Navigation Using Deep Reinforcement Learning

Mateo Neira

June 30, 2020

Introduction

The goal of reinforcement learning is to learn a good policies for sequential decision problems by optimizing a cumulative future reward signal. To solve sequential decision problems we can learn estimates for the optimal value of each action, defined as the expected sum of future rewards when taking that action and following the optimal policy thereafter. Under a given policy $\pi = P(a|s)$, the true value of an action a in a state s is

$$Q_\pi(s, a) \equiv \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi],$$

Where $\gamma \in [0, 1]$ is a discount factor that trades off the importance of immediate and later rewards. The optimal value is then $Q_*(s, a) = \max_\pi Q_\pi(s, a)$. An optimal policy can be derived by selecting the highest valued action in each state..

Q-Learning [6], a form of temporal-difference learning, can be used to to learn optimal action values. However, in many cases learning the entire state space is combinatorial and enormous. In these cases a better goal is to find a good approximate solution. Various methods have been proposed to learn a parameterized value function $\hat{Q}(s, a; \theta_t) \approx Q_*(s, a)$. Although many different function approximation methods could be used, artificial neural networks have been shown to be successful in a wide range of tasks. Here we implement Deep Q-learning, which uses a multilayered neural network, as well additional modifications that have been proposed in the literature and compare the results for learning a navigation task in a continuous state space environment.

0.1 Environment

To test the algorithms an agent is trained to navigate a square world using The Unity Machine Learning Agents Toolkit [1]. The goal of the agent is to collect as many yellow bananas while avoiding blue ones. A reward of +1 is received for every yellow banana collected, and a rewards of -1 for every blue one. The agent perceives a continuous 37 dimensional state space that contains the agent's velocity, and ray-based perception of objects along the agents forward direction. The agent can take 4 discrete actions; move forward, move backward, turn left, or turn right. The task is episodic, and in order to solve the environment the agent must get an average score of +13 over 100 consecutive episodes.

1 Models

1.1 Deep Q Learning

Deep Q Learning (DQN) [2], leverages advances in deep learning to learn policies from high dimensional sensory input. The algorithm combines Q-learning with a flexible deep neural network to approximate the Q function. Although reinforcement learning is known to be unstable or to diverge when a nonlinear function approximator is used to represent the Q function, DQN address these instabilities by incorporating two key ideas: Experience replay, and a fixed target Q.

Experience Replay: using a multilayered artificial neural network to represent the Q function can be thought of as a supervised learning problem, in which case the input data must be independent and identically distributed (*i.i.d.*) in order for the model to me able to generalize well. In Q-learning this becomes problematic because there are correlations present in the sequence of observations. This is solved in DQN by storing the sequence of experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t in a data buffer $D_t = (e_1, ..., e_t)$. During

learning the Q-learning updates are applied on samples of experiences $(s, a, r, s') \sim U(D)$, drawn uniformly at random from the data buffer, removing correlations in the observation sequence.

Fixed target Q: in Q-learning, both input and target values change constantly during training process, making supervised learning of the Q function using multilayered networks unstable. To solve for this problem, DQN uses a iterative update that adjusts the action-values (Q) towards target values that are only periodically updated. Two networks are created θ and θ', the first network is used to retrieve Q values while the second includes all updates during training. The target network θ' is only updated with the θ parameters after a fixed number of steps, and the Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_a Q(s', a'; \theta_i^{'}) - Q(s, a; \theta_i)_2] \tag{1}$$

1.2 Double Deep Q Networks

DQN has been shown to overestimate action values, resulting in poorer policies. The max operator in DQN (1) uses the same values both to select and to evaluate an action, making it more likely to select overestimated values, resulting in overoptimistic value estimates. Double DQN [4] reduces this overestimation by decomposing the max operation in the target into action selection and action evaluation:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma Q(s', \arg\max_a Q(s', a'; \theta_i), \theta_i^{'}) - Q(s, a; \theta_i)_2] \tag{2}$$

Notice that the selection of the action, in the $\arg\max$, is still due to the online weights θ_t (2). This means that, as in Q-learning, we are still estimating the value of the greedy policy according to the current values, as defined by θ. However, we use the second set of weights θ' to fairly evaluate the value of this policy. The update to the target network stays unchanged from DQN, and remains a periodic copy of the online network θ.

1.3 Prioritized experience replay

Prioritized experience replay (PER) [3] was developed to make learning more efficient by replaying important transitions more frequently. As mentioned in section 1.1, DQN uses a large sliding window replay memory, sampled uniformly at random, to overcome the correlations present in sequential experiences. The key idea behind prioritized experience replay is that an agent can learn more efficiently from some experiences than others.

