به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی برق

مقدمه ای بر یادگیری ماشین ـ دکتر صالح کلیبر

نیم‌سال اول ۱۳۹۹

گزارش تمرین کامپیوتری سری پنجم

امید شرفی (۹۶۱۰۱۸۳۸)

فهرست مطالب

۱ بخش اول ۲

۲ بخش دوم ۲

۱.۲ ReplayBuffer ۲

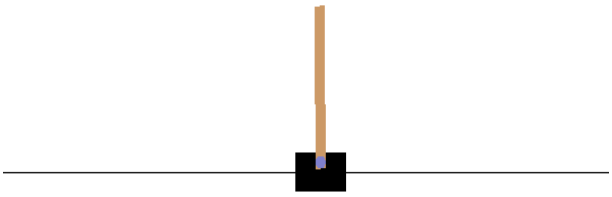
۲.۲ SummaryWriter ۴

۳ بخش سوم ۵

۴ بخش چهارم ۵

۱ بخش اول

مساله ی CartPole از کتابخانه Gym مشابه داکیومنت سایت حل شد و بهترین پاسخ نگه داشتن به طول ۲۰۰ که سقف مدنظر ما بود به دست آمد.



```
The best length is: 54.4
The best length is: 82.84
The best length is: 200.0
The best length is: 200.0
The best length is: 200.0
The best length is: 200.0
The best length is: 200.0
The best length is: 200.0
The best length is: 200.0
The best length is: 200.0
game lasted 200 moves
```

شکل ۱: خروجی مساله آونگ معکوس

۲ بخش دوم

۱.۲ ReplayBuffer

همانطور که در صورت سوال هم گفته شد، در یادگیری تقلیدی فاکتورهای مساله که رشته ی مرحله به مرحله ی یادگیری ما را میسازند شامل مشاهده یا همان حالت ما، اکشن ما، ریوارد و مشاهده و حالت حاصله در مرحله ی بعد هست. در کلاس ReplayBuffer دقیقا همین مشخصات و یک ترمینال به عنوان اتریبیوت های کلاس ذخیره میشود که دنباله ما را تشکیل میدهد و به path مرحله به مرحله اضافه میشود.

```python
import time
import numpy as np
import gym
import os

from hw5.infrastructure.utils import *

class ReplayBuffer(object):

    def __init__(self, max_size=1000000):

        self.max_size = max_size

        # store each rollout
        self.paths = []

        # store (concatenated) component arrays from each rollout
        self.obs = None
        self.acs = None
        self.rews = None
        self.next_obs = None
        self.terminals = None

    def __len__(self):
        if self.obs:
            return self.obs.shape[0]
        else:
            return 0
```

شکل ۲: کلاس ReplayBuffer

متد addrollouts از کلاس ReplayBuffer دقیقا وظیفه ی اضافه کردن موارد گفته شده در بالا را دارد و با گرفتن مسیرها به آرایه paths کلاس مسیرها را اضافه میکنیم و در ادامه هرکدام از اتریبیوت های ذکر شده در بالا حاصل convertlistofrollouts را در این مرحله به انتهای خود اضافه میکنند که در حقیقت آپدیت و افزوده شدن مقادیر این مرحله صورت بگیرد.

```python
def add_rollouts(self, paths, concat_rew=True):

    # add new rollouts into our list of rollouts
    for path in paths:
        self.paths.append(path)

    # convert new rollouts into their component arrays, and append them onto our arrays
    observations, actions, rewards, next_observations, terminals = convert_listofrollouts(paths, concat_rew)

    if self.obs is None:
        self.obs = observations[-self.max_size:]
        self.acs = actions[-self.max_size:]
        self.rews = rewards[-self.max_size:]
        self.next_obs = next_observations[-self.max_size:]
        self.terminals = terminals[-self.max_size:]
    else:
        self.obs = np.concatenate([self.obs, observations])[-self.max_size:]
        self.acs = np.concatenate([self.acs, actions])[-self.max_size:]
        if concat_rew:
            self.rews = np.concatenate([self.rews, rewards])[-self.max_size:]
        else:
            if isinstance(rewards, list):
                self.rews += rewards
            else:
                self.rews.append(rewards)
            self.rews = self.rews[-self.max_size:]
        self.next_obs = np.concatenate([self.next_obs, next_observations])[-self.max_size:]
        self.terminals = np.concatenate([self.terminals, terminals])[-self.max_size:]
```

