Reinforcement Learning Assignment 3

Yifan Bai(260562421), Bhavya Patwa(260964036)

April 2020

1 TQ 1 - Off Policy RL

Acknowledgement

Insipirations taken from this paper: https://www.cs.mcgill.ca/ dprecup/publications/PSD-01.pdf

1.1 Question 1

Looking at the right hand side of the equation, we have:

$$\Delta\theta_t = \alpha(R_t^\lambda - \theta^T\phi_t)\phi_t$$

Where α is learning rate, R_t^λ is importance sampling corrected lambda returns,

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)}$$

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + ... + \gamma^n\theta^T\phi_{t+n}$$

LHS of Eq2 can be expressed as:

$$\Delta\tilde{\theta} = \alpha(\tilde{R_t^\lambda} - \theta^T\phi_t)\phi_t\rho_1...\rho_t$$

From Importance Sampling:

$$E[\tilde{R_t^\lambda}|s_0, a_0] = E[R_t^\lambda|s_0, a_0]$$

Where $\tilde{R_t^\lambda}$ is the returns following policy μ and R_t^λ and the returns following policy π. Combine the above,

$$E_\mu[\Delta\tilde{\theta}] = E_\mu[\sum_{t=0}^{\infty}\alpha(\tilde{R_t^\lambda} - \theta^T\phi_t)\phi_t\rho_1...\rho_t]$$

$$= E_\mu[\sum_{t=0}^{\infty}\alpha[\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1}\tilde{R_t^n} - \theta_T\phi_t]\phi_t\rho_1...\rho_n]$$

Sorry for the messy equations...

$$= \sum_{t=0}^{\infty}\prod_{j=1}tp_{S_j-1,S_j}^{a_j-1}b(s_j,a_j)\phi_t\prod_{k=1}^{\infty}\frac{\pi s_k, a_k}{bs_k, a_k} \times (E_b[\tilde{R_t^n}|s_t, a_t] - \theta^T\phi_t)$$

Where $p_{x,y}^a$ is the probability that action A will transfer state x to y in the next time step. Given the Markovian property that $\tilde{R_t^{(n)}}$ does not reply on $a_0, a_1, ..., a_{t-1}$, we can then write that,

$$E_\mu[\tilde{\theta}] = \sum_{t=0}^{\infty}\prod_{j=1}tp_{S_j-1,S_j}^{a_j-1}b(s_j,a_j)\pi(s_j,a_j)\phi_t((E_b[\tilde{R_t^n}|s_t,a_t] - \theta^T\phi_t)) = E_\pi[\sum_{t=0}^{\infty}(R_t^n - \theta^T\phi_t)\theta_t]$$

Hence the equivalency

1.2 Question 2

Citing from the paper [1]:

Let us introduce a non-negative random variable, g_t, that depends ONLY on events up to and including timestep t. It represents the extent of which an episode is considered to stat at time t. The function $g : \Omega_t \mapsto R^+$ provides expected value of g_t for any trajectory up through t. We can introduce the update rule as:

$$\Delta\theta_t = \alpha(\tilde{R_t^n} - \theta^T\phi_t)\phi_t \sum_{k=0}^{t} g_k\rho_k + 1...\rho t$$

If we let $g_0 = 1$ and $g_t = 0, t \geq 0$, then we have the same update rule as the original importance sampled $TD(\lambda)$

Using what was proved from the last question, as well as what was discussed in the paper, we can adapt using eligibility trace as follows:

- For each episode:
  - Initialize trace $\vec{e_0} = c_0\phi_0 = \phi_0$, $c_0 = 1$
  - For every transition (s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})
    - Let importance sampling ratio $\rho_{t+1} = \frac{\pi(s_{t+1},a_{t+1})}{\mu(s_{t+1},a_{t+1})}$
    - $\delta_t = r_{t+1} + \gamma\rho_{t+1}\theta^T\phi_{t+1} - \theta^T\phi_t$
    - $\Delta\theta_t = \alpha\delta_t\vec{e_t}$
    - $c_{t+1} = g_{t+1} + c_t \times \rho_{t+1} = c_t \times \rho_{t+1}$
    - $e^{\vec{t+1}} = \gamma\lambda\rho_{t+1}e_t + c_{t+1}\phi_{t+1}$
- At end of an episode:
  - $\theta \leftarrow \theta + \sum_t \Delta\theta_t$

Repeat this algorithm episodically.

