2048 Game

Abdallah Saba 900204018
Mahmoud ElGhandour 900201365

Problem Statement

- The game 2048 is a really successful game with millions of downloads; our goal is to optimize the reinforcement learning process in order to reach tiles with the highest value possible and to maximize the score along the way.

2048 explained:

- 2048 is a puzzle video game that is built on mathematical thinking, and with that, it has gathered the attention of many AI researchers in the past few years. The game is a 4X4 board game that requires moving all tiles around to merge similar tiles. Whenever you merge identical tiles, it creates a new tile with double the value. The game's goal was to reach the Tile with the value "2048". However, AI-trained models were able to reach tiles with higher values.

Game Rules:

- You can only move all the tiles together with 4 variations (Up, Down, Left, Right). Your score increases based on the value of the tiles that you merge. When two similar tiles collide, they create a new tile with double the original value. You lose if all the tiles are full and there are no more possible moves. The problem is that the human brain can fill up the board by mistake, and we are sure that the Deep learning model can dodge such errors.

Game Environment

- As a deep reinforcement learning model, we need to create an environment that can be used to train the model.
- We used the OpenAI Gym library to create the environment.
- The environment is a 4X4 board with 16 tiles.
- The environment has 4 actions (Up, Down, Left, Right).
- The environment has a reward system that rewards the model based on the value of the tiles that it merges.
- The environment has a reset function that resets the board to its initial state.
- The environment has a render function that renders the board to the screen.
- The environment has a step function that takes an action and returns the new state, reward, and if the game is over or not.
- The environment has a close function that closes the environment.

