COSI 165 Project Final Report
Team 2

Title: DQN playing Cart Pole and Atari Games

Team members: HyunJae Pi, Erchi Zhang

Contributions: Both HP and EZ equally contributed to this term project. EZ worked on hyperparameter tuning of CartPole and the simulation of 2-layer ConvNet. HP worked on hyperparameter tuning of Pong and the simulation of ResNet.

Abstract

Inspired by a series of works from Google DeepMind that tackles challenging topics in deep neural network and reinforcement learning, the goals of our term project are to reproduce the results from Mnih et al. 2013 & 2015 [2, 3] and improve its performance. Despite some success in a variety of domains, one of the problems is the applicability of reinforcement learning (RL) to situations with real-world complexity (e.g. high-dimensional sensory inputs) has been limited. Mnih et al. 2013 was the first paper that addressed this issue by developing a Deep Q-network agent playing classic Atari games [2]. Previously, we proposed to 1) build an agent playing a Cart Pole game, 2) build an agent playing Atari games, 3) compare the model performance by testing a convolutional neural network (ConvNet) and a residual neural network (ResNet). We managed to complete all our proposed works and now have better understanding of the model. One questions in the proposal was to why Minh et al. 2013 used the 2-layer ConvNet while Minh et al. 2015 used the 3-layer. The simulation results to this somewhat boring question gave a surprising finding; the 2-layer ConvNet outperformed the 3-layer ConvNet and ResNet. This finding suggests that the simpler model is better when the problem is not over-complicated. Overall, this project provided an opportunity to build deeper intuition into the deep neural network and reinforcement learning.

Introduction

Deep Q Network (DQN), an artificial agent developed by Mnih et al. 2013 & 2015, is able to solve a wide range of problems with real-world complexity (e.g. high-dimensional inputs) by combining reinforcement learning and deep neural networks [2, 3]. Their algorithm improved a classic RL algorithm Q-Learning with deep neural network and experience replay. Q-Learning was developed on basis of the optimal action-value function (Q-function), expressed as $Q^*(s, a)$, which is defined as the maximal return which can be gained starting from a state (or an observation) $\boldsymbol{s}$, taking action $\boldsymbol{a}$, and following the optimal policy π thereafter. Deep Learning was used as a function approximator to estimate the Q-values based on the Q-function, which significantly improved the efficiency higher than the original Q-learning algorithm. Experience replay, another key component of the algorithm, was used to make the updates of the network more stable. In each time step, the agent's experience (e.g. a transition from one state to another) is added to a replay buffer. When the loss and its gradient were estimated during the training, a mini-batch of transitions sampled from the replay buffer was used as a target value. With this design, DQN can give considerably good results on training a model to play classic Atari games. Here, we propose to apply DQN to train a model to play Cart Pole and Atari games. The reason we test Cart Pole is that it can be run on CPU therefore easy to debug. For our research, we will use DQN on games in gym, a python package that provides an RL environment.

