Rainbow Combining Improvements in Deep Reinforcement Learning

Udacity

Christian Leininger

Supervisor:
Advisor:

1 Introduction

This work is about solving the First project of the deep reinforcement learning course. The goal is to train an agent to collect bananas in a squarde world. In this episodic task the agent must get an average score of + 13 over 100 consecutive episodes. By collecting a yellow banana the reward is +1 and for a blue banana -1.

2 Environment

The Environment is given from Unity Machine Learning Agents (ML-Agents). The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction and the agent has 4 discrete actions forward, backward, left and right

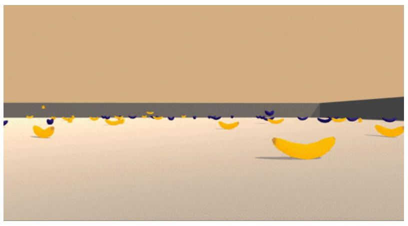
Figure 1: The banana enviroment

3 Learning algorithm

3.1 Double DQN

The DQN algorithm is known to sometimes learn unrealistically high action values because it includes a maximization step over estimated action values, which tends to prefer overesti-

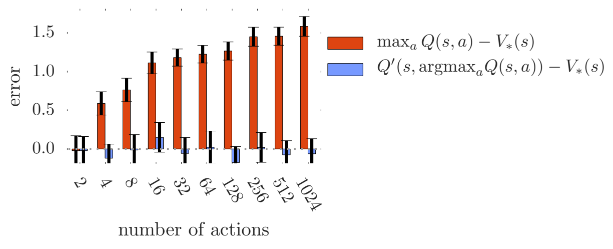

Figure 2: The orange bars show the bias in a single Q-learning update when the action values are $Q(s,a) = V_*(s) + \epsilon_a$ and the errors $\{\epsilon_a\}_{a=1}^m$ are independent standard normal random variables. The second set of action values Q', used for the blue bars, was generated identically and independently. All bars are the average of 100 repetitions.

mated to underestimated values. In the original paper it was shown that this overestimates are increases with the number of action.
DQN were indeed leading to poorer policies and that it is beneficial to reduce them. In Standard DQN the max part in estimating the Q-values and selecting the action are using the same Network weights. Those weights we can see as random variables. In my probability lecture at my university it was shown the following term $E[max(X_1, X_2] \geq max(E[X_1], E[X_2]$. We have to de-correlate the random variables. This is done by using different networks for the Q-values and the action selection. Since we have already a local and target network. This method was proposed in [5].
Its update is the same as for DQN, but replacing the target Y_t^{DQN} with

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + Q(S_{t+1},_a Q(S_{t+1}, a; \theta_t), \theta_t^-) \,.$$

In comparison to Double Q-learning, the weights of the second network θ_t' are replaced with the weights of the target network θ_t^- for the evaluation of the current greedy policy. The update to the target network stays unchanged from DQN, and remains a periodic copy of the online network. This version of Double DQN is perhaps the minimal possible change to DQN towards Double Q-learning.The goal is to get most of the benefit of Double Q-learning, while keeping the rest of the DQN algorithm intact for a fair comparison, and with minimal computational overhead.

3.2 Prioritized Experience Replay

In the Standard Experience Replay the samples are selected with an uniformed distribution. Every sample has the same probability to be select to learn from it. The idea from Prioritized Experience Replay [4]. The TD- error is a measure of the importance of a sample. If we would just greedily take the samples with the highest TD error, it was shown that this would lead in to several issues that's why they proposed the following probability to sample the transition i

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{1}$$

where $p_i > 0$ is the priority of transition i. The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

Algorithm 1 Double DQN with proportional prioritization

```
1: **Input:** minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and $\beta$, budget $T$.
2: Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
3: Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
4: for $t = 1$ to $T$ do
5:    Observe $S_t, R_t, \gamma_t$
6:    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
7:    if $t \equiv 0 \mod K$ then
8:       for $j = 1$ to $k$ do
9:          Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
10:         Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
11:         Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
12:         Update transition priority $p_j \leftarrow |\delta_j|$
13:         Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
14:      end for
15:      Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
16:      From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
17:   end if
18:   Choose action $A_t \sim \pi_\theta(S_t)$
19: end for=0
```

Prioritized replay introduces bias because it changes this distribution in an uncontrolled fashion, and therefore changes the solution that the estimates will converge to (even if the policy and state distribution are fixed).
We can correct this bias by using importance-sampling (IS) weights