PER replays experiences with high expected learning progress more frequently; the expected learning progress is measured by the magnitude of their temporal-difference (TD) error. Here we use a proportional prioritization with stochastic sampling where the probability of sampling experience i is given by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{3}$$

where p_i>0 is the priority of transition i, which is equal to the TD error δ_i, plus a small positive constant ϵ that prevents edge-cases being revisited once the TD error is zero. The exponent α controls how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

The prioritized experience replay introduces bias into the stochastic update rule, since the original Q-learning update is derived from an expectation over all experiences. This is corrected by using an importance-sampling (IS) weights:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \tag{4}$$

that fully compensates for the non-uniform probabilities $P(i)$ if $\beta = 1$. These weights are incorporated into the Q-learning update by replacing δ_i with $w_i\delta_i$.

1.4 Dueling Deep Q-Network

In the models presented previously, the non-linear function approximator is a conventional multilayered neural network. Dueling Deep Q-Network [5] improves on these models by introducing a neural network that is

Algorithm	# episodes
DQN	310
DDQN	434
PER	443
Dueling Network	481

Table 1: Number of episodes learn to solve the environment.

better suited for model-free reinforcement learning. The dueling architecture separates the representation of the state values:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s)}\left[Q_\pi(s,a)\right] \tag{5}$$

and state-dependent actions advantages:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{6}$$

The two streams are combined via a aggregating layer to produce an estimate of the Q-function. The key insight behind this architecture if that for many states it is unnecessary to estimate the value of each action choice. Since the value of the dueling network is a Q function, it can be trained using the same frame as DQN and DDQN.

2 Results

For our experiment, we build on each model sequentially, so for the last model *Dueling Deep Q-Network* it includes both *DDQN* and *PER*. The 4 model implementations described above were tested on the navigation task with a fixed set of hyperparameters. Our network architecture is also fixed, changing only in the case of the Dueling Deep Q-network. The architecture is composed of two fully connected layers with 64 units each and one output stream equal to the dimension of the actions space, while the architecture of the dueling deep q-network has a value and advantage output streams.

All experience transitions are stored in a sliding window memory that retains the last 10^4 transitions. The algorithm processes minibatches of 128, with a learning rate of 0.001 and a discount factor $\gamma = 0.99$. The target network θ' is updated every 4 learning steps, with a soft update $\theta' = \tau\theta + (1-\tau)\theta'$, with $\tau = 0.001$. In the case of *PER*, we set $\alpha = 0.6$ and $\beta = 0.4$, as per the original paper.

The number of episodes it took to solve the environment are shown in table 1. *DQN* outperformed the other implementations, learning to solve the environment after 310 episodes. learning speed declined with each additional modification to the *DQN* algorithm. Summary plots of learning can be seen in figure 1 for each implementation, and a summary plot in figure 2.

The decline in performance could be attributed to requirement of fine-tuning of hyperparameters for every modification to the algorith. For example, According to [3] because prioritized replay picks high-error transitions more often, the typical gradients are larger, so the step-size parameter should be scaled accordingly. This might be one of the reasons we don't see an improvement learning.

3 Conclusions

We implemented and tested a deep q-network, and several improvements that have been proposed in the literature, to train an agent to solve a navigation task in a continuous state environment with discrete action space. The agent is able to learn to complete the navigation task in the environment within a relatively small amount of training steps. Having a fixed set of hyper-parameters we did not observe an increase in learning speed with the different improvements that have been proposed, suggesting that fine-tuning is needed for each case.

References

[1] Arthur Juliani et al. "Unity: A general platform for intelligent agents". In: *arXiv preprint arXiv:1809.02627* (2018).

