بسمه تعالی

مقدمه ای بر یادگیری ماشین

نیمسال اول ۱۳۹۹-۱۴۰۰

تمرین سری پنجم

مدرس: صابر صالح

• مهلت تحویل تمرین های عملی: ۱۳۹۹/۱۱/۱۷ •

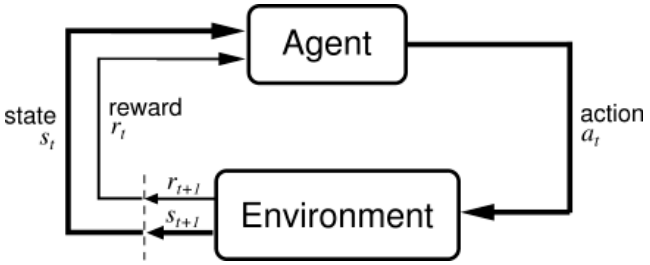
توضیحات

در این تمرین قصد داریم با Imitation Learning آشنا شویم. در بخش اول با یکی از ابزارهای اساسی که در طول این تمرین با آن سروکار داریم آشنا می‌شوید. در بخش دوم وارد بحث Imitation Learning می‌شویم و با یک‌سری از مفاهیم پایه آشنا می‌شویم. در بخش سوم اقدام به کامل کردن کدهای پیوست می‌کنید تا کار پیاده‌سازی، کامل شود.

۱ آشنایی با OpenAI Gym

کتابخانه gym توسط نهاد OpenAI جهت توسعه و مقایسه الگوریتم‌های متنوع در حوزه Reinforcement Learning معرفی شد. کتابخانه‌ی gym دارای محیط های متنوع جهت شبیه سازی است که در قسمت اول اقدام به شبیه سازی یکی از این محیط ها خواهیم کرد. برای آشنایی بیشتر با پروژه‌ی OpenAI Gym می‌توانید از این‌جا کمک بگیرید.

دو مفهوم بنیادی در حوزه Reinforcement Learning عبارت‌اند از Agent و Environment. به طور خلاصه، Agent موجودی است که الگوریتم ما را اجرا می‌کند و Environment درواقع محیطی است که قرار است Agent در آن یک فعالیتی داشته باشد. در هر مرحله، این Agent یک Action در Environment انجام می‌دهد. متناسب با Action ای که Agent در محیط انجام می‌دهد، یک Reward و یک Observation از محیط دریافت می‌کند. می‌توانید فرض کنید که Observation نمایان‌گر وضعیت یا State بعدی است. هدف الگوریتم موجود در Agent این است که با توجه به این نتایج، Action های درستی انجام شوند تا Agent در مسیر مطلوب قرار بگیرد. این مدل در شکل زیر نمایش داده شده است.



شکل ۱: مدل Agent-Environment

در این قسمت قصد داریم که مسئله‌ی CartPole (آونگ واژگون) را پیاده سازی کنیم. (فایل hw5\Gym-Introduction.py)
در این مسئله Stateهای سیستم دارای چهار مولفه‌ی (cart position, cart velocity, pole angle, velocity of pole angle) می‌باشند. قصد داریم که ۱۰۰ بار محیط مدنظر را شبیه سازی کنیم و در هر بار شبیه سازی، یک سری وزن تصادفی تولید می‌کنیم. در هر بار شبیه سازی، فرآیند یادگیری را به این صورت انجام می‌دهیم که Action را بر اساس ضرب داخلی وزن‌ها در Stateهای محیط انتخاب می‌کنیم. اگر حاصل مثبت بود، به سمت راست حرکت کرده و در غیر این صورت، به سمت چپ حرکت می‌کنیم. پس از پایان هر شبیه سازی، میانگین تعداد مراحل مورد نیاز برای رسیدن به حالت مطلوب را محاسبه کرده و با بهترین طول تعداد مراحل مقایسه می‌کنیم که اگر بیشتر بود، بهترین طول مراحل و وزن‌ها بروزرسانی می‌شوند. پس از پایان یافتن فرآیند و پیدا کردن وزن های بهینه، محیط را یک بار دیگر به کمک بهترین وزن‌ها شبیه سازی کرده و خروجی گرافیکی را ذخیره می‌کنیم.

۲ ورود به Imitation Learning

در این قسمت از تمرین، هدف اصلی آشنایی با Imitation Learning و به طور ویژه‌تر Behavior Cloning می‌باشد. در این روش یک Expert ، یک کارِ نیازمندِ مهارت (مثلاً یک بازی) را در محیطی انجام می‌دهد تا Agent از رفتار این Expert آموزش ببیند و بتواند خود، آن کار را انجام بدهد. برای رسیدن به این هدف، سعی می‌کنیم که رفتارِ Expert در یک محیط را به طور موثری ذخیره کنیم تا Agent بتواند از این اطلاعات برای یادگیریِ یک Policy مناسب جهت موفقیت در محیط استفاده کند. به این ترتیب برای یادگیری Policy می‌توانیم از داده‌های ثبت شده و یک الگوریتم Supervised Learning مناسب استفاده کنیم.

به این ترتیب می‌توان انتظار داشت که قسمت اصلی داده‌ای که اصولاً در این روش ثبت و از آن استفاده می‌کنیم، حالت در مرحله‌ی t، Action انجام شده در مرحله‌ی t و حالت در مرحله‌ی $t+1$ می‌باشد. در ادبیات Imitation Learning گاهی به این مجموعه از داده‌ها که از عملکرد یک Expert در محیط حاصل می‌شوند Replay Buffer گفته می‌شود.

