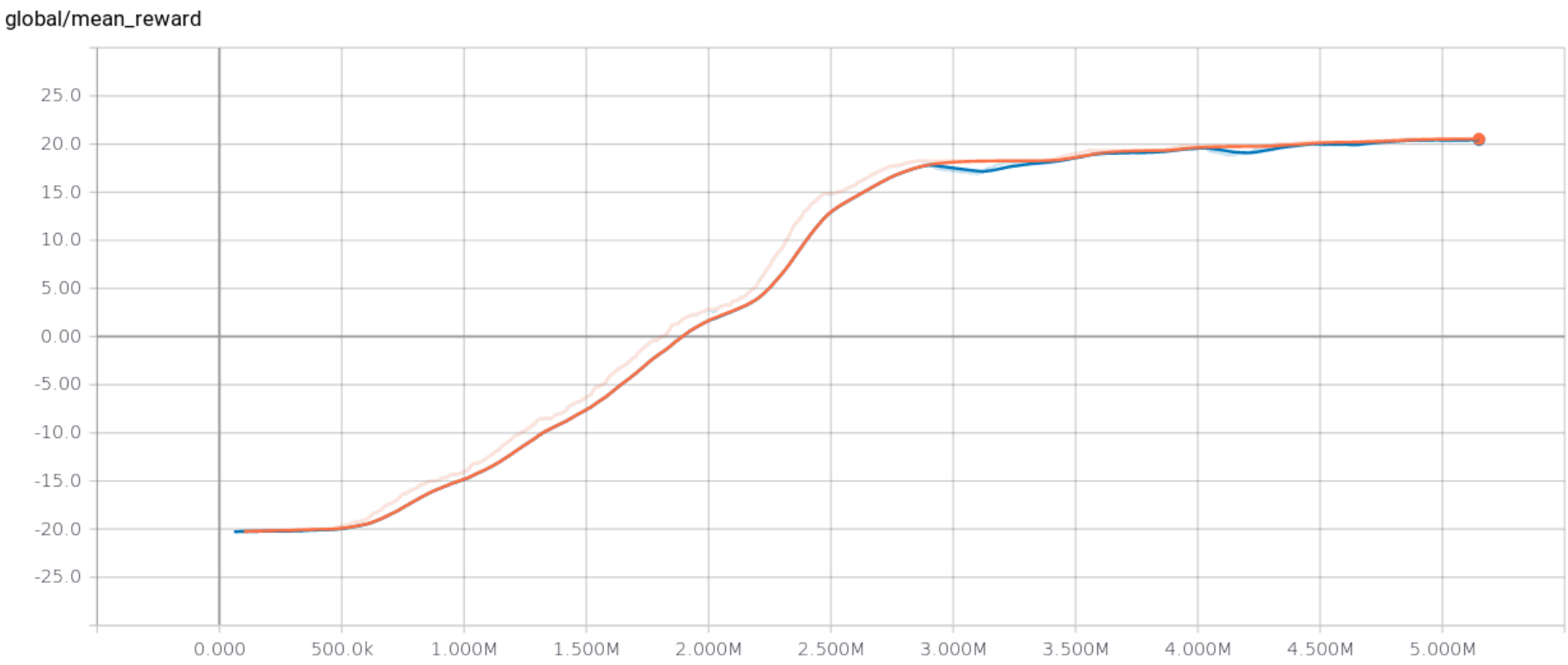
Week 5 - Deep Q Networks and Rainbow Algorithm

Submitted by hollygrimm on Mon, 07/09/2018 - 09:13

The first part of this week was spent working on homework 3 for CS294 "Using Q-Learning with convolutional neural networks" [4] for playing Atari games, also known as Deep Q Networks (DQN). (Source on GitHub)

Like last week, training was done on Atari Pong. I was able to improve my +6 score using Policy Gradients to receive a +20 reward after 5 million games with DQN:



Game Video:

Neural Network

DQN uses a Neural Network to learn Q values. The network was composed of three convolutional layers and two fully connected layers, similar to the Nature DQN paper [3].

Replay Buffer

If the Q network is trained on sequential states, the data from timestep to timestep will be strongly correlated and the network would tend to overfit to local regions.

