Project report
IE 534 Deep Learning

I List of team member

Qianli Chen (UIN:671407973, UID: qchen35),
Haoyang Li (UIN: 675358595, UID: hli70),
Muyun Lihan (UIN:653549073, UID: mlihan2).

II Project name

Implement one of the following deep reinforcement learning papers for Atari Games in PyTorch: (B) "Deep exploration via bootstrapped DQN", NIPS, 2016 by Osband et al.

III Paper summary

The original paper presents a novel approach, bootstrapped DQN, to improve classical reinforcement learning (RL) strategies in a computationally and statistically efficient manner. Unlike common dithering strategies (e.g., ε-greedy exploration), the proposed algorithm encourages temporally-extended (or deep) exploration by creating a new architecture that supports choosing actions from randomized Q-functions that are trained on bootstrapped data.

Bootstrapped DQN modifies DQN to approximate a distribution over Q-values via bootstrapping. The network consists of a shared architecture with K independent bootstrapped heads. Each head is trained on its own subset of data representing a different Q-function. The shared network learns a joint feature representation across all the data. Experience for training is stored in a replay buffer containing flags identifying which heads are trained on which data. A core idea of the bootstrapped DQN is the bootstrap mask. Each individual experience is given a randomly sampled mask. In the process of training the network on a mini-batch sampled from the replay buffer, the mask decides whether or not a specific Q-value function is to be trained upon that experience.

The authors evaluates the proposed algorithm across 49 Atari games on the Arcade Learning Environment (ALE). Observing from the overall performance of the algorithm, Bootstrapped DQN improves upon the final score across most games. More significantly, bootstrapped DQN outperforms DQN in terms of the cumulative rewards through learning. In addition, the algorithm is able to learn much faster than DQN. Given the network architecture, it is also demonstrated in this work that bootstrapped DQN is not only computationally tractable but naturally scalable to massive parallel systems as well.

IV Implementation

1. Dataset

For our project, we use the Atari game dataset implemented by OpenAI Gym. Gym is a collection of environments/problems designed for testing and developing reinforcement learning algorithms—it saves the user from having to create complicated environments. Gym is written in Python, and there are multiple environments such as robot simulations or Atari games. The games we selected to test the models are *Pong* and *Breakout*, PongDeterministic-v4 and BreakoutDeterministic-v4, to be precise. The observation at each step/frame is an RGB image of the screen during the gameplay, which is an array of shape (210, 160, 3). For *Pong*, the action space is {0, 2, 5} which correspond to no-op, up, and down, respectively. The agent plays against the computer and whoever reaches 21 points wins the game. For *Breakout*, the action space is {0, 1, 2, 3} which correspond to no-op, fire, left, and right, respectively. The agent tries to use the ball to hit and destroy layers of bricks and to achieve higher scores. For training purpose, we only uses {0, 2, 3} following the start of a game and uses the loss of the first life as the end of the game. Sample screenshots of both games are shown in Figure 1.

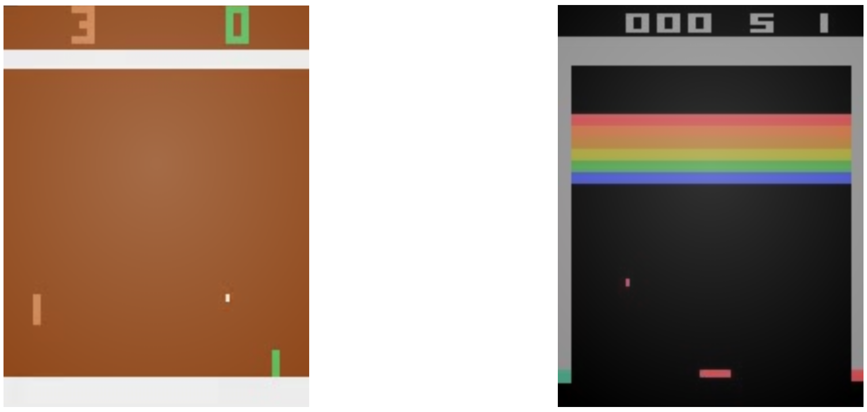

Figure 1: Screenshots of *Pong* and *Breakout*

The reference for the introduction and documentation of gym can be found in [3,4]

2. Model architectures

We followed the DQN algorithm described in [2] and bootstrapped algorithm described in [1]. The development of our code for DQN and bootstrapped DQN are based on a basic implementation of the vanilla DQN [5] which was not working initially. Some modifications are made to be able to train the basic DQN agent successfully on the game *Pong*. After that, we implemented the bootstrapped DQN architecture.

