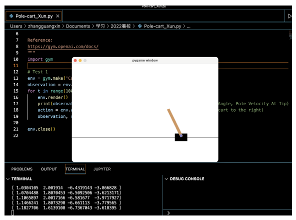
Fundamental of Control Theory Course Report

Guangxin Zhang
Peking University
Yuanpei College

1.Make the model work.


Pole-cart_Xun.py
Users > zhangguangxin > Documents > 学习 > 2022暑校 > Pole-cart_Xun.py > ...
pygame window
PROBLEMS
OUTPUT
TERMINAL
JUPYTER
DEBUG CONSOLE

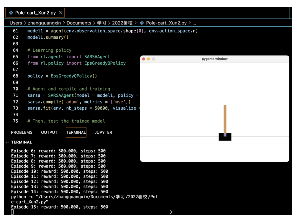
```
Reference:
https://gym.openai.com/docs/
"""
import gym

# Test 1
env = gym.make('Ca
observation = env.
for t in range(100
    env.render()
    print(observat                                 Angle, Pole Velocity At Tip)
    action = env.a                                 cart to the right)
    observation,

env.close()
```

```
[ 1.0304105  2.001914  -6.4319143 -3.866828 ]
[ 1.0704488  1.8070453 -6.5092506 -3.6213171]
[ 1.1065897  2.0017166 -6.581677  -3.9717927]
[ 1.1466241  1.8073298 -6.661113  -3.779565 ]
[ 1.1827706  1.6139108 -6.7367043 -3.618395 ]
```

Figure 1: Pole-cart_Xun.py


Pole-cart_Xun2.py
Users > zhangguangxin > Documents > 学习 > 2022暑校 > Pole-cart_Xun2.py > ...
pygame window
PROBLEMS
OUTPUT
TERMINAL
JUPYTER

```
model1 = agent(env.observation_space.shape[0], env.action_space.n)
model1.summary()

# Learning policy
from rl.agents import SARSAAgent
from rl.policy import EpsGreedyQPolicy

policy = EpsGreedyQPolicy()

# Agent and compile and training
sarsa = SARSAAgent(model = model1, policy =
sarsa.compile('adam', metrics = ['mse'])
sarsa.fit(env, nb_steps = 50000, visualize =

# Then, test the trained model
```

```
Episode 6: reward: 500.000, steps: 500
Episode 7: reward: 500.000, steps: 500
Episode 8: reward: 500.000, steps: 500
Episode 9: reward: 500.000, steps: 500
Episode 10: reward: 500.000, steps: 500
Episode 11: reward: 500.000, steps: 500
Episode 12: reward: 500.000, steps: 500
Episode 13: reward: 500.000, steps: 500
Episode 14: reward: 500.000, steps: 500
python -u "/Users/zhangguangxin/Documents/学习/2022暑校/Pol
e-cart_Xun2.py"
Episode 15: reward: 500.000, steps: 500
```

Figure 2: Pole-cart_Xun2.py

2.What does DQN mean?

DQN Is the short term of Deep Q-Network.To understand DQN, first explain Q learning.
(1)Q-learning

The core of reinforcement learning is to select an appropriate policy to maximize the total reward at the end of each epoch. Q learning algorithm is one of reinforcement learning.

Definition：**Policy**

The agent's policy is $\pi(a|s) = P(A_t = a|S_t = s)$, which represents the probability of taking action A under the condition that the input state is S.

Definition：**state-value function**

In Markov decision process (MDP), V_π is defined to represent the state-value function based on policy π, and is the expectation of reward (G_t) that can be obtained by following policy π starting from state S.

Definition：**Action-value function**

$Q^\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = r(s,a) + \gamma \sum\limits_{s'\in\mathcal{S}} P(s'|s,a)V^\pi(s')$ represents the expected reward of action A performed on the current state S when MDP follows π policy. That is, when using policy π, the value of action A in state S is equal to the immediate reward $r(s,a)$ plus the product of the state transition probability of all possible next states that have been attenuated with the corresponding value. We use the discount factor ($\gamma \in [0,1)$), because the long-term benefits are uncertain, and sometimes we need to get some rewards as soon as possible, so we need to discount the long-term benefits.

