Research Progress-Yifu Tian

完成基于Deep Q-Network算法和highway-env仿真环境的车道变更策略的实验

Deep Q-Network

强化学习中的策略可以按照目标策略和行为策略进行分类:

- 目标策略 target policy: 智能体要学习的策略
- 行为策略 behavior policy: 智能体与环境交互的策略, 即用于生成行为的策略

Q-learning 是一种off-policy TD方法. 所谓off-policy就是指行为策略和目标策略不是同一个策略, 智能体可以通过离线学习自己或别人的策略来指导自己的行为; 与之相反, on-policy的行为策略和目标策略是同一个策略

Q-learning算法: 该算法中存在一张表格, 记录每个状态下执行每个动作所得到的Q值, 在选取动作时会进行表格的查阅, 然后选取Q值最大的动作

DQN算法: 当状态和动作为连续的, 无限的, 通过表格的方式记录就不合理了, 采用神经网络来替代表格, 输入为状态, 输出为动作

- 原始论文：Playing Atari with Deep Reinforcement Learning
- 算法流程:

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

对于每个状态, 我们会给出一个策略, 智能体会执行一个动作, 并得到相应的奖励. 我们训练的目的是使得累计的奖励最大

1. 经验池的储存: 输入状态, 输出动作, 我们将每一次的[当前状态, 动作, 奖励, 下一状态]存储进经验池, 当经验池满容量时, 进行训练, 此后有新的样本将不断替换经验池中的原有样本, 反复训练

2. 双网络：在DQN中存在两个神经网络，一个是Q-target，一个是Q-eval。其中Q-eval网络会不断进行训练和Q值的更新，而Q-target则是相对固定，只有经过固定训练轮次后才会与Q-eval同步。损失函数就是两个网络的Q值之差，我们需要使其差值越来越小。如果两个网络都在改变，很有可能会“错过”，当固定一个网络时，网络的训练会更有目标性，即更容易收敛。
3. e-greedy算法：上述我们在选择动作时，一般是选择Q值最大的那个动作。但如果有的执行动作不被选择过，它的Q值将始终是0，因此可能永远都不会被选择。所以我们引入动作选择时的随机性，使其具有一定概率去随机选择动作而不是完全依靠Q值的大小选择最优动作。

我尝试了从头开始构建DQN算法，最后代码可以跑通，但看起来小车在每一步的交互中没有为其后面的策略带来优化；后面我发现，在stable_baselines3这个库中已经集成了DQN算法，于是我直接调用再次尝试

highway-env

在用Carla进行仿真之前, 打算先用highway-env简单上手一下RL, 了解到一个自动驾驶模拟环境highway-env, 由Edouard Leurent开发和维护, 其中包含6个场景

- 高速公路"highway-v0"
- 汇入"merge-v0"
- 环岛"parking-v0"
- 十字路口"intersection-v0"
- 赛车道"racetrack-v0"
  官方文档

总共分为四个步骤

- 安装环境
- 配置环境
- 训练模型
- 总结

安装环境

问题记录

1. gym版本变更导致参数报错
2. torchvision已经安装好了但是无法import
   解决: 因为我安装了多个python版本, 所以要先check一下torchvision安装到哪个版本下然后切换到对应的版本

天近遥 2022.10.30 1

push_memory expected sequence of length 5 at dim 1 (got 4)怎么解决啊

yu2748114477 回复 天近遥 2022.11.11

解决了，改成下面的代码

while True:

done = False

start_time = time.time()

s = env.reset()

s = s[0]

while not done:

e = np.exp(-count / 300) # 随机选择action的概率，随着训练次数增多逐渐降低

a = dqn.choose_action(s, e)

s_, r, done, truncated, info = env.step(a)

env.render()

3.

4. 环境不存在
   解决: 不要直接import gym而是import gymnasium as gym

Code-v1

从头构建DQN

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as T
from torch import FloatTensor, LongTensor, ByteTensor
from collections import namedtuple
import random

# DQN网络模型基本参数
Tensor = FloatTensor

EPSILON = 0     # epsilon used for epsilon greedy approach
GAMMA = 0.9
TARGET_NETWORK_REPLACE_FREQ = 40        # target网络更新频率
MEMORY_CAPACITY = 100 # 经验库容量
BATCH_SIZE = 80 # 批量训练
```

```
LR = 0.01          # 学习率

# 实现DQN网络类
class DQNNet(nn.Module):
    def __init__(self):
        super(DQNNet,self).__init__()

        # 定义两个线性层，用于处理输入数据
        self.linear1 = nn.Linear(35,35)
        self.linear2 = nn.Linear(35,5)
    def forward(self,s):
        s=torch.FloatTensor(s)
        s = s.view(s.size(0),1,35)
        s = self.linear1(s)
        s = self.linear2(s)
        return s