شکل ۳: کلاس ReplayBuffer

و در نهایت متد samplerecentdata اطلاعات ذکر شده را برای آخرین مرحله و samplerandomdata به صورت رندم به تعداد batchsize که هنگام فراخوانی تابع پاس دادیم از دنباله مذکور برمیگرداند.

```
def sample_random_data(self, batch_size):
    assert self.obs.shape[0] == self.acs.shape[0] == self.rews.shape[0] == self.next_obs.shape[0] == self.terminals.shape[0]

    # Return batch_size number of random entries from each of the 5 component arrays above

    n= self.obs.shape[0]
    indecis= np.random.choice(n, batch_size)

    return self.obs[indecis], self.acs[indecis], self.rews[indecis], self.next_obs[indecis], self.terminals[indecis]

def sample_recent_data(self, batch_size=1):
    return self.obs[-batch_size:], self.acs[-batch_size:], self.rews[-batch_size:], self.next_obs[-batch_size:], self.terminals[-batch_size:]
```

شکل ۴: کلاس ReplayBuffer

SummaryWriter ۲.۲

همانطور که در صورت سوال هم اشاره شد به منظور استفاده و لاگ کردن نتایج از ابزار tensorboardX استفاده میشود که کلاس SummaryWriter وظیفه ی ذخیره سازی نتیجه در قالب فایل event را برای ما دارد که بعد از اجرای کد در پوشه ی data برای هر دیتاست ایجاد و ذخیره میشود.

```
class SummaryWriter(object):
    """Writes entries directly to event files in the logdir to be
    consumed by TensorBoard.

    The `SummaryWriter` class provides a high-level API to create an event file
    in a given directory and add summaries and events to it. The class updates the
    file contents asynchronously. This allows a training program to call methods
    to add data to the file directly from the training loop, without slowing down
    training.
    """

    def __init__(self, logdir=None, comment='', purge_step=None, max_queue=10,
                 flush_secs=120, filename_suffix='', write_to_disk=True, log_dir=None, **kwargs):
        """Creates a `SummaryWriter` that will write out events and summaries
        to the event file.

        Args:
            logdir (string): Save directory location. Default is
              runs/**CURRENT_DATETIME_HOSTNAME**, which changes after each run.
              Use hierarchical folder structure to compare
              between runs easily. e.g. pass in 'runs/exp1', 'runs/exp2', etc.
              for each new experiment to compare across them.
            comment (string): Comment logdir suffix appended to the default
              ``logdir``. If ``logdir`` is assigned, this argument has no effect.
            purge_step (int):
              When logging crashes at step :math:`T+X` and restarts at step :math:`T`,
              any events whose global_step larger or equal to :math:`T` will be
              purged and hidden from TensorBoard.
              Note that crashed and resumed experiments should have the same ``logdir``.
            max_queue (int): Size of the queue for pending events and
              summaries before one of the 'add' calls forces a flush to disk.
              Default is ten items.
            flush_secs (int): How often, in seconds, to flush the
              pending events and summaries to disk. Default is every two minutes.
            filename_suffix (string): Suffix added to all event filenames in
              the logdir directory. More details on filename construction in
              tensorboard.summary.writer.event_file_writer.EventFileWriter.
            write_to_disk (boolean):
              If pass `False`, SummaryWriter will not write to disk.
```

شکل ۵: توضیحات کلاس SummaryWriter

همانطور که در بالا و در داکیومنت SummaryWriter در سایت tensorboardx هم آماده است توضیح مربوط به پارامترهای خواسته شده به صورت زیر است :

