Weekly report (July 17, 2018 - July 21 , 2018)

Wyk
182507635559@163.com

Abstract

This week, I detailedly read the papers *Deep Reinforcement Learning with Double Q-learning* and *PRIORITIZED EXPERIENCE REPLAY*. And try to implement them based on my previous works.

1 Learn some extensions of DQN

I've learned and implement the basic DQN model in the previous weeks. Since the DQN first published, many extensions have been proposed that enhance its speed or stability. This week I learned some of them following the advices of Mr.Hua and Mr. Song.

1.1 Double DQN

Reinforcement Learning with Double Q-learning (DDQN; van Hasselt, Guez, and Silver 2016) addresses an overestimation bias of Q-learning , by decoupling selection and evaluation of the bootstrap action.

1.1.1 Paper reading notes

The max operator in standard Q-learning and DQN, uses the same values both to select and to evaluate an action.:

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$

This includes a maximization step over estimated action values, which tends to prefer overestimated to underestimated values, resulting in unrealistically high action values. And according to previous results, it have been attributed to insufficiently flexible function approximation and noise.

To address this issue, the idea of Double Q-learning is to reduce overestimations by decomposing the max operation in the target into action selection and action evaluation. We evaluate the greedy policy according to the online network(evaluation network), but using the target network to estimate its value. In reference to both Double Q-learning and DQN, we refer to the resulting algorithm as Double DQN. Its update is the same as for DQN, but replacing the target Y with :

$$Y_t^{DDQN} = R_{t+1} + \gamma Q_{target}(s_{t+1}, \arg\max_a Q_{eval}(s_{t+1}, a))$$

Argmax return the action that maximize the the Q-eval network. this means we are estimating the value of the greedy policy according to the current values with the Q_eval network. However, we use the Q_target network to fairly evaluate the value.
This algorithm not only yields more accurate value estimates, but leads to much higher scores on several games. This demonstrates that the overestimations of DQN were indeed leading to poorer policies and that it is beneficial to reduce them.

1.1.2 Implementation

Double DQN is an update version of DQN, so I can implement it base on the basic DQN code I wrote. And since I'm using the code, I also update it into a better version. Both code can

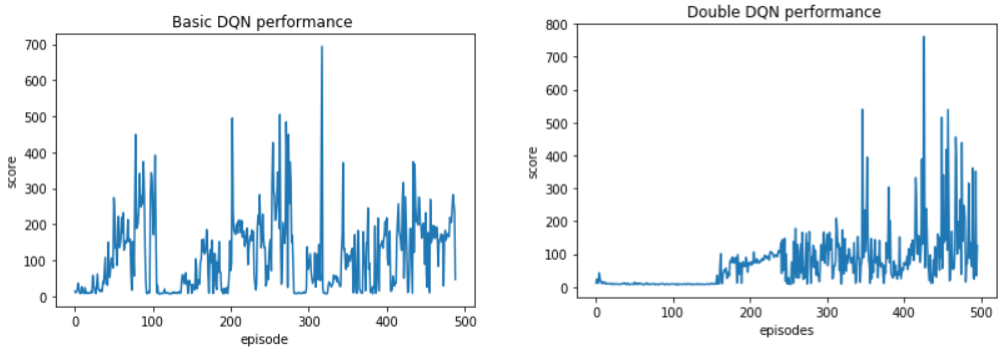
be seen on my Github `https://github.com/W-yk/Deep_learning/tree/master/DQN/DQNPROJECT`.

The implementation detail base on the old model have two main parts:

1. create a new network called target network.
2. change the label as the paper describes.

For part one, according to the paper we need a new network to determine the action value, and the old one is for the greedy policy. What we need to do is create the target model by calling the same function used for building the old one, and update it ever C step with the weights of the model we actually train on.
The second part is to modify the target value we feed to the network in order to prevent overestimating. According to the paper we need to decouple selection and evaluation of the target value. Using the online model to choose an action for the greedy policy and get the corresponding action value with the target model's perdition. the bootstrap action.

The performance of both model is shown below:



The double DQN model is more robust and stable as shown in the picture, though may not be obvious with only 500 episodes. I didn't train them for more because of the low training speed, but with more time trained the difference will be much obvious.

1.2 Prioritized experience replay

PRIORITIZED EXPERIENCE REPLAY (Schaul et al. 2015) improves data efficiency, by replaying more often transitions from which there is more to learn.

1.2.1 Paper reading notes

Previously we talked about Experience replay. It gives DQN agents the ability to remember and replay memories. And with experience stored in a replay memory, it becomes possible to break the temporal correlations by mixing more and less recent experience for the updates, which stabilized the training of a value function.

