گذارش target network با DQN

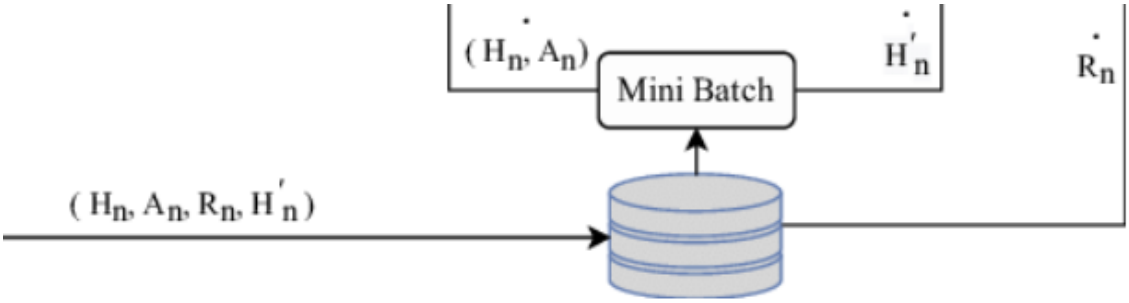
بنابراین، با فراخوانی کتابخوانه های مورد نیاز شروع کنیم سپس به توضیح هر مرحله از آن و نحوه اجرای آن می پردازم

```
import os, random
import gc
import torch
import pygame
import numpy as np
import torch.nn as nn
import gymnasium as gym
import torch.optim as optim
from collections import deque
import matplotlib.pyplot as plt
import torch.nn.functional as F
from tensorflow.python.framework import random_seed
device = torch.device("cuda")
gc.collect()
torch.cuda.empty_cache()
os.environ['CUDA_LAUNCH_BLOCKING'] = '1' # Used for debugging; CUDA related
errors shown immediately.
seed = 1
np.random.seed(seed)
np.random.default_rng(seed)
os.environ['PYTHONHASHSEED'] = str(seed)
random_seed.set_seed(seed)
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
if torch.cuda.is_available():
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
```

این کد همچنین شامل بخش های برای دیباگینگ و جلو گیری از تصادفی بود نتایج هنگام پارامتر تیونینگ هست

بخش2:

این قسمت از کد مربوط به حافظه کلاسی است که 4 عملکرد دارد و در نظرات مربوط به آن توضیح داده شده است.



که تابع init مقادیر را دریافت میکند

تابع بعدی کار ذخیره سازی تجارب را دارد

تابع Sample به صورت تصادفی به نداره بچ نمونه از تجارب برای یادگیری انتخواب میکند

و یک تابع برای شمردن تجارب موجود در حافظه

```
capacity): ,def __init__(self
```

```
done): ,reward ,next_state ,action ,state ,def store(self
```

```
batch_size): ,def sample(self
```

```
def __len__(self):
```

بخش 3:

در این قسمت کد مربوط به شبکه DQN است و وقتی نمونه آن ساخته و فراخوانی می شود و Target Network را هنگامی که یک مونه از ان ساخته میشود توضیح می دهم و هر عملکرد کلاس به وضوح توضیح داده می شود.

```
input_dim) ,num_actions ,def __init__(self
x) ,def forward(self
```

این کلاس دو متد مهم دارد اولی برای برای ایجاد شبکه و دومی برای استفاده در اموزش مورد استفاده قرار میگیرد

تویضح بیشتر کد به صورت کامنت به ان اضافه شده است

بخش 4:

می توانم بگویم DQNAgent بخش اصلی داستان است که مانند train و test در کلاس تست و امورش توابع قبلی را با هم ترکیب می کند .

این کلاس شامل توابه زیر میباشد

طبق معمول init اولین تابع هر کلاس هست که کار دریافت ابر متغیر ها را دار و همچنین کار تعریف نمونه از شبکه ها ،ریپلای مموری و اپتیمایزر را دارد

```
,epsilon_decay ,epsilon_min ,epsilon_max ,env ,def __init__(self
,learning_rate ,clip_grad_norm
memory_capacity): , discount,temperature,temperature_decay,min_temperature
```

دو تابع برای اتخواب اکشن داریم

1. E-greedy است که این روش باید یک حالت را دریافت می کند و عمل را بازگشت میدهد و اگر مدل در تکرار های اول است باید یک اکشن تصادفی را انتخاب کند اما به مرور زمان از آموخته هایش استفاده کند.

```
state): ,def select_action_g(self
```

2.این تابع مانند عملکرد قبلی و روشی برای انتخاب عمل به نام Boltzmann است

```
state): ,def select_action_b(self
```

این تابع به طور تصادفی یک عمل را انتخاب می کند اما اقداماتی که دارای Q-value بیشتر هستند خوش شانس هستند و میانگین دمای بالا مقادیر Q اهمیت زیادی ندارند. این تغییر معنی زیادی دارد:

1. شدت اکتشاف: کاوش با طمع ε در مقایسه با اکتشاف بولتزمن، ناگهانی تر و کمتر انعطاف پذیر است. اکتشاف بولتزمن امکان انتقال نرم‌تر بین اکتشاف و بهره‌برداری را فراهم می‌کند.