To address this issue, the Q network can be trained on a replay buffer. As the agent is interacting with the environment, the data is being stored in the buffer. Actions are selected using the epsilon-greedy policy with a linear schedule across 1,000,000 timesteps from 1.0 to 0.1.

The replay buffer holds 1 million transitions and the Q network only trains after a threshold of 50,000 transitions have been stored. After the threshold has been reached, it will train every 4 transitions. Mini-batches are sampled from the buffer when training. When the buffer reaches 1 million transitions, the old memories are dropped.

Target Network

When updating the Q values for a state and action, due to the nature of the neural network, it will also update the Q values for similar states and actions. When updating the Q network with these values, you end up trying to hit a moving target.

To stabilize training, a Target Network is introduced. The Target Network has the same structure as the Q (Online) Network but with different parameters φ' and φ respectively. The Target network parameters are fixed for several iterations, and after a certain number of timesteps (typically 10,000), the Q network will be copied over to the Target Network.

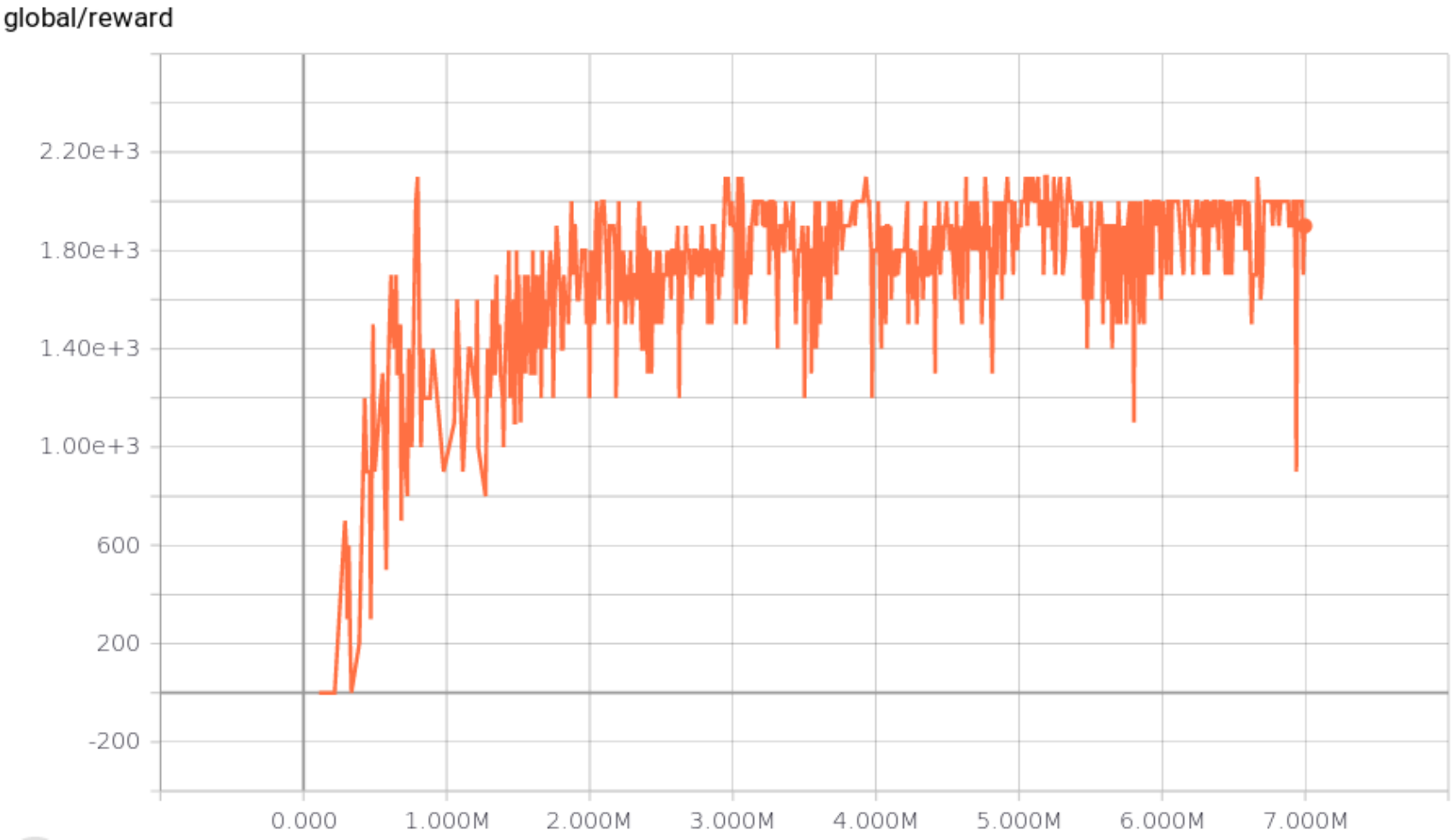
Huber Loss

For computing loss, Huber Loss is used instead of squared error where dealing with very large gradients. With Huber Loss, slope becomes constant as you move away from zero. As a result, it penalizes the network less for making large mistakes and focuses on smaller errors more.

Sonic the Hedgehog Trained with Rainbow

The second part of my week was spent working on training "Sonic the Hedgehog" using the Rainbow Algorithm [5]. As a framework, I used Alex Nichol's project anyrl-py [6] [7].

I trained (Source on GitHub) for seven million timesteps. Below is the reward for each game played; the reward scores maxed out at 2100:



This is the agent with mostly random actions at the beginning of training:

0:00 / 0:13

This agent has learned how to quickly get through the beginning of the game, but struggles in the water:

0:00 / 0:39

Rainbow Algorithm

Rainbow takes the standard DQN algorithm and adds the following features:

Prioritized Experience Replay

Replaying all transitions from the replay buffer with equal probability is wasteful. It's better to prioritize the data sampled from the buffer using the absolute Bellman error where the predicted reward diverges greatly from the expected reward. In addition, more recent transitions have a higher priority. [8]

Double DQN (DDQN)

