Katowice, 26.06.2021

BIAI

Python

vs

tic-tac-toe

1. Wstęp

Celem projektu było stworzenie modelu będącego w stanie grać i wygrywać w grę w kółko i krzyżyk. Poza tym celem projektu było zaznajomienie się ze sposobem tworzenia oraz działania sieci neuronowych.

2. Interfejs

Uzyskany model pozwala na konsolową grę w kółko i krzyżyk na wytrenowanej sieci. W kodzie możliwy jest wybór ilości gier, oraz wybór pomiędzy przeciwnikiem losowym oraz naszą siecią.

3. Dane

Sieć nie potrzebuje zewnętrznego datasetu, ponieważ sama sobie generuje dataset. Sieć trenuje się poprzez rozgrywanie rozgrywek z samym sobą (tą samą iteracją sieci). Model uzyskany z tego treningu testowany jest przeciwko starszemu modelowi, a wygrywająca sieć rozpoczyna proces od nowa.

Próbowaliśmy wykorzystać gotowy dataset, znaleziony w internetowej bazie danych (https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame), jednak doszliśmy do wniosku, że takie podejście nie ma sensu. Ten sposób trenowania sieci byłby słabo skalowalny, zwłaszcza w porównaniu do przyjętego przez nas podejścia. Byłby on również najprawdopodobniej wolniejszy z powodu potrzeby sprawdzania, czy stan naszej planszy to jeden z około 700 dostępnych stanów.

4. Kod pozwalający na granie z wytrenowanym modelem

Cały kod znajduje się na naszym repozytorium na GitHubie:
https://github.com/michalpatryk/tensorflow-tictactoe-game

Poniżej znajduje się przykładowy kod z pliku *pit.py*, który obsługuje grę sieci z losowym lub żywym graczem.

```
import Arena
from MCTS import MCTS
from TicTacToe.AITTTGame import Game
from TicTacToe.TTTPlayer import *
from TicTacToe.pytorch.NNet import NNet
from utils import *

game = Game()
rp = RandomPlayer(game).play

nn = NNet(game)
nn.load_checkpoint('./temp/', 'best.pth.tar')
args1 = dotdict({'numMCTSSims': 50, 'cpuct': 1.0})
mcts1 = MCTS(game, nn, args1)
n1p = lambda x: np.argmax(mcts1.getActionProb(x, temp=0))

arena = Arena.Arena(n1p, HumanTicTacToePlayer(game).play, game,
display=Game.display)
#arena = Arena.Arena(n1p, rp, game, display=Game.display)
arena.playGames(2, verbose=True)
# print(arena.playGames(2, verbose=True))
```

5. Biblioteki

Podstawą naszego kodu są biblioteki tensorflow oraz pytorch (torch) skompilowaną wraz z CUDA 11.1. Dodatkowo wykorzystaliśmy bibliotekę numpy do sprawnego operowania na tablicach danych.

6. Wyniki

Pomimo relatywnie krótkiego czasu trenowania (5 minut z wykorzystaniem technologii CUDA) model bez problemu wygrywa z ludźmi. Niestety trening nie jest w pełni wyważony - już po kilku iteracjach sieć zawsze remisuje, co powoduje jego nie zaakceptowanie. Po rozegraniu kilku partii z wytrenowaną siecią, nie zauważyliśmy aby popełniała ona jakiekolwiek błędy w rozgrywce. Praktycznie wszystkie mecze zakończyły się naszą porażką albo remisem.

7. Podsumowanie

Projekt pozwolił nam zapoznać się z biblioteką tensorflow, jak i interfejsami do niej - keras oraz pycharm. Niestety z powodu problemów z zapisywaniem modelu przez kerasa musieliśmy z niego zrezygnować. Okazało się to dobrym posunięciem - użycie biblioteki torch zdecydowanie (2-10x) przyspieszyło proces treningu sieci z powodu lepszej utylizacji procesorów CUDA.