LEARNING TO PREDICT BY THE METHODS OF TEMPORAL DIFFERENCES

From Reinforcement to Deep Reinforcement Learning

28-03-2019

Matthia Sabatelli[1]

[1]Montefiore Institute, Department of Electrical Engineering and Computer Science, Université de Liège, Belgium

Overview

1. Reinforcement Learning
2. Deep Reinforcement Learning

Reinforcement Learning

Supervised Learning setting

- How does Reinforcement Learning (RL) differ from other machine learning paradigms?

In supervised learning (SL) we have $\mathcal{X}_t$, $\mathcal{Y}_t$, and a probability distribution $p_t(x, y)$ defined over $\mathcal{X}_t \times \mathcal{Y}_t$. The goal is to build a function $f : \mathcal{X}_t \to \mathcal{Y}_t$ that minimizes the expectation over $p_t(x, y)$ of a given loss function ℓ assessing the predictions made by f:

$$E_{(x,y)\sim p_t(x,y)}\{\ell(y, f(x))\}, \tag{1}$$

We then learn this function via input-output pairs $LS_t = \{(x_i, y_i)|i = 1, \ldots, N_t\}$ drawn independently from $p_t(x, y)$.

However the RL setting is much different

- We do not assume any supervision but only a **reward** signal.
- Feedback to the learner can be delayed and is not instantaneous
- Time matters and earlier decisions might affect later ones

More generally we have to deal with time dependencies and causality

RL-cooking recipe

The core high level concepts of a RL system are:

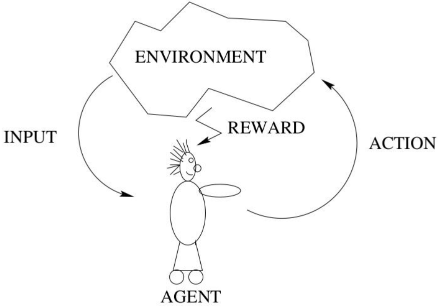
- An Environment
  - A set of possible states $\mathcal{S}$
  - A set of possible actions $\mathcal{A}$
- An Agent
  - Policy $\pi : s \rightarrow a$
  - Value function ($V(s_t)$ or $Q(s_t, a_t)$)
  - Model (optional)
- A Reward signal $\Re(s_t, a_t, s_{t+1})$,

Depending on the RL set-up some (or all) of these elements can be inter-connected, and they define what the RL-agent needs to **learn**!

Markov Decision Processes

Markov decision processes are based on Multi-Armed Bandit theory and include the elements which we have seen before:

- A set of possible states $\mathcal{S}$
- A set of possible actions $\mathcal{A}$
- A Reward signal $\Re(s_t, a_t, s_{t+1})$,
- A transition probability distribution $p(s_{t+1}|s_t, a_t)$



Markov Property

- The current state and action give **all the necessary information** for predicting to which next state the agent will step
- The reward obtained at r_t is only determined by the **previous** state and action

$$\mathrm{p}(\mathrm{r}_t = \Re | s_t, a_t) = p(r_t = \Re | s_t, a_t, ..., s_1, a_1)$$

Thus, for predicting the future it does not matter how the agent arrived in a particular state

Value Functions

In Value-based RL we want our agent to predict the *'goodness'* of each state, and we can do this in two ways:

$$\begin{aligned}\mathbf{V}^{\pi}(\mathbf{s_t}) &= E_{\pi}\left[G_t | S_t = s\right] \\ &= E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i \Re_{t+i+1} \middle| S_t = s, \pi\right]\end{aligned} \tag{2}$$

or by learning a State-Action Value function

$$\mathbf{Q}^{\pi}(\mathbf{s_t}, \mathbf{a_t}) = E_{\pi}\left[\sum_{i=0}^{\infty} \gamma^i \Re_{t+i+1} \middle| S_t = s, A_t = a\right] \tag{3}$$

Both functions express the **desirability** of being in a particular state and are both dependent on π

Why do we need Value functions?

Why should we care about them?