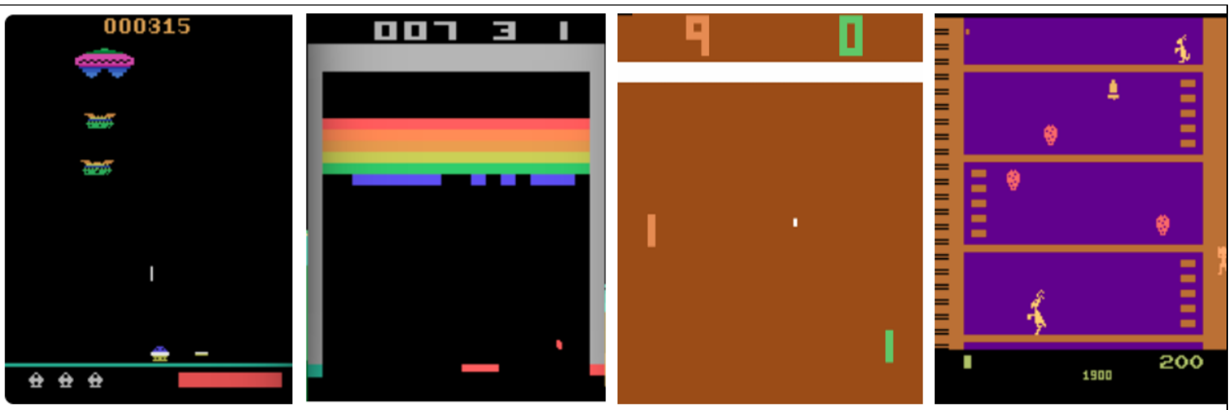


Figure 1. Examples of Atari games. From left: Space Invaders, Breakout, Pong,

Proposal

Problems

1. How can we use DQN to train models to get good performance on Cart Pole and Atari games?
2.
2. Is the DQN with ResNet better than ConvNet? Is it with 3 ConvNets better than with 2 ConvNets?
3. Can prioritized sweeping improve the performance of DQN?

Model

For training Cart Pole, we use RL and a deep neural network (DNN) with four fully connected layers. For the Atari game, we use RL and a convolutional neural network (CNN). We will test both ConvNet and ResNet, to compare their performance. There will be 2 or 3 conv layers (depending on the performance of the model), and 2 fully connected layers.

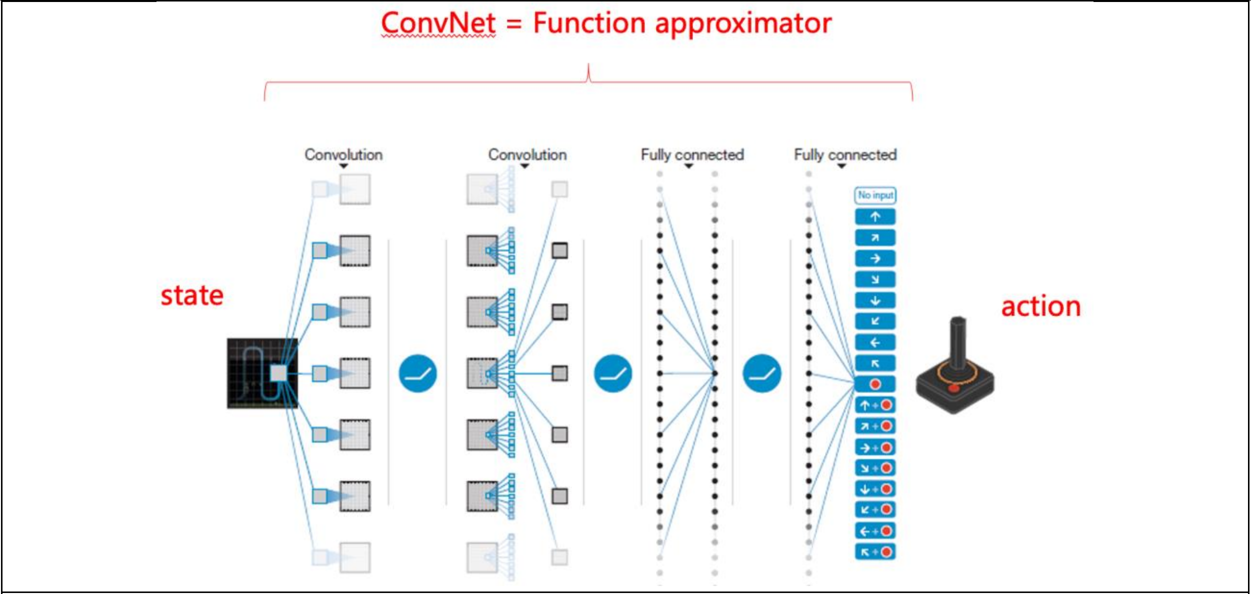


Figure 2. DQN model. ConvNet serves as a function approximator in reinforcement learning. The image was modified from Minh et al 2015.