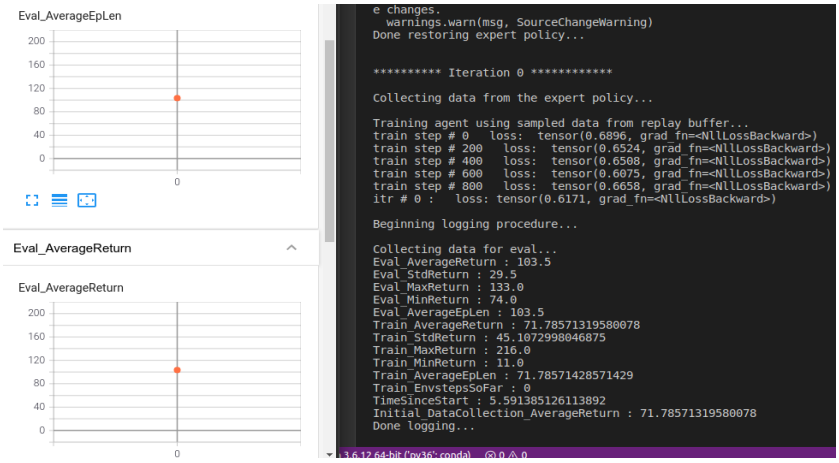
- logdir : مشخص کننده مکانی هست که خروجی های هر دور اجرا در آن قرار میگیرد.
- maxqueue : این مقدار که یک عدد است مشخص میکند که در صف event های در حال اجرایمان چه تعداد آیتم میتواند قرار گیرد قبل از این که add صدا شود که این خلاصه ذخیره شود و صف حافظه ما فلاش و خالی شود.
- flushsecs : این ویژگی هم یک عدد مشخص کننده زمان بر حسب ثانیه ای هست که آنقدر زمان که بگذرد نتایج event های موجود در دیسک ذخیره میشود و فلاش رخ میدهد.

۳ بخش سوم

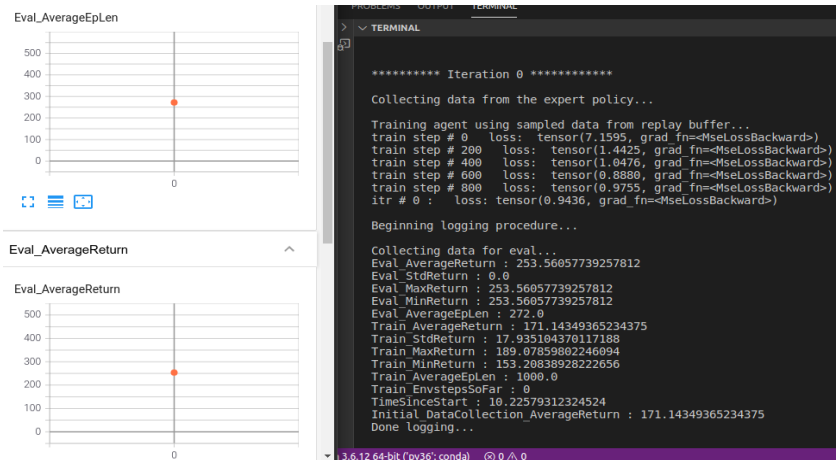
به ترتیب فایل README کارها پیش برده شد و فایل کدها تکمیل شد.

