Summer of Code 2023

Report
May - July 2023

Breakout Genius: Using RL to Build an AI Game Master

Playing Atari with Deep RL

Khushi Gondane
Roll no.: 210050056

CONTENTS

INTRODUCTION

The assignment aims to implement Deep Q-Network (DQN) to combine Q-learning and deep neural networks for playing Atari games. It includes the concepts of deep reinforcement learning, i.e. experience replay, target networks, and the use of convolutional neural networks for function approximation.

Implementation

1. **Neural Network:**

   The neural network had the following layers:

   1. A convolutional layer with in_channels=4 (representing stacked frames), out_channels=16, kernel_size=8, and stride=4, followed by a ReLU activation function.
   2. Another convolutional layer with in_channels=16, out_channels=32, kernel_size=4, and stride=2, followed by a ReLU activation function.
   3. A flattening layer to convert the output to 1D, which is then fed into a linear layer with output_size=256 followed by ReLU activation.
   4. A linear layer with output_size='number of actions' (representing the Q-values of actions).

```
# First Convolutional Layer
self.conv1 = nn.Conv2d(num_frames, 16, kernel_size=8, stride=4)
self.relu1 = nn.ReLU()
```

```
    # Second Convolutional Layer
    self.conv2 = nn.Conv2d(16, 32, kernel_size=4, stride=2)
    self.relu2 = nn.ReLU()

    # Flatten the output for the linear layer
    self.flatten = nn.Flatten()

    # Linear Layer
    conv_out_size = self.calculate_conv_output_size(state_shape,
num_frames)
    self.fc1 = nn.Linear(conv_out_size, 256)
    self.relu3 = nn.ReLU()

    # Final Linear Layer
    self.fc2 = nn.Linear(256, n_actions)
```

Defined the functions calculate_conv_output_size(), forward(), get_qvalues(), sample_actions(), evaluate()

2. **Experience Replay**

The replay buffer of fixed size stores past agent experiences (state, action, reward, next state, done) and randomly sampled experiences for training. It replaces previous information with new information. Experience replay improves overall learning stability.

```
class ReplayBuffer:
  def __init__(self, size):
    self.size=size
    self.buffer=[]
    self.position=0


  def __len__(self):

    return len(self.buffer)
```

```
  def add(self, state, action ,reward, next_state, done):
    experience=(state, action ,reward, next_state, done)
    if len(self.buffer)<self.size:
      self.buffer.append(experience)
    else:
      self.buffer[self.position] = experience

    self.position = (self.position + 1) % self.size

  def sample(self, batch_size):

batch=random.sample(self.buffer,min(batch_size,len(self.buffer
)))
    return batch
```

3. Functions to play and compute loss

play_and_record(): makes the agent play and save data in exp_replay.

```
def play_and_record(start_state, agent, env, exp_replay,
n_steps = 1):
  state = start_state
  for _ in range(n_steps):
    state_t = torch.tensor([state], dtype=torch.float32,
device=device)
    qvalues_t = agent(state_t)
    qvalues = qvalues_t.cpu().detach().numpy()[0]
    action = agent.sample_actions(qvalues)[0]
    next_state, reward, done, _ = env.step(action)
    exp_replay.add(state, action, reward, next_state, done)
    if done:
      state = env.reset()
    else:
      state = next_state
```

compute_td_loss(): computes predicted Q-values of agent's actions and target Q-values using target network, then returns the loss.

```
def compute_td_loss(agent, target_network, device, batch_size,
exp_replay ,gamma = 0.99,):
  states, actions, rewards, next_states, dones =
exp_replay.sample(batch_size)
  states = torch.tensor(states).to(device)
  actions = torch.tensor(actions).to(device)
  rewards = torch.tensor(rewards).to(device)
  next_states = torch.tensor(next_states).to(device)
  dones = torch.tensor(dones).to(device)

    # Compute the predicted Q-values of the actions using the
agent
  q_values = agent(states)
  predicted_qvalues = q_values.gather(1,
actions.unsqueeze(1)).squeeze(1)

    # Compute the target Q-values of the actions using the
target network
  with torch.no_grad():
    target_q_values = target_network(next_states)
    target_qvalues_of_actions = rewards + gamma *
target_q_values.max(1).values * (1 - dones)

    # Compute the TD loss (Mean Squared Error)
  loss = torch.nn.MSELoss()(predicted_qvalues,
target_qvalues_of_actions)
  return loss
```

4. Epsilon-Greedy Exploration

Used epsilon-greedy exploration to balance exploration and exploitation. During action selection, the agent would select the best action

(exploitation) with probability 1 - epsilon, and choose a random action (exploration) with probability epsilon.

5. **Training**

   Using the data gathered from the environment, the DQN model was trained, and the agent discovered how to maximize cumulative rewards by maximizing Q-values. It was decided to stabilize the learning process by using a target network with set parameters.

   Used Adam optimiser for training with learning rate 2*e - 5

```
optimizer = torch.optim.Adam(agent.parameters(), lr=2e-5)
```

6. **Main loop**

   For each step, updated the exploration probability (epsilon), called the play_and_record() function and stored the experience in replay buffer, computed loss, updated network parameters of agent after backward propagation, obtained mean reward and final score using evaluate() function.

7. **Conclusion**

   The assignment successfully used DQN to solve complicated decision-making problems, demonstrating the power of deep reinforcement learning and its potential for use in practical settings.

Libraries used

```
import random
import numpy as np
import torch
import torch.nn as nn
import gym
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve, gaussian

import os
import io
import base64
import time
import glob
from IPython.display import HTML
import torch.nn.functional as F
from gym.wrappers import AtariPreprocessing
from gym.wrappers import FrameStack
from gym.wrappers import TransformReward
from tqdm import trange
from IPython.display import clear_output
import matplotlib.pyplot as plt
import torch.optim as optim
from numpy import asarray
from numpy import savetxt
```

Resources

https://drive.google.com/drive/folders/1_7KkyMia6P5m3gVDw4RpXy0VlP_M2nRt?usp=sharing

Part 1 :-

https://www.youtube.com/playlist?list=PLkDaE6sCZn6Ec-XTbcX1uRg2_u4xOEky0

Part 2 :-

https://www.coursera.org/learn/convolutional-neural-networks/home/week/1

week1-week3 videos for part-2

Part 3 :-

https://www.youtube.com/playlist?list=PLqnslRFeH2UrcDBWF5mfPGpqQDSta6VK4

Part 4 :-

https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver

first 5 videos

Part 5 :-

https://www.youtube.com/watch?v=Psrhxy88zww&list=PLwRJQ4m4UJjNymuBM9RdmB3Z9N5-0IlY0&index=2

L1 and L2

https://arxiv.org/abs/1312.5602