Plans

1. Train a DQN to play Cart Pole. While the major framework is the same, this is a much simpler version. It can be run on a CPU and easier to debug.
2. Train a DQN to play Atari games. By replacing DNN and CartPole with CNN and an Atari game, respectively, we will train a DQN on GPU.
3. Deepen intuition of the model. We will change the number of conv layers and compare ConvNet with ResNet.
4. Test prioritized sampling. If time allows, we will test a different sampling method from previous research's sampling method. Mnih et al. 2013 & 2015 used uniform sampling, while they suggested that prioritized sampling may give better results [2, 3]. Thus, we will try prioritized sampling).

Results

DQN model can train CartPole.

CartPole is a classical gym environment where the agent should keep the pole straight up vertically as long as possible. Since the time required for training CartPole agent (several minutes) is far less than the time required for training an Atari game agent (several days), we decided to work on the CartPole project first. Once we have successfully completed the CartPole training, we can apply this experience on training Atari agent and thus finish the whole project with more ease.

We applied a DQN model into CartPole, and used RL and a deep neural network (DNN) with four fully connected layers. Indeed, our DQN agent was able to learn the task and successfully play the CartPole game, as you can see it in 'CartPole_before.gif' and 'CartPole_after.gif'. Fig. 3 shows one example of how reward increases as the agent learned.

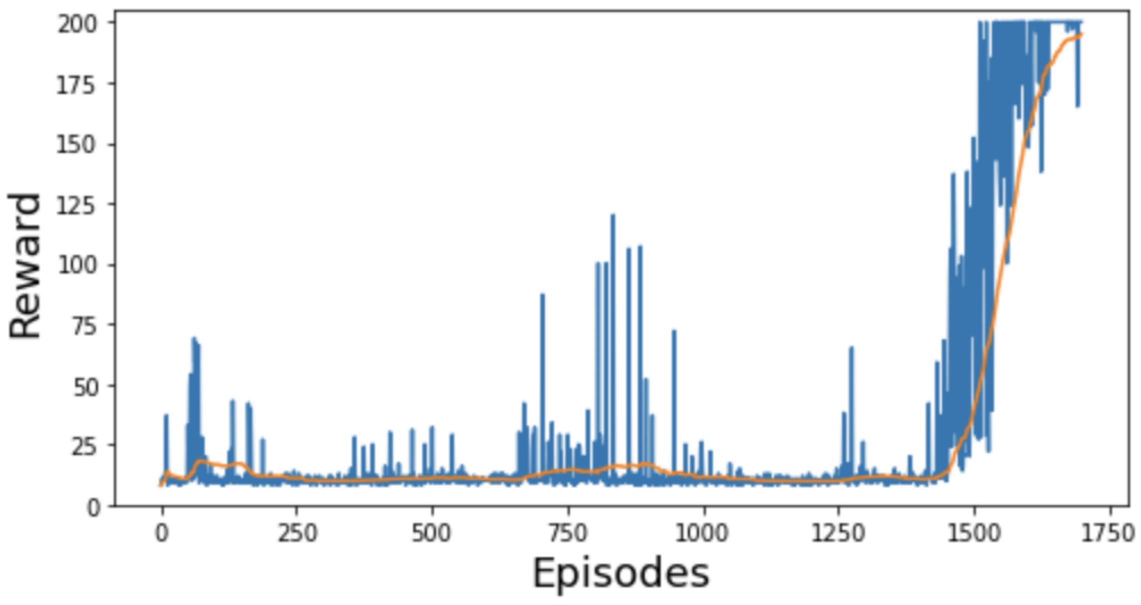


Figure 3. example of how rewards increase as the agent learns

On average, it took 2000 +/- 600 episodes. We also tested parameters that affect the performance of DQN to obtain the best result (Fig.4).

