Deep Reinforcement Learning Methods for Recommender Systems

Advanced Deep Learning course project

Brieuc DE VOGHEL
brieucjeanyves.devoghel@mail.polimi.it

Abstract

Recommender systems are omni-present in our everyday lives. Since the recommendation problem has been shown to be of sequential nature and since it can be formulated as a Markov decision process, reinforcement learning (RL) methods can be employed to solve it. In our work, we compare two different RL methods and approaches to the recommendation problem. We first formulate how the recommendation problem fits in a RL framework. Then, we present the methods and their specific RL algorithms. The first method [3] introduces Deep Deterministic Policy Gradient (DDPG), an actor-critic model-free algorithm able to learn policies in high-dimensional, continuous action spaces. The second method [1] introduces Twin Delayed DDPG (TD3), building upon DDPG and addressing the problems of bias overestimation and high variance build-up. Finally, we apply the methods on the recommendation problem of online retail items suggestion and compare the advantages and drawbacks of the different approaches; finding that TD3 addresses and corrects the main limitations attributed to the first algorithm.

1. Introduction

Recommender systems (RS) assist service providers by generating the best suggestions for their users, fitting their needs and preferences. RS are key to most successful services in a variety of domains including search engines, streaming services, news outlets, social media, etc. Traditionally, the recommendation procedure was considered as a static process and recommendations were made following a fixed greedy strategy. These approaches may fail given the dynamic nature of the users' preferences, and the recommendable items catalog.

In this work, we consider the recommendation procedure as sequential interactions between users and a recommender agent [2, 5]. We present how the recommendation problem fits in a reinforcement learning framework and present two methods addressing this challenge. Applying deep reinforcement learning (DRL) to this problem allows to train an agent on an environment leveraging models of the complex user behavior and thus not being limited by its dynamic nature. Compared to other methods such as POMDP [5] and Q-learning [8], DRL offers much more flexibility and the possibility to work with continuous, high-dimensional observation and action spaces, which is critical in the recommendation system setting. This becomes critical as the number of items to suggest becomes unlimited and as we face the "cold start problem" : in dynamic settings, when a new item is to be recommended but it has no previous interaction events with any user, the previous mentioned methods become completely unpractical.

The model-free deep reinforcement learning methods we are presenting and applying to the recommender problem do not estimate transition probabilities and do not need to store the Q-values table. The approximator functions allowing to estimate these are parameterized by neural networks and are flexible to support huge state and action spaces.

2. Related work

In this work we present an application of two reinforcement learning algorithms able to learn policies in continuous and high-dimensional action-spaces.

Deep Deterministic Policy Gradient (DDPG) is the first approach we are considering [3]. Its actor-critic framework is inspired by Deep Q Network (DQN) architectures [4] and has been combined with innovation coming from the Deterministic Policy Gradient (DPG) algorithm [7]. It was developed to robustly solve challenging problems across a variety of domains with continuous action spaces. Its fact of being deterministic, which is defined by direct action prediction (as opposed to probabilistic, which outputs a probability density function over the possible actions), makes this method applicable to continuous, high dimensional, action spaces. A replay buffer allows for experience replay and to be very sample efficient for the learning process. The use of separate target networks oppose the Q update, which is prone to divergence, to only change slowly thanks to the limited soft target update; greatly improving the stability of

learning. Its few limitations leading to suffering from an overestimation bias and error build-up were later addressed by our second suggested method.

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an improvement on DDPG, suggesting to solve some of it's predecessor's limitations [1]. First, it takes inspiration from Double Q-learning [9], suggesting to leverage that clipping the value estimate with two networks can put an upper-bound to the initially over-estimation problem faced by DDPG. This is to ensure safer policy updates with more stable learning targets. This has been tested empirically and shows some great potential. The method also introduces delayed policy updates to make the learning process even more stable. It argues that less frequent policy updates will use a value estimate with less variance, as it will have continued to converge further; resulting in higher quality policy updates. Lastly, building on the notion that similar actions should have similar values, it indicates that modifying the training procedure to regularize training in smoothing the value estimate helps converging more easily and making it less susceptible to inaccuracies.

Hence, TD3 maintains a pair of critics along a single actor. For each time step, the critics are updated towards the minimum of the two critics target networks. A small random Gaussian noise is added for some exploration. Every d (typically two) iterations, the policy is updated with respect to the predicted Q value following the deterministic policy gradient.

3. Proposed approach

In this section, we define the notations and the problem of recommender system via reinforcement learning. We then explain how an actor-critic based agent can be used in this setting. We discuss how to train the framework and how to use it for recommendations.