2 TQ 2 - Paper Reading - Track A

2.1 Question 1

From the paper,

"**To perform experience replay we store the agent's experiences** $e_t = (s_t, a_t, r_t, s_{t+1})$" [2]

This means instead of running Q-learning on state-action pairs as they occur during simulation or actual experience, the system stores the data discovered for $[state, action, reward, nextstate]$, typically in a large table. Note this does not store associated values - this is the raw data to be fed into action-value calculations later.

The learning phase is then logically separate from gaining experience, and based on taking random samples from this table. Ideally we would still want to separate the processes - acting and learning - because improving the policy will lead to different behaviour that should explore actions closer to optimal ones, which is better to learn from. However, we can split this in many different ways, for instance, take one step, learn from three random prior steps etc. The Q-Learning targets when using experience replay use the same targets as the online version, so it retains the same equation. The loss equation is also same as the one for DQN without experience replay. The only difference is only which s, a, r, s', a' being fed into it.

In DQN, the DeepMind team also maintained two networks and switched which one was learning and which one feeding in current action-value estimates as "bootstraps". This helped with stability of the algorithm when using a non-linear function approximator. That's what the bar stands for in $\bar{\text{i}}$ - it denotes the alternate frozen version of the weights.

Advantages of experience replay:

- More efficient use of previous experience, by learning with it multiple times. This is key when gaining real-world experience is costly, we can get full use of it. The Q-learning updates are incremental and do not converge quickly, so multiple passes with the same data is beneficial, especially when there is low variance in immediate outcomes (reward, next state) given the same state, action pair.

- Better convergence behaviour when training a function approximator. Partly this is because the data is more like i.i.d. data assumed in most supervised learning convergence proofs. When learning on-policy, the current parameters determine the next data sample that the parameters are trained on. For example, if the maximizing action isto move left then the training samples will be dominated by samples from the left-hand side. This might cause unwanted feed-back loops which makes parameters to either stuck in a local minimum or diverge badly. ER helps average over many previous states which then smoothes out learning and avoid these instability or divergence issues.

Consider an example. Say during the training we are in one state and we take an action according to ϵ-greedy policy and end up in another state, and get rewards and the next state. Here the reward can be the score of the game and the states can be the pixel patterns in the screen. And then we take the error between our function aproximator and the value we got from the greedy policy again using already frozen function approximator. But with the experience replay when optimizing the approximator we take some random state action data set. It is the exact same data from the real trajectory, but instead of learning only from the most recent step, you save it in a big table and sample from that table (usually multiple samples, with a store of thousands of previous steps to choose from).

Another possible benefit is that it could also improve off-policy learning. In Q-learning with ϵ-greedy update value functions according to greedy policy. Every timestep, our NN parameters get updated by mini-batch, which is not the exact timestep statistics but what happened before. This would help uncorrelating data.

2.2 Question 2

Prioritized Experience Replay (PER) centres around its idea on the fact that some experiences may be more important than others for our training, but might occur less frequently. In another word, there is no guarantee in which experiences being recorded or occurrd have similar level of frequency versus the importance (in the hindsight). Because we sample the batch uniformly (selecting the experiences randomly) these rich experiences that occur rarely have practically no chance to be selected, or it could be very 'under-represented'.

In fact there are two choices to be made: which experiences to store, and which experiences to replay and how to do it. The paper only address the second one.

