به نام خدا

گزارش پروژه درس یادگیری ماشین

استاد درس:

دکتر جواد سلیمی

دانشجو:

مهرداد چهارباغی

دانشگاه کاشان

دانشکده برق و کامپیوتر

زمستان 1402

پروژه اول:

الگوریتم‌های Contextual bandit و پیاده‌سازی آن‌ها را با با Pearl و با استفاده از یک محیط Contextual bandit برای ارائه داده‌ها از مجموعه داده‌های UCI نشان می‌دهیم و عملکرد SquareCB، LinUCB و LinTS را آزمایش می‌کنیم.

ابتدا باید توضیحاتی راجع به این سه و همچنین خود Contextual bandit داده شود که در ادامه آورده شده است.

- **Contextual bandit:**

نوعی مسئله در یادگیری تقویتی است. که در این الگوریتم نه تنها باید از بین اکشن های مختلف (یا "**Bandit**") برای به حداکثر رساندن پاداش خود انتخاب کند، بلکه **contextual** (یا حالت) را نیز در نظر می گیرد که در آن تصمیم گرفته می شود. بخش "contextual" به اطلاعات اضافی موجود در زمان تصمیم گیری اشاره دارد که می تواند برای پیش بینی نتیجه یا پاداش هر اقدام با دقت بیشتری مورد استفاده قرار گیرد.

در این مسئله ، قبل از تصمیم گیری، الگوریتم با اطلاعات Contextual مربوط به هر تصمیم ارائه می شود. این Context می تواند به طور قابل توجهی بر پاداش مورد انتظار یک عمل تأثیر بگذارد.

چالش اصلی در مشکلات **Contextual bandit** ایجاد تعادل بین اکتشاف[1] (آزمودن اقدامات مختلف برای جمع‌آوری اطلاعات بیشتر در مورد پاداش‌های آنها) و بهره‌برداری[2] (انتخاب شناخته‌شده‌ترین اقدام برای به حداکثر رساندن پاداش) است. الگوریتم باید از تصمیمات گذشته و نتایج آنها درس بگیرد و استراتژی خود را بر اساس بازخورد پاداش و Contextای که در آن تصمیم گرفته شده است به روز کند.

- **SquareCB**

الگوریتم SquareCB فقط به یک مدل رگرسیون نیاز دارد که با آن تابع پاداش[3] را یاد می گیرد. با توجه به مدل پاداش، SquareCB سیاست زیر را اجرا می کند.

[1] Exploration
[2] Explotation
[3] Reward

$$\widehat{a}_* \in \arg\max_a \hat{r}(x,a)$$
$$\hat{r}_* \in \max_a \hat{r}(x,a)$$
$$\text{If } a \neq \widehat{a}_* : \pi(a,x) = \frac{1}{A + \gamma(\hat{r}_* - \hat{r}(x,a))}$$
$$\text{If } a = \widehat{a}_* : \pi(a,x) = 1 - \sum_{a' \neq \widehat{a}_*} \pi(a',x).$$

این سیاست اکتشافی که اکتشاف[4] و بهره برداری[5] را به شیوه ای هوشمندانه متعادل می کند.

برای استفاده از الگوریتم SquareCB در Pearl، یادگیرنده سیاست[6] را به عنوان NeuralBandit تنظیم می کنیم. NeuralBandit یک کلاس پایه است و از تخمین تابع پاداش با معماری عصبی پشتیبانی می کند. با داشتن دسترسی به یک مدل پاداش تخمینی، می‌توانیم ماژول کاوش را با ماژول SquareCBExploration نمونه سازی می کنیم.

