علی آراسته ۹۶۱۰۱۱۶۵

مقدمه‌ای بر یادگیری ماشین – تمرین عملی سری پنجم

۱. آشنایی با OpenAI Gym

در این سوال مسئله CartPole مورد بررسی قرار می‌گیرد. در ابتدا یک env از نوع مسئله CartPole ساخته می‌شود. سپس محیط مورد نظر ۱۰۰ بار شبیه‌سازی می‌شود. در هر بار شبیه‌سازی، ابتدا یک بردار ۴ تایی به صورت تصادفی و .i.i.d از توزیع unif(-1, 1)، به عنوان بردار وزن انتخاب می‌شود. در ادامه به ازای هر بردار تصادفی وزن، آزمایش مورد نظر ۱۰۰ بار تکرار می‌شود و تعداد مراحل دوام آوردن در هر بار آزمایش ثبت می‌گردد. در آخر میانگین تعداد مراحل در ۱۰۰ بار آزمایش محاسبه می‌شود. اگر این مقدار از بهترین مقدار میانگین ثبت شده بیشتر باشد، بیشترین تعداد مراحل و بهترین بردار وزن به‌روزرسانی می‌شوند. همچنین بعد از هر ۱۰ بار شبیه‌سازی، بیشترین تعداد مراحل در خروجی پرینت می‌شود.

The best length is: 60.28

The best length is: 84.33

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

پس از پایان شبیه‌سازی، آزمایش مورد نظر باری دیگر با بهترین بردار وزن انجام می‌شود و ویدئو آن ثبت می‌گردد که در فایل‌های ضمیمه موجود است.game lasted 200 moves

۲. ورود به Imitation Learning

- بررسی فایل‌های utils.py و replay_buffer.py

ابتدا به بررسی فایل utils.py می‌پردازیم. در این فایل چندین تابع تعریف شده اند.

تابع اول ()sample_trajectory است. این تابع محیط شبیه‌سازی، policy تصمیم‌گیری، ماکسیمم طول مسیر و دو متغیر دیگر را به عنوان ورودی دریافت می‌کند. این تابع با استفاده از پارامترهای ورودی، یک بار آزمایش مورد نظر را در محیط شبیه‌سازی داده شده انجام می‌دهد و مقادیر observation، action انجام شده بر اساس policy تصمیم‌گیری و observation، reward بدست آمده بر اساس action انجام شده و observation بعدی در نتیجه action انجام شده را برای هر مرحله ذخیره می‌کند و با استفاده از تابع ()Path این مقادیر را در قالب یک دیکشنری بازمی‌گرداند.

تابع sample_trajectories() عملکردی شبیه تابع sample_trajectory() دارد. با این تفاوت که این تابع یک ورودی اضافی به عنوان حداقل تعداد مراحل خروجی دریافت می‌کند. این تابع با استفاده از تابع sample_trajectory()، آنقدر آزمایش مورد نظر را در محیط داده شده انجام می‌دهد تا مجموع تعداد مراحل از پارامتر داده شده بیشتر شود.

تابع sample_n_trajectories() یک ورودی اضافی به عنوان تعداد تکرار آزمایش دریافت می‌کند. این تابع نیز با استفاده از تابع sample_trajectory()، آزمایش مورد نظر را به اندازه پارامتر داده شده در محیط داده شده انجام می‌دهد.

در آخر نیز تابع convert_listofrollouts() قرار دارد که با دریافت خروجی سه تابع دیگر، مقادیر مربوط به هر یک موارد ذکر شده در آزمایش‌های مختلف را با یکدیگر concatenate می‌کند.

حال فایل replay_buffer.py را مورد بررسی قرار می‌دهیم. در این فایل کلاس ReplayBuffer تعریف شده است.

در تابع سازنده این کلاس ماکسیمم تعداد مراحل قابل ذخیره‌سازی در بافر شی ساخته شده از کلاس مشخص می‌شود.

تابع add_rollouts()، خروجی آزمایش‌های انجام شده توسط توابع فایل utils.py را به عنوان ورودی دریافت می‌کند و با در نظر گرفتن ملاحظاتی، این مقادیر را در بافرهای خود ذخیره می‌کند. در انتها اگر تعداد مراحل ذخیره شده در بافر بیش از ماکسیمم تعیین شده در تابع سازنده باشد، داده‌های مربوط به مراحل قدیمی اضافی پاک می‌شوند.

توابع sample_random_data() و sample_recent_data() برای استخراج داده مربوط به تعدادی مرحله (به اندازه پارامتر ورودی)، یکی به صورت تصادفی و دیگری به صورت اولویت مراحل اخیر، استفاده می‌شوند.

- بررسی پارامترهای تابع سازنده کلاس SummaryWriter

پارامتر log_dir: آدرس فولدری که فایل‌های log در آن ذخیره می‌شوند.

پارامتر max_queue: اندازه صف رویدادها و خلاصه‌های معلق، قبل از این که یک فراخوانی ’add‘ باعث یک تخلیه روی دیسک شود. در واقع اگر تعداد رویدادها و خلاصه‌های معلق بیشتر از اندازه صف شود، همه داده‌های موجود در صف روی دیسک ذخیره می‌شوند.

پارامتر flush_secs: هر چند وقت یک بار (بر حسب ثانیه) رویدادها و خلاصه‌های معلق روی دیسک تخلیه شوند.

۳. پیاده‌سازی Imitation Learning

ابتدا فایل torch_utils.py را تکمیل می‌کنیم. در این فایل تابع build_map() پیاده‌سازی شده بود.

این تابع بر اساس مقادیر ورودی (اندازه ورودی، اندازه خروجی، تعداد لایه‌های مدل Sequential، اندازه پهنای هر لایه پنهان، نوع تابع فعال‌سازی و نوع تابع فعال‌سازی خروجی) مدل Sequential مدنظر را ایجاد می‌کند. همچین پس از ایجاد مدل، برای مقداردهی اولیه پارامترهای آن از روش Xavier، از روش Xavier استفاده می‌کند.

سپس به بررسی فایل MLP_policy.py می‌پردازیم.

برای تابع get_action()، اگر مدل گسسته باشد، با عبور دادن خروجی مدل (logits) از یک لایه nn.Softmax()، بردار احتمال مورد نظر را بدست می‌آوریم و سپس با استفاده از توابع torch.distributions.Categorical() و sample()، یک نمونه از توزیع چندجمله‌ای مطابق با بردار احتمال داده شده انتخاب می‌کنیم. اما اگر مدل پیوسته باشد، خروجی‌های مدل را به عنوان میانگین‌های توزیع‌هایی گوسی در نظر می‌گیریم که انحراف معیار آن‌ها برابر logstd است. در این حالت action مورد نظر به صورت زیر دست می آید:

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

در تابع define_train() برای هر دو حالت از torch.optim.Adam() به عنوان تابه بهینه‌ساز استفاده می‌شود. سپس اگر مدل گسسته باشد، تابع torch.nn.CrossEntropyLoss() و اگر مدل پیوسته باشد، تابع torch.nn.MSELoss() به عنوان self.loss_fn انتخاب می‌شود.

در تابع update()، ابتدا actions خروجی مدل به ازای ورودی observations بر اساس نوع مدل (گسسته یا پیوسته) بدست می‌آید. سپس با استفاده از self.loss_fn، مقدار loss بدست می‌آید. در آخر با استفاده از تابع backward()، مشتق تابع loss در نقطه مورد نظر بر اساس تمام ورودی‌ها محاسبه می‌شود و تابع بهینه‌ساز یک مرحله جلو می‌رود. اگر مدل پیوسته باشد، بردار logstd نیز به طور مناسب به‌روزرسانی می‌شود.

در مورد فایل loaded_guassian_policy.py و تابع get_action() موجود در آن مانند تابع get_action() موجود در فایل MLP_policy.py عمل می‌کنیم.

در آخر فایل rl_trainer.py را کامل می‌کنیم. در این فایل تنها کافیست در موارد استفاده از تابع sample_trajectories()، policyهای مناسب را انتخاب کنیم.

۴. تست‌ها

با استفاده از دستورهای داده شده، تست‌های مورد نظر را انجام می‌دهیم. خروجی به صورت زیر بدست می‌آید:

:CartPole_v0

********** Iteration 0 **********

Collecting data from the expert policy...

Training agent using sampled data from replay buffer...

train step # 0 loss: 0.6873262

train step # 200 loss: 0.4988811

train step # 400 loss: 0.57866675

train step # 600 loss: 0.48271343

train step # 800 loss: 0.495741

itr # 0 : loss: 0.5799124

Beginning logging procedure...

Collecting data for eval...

Eval_AverageReturn : 200.0

Eval_StdReturn : 0.0

Eval_MaxReturn : 200.0

Eval_MinReturn : 200.0

Eval_AverageEpLen : 200.0

Train_AverageReturn : 200.0

Train_StdReturn : 0.0

Train_MaxReturn : 200.0

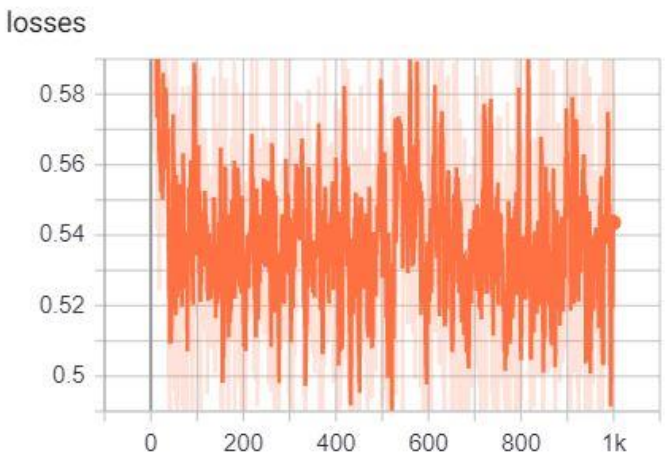
Train_MinReturn : 200.0

Train_AverageEpLen : 200.0

Train_EnvstepsSoFar : 0

TimeSinceStart : 4.026633024215698

Initial_DataCollection_AverageReturn : 200.0



********** Iteration 0 **********

Collecting data from the expert policy...

Training agent using sampled data from replay buffer...

train step # 0 loss: 1.3992506

train step # 200 loss: 1.0066552

train step # 400 loss: 1.0178783

train step # 600 loss: 0.97849554

train step # 800 loss: 0.9872981

itr # 0 : loss: 0.97366977

Beginning logging procedure...

Collecting data for eval...

Eval_AverageReturn : 216.51358032226562

Eval_StdReturn : 0.0

Eval_MaxReturn : 216.51358032226562

Eval_MinReturn : 216.51358032226562

Eval_AverageEpLen : 302.0

Train_AverageReturn : 253.76011657714844

Train_StdReturn : 73.06890106201172

Train_MaxReturn : 330.295654296875

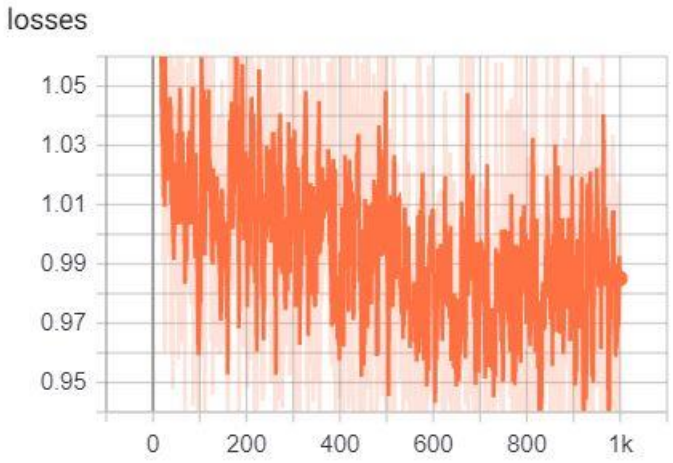
Train_MinReturn : 57.69931411743164

Train_AverageEpLen : 241.11111111111111

Train_EnvstepsSoFar : 0

TimeSinceStart : 3.437389612197876

Initial_DataCollection_AverageReturn : 253.76011657714844



********** Iteration 0 **********

Collecting data from the expert policy...

Training agent using sampled data from replay buffer...

train step # 0 loss: 6.836036

train step # 200 loss: 0.90017486

train step # 400 loss: 1.0944022

train step # 600 loss: 0.8829808

train step # 800 loss: 0.68888915

itr # 0 : loss: 1.0422677

Beginning logging procedure...

Collecting data for eval...