Parameter tunings

Tuning parameters is an essential step for obtain the best results. Although there are some existing codes and references on website for training CartPole model, they have suggested different parameters. As Greg Surma [4] suggested that 1 hidden layer is enough for obtain perfect performance, Abhav Kedia [5] recommended 2 hidden layers, Minh et al used even more hidden layers in his project, we have tested 1 vs. 2 vs. 3 vs. 5 hidden layers, and 5 hidden layers gave the best results. Similarly, we repeatedly tested different gamma values, synchronize frequencies "sync_freq", and epsilon values, and the final parameters we got for optimal model is hidden layer numbers to be 5, epsilon value to be 0.1, gamma to be 0.99, and sync_freq to be 5000. In Figure 4, you can observe that the optimal model we got performs very well, as it generally uses much less episodes than the model that using other parameters to achieve 200 rewards.

In summary, we were able to train the DQN agent to learn CartPole and identify the critical parameters. This code was used as a backbone for train Atari games. We also tested saving trained model function and saving result as gif file function, which were applied to training Pong agent and display its results later on.

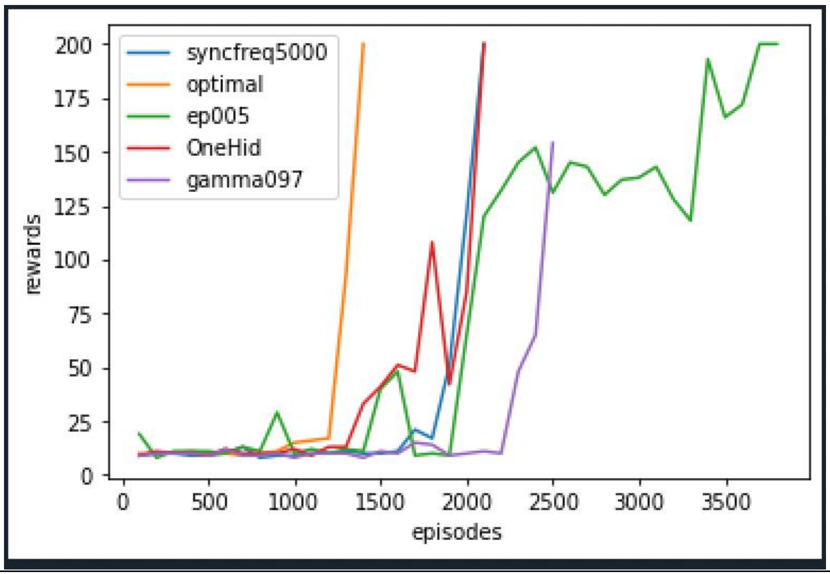
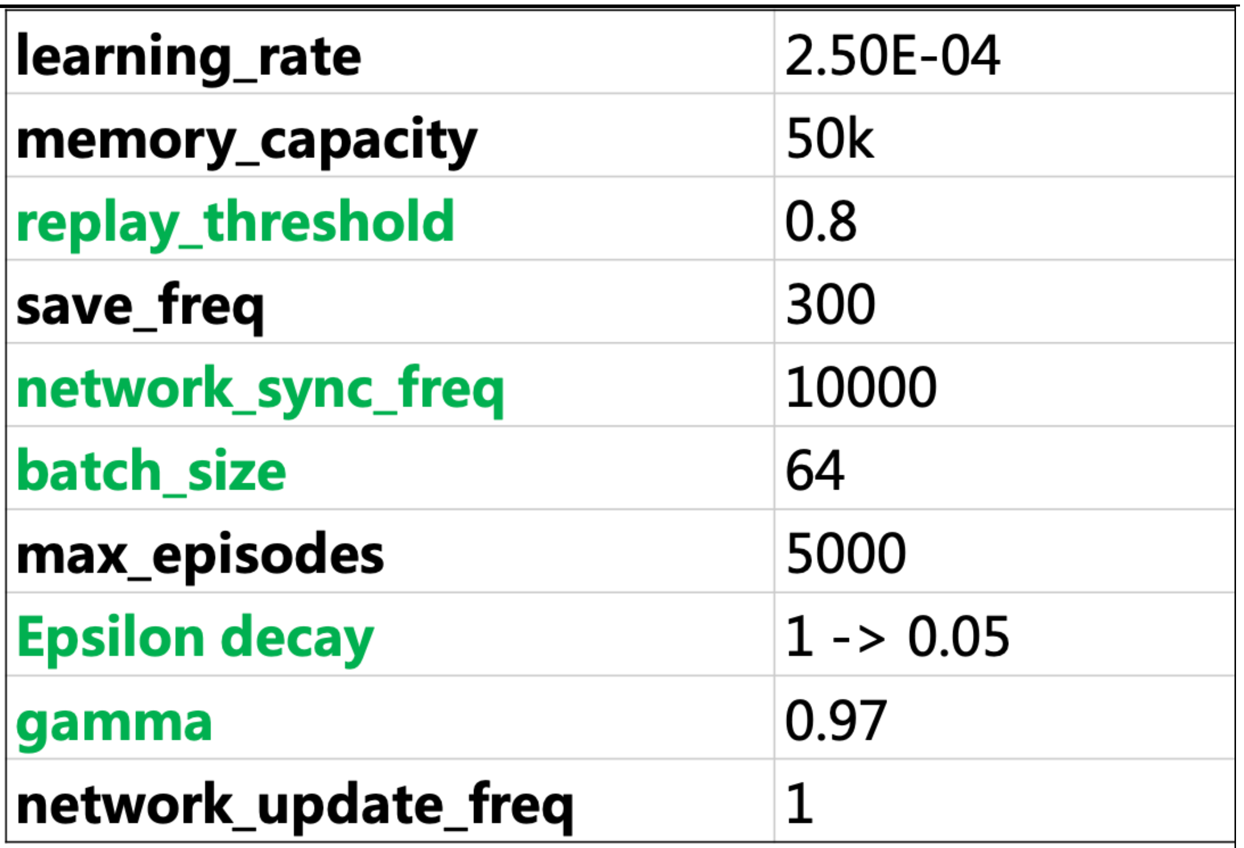