3.1. Problem statement

We study the recommendation task in which a recommender agent (RA) interacts with environment $\mathcal{E}$ by choosing which items to recommend over a sequence of steps, so as to maximize its reward. We consider a standard RL formulation based on a Markov Decision Process (MDP), which includes a sequence of states, actions, and rewards. More specifically, it consists of a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ as follows :

- Observation space $\mathcal{S}$: an observed state $s_t = \{s_t^1, ..., s_t^N\} \in \mathcal{S}$ is defined as an embedded feature vector representing the user at time t. The embedding and original feature vector encode the user's profile, preferences, browsing history, etc. as defined by [6]. N is the size of the embedding vector.
- Action space $\mathcal{A}$: an action $a_t = \{a_t^1, ..., a_t^K\} \in \mathcal{A}$ is defined as the feature vector describing the recommendation made to a user at time t based on current state s_t. K is the number of features by which items are described in the environment $\mathcal{E}$.
- Transition probability $\mathcal{P}$: transition probability $p(s_{t+1}|s_t, a_t)$ defines the probability of state transition from state s_t to s_{t+1} when RA takes action a_t. The user behavior (click/view, skip, leave - more on this at the next paragraph) defines how the next state s_{t+1} updates, as defined in [6].
- Reward $\mathcal{R}$: after RA takes an action a_t at the observed state s_t, the user browses the recommended items and provides feedback based on its behavior (click/view, skip. leave - more on this at the next paragraph). RA receives the immediate reward $r(s_t, a_t)$ according to this feedback.
- Discount factor γ : $\gamma \in [0, 1]$ defines the discounted weight factor attributed to the future reward when we measure it's present value. In particular, when $\gamma = 0$, RA only considers the immediate rewards, whereas when $\gamma = 1$, all future rewards count as much as that of the current action.

User behavior when recommendation is made The environment calculates the dot product of the action feature vector a_t and the item attributes e_i (feature vector of size K) for each existing item in the environment, yielding a appreciation score per item :

$$score_i = a_t \cdot e_i^T \tag{1}$$

The top-k scoring items are then suggested to the user. The user, through the user model (discussed in secion 3.2), can decide to click/view an item, skip the made recommendations, or leave. When it skips or leaves, the reward is 0. Otherwise, the rewards corresponds to the number of clicks/views. If the user leaves, the episode finishes; if the user skips, a new action (i.e. recommendation) is asked from the RA.

The feature vector describing the actions (and items) are the attributes used to represent the items. For instance this can be information like brand, price, item category, etc. This has the other advantage of being able to work within a very dynamic system in terms of possible item suggestion. The recommender agent can even be item-agnostic as long as know what type of feature vector is expected. The "cold start problem" of having items to be recommended, but having no prior interaction data is no longer relevant since having intrinsic features describing the item suffices.

Figure 1 illustrates the agent-environment interactions in MDP, with the user being part of the environment, together

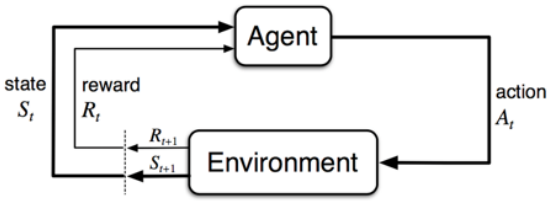

Figure 1. The agent-environment interactions in MDP.

with the existing items (abstracted away, as per [6]). The recommender agent takes actions (makes recommendation) to users by interacting with the environment (user) in a way to maximize the expected discounted return.

With the notations above, the problem can be formally defined as follows : *Given the historical MDP, i.e.* $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$*, the goal is to find a recommendation policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ *which can maximize the cumulative reward for the recommender system.* [**?**]

3.2. Online environment simulator

To tackle the training of an agent and allowing for its evaluation, we use an online environment simulator as defined in [6]. The task of the simulator is to predict a reward based on the current state and the selected action, i.e. $f : (s_t, a_t) \rightarrow r_t$. The simulator also updates the state (representation of the user) according to a model, mimicking the user behavior, i.e. $g : (s_t, a_t) \rightarrow s_{t+1}$. This model in an intrinsic part of the simulator. We can expect users with similar interests (represented by similar states s_i) to make similar decisions when given the same recommendation, i.e. action a_i. Making similar decisions imply predicting similar rewards and updating the state in a similar fashion.

The online aspect represents the need for an agent to interact with the environment in order to be able to know what the reward and update to the state will be.

3.3. Actor-critic framework

The agent uses an actor-critic framework in order to learn from, and act on the environment. The actor predicts the action to take given a particular state :

$$f_{\theta^\pi} : s_t \rightarrow a_t \tag{2}$$

where f_{θ^π} is a function parameterized by θ^π, mapping from the state space to the action space, following policy π.

