DM887 Assignment2 Q1

Jiawei Zhao

19.03.2024

Implementation of Least-Squares Temporal Differences (LSTD) Deep Q-Learning with Nonlinear Feature Extraction that maps a state to lower-dimensional latent embedding. While the action-value function should be a linear function of the output of the feature extractor.

Algorithm 1 Initialization

1: Initialize a variational autoencoder $\mathbf{A}$ as the feature extraction network with initial weights w_0
2: Initialize the learning rate of $\mathbf{A}$ as $\alpha = 1 \times 10^{-3}$
3: Initialize an Adam optimizer o with a learning rate α for $\mathbf{A}$ [w_0 are drawn from Glorot uniform initialization]
4: Initialize the minibatch size of DQN as $k = 32$
5: Initialize the total number of training episodes $N = 1050$, and the number of episodes at each seperate phase of training $N_0 = 50$ [It would be a wise choice to seperate the warm-up phase, the antoencoder update phase, and the LSTD update phase]
6: [Initialization of training hyperparameters]
7: Initialize the number of maximum time step per episode $T = 500$
8: Initialize the weights θ_0 for linear approximation function $y = f(phi(s))$ which will be used for LSTD [w_0 are drawn from a zero-mean Gaussian distribution with $\sigma = 1$ and a fixed seed 42]
9: Initialize a replay memory buffer $\mathcal{D}$ with a capacity $N \times T$
10: Initialize a discount factor $\gamma = 0.9$
11: Initialize a small constant $\lambda = 1 \times 10^{-3}$ to initialize A^{-1} for LSTD
12: Given the number of actions as N_a and embedding dimension N_e, initialize a matrix θ with shape $N_a \times N_e$
13: Initialize $A^{-1} = \lambda^{-1}I$ and $b = 0$ to store the tensors that suffice $\theta_t = A_t^{-1}b_t$ at each time step t [A relatively large discount factor encourages long-term planning and faster convergence during training]

[1]

Algorithm 2 Warm-up phase

1: Freeze the weights of $f(phi(s))$, i.e. θ
2: Freeze the weights of $\mathbf{A}$, i.e. w
3: **for** $episode\ e_0 = 1$ to N_0 **do**
4:     Initialize state s
5:     Preprocess s into s_0 to adapt it as input of $\mathbf{A}$ [preprocessing of high-dimensional states is necessary w.r.t. autoenconders]
6:     **for** each time step $t = 1$ to T **do**
7:         Encode state s_t using $\mathbf{A}$ to get latent embedding $\phi(s_t)$
8:         Select action a_t using ϵ-greedy policy with $\epsilon = 0.3$ [The learning curves of Game B of Assignment 1 corroborate that a relatively high ϵ could improve Q more efficiently when an e does not end prematurely in the majority of cases]
9:         Execute a_t and obtain reward r_t and new state s_{t+1}
10:        **if** $s_{t+1} \notin S$ **then**
11:            **break**
12:        **end if**
13:        Store the transition of the current t, i.e. (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
14:        Encode state s_t using $\mathbf{A}$ to get latent embedding $\phi(s_t)$
15:        Select action a_t using ϵ-greedy policy with $\epsilon = 0.3$ [The learning curves of Game B of Assignment 1 corroborate that a relatively high ϵ could improve Q more efficiently when an e does not end prematurely in the majority of cases]
16:        Execute a_t and obtain reward r_t and new state s_{t+1}
17:        Store the transition of the current t, i.e. (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
18:     **end for**
19: **end for**
20: $e = e + N_0$
[1]

Algorithm 3 Autoencoder update phase

1: Freeze the weights of $f(phi(s))$, i.e. θ
2: Unfreeze the weights of **A**, i.e. w
3: Reset A^{-1} and b to the default value at the initialization phase
4: **for** *episode* $e_0 = 1$ to N_0 **do**
5:      Repeat the same steps at the warm-up phase
6:      **for** each time step $t = 1$ to T **do**
7:          Repeat the same steps at the warm-up phase
8: [Start updating the autoencoder using minibatches]
9:          Sample a minibatch of transitions d from $\mathcal{D}$ with a batchsize k
10:          Use the s_t of d as input to **A**
11:          First encode, then decode d using **A**
12:          Calculate the loss L of s_t by comparing the input and output of **A**
13:          Update w with o as per L
14:      **end for**
15: **end for**
16: $e = e + N_0$

[1]

Algorithm 4 LSTD update phase
___
1: Freeze the weights of $f(phi(s))$, i.e. θ
2: Unfreeze the weights of $\mathbf{A}$, i.e. w
3: **for** *episode* $e_0 = 1$ to N_0 **do**
4:      Repeat the same steps at the warm-up phase
5:      **for** each time step $t = 1$ to T **do**
6:          Repeat the same steps at the warm-up phase
7: [Start using the online LSTD algorithm to update θ]
8:          Calculate $\tau = \phi(s_t) - \gamma\phi(s_{t+1})$
9:          Calculate $v = \tau^T A^{-1}$
10:          Update $A^{-1} = A^{-1} - \frac{A^{-1}\phi(s)v^T}{1+v\phi(s)}$
11:          Update $b = b + r\phi(s)$
12:          Given the action a of the current time step, update $\theta_a = A^{-1}b$
13:          Update state $s_t = s_{t+1}$
14:      **end for**
15: **end for**
16: $e = e + N_0$

[1]
___

Algorithm 5 Training procedure

1: Run *Initialization*
2: Run *Warm-up phase*
3: **while** *episode $e \leq N$* **do**
4:      Run *Autoencoder update phase*
5:      **if** $e + N_0 \geq N$ **then**
6:         Reset A^{-1} and b to the default value at the initialization phase
7:      **end if**
8:      Run *LSTD update phase*
9: **end while**
---

[1]