- Values encode the **knowledge** of the agent
- If they are precise the agent will know everything he needs to know
- It will **predict** the reward signals coming from the environment, therefore choosing appropriate **actions**
- This leads to **Optimal Policies** π^* which satisfy the *Bellman* optimality equation .

$$Q^{\pi}(s_t, a_t) = \sum_{s_{t+1}\in\mathcal{S}} p(s_{t+1}|s_t, a_t)\Big(\Re(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}\in\mathcal{A}} Q^{\pi}(s_{t+1}, a_{t+1})\Big).$$

$$\pi^*(s_t) = \underset{a\in\mathcal{A}}{argmax}\ Q^{\pi}(s_t, a_t). \tag{4}$$

$$V^*(s_t) = \underset{a\in\mathcal{A}}{argmax}\ Q^{\pi}(s_t, a_t). \tag{5}$$

Why do we need Value functions?

Learning a Value function is a fundamental problem in RL

- **Learning**
  - The environment is unknown, we do not have any **prior**
  - We can only deal with it while interacting with it
- **Planning**
  - A model of the environment is given
  - Goal of the agent is to plan within this model

- $V(s_t)$ is used for evaluating states, but does not give any information about which **policy** to follow!
- If V is know with could try all possible actions and select s_{t+1} with the highest V value
- $Q(s_t, a_t)$ we can **immediately select** the action with the highest Q value

Exploration vs Exploitation

- What do we do until we have learned such functions?
- **Exploration Strategies**
  - ϵ greedy exploration

$$a_t = \begin{cases} max_a Q(s_t, a_t) \text{with prob 1-}\epsilon \\ \text{random action with prob } \epsilon \end{cases} \tag{6}$$

  - Boltzmann exploration [1]

$$P(a) = \frac{e^{\frac{Q(a)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}} \tag{7}$$

Both ϵ and τ are parameters that need to be tuned with appropriate exploration schedules!

[1] Wiering, M. A. (1999). Explorations in efficient reinforcement learning (Doctoral dissertation, University of Amsterdam).

How it was done

Before Sutton's work [2] a Value function could be learned via **Monte Carlo** methods

- For each state s_t at the end of a RL episode we compute the **Return** $G_t(s_t)$

$$G_t(s_t) = \sum_{i=0}^{N} \gamma^i r_{t+1} \tag{8}$$

- This denotes **the sum of rewards** in one episode, from the first visited state until the end
- A Value of a single state is simply the average of all the returns that are obtained when visiting that state

$$V(s_t) = \frac{\sum_{i=1}^{k} G_t(s)}{N(s)} \tag{9}$$

[2] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine learning, 3(1), 9-44.

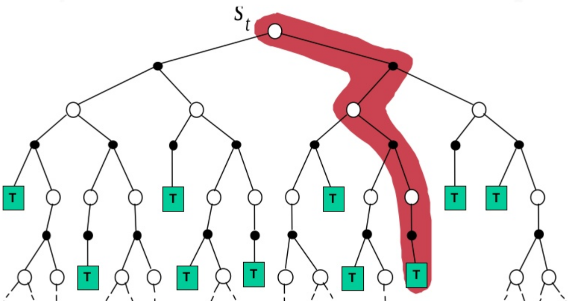
MC-Learning

Monte Carlo based algorithms then learn the value of each state at time step t by updating:

$$V(S_t) \leftarrow V(S_t) + \alpha\left[G_t - V(S_t)\right] \tag{10}$$

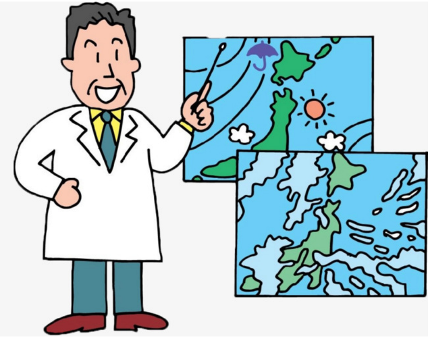
The are two main **drawbacks** of this approach

