به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین امتیازی

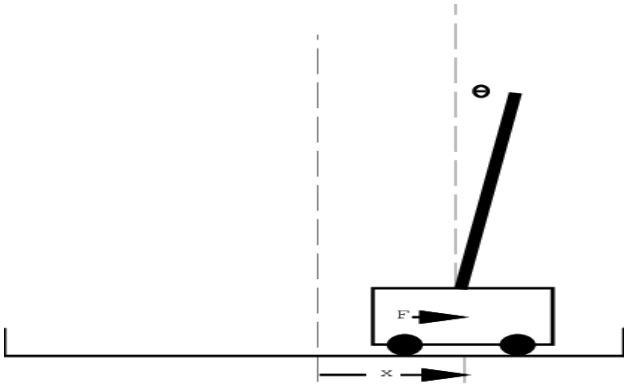
نام و نام خانوادگی	محمدحسین بدیعی
شماره دانشجویی	810199106
تاریخ ارسال گزارش	12 بهمن

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

سوال 1 – CartPole

پاسخ بخش الف)

مسأله CartPole یکی از مشهورترین مسائل یادگیری تقویتی یا همان reinforcement learning می‌باشد که گاها به آن سیستم پاندول معکوس یا سیستم آونگ معکوس هم می‌گویند.



شکل 1سیستم **CartPole**

فورموله‌سازی مسأله :

این سیستم متشکل از یک ارابه و میله‌ای متصل به آن می‌باشد و این میله یا وزنه به طور مدام به سمت چپ و راست حرکت می‌نماید و توسط نیرویی نگه‌دارنده می‌بایست این ارابه را به نحوی نگه داریم که تعادل خود را حفظ کند. (گاها ذکر می‌شود که ارابه علاوه بر اینکه می‌بایست تعادل را حفظ کند باید به گوشه‌ها نیز برخورد نکند). ما در مسأله برای زمانیکه زاویه آونگ از یک حدی بیشتر شود اصطلاحا عبارت failure را به کار می‌بریم.

حال به سراغ حل مسأله می‌رویم به صورتی که هدف تعیین شده یعنی حفظ تعادل در محدوده‌ی مشخص شده برآورده شود. در ابتدا توجه فرمایید که ما مسأله را به صورت مرحله به مرحله یا به طور مصطلح episodic در نظر گرفته و در هر مرحله اقدام به حل برای نیل به هدف مطلوب یعنی حفظ تعادل می‌نماییم.

هر مرحله متشکل از چندین step می‌باشد. در واقع هر actionای که توسط سیستم یادگیری طراحی شده انجام می‌شود، یک step به حساب می‌اید (به عنوان مثال یک step بدین‌گونه می‌تواند باشد که با اعمال یک نیرو، فلان مقدار این ارابه به سمت راست یا چپ حرکت نماید). طبیعتا هر اندازه که بیشتر این سیستم در حالت تعادل نگه‌داشته شود، stepهای مسأله بیشتر خواهد شد و نیز اگر زاویه از حد آستانه بیشتر در نظر گرفته شود و یا اینکه ارابه از محدوده‌ی مجاز موقعیتی تجاز نماید، failure رخ داده و stepها در همان مرحله‌ی failure متوقف خواهد شد. حالتی که به عنوان حالت همگرایی توصیف می‌شود این است که وقتی stepها از یک حدی بیشتر شده و در عین حال ارابه در حالت تعادل باقی می‌ماند، عملا می‌گوییم همگرایی رخ داده و از این به بعد در ادامه نیز ارابه در حالت تعادل باقی خواهد ماند و لذا stepها متوقف خواند شد و ارابه نیز از این مرحله به بعد می‌تواند حالت تعادلی خود را حذف کند (حالت تعادل ماندگار رخ می‌دهد).

حال برای حل این مسأله یادگیری تقویتی یک تابع پاداش برای عامل خود در نظر می‌گیریم. پاداش در واقع بازخود از محیط را به ازای هر action عامل به ارمغان می‌آورد. عملکرد بدین صورت است که عامل با انجام action های درست پاداش دریافت کنیم. رویکرد بدین صورت است که تابع هدف مشخص می‌کند که این action با توجه به حالات به چه میزان خوب است و agent را به انجام آن با ارائه‌ی پاداش برای step های بعدی سوق دهد یا فلان action به چه میزان بد است و از انجام آن توسط agent جلوگیری به عمل آورد. دقت بفرمایید که هیچ گاه به عامل (Agent) گفته نمی‌شود که کار درست در هر وضعیت کدام است و فقط به وسیله‌ی معیاری، به عامل فهمانده می شود که یک عمل (Action) به چه میزان خوب و یا به چه میزان بد می باشد.