The critic estimates the value of taking a particular action given a certain state. This value function $Q(s_t, a_t)$ is a measure of how good the action a_t matches the current state s_t. In real recommender systems, many state-action pairs may never appear in real traces or interaction sequences, such that it is hard to update the Q function appropriately. Therefore, it is more flexible to use an approximator function to

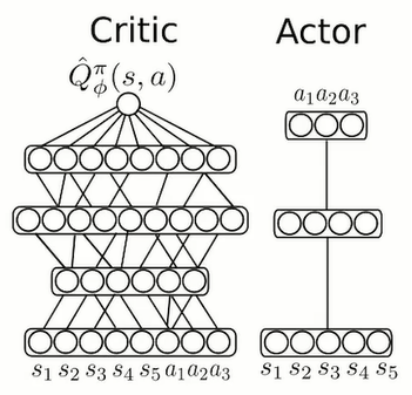

Figure 2. Representation of the actor and critic networks of the framework.

estimate the state-action value, i.e. $Q(s, a) \approx Q(s, a, \phi)$, we say that the function Q is parameterized by ϕ. In practice, both functions are implemented using deep neural networks, known to be excellent approximators for non-linear functions.

In our case, we use the DDPG [3] and TD3 [1] algorithms to train and evaluate an actor and a critic. Training the networks consists thus in maximizing the expected rewards and following the Bellman equation to update the value function leading to the optimal policy [3] :

$$Q(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})] \tag{3}$$

When the agent is trained, the evaluation is done using the actor only. Based on a given state, it predicts the action to perform, i.e. the recommendation to make.

4. Experiments

In this section we discuss the experiments conducted with the Virtual-Taobao simulation environment [6] to evaluate how the two analyzed algorithms (DDPG and TD3) compare. We focus on evaluating the stability and the overestimation bias of the two algorithms, as this is mainly what TD3 is addressing about DDPG, making it both faster learning and better performing.

Dataset. The used environment is based on interaction events of customers with items suggested on the Taobao retail platform. Using historical data, [6] modeled with supervised learning the user behavior in order to obtain a reliable model, able to reflect user behavior in function of suggested items faithfully. As understanding how the model was trained is beyond the scope of this work, we won't go in further detail. Using this user model is sufficient to obtain an environment able to process actions, update it's observed state and predict a reward, as discussed in section 3.2.

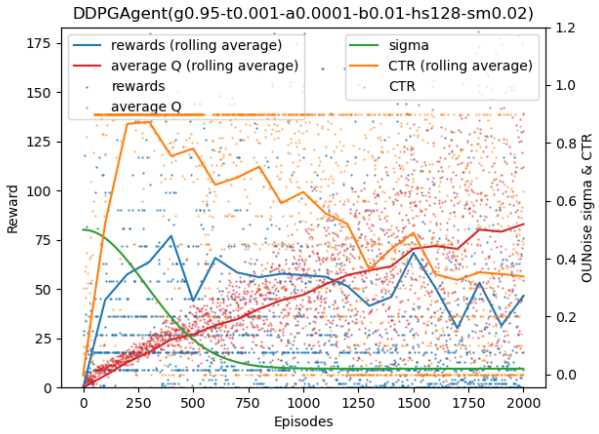

Figure 3. Experimental results of the DDPG agent. We observe a learning agent, but suffering from catastrophic forgetting (see orange CTR). Over-estimation of the Q value also builds up over time and variance increases (see red episode-average Q value estimation).

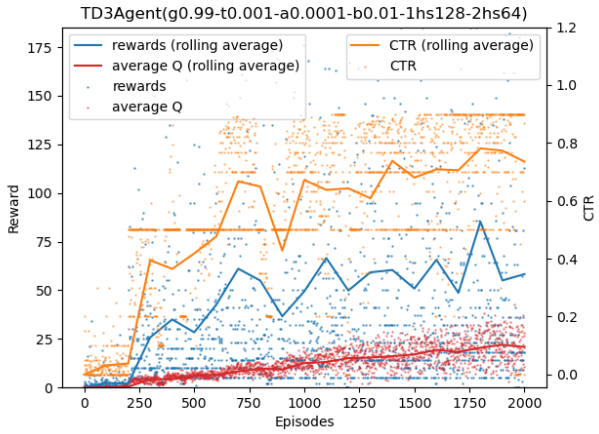

Figure 4. Experimental results of the TD3 agent. Estimated Q values are kept reasonable (red) and agent learns to improve CTR which becomes more stable (orange).