It also have some problems. For human memory have strong, important ones, but for DQN agent the experience are all the same when doing experience replay. So when the environment have very few rewards, the positive experience will be small in the memory pool, which makes the learning speed very slow.

Prioritized experience replay means we sample batch according to priority rather than sample randomly. And the priority is depend with something called TD-error, which indicates how surprising or unexpected the transition is: specifically, how far the value is from its next-step bootstrap estimate, in other words, how much the agent can learn from it.

But if we just perform a greedy policy on the prioritized experience, One consequence is that transitions that have a low TD error on first visit may not be replayed for a long time, and, greedy prioritization focuses on a small subset of the experience: errors shrink slowly, especially when using function approximation, meaning that the initially high error transitions get replayed frequently.

This lack of diversity that makes the system prone to over-fitting. So in practice we sample it randomly according to the priority distribution.

1.2.2 Implementation

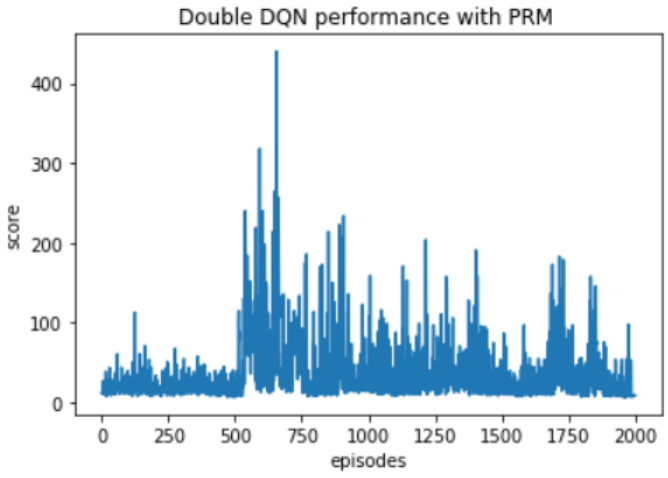
First the algorithm form the paper:

Algorithm 1 Double DQN with proportional prioritization

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

In order to efficiently searching for the experience with the highest priority, the paper use a sum tree to store the transitions.For a quicker speed to sample from the prioritized data distribution. It took me quite some time to finally understand it and the code to build it referenced *MorvanZhou*'s code on his github.

After some time spent coding, I finally completed the agent. But the results were not something I expected.



I really don't get the reasons behind this, PER were meant to speed up the training process. But with more episodes I spent, the performance is even below DQN model with less training time. I'm confused but couldn't find the mistake in my code.

https://github.com/W-yk/Deep_learning/blob/master/DQN/DQNPROJECT/DoubleDQN_with_prioritized_replay_memory.py

2 Self-supervised learning

I watched the Yann LeCun's speech *Learning World Models: the Next Step towards AI* at IJCAI 2018 while getting stuck for above works, And in it Yann specially emphasize a method call self-supervised learning, I'm very interested with this idea. So after finishing the speech I search for the related works, and found the following paper: *Curiosity-driven Exploration by Self-supervised Prediction*. In which an agent is trained to play Super Mario Bros.

(a) learn to explore in Level-1 (b) explore faster in Level-2

In this paper, a method called Curiosity-Driven Exploration is introduced to solve extremely sparse rewards cases like finding the right way, playing Super Mario Bros and some many real world scenarios. As the paper describes, in such cases, curiosity can serve as an intrinsic reward signal to enable the agent to explore its environment and learn skills that might be useful later in its life. They formulate curiosity as the error in an agents ability to predict the consequence of its own actions in a visual feature space learned by a self-supervised inverse dynamics model. And instead of hand-designing a feature representation for every environment, they try to come up with a general mechanism for learning feature representations such that the prediction error in the learned feature space provides a good intrinsic reward signal. And this can be called self-supervised learning, as the agent produces labels it need to learn with its own model.

Unlike DQN, in this model, agents are not doing explorations randomly but choose to do things they are unfamiliar with. Curiosities matter so much for us humans, I think they do for agents too. I hope to further study this sometime.

3 Summary

After consulting the teachers, I changed my plan for this week, and start preparing for a mini project. The work I mentioned before is very challenging, but I already had the psychological preparation. I'm sorry if this week's report seems rough, for time spent with above works are quite much.

Plans for next week

1. Try to determine the problem in my implementation of PRIORITIZED EXPERIENCE REPLAY,
2. Start learning algorithms and data structures.