The input is 4x84x84 tensor with a rescaled, grayscale version of the last four observations. The convolutional network adopted in this project is described as follow. The first convolutional layer has 32 filters of size 8 with a stride of 4, followed by a rectifier nonlinearity. The second convolutional layer has 64 filters of size 4 with a stride of 2, followed by a rectifier nonlinearity. The third convolutional layer has 64 filters of size 3, followed by a rectifier nonlinearity. The fourth convolutional layer has 512 filters of size 7 with a stride of 4, followed by a rectifier nonlinearity. The output layer is a convolutional layer of size equal to the number of valid actions. In the bootstrapped DQN implementation, we define k separate models of the same network structure as the bootstrapped heads. Due to time constraints, parallelism of the shared network architecture is not implemented and explored in this work. As described in the original paper, we use an independent Bernoulli mask with parameter p=0.5 for each head in each episode. The bootstrapped mask m_t decides, for each value function of each head Q_k, whether or not it should train upon the experience generated at step t.

3. Hyperparameters

The hyperparameters adopted to train the models are summarized as follow:

- Optimizer: RMSProp, lr=0.0025, alpha=0.95, eps=1e-02
- Batch size: 64 for *Pong* and 32 for *Breakout*
- Discount factor: 0.99
- Target network update step: 1,000
- Memory buffer size: 100,000
- Number of bootstrapped heads: 2, 5, 10
- Bootstrapped DQN mask: Bernoulli with p=0.5
- ε-greedy policy for DQN: ε decays linearly from 1.0 to 0.1 over first 1,000,000 steps

- reward clipping: positive rewards clipped to 1, negative rewards to -1, zero unchanged

4. Computational hours

In total, we have used approximately 600 Blue Waters GPU hours, 50 personal computer GPU hours, and 100 Bridges GPU hours.

V Results and discussion

1. Pong

Basic DQN with ε-greedy and Bootstrapped DQN with k heads (k=2, 5, 10) are trained on the game *Pong*. Average scores and Q-values during training are shown in Figure 2. Scores of Bootstrapped DQN are evaluated using the ensemble policy (discussed later). We observe that, with 5 and 10 heads, the Bootstrapped DQN is able to achieve human performance faster than the DQN. Taking 14.6 being the human-level score as suggested in Table 1 in the paper, Bootstrapped DQN reaches this scores at ~70000 steps of training while DQN achieves this at ~90000 steps. This yields that the Bootstrapped DQN reaches human performance faster than the DQN for *Pong* with a factor of 1.28 which is similar to what is shown in Figure 7 of the paper. On the other hand, Bootstrapped DQN with k=2 performs equally to the DQN in terms of the score although it reaches better Q-value. In Appendix C.1 of the paper, it is discussed that the Bootstrapped DQN can implement deep exploration even with a small number of heads. However, the results are more stable and robust for larger k. This might explain that why the results we obtained for k=2 does not show any improvement against basic DQN. With regards to the fact that the Bootstrapped DQN shows similar performance for k=5 and k=10, we think that the need for efficient and deep exploration is not urgent in the case of *Pong*. Therefore, increasing the number of heads does not help the agent reach higher scores faster.