Figure 4. Hyperparameter tuning can improve the learning of CartPole game.

An DQN agent can play an Atari game, Pong

Initially, we started it with ‘Breakout’ game but switched to ‘Pong’ (Fig 1) because it was simpler than Breakout and took less time to see the progress. However, it still takes at least one million steps to tell whether the code was working or not. Although the majority of the code was the same as the code for CartPole, it still required some modification and we spent significant time to debug. The most difficult part of the project was that we had to find the working parameters. We had to use the different set of parameters from the ones in the paper because of the limitation of the resources (e.g., memory). In addition, parameters were inter-dependent and small change in a critical parameter can destabilize the model easily. Although we believe the logic of our code was fine and it didn’t have bugs (see Discussion), somehow our DQN model didn’t converge. We found online that Dueling DQN (DDQN) was more stable and converged better. We decided to replace DQN with DDQN. We tried a few different combinations of parameters and finally found a condition that the agent could learn the task. We set this model and its performance (e.g. how fast the agent learns) as the baseline model and the baseline performance, respectively. The parameters used for this baseline model are presented in Table. 1. The black line in Fig. 4 shows the learning curve of the baseline model.



learning_rate	2.50E-04
memory_capacity	50k
replay_threshold	0.8
save_freq	300
network_sync_freq	10000
batch_size	64
max_episodes	5000
Epsilon decay	1 -> 0.05
gamma	0.97
network_update_freq	1