- The **variance** of the updates is very high
- Very **slow** convergence since we have to wait until a RL episode ends before updating $V(S_t)$



Temporal-Difference Learning

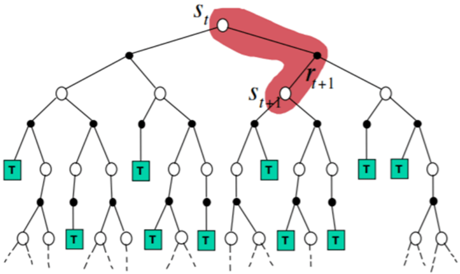
- A combination between Monte-Carlo ideas and Dynamic Programming strategies
- TD-methods can learn directly from **experience** $\approx$ *on the fly*
- Learn wrt what has already been learned (strong recursive component)
- Do not require episodes to end before learning: **Bootstraping**



Learning V with TD-Learning

- TD-Learning only cares about s_{t+1} and its relative reward
- At each time-step we create a **target** to learn from $\Rightarrow$ we do not have access to $G_t(s_t)$, but neither to $V^*(s)$!
- This target is (partially) given by the function that we want to learn!

$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$



Learning V with TD-Learning

- For each step from state s_t to s_{t+1} with reward r_t we do:

$$V(s_t) := V(s_t) + \alpha \left[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\delta_t} - V(s_t) \right] \tag{11}$$

- The TD-error δ_t is the error that is made **at that exact time** wrt the better estimate $r_{t+1} + \gamma V(S_{t+1})$.
- We are learning V^* by guessing it at $V(s_t)$ wrt **another guess** at s_{t+1}

Is TD-Learning sound?

- The idea of learning *on the fly* is certainly appealing, but can we trust it?
- Will we really get better at guessing by guessing?
- **Fortunately Yes**, both TD-methods as MC ones converge asymptotically to the same correct predictions
- **BUT** There is no mathematical proof that shows the superiority of TD-methods over MC ones, even though **experimentally** they do work better!
- TD-methods are also **computationally congenial**

Learning the Q function

- So far we have seen that we can learn a Value function with TD-learning
- We also know that besides the V one, there also is the Q function which plays an important role in RL
- It is straightforward to derive π from $Q(s, a)$

$$\pi^*(s) = \underset{a \in \mathcal{A}}{argmax}\ Q^{\pi}(s, a). \tag{12}$$

Q-Learning

- Probably the most used/known RL algorithm [3]
- The learned state action-value function directly approximates Q^*
- Even if a random π is followed Q-Learning still converges (eventually)
- Is a **greedy** algorithm
- We change a single Q-value given (s_t, a_t, r_t, s_{t+1})
- With respect to what?

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \Big[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \Big]. \tag{13}$$

[3] Watkins, C. J., Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.

Off-policy vs On-policy

- **On-policy**
  - Update and learn π from episodes that have been generated using π itself
  - $\approx$ Learning while *doing the job*
- **Off-policy**
  - Learn π from episodes that are generated by a π which is not the one being followed by the agent
  - $\approx$ Learn by letting *someone else do the job*

Was learning the V function via TD methods online or offline?

$$V(s_t) := V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right] \tag{14}$$

And Q Learning?

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\left[r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]. \tag{15}$$

Q-Learning issues and biases

Despite having convergence guarantees Q-Learning has been shown to suffer from numerous biases

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha\left[r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]. \tag{16}$$

- The $\max_{a_{t+1}\in\mathcal{A}}$ operator makes the algorithm **over-optimistic** and **greedy** when it shouldn't be [4]
- The algorithm has been proven to be **delusional** and to lead to *"bizarre"* policy updates [5]

---

[4] Hasselt, H. V. (2010). Double Q-learning. In Advances in Neural Information Processing Systems (pp. 2613-2621).

[5] Lu, T., Schuurmans, D., Boutilier, C. (2018). Non-delusional Q-learning and value-iteration. In Advances in Neural Information Processing Systems (pp. 9971-9981).

