**程序报告**

学号：2013536 姓名：汤清云

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

1. 使用深度优先算法思想完成路径搜索，代码模仿广度优先编写。
2. 学习强化学习算法思想，Qlearning算法，靠奖励来影响机器人的动作选择。
3. 综合DQLearning算法思想，修改参数，提高机器人动作选择的正确率。
4. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

采用方法：DQlearning

问题：不能保证百分百通过所有测试用例。

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

深度优先算法实现：

import numpy as np

# 机器人移动方向

move\_map = {

'u': (-1, 0), # up

'r': (0, +1), # right

'd': (+1, 0), # down

'l': (0, -1), # left

}

def my\_search(maze):

"""

任选深度优先搜索算法、最佳优先搜索（A\*)算法实现其中一种

:param maze: 迷宫对象

:return :到达目标点的路径 如：["u","u","r",...]

"""

h, w, \_ = maze.maze\_data.shape

visited = np.zeros((h, w), dtype=np.int32)

path = []

path1 = []

def dfs(x, y):

nonlocal path1

nonlocal path

if visited[x][y] == 1:

return False

if x == h - 1 and y == w - 1:

return True

visited[x][y] = 1

actions = maze.can\_move\_actions((x, y))

for action in actions:

temp = (x + move\_map[action][0], y + move\_map[action][1])

path1.append(action)

if dfs(temp[0], temp[1]):

path = path1.copy()

path1 = path1[:-1]

visited[x][y] = 0

# -----------------请实现你的算法代码--------------------------------------

dfs(0, 0)

# -----------------------------------------------------------------------

return path

测试代码：

from QRobot import QRobot

from ReplayDataSet import ReplayDataSet

from torch\_py.QNetwork import QNetwork

import numpy as np

import random

import torch

import torch.nn.functional as F

from torch import optim

class Robot(QRobot):

valid\_action = ['u', 'r', 'd', 'l']

target\_model = None

eval\_model = None

batch\_size = 64

learning\_rate = 1e-3

step = 1 # 记录训练的步数

EveryUpdate = 64 # the interval of target model's updating

device = torch.device("cuda:0") if torch.cuda.is\_available() else torch.device("cpu")

def \_\_init\_\_(self, maze):

"""

初始化 Robot 类

:param maze:迷宫对象

"""

super(Robot, self).\_\_init\_\_(maze)

self.maze = maze

self.epsilon0 = 0.99

self.epsilon = self.epsilon0

self.gamma = 0.90

self.maze\_size = maze.maze\_size

self.table = {}

self.state = None

self.step = 0

maze.set\_reward(reward={

"hit\_wall": -10.,

"destination": 50.,

"default": -1.,

})

"""创建网络"""

self.target\_model = None

self.eval\_model = None

self.learning = True

self.testing = True

self.\_build\_network()

self.target\_replace\_op()

self.target\_model.eval()

"""申请一段内存来存储数据"""

max\_size = min(self.maze\_size \*\* 2 \* 3, 1e4)

self.memory = ReplayDataSet(max\_size=max\_size)

# self.memory.build\_full\_view(maze=maze)

if self.maze\_size == 11:

self.epsilon = 0.3

self.epsilon0 = 0.3

runner = Runner(self)

runner.run\_training(50, int(self.maze\_size \*\* 2 \* 1.5))

def set\_status(self, learning=True, testing=False):

self.learning = learning

self.testing = testing

def update\_parameter(self):

if self.testing:

self.epsilon = 0.0

else:

self.step += 1

self.epsilon = self.epsilon0 / self.step

return self.epsilon

def update(self):

self.state = self.sense\_state()

self.table.setdefault(self.state, {a: 0.0 for a in self.valid\_action})

action = self.\_choose\_action(self.state)

reward = self.maze.move\_robot(action)

next\_state = self.sense\_state()

self.table.setdefault(next\_state, {a: 0.0 for a in self.valid\_action})

if self.learning and not self.testing:

if self.learning:

self.table[self.state][action] += self.alpha \* (

reward + self.gamma \* float(max(self.table[next\_state].values())) - self.table[self.state][

action])

self.update\_parameter() # update parameters

return action, reward

def \_build\_network(self):

seed = random.randint(1, 100)

# seed = 1

random.seed(seed)

#建立模型

self.target\_model = QNetwork(state\_size=2, action\_size=4, seed=seed).to(self.device)

#构建eval模型

self.eval\_model = QNetwork(state\_size=2, action\_size=4, seed=seed).to(self.device)