With PER, we try to change the sampling distribution by using a criterion to define the priority of each tuple of experience. We want to take in priority experience where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it. We use the absolute value of the magnitude of our TD error:

$$p_t = |\delta_t| + e$$

Where δ_t is the magnitude of our TD error and e is a constant that guarantees no experience has 0 probability to be taken. This is particularly suitable for incremental, online RL algorithms, such Q-learning, that already compute the TD-error and update the parameters in proportion to δ. The TD-error can be a poor estimate in some circumstances as well, e.g. when rewards are noisy. This would also lead to always training the same experiences (that have big priority), and thus over-fitting. To cope with that, we introduce a new scheme: Stochastic Prioirty Sampling. Please refer to the next section for more details, when we compare the advantages and disadvantages of PER.

2.3 Question 3

As we mentioned in the last part, one obvious advantage of PER is that it allows us to sample in priority the experience where there is a big difference between our prediction and the TD target, since it means that we have a lot to learn about it. One strategy is what we discussed as 'Greedy Prioritization'. The central component is the criterion by which the importance of each transition is measured. One idealized would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable approximation is magnitude of TD error, indicating how 'surprisng' the transition is, i.e. how far the value is from the next-step bootstrap estimation. However, in spite of advantages it offers compared to ordinary ER, which employs uniform sampling, there are a few disadvantages:

- Only replayed elements are getting updated. This means elements with small transition on the first run may never be visited.
- Sensitive to noise spikes and prone to overfitting for slowly shrinking error. High initial transitions are replayed more frequently.

However, the thing is that we still have room to play and address the isses. Then comes 'Stochastic Prioritization'. In essence, it uses a 'softmax' function:

$$P(i) = \frac{p_i^a}{\sum_k p_k^a}$$

Note that if $a = 0$, we recover the uniform sampling;$a = 1$, we select only those with highest priorities.

As a consequence, during each time step, we will get a batch of samples with this probability distribution and train our network on it. However there still exists a problem: With normal Experience Replay, we use a stochastic update rule. As a consequence, the way we sample the experiences must match the underlying distribution they came from.

When we do have normal experience, we select our experiences in a normal distribution — simply put, we select our experiences randomly. There is no bias, because each experience has the same chance to be taken, so we can update our weights normally. With priority sampling, we introduce bias toward high-priority samples. If we update weights normally, there is a risk of overfitting. Samples that have high priority are likely to be used for training many times in comparison with low priority ones. Consequently, we would just update weights with small portion of experiences 'deemed' to be interesting. To cope with that, we try to 'anneal' the bias using Importance Sampling weights that adjusts the updates by reducing weights of frequently seen samples.

$$w_i = (\frac{1}{N}\frac{1}{P(i)})^{\beta}$$

The weights of high-priority samples have less adjustment (because the network will see these experiences many times), whereas those for low-priority will have a full update. β controls how much these importance sampling weights affect learning. In practice, the β parameter is annealed up to 1 over the duration of training, because these weights are more important in the end of learning when our Q values begin to converge. The unbiased nature of updates is most important near convergence [3].

3 Statement of Contribution

Both team members contributed equally to the written part of this assignment, from brainstorming ideas, formulating solutions, to reviewing solutions and typing out the document. We hereby state that all the work presented in this report is that of the authors. The link to Google Colab notebooks are:

Question 1
`https://colab.research.google.com/drive/1XuvgcUjTf7kzRavsnJM4WKadXjenlEO9`
Question 2
`https://colab.research.google.com/drive/1qXu1aWxGQu2jlsxrrZnp1ToOtJKAfLTI`

4 References

1 Precup, D., Sutton, R. S., Dasgupta, S. (2001). Off-Policy Temporal-Difference Learning with Function Approximation. Retrieved March 30, 2020, from https://www.cs.mcgill.ca/ dprecup/publications/PSD-01.pdf

2 Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

3 Emami, P. (2016, January 26). Prioritized Experience Replay. Retrieved March 30, 2020, from https://pemami4911.gith summaries/deep-rl/2016/01/26/prioritizing-experience-replay.html