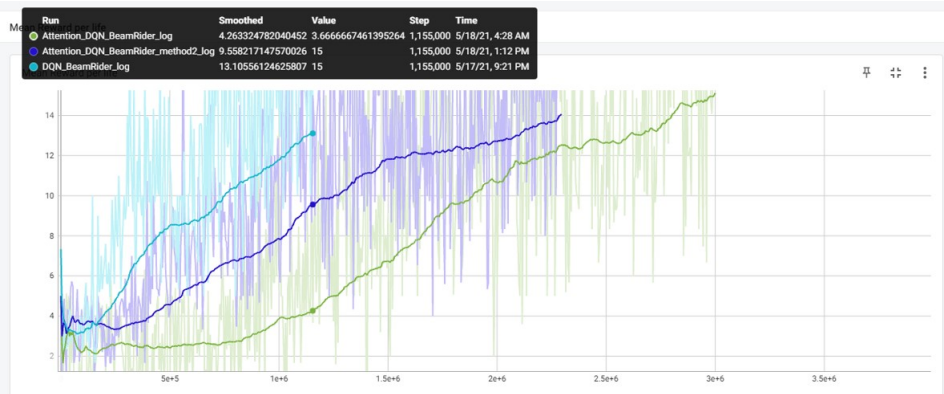

Fig. 11: The Mean Reward per Life for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

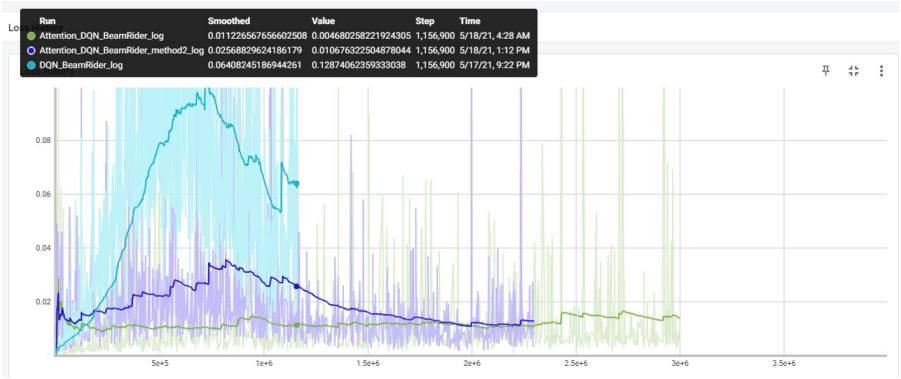

Fig. 12: The Loss for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

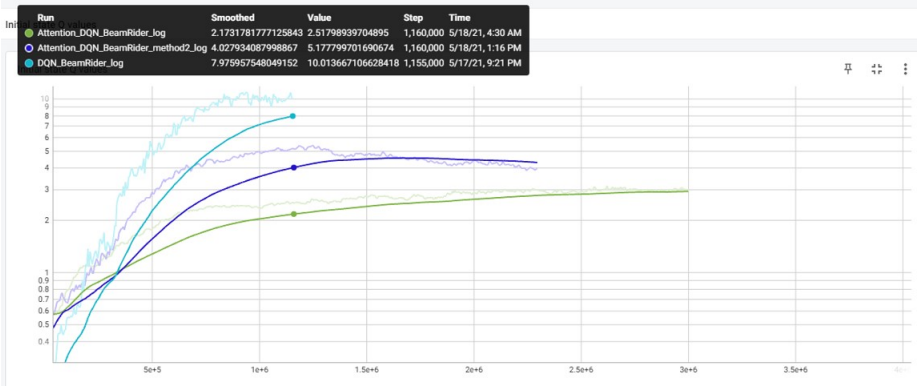

Fig. 13: The Initial Q Value for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