Eval_AverageReturn : 253.09841918945312

Eval_StdReturn : 0.0

Eval_MaxReturn : 253.09841918945312

Eval_MinReturn : 253.09841918945312

Eval_AverageEpLen : 466.0

Train_AverageReturn : 164.8460235595703

Train_StdReturn : 15.891426086425781

Train_MaxReturn : 187.18746948242188

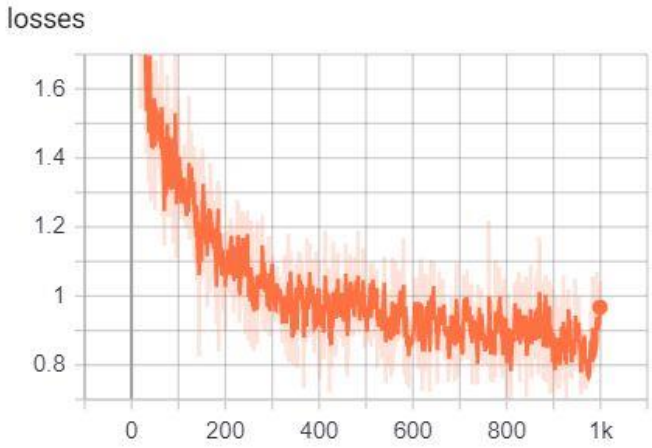
Train_MinReturn : 151.5655975341797

Train_AverageEpLen : 1000.0

Train_EnvstepsSoFar : 0

TimeSinceStart : 6.702923059463501

Initial_DataCollection_AverageReturn : 164.8460235595703



********** Iteration 9 **********

Collecting data to be used for training...

Relabelling collected observations with labels from an expert policy...

Training agent using sampled data from replay buffer...

train step # 0 loss: 1.0658088

train step # 200 loss: 1.0600219

train step # 400 loss: 1.1043267

train step # 600 loss: 1.1575863

train step # 800 loss: 1.1398376

itr # 9 : loss: 1.142091

Beginning logging procedure...

Collecting data for eval...

Eval_AverageReturn : 221.1179656982422

Eval_StdReturn : 0.0

Eval_MaxReturn : 221.1179656982422

Eval_MinReturn : 221.1179656982422

Eval_AverageEpLen : 304.0

Train_AverageReturn : 264.1106872558594

Train_StdReturn : 32.964229583740234

Train_MaxReturn : 298.141357421875

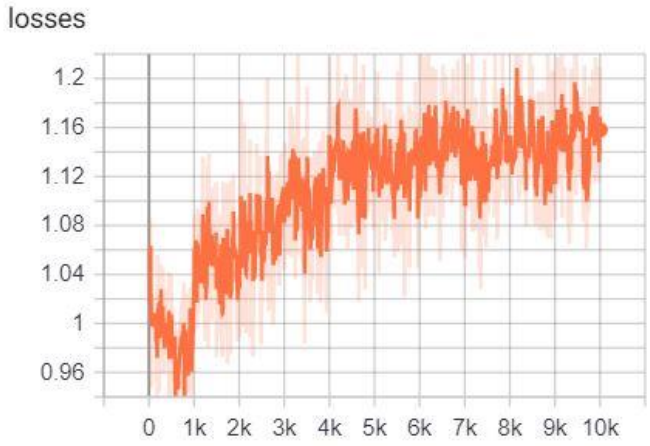
Train_MinReturn : 217.82135009765625

Train_AverageEpLen : 309.5

Train_EnvstepsSoFar : 10825

TimeSinceStart : 32.77191495895386

Initial_DataCollection_AverageReturn : 253.76011657714844



********** Iteration 9 **********

Collecting data to be used for training...

Relabelling collected observations with labels from an expert policy...

Training agent using sampled data from replay buffer...

train step # 0 loss: 0.82150376

train step # 200 loss: 0.792761

train step # 400 loss: 0.8096359

train step # 600 loss: 0.71625435

train step # 800 loss: 0.636304

itr # 9 : loss: 0.90246737

Beginning logging procedure...

Collecting data for eval...

Eval_AverageReturn : 276.7572326660156

Eval_StdReturn : 0.0

Eval_MaxReturn : 276.7572326660156

Eval_MinReturn : 276.7572326660156

Eval_AverageEpLen : 449.0

Train_AverageReturn : 284.6541442871094

Train_StdReturn : 9.78848648071289

Train_MaxReturn : 297.7679748535156

Train_MinReturn : 274.2576599121094

Train_AverageEpLen : 380.3333333333333

Train_EnvstepsSoFar : 10854

TimeSinceStart : 41.31275200843811

Initial_DataCollection_AverageReturn : 164.8460235595703