- **LinUCB**

در Pearl از نسخه عصبی الگوریتم LinUCB استفاده می‌شود که از نوع UCB کاوش با معماری های عصبی استفاده می کند. LinUCB و نسخه عصبی آن، تعمیم‌هایی از الگوریتم UCB (Upper Confidence Bound) هستند. هر دو یک سیاست به شکل زیر را اجرا می کنند:

$$\pi(a,x) \in \arg\max_a \hat{r}(x,a) + \text{score}(x,a)$$

یعنی، هر دو از تابعی استفاده می‌کنند که پاداش مورد انتظار را با یک عبارت پاداش اضافی تخمین می‌زند، که پتانسیل انتخاب یک عمل را با توجه به زمینه خاصی کمیت می‌دهد. یک روش معمول برای تخمین تابع امتیاز، در حالت خطی، زمانی که ویژگی‌ها φ(x,a) هستند، از طریق:

$$\text{score}(x,a) = \alpha ||\phi(x,a)||_{A^{-1}}$$

به طوریکه

$$A = \lambda I + \sum_{n \leq t} \phi(x_n, a_n)\phi^T(x_n, a_n)$$

[4] Exploration
[5] Explotation
[6] Policy Learnr

- **LinTS**

در ادامه، نحوه استفاده از نسخه عصبی الگوریتم LinTS را با کتابخانه Pearl، یعنی الگوریتمی که **از کاوش نمونه‌برداری تامپسون**[7] با معماری‌های عصبی استفاده می‌کند، توضیح می‌دهیم. نمونه برداری LinTS ارتباط نزدیکی با الگوریتم LinUCB دارد، با یک اصلاح کلیدی که اغلب همگرایی آن را در عمل بهبود می بخشد: ***تابع امتیاز را از یک احتمال نمونه برداری کنید، به جای اینکه آن را به طور قطعی ثابت کنید.*** در عمل، اینکار اغلب بازوهای کاوش بیش از حد را کاهش می دهد، زیرا امتیاز ممکن است کمتر از الگوریتم LinUCB باشد.

مراحل:

- **نصب:**

```
%pip uninstall Pearl -y
%rm -rf Pearl
!git clone https://github.com/facebookresearch/Pearl.git
%cd Pearl
%pip install .
%cd ..
```

ابتدا باید با دستوراتی که در کد بالا آورده شده،پکیج Pearl را نصب کنیم.

- **وارد کردن ماژول‌ها و کتابخانه ها:**

چون تعداد ماژول هایی که باید وارد شوند زیاد است از نوشتن آنها اینجا خودداری کرده چون تمام آنها در کد پیاده‌سازی آورده شده است.

- **بارگذاری محیط:**

```
1. # load environment
2. device = torch.device("cuda" if torch.cuda.is_available() else
   "cpu")
3.
4. # Download UCI dataset
5. uci_data_path = "./utils/instantiations/environments/uci_datasets"
6. if not os.path.exists(uci_data_path):
7.     os.makedirs(uci_data_path)
8.     download_uci_data(data_path=uci_data_path)
```

ابتدا باید دیتاست مورد نیاز را دریافت کرده تا بتوانیم از آن استفاده کنیم.این کار در خطوط 4 تا 8 انجام شده است.

[7] Thompson sampling exploration

```
 Built CB environment using the pendigits UCI dataset
pendigits_uci_dict =  {
    "path_filename": os.path.join(uci_data_path,
"pendigits/pendigits.tra"),
    "action_embeddings": "discrete",
    "delim_whitespace": False,
    "ind_to_drop": [],
    "target_column": 16,
}
env = SLCBEnvironment(**pendigits_uci_dict)
```

خب نوبت ساخت محیط Contexual Banditمی‌رسد. در اینجا از دیتاست pendigit استفاده شده است.

```
number_of_steps = 10000
record_period = 400
```

و در اینجا هم تعداد قدم ها(مراحل).