# 实现DQN算法主要逻辑
class DQN(object):
    def __init__(self):
        self.net,self.target_net = DQNNet(),DQNNet()
        self.learn_step_counter = 0
        self.memory = []
        self.position = 0
        self.capacity = MEMORY_CAPACITY
        self.optimizer = torch.optim.Adam(self.net.parameters(), lr=LR)
        self.loss_func = nn.MSELoss()

    # 动作选择，遵循e-greedy算法
    def choose_action(self,s,e):
        x=np.expand_dims(s, axis=0) # 拓展数组的形状和维度，使其满足输入格式
        if np.random.uniform() < 1-e:  # 从(0, 1)范围内随机取值
            actions_value = self.net.forward(x)  # 向前传播
            action = torch.max(actions_value,-1)[1].data.numpy()
            action = action.max()     # 选取Q值最大的一个动作
        else:
            action = np.random.randint(0, 5) # 随机选取动作
        return action

    # 放入经验库
    def push_memory(self, s, a, r, s_):
        if len(self.memory) < self.capacity:
            self.memory.append(None)
```

```
        self.memory[self.position] = Transition(torch.unsqueeze(torch.FloatTensor(s),
0),torch.unsqueeze(torch.FloatTensor(s_), 0),\

torch.from_numpy(np.array([a])),torch.from_numpy(np.array([r],dtype='float32')))#
        self.position = (self.position + 1) % self.capacity
        # 如果有超过经验库容量的样本，则从头开始替换经验库里的样本
    # 随机选取batch个样本进行训练
    def get_sample(self,batch_size):
        sample = random.sample(self.memory,batch_size)
        return sample

    # 训练
    def learn(self):
        if self.learn_step_counter % TARGET_NETWORK_REPLACE_FREQ == 0:
            self.target_net.load_state_dict(self.net.state_dict())
        self.learn_step_counter += 1

        transitions = self.get_sample(BATCH_SIZE)
        batch = Transition(*zip(*transitions))

        b_s = Variable(torch.cat(batch.state))
        b_s_ = Variable(torch.cat(batch.next_state))
        b_a = Variable(torch.cat(batch.action))
        b_r = Variable(torch.cat(batch.reward))

        q_eval =
self.net.forward(b_s).squeeze(1).gather(1,b_a.unsqueeze(1).to(torch.int64))
        q_next = self.target_net.forward(b_s_).detach() #
        q_target = b_r + GAMMA * q_next.squeeze(1).max(1)[0].view(BATCH_SIZE, 1).t()
        loss = self.loss_func(q_eval, q_target.t())
        self.optimizer.zero_grad() # reset the gradient to zero
        loss.backward()
        self.optimizer.step() # execute back propagation for one step
        return loss
Transition = namedtuple('Transition',('state', 'next_state','action', 'reward'))


import gymnasium as gym
import highway_env
from matplotlib import pyplot as plt
import numpy as np
import time
import os
```

```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'TRUE'

# 配置环境参数
config = \
    {
    "observation":
         {
        "type": "Kinematics",
        "vehicles_count": 5,
        "features": ["presence", "x", "y", "vx", "vy", "cos_h", "sin_h"],
        "features_range":
            {
            "x": [-100, 100],
            "y": [-100, 100],
            "vx": [-20, 20],
            "vy": [-20, 20]
            },
        "absolute": False,
        "order": "sorted"
        },
    "simulation_frequency": 8,  # [Hz]
    "policy_frequency": 2,  # [Hz]
    }

env = gym.make("highway-v0")
env.configure(config)

# 初始化DQN对象，设置count用于追踪训练过程
dqn=DQN()
count=0

reward=[]
avg_reward=0
all_reward=[]

time_=[]
all_time=[]

collision_his=[]
all_collision=[]

'''
训练循环
```

```
- reset环境并开始新的回合
- 每个回合中，根据当前状态选择动作，执行动作并接收新状态和奖励
- 存储转换到记忆库
- 每当记忆库中有足够的数据时，执行学习步骤来更新网络
- 记录并可视化训练过程中的奖励，时间和碰撞率
'''
while True:
    done = False
    start_time=time.time()
    s = env.reset()[0]

    while not done:
        e = np.exp(-count/300)  #随机选择action的概率，随着训练次数增多逐渐降低
        a = dqn.choose_action(s,e)
        s_, r, done, truncated, info = env.step(a)
        env.render()

        dqn.push_memory(s, a, r, s_)

        if ((dqn.position !=0)&(dqn.position % 99==0)):
            loss_=dqn.learn()
            count+=1
            print('trained times:',count)
            # 每训练40次统计一次平均值
            if (count%40==0):
                avg_reward=np.mean(reward)
                avg_time=np.mean(time_)
                collision_rate=np.mean(collision_his)

                all_reward.append(avg_reward)
                all_time.append(avg_time)
                all_collision.append(collision_rate)

                plt.plot(all_reward)
                plt.show()
                plt.plot(all_time)
                plt.show()
                plt.plot(all_collision)
                plt.show()

                reward=[]
                time_=[]
```

```
            collision_his=[]

        s = s_
        reward.append(r)

    end_time=time.time()
    episode_time=end_time-start_time
    time_.append(episode_time)

    is_collision=1 if info['crashed']==True else 0
    collision_his.append(is_collision)
```

把代码跑通了, 但得到的训练结果不理想(平均碰撞率恒为1), 并且env.render()这个函数并没有显示画面.

解决:
render_mode="rbg_array"