When using policy π, the value of state S is equal to the sum result of the probability of taking all actions based on policy π in this state multiplied by the corresponding values :$V^\pi(s) = \sum\limits_{a\in\mathcal{A}} \pi(a|s)Q^\pi(s,a)$

Its Behrman expectation equation is $Q^\pi(s,a) = E_\pi[R_t + \gamma Q^\pi(s',a')|S_t = s, A_t = a] = r(s,a) + \gamma \sum\limits_{s'\in\mathcal{S}} P(s|s',a) \sum\limits_{a'\in\mathcal{A}} \pi(a'|s')Q^\pi(s',a')$

Define optimal value-function $V^*(s) = \max\limits_{\pi} V^\pi(s), \forall s \in \mathcal{S}$, and optimal action-value function $Q^*(s,a) = \max\limits_{\pi} Q^\pi(s), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$.To maximize $Q^\pi(s,a)$, the optimal policy needs to be executed for the current state action pair (s,a).The relationship between the optimal value state function and the optimal action value function is obtained: $Q^*(s,a) = r(s,a) + \gamma \sum\limits_{s'\in\mathcal{S}} P(s'|s,a)V^*(s')$ The optimal action value is the state value when selecting the action that maximizes the action value:$V^* = \max\limits_{a\in\mathcal{A}} Q^*(s,a)$

Finally, the Behrman optimal equation is obtained: $Q^*(s,a) = r(s,a) + \gamma \sum\limits_{s'\in\mathcal{S}} P(s|s',a) \max\limits_{a'\in\mathcal{A}} Q^*(s',a')$

Using the temporal difference(TD) method, the Action-value function can be updated:$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max\limits_{a} Q(s_{t+1},a) - Q(s_t,a_t)]$

(2)DQN

In cases where state and behavior can be exhausted, Q(s,a) is generally recorded using an S-A table. If states and combinations are inexhaustible, deep learning needs to be introduced.

With the idea of function approximation, the neural network is used to represent function Q. The input of the neural network is state S and action A, and the output is a scalar to represent the value obtained by using action A in S state. The neural network is called Q network. For a set of data (s_i, a_i, r_i, s_i'), its loss function can be defined as mean square error.: $\omega^* = \arg\min\limits_{\omega} \frac{1}{2N} \sum\limits_{i=1}^{N} [Q_\omega(s_i,a_i) - (r_i + \gamma \max\limits_{a'\in\mathcal{A}} Q_\omega(s_i',a'))]^2$

In DQN, the method of experience replay is adopted to maintain a replay buffer pool, and the quad data (state, action, reward, next state) obtained from the environment is stored in the working memory every time. During the training of Q network, random sampling is taken from the working memory for training.

DQN uses the target network. Since the constant update of the Q network in the training process will lead to the change of the target, the Q network in the TD error target network can be fixed and use 2 sets of networks.

a. The original target network $Q_\omega(s,a)$ is used to calculate $Q_\omega(s,a)$ items in the original loss function $\frac{1}{2}[Q_\omega(s,a) - (r + \gamma max_{a'} Q_{\omega^-}(s',a'))]^2$, and updated with the normal gradient descent method.

b. Target network, willing to calculate $r + \gamma max_{a'} Q_{\omega^-}(s',a')$items in the loss function, where ω^- represents parameters in the target network. To make the update target more stable, the target network is updated every C step.

3.Why the best reward is 500?

Gym is a standard API for reinforcement learning, and a diverse collection of reference environments.In the cart Pole V1 setup, our goal was to keep the pole balanced for 500 frames. We will fail once the pole is tilted more than 15 degrees from totally vertical or the cart moves more than 2.4

units From the middle position. After each frame, the score increases by one, and at 500 frames the attempt is truncated, so the best score is 500.

4.What does 'SARSAAgent' mean?

A SARSA(State–action–reward–state–action) agent is a value-based reinforcement learning agent that trains a critic to estimate the return or future rewards. For a given observation, the agent selects and outputs the action for which the estimated return is greatest.