لذا agent باید ادراک و حالات و پاداش‌های کسب شده را ذخیره کند. لذا به حافظه نیاز داریم.

سپس حالات سیستم را به صورت زیر در نظر می‌گیریم (هر سیستم مکانیکی مانند سیستم مسأله ما حالاتی دارد که در نتیجه‌ی اعمال نیرو به سیستم به حالات جدید تغییر پیدا می‌کنند).

- موقعیت ارابه x
- زاویه میله θ
- سرعت ارابه $\dot{x}$
- سرعت زاویه‌ای میله $\dot{\theta}$

پس حالات agent را در نظر گرفتیم. در هر step این حالات آپدیت می‌شوند تا زمانیکه همگرایی (تعادل پایدار) حاصل شود.

حال action ها یا همان عمل‌هایی که agent قادر است در محیط انجام دهد را در نظر می‌گیریم. واضح است که یا می‌تواند در اثر اعمال نیرو به راست حرکت کند یا به چپ. لذا برای actionها داریم:

- Left action
- Right action

محیط عامل در پیاده سازی توسط ابزار gym تعریف می‌کنید. توجه نمایید که محیط لازمه‌ی انجام هر action و تغییر حالات سیستم است. لذا بدون تعریف محیط دیگر action کجا انجام شود و حالات به کجا آپدیت شوند؟ لذا محیط را با این ابزار قوی فراهم می‌کنیم.

در این تمرین ذکر شده است که از policy gradient به عنوان تکنیک بهینه‌سازی پارامترها استفاده کنیم.Policy نحوه برخورد با هر عمل و نحوه تصمیم گیری در هر یک از شرایط مختلف را تعیین می‌کند. و تعیین‌کننده شیوه رفتار عامل در زمان بوده و عامل را به حالت های بهتر سوق می‌دهد.

حال مراحل را با پیاده‌سازی انجام شده دنبال می‌کنیم.

پیاده‌سازی مسأله :

ما برای پیاده‌سازی یک کلاس با نام CartPole در فایل main.py ایجاد نمودیم.

در ابتدا قسمت main را در نظر می‌گیریم و ما شاکله پیاده‌سازی را از بخش‌های initialization و تعریف یک instance از کلاس شروع کردیم. سپس مدل را طراحی کرده. سیاست یعنی نحوه برخورد با هر عمل را مشخص نموده. سپس به یادگیری و شروع شبیه‌سازی پرداخته و در نهایت امتیازها را پلات می‌نماییم.

```
85 if __name__ == "__main__":
86   model = CartPole()
87   model.constructModel()
88   model.setPolicy()
89   model.train()
90   model.plotScores()
```

کد پیاده‌سازی2 شکل

در initialization با instance گرفتن از کلاس پارامترهای کلاس را تعریف می‌نماییم.

```
 8 class CartPole(torch.nn.Module):
 9   def __init__(self, state_size=4, action_size=2, hidden_size=32):
10     super(CartPole, self).__init__()
11     self.scores = []
12     self.environment = gym.make('CartPole-v0')
13     self.environment.seed(20)
```

کد پیاده‌سازی3 شکل

همانطور که مشاهده می‌فرمایید از محیط CartPole یک instance گرفته و در متغیر environment ذخیره کردیم.

سپس مدل ساده خطی از شبکه عصبی ساختیم. دقت فرمایید که ورودی‌ها می‌بایست به تعداد حالات و خروجی به تعداد عمل‌ها باشد. توجه کنید که با توجه به اینکه باید رویکرد احتمالاتی داشته باشیم، در نورون خروجی از تابع softmax استفاده می‌کنیم.

```
19   def constructModel(self):
20     self.inputLayer = torch.nn.Linear(4, 32)
21     self.hiddenLayer = torch.nn.Linear(32, 2)
22
```

```
24     return torch.nn.functional.softmax(self.hiddenLayer(torch.nn.functional.relu(self.inputLayer(state))), dim=1)
```

شکل 4کد پیاده‌سازی

حال به سراغ یادگیری می‌رویم.

