HW5P1

December 10, 2019

1 Problem 1 - Deep Reinforcement Learning

1.0.1 1

The fundamental difference between episodic and continuous tasks is that an episodic task has an end, whereas continuous tasks go on forever. - An episodic task lasts a finite amount of time. For example, playing a single game of Go is an episodic task, which you win or lose. In an episodic task, there might be only a single reward (as with winning in games), at the end of the task, and one option is to distribute the reward evenly across all actions taken in that episode. - In a continuous task, rewards might be assigned with discounting, so more recent reactions receive greater reward, and actions a long time in the past receive a vanishingly small reward. For example, tuning of a heating system has no perceivable end- it is a continuous process at which the correct temperature has to be maintained.

1.0.2 2

a) Reinforcement learning is a type of online learning, where data is collected on the go. Every time a decision has to be taken, the learner should decide whether to pick an action based on previous data collected, or to choose a different action to discover the consequence of a certain action.

- The former- where decisions are based on previously analysed data is called exploitation. There is a high probability that the action leads us in the direction of the reward.
- The latter- where we take a different decision to gather data on what the consequence could be, is called exploration. Here, our aim is to collect data so that we can make more informed decisions in the future.

b,d) ϵ is a parameter used to quantify the above- exploration vs exploitation. The goal of RL is to maximize future rewards- thus, a balanced component of both exploration and exploitation is required. By associating a probability of ϵ for exploration, we greedily select whether to explore or exploit.

c) In the experiment, a schedule was followed for ϵ, which decreased over time upto a million actions, post which it was fixed at 0.1. This can be explained by the fact that in the initial set of actions, the amount of data stored to base decisions off is minimal, and thus the actions need not perform better than a random action. As we progress further in time, our sie of collected data increases, and we can predict the reward scenarios for a lot of actions

accurately. Finally, once the model has gained enough "experience", the probability that the random action results in a better reward is very small, and thus reflected in the choice of ϵ.

1.0.3 3

In a normal Q Learning Algorithm, the rewards are calculated based on the data of the previous N steps. However, these N samples collected are strongly co-related- each successive sample was selected due to a certain choice of action taken at that stage. Thus, the data we generate is skewed towards a certain set of actions, and we do not have samples for other potential actions that could have been taken at any of these steps. Thus, there is an inherent bias in our data. Training on this data may not give the peak performance that our model can achieve. This is overcome by using Deep Q Learning, the algorithm described in the paper. Instead of using only the previous N samples, all setps taken so far are stored in a "replay" memory. N randomized samples are selected from this memory for training. This reduces the bias in the data (bias wrt reward may be good, but not with respect to actions), and contributes towards better training. Second, it also results in better training effeciency- data is seen by multiple training rounds- ie the same data is used to update weights multiple times. Finally, using a sample of N random points averages moves over the entire distribution of data, and not just on the previous steps.

Steps of the algorithm: Over M episodes, with each episode having a sequence of t steps, do - Determine action: random vs one that gives maximum reward based on previous experience with a probaility ϵ. - Execute action, and observe the consequent state and its associated reward. - Prepare for next step- add data point and its assocaited reward to history. Use current state as input to next step. - Select a minibatch of N samples and train the model: set predicted reward as current_reward + γ* previous_reward (temporal distribution- recent events have a higher reward) - Use gradient descent to update the model based on predicted reward.

1.0.4 4

The target network maintains a separate weight vector which is updated only once over a batch of iterations. This is done to create a temporal gap between the target action-value function and the action-value function that updates continually. Using a separate target network makes divergence unlikely, because it adds a delay between the time that Q values are updated and the time that the target Q_T values are updated.

1.0.5 5

As mentioned earlier, experience replay allows better data effeciency- the same data is seen by multiple steps, and is consequently used to update the weights of the paramenters at different steps. Experience replay may also help to prevent overfitting by allowing the agent to learn from data generated by previous versions of the policy.

1.0.6 6

The number of scenarios associated with a reward is usually sparse during learning. Using a prioritised replay- selecting certain samples with a higher probability than others, we can thus

counteract this inherent bias such that the data points are sampled with a fair distribution of actions directed better towards maximizing the expected reward. Additionally, this also reduces the variance of the model, resulting in faster convergence of traditional SGD models, and thus lower training time. This ensures that the parameter multiple leaners can read the same parameters at a given point in time.

1.0.7 7

APEX vs Gorilla: Similarities: - Both systems use a distributed setup- several actors and learners work in parallel. - Both APEX and Gorilla use experience replays as a method of using data more effeciently, and to train faster by reducing the variance of the model. - Both use a 2 step mechanism in their algorithm: actors play out the scenario, evaluates the policy on the actual environment and a learner uses this data to learn new parameters.

Differences: - Gorilla uses a traditional experience replay based system, whereas APEX uses a priority based system. - APEX uses a shared centralised memory- Datapoints are sampled from datasets generated by multiple learners playing out and evaluating different scenarios. Gorilla shares its model parameters, and not the experiences themselves. - In Apex, learning is independent of the policy- this allows different actors to play out widely different scenarios. With Gorilla, the parameters are shared. Thus, the scenarios generated may not differ from each other by a great extent (only by our choice of ϵ, our random walk parameter).

[ ]: