Evaluating Demonstrations (the Good, the Bad and the Worse)

Gabriele Bani, Davide Belli, Gabriele Cesa, Andrii Skliar

Introduction

Using **single** human demonstration has been shown to outperform humans and beat state of the art models in hard exploration problems [1].
However, it takes an experienced professional to provide good demonstration to the model, which might be impossible in real problems. It might also be difficult to obtain optimal demonstrations. Can we still learn **optimal policies** from **sub-optimal demonstrations**?

Environments

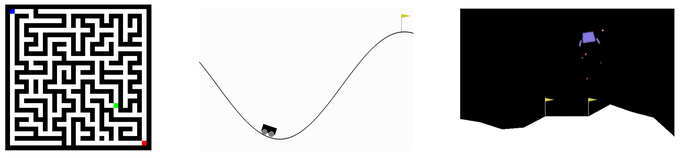

Figure: Used environments: a custom Maze gridworld, MountainCar-v0 and LunarLander-v2 from OpenAI Gym. We use Q-learning for the Maze and DQN with 1 hidden ReLU layer of size 128 for the two continuous environments. **Note**, that DQN takes actual state features as an input and not the raw pixels.

Methods

Basic **idea**: divide the trajectory in n splits. Train on the last one until convergence, then select the previous split. Repeat until the first split, so to learn from increasingly difficult exploration problems.

Algorithm 1 Backward training

Input: One trajectory $\gamma := \{(s_1, a_1, r_1, s'_1), \dots (s_M, a_M, r_M, s'_M)\}$
Input: Number of splits n, Threshold t, Running average size l
Input: Maximum number of iterations per split m
1: $G :=$ total return of γ
2: Divide the trajectory in n equal sized splits $p_1, \dots p_n$
3: **for** $k \in \{n, \dots 1\}$ **do**:
4: $victories :=$ number of victories over the last l runs
5: $i \leftarrow 0$
6: **while** $victories < t$ and $i < m$ **do**
7: Sample an element (s, a, r, s') uniformly from p_k
8: Reset the environment to s
9: Run an episode of Q-learning from l, get total return G'
10: Set current outcome to victory if $G' \geq G$
11: $i \leftarrow i + 1$
12: $victories :=$ number of victories over the last s runs

Parameters n, t, l, m are introduced, the most critical being the number of splits n. A higher n results in a lower the number of states per split. This means that the algorithm needs to explore less before getting in a state for which it knows good Q-values. We observe that in general higher n leads to easier training.

Encouraging Exploration

To **avoid overfitting** to the given demonstration, we encourage additional exploration by linearly decreasing for e episodes the probability ϵ of a non-greedy choice. At every split change we reset ϵ to 1 , and decay it to 0.05 over e runs. The choice of e can impact dramatically the outcome of the learning, and is tightly related to the number of splits n.

Types of Demonstrations

We manually select demos among trajectories generated by partially trained models (MountainCar and LunarLander) or among random paths to the target (Maze). To evaluate the impact on our models from different quality of demonstrations, we use the following types of trajectories:

- An **optimal** demo: gets (close to) best possible reward (e.g. shortest path in Maze, climbing in 87 steps in MountainCar)
- An **average** demo: gets to high reward state but inefficiently (e.g. sub-optimal path in Maze; climbing in 113 steps in MountainCar)
- A **bad** demo: gets to high reward state, but very inefficiently (e.g. very long path in Maze; climbing in 199 out of 200 steps in MountainCar)

Results

In Fig. 1 we show how removing the condition $i < m$ in line (6), the amount of exploration is the same for all demonstrations, and all demonstrations lead to near-optimal reward. (100 samples with $n = 5$, $m = 300$, $e = 0$)

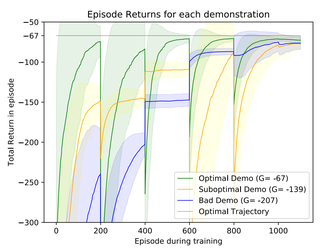

Figure: Reward ($\pm std$) obtained over number of episodes in Maze. The spikes are due to the exploration being high at the beginning of each split. The optimal demo results in both faster convergence and better returns in every split. The bad demo overcomes the suboptimal halfway through the training because it has explored more.

Next, we explore whether using the condition in line (6) allows efficient use of the demonstration in terms of number of episodes in fig. 2

Maze:

- optimal demo (notice small *std* in the end) faster than train from scratch.
- A bad demo leads to better rewards than the suboptimal demo (due to larger exploration over the maze when following the first one)

MountainCar:

- the bad demo is easily beaten, converging early to a suboptimal policy
- average demo sometimes diverges; optimal demo almost always leads to optimal rewards, working better than training from scratch.

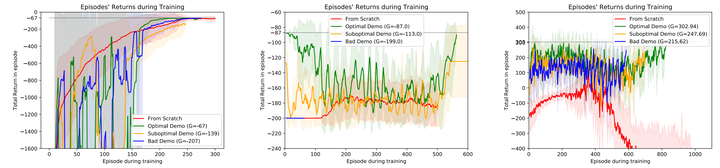

Figure: Returns over episodes in **Maze** (left; $e = 10, n = 5, l = 15, t = 3$, statistics over 29 samples), **MountainCar** (middle; $e = 20, n = 15, l = 10, t = 4$, statistics over 12 samples) and **LunarLander** (right; $e = 20, n = 30, l = 10, t = 5$, statistics over 3 samples). Each line corresponds to a different training strategy (from scratch or with a different demonstration). The high *std* (shaded areas around the curves) in Maze is due to different runs changing splits at different times. In LunarLander, training from scratch is very difficult, while demonstrations reliably lead to good results.

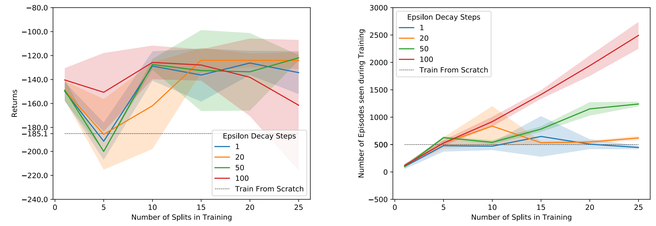

Figure: (average) return of the last 10 episodes (**left**) and number of episodes seen (**right**) in the backward training for different n and e parameters in MountainCar to check how they influence the learning outcomes (statistics from at least 6 independent runs for each point in the plot, with a 1-std on both sides). Though, generally, the choice of these 2 hyper-parameters has smaller influence on the final returns, it seems that a large number of splits together with limited exploration lead to slightly better returns and faster training. Surprisingly, using only 1 split leads to very quick training with little loss in performances.

Conclusion

- Non optimal demonstrations can lead to optimal results, but better demonstrations lead to better learning and give more reliable results.
- In Maze, using bad demonstrations rather than suboptimal ones results in a better final policy because of a higher degree of exploration.
- With more complex environments, we expect demonstrations to allow for a much faster training than training from scratch.
- The current implementation is very sensitive to hyperparameter choices; there is a need for a more automatic and reliable version of the backward algorithm to overcome this issue.

References

[1] Tim Salimans and Richard Chen.
Learning Montezuma's Revenge from a Single Demonstration.
arXiv e-prints, page arXiv:1812.03381, December 2018.