Deep Reinforcement Learning Mini Project

In this mini project, you will train a simple Deep Q-Network (DQN) agent to play a game in `gym`[1]. I recommend choosing `Pendulum-v1`, `CartPole-v1`, or `MountainCar-v0`, but you can choose any game you like. There are `jax` versions of `gym` like `gymnax` and `pgx`, which you could explore and use, but it is not necessary. Hand-in is due Sunday, September 15th, at 23:59. If your dog eats your cat, and your cat eats your homework, please let me know. The purpose of the mini-projects is to help you become ready for the final project, which is a bit more open-ended. We are on your side, and we want you to succeed.

The game

`gym` is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like `Pong` or `StarCraft`. The games are called environments, and they have a common interface that makes it easy to switch between them. The interface has three methods: `reset`, `step`, and `action_space`. The `reset` method initializes the environment and returns the initial observation and state. The `step` method takes an action and returns the next observation, state, reward, and a boolean indicating if the game is over. The `action_space` method returns the possible actions the agent can take.

The task

To train the deep Q-network, you will need to implement at least the following components:

- An MLP that takes the observation as input and returns the Q-values for each action.
- An agent that selects actions based on the Q-values.
- A memory buffer that stores transitions to be used for training.
- A training loop that samples from the buffer and updates the parameters of the MLP.

You should hand in a `.zip` file with the following files:

1. A `.py` script that I can run that trains your agent on the game.
2. A `.md` file with 250 words that explains your code and the results.
3. A `.pkl` file with the trained parameters of your agent.
4. A `.gif` of your agent is playing the game.
5. A `.gif` of a random agent playing the game.

You can base your hand-in zip on the `prjs/one/` directory of our repository if you like.

[1]https://gymnasium.farama.org/

The help

Deep Q-learning is a simple and powerful algorithm, but it can be a bit tricky to get right. From a signal processing perspective, it is actually an *infinite impulse response filter*, and there is a recurrent aspect to it that can be a bit tricky to wrap your head around. Implementing it for non-techies is sometimes type 3 fun. Talk to each other, ask questions.

The code

You are allowed to use `random`, `tree`, `grad`, `jit`, `lax`, `nn` and `vmap` from `jax` as well as `optax`, `chex`, along with other standard Python libraries like `numpy`, `matplotlib`, and `gym`. The agent memory buffer could be a `deque` from the `collections` module (store `n` entries and throw away the oldest when beyond capacity). When your agent has enough transitions, you can sample a batch from the buffer and update the parameters of your agent.

```python
# %% Imports (add as needed) #############################################
import gymnasium as gym  # not jax based
from jax import random
from tqdm import tqdm
import jax.numpy as jnp
from collections import deque

# %% Constants ###########################################################
env = gym.make("CartPole-v1")
keys = random.split(random.PRNGKey(0))
memory = deque(maxlen=1000)  # <- replay buffer
batch_size = 32
gamma = 0.99
epsilon = 0.1

# %% Model ###############################################################
def random_policy_fn(rng, obs): # action (shape: ())
    n = env.action_space.__dict__['n']
    return random.randint(rng, (1,), 0, n).item()

def action_fn(rng, params, obs):
        return  random_policy_fn(rng,  obs)  if  random.uniform(rng)  <  epsilon  else
policy_fn(rng, params, obs)

def policy_fn(rng, params, obs):
    raise NotImplementedError
```

```python
def sample_batch(rng, memory, batch_size):
    idxs = random.randint(rng, (batch_size,), 0, len(memory))
    batch = [memory[i] for i in idxs]
    return batch

# %% Environment ##########################################################
obs, info = env.reset()
for i in tqdm(range(1000)):

    keys = random.split(keys[0], 3)
    action = random_policy_fn(keys[1], obs)

    next_obs, reward, terminated, truncated, info = env.step(action)
    memory.append((obs, action, reward, next_obs, terminated | truncated))
    obs, info = next_obs, info if not (terminated | truncated) else env.reset()

    if len(memory) >= batch_size:
        batch = sample_batch(keys[2], memory, batch_size)
        # params = update_fn(params, batch)
        continue

env.close()


# %%

rng = random.PRNGKey(0)
idxs = random.randint(rng, (4,), 0, len(memory))
batch = [memory[i] for i in idxs]
```