- **پیاده سازی SquareCB**

با توجه به طراحی ماژولار Pearl، ما از OneHotActionTensorRepresentationModule به عنوان ماژول نمایش اکشن استفاده می کنیم. یعنی، زمانی که مجموعه عمل تعداد محدودی از عناصر {1،2،.،، N} به عنوان یک بردار One-hot باشد.

```
1. action_representation_module =
   OneHotActionTensorRepresentationModule(
2.     max_number_actions= env._action_space.n,
3. )
4.
5. agent = PearlAgent(
6.     policy_learner=NeuralBandit(
7.         feature_dim = env.observation_dim + env._action_space.n,
8.         hidden_dims=[64, 16],
9.         training_rounds=50,
10.             action_representation_module=action_representation_mod
   ule,
11.             exploration_module= SquareCBExploration(gamma =
   env.observation_dim * env._action_space.n * number_of_steps)
12.          ),
13.          replay_buffer=FIFOOffPolicyReplayBuffer(100_000),
14.          device_id=-1,
15.      )
```

در خط اول نوع اکشن و فضای آن مشخص شده است.

در خط 5 تا 12 : ویژگی ایجنت خود را مشخص میکنیم.در اینجا سیاست یادگیری ما از نوع عصبی خواهد بود.
و تابع اکتشاف هم از نوع SquareCB.

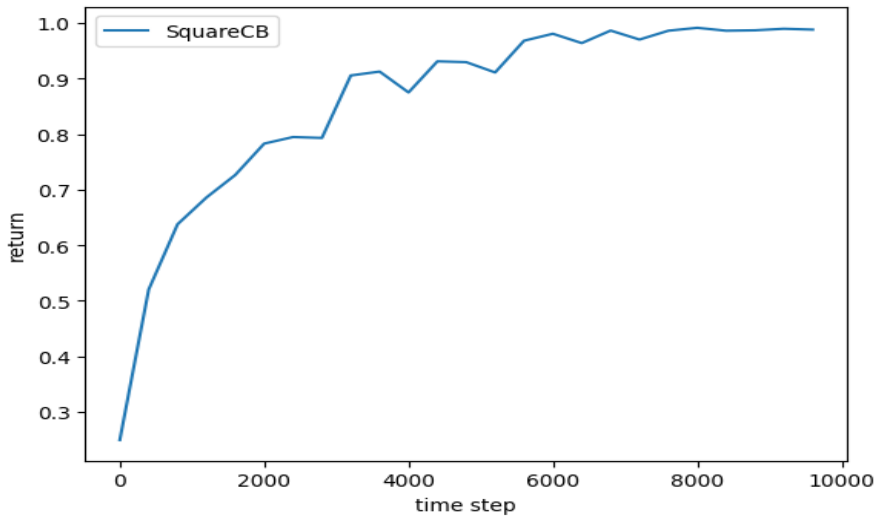
در ادامه:

```
info = online_learning(
    agent=agent,
    env=env,
    number_of_steps=number_of_steps,
    print_every_x_steps=100,
    record_period=record_period,
    learn_after_episode=True,
)
```

همه ویژگی ها رو یکجا تجمیع می‌کنیم.

و اما در آخر نتیجه را نمایش می‌دهیم:

```
torch.save(info["return"], "SquareCB-return.pt")
plt.plot(record_period * np.arange(len(info["return"])), info["return"],
label="SquareCB")
plt.xlabel("time step")
plt.ylabel("return")
plt.legend()
plt.show()
```