#优化器

self.optimizer = optim.Adam(self.eval\_model.parameters(), lr=self.learning\_rate)

def target\_replace\_op(self):

#更新参数

# for target\_param, eval\_param in zip(self.target\_model.parameters(), self.eval\_model.parameters()):

# target\_param.data.copy\_(self.TAU \* eval\_param.data + (1.0 - self.TAU) \* target\_param.data)

self.target\_model.load\_state\_dict(self.eval\_model.state\_dict())

def \_choose\_action(self, state):

if self.learning:

if random.random() < self.epsilon:

return random.choice(self.valid\_action)

else:

return max(self.table[self.state], key=self.table[self.state].get)

elif self.testing:

return max(self.table[self.state], key=self.table[self.state].get)

else:

return random.choice(self.valid\_action)

state = np.array(state)

state = torch.from\_numpy(state).float().to(self.device)

if random.random() < self.epsilon:

action = random.choice(self.valid\_action)

else:

self.eval\_model.eval()

with torch.no\_grad():

q\_next = self.eval\_model(state).cpu().data.numpy() # use target model choose action

self.eval\_model.train()

action = self.valid\_action[np.argmax(q\_next).item()]

return action

def \_learn(self, batch: int = 16):

if len(self.memory) < batch:

print("the memory data is not enough")#当前超出内存，结束学习

return

state, action\_index, reward, next\_state, is\_terminal = self.memory.random\_sample(batch)

#将数据转换为张量类型

state = torch.from\_numpy(state).float().to(self.device)

action\_index = torch.from\_numpy(action\_index).long().to(self.device)

reward = torch.from\_numpy(reward).float().to(self.device)

next\_state = torch.from\_numpy(next\_state).float().to(self.device)

is\_terminal = torch.from\_numpy(is\_terminal).int().to(self.device)

self.eval\_model.train()

self.target\_model.eval()

Q\_targets\_next = self.target\_model(next\_state).detach().max(1)[0].unsqueeze(1)

#获得最大预测值

Q\_targets = reward + self.gamma \* Q\_targets\_next \* (torch.ones\_like(is\_terminal) - is\_terminal)

#获得期望值

self.optimizer.zero\_grad()

Q\_expected = self.eval\_model(state).gather(dim=1, index=action\_index)

#计算丢失

loss = F.mse\_loss(Q\_expected, Q\_targets)

print(self.step, loss)

loss\_item = loss.item()

#最小化

self.optimizer.zero\_grad()

loss.backward()

self.optimizer.step()

#将训练模型赋值给测试模型

# self.target\_replace\_op()

return loss\_item

def train\_update(self):

"""

以训练状态选择动作并更新Deep Q network的相关参数

:return :action, reward 如："u", -1

"""

action, reward = "u", -1.0

# -----------------请实现你的算法代码--------------------------------------

self.set\_status(True, False)

return self.update()

state = self.sense\_state()

action = self.\_choose\_action(state)

reward = self.maze.move\_robot(action)

next\_state = self.sense\_state()

is\_terminal = 1 if next\_state == self.maze.destination or next\_state == state else 0

self.memory.add(state, self.valid\_action.index(action), reward, next\_state, is\_terminal)

batch\_size = min(32, (self.maze\_size-1)\*\*2)

self.\_learn(batch=batch\_size)

"""--间隔一段时间更新target network权重--"""

if self.step % self.EveryUpdate == 0:

# self.\_learn(batch=16)

self.target\_replace\_op()

# self.epsilon = max(0.01, self.epsilon - 0.02)

"""---update the step and epsilon---"""

self.step += 1

delta = 0

if self.maze\_size == 3:

delta = 0.0015

elif self.maze\_size == 5:

delta = 0.00075

else:

delta = 0.000375

self.epsilon = max(0., self.epsilon - delta)

print(self.epsilon)

# -----------------------------------------------------------------------

return action, reward

def test\_update(self):

action, reward = "u", -1.0

# -----------------请实现你的算法代码--------------------------------------

self.set\_status(False, True)

return self.update()

state = np.array(self.sense\_state(), dtype=np.int16)

state = torch.from\_numpy(state).float().to(self.device)

self.eval\_model.eval()

with torch.no\_grad():

q\_value = self.eval\_model(state).cpu().data.numpy()

action = self.valid\_action[np.argmax(q\_value).item()]

reward = self.maze.move\_robot(action)

# -----------------------------------------------------------------------

return action, reward

1. **实验结果**

（实验结果，必填）

====================================================================



1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

达到目标预期。

困难：将所学的DQlearning转化为python代码较难，作业部分借鉴学习csdn代码