Project 1: Navigation

Rory McGrath

January 31, 2019

1 Overview

For this project an agent was trained to navigate a large square world and collect bananas. A reward of +1 was provided for collecting a yellow banana, and a reward of -1 was provided for collecting a blue banana. The goal of the agent is to collect as many yellow bananas as possible within a given timeframe while avoiding blue bananas.

For this project the state space has 37 dimensions. These attributes represented the agent's velocity along with the ray-based perception of objects around the agent's forward direction. The action was able to perform 4 distance actions:

Action Value	Description
0	move forward
1	move backward
2	turn left
3	turn right

The given task to solve is episodic. A agent is said to have 'solved' the environment if it can achieve a minimum score of +13 over 100 consecutive episodes.

2 Proposed Methods

In order to solve this environment Deep Q-Networks [3] were implemented. In general, Deep Q-Networks are used as a functional approximation when estimating the optimal action-value function Q_*. Instead of performing a table lookup, which does not scale well with the action space and state space, a neural network is trained on past experiences to estimate the action-value function for the current state and actions.

One of the main issues when initially using an artifical neural network (ANN) to estimate an optimal action-value function is how to you obtain the targets for back-propagation? The answer is relatively straight forward. Here we assume we are solving the environment using temporal difference (TD) and are provided with the data in the SARSA format, (state, action, reward, next_state, next_action) denoted by (s, a, r, s', a').

1. First we estimate the Q value of s' using forward propagation of the ANN to get the value of Q(s',a').

2. Next we calculate the Q value for the current state s by substituting the estimated next state values Q(s’,a’) into the equation:

$$Q(s,a) = r + \gamma Q(s', a')$$

3. We then estimate the value of Q(s, a) denoted by $\hat{\text{Q}}$(s, a) using forward propagation of our ANN.

4. Finally our ANN is then trained via back propagation using $\hat{\text{Q}}$(s, a) as our prediction and our calculated Q(s, a) as the target.

Although this may seem confusing to update our estimate with an estimate, by introducing the actual observed reward r and assuming sufficient exploration of the state space the Q values will tend towards their actual values.

Their are, however, two main issues with this naive approach. Firstly, as we are processing the (s, a, r, s’, a) tuples as they are created, the sequence of tuples can be highly correlated given the nature of the environment. This can cause action values to oscillate or diverge catastrophically. Secondly, since we are updating a guess with a guess this can lead to harmful correlations if we updated the ANN weights on each time step. To combat these issues Experience Replay[2] and Fixed Q-Targets[3] were implemented.

3 Results

Using the methodology described above [2] an agent was trained to solve the environment. The agent was written in pyTorch and was run using the openai gym toolkit.

The ANN used for the agent consisted of an input layer of 37 nodes, one dense hidden layer of 148 nodes and an output layer of 4 nodes. All activation functions were rectified linear units (ReLUs). As we are not dealing with probabilities a Softmax function on the output layer was not required.

The following hyper-parameters were used to train the agent:

Learning Updates

Parameter	Value
γ	0.99
α	$5e^{-4}$
ϵ	1.0 to 0.01 with a rate of decay of 0.995 per episode

Experience Replay

Parameter	Value
Buffer Size	$1e^5$
Batch Size	64

Fixed Q

Parameter	Value
τ	$1e^{-3}$
Update Rate	every 4 time-steps

The model was trained for 1400 episodes. After 531 episodes the model achieved it’s target of an average of +13 over 100 episodes. A plot of the reward verses episode number is given in [Fig 1].

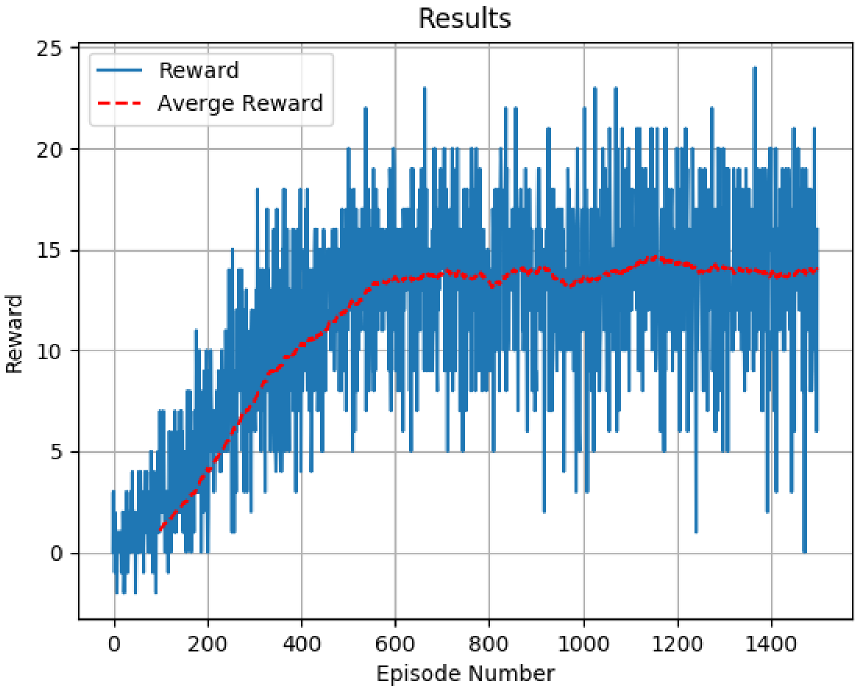

Figure 1: Training results

4 Future Work

While the initial results of this agent are promising, the agents behaviour is far from optimal. There are several improvements that could be implemented to decrease training time and increase the average reward.

A hyper-parameter optimisation technique such as grid search could be implemented to help converge on good choices for the parameters specified above. Ideally this would be run on a gpu server as running it on a cpu would take a considerable amount of time.

The Replay Buffer could be updated to implement prioritised experience replay [4]. The key idea here is that 'important' transitions would be sampled more frequently than unimportant ones. The 'important' of transition is determined by the magnitude of it's temporal-difference error. This should allow the agent to learn more efficiently. Given that we are no longer uniformly sampling from the experiences importance sampling will have to be added to calculate the impact of each weight when updating.

The agents learning method could be updated to implement Double DQN [1]. Currently the agent uses the same value to select an action and then evaluate that action. This can lead to over optimistic value estimates. To overcome this the action selection can be decoupled from the evaluation. In the case of our implementation we can partially decoupling the action selection and evaluation by selection the action using the local Q network and evaluating the action on the target Q network.

References

[1] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *AAAI*. 2016.

[2] Long-Ji Lin. “Self-improving reactive agents based on reinforcement learning, planning and teaching”. In: *Machine Learning* 8.3 (1992), pp. 293–321. ISSN: 1573-0565. DOI: `10.1007/BF00992699`. URL: `https://doi.org/10.1007/BF00992699`.

[3] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (Feb. 2015), 529 EP –. URL: `https://doi.org/10.1038/nature14236`.

[4] Tom Schaul et al. “Prioritized Experience Replay”. In: *CoRR* abs/1511.05952 (2015).