ChainMDP description

Team 4

June 8th 2022

1 Default Parameters

These are default parameters for our agent. All values are provided in file chain_test.py. There is no initial weight.

```
sa_list = []


for i in range(env.n):
    for j in range(2):
        sa_list.append((i, j))

agent_params = {'gamma'             : 0.9,
                'kappa'             : 1.0,
                'mu0'               : 0.0,
                'lamda'             : 4.0,
                'alpha'             : 3.0,
                'beta'              : 3.0,
                'max_iter'          : 100,
                'sa_list'           : sa_list}
```

2 Initialization

Initialize agent by calling agent.

```
agent = agent(agent_params).
```

3 Training method

Below is the code used in training for k episodes. Just modify number in training(k) for training for k episodes.

```
def training(k):

    for episode in range(k):
        s = env.reset()
        done = False

        while not done:
            a = agent.take_action(s, 0)

            # Step environment
            s_, r, done, t = env.step(a)
            agent.observe([t, s, a, r, s_])
            agent.update_after_step(10, True)

            # Update current state
            s = s_
```

4 Performance Evaluation

As it is difficult to provide explicit parameters, performance can be evaluated after training 1000 episodes with function in section 3. After training, evaluation for 50 episodes can be done with code below.

```
total_reward = 0

for episode in range(test_length):  #test_length = 50 in this case
    cum_reward = 0.0
    done = False
    s = env.reset()

    while not done:
        action = agent.take_action(s, 0)
        s_, r, done, t = env.step(action)
        cum_reward += r
        s = s_
    total_reward += cum_reward

return total_reward/test_length
```

5 Sample Efficiency

Our method does not have weights, therefore sample efficiency can be evaluated similarly as performance. While training 1000 episodes, rewards can be evaluated in each episode. Code is given below.

```
reward = []

for episode in range(episode_length):

    s = env.reset()
    done = False
    cum_reward = 0

    while not done:
        a = agent.take_action(s, 0)
        # Step environment
        s_, r, done, t = env.step(a)
        agent.observe([t, s, a, r, s_])
        agent.update_after_step(10, True)
        # Update current state
        s = s_
        cum_reward += r
    reward.append(cum_reward)

return reward
```

In above reward array, i^{th} component of reward contains reward of $i - 1^{th}$ episode.

References

[1] Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling, 2013.

[2] E. Markou and C. E. Rasmussen. Bayesian methods for efficient Reinforcement Learning in tabular problems. In *NeurIPS Workshop on Biological and Artificial RL*, 2019.

[3] E. Markou. https://github.com/stratisMarkou/sample-efficient-bayesian-rl, 2019.