2. عملکرد: اکتشاف بولتزمن اغلب در محیط‌هایی ترجیح داده می‌شود که اکتشاف برای کشف خطمشی بهینه بدون گیر کردن در بهینه محلی ضروری است. می تواند منجر به همگرایی و عملکرد بهتر، به ویژه در محیط های پیچیده شود.

یکی از مهم ترین توابع این کد تابع بعدی است و به صورت زیر عمل میکند.

```
done): ,batch_size ,def learn(self
```

تجربیات نمونه گیری:

این روش با نمونه‌برداری دسته‌ای از تجربیات (حالت‌ها، اقدامات، حالت‌های بعدی، پاداش‌ها، انجام‌ها) از حافظه پخش شروع می‌شود.

ابعاد اقدامات، پاداش‌ها و انجام‌ها با استفاده از unsqueeze(1) تنظیم می‌شوند تا با شکل ورودی مورد انتظار برای عملیات‌های بعدی مطابقت داشته باشد.

Forward Pass و انتخاب Q-value:

یک گذر رو به جلو از طریق شبکه اصلی برای به دست آوردن مقادیر Q پیش بینی شده برای حالت های فعلی انجام می شود.

مقادیر Q مربوط به اقدامات انجام شده با استفاده از روش جمع آوری انتخاب می شوند.

محاسبه مقدار Q هدف:

حداکثر Q-value برای حالت های بعدی با استفاده از شبکه هدف محاسبه می شود. متن torch.no_grad() برای اطمینان از اینکه هیچ گرادیانی برای این عملیات محاسبه نمی شود استفاده می شود.

برای حالت های پایانی (که با انجام شده مشخص می شود)، مقدار Q روی صفر تنظیم می شود.

محاسبه ضرر:

مقادیر Q-هدف (y_js) با استفاده از پاداش ها و مقادیر Q-آینده با تخفیف محاسبه می شوند.

loss به عنوان تفاوت بین مقادیر Q پیش بینی شده و مقادیر Q هدف با استفاده از تابع ضرر مشخص شده (معیار خود) محاسبه می شود.

ثبت و plot برداری:

Loss در حال اجرا و شمارش های آموخته شده برای اهداف ورود به سیستم به روز می شوند.

اگر قسمت انجام شود، میانگین loss برای قسمت محاسبه شده و به تاریخچه loss اضافه می شود. از دست دادن در حال اجرا و تعداد آموخته ها سپس بازنشانی می شوند.

پس انتشار و بهینه سازی:

گرادیان ها با استفاده از self.optimizer.zero_grad() صفر می شوند.

گذر به عقب برای محاسبه گرادیان ها با توجه به ضرر انجام می شود.

برش گرادیان برای جلوگیری از انفجار شیب اعمال می شود.

مرحله بهینه سازی برای به روز رسانی پارامترهای شبکه اصلی انجام می شود.

خوب حال نوبت به 4 متد دیگر میرسد اولی برای به روز رسانی شبکه دومی و سومی برای به روز رسانی اپسیلون و دما و اخری برای دخیره کردن وزن های شبکه است

بخش 5:

در اینجا قصد دارم در مورد آموزش و تست بخشی از کد صحبت کنم

باز هم مثل همیشه ازتابع init شورع میکنم که عبارت است از مقداردهی اولیه هایپرپارامترها، تعریف محیط و Agent با پارامترهای آن

```
:(def train(self
"""
Reinforcement learning training loop.
"""
```

سپس تابع اموزش را درایم که همون طور که از اسمش پیداست وظیفه اموزش مدل را انجام میدهد در هر تکرار با توجه این این که چه سیاستی را انتخواب میکنیم یک عمل با توجه به حالت انتخواب میشود سپس عمل را درمحیط اعمال میکنیم سپس حالت بعدی پاداش و دیگر اطلاعات را از محیط دریافت میکنیم و در حافظه دخیره میکینم.

سپس اگر طول حافظه از بچ بیشتر بود انگاه عمل اموزش را انجام میدهیم و اگر زمان به روز رسانی بود وزن های شبکه هدف را به روز میکنیم بقیه کد در این تابع مربوط به بروز رسانی پرامتر ها و پرینت کردن و ploting میباشد

تابع بعدی تابع تست مدل هست که اپتدا وزن های ذخیره شده مدل را بار گذاری کرده و سپس چرخه عمل و حالت را تکرار میکند و پاداش هر تکرار را ذخیره میکند و نمایش می دهد

```
:(max_episodes ,def test(self
```

سپس یک تابع داریم برای نمایش نمودار های loss و پاداش و غیره

در بخش اخر main check را داریم که چک میکند که کد های قبلی درس کار میکنند سپس یک سری ابر متغیر دریافت میکند و انها را به کلاس اموزش و تست می دهد تا پردازش های گفته شده انجام شود

در مورد برخی از این پرامتر ها مثل دما در گذارش تمرین قبل توضیح داده شده است در این تمرین به علت بزرگ تر بودن فضای عمل و حالت نیاز به بچ سایز بزرگتری است دریک فایل جداگانه به بررسی و مقایسه دو اگوریتم میپردازم.

پایان