2048-reinforcement_learning / DeepQ_learning.py

Shilun-Allan-Li rename

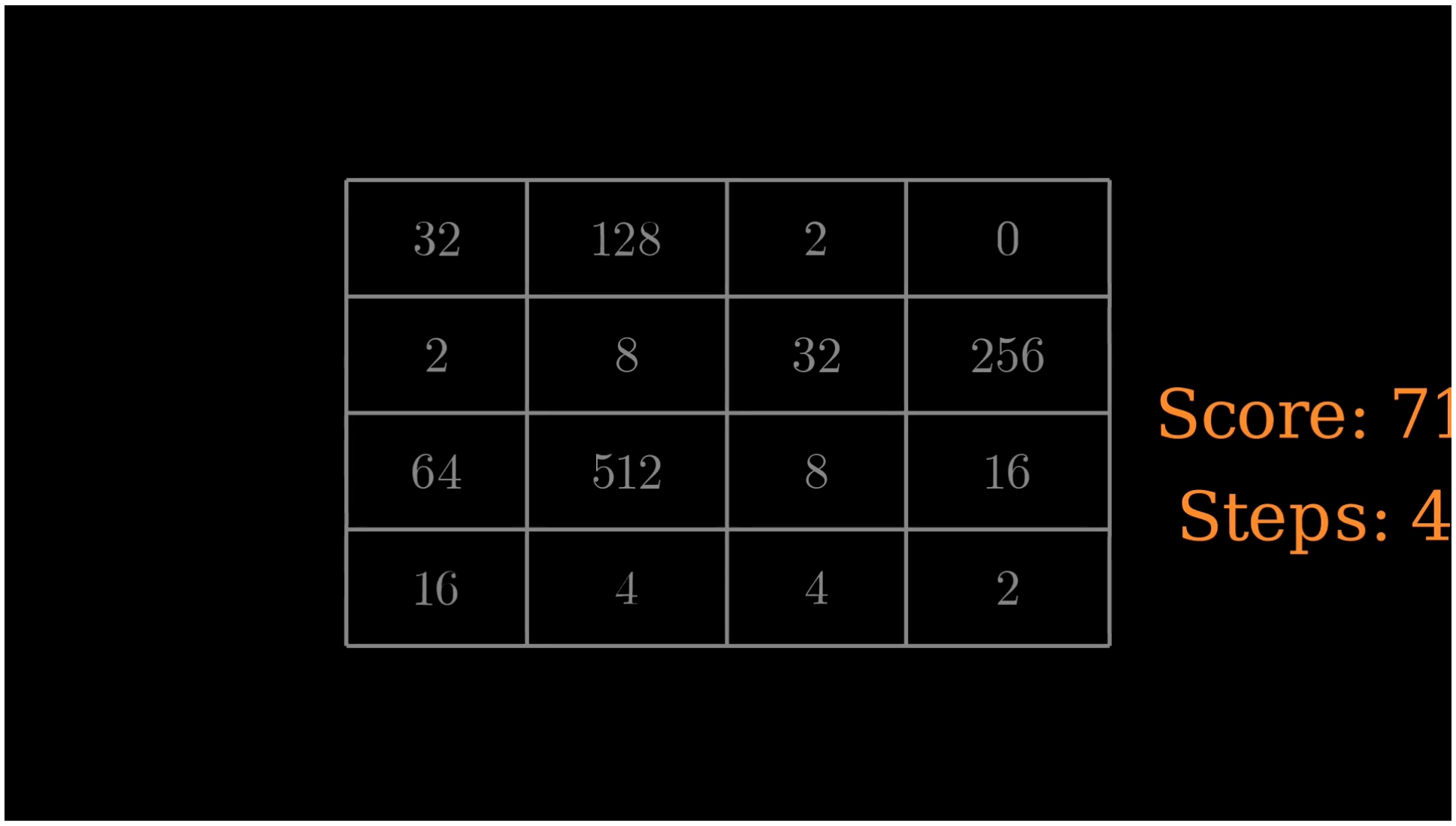
131 lines (118 loc) · 3.73 KB

Code Blame Raw

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on a lonly night before midterm
4
5   @author: Allan, Veronica
6   """
7
8   from game import Game
9   import numpy as np
10  import random
11  import sys
12  import torch.nn as nn
13  import torch
14  import torch.optim as optim
15
```

Demo Example

Here this is one game that was stopped after 500 moves, and we used the "manim" library in python to visualize the gameplay by our model into a video.



State of the art

This table compares between the results of our model and the results of other models.

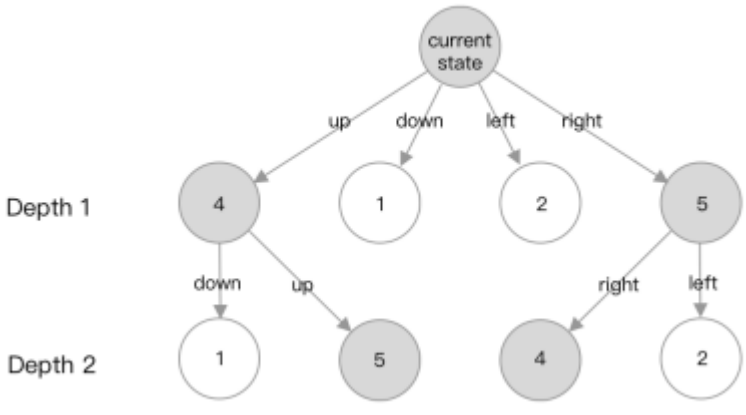
Model Name	Year Published	Framework Used	Metric Used	Metric Value
AlphaZero	2017	PyTorch (torch)	Average Score	500,000
MuZero	2019	PyTorch (torch)	Average Score	300,000
Stochastic MuZero	2021	PyTorch (torch)	Average Score	500,000
Beam Search	2021	PyTorch (torch)	Average Score	1,024 *
DQN (model 2)	2021	PyTorch (torch)	Average Score	256 *
Our model	2023	gym stable_baselines3	Average Score	11800
Hung Guei's model	2023	TDL2048+	Average Score	625,377

(*) this is the average for the tile values that the model has reached.

Orignial Model from Literature

DQN

- A rchitecture & Main Idea: there were two architectures that were used that consisted of 4 layers and they both had similar results. Explained below is the most successful model (DQN2)
- Layer 1: it had 2 parallel convolutional layers, and 2 different kernel modes to stabilize the learning process.
- Layer 2: it had 4 parallel convolutional layers, which take the previous output and transform it using different kernel sizes.
- Layer 3: in the third layer they used a dropout layer which helped to minimize overfitting.
- Layer 4: it is a linear layer that outputs the Q-value of the 4 possible actions to determine which is the best action



Proposed Updates

we had 4 major updates in mind that would help us improve the model and the results.

- DEEP Q-NETWORK (DQN)
- LINEAR PROGRAMMING
- MARKOV DECISION PROCESS (MDP)
- DYNAMIC PROGRAMMING

and we decided to contunie with the DQN update because it was the most promising update and it was the most related to our problem.

Update #1:Choosing the best action

the game environment that we were using was just choosing a random action from the 4 possible actions, and we thought that we could improve the model by choosing the best action from the 4 possible actions. So we initially improved that by just choosing the best action from the 4 possible actions, and we got a better result than the original model. Then we thought that we could improve that by choosing the best action from the 4 possible actions, Then when we learnt deep reinforcement learning we started using the bellman equation in order to predict the scores from the future moves and act acorrdingly which improved our results even more.

```
gamma=0.5
rotated_obs11 = np.rot90(updated_obs1, k=0)
reward11, updated_obs11 = self._slide_left_and_merge(rotated_obs11)
rotated_obs12 = np.rot90(updated_obs1, k=1)
reward12, updated_obs12 = self._slide_left_and_merge(rotated_obs12)
rotated_obs13 = np.rot90(updated_obs1, k=2)
reward13, updated_obs13 = self._slide_left_and_merge(rotated_obs13)
rotated_obs14 = np.rot90(updated_obs1, k=3)
reward14, updated_obs14 = self._slide_left_and_merge(rotated_obs14)
reward11=(reward11*pow(gamma,level))+reward_1
reward12=(reward12*pow(gamma,level))+reward_1
reward13=(reward13*pow(gamma,level))+reward_1
reward14=(reward14*pow(gamma,level))+reward_1

rewardleft=[0.0] * 4
if (level < 3):
    rewardleft[0] = self.all_moves(updated_obs11, reward11, level+1)
    rewardleft[1] = self.all_moves(updated_obs12, reward12, level+1)
    rewardleft[2] = self.all_moves(updated_obs13, reward13, level+1)
    rewardleft[3] = self.all_moves(updated_obs14, reward14, level+1)
MaxReward1=max(rewardleft)
MaxR=max(MaxReward1, reward14, reward13, reward12, reward11)
return MaxR
```

Update #2: Using the stable_baselines3 library

Until then we were only working on emproving on the game engine. Then we thought that we could improve the model by using a better library that is more optimized for reinforcement learning. So we started using the stable_baselines3 library which is a library that is built on top of the pytorch library and it is optimized for reinforcement learning. We used the DQN model from the stable_baselines3 library and we got a better result than the original model. We kept changing in the hyper-parameters of the DQN model until we were able to choose the ones that gave us the best results.

```
"""
(policy, env, learning_rate=0.0001, buffer_size=1000000, learning_starts=50000, batch_size=32, tau=1.0, gamma=0.9
train_freq=4, gradient_steps=1, replay_buffer_class=None, replay_buffer_kwargs=None, optimize_memory_usage=False,
target_update_interval=10000, exploration_fraction=0.1, exploration_initial_eps=1.0, exploration_final_eps=0.05,
max_grad_norm=10, stats_window_size=100, tensorboard_log=None, policy_kwargs=None, verbose=0, seed=None, device=
"""
model = DQN("MlpPolicy", env, verbose=1, learning_rate=0.0001, gamma=0.8, train_freq=4)
model.learn(total_timesteps=1000, log_interval=4)

# Number of iterations
num_iterations = 1  # Number of games played
```

Results

in our latest run we ran the model on 300 games and we were able to get:

- an average score of 12000
- average moves per game were 700
- and the maximum score that we got was around 32000

```
print('Average Moves: {}'.format(moves_average))
print('Average Scores: {}'.format(rewards_average))
print('maxmimum Score :{}'.format(max_score))
```

✓ 0.0s

```
Average Moves: 711.040404040404
Average Scores: 11882.545454545454
maxmimum Score :31828
```

Technical report

- Programming framework: Gym, stable_baselines3, manim
- Training hardware: google colab and visual studio code
- Training time: around 3 hours and 20 mins
- Number of epochs: 300 episodes
- Time per epoch: 0.6667 minute per epoch
- Any other important detail or difficulties

Conclusion

We were a bit restricted by using the stable_baselines3 library and its DQN model, altough it improved the scores but we didn't have full control over the Layers of this model. but we were not able to get the same results as the ones in the literature. We think that we could have gotten better results if we had more time to work on the project as we learnt Reinforcement learning late in the semester and altough we tried learning it on our own, we only understood the bigger picter after the second milestone.

References

1. Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T. K., & Silver, D. (2021, October). Planning in stochastic environments with a learned model. In International Conference on Learning Representations.
2. Desrosiers, Jacques & Lübbecke, Marco. (2006). A Primer in Column Generation.p7-p14 10.1007/0-387-25486-2_1.
3. Foster, D. (2019, December 1). MuZero: The Walkthrough (Part 1/3) | by David Foster | Applied Data Science. Medium. Retrieved September 26, 2023, from https://medium.com/applied-data-science/how-to-build-your-own-muzero-in-python-f77d5718061a [4]
4. Guei, H. (2022). On Reinforcement Learning for the Game of 2048. arXiv preprint arXiv:2212.11087.
5. Li, S., & Peng, V. (2021). Playing 2048 With Reinforcement Learning. arXiv preprint arXiv:2110.10374.
6. Stanford. (n.d.). Simple Alpha Zero. Stanford. Retrieved September 26, 2023, from https://web.stanford.edu/~surag/posts/alphazero.html
7. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MITPress.
8. van Heeswijk, W. (2021, January 17). Using Linear Programming to Boost Your Reinforcement Learning Algorithms. Towards Data Science. Retrieved September 26, 2023, from https://towardsdatascience.com/using-linear-programming-to-boost-your-reinforcement-learning-algorithms-994977665902