

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 [w_0 are drawn from Glorot uniform initialization]
- 2: Initialize the learning rate of \mathbf{A} as $\alpha = 1 \times 10^{-2}$
- 3: Initialize an Adam optimizer o with a learning rate α for \mathbf{A}
- 4: Initialize the minibatch size of DQN as $k = 32$
- 5: Initialize the total number of training episodes $N = 500$, the number of episodes at each cycle consisting of three phases of training $N_0 = 50$, and the number of episodes at each separate phase of training $N_1 = 10$ [It would be a wise choice to separate the warm-up phase, the autoencoder update phase, and the LSTD update phase at each cycle]
- 6: Initialize the number of maximum time step per episode $T = 1000$
- 7: Initialize the weights θ_0 for linear approximation function randomly between $(0, 1)$ which will be used for LSTD
- 8: Initialize a replay memory buffer \mathcal{D} with a capacity $N \times T$
- 9: Initialize a discount factor $\gamma = 0.9$
- 10: Initialize a small constant $\lambda = 1 \times 10^{-3}$ to initialize A^{-1} for LSTD
- 11: Given the number of actions as N_a and embedding dimension N_e , initialize a matrix θ with shape $N_a \times N_e$
- 12: Initialize a tensor $A_a^{-1} = \lambda^{-1}I$ and a tensor $b_a = 0$ to store the tensors that suffice $\theta_a = A_a^{-1}b_a, \forall a \in A$ 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.  $\theta$ 
2: Freeze the weights of  $\mathbf{A}$ , i.e.  $w$ 
3: for episode  $e_0 = 1$  to  $N_1$  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. autoencoders]
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  $\epsilon$ -greedy policy with  $\epsilon = 0.2$  with an  $\epsilon$  decay
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:  end for
15: end for
16:  $e = e + N_1$ 
[1]
```

Algorithm 3 Autoencoder update phase

```
1: Freeze the weights of  $f(\phi(s))$ , i.e.  $\theta$ 
2: Unfreeze the weights of  $\mathbf{A}$ , i.e.  $w$ 
3: Reset  $A_a^{-1}$  and  $b_a$  to the default value at the initialization phase,  $\forall a \in A$  at each time step  $t$ 
4: for episode  $e_0 = 1$  to  $N_1$  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  $\mathbf{A}$ 
11:    First encode, then decode  $d$  using  $\mathbf{A}$ 
12:    Calculate the loss  $L$  of  $s_t$  by comparing the input and output of  $\mathbf{A}$ 
13:    Update  $w$  with  $o$  as per  $L$ 
14:   end for
15: end for
16:  $e = e + N_1$ 
```

[1]

Algorithm 4 LSTD update phase

```
1: Freeze the weights of  $f(\phi(s))$ , i.e.  $\theta$ 
2: Unfreeze the weights of  $\mathbf{A}$ , i.e.  $w$ 
3: for episode  $e_0 = 1$  to  $N_1$  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  $\theta$ ]
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:  $e = e + N_1$ 
[1]
```

Algorithm 5 Training procedure

```
1: Run Initialization
2: while episode  $e \leq N$  do
3:   Run Warm-up phase
4:   Run Autoencoder update phase
5:   Run LSTD update phase
6:   Run Autoencoder update phase
7:   if  $e + N_1 \geq N$  then
8:     Reset  $A_a^{-1}$  and  $b_a$  to the default value at the initialization phase,  $\forall a \in A$  at each time step  $t$ 
9:   end if
10:  Run LSTD update phase
11: end while
```

[1]