Deep Reinforcement Learning

Scaling Reinforcement Learning Up

We want to use RL techniques to solve **large** problems

- Backgammon: 10^{20} states
- Go: 10^{170} states
- Autonomous driving in continuous action-space

Scaling Reinforcement Learning Up

Learning a Value function was done with **lookup tables**

- Each state s has an entry $V(s)$
- Or each state-action pair s, a has an entry $Q(s, a)$

- Problem when dealing with **large** MDPs
  - There are too many states/actions to store in memory
  - Learning them all is too slow and computationally intensive

- Estimate value functions with a **function approximator**
  - $V_\theta(s) \approx V_\pi(s)$
  - $Q_\theta(s, a) \approx Q_\pi(s, a)$

In principle **any** function approximator can be used, linear vs non-linear ... Neural Networks, Regression Trees [6], ...

[6] Ernst, D., Geurts, P., Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. JMLR, 503-556.

TD-Learning with Neural Networks

- We want to approximate the true $V^{\pi}(s)$ as much as possible, however remember that this function is not available
- We need to construct a **target** for learning this function

This is done exactly as in the tabular case by constructing an approximated target: $r_t + \gamma V(St_{t+1}, \theta)$ which can be incrementally learnt with

$$\triangle\theta_t = (r_t + \gamma V(s_{t+1}, \theta) - V(s_t, \theta))\nabla_\theta V(s_t, \theta) \tag{17}$$

The Rise of Deep Reinforcement Learning

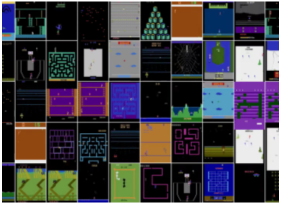
Deep Q-Networks

The idea is to approximate the state-action value function that is usually learnt by:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha \big[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \big]. \tag{18}$$

Similar to how we have approached the V function we can now approximate the Q function that can be learnt as a **regression** problem:

$$L_\theta = E\big[(r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta))^2 \big]. \tag{19}$$



There are however some **problems**...

- Minimizing L_θ after each RL-transition $\langle\, s_t,\, a_t,\, r_t,\, s_{t+1} \,\rangle$ can be very slow
- The network will risk to only focus on RL-trajectories that are **highly correlated** between eachother
- We need to introduce a stochastic element when learning that $\approx$ breaks causality

The solution is called **Experience Replay** [7]

[7] Moore, A. W. (1990). Efficient memory-based learning for robot control.

- Essentially consists of a memory buffer, D, of size N, in which experiences are stored in the form $\langle\ s_t,\ a_t,\ r_t,\ s_{t+1}\ \rangle$
- Once this memory buffer is filled it is possible to uniformly sample batches of experiences $\langle\ s_t,\ a_t,\ r_t,\ s_{t+1}\ \rangle \sim U(D)$ for optimizing the Q-Network

$$L_\theta = E_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)}\left[\Big((y_t) - Q(s_t, a_t, \theta)\Big)^2\right]. \tag{20}$$

Deep Q-Networks and targets

- An Experience Replay buffer is however not enough to guarantee stable learning of the Q-Network
- A *hack* was introduced in [8] and is known as the **Target-Network**

A fair approximation of the Q function would be:

$$L_\theta = E\Big[\big(r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta)\big)^2\Big]. \tag{21}$$

However, even if combined with experience replay DQN is known to not be performing well, leading to the following *trick*

$$L_\theta = E\Big[\big(r_t + \gamma \max_{a_{t+1}\in\mathcal{A}} Q(s_{t+1}, a_{t+1}, {\color{red}\theta^-}) - Q(s_t, a_t, \theta)\big)^2\Big]. \tag{22}$$

[8] Mnih, Volodymyr, et al. Human-level control through deep reinforcement learning. Nature 518.7540 (2015): 529.

- So far we have seen how to learn one single value function with TD-Learning and neural networks
- But how about learning both the V function and the state-action Q function at the same time?
- One function can learn from the other and **accelerate** training.