Table1. The parameters used for Pong

Hyper-parameter tuning can improve the baseline model performance

Among several parameters that we tested, we identified a few critical parameters that can either improve the performance or destabilize the model. The findings are summarized in Fig. 2. In this simulation, we only modified one parameter value at a time while fixing the rest. The first parameter we found critical was 'Replay_th', the ratio of the amount of replay memory filled before the training. While the baseline model used the value that was suggested in the Minh et al 2015, this value wasn't optimal with the other parameter values in our simulation. We reduced it from 0.8 to 0.2 and got some improvement. Next, we tested the parameter, epsilon, the probability of the exploration that the agent makes. In the model, we have to gradually decrease this parameter to stabilize the model. The baseline model starts to decrease the value after filling the required replay memory with random exploration. However, we found that decreasing this value from the beginning significantly improve the performance (Fig.5). Finally, we also found that 'sync_freq' played an important role. The baseline model synchronized the target network every 10000 steps. We found that if we synchronize it every episodes (1000 ~ 3000 steps), the agent learned faster. With these findings, we tried to combined the conditions that improve the learning. Interestingly, combining two conditions didn't always improve the learning. Instead, it destabilized the network. For instance, while 'sync_freq=every episode' and 'epsilon decay =immediate' worked well separately, the combination of two destabilize the network (Fig. 5).

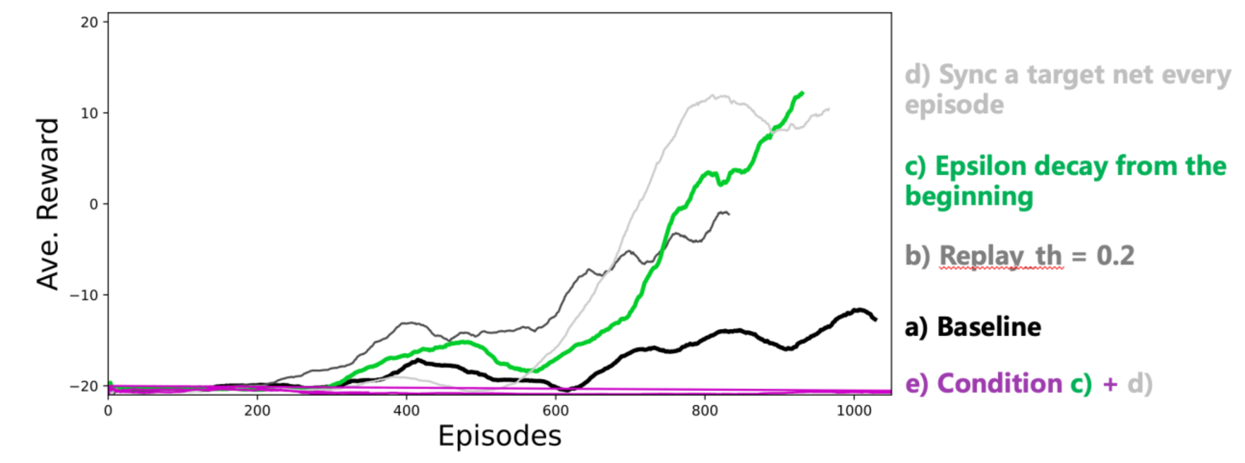


Figure 5. The baseline model and hyperparameter tuning that improves the learning.

2-layer ConvNet is better than 3-layer ConvNet

Minh et al 2013 used 2-layer ConvNet while Minh et al 2015 used the 3-layer one. Initially, we thought 3-layer ConvNet improved the model learning and performance and wondered how much better 3-layers would be. By answering this question, we wanted to build intuition of hyper-parameters of deep neural network. Surprisingly, the 2-layer ConvNet outperformed the 3-layer one (Fig. 6). We wondered why Minh et al 2015 used the 3-layer model. The difference between 2013 and 2015 papers is that in 2013, they only tested a few Atari games but in 2015 they tested all games that include much more complex game than Pong or Breakout. Therefore, we assume that the model structure and parameters used in 2015 paper was a working condition for all games. Meanwhile, if the game is simple enough, the model with 2 layers works better. This finding was not only shocking but also helped us to better understand the deep neural network in general.

2-layer ResNet learns faster than the 3-layer ConvNet but slower than 2-layer ConvNet

When we submitted our proposal, we expected ResNet would outperform ConvNet. However, after seeing the result that the 2-layer ConvNet was better than the 3-layer, we had no idea how the outcome would by ResNet. One of our goal is to improve the model performance, we decided to keep 2-layer ConvNet and only add one identity module to compare it with our best performing model. The simulation results showed that ResNet was better than the 3-layer ConvNet but still worse than the 2-layer ConvNet (Fig. 6).

