Assignment3

20211145 이하은

1. Implement the Mountain Car problem from the textbook using the Sarsa(λ) algorithm described in the textbook and submit it as a single Python file (Assignment3.py). (o)

2. Copy the pseudo-code for the Sarsa(λ) algorithm from the textbook into your report and map each line of the pseudo-code to the corresponding part of your implemented algorithm in the report (Assignment3.pdf).

* 수도코드와 실제 코드 각 위치에 숫자로 표기하였습니다!

Sarsa(λ) with binary features and linear function approximation for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a function $\mathcal{F}(s,a)$ returning the set of (indices of) active features for s, a
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize: $\mathbf{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1, \ldots, z_d)^\top \in \mathbb{R}^d$

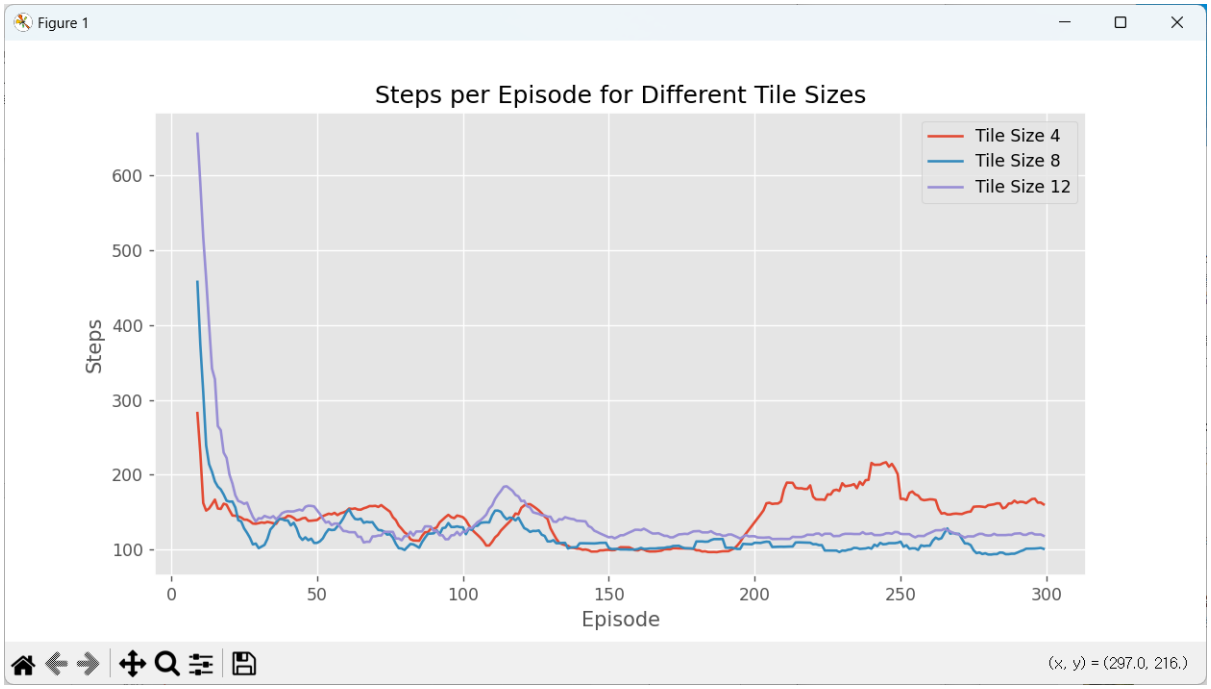
Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
  $\mathbf{z} \leftarrow \mathbf{0}$
  Loop for each step of episode:
    Take action A, observe R, S'
    $\delta \leftarrow R$
    Loop for i in $\mathcal{F}(S, A)$:
      $\delta \leftarrow \delta - w_i$
      $z_i \leftarrow z_i + 1$ (accumulating traces)
      or $z_i \leftarrow 1$ (replacing traces)
    If S' is terminal then:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
      Go to next episode
    Choose $A' \sim \pi(\cdot|S')$ or near greedily $\sim \hat{q}(S', \cdot, \mathbf{w})$
    Loop for i in $\mathcal{F}(S', A')$: $\delta \leftarrow \delta + \gamma w_i$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z}$
    $S \leftarrow S'; A \leftarrow A'$