The idea proposed by **Deep Quality-Value Learning** [9]

[9] Sabatelli, M. et al. Deep Quality-Value (DQV) Learning Advances in Neural Information Processing Systems (NeurIPS), Deep Reinforcement Learning Workshop.

Deep Quality-Value Learning

- Novel DRL algorithm which combines the benefits of TD-Learning on two different levels
- Stability of the algorithm is ensured by the most recent contributions present in the DRL literature
- Main idea is to learn two value functions **simultaneously** that share the same targets to bootstrap

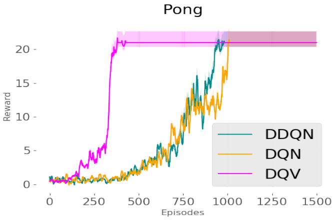
Learning the V function

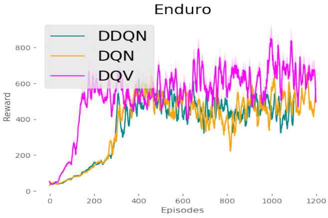
$$L_\Phi = E\Big[\big(r_t + \gamma V(s_{t+1}, \Phi^-) - V(s_t, \Phi)\big)^2\Big] \tag{23}$$

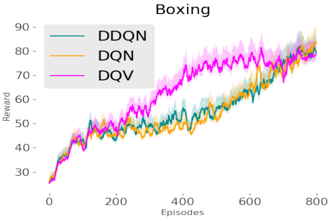
Learning the Q function

$$L_\theta = E\Big[\big(r_t + \gamma V(s_{t+1}, \Phi^-) - Q(s_t, a_t, \theta)\big)^2\Big]. \tag{24}$$

Deep Quality-Value Learning







There are different **pros** and **cons** of this algorithm

- Learning two Value functions is beneficial and yields faster and better learning
- Using $(r_t + \gamma V(s_{t+1}, \Phi^-)$ as a target allows us to get rid off a nasty $\max_{a_{t+1} \in \mathcal{A}}$ operator

However this comes at a cost

- We need to make two Value functions co-exist without making them interfer between eachother (Φ^-)
- We have two parametrized neural networks to keep in memory instead of only one

Learning both the Q and the V function can however been seen as an instance of *Multi-Task Learning* and can be tackled with **hardly-parametrized** neural networks.

Dueling Architectures for DQV

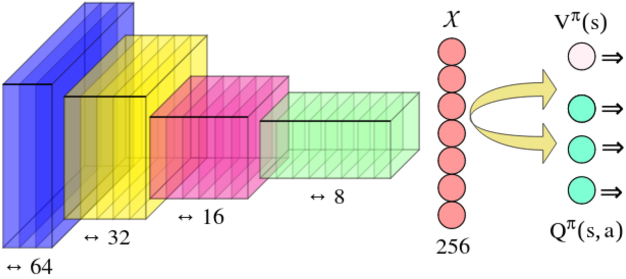


Figure: A value based multi-task RL architecture.

Going off-policy with the DQV idea

$$L_\theta = E_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \Big[\big\{ r_t + \gamma V(s_{t+1}, \Phi) - Q(s_t, a_t, \theta) \big\}^2 \Big], \quad (25)$$

$$L_\Phi = E_{\langle s_t, a_t, r_t, s_{t+1}\rangle \sim U(D)} \Big[\big\{ r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a_{t+1}, \theta) - V(s_t, \Phi) \big\}^2 \Big]. \quad (26)$$

Conclusion

- To recap Deep Reinforcement Learning is a nice combination of 30 years old ideas and Deep Learning advancements
- However the transition from RL $\Rightarrow$ to DRL has opened several research possibilities
  - New biases in algorithms keep being discovered
  - Which in combination with neural networks still make the DRL field "green"
  - Long trainig times + large sets of trajectories + sparse rewards environments + target networks ...

I believe the solution to some of these problems can be found in the original **RL theory** which once understood will give as a higher control over **DRL**.

The End