۴ بخش چهارم

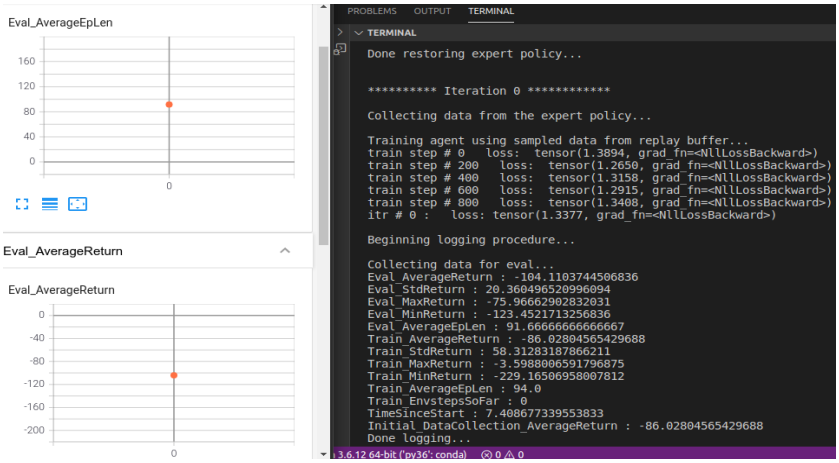
کد برای مساله CartPole و مساله LunarLander در دو حالت گسسته و پیوسته اجرا شد که نتایج در زیر آمده است. با توجه به این که فرمت مساله بدون dagger حل شده است و هیچ لیبل گذاری مجددی توسط مدل اکسپرت وجود ندارد نتایج صرفا به فرمت یک تکرار کلی و یک بار انجام فرآیند تکرارهای برای آموزش انجام شده و در نتیجه نتایج در tensorboard به صورت تک نقطه دیده میشود.



شکل ۶: CartPole

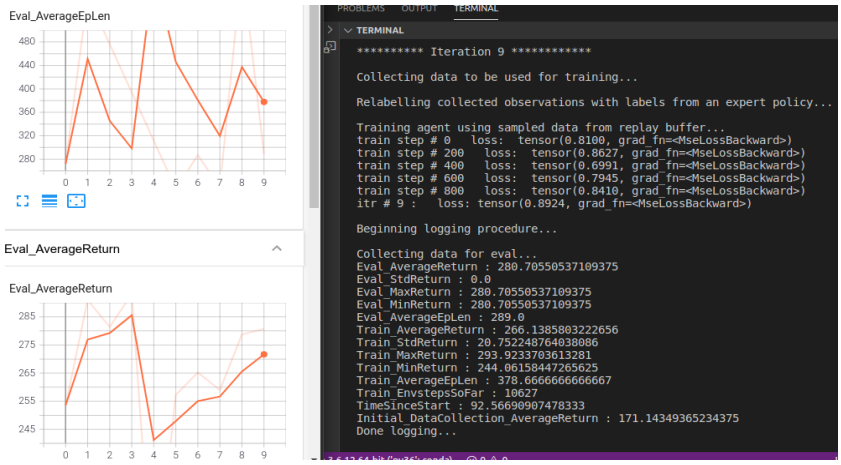


شکل ۷: LunarLanderContinuous

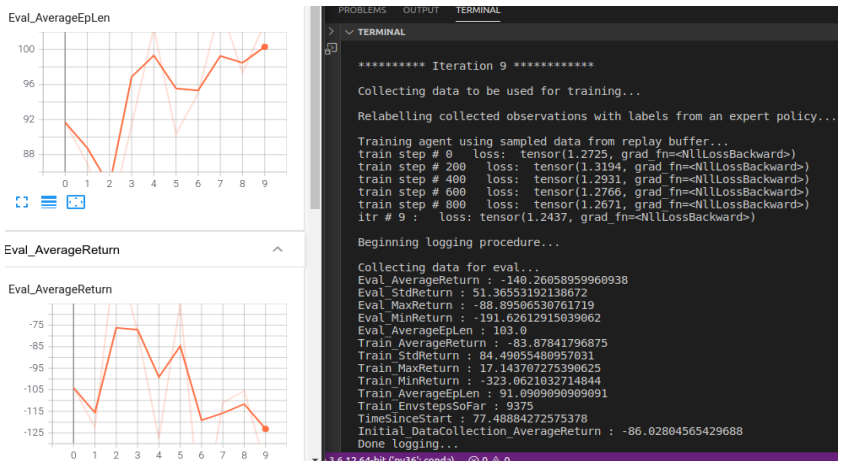


شکل ۸: LunarLander

همچنین خروجی ها برای حالت dagger در ده تکرار به فرم زیر هست.



شکل ۹: LunarLanderContinuous



شکل ۱۰: LunarLander