**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по практической работе №1**

**по дисциплине «Обучение с подкреплением»**

**Тема: Реализация DQN для среды CartPole-v1**

| Студент гр. 0306 |  | Кумаритов А.О. |
| --- | --- | --- |
| Преподаватель |  | Глазунов. С.А. |

Санкт-Петербург

2025

**Задание:**

Реализовать DQN для среды CartPole-v1.

**Задания для эксперимента:**

Измените архитектуру нейросети (например, добавьте слои).

Попробуйте разные значения gamma и epsilon\_decay.

Проведите исследование как изначальное значение epsilon влияет на скорость обучения

**Описание среды:**

Action space состоит из числа, принимающего два значения:

0 - движение каретки налево

1 - движение каретки направо

Observation space состоит из 3 чисел:

Cart position - значения от -4.8 до 4.8

Cart velocity - значения от минус бесконечности до плюс бесконечности

Pole angle - от -0.418 rad (-24 градуса) до 0.418 rad (24 градуса)

Pole angular velocity - значения от минус бесконечности до плюс бесконечности

Rewards +1 за каждый шаг, включая терминальный.

Starting state - всем переменным из observation space присваивается значение от -0.05 до 0.05.

Терминальный шаг наступает в трёх случаях:

Pole angle меньше -12 градусов или больше 12 градусов

Cart position меньше -2.4 или больше 2.4

Номер эпизода больше 500.

**Описание алгоритма:**

Базовое описание алгоритма представлено на рисунке 1.

|  |
| --- |
| Рис. 1 - алгоритм Deep Q-learning (DQN) |

В реализации следующие параметры:

BATCH\_SIZE - количество переходов, которые выбираются из памяти

GAMMA - коэффициент дисконтирования

EPS\_START - начальное значение эпсилон

EPS\_END - конечное значение эпсилон

EPS\_DECAY - скорость экспоненциального затухания, чем выше, тем медленнее затухание

TAU - скорость обновления целевой сети

LR - скорость обучения оптимизатора

num\_episodes - количество эпизодов

В реализации используем два класса:

Transition - именованный кортеж, хранящий переход в среде: соответствие состоянию и действию к следующему состоянию и награде.

ReplayMemory - буфер, хранящий в себе ограниченное количество наблюдаемых при взаимодействии со средой переходов. В нём реализован метод sample для выбора случайных BATCH\_SIZE элементов.

Под Q-функцией, выбирающей для агента лучшее действие, используется конфигурация нейронной сети, представленная на рисунке 2:

|  |
| --- |
| Рис. 2 - конфигурация нейронной сети |

При реализации алгоритма использовался ресурс <https://docs.pytorch.org/tutorials/intermediate/reinforcement_q_learning.html>

**Выполнение экспериментов.**

**Изменение архитектуры нейросети:**

В рамках эксперимента по сравнению влияния различной архитектуры было три запуска с hidden\_size у нейронной сети равной 64, 128, 256. Этот параметр определяет ёмкость нейронной сети. Результат представлен на рисунке 3:

|  |
| --- |
| Рис. 3 - влияние различной архитектуры сети (hidden\_size) |

Лучший результат обучения показал размер скрытого слоя нейронной сети равный 256. Обучение происходило к приблизительно 400-му эпизоду. Из-за ограниченной емкости нейросети с размером скрытого слоя 64 был получен худший результат для аппроксимации Q-функции. Среднее значение в 128 показало средний результат, обучение которого происходило в районе 575 эпизода. В данных условиях использование большего количества скрытых слоев улучшает процесс обучения.

**Разные значения gamma:**

В рамках эксперимента по сравнению влияния различных значений gamma было три запуска с gamma равной 0.8, 0.9, 0.99. Этот гиперпараметр, коэффициент дисконтирования, отвечает за то, насколько агент учитывает будущие или немедленные награды. Если значение ближе к 1, то агент оптимизирует сумму наград, если ближе к 0, то он учитывает только немедленные награды. Результат представлен на рисунке 4:

|  |
| --- |
| Рис. 4 - влияние различных значений gamma |

Лучший результат обучения показало значение gamma равное 0.99. Среднее значение в 0.9 показало худший, но стабильный результат по длительности эпизода: наблюдался рост производительности к 200 эпизоду. Запуск с gamma равной 0.9 показал средний и нестабильный результат: достигнув максимума продолжительности к 425 эпизоду, далее продолжительность начала снижаться. Это говорит о важности оптимизации будущих наград в данной среде.