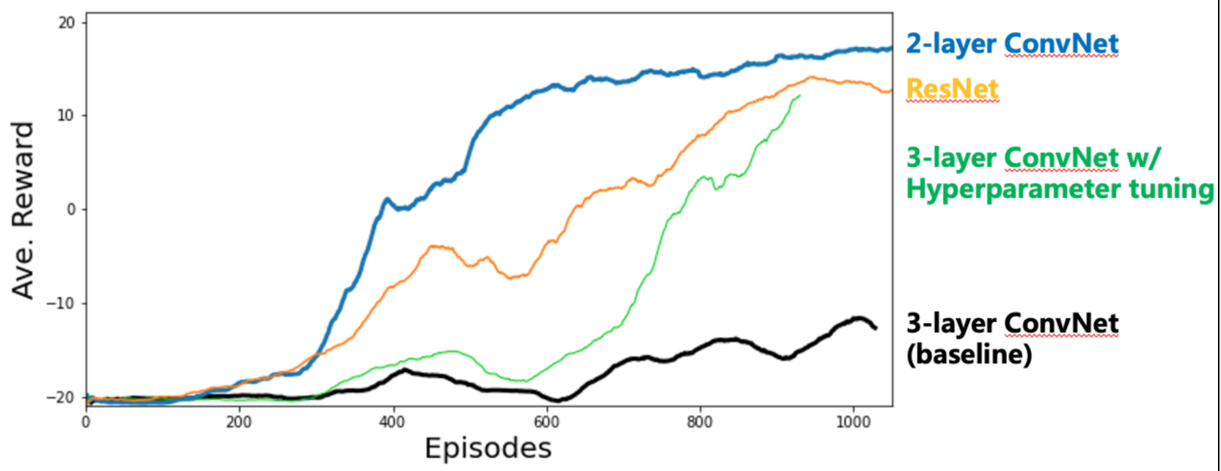


Figure 6. The 2-layer ConvNet outperforms the 3-layer ConvNet and ResNet.

Discussion

This term project aims to reproduce the results from Minh et al 2013 and 2015 and improve the model if possible. We managed to complete the work we proposed except testing prioritized sampling method, which we indicated that we might not be able to test due to time limit. Overall, it was a very interesting project that we learned not only theoretical aspects of deep learning but also practical challenges. Before this project, we have not run a code that requires several days of running even with GPU. Although debugging was very difficult, this experience prepared us to become more organized and plan better with more efficient strategy when we work on a big project in the future.

The most interesting finding was that the performance of the 2-layer ConvNet was the best among the more complicated models. At a first glance, it seems counter-intuitive yet it makes sense considering the simplicity of Pong game. In the class, we learned that the more complex models are not always better and our finding demonstrate the true example of this.

Minh et al. 2013 and 2015 contributed significantly to the field of deep neural networks and reinforcement learning. After this seminal works, new ideas and algorithms came out and improve the model. The prioritized sampling is one of them. If we continue to work on this project, we would like to continue to test these new ideas.

References

[1] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, van den Driessche G, Graepel T, Hassabis D. Mastering the game of Go without human knowledge. Nature. 2017 Oct 18;550(7676):354-359. doi: 10.1038/nature24270. PMID: 29052630.

[2] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. *NIPS Deep Learning Workshop*, 2013.

[3] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature. 2015 Feb 26;518(7540):529-33. doi: 10.1038/nature14236. PMID: 25719670.

[4]Greg Surma.(2018, Sep 26). Cartpole-introduction to reinforcement learning (dqn-deep q-learning).https://gsurma.medium.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288

[5] Abhav Kedia. (2020, Apr 8). Creating deep neural networks from scratch, an introduction to reinforcement learning. https://towardsdatascience.com/creating-deep-neural-networks-from-scratch-an-introduction-to-reinforcement-learning-part-i-549ef7b149d2