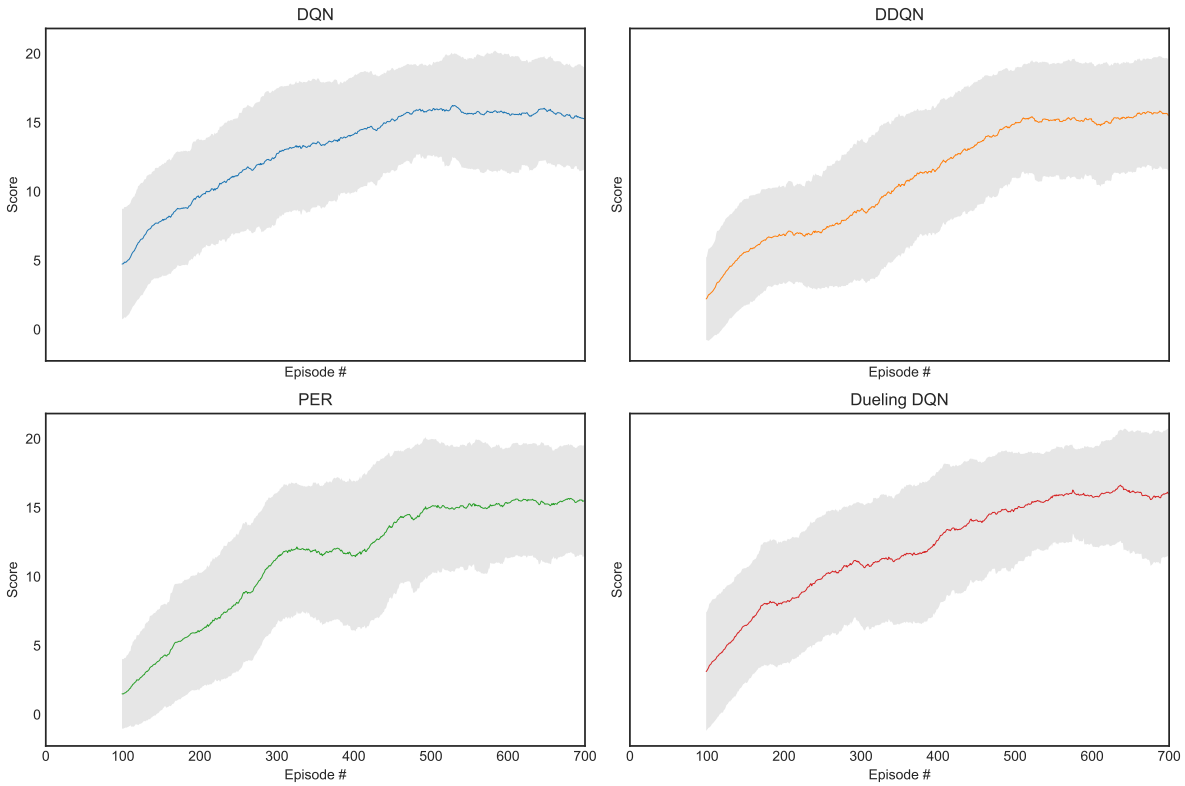

Figure 1: Training speed of models showing moving average and variance of 100 episodes during training.

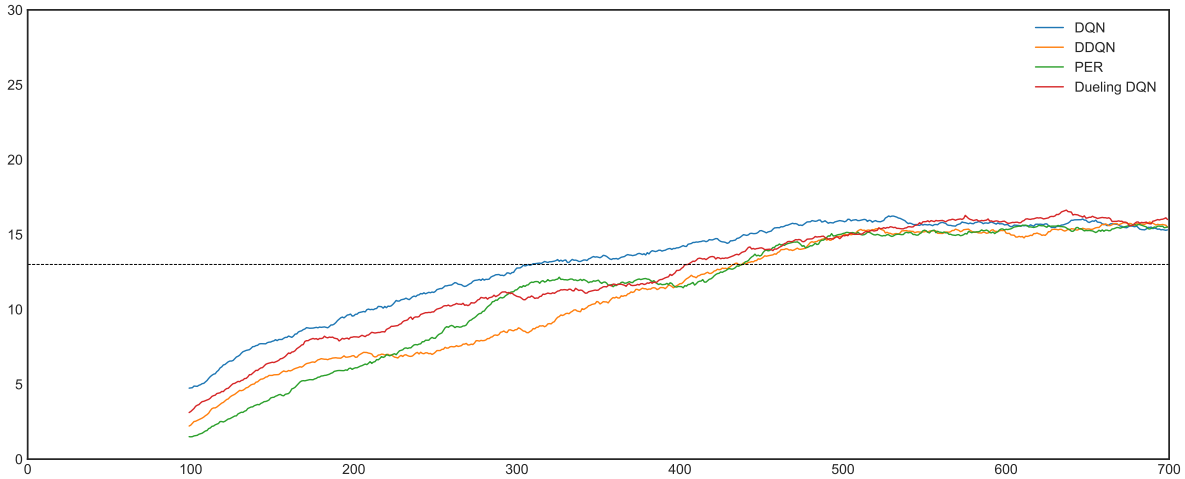

Figure 2: Summary plot of learning speed showing moving average of 100 episodes over training.

[2] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[3] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).

[4] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double q-learning”. In: *Thirtieth AAAI conference on artificial intelligence*. 2016.

[5] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. In: *International conference on machine learning*. 2016, pp. 1995–2003.

[6] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.