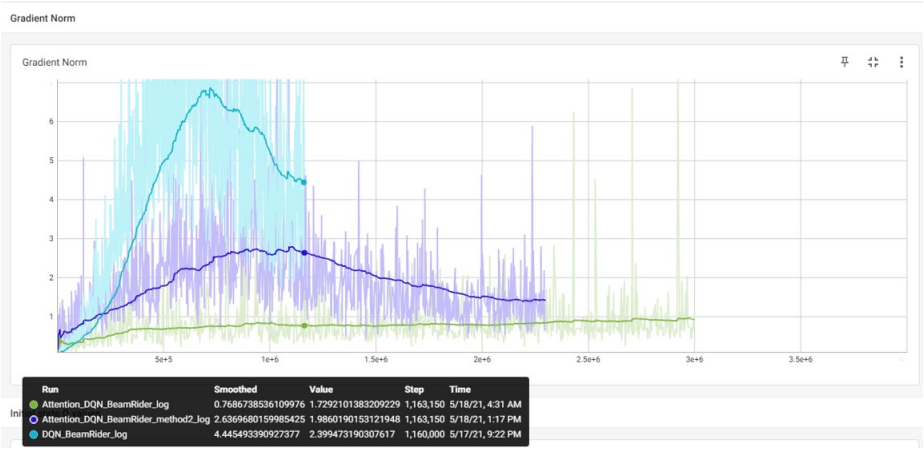

Fig. 14: The Gradient Norm for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

TABLE IV: Final Reward For Three Games with Different Models

	DQN	Attention DQN method1	Attention DQN method2
Breakout	140	42	120
Beam Rider	2496	4871.8	2748
Pong	21	21	21

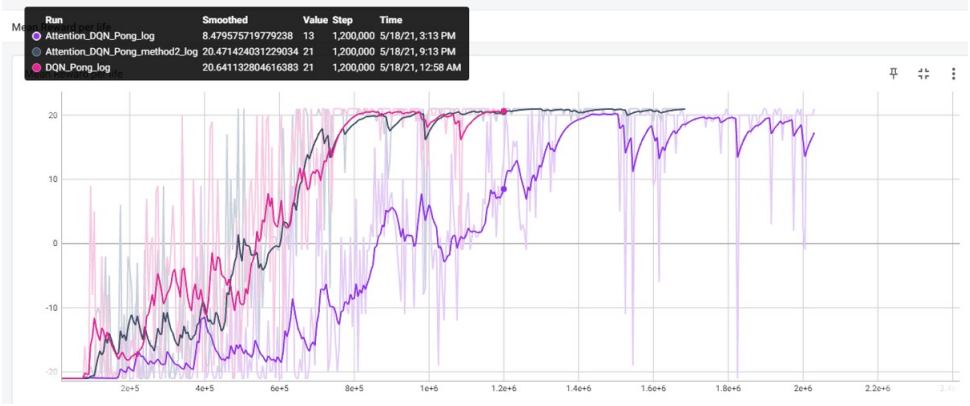

Fig. 15: The Mean Reward per Life for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

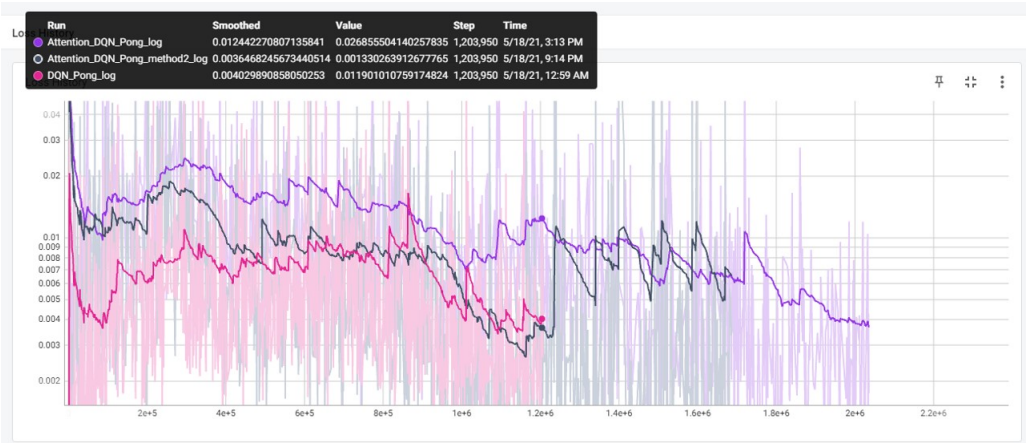

Fig. 16: The Loss for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

C. Attention Layer Display and Explanation

We choose the top 10 important areas and the last 10 important areas to display and to see what the attention layer attends in different games.

a) Breakout: According to the Fig.19 and Fig.20, we see that the attention layer attends to the ball and the board for beating the ball in the game. And the least important area is the wall. It is clear that the model make decisions using the information(location, speed) of the ball and the location of the board.

b) Beam Rider: According to the Fig.21 and Fig.22, we see that attention layer attends to the beam rider itself, the beam, the shot goals, and its remaining life. The least important area is the background.

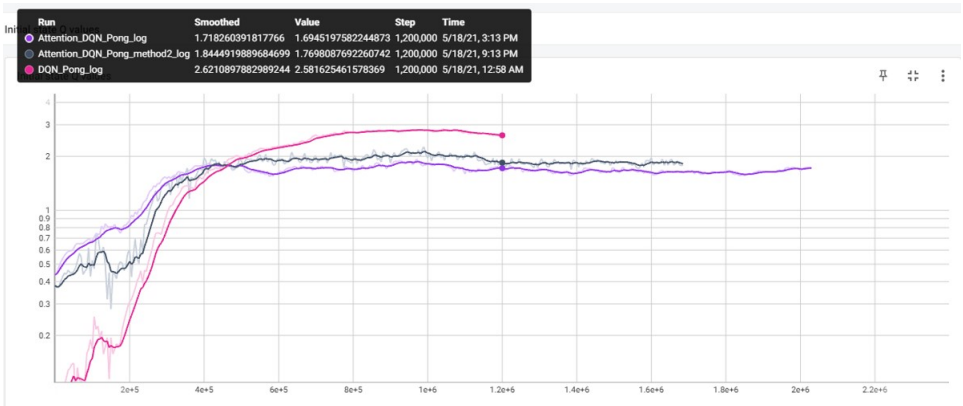

Fig. 17: The Initial Q Value for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

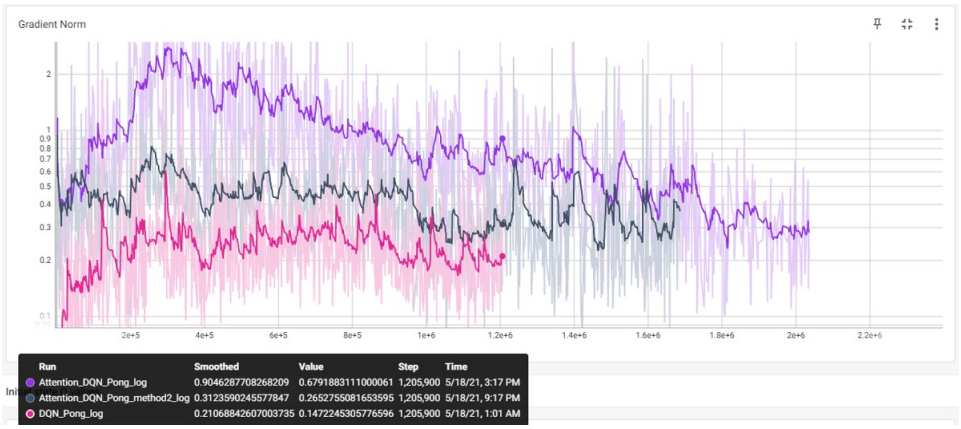

Fig. 18: The Gradient Norm for DQN, Attention DQN with Method 1, and Attention DQN with Method 2 models

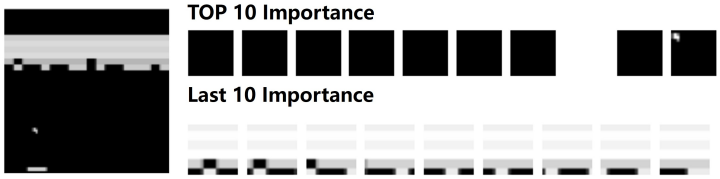

Fig. 19: Top 10 important areas and the last 10 important areas in Breakout with Attention DQN method 1

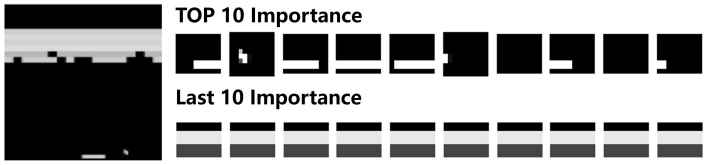

Fig. 20: Top 10 important areas and the last 10 important areas in Breakout with Attention DQN method 2