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

that fully compensates for the non-uniform probabilities $P(i)$ if $\beta = 1$. These weights can be folded into the Q-learning update by using $w_i\delta_i$ instead of δ_i As data structure the sum tree was used
This Algorithm was proposed in the paper:

3.3 Dueling Network Architectures

The dueling network[6] represents two separate estimators: one for the state value function and one for the state-dependent action advantage function.
We define another important quantity, the *advantage function*, relating the value and Q functions: $\mathrm{A}^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). Notethat \mathrm{A}^\pi(s,a) a \sim \pi(s) = 0$. Intuitively, the value function V measures the how good it is to be in a particular state s. The Q function, however, measures the the value of choosing a particular action when in this state. The advantage function subtracts the value of the state from the Q function to obtain a relative measure of each action.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right). \quad (2)$$

This is an change on the network architecture and is implemented in the model.py file.

3.4 Asynchronous Methods

The following Algorithm was propoesed in the research paper "Asynchronous Methods for Deep Reinforcement Learning" [3]

It uses the n-step method where n is a hyper parameter that indixate of the amount of real rewards the agents gets before uses the approximate Q-value.

Algorithm 2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

```
0: // Assume global shared parameter vector $\theta$.
0: // Assume global shared target parameter vector $\theta^-$.
0: // Assume global shared counter $T = 0$.
0: Initialize thread step counter $t \leftarrow 1$
0: Initialize target network parameters $\theta^- \leftarrow \theta$
0: Initialize thread-specific parameters $\theta' = \theta$
0: Initialize network gradients $d\theta \leftarrow 0$
0: repeat
0:   Clear gradients $d\theta \leftarrow 0$
0:   Synchronize thread-specific parameters $\theta' = \theta$
0:   $t_{start} = t$
0:   Get state $s_t$
0:   repeat
0:     Take action $a_t$ according to the $\epsilon$-greedy policy based on $Q(s_t, a; \theta')$
0:     Receive reward $r_t$ and new state $s_{t+1}$
0:     $t \leftarrow t + 1$
0:     $T \leftarrow T + 1$
0:   until terminal $s_t$ or $t - t_{start} == t_{max}$
0:   $R = \begin{cases} 0 & \text{for terminal } s_t \\ \max_a Q(s_t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$
0:   for $i \in \{t-1, \ldots, t_{start}\}$ do
0:     $R \leftarrow r_i + \gamma R$
0:     Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \frac{\partial\left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$
0:   end for
0:   Perform asynchronous update of $\theta$ using $d\theta$.
0:   if $T \mod I_{target} == 0$ then
0:     $\theta^- \leftarrow \theta$
0:   end if
0: until $T > T_{max}$ =0
```

3.5 Noisy Networks for Exploration

By adding noise to the network weights this method improves the performance in many cases. The parameters of the noise are learned with gradient descent along with the remaining network weights. NoisyNet is straightforward to implement and adds little computational overhead. [2]
Consider a linear layer of a neural network with p inputs and q outputs, represented by y = wx + b, where $x \in \mathbb{R}^p$ is the layer input, $w \in \mathbb{R}^{q\times p}$ the weight matrix, and $b \in \mathbb{R}^q$ the bias. The corresponding noisy linear layer is defined as: y$(\mu^w + \sigma^w \odot^w)x + \mu^b + \sigma^b \odot^b, where \mu^w + \sigma^w \odot^w$ and $\mu^b + \sigma^b \odot^b$ replace w and b in Eq. (3.5), respectively. The parameters $\mu^w \in \mathbb{R}^{q\times p}$, $\mu^b \in \mathbb{R}^q$, $\sigma^w \in \mathbb{R}^{q\times p}$ and $\sigma^b \in \mathbb{R}^q$ are learnable whereas $^w \in \mathbb{R}^{q\times p}$ and $^b \in \mathbb{R}^q$ are noise random variables (the specific choices of this distribution are described below). We provide a graphical representation of a noisy linear layer in Fig. **??** (see Appendix **??**).