**Разные значения epsilon\_decay:**

В рамках эксперимента по сравнению влияния различных значений eps\_decay было три запуска с eps\_decay равной 250, 500, 750. Этот гиперпараметр отвечает за то, насколько быстро уменьшается значение от eps\_start до eps\_end. Чем больше значение, тем дольше происходит процесс уменьшения. Результат представлен на рисунке 5:

|  |
| --- |
| Рис. 5 - влияние различных значений eps\_decay |

Лучший результат обучения показало среднее значение eps\_decay равное 750. Обучение происходило приблизительно после 350-го эпизода. Следующим по скорости обучения был запуск со значением eps\_decay равным 250: обучение происходило к 500 эпизоду, а запуск со значением eps\_decay равных 500 показали худший результат: обучение происходило к 575 эпизоду.

**Влияние изначального значения epsilon на скорость обучения:**

В рамках эксперимента по сравнению влияния различных значений eps\_start было три запуска с eps\_start равной 0.25, 0.5, 0.75. Этот гиперпараметр определяет начальное значение вероятности случайного действия в начале обучения. Результат представлен на рисунке 6:

|  |
| --- |
| Рис. 6 - влияние различных значений eps\_start |

Лучший результат обучения показало среднее значение eps\_start равное 0.5. Обучение происходило приблизительно к 400-му эпизоду. Два других значения eps\_start в 0.25 и 0.75 показали схожий нестабильный результат. Обучение происходило приблизительно после 850-го эпизода. Результат показывает важность баланса между исследованием и использованием в данной среде.

**Выводы.**

Был реализован DQN для среды CartPole-v1. Были проведены исследования при различных значениях hidden\_size, gamma, eps\_decay, eps\_start. Оптимальным для высоких результатов обучения, исходя из полученных данных, является запуск с hidden\_size = 256, gamma = 0.99, eps\_decay = 750, eps\_start = 0.5.

**ПРИЛОЖЕНИЕ А.**

**ИСХОДНЫЙ КОД ПРОГРАММЫ.**

Исходный код main.py

import gymnasium as gym

import math

import matplotlib.pyplot as plt

import numpy as np

import random

import torch

from itertools import count

from torch import nn

from torch import optim

from dqn import DQN

from replay\_memory import ReplayMemory

from transition import Transition

BATCH\_SIZE = 128

GAMMA = 0.99

EPS\_START = 0.5

EPS\_END = 0.05

EPS\_DECAY = 500

TAU = 0.005

LR = 1e-4

num\_episodes = 1000

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

env = gym.make("CartPole-v1")

def optimize\_model(memory, policy\_net, target\_net, optimizer, gamma):

if len(memory) < BATCH\_SIZE:

return

transitions = memory.sample(BATCH\_SIZE)

batch = Transition(\*zip(\*transitions))

non\_final\_mask = torch.tensor(tuple(map(lambda s: s is not None,

batch.next\_state)), device=device, dtype=torch.bool)

non\_final\_next\_states = torch.cat([s for s in batch.next\_state

if s is not None])

state\_batch = torch.cat(batch.state)

action\_batch = torch.cat(batch.action)

reward\_batch = torch.cat(batch.reward)

state\_action\_values = policy\_net(state\_batch).gather(1, action\_batch)

next\_state\_values = torch.zeros(BATCH\_SIZE, device=device)

with torch.no\_grad():

next\_state\_values[non\_final\_mask] = target\_net(non\_final\_next\_states).max(1).values

expected\_state\_action\_values = (next\_state\_values \* gamma) + reward\_batch

criterion = nn.SmoothL1Loss()

loss = criterion(state\_action\_values, expected\_state\_action\_values.unsqueeze(1))

optimizer.zero\_grad()

loss.backward()

torch.nn.utils.clip\_grad\_value\_(policy\_net.parameters(), 100)

optimizer.step()

def select\_action(state, policy\_net, steps\_done, eps\_start, eps\_decay):

sample = random.random()

eps\_threshold = EPS\_END + (eps\_start - EPS\_END) \* \

math.exp(-1. \* steps\_done / eps\_decay)

steps\_done += 1

if sample > eps\_threshold:

with torch.no\_grad():

return policy\_net(state).max(1).indices.view(1, 1), steps\_done

else:

return torch.tensor([[env.action\_space.sample()]], device=device, dtype=torch.long), steps\_done

def train\_dqn(hidden\_size=256, gamma=GAMMA, eps\_start=EPS\_START, eps\_decay=EPS\_DECAY):

steps\_done = 0

n\_actions = env.action\_space.n

state, \_ = env.reset()