The estimated Q values from the Neural Network are often overestimated. When you calculate the target value, it's based on the highest valued action for a state. Updating the Q network with these values will keep amplifying the Q values.

Since we already have two networks, the Q network and Target network, you can use the Q network (φ) to select the best action using argmax, and then the Target network (φ') to get the value estimate. [9] This decorrelates the noise and avoids amplification.

Dueling DQN

The advantage value is the difference between the Q(s, a) value and the state value V(s). For an action it's measuring how much worse an action is in comparison to the best action in a state.

With Dueling DQN, you change the architecture of the neural network by splitting it into two separate estimators after convolution. The first estimator is the state value V(s), a single number. The second estimator outputs one number per action, the advantages. To calculate the Q values, the advantage is summed with the state value and the average value of the advantages is subtracted. [10]

Noisy Nets for Exploration

Add noise to network parameters for better exploration [11] and [12]

N-step Q-Learning

When calculating the target value in Q-Learning, the target value is based on only the current reward. For N-step Q-Learning, rewards from N steps are added together and the Q function value is added only at the very end.

Distributional RL

Takes a single output reward value and replaces it with a distribution of rewards across N values. For the paper, they used 51 bins and called the algorithm C51. [13]

References

1. Richard Sutton and Andrew Barto. "Reinforcement Learning: An Introduction". Chapter 16.5: DQN
2. Vlad Mnih. "Deep RL Bootcamp Core Lecture 3 DQN + Variants". Video | Slides
3. Vlad Mnih et al. "Human-level control through deep reinforcement learning". PDF
4. Sergey Levine. "Advanced Q-learning algorithms". CS294 Fall 2017 Course at Berkeley. Video | Slides
5. Matteo Hessel. "Rainbow: Combining Improvements in Deep Reinforcement Learning". PDF
6. Alex Nichol et al. "Gotta Learn Fast: A New Benchmark for Generalization in RL". PDF
7. Alex Nichol. anyrl-py. GitHub
8. Tom Schaul et al. "Prioritized Experience Replay". PDF
9. Hado van Hasselt et al. "Deep Reinforcement Learning with Double Q-learning". PDF
10. Ziyu Wang et al. "Dueling Network Architectures for Deep Reinforcement Learning". PDF
11. Meire Fortunato et al. "Noisy Networks for Exploration". PDF
12. Matthias Plappert et al. "Parameter Space Noise for Exploration". PDF
13. Marc G. Bellemare et al. "Going beyond average for reinforcement learning". Blog

Tags
Reinforcement Learning OpenAI Pong Sonic the Hedgehog