در ابتدا نرخ یادگیری را تعیین می‌کنیم و تابع بهینه‌ساز مسأله را **adam** در نظر می‌گیریم.

```
35  def train(self):
36    self.spi = deque(maxlen=100)
37    self.optimizer = torch.optim.Adam(self.policy.parameters(), lr=1e-2)
38    self.train_(episodes=5000)
39
```

شکل 5کد پیاده‌سازی

اکنون هر عامل متناسب با عملی که در محیط انجام می‌دهد می‌بایست پاداش دریافت کند. سیاست نحوه برخورد با هر عمل و نحوه تصمیم گیری در هر یک از شرایط مختلف با لحاظ الگوریتم policy gradient تعیین شده است. دقت کنید که ما چون نمیدانیم در کی به همگرایی می‌رسیم لذا پاداشی که در اثر عمل فعلی انجام می‌دهیم را نسبت به ارزشی که از تاثر عمل فعلی قرار است کسب کنیم، بیشتر می‌دانیم. لذا پارامتری را به عنوان عامل تخفیف تنظیم کرده و با استفاده از این پارامتر کسب پاداش در حال حاضر را مقتنم‌تر می‌دانیم. ابتدا به کد این قسمت توجه فرمایید.

```
50  def train_(self,episodes=1000):
51      gama=1.0;
52      for k in range(1, episodes):
53          rSum = 0;
54          loss = []
55          MMU = []
56          reward = []
57          x = self.environment.reset()
58          for t in range(1000):
59              action, log_prob = self.act(x)
60              MMU.append(log_prob)
61              x, reward_, flag, _ = self.environment.step(action)
62              reward.append(reward_)
63              if flag:
64                  self.spi.append(sum(reward))
65                  self.scores.append(sum(reward))
66                  break
67
68          discounts = [gama ** i for i in range(np.shape(reward)[0] + 1)]
69          for a,b in zip(discounts, reward):
70            rSum += a * b
71
72          for log_prob in MMU:
73              loss.append(-log_prob * rSum)
74          loss = torch.cat(loss).sum()
75          self.optimizer.zero_grad()
76          loss.backward()
77          self.optimizer.step()
78
79          if (k%50 == 0):
80            print('Avg Score of episode ', k, 'is', np.mean(self.spi))
81          if np.mean(self.spi) >= 200:
82            break
```

کد پیاده‌سازی **6** شکل

همانطور که مشاهده می‌فرمایید یک جمع پاداش را به عنوان بازده در نظر گرفته و به صورت زیر محاسبه می‌نماییم.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

لذا در مرحله t مطابق با فوق پیاده کردیم که در کد بالا مشاهده می‌فرمایید. دقت کنید که گاما بین صفر تا یک می‌بایست باشد و با قرار دادن آن مساوی یک واضح است که تاثیر discount factor را از بین می‌بریم. لذا این پارامتر را با توجه به اینکه قسد داریم چه میزان تاثیر عمل عامل در لحظه فعلی را به عنوان ارزش در آینده تعیین کنیم، انتخاب کرده که ما مقدار آن را در مسأله یک لحاظ کردیم. (می‌توانید در کد هر مقدار دیگری را برای این پارامتر لحاظ کنید.) همچنین توجه کنید که reward r_t دریافتی در زمان t می‌باشد. در نهایت به الگوریتم دستور دادیم که اگر میانگین score ها در episodeهای شبیه‌سازی از مقدار 200 فراتر رفت اعلام همگرایی و توقف step کند.

در نهایت توابعی را برای کلاس تعریف کردیم که خروجی score را بر حسب iteration ها چاپ نماید.

```
38  def plotScores(self):
39      fig = plt.figure()
40      plt.plot(np.arange(1, np.shape(self.scores)[0]+1), self.scores)
41      plt.ylabel('Score')
42      plt.xlabel('Iteration')
43      plt.title('Cart Pole Using Policy Gradient Method')
44      plt.show()
```

شکل 7کد پیاده‌سازی

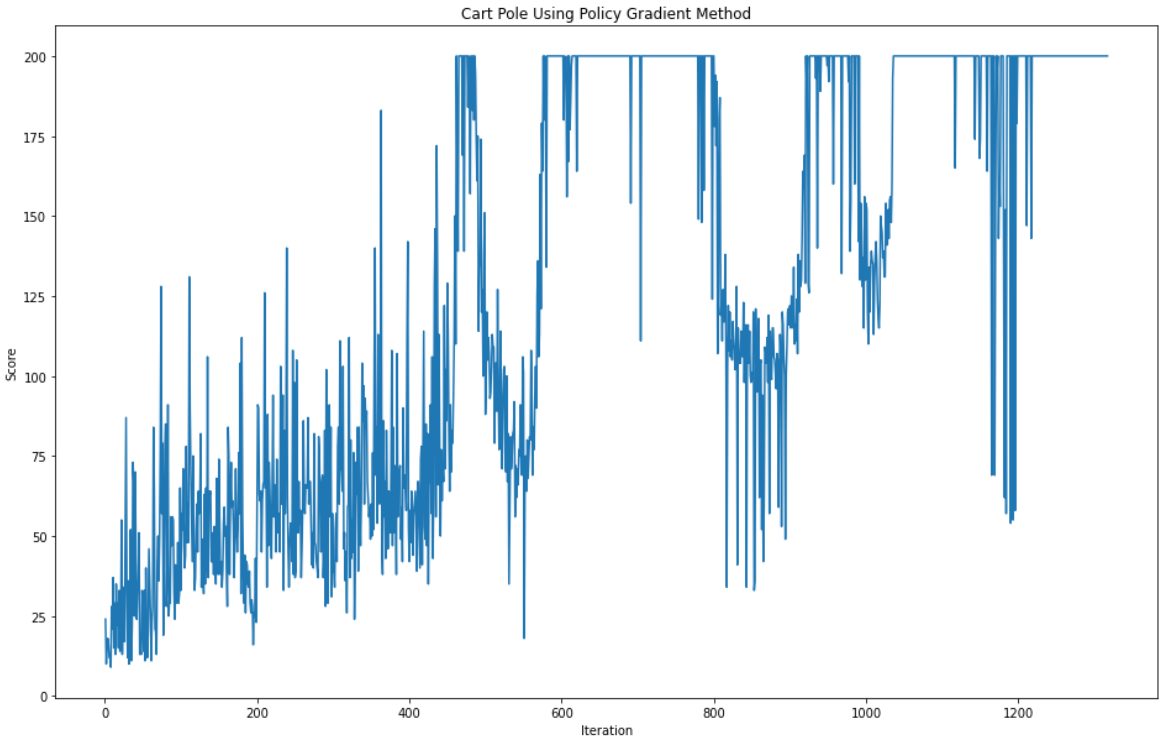
نتایج:

میانگین score ها در طول شبیه‌سازی

```
Avg Score of episode  50 is 28.66
Avg Score of episode  100 is 35.31
Avg Score of episode  150 is 48.8
Avg Score of episode  200 is 51.56
Avg Score of episode  250 is 56.42
Avg Score of episode  300 is 62.36
Avg Score of episode  350 is 61.48
Avg Score of episode  400 is 67.45
Avg Score of episode  450 is 73.02
Avg Score of episode  500 is 114.8
Avg Score of episode  550 is 121.31
Avg Score of episode  600 is 117.21
Avg Score of episode  650 is 171.49
Avg Score of episode  700 is 197.7
Avg Score of episode  750 is 198.65
Avg Score of episode  800 is 196.9
Avg Score of episode  850 is 155.91
Avg Score of episode  900 is 106.4
Avg Score of episode  950 is 131.87
Avg Score of episode  1000 is 174.81
Avg Score of episode  1050 is 169.61
Avg Score of episode  1100 is 178.39
Avg Score of episode  1150 is 198.81
Avg Score of episode  1200 is 185.25
Avg Score of episode  1250 is 185.34
Avg Score of episode  1300 is 198.9
```

شکل 8 نتایج ماینگیم امتیازات در طول شبیه سازی

و در نهایت نیز امتیازات کسب شده تا هر مرحله را پلات نمودیم.



شکل 9 امتیازات بدست آمده در طول شبیه‌سازی به ازای هر **iteration**

پاسخ بخش ب)

روش‌های موثری که برای این چالش مطرح شده که در مقالات ذکر شده بود به شرح زیر است.

- Genetic Algorithm
- Novelty Search

Novelty Search یک الگوریتم اکتشافی است که توسط novelty بودن یک رفتار هدایت می شود. الگوریتم ژنتیک نیز یک جستجوی اکتشافی است که از نظریه تکامل طبیعی چارلز داروین الهام گرفته شده است. این الگوریتم فرآیند انتخاب طبیعی را منعکس می‌کند که در آن بهترین افراد برای تولید مثل انتخاب می‌شوند تا فرزندان نسل بعدی تولید شوند.