```
125    highway_env.register_highway_envs()
126    env = gym.make("highway-v0", render_mode="rgb_array")
127    env.configure(config)
```

然后我看这个小车一直撞车, 感觉它并没有从之前的经验中学习到(也有可能是训练次数的问题), 于是我想先看看它的奖励函数是什么样的

> We generally focus on two features: a vehicle should
>
> - progress quickly on the road;
> - avoid collisions.
>
> Thus, the reward function is often composed of a velocity term and a collision term:
>
> $$R(s,a) = a\frac{v - v_{\min}}{v_{\max} - v_{\min}} - b\,\text{collision}$$
>
> where v, $v_{\min}$, $v_{\max}$ are the current, minimum and maximum speed of the ego-vehicle respectively, and a, b are two coefficients.

没啥头绪...只能看着它一遍又一遍地跑, 但不知道怎么改. 一边等一边找些资料来看

Code-v2

参考了一下其他代码
[基于DQN强化学习的高速路决策控制 - Colin.Fang的文章 - 知乎]

```python
import gymnasium as gym
import highway_env
from stable_baselines3 import DQN


# Create environment
env = gym.make("highway-fast-v0")

model = DQN('MlpPolicy',
            env,
            policy_kwargs=dict(net_arch=[256, 256]),
            learning_rate=5e-4,
            buffer_size=15000,
            learning_starts=200,
            batch_size=32,
            gamma=0.8,
            train_freq=1,
            gradient_steps=1,
            target_update_interval=50,
            verbose=1,
            tensorboard_log="./logs")

model.learn(int(2e4))
model.save("highway_dqn_model")
```

```
|    exploration_rate | 0.05     |
| time/               |          |
|    episodes         | 808      |
|    fps              | 3        |
|    time_elapsed     | 4923     |
|    total_timesteps  | 17539    |
| train/              |          |
|    learning_rate    | 0.0005   |
|    loss             | 0.109    |
|    n_updates        | 17338    |
----------------------------------
----------------------------------
| rollout/            |          |
|    ep_len_mean      | 26.7     |
|    ep_rew_mean      | 21       |
|    exploration_rate | 0.05     |
| time/               |          |
|    episodes         | 812      |
|    fps              | 3        |
|    time_elapsed     | 4979     |
|    total_timesteps  | 17642    |
| train/              |          |
|    learning_rate    | 0.0005   |
|    loss             | 0.0245   |
|    n_updates        | 17441    |
----------------------------------
----------------------------------
| rollout/            |          |
|    ep_len_mean      | 26.7     |
|    ep_rew_mean      | 21       |
|    exploration_rate | 0.05     |
| time/               |          |
|    episodes         | 816      |
|    fps              | 3        |
|    time_elapsed     | 5013     |
|    total_timesteps  | 17762    |
| train/              |          |
|    learning_rate    | 0.0005   |
|    loss             | 0.0635   |
|    n_updates        | 17561    |
----------------------------------
```

highway_dqn_test.py > ...

```python
import gymnasium as gym
import highway_env
from stable_baselines3 import DQN
from stable_baselines3.common.evaluation import evaluate_policy


# Create environment
env = gym.make("highway-fast-v0", render_mode="rgb_array")

# load model
model = DQN.load("highway_dqn_model", env=env)

mean_reward, std_reward = evaluate_policy(
    model,
    model.get_env(),
    deterministic=True,
    render=True,
    n_eval_episodes=10)