- **LinUCB**

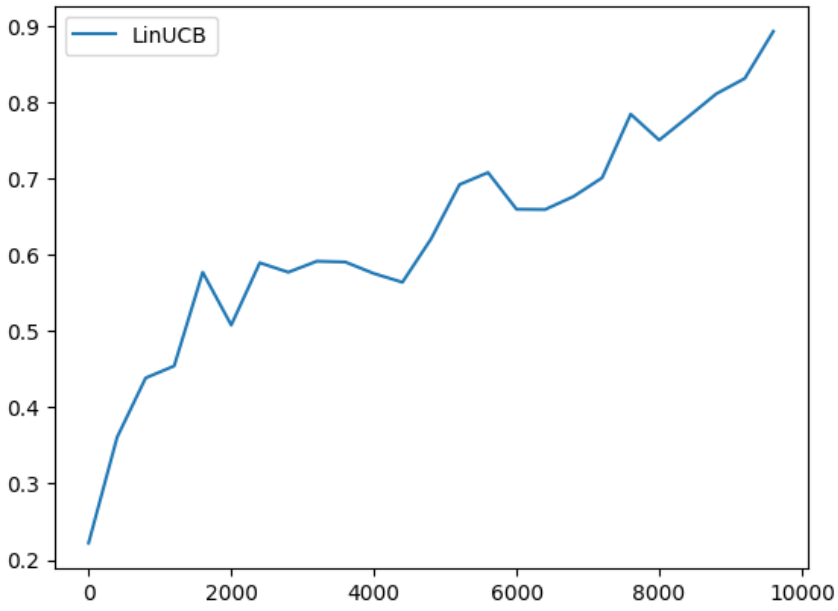
برای پیاده سازی الگوریتم LinUCB در Pearl، از ماژول یادگیرنده با سیاست **NeuralLinearBandit** استفاده می کنیم. این ماژول از (i) یادگیری یک مدل پاداش پشتیبانی می کند، و (ii) یک تابع امتیاز را با تخمین عدم قطعیت با استفاده از ویژگی های لایه آخر محاسبه می کند. علاوه بر این، ماژول کاوش را روی **UCBExploration** تنظیم می کنیم.

```
agent = PearlAgent(
    policy_learner=NeuralLinearBandit(
        feature_dim = env.observation_dim + env._action_space.n,
        hidden_dims=[64, 16],
        training_rounds=50,
        action_representation_module=action_representation_module,
        exploration_module= UCBExploration(alpha=1.0)
    ),
    replay_buffer=FIFOOffPolicyReplayBuffer(100_000),
    device_id=-1,
)
info = online_learning(
    agent=agent,
    env=env,
    number_of_steps=number_of_steps,
    print_every_x_steps=100,
    record_period=record_period,
    learn_after_episode=True,
)
```

در اینجا هم نوع سیاست یادگیر را روی عصبی خطی تنظیم می کنیم و ماژول اکتشاف را رو UCB .

و نمایش نتیجه:

```
torch.save(info["return"], "LinUCB-return.pt")
plt.plot(record_period * np.arange(len(info["return"])), info["return"],
label="LinUCB")
plt.xlabel("time step")
plt.ylabel("return")
plt.legend()
plt.show()
```