n\_observations = len(state)

policy\_net = DQN(n\_observations, n\_actions, hidden\_size).to(device)

target\_net = DQN(n\_observations, n\_actions, hidden\_size).to(device)

target\_net.load\_state\_dict(policy\_net.state\_dict())

optimizer = optim.AdamW(policy\_net.parameters(), lr=LR, amsgrad=True)

memory = ReplayMemory(10000)

episode\_durations = []

for \_ in range(num\_episodes):

state, \_ = env.reset()

state = torch.tensor(state, dtype=torch.float32, device=device).unsqueeze(0)

for t in count():

action, steps\_done = select\_action(state, policy\_net, steps\_done, eps\_start, eps\_decay)

observation, reward, terminated, truncated, \_ = env.step(action.item())

reward = torch.tensor([reward], device=device)

done = terminated or truncated

if terminated:

next\_state = None

else:

next\_state = torch.tensor(observation, dtype=torch.float32, device=device).unsqueeze(0)

memory.push(state, action, next\_state, reward)

state = next\_state

optimize\_model(memory, policy\_net, target\_net, optimizer, gamma)

target\_net\_state\_dict = target\_net.state\_dict()

policy\_net\_state\_dict = policy\_net.state\_dict()

for key in policy\_net\_state\_dict:

target\_net\_state\_dict[key] = policy\_net\_state\_dict[key] \* TAU + target\_net\_state\_dict[key] \* (1 - TAU)

target\_net.load\_state\_dict(target\_net\_state\_dict)

if done:

episode\_durations.append(t + 1)

break

return episode\_durations

def run\_and\_plot(param\_values, param\_name, train\_kwargs, filename\_prefix):

plt.figure(figsize=(10, 6))

color\_cycle = plt.rcParams['axes.prop\_cycle'].by\_key()['color']

for i, val in enumerate(param\_values):

print(f"Обучение при {param\_name} = {val}")

steps\_done = 0

kwargs = train\_kwargs(val)

durations = train\_dqn(\*\*kwargs)

color = color\_cycle[i % len(color\_cycle)]

plt.plot(range(len(durations)), durations, linestyle='--', alpha=0.3,

color=color, label=f"{param\_name}={val} (raw)")

smoothed = np.convolve(durations, np.ones(10) / 10, mode='valid')

plt.plot(range(len(smoothed)), smoothed, linestyle='-', color=color,

label=f"{param\_name}={val} (smoothed)")

plt.title(f"DQN: сравнение по параметру {param\_name}")

plt.xlabel("Эпизод")

plt.ylabel("Длительность")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.savefig(f"{filename\_prefix}\_{param\_name}.png")

def different\_hidden\_size():

hidden\_sizes = [64, 128, 256]

run\_and\_plot(hidden\_sizes, "hidden\_size", lambda hz: {"hidden\_size": hz}, "results")

def different\_gamma():

gammas = [0.8, 0.9, 0.99]

run\_and\_plot(gammas, "gamma", lambda g: {"gamma": g}, "results")

def different\_epsilon\_decay():

decays = [250, 500, 750]

run\_and\_plot(decays, "eps\_decay", lambda d: {"eps\_decay": d}, "results")

def different\_epsilon\_start():

starts = [0.25, 0.5, 0.75]

run\_and\_plot(starts, "eps\_start", lambda s: {"eps\_start": s}, "results")

def main(seed=42):

torch.manual\_seed(seed)

random.seed(seed)

env.reset(seed=seed)

np.random.seed(seed)

different\_hidden\_size()

different\_gamma()

different\_epsilon\_decay()

different\_epsilon\_start()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Исходный код dqn.py

from torch import nn

class DQN(nn.Module):

def \_\_init\_\_(self, n\_observations, n\_actions, hidden\_size):

super(DQN, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(n\_observations, hidden\_size),

nn.ReLU(),

nn.Linear(hidden\_size, hidden\_size // 2),

nn.ReLU(),

nn.Linear(hidden\_size // 2, n\_actions)

)

def forward(self, x):

return self.model(x)

Исходный код replay\_memory.py

import random

from collections import deque

from transition import Transition

class ReplayMemory(object):

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

self.memory = deque([], maxlen=capacity)

def push(self, \*args):

self.memory.append(Transition(\*args))

def sample(self, batch\_size):

return random.sample(self.memory, batch\_size)

def \_\_len\_\_(self):

return len(self.memory)

Исходный код transition.py

from collections import namedtuple

Transition = namedtuple('Transition',

('state', 'action', 'next\_state', 'reward'))