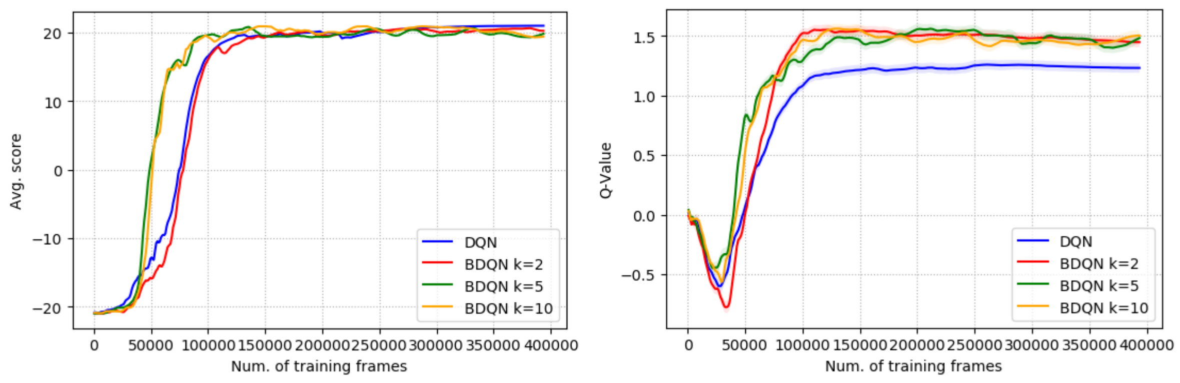

Figure 2: Average *Pong* scores and Q-values during training for DQN and Bootstrapped DQN with different number of heads

As suggested in the paper, while evaluating the Bootstrapped DQN, we use an ensemble policy to determine the action at each step. To wit, the action with the most votes across all heads is chosen at a given state. Figures 3 and 4 show the final scores from different heads as well as the ensembled one at training frame ~100000. We observe behaviors similar to what the paper claimed. The ensemble policy often outperforms other individual policies. At less important steps, diverse decisions are made by different heads while at critical steps most heads tend to agree on the same crucial action. We also observe that, at the early stage of training, the ensemble policy usually makes worse decisions than some individual heads since most heads are not trained well at that point. As the training progresses, the performance of the ensemble policy becomes increasingly better than individual ones.

