Navigation Deep Q Learning Task

Task description

The problem tackled in this work was training an agent to navigate a 3D environment, while collecting rewards (in the form of yellow bananas) and avoiding penalties (in the form of blue bananas). The agent was given a score of +1 for each yellow banana collected, and -1 for each blue banana.

The problem has a 37 dimension state space, corresponding to the velocity of the agent and its surroundings (represented in a ray based perception). There are 4 discrete actions, corresponding to moving forwards, left, right or backwards.

Approach taken

The approached taken was a simple variant of Deep Q learning. Deep Q learning is a technique that allows the classic Q learning algorithm to be applied while using a neural network for non linear function approximation. Q learning attempts to train an agent by learning the below Q function, which represents the value of an action taken in a state.

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

In deep Q learning, the Q function is represented using a deep neural network, which is trained using the above function as a loss (the difference between the current Q value and . Historically this setup was considered unstable and difficult to train, two innovations in the DQN work reduced these issues to allow successful training. The first of these was the use of a replay buffer. In traditional Q learning, the Q estimates are adjusted after each action made by the agent. This introduces a bias, due to temporal correlation between the actions in an episode. By

Footnote: there is a small bug in the implementation. When reporting scores for good runs (those with an average score of the past 100 runs of 14 of more), the notebook prints off an index for the iteration 100 less than the correct value. This is visually jarring, and incorrect, but has no impact beyond that.

adding the experiences to a buffer and sampling randomly from this at intervals to learn, this temporal correlation is broken.
The second novel feature introduced was a fixed target network. When minimising the loss between the current Q function and the new estimate, a fixed set of weights, updated periodically, is used to calculate the current Q function. This improved training stability.

In this work, a very simple small neural network was used to represent the Q function, consisting of three fully connected layers, using ReLU activation. The input layer had size 37, corresponding to the state space, the two hidden layers had size 64, and the output layer had size 4, corresponding to the action space. The Adam optimiser was used in training.

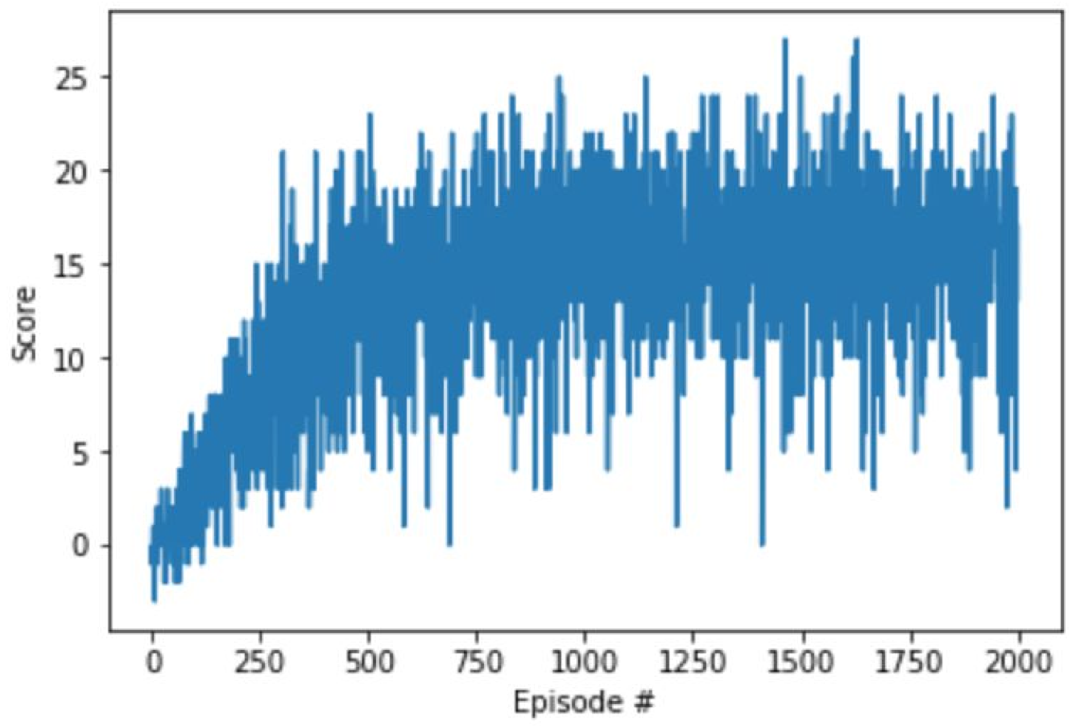
Hyperparameters

- Buffer size (number of items held in replay buffer) = 100000
- Batch size = 65
- Gamma = 0.99
- Tau = 0.001
- Learning rate = 0.0005
- Network is updated every 4 steps

Results

The agent described above was trained for 2000 iterations (this took around 45 minutes with a GPU). The score over time is shown in the graph below. There was high variance in the scores achieved, but the average score trended upwards for around 1000 iterations, reaching an average score of around 16.5, at which point it plateaued, and declined slightly.

Footnote: there is a small bug in the implementation. When reporting scores for good runs (those with an average score of the past 100 runs of 14 of more), the notebook prints off an index for the iteration 100 less than the correct value. This is visually jarring, and incorrect, but has no impact beyond that.



Next steps

There are several possible extensions for this work. The first is extending the deep Q learning implementation used with some of the improvements made to the algorithm since its publication, such as Double DQN to reduce Q value overestimation, and prioritised replay buffer, which may reduce some of the instability seen in training. The second is applying standard ML techniques, such as early stopping. With this agent, the average score after 1000 iterations was one of the strongest, and it would have been better to save those weights than those learned later on.

Finally, it would be interesting to apply the approach to a harder variant of the task. If the state used was all of the available pixels, rather than the simplified ray based representation used here, the task would be much harder. A convolution neural network based approach may do well with this task.

Footnote: there is a small bug in the implementation. When reporting scores for good runs (those with an average score of the past 100 runs of 14 of more), the notebook prints off an index for the iteration 100 less than the correct value. This is visually jarring, and incorrect, but has no impact beyond that.