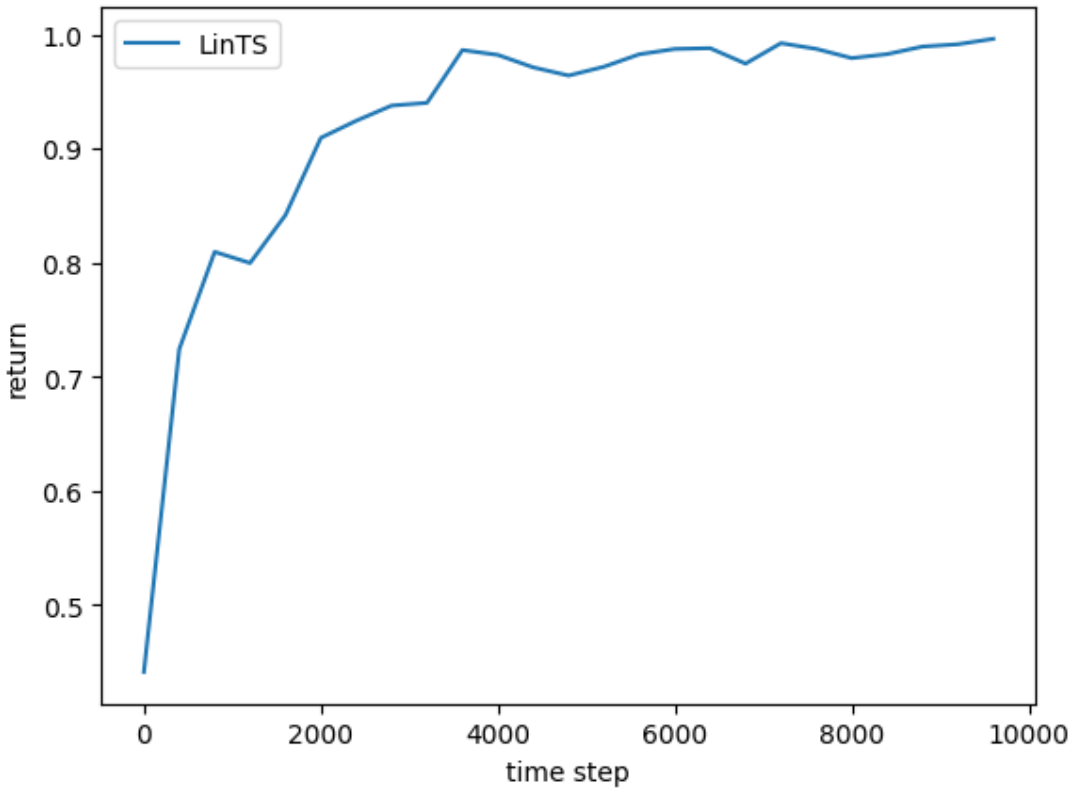
- **LinTS**

برای پیاده سازی الگوریتم LinTS در Pearl، از ماژول یادگیری سیاست NeuralLinearBandit استفاده می‌کنیم. علاوه بر این، ماژول کاوش را روی ThompsonSamplingExplorationLinear تنظیم می کنیم. این عامل را قادر می سازد تا امتیاز را بر اساس عدم قطعیت تخمینی خود نمونه برداری کند، به جای اینکه آن را مانند الگوریتم LinUCB تصحیح کند.

```
action_representation_module = OneHotActionTensorRepresentationModule(
    max_number_actions= env._action_space.n,
)

agent = PearlAgent(
    policy_learner=NeuralLinearBandit(
        feature_dim = env.observation_dim + env._action_space.n,
        hidden_dims=[64, 16],
        training_rounds=50,
        action_representation_module=action_representation_module,
        exploration_module= ThompsonSamplingExplorationLinear()
    ),
    replay_buffer=FIFOOffPolicyReplayBuffer(100_000),
    device_id=-1,
)
```

```
info = online_learning(
    agent=agent,
    env=env,
    number_of_steps=number_of_steps,
    print_every_x_steps=100,
    record_period=record_period,
    learn_after_episode=True,
)
torch.save(info["return"], "LinTS-return.pt")
plt.plot(record_period * np.arange(len(info["return"])), info["return"],
label="LinTS")
plt.xlabel("time step")
plt.ylabel("return")
plt.legend()
plt.show()
```



پروژه دوم:

پیاده سازی دریاچه یخ زده[8] با استفاده از Pearl

این پروژه نحوه استفاده از DQN را برای حل محیط FrozenLake-v1 نشان می دهد. این محیط مشاهداتی[9] را به عنوان دارد. در ادامه، نحوه استفاده از پکیج و ماژول Pearl`s OneHotObservationsFromDiscrete را برای تبدیل مشاهدات به نمایش های One-hot آنها نشان می دهیم.در ادامه همراه با کد پیاده سازی قدم به قدم جلو می‌رویم و توضیح داده می‌شود که هرقسمت چه کاری انجام می‌دهد.

مراحل:

- **نصب:**

```
%pip uninstall Pearl -y
%rm -rf Pearl
!git clone https://github.com/facebookresearch/Pearl.git
%cd Pearl
%pip install .
%cd ..
```

ابتدا باید با دستوراتی که در کد بالا آورده شده،پکیج Pearl را نصب کنیم.