- فایل hw5\infrastructure\replay_buffer.py, hw5\infrastructure\utils.py را مشاهده کنید. توضیحی مختصر از اجزا و عملکرد کلاس ReplayBuffer بنویسید.

- در بحث Imitation Learning مانند هر روش دیگری در یادگیری ماشین نیاز داریم که نتایجی را ثبت و بررسی کنیم؛ به خصوص که در Imitation Learning هدف آموزش دادنِ یک Agent برای فعالیت در یک محیط است و لازم داریم که عملکردِ Agent آموزش‌یافته را در قالب فیلم یا عکس ذخیره و بررسی کنیم. به همین جهت نیاز به یک ابزار قدرتمند و البته کم دردسر(!) داریم. پکیج tensorboardX واسطی موثر برای برنامه نویس ایجاد می‌کند تا بتواند به روشی استاندارد شده و ساده داده‌های موردنظرش را به شکل مورد علاقه‌اش log کند. برای آشنایی با tensorboardX به logger.py مراجعه کنید و به سوالات زیر پاسخ دهید. در این فایل یک کلاس به نام Logger ایجاد شده است.

  - همان‌طور که در __init__ دیده می‌شود، یک instance از کلاس SummaryWriter ایجاد شده است و مطابق این خط به کلاس Logger اضافه شده است: self._summ_writer = SummaryWriter(log_dir, flush_secs=1, max_queue=1)
    به طور خلاصه نقش هر یک از پارامترهای flush_secs ، log_dir و max_queue را بیان کنید.

۳ پیاده‌سازی Imitation learning

ترتیب فایل هایی که باید کامل شوند و در نهایت دستور هایی که باید ران شوند در فایل README آماده اند.

- همه کد ها در فایل hw5\scripts\hw5_behavior_cloning.py استفاده شده اند و شما در نهایت همین فایل در هر قسمت ران میکنید. این فایل از hw5\infrastructure\rl_trainer.py استفاده میکند که وظیفه پیاده سازی الگوریتم بر عهده آن است. در واقع یک Agent در آن ساخته شده و با استفاده از داده هایی که جمع آوری کرده و Expert داده شده به آن، Agent را آموزش می دهد.

- قبل از پیاده سازی الگوریتم، فایل hw5\infrastructure\torch_utils را کامل کنید. در این فایل شما یک شبکه عصبی با تعداد لایه ها و پهنا (تعداد نورون ها در هر لایه) را پیاده خواهید کرد. برای سادگی، پهنای لایه های پنهان یکسان فرض شده است. همچنین شما باید از مقداردهی اولیه[1] Xavier استفاده کنید تا شبکه سریع تر یاد گرفته شود و همچنین در مینیمم های محلی گرفتار نشود.

- فایل hw5\policies\MLP_policy.py را کامل کنید. این قسمت اصلی پیاده سازی است. برای ساختن Policy ما مسئله را به دو حالتی که تصمیم[2] ها پیوسته یا گسسته باشند تقسیم می‌کنیم.

  در حالت گسسته، شبکه عصبی بردار مشاهده را دریافت کرده و به ما بردار logits را می‌دهد که در واقع لگاریتم نرمالایز نشده بردار احتمال یک متغیر تصادفی Multinomial می‌باشد. سپس این بردار logits به یک توزیع Multinomial داده شده تا تصمیم گیری به صورت تصادفی انجام گیرد. بنابراین برای یادگیری مدل، باید پارامتر های شبکه عصبی یادگیری شود.

  در حالت پیوسته اما شبکه عصبی بر اساس بردار مشاهده های ورودی، میانگین یک توزیع گاوسی را خروجی میدهد که لگاریتم انحراف معیار آن برابر logstd میباشد. یعنی:

$$a_i \sim \text{mean}(s_i) + e^{\text{logstd}}\mathcal{N}(0,1)$$

  پس برای یادگیری مدل، پارامتر های شبکه عصبی و logstd باید یادگیری شوند.

  سپس این تصمیم ها، با تصمیم هایی که توسط Expert گرفته شده اند مقایسه شده و یادگیری انجام می‌شود.

- hw5\policies\loaded_gaussian_policy.py را کامل کنید که در واقع همان expert را پیاده سازی می‌کند. در ابتدا مدل یادگیری شده را لود میکنید که در حالت گسسته شبکه عصبی، و در حالت پیوسته شبکه عصبی و logstd است و سپس مطابق آنچه در قسمت قبل گفته شد تصمیم گیری می‌شود.

- در نهایت باید دو بخش در rl_trainer کامل شود. در بخش rl_trainer.collect_training_trajectories دقت کنید در ابتدا داده ای وجود ندارد و در مرحله اول از Expert برای جمع آوری داده استفاده می‌شود. در مرحله های بعدی اما از خود مدل برای جمع آوری داده استفاده خواهد شد.

  همچنین اگر از DAgger استفاده شود، داده هایی که با مدل به دست آمده اند، دوباره لیبل زده خواهند شد. یعنی تصمیم های مدل با تصمیم Expert جایگزین می‌شوند. شما این را در rl_trainer.do_relabel_with_expert پیاده می‌کنید.

۴ تست ها

شما با دو محیط CartPole-v0, LunarLander-v2 کار میکنید. همانطور که گفته شد دستور ها در README قرار دارند. چنانچه ویدیو ها در سیستم شما مشکل دارند میتوانید با اضافه کردن video_log_freq=-1– آن را غیر فعال کنید.

[1] Initialization
[2] action