3.6 A Distributional Perspective on Reinforcement Learning

In this paper we take away the expectations inside Bellman's equations and consider instead the full distribution of the random variable Z^π. From here on, we will view Z^π as a mapping from state-action pairs to distributions over returns, and call it the *value distribution*. Our first aim is to gain an understanding of the theoretical behaviour of the distributional analogues of the Bellman operators, in particular in the less well-understood control setting. The reader strictly interested in the algorithmic contribution may choose to skip this section. [1]

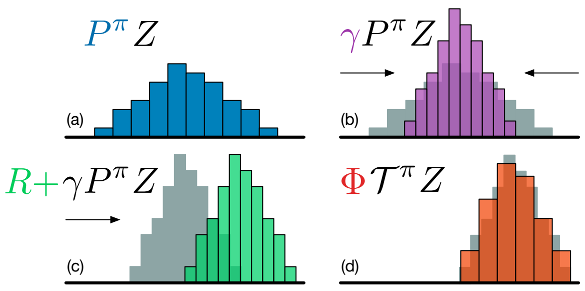

Figure 3: A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section **??**).

The Wasserstein Metric is the tool to analysis the difference between cumulative distribution functions

4 Hyper-parameter search

In the file hyperparametersearch some of the them are tested to find the optimal value. The epsilon decay is a measure of how fast the epsilon value decreases from 1 to is chosen min value. which is 0.01 percent. The epsilon value is the threshold for if the next action is random or the one with the max Q-value. It seem that the task is easy and the agent is already after a few episodes ready to take mostly greedy actions. In order to solve the task as soon as possible. The other parameters are the alpha and beta value from the Prioritized Experience Replay and different values for the n$_steplookahead$

5 Results

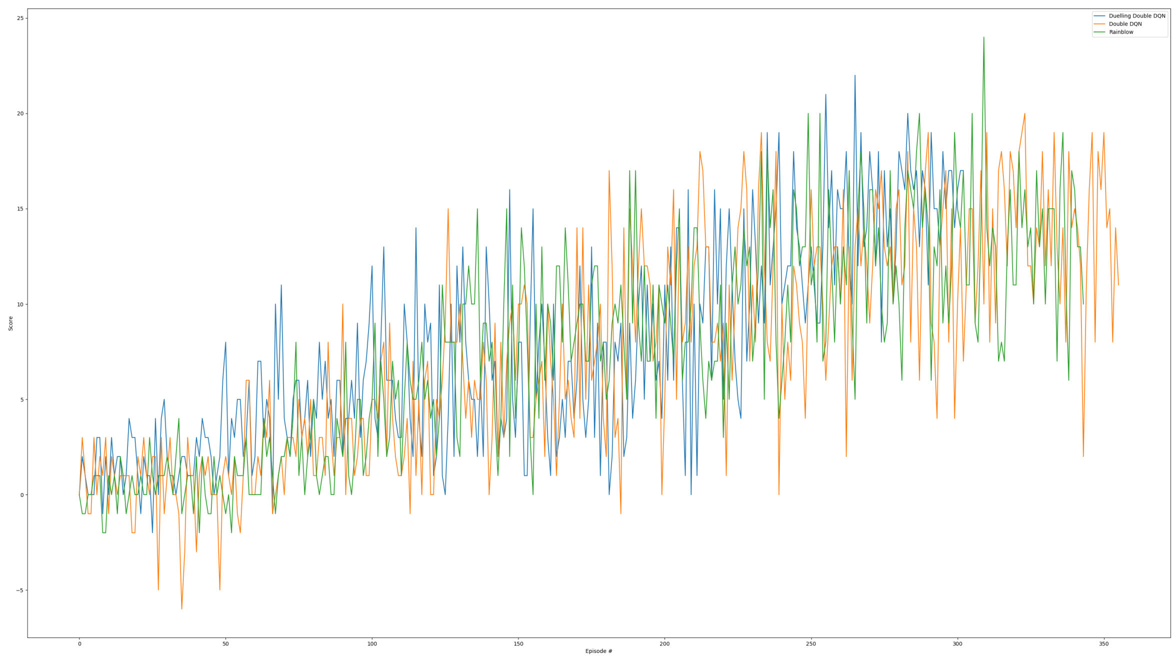

Figure 4: Scores of the different agents

The figure shows the performance 3 different algorithm in on run. Since the result is a Random variable which has an expectation and variance. By running many of them we could get an approximation of them.

6 Future ideas to improve the performance

Try more different parameters to find the optimal values. We could combine the reinforcement idea with planning to improve sample efficiency.

References

[1] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-*

Volume 70, pages 449–458. JMLR. org, 2017.

[2] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[5] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[6] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.