

Direct Future Prediction

September 18, 2017

Introduction

After learning the difference between Q-learning and SARSA. I started to look for some more interesting and challenging problems. Among all, the deep reinforcement learning paper that I have read, the one that I found more interesting is [1]. The DFP architecture presented in this paper won the Visual Doom AI Competition in 2016, while it is much simpler than other solutions.

Respect to a traditional DQN, the authors focus on re-building the reward signals. They propose to replace the traditional scalar reward with a more frequent high-dimensional reward vectors. As we will see, this change provides two main advantages:

1. contrary to a scalar reward, a vector provides richer information
2. a more frequent reward provides to the agent more experience on which learning is more intensive.

Instead of following the usual RL setup composed by a monolithic state and a scalar reward, the authors consider two streams:

1. a high-dimensional observation stream s_t
2. and, a low-dimension measurements stream m_t (key information about the state)

while assuming that the overall goal can be expressed as a linear combination of a set of future measurements.

$$\begin{aligned} F_t &= [m_{t+\tau_1} - m_t, \dots, m_{t+\tau_n} - m_t] \\ \mathbf{g}_t &= u(F_t) = w_1(m_{t+\tau_1} - m_t) + \dots + w_n(m_{t+\tau_n} - m_t) \end{aligned} \quad (1)$$

where F_t represent a set of n future measurements which are predicted by the agent.

In a Doom-like environments, it is possible to think about s_t as the raw pixel, while the measurements are the life score, the amount of weapons or the amount of kills done by the agent. The key idea behind this setting is that the agent is trained to predict the effect of the different actions on the feature measurements

w.r.t. the current goal ($F_t = f(s_t, m_t, A|g_t)$), which is easier than predicting the entire next state (specially for high dimensional inputs). The trick is that, due to the goal's assumption, predicting F_t is all what the agent needs to know in order to choose the best action; moreover the measurements stream is the only part of the state that we need to predict (traditional RL assume that the entire state have to be predicted). This difference between reward function and measurements gives an other advantages: at testing time it is possible to define a different goal, as soon it is a linear combination of the measurements, without retraining. This transforms the traditional RL problem into a SL problem, where the supervision is provided by the environments. In this case the authors proposed to used supervised learning in a Monte Carlo fashion, instead of using a TD-error.

Model

At each time stamp t the agent receives an observation $\mathbf{o}_t = \langle s_t, m_t \rangle$, which is the concatenation of the sensory stream and the measurements stream. It is possible to learn the future measurements through a parameterized function:

$$\mathbf{p}_t^a = f_\theta(\mathbf{o}_t, a, \mathbf{g}_t) \quad (2)$$

At test time, combining Eq. 1 and Eq. 2, it is possible to choose the best action which maximize the predicted reward as:

$$a_t = \arg \max_{a \in A} u(f_\theta(\mathbf{o}_t, a, \mathbf{g}_t)) \quad (3)$$

Training

The agent is trained to predict the future measurements, but the target value is obtained by the environment through experience. Initially the agent act randomly and start to collect experience in a memory $M = \{\dots, \langle \mathbf{o}_i, a_i, \mathbf{g}_i, F_i \rangle, \dots\}$. This experience is used to form mini-batch and minimize a regression loss function

$$\mathcal{L}(\theta) = \sum_{i=1}^N \|f_\theta(\mathbf{o}_i, a_i, \mathbf{g}_i) - F_i\|^2 \quad (4)$$

while following a policy function; please note that in this experiments I used a softmax with temperature function instead of the proposed ϵ -greedy policy.

The overall architecture is shown in Fig. 1

Environments

As for the previous case I had setup a 5×5 2-D Maze environments, but in this case we have a drone that is supposed to make some delivery. Every time the drone makes a delivery, a new position for the delivery point gets generated.

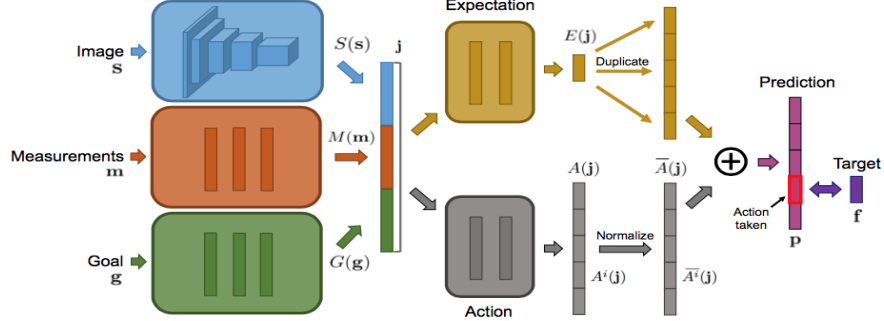


Figure 1: Architecture

This process goes on until a fixed number of steps is reached. To make the task more realistic, the drone has a battery which drops by 0.025 per each moves that the drone does. The agent have to learn when to move the drone to the delivery point or goes to the spot where it can recharge its battery.

An example of the environments is given in Fig. 1

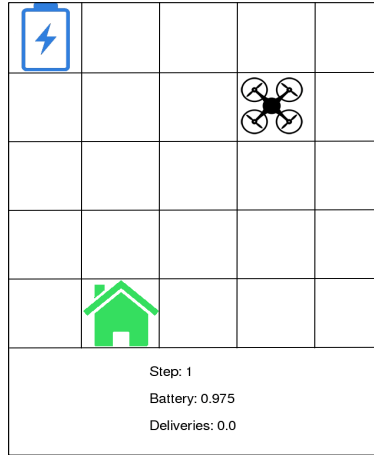


Figure 2: Architecture

In this toy example we have two measurements:

- the number of delivery done,
- and the level of the battery.

Similarly we have two possible goals:

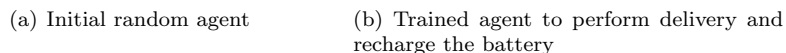
1. maximize the delivery

2. recharge the battery when the level is under 30%

Experiments

I run the experiments on my laptop, creating an asynchronous version of the algorithm based on the update strategy proposed in the A3C. This allowed me to run 4 networks on my CPU simultaneously and generate enough experience in a short amount of time. I'm not able to parallelize it on GPUs since I need a GPU for each instance of the network and I don't have any good GPU, so not much to do about it right now.

Anyway, in Fig. 3(a) it is possible to see how the drone is not able to navigate the environment. It was just lucky enough to recharge the battery by chance and not go out of energy. To the other side, in Fig. 3(b) it is possible to see a smart agent that is able to maximize the number of deliveries while keeping the battery charged.



(a) Initial random agent (b) Trained agent to perform delivery and recharge the battery

Figure 3: Drone experiments

I'm quite fascinated about this architecture and I'm planning to do some other experiments. One possibility is to see if it is able to learn NLP based goals and change the goals at test time. I find this goal interesting because the agent has to learn a bit of language understanding while at test time we can change the goal to see how it performs in the new task.

Moreover, I would also see if it is possible to teach the agent to follow a mixture of goals. If the delivery is "close" to the recharge point, it might be worth to choose the path that passes over the battery as well, instead of the one that doesn't.

References

- [1] DOSOVITSKIY, A., AND KOLTUN, V. Learning to act by predicting the future. *CoRR abs/1611.01779* (2016).