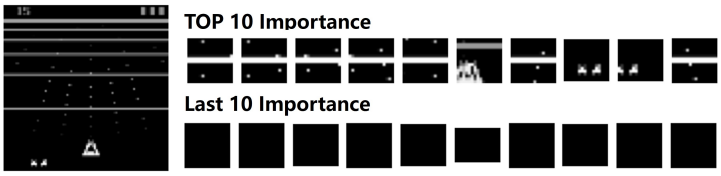

Fig. 21: Top 10 important areas and the last 10 important areas in Beam Rider with Attention DQN method 1

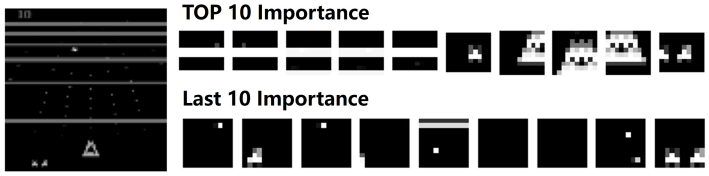

Fig. 22: Top 10 important areas and the last 10 important areas in Beam Rider with Attention DQN method 2

c) Pong: According to the Fig.23 and Fig.24, we see that attention layer attends to the racket itself and the ball. The least important area is the background. The selective areas that Attention DQN with method 1 attends to are not appropriate according to Fig.23.

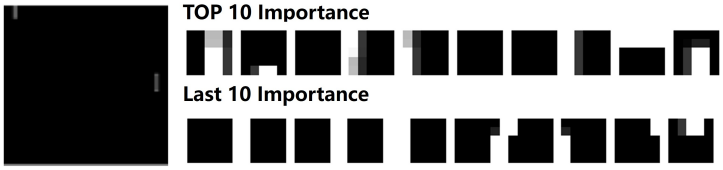

Fig. 23: Top 10 important areas and the last 10 important areas in Pong with Attention DQN method 1

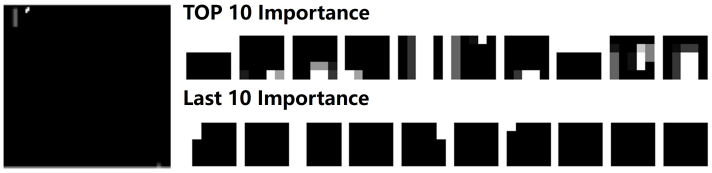

Fig. 24: Top 10 important areas and the last 10 important areas in Pong with Attention DQN method 2

V. Conclusions

A. Experiment Conclusion

In conclusion, from the above results record, we see that selective attention models can help the training process based on DQN. After adding attention layers, the model can actually be focused on those important areas if the model is well-trained, thus to improve the performance of the model. Specifically, The attention DQN with method 1 performs very well in two games(Beam Rider, Pong). Especially for Beam Rider, it can achieve double scores of regular DQN network.

B. Project Summary

We successfully combine the Selective-Attention model with DQN by designing the attention-DQN and it performs well on the game 'Pong' and 'Beam Rider'. However, since the time is limited, we do not have enough time to fine tune our model and test it on more Atari games. We also tried to train our model on the 'skiing' and 'Car Racing' planed before, but the results are not good thus we do not display it in our report. In the future, we will continue to modify our attention-dqn model to make it more general for more games.

C. Source Code

All experiment notebooks are published in the https://github.com/HaoyangPei/Attention-DQN

References

[1] Tang Y, Nguyen D, Ha D. Neuroevolution of self-interpretable agents[C]//Proceedings of the 2020 Genetic and Evolutionary Computation Conference. 2020: 414-424.
[2] https://github.com/google/brain-tokyo-workshop/tree/master/AttentionAgent
[3] Yuezhang L, Zhang R, Ballard D H. An initial attempt of combining visual selective attention with deep reinforcement learning[J]. arXiv preprint arXiv:1811.04407, 2018.
[4] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.
[5] https://github.com/openai/baselines/blob/master/baselines/common