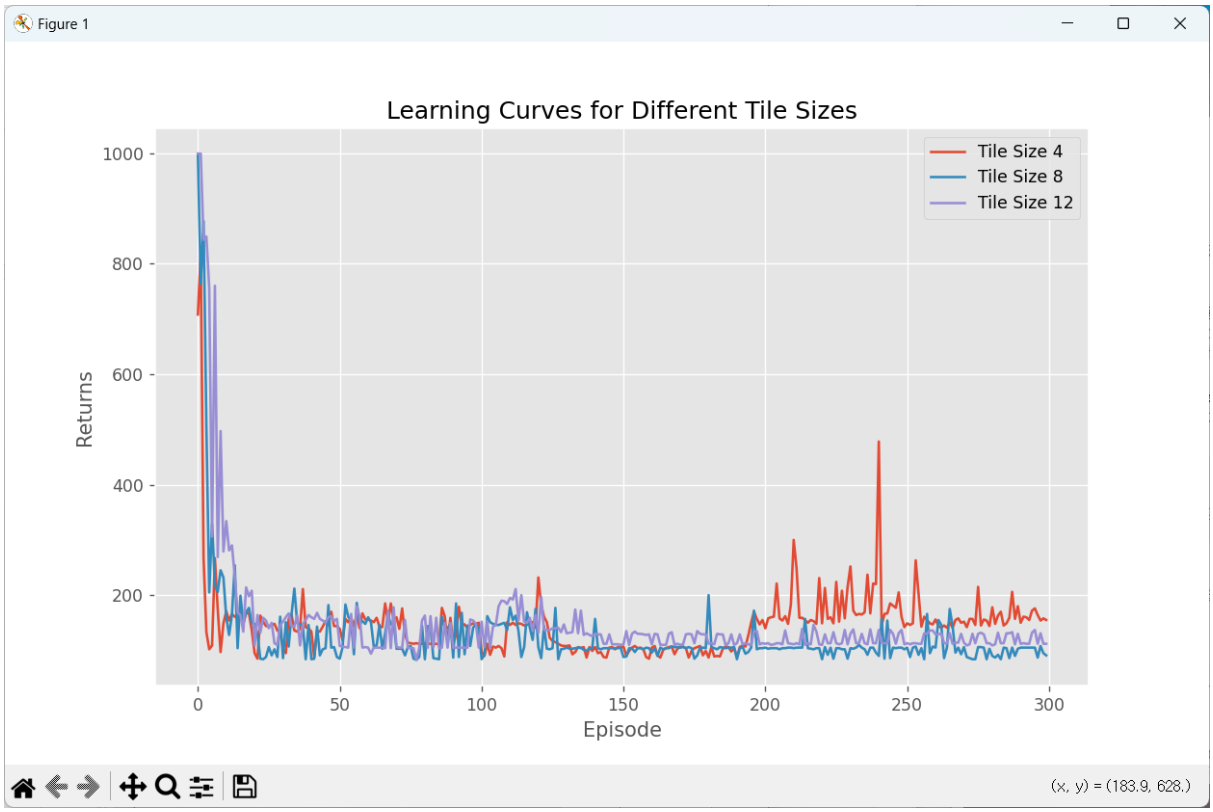
```python
def sarsa_lambda(lmbda, env, estimator, gamma=1.0, epsilon=0):

    """
    Sarsa(Lambda) algorithm
    for finding optimal q and pi via Linear
    FA with eligibility traces.
    """

    # Reset the eligibility trace
    estimator.reset(z_only=True)

    # Create epsilon-greedy policy
    policy = make_epsilon_greedy_policy(
        estimator, epsilon, env.action_space.n)

    # Reset the environment and pick the first action
    #state = env.reset()
    state, _ = env.reset()
    action_probs = policy(state)
    action = np.random.choice(np.arange(len(action_probs)), p=action_probs)
```

```python
def update(self, s, a, target):
    """
    Updates the estimator parameters
    for a given state and action towards
    the target using the gradient update rule
    (and the eligibility trace if one has been set).
    """
    features = self.featurize_state_action(s, a)
    estimation = np.sum(self.weights[features])  # Linear FA
    delta = (target - estimation)

    if self.trace:
        # self.z[features] += 1  # Accumulating trace
        self.z[features] = 1  # Replacing trace
        self.weights += self.alpha * delta * self.z
    else:
        self.weights[features] += self.alpha * delta
```

```python
ret = 0
# Step through episode
for t in itertools.count():
    # Take a step
    # next_state, reward, done, _ = env.step(action)
    next_state, reward, done, truncated, _ = env.step(action)
    done = done or truncated
    ret += reward

    if done:
        target = reward
        estimator.update(state, action, target)
        break

    else:
        # Take next step
        next_action_probs = policy(next_state)
        next_action = np.random.choice(
            np.arange(len(next_action_probs)), p=next_action_probs)

        # Estimate q-value at next state/action
        q_new = estimator.predict(
            next_state, next_action)[0]
        target = reward + gamma * q_new
        # Update step
        estimator.update(state, action, target)
        estimator.z *= gamma * lmbda

    state = next_state
    action = next_action

return t, ret
```