Experiments setup. We set both DDPG and TD3 to use actor and critic networks to be dense neural networks with 2 hidden layers of 128 and 64 hidden neurons. DDPG suggests an Ornstein–Uhlenbeck process noise for the action (starting with $\sigma = 0.5$ and a decay over half the total training time); whereas TD3 sticks to a Normal Guassian noise of mean 0 and variance 0.1. All other hyper-parameters are kept the same between DDPG and TD3: 0.0001 and 0.01 learning rate for the actor and critic respectively; discount factor $\gamma = 0.95$; $\tau = 0.001$. Replay buffer size is of 1 million with a sample size of 400 per learning step. The models were given 2000 episodes to train on. The models are implemented in PyTorch and are light enough to be trained on a laptop.

The environment suggests a top-10 list of items per page to the user and no limitation is set on the number of pages a user can go through.

Evaluation metric As the users can browse as many pages as they would like, observing the rewards per episode is, on itself, not informative. Conveniently, the click-though-rate (CTR) is a very informative and well-used metric in the recommender settings:

$$CTR_{episode} = \frac{view\_count_{episode}}{max\_page_{episode} \cdot page\_length} \quad (4)$$

It is used to gauge how well a recommendation is performing, with 1 being the maximum (every suggestion is viewed by the user) and 0 the worst (no suggestions are selected).

Results and discussion. From figure 3 and 4, we observe that both algorithms learn to recommend better items to the user and manage to improve the CTR. DDPG suffers from catastrophic forgetting as Q value estimation errors build up over time and makes the CTR drop after a peak. Over several runs, DDPG also tend to get stuck in local optima, due to its limitations discussed previously. We clearly see its major limitation which is high variance build-up and over-estimation of the Q value compared to the actual expected returns (red against blue observations). Despite its catastrophic forgetting, DDPG is still able to obtain impressive results quite early in its training phase.

On the other hand, we can confirm that TD3 indeed addresses the issues : its estimated Q value does not suffer from over-estimation, although there is still some variance left, albeit being lower than DDPG. The training also converges to a higher performance than DDPG by quite a margin, despite being a bit slower to train and obtain conclusive performances.

The 2000 episodes are not enough to observe longer term trends, but nothing surprising happens when running with up to 5000 episodes : DDPG's variance and over-estimation continues to build up, but the catastrophic forgetting stabilizes around the same value obtained after 2000 episodes. TD3's performances (CTR) stay stable and do not improve much more; estimated Q value's variance continue to build up but at a lower rate, and no over-estimation is observed.

Unfortunately, the used environment encapsulates the top-k item selection and uses embedding for the user representation. This limits our possible evaluation of the predicted recommendation to the only metrics we get which are rewards per recommendation action.

5. Conclusion

In this work, we apply two deep reinforcement learning algorithms applicable to continuous high-dimensional state and action spaces to the recommender problem.

We discuss how reinforcement learning fits in the recommendation system settings. Using deterministic algorithms allow to work in feature spaces and thus not suffering from limitations in e.g. the number of items that can be recommended. We have also seen that the recommender agent can also be completely item-agnostic in terms of what it is recommending.

Using an environment simulator for online retail recommendation, we apply the two algorithms and analyze their advantages and shortcomings. We observe the shortcomings in DDPG addressed by TD3 and validate its superiority in stability and in not over-estimating action values.

There are several interesting improvement directions. Session-based recommendations leveraging previous made recommendations to the same user are not addressed by the considered frameworks and could in certain environments be of great benefit. Also, different replay buffer management strategies could be considered as it is shown to be problem specific and has not been addressed in this work.

References

[1] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018. 1, 2, 3

[2] B. Hu, C. Shi, and J. Liu. Playlist Recommendation Based on Reinforcement Learning. In Z. Shi, B. Goertzel, and J. Feng, editors, *2nd International Conference on Intelligence Science (ICIS)*, volume AICT-510 of *Intelligence Science I*, pages 172–182, Shanghai, China, Oct. 2017. Springer International Publishing. Part 3: Big Data Analysis and Machine Learning. 1

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2015. 1, 3

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015. 1

[5] G. Shani, D. Heckerman, and R. I. Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(43):1265–1295, 2005. 1

[6] J. Shi, Y. Yu, Q. Da, S. Chen, and A. Zeng. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. *CoRR*, abs/1805.10000, 2018. 2, 3

[7] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China, 22–24 Jun 2014. PMLR. 1

[8] N. Taghipour and A. Kardan. A hybrid web recommender system based on q-learning. pages 1164–1168, 01 2008. 1

[9] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015. 2