env.render()
print(mean_reward)
```

这个代码只给出了训练100次得到的模型结果，并且模型性能不算好，输出不够直观

Code-v3

自己改写了一下代码，调整了一些参数，加了一些输出数据和图像的代码进去，方便实验记录，最后的完整代码整合到这个压缩包里

DQN_highway.zip
模型训练好之后就是评估模型
以下是一些关键的评估指标

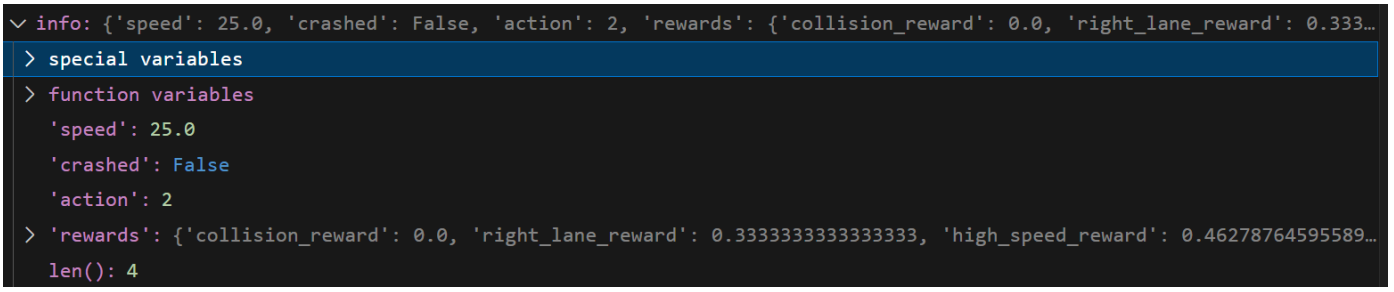
1. 累积奖励
   计算在单次回合或多个回合中模型获得的总奖励. 一个高效的变道模型应该能够在保持高速的同时避免碰撞，因此累积奖励会较高。
2. 平均奖励
   将累积奖励除以回合数或时间步数，得到平均奖励。这反映了模型在每个时间步骤或每回合的平均表现。
3. 成功率
4. 碰撞率

代码补充好之后出现了几个问题

1. 我希望先渲染, 最后再绘制指标的图像
2. 碰撞率和成功率的图像为空白, 这可能与env_info的调用有关
3. 模块化整个程序, 最后在一个主程序中将其组织起来

第一个和第三个问题解决了
现在看看碰撞率和成功率的图像问题

找到info的相关信息和参数



info是一个长度为1的list型, 里面含有一个dict, 其中'crashed'键对应的值为boolean型, 发生碰撞时为True, 无碰撞时为False. 于是加了个这个代码

```
# # 检查是否碰撞或成功完成任务
# if info[0].get('crashed') == True:
#     collisions += 1
# else: successes += 1
```

上述代码是错的, 应该注意: 在每个while循环中, 只有当整个循环内小车都没有发生碰撞, 才让successes+=1, 正确代码实现如下:

```python
def evaluate_model(model, env, n_eval_episodes=10):

    episode_rewards = []
    collisions = 0
    successes = 0

    for episode in range(n_eval_episodes):
        obs = env.reset()
        done = False
        total_rewards = 0

        flag = 0
        while not done:
            action, _states = model.predict(obs, deterministic = True)
            obs, reward, done, info = env.step(action)
            total_rewards += reward

            # 检查是否碰撞或成功完成任务
            if info[0].get('crashed') == True:
                collisions += 1
                flag = 1
                # break
        # 只有当整个过程中都不发生碰撞时，即flag恒为1，才把successes+1
        if flag == 0: successes += 1
        else: flag = 0

        episode_rewards.append(total_rewards)

    collision_rate = collisions / n_eval_episodes
    success_rate = successes / n_eval_episodes

    return collision_rate, success_rate, episode_rewards
```

实验结果记录与模型评估

我一共做了四组实验（不算完整），分别是训练100次，200次，500次和1000次得到的模型性能比较，把验证模型的步骤设为10回合，其他参数也同样保证一致。（这只是一个初步的实验，后续可以补充更详细的实验结果）

训练100次得到的模型结果表现一般，模型不稳定；训练200次得到的结果表现最优，大部分都是0.9或1.0的成功率；500次和1000次得到的模型最差，碰撞率在90%情况下都为1.0

记录表格如下：
DQN-highway-results-record.xlsx

总结

这一块内容给我最大的感受就是RL训练出的模型的不稳定性，因为自动驾驶永远不可能停留于纸面分析，哪怕碰撞率是0.1也不能接受。再者就是调参的技巧，后面接触到各种算法（比如PPO算法）也是需要的

Carla仿真平台的环境配置

这一块其实遇到了一些问题，一是不确定是用windows还是ubuntu作为开发环境，了解到linux可以提供ros与carla的一些协同，所以后续的安排是装个双系统然后把Carla的环境配置好；二是Carla比较吃电脑的配置，刚好我也在考虑换个电脑，所以等待了一段时间才开始着手搭建Carla（之前用的highway-env环境简单实现了基于DQN算法的车道变更策略）

Reference

stable-baseline3
基于DQN强化学习的高速路决策控制 - Colin.Fang的文章 - 知乎
基于PPO自定义highway-env场景的车辆换道决策 - Colin.Fang的文章 - 知乎

DQN算法实现注意事项及排错方法

DQN实现高速超车（复现 deeptraffic:MIT 6.S094: Deep Learning for Self-Driving Cars）
深度强化学习训练与调参技巧
Carla教程