- وارد کردن ماژول‌ها و کتابخانه های مورد نیاز:

```
from pearl.utils.functional_utils.experimentation.set_seed import set_seed
from pearl.policy_learners.sequential_decision_making.deep_q_learning import DeepQLearning
from pearl.replay_buffers.sequential_decision_making.fifo_off_policy_replay_buffer import FIFOOffPolicyReplayBuffer
from pearl.utils.functional_utils.train_and_eval.online_learning import online_learning
from pearl.pearl_agent import PearlAgent
from pearl.utils.instantiations.environments.gym_environment import GymEnvironment
from pearl.utils.instantiations.environments.environments import (
    OneHotObservationsFromDiscrete,
)
from pearl.utils.instantiations.spaces.discrete import DiscreteSpace
import torch
```

[8] Frozen Lake
[9] Observation

```
import matplotlib.pyplot as plt
import numpy as np
from pearl.action_representation_modules.one_hot_action_representation_module
import (OneHotActionTensorRepresentationModule)
set_seed(0)
```

- حل مسئله:

مسئله را با استفاده از Vanilla DQN حل می‌کنیم:

```
number_of_steps = 20000
record_period = 400
```

ابتدا تعداد قدم[10] هارو مشخص می‌کنیم.

```
env = OneHotObservationsFromDiscrete(
    GymEnvironment(
        "FrozenLake-v1", is_slippery=False, map_name="4x4",
    )
)
```

در اینجا مسئله ای را که میخواهیم حل کنیم ویژگی های آن را مشخص می‌کنیم.(به طور مثال ابعاد زمین،لغزنده بودن...)

```
action_representation_module = OneHotActionTensorRepresentationModule(
    max_number_actions= env.action_space.n,
)
```

تعیین نوع اکشن و فضای آن

```
1 state_dim = env.observation_space.n
2 agent = PearlAgent(
3    policy_learner=DeepQLearning(
```

[10] Step

```
4         state_dim=state_dim,
5         action_space=env.action_space,
6         hidden_dims=[64, 64],
7         training_rounds=1,
8         action_representation_module=action_representation_module
9     ),
10     replay_buffer=FIFOOffPolicyReplayBuffer(1000),
)
```

در اینجا در خط اول فضای ابعاد حالت را مشخص می‌کنیم.

در خط 2 تا 9 نوع ایجنت[11] که در اینجا از نوع ایجن Pearl قاعدتا باید باشد به همراه سیاستی که باید پیش بگیرد و دیگر ویژگی ها آن را مشخص می‌کنیم.

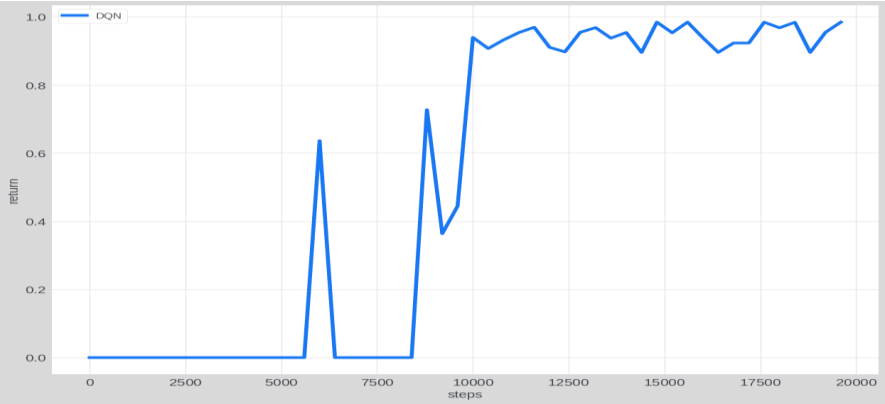
```
info = online_learning(
    agent=agent,
    env=env,
    number_of_steps=number_of_steps,
    print_every_x_steps=100,
    record_period=record_period,
    learn_after_episode=False,
)
```

در اینجا نوع یادگیری که اینجا آنلاین هست و آرگومانهای ورودی آن مثل ایجنت که از قبل مشخص کردیم،محیط،تعداد قدم‌ها و ... مشخص می‌کنیم.

```
torch.save(info["return"], "DQN-return.pt")
plt.plot(record_period * np.arange(len(info["return"])), info["return"],
label="DQN")
plt.xlabel("steps")
plt.ylabel("return")
plt.legend()
plt.show()
```

[11] Agent

نمایش نتیجه :



با افزایش تعداد قدم ها، می‌توانیم افزایش در مقدار متوسط بازگشتی را مشاهده کنیم.