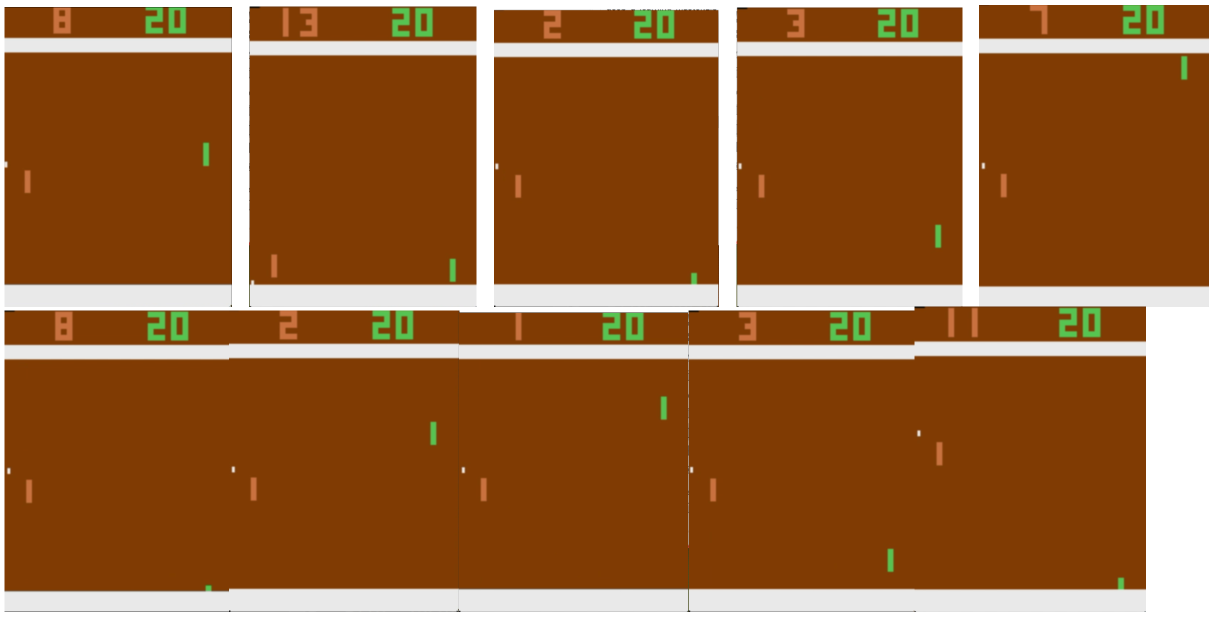

Figure 3: Final scores of *Pong* for different heads

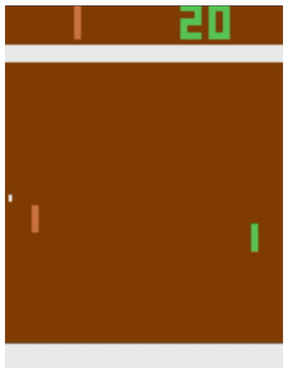

Figure 4: Final score of *Pong* for the ensemble policy

2. Breakout

We have extended the same DQN and Bootstrapped DQN methods used in *Pong* to the training of another Atari game, *Breakout*. Due to limited walltime (48 hours) available on Blue Waters and varied computational costs for each model, different training schemes ended up with various progress when the walltime was reached. Compared to results in the paper, our models seemed to only start the initial exploration within our limited training time before the replay buffer was released from memory especially for Bootstrapped DQN models with multiple heads. We observe that Breakout took much more frames to complete each episode as the game progresses further, which limited the exploration for the state space corresponding to mid/late games. As a result, our models were mainly trained on the onset stage of *Breakout*. We observe that the agent failed to react at a certain point once it passes through the early stage of the game, indicating the exploration chain has not been trained to that depth.

In Figure 5, average *Breakout* scores and Q-values are compared for DQN and Bootstrapped DQN models as the training proceeds. Here, we are using clipped accumulated rewards as evaluation scores instead of real game scores to be consistent with reward clipping in the training process. For bootstrapped DQN models, we have divided their number of frames by the number of heads used in each model so that this reflects the assumed parallel computational hours comparable to single head DQN models. In terms of the model evaluation, the vanilla DQN achieved the highest score among all models mostly because it was trained with more episodes/frames, allowing for longer and deeper exploration. For the Bootstrapped DQN with 5 and 10 heads, their performances were similar to the early stage of vanilla DQN. Unexpectedly, Bootstrapped DQN with 2 heads and DQN with ε-greedy achieved much lower scores. We believe the failure of Bootstrapped DQN with 2 heads resulted from its lack of effective ensemble policy during evaluation, even though the initial exploration was successful suggested by its high action value/Q-value. In contrast, DQN with ε-greedy failed even at the initial exploration stage because the random actions hugely prevent the game from progressing further. In summary, Bootstrapped DQN models all started to train the early stage of the *Breakout* shown by their Q-values, buts its advantages over DQN were not apparently shown due to the lack of training time.

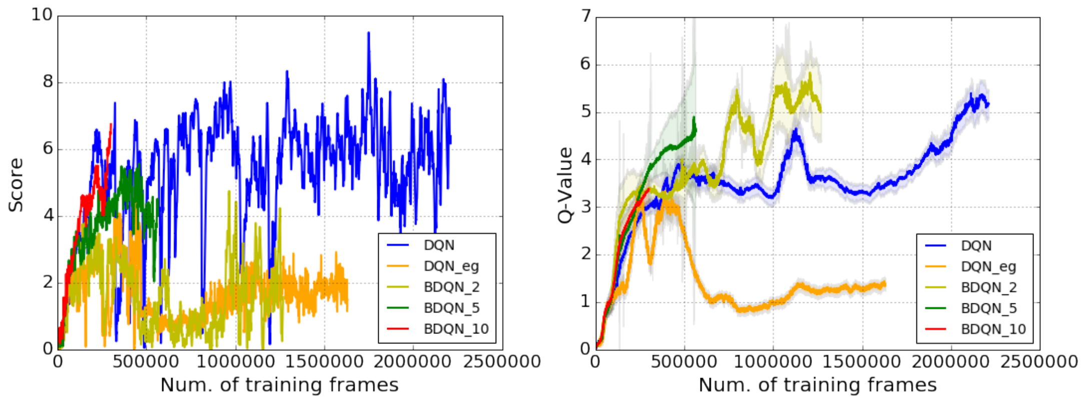

Figure 5: Average Breakout scores and Q-values during training for vanilla DQN, DQN with ε-greedy (DQN_eg) and Bootstrapped DQN with different number of heads

References

[1] Deep exploration via bootstrapped DQN. https://arxiv.org/abs/1602.04621

[2] Human-level control through deep reinforcement learning. https://www.nature.com/articles/nature14236

[3] Introduction to reinforcement learning and OpenAI Gym. https://www.oreilly.com/learning/introduction-to-reinforcement-learning-and-openai-gym

[4] Gym documentation. https://gym.openai.com/docs/

[5] https://github.com/pianomania/DQN-pytorch