5.modify the network size and compare the different performance

I add the number of layers of the network and the number of neurons in each layer, to see how they behave differently.

Due to the randomness of sampling, agent's performance is not stable. So I did 30 repetitions of each situation, averaging them out and measuring their performance.

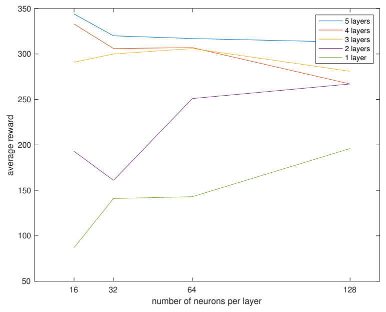


Figure 3: Different layers & Different neuron numbers

As can be seen from the figure, with the deepening of neural network layers and the increase of neurons, the average score gradually increases. However, when network is more complex than 3 layers and 16 neurons each layer, such improvement becomes insignificant, because this task is too simple and the demand for neural network prediction ability is not strict. When the parameters are too many, they can even be overfitted.

6.Explain the agent and the policy (EpsGreedyQPolicy)

An agent is something that acts. SARSAAgent's behavior is controlled by the SARSAA algorithm. The action-value function is directly estimated by TD algorithm: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ Then use the $\epsilon - greedyalgorithm$ to choose action.

Fake code:

```
Initialize Q(s,a), $\forall s \in S, a \in A(s)$,arbitrarily, and Q(terminal-state,.) = 0
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
        $S \leftarrow S'; A \leftarrow A'$
    until S is terminal
```

7.Explain the whole coding structure inside the agent, such as what is forward, backward, and step and the associated connections?

I find the source code of SARSAAgent from https://github.com/keras-rl/keras-rl/blob/master/rl/agents/sarsa.py

The core operating logic of an agent is as follows:

(1) Initialize relevant parameters, create a trainable neural network model, including a lambda layer. The role of the Lambda layer is to sift through randomly sampled data, remove very unsatisfactory attempts, and learn from data that performs ok. Set up relevant optimizers, loss functions, etc.

(2) (Forward function) In the forward function, the agent calculates the Q value of the action under the current observation, returns the corresponding action according to the policy (different policies during training and testing), and updates the list of obserbvation and Actions

(3) (step function) The agent passes the action returned from the forward function to the step function, interacts with the environment, and returns the state of the agent after taking the action, the reward obtained, and whether it finishes.

(4) (back unction) is used for parameter updating.

In the test phase, parameters do not need to be updated. During the training phase, train the network on a single stochastic batch, and start by extracting the necessary parameters from it. Calculate the corresponding Q value and the target to be achieved after taking this action:$r_t + \gamma \times Q(s_{t+1}, a_{t+1})$ Then perform a single update on the entire batch. Use a dummy target since the actual loss is computed in a Lambda layer that needs more complex input. it is still useful to know the actual target to compute metrics properly.Lambda layer that needs more complex input. However,it is still useful to know the actual target to compute metrics properly.

Reference

[1] AUER P, CESA-BIANCHI N,FISCHER P. Finite-time analysis of the multiarmed bandit problem[J]. Machine Learning,2002,47(2):235-256

[2] JAAKKOLA T, JORDAN M I, SINGH S P. On the convergence of stochastic iterative dynamics programming algorithms [J]. Neural Computation, 1994,6(6):1185-1201

[3]MELO F S. Convergence of Q-learning: a simple proof [R].Institute of Systems and Robotics, Tech.2001:1-4

[4]RUMMERY G A,MAHESAN N. On-line q-learning using connnectionist systems [J].Technical Reprt, 1994

[5] SUTTON R S, BARTO A G. Reinforcement learning: an introduction [M]. Cambridge:MIT press,2018.

[6] Zhang W, Shen J, Yu Y. Hands-on Reinforcement Learning [M]. 1. POSTS & TELECOM PRESS, 2022.

[7]Keras.keras-rl[EB/OL].[2022.5.16].https://github.com/keras-rl/keras-rl/blob/216c3145f3dc4d17877be26ca2185ce7db462bad/rl/agents/sarsa.py#L148.