3. Run 300 episodes and plot the changes in the steps required to complete an episode for different numbers of tiles of 4, 8, and 12.





"Tile size= 8" 일 때, 가장 좋은 성능을 보인다.

4. During the 300 episodes, compare the short-term memory vector z and the long-term weight vector w in a single graph.

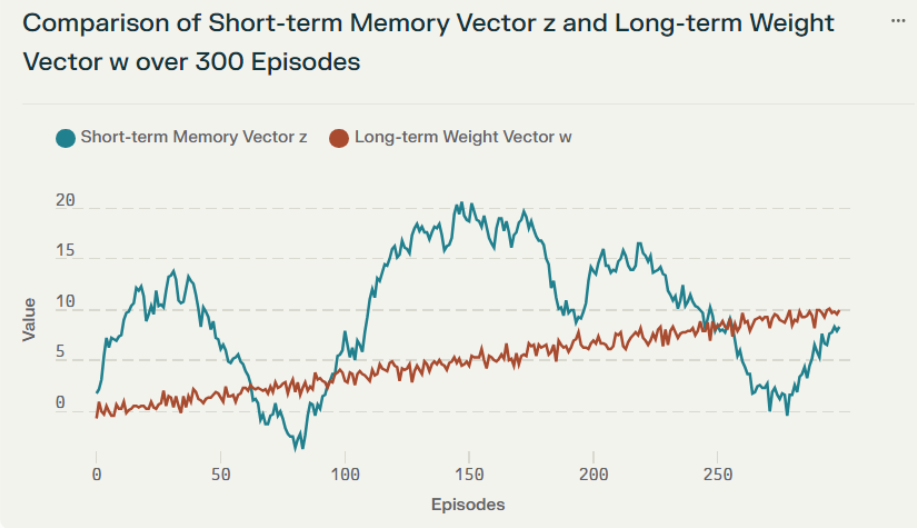
교수님, 죄송합니다.
벡터를 x축을 에피소드로 하고, y축을 vector v와 w를 norm 취한 것으로 출력하고자 시도했지만, 코드 에러로 해결하지 못했습니다.

이외에도 value값을 찍어보고자 하였지만, 성공하지 못했습니다.

하지만 조사결과, Z와 W를 비교해보면 다음과 같은 특성이 있었습니다.

측면	단기 메모리 벡터 z	장기 가중치 벡터 w
변동성	높음	낮음
변화에 대한 민감도	즉각적	점진적
추세	변동성 있음	부드럽고 상승세
목적	일시적 효과 포착	장기 지식 축적

그에 따른 예측되는 결과는 다음과 같습니다.



5. Using the results from Step 4, explain why eligibility traces are necessary. You should write the answer in KOREAN.

위의 결과를 토대로 보면, Z 벡터는 최근 결과에 매우 민감하게 반응하며 불규칙한 패턴을 보입니다. 반면, W 벡터는 더 안정적이고 점진적인 변화를 나타낸다. Eligibility traces

는 이 두 특성을 결합하여 단기적인 변화에 대응하면서도 장기적인 안정성을 유지하기 위한 것으로 보입니다.

과거의 상태와 행동이 현재의 보상에 미치는 영향을 추적하여, 멀리 떨어진 행동의 영향도 고려하여 Z에서는 좀 더 안정성을 제공하고, w에서는 최근 결과도 더 반영하도록 하는 효과가 있을 것입니다.