Tic Tac Toe AI - Prezentáció (magyarázatokkal)

1. Bevezetés

- **Projekt**: Tanítható mesterséges intelligencia Tic Tac Toe játékhoz.
- **Cél**: Az AI képes legyen tanulni saját tapasztalataiból, hibáiból, és egyre job döntéseket hozni.
- **Eszközök**: Python, TensorFlow/Keras (gépi tanulás), ttkbootstrap (grafikus felület).

2. A játék és környezete

- **Tic Tac Toe**: 3x3-as tábla, két játékos (X és O), cél három egymás melletti saját jel.
- **Állapot**: 9 elemű vektor:
- `0`: üres mező
- `1`: játékos (X)
- `-1`: AI (O)
- **Lépés**: egy index (0–8), melyik mezőre lépjen az AI.

3. A tanulás alapjai – Q-érték

- **Q-érték**: az AI becslése arra, mennyire "jó" egy adott lépés egy adott állapotban.
- Minél nagyobb a Q-érték, annál előnyösebbnek ítéli meg az adott lépést.
- A Q-értékeket egy neurális háló becsüli meg minden egyes lehetséges lépésre.

4. A neurális háló architektúrája

Code Blame 11 lines (10 loc) · 339 Bytes Raw

```python
from tensorflow.keras import layers, models

def create_model():
    model = models.Sequential([
        layers.Input(shape=(9,)),
        layers.Dense(128, activation='relu'),
        layers.Dense(128, activation='relu'),
        layers.Dense(9, activation='linear')
    ])
    model.compile(optimizer='adam', loss='mse')
    return model
```

Magyarázat:

- `Input(shape=(9,))`: bemenet 9 számból áll (a tábla állapota)
- `Dense(128)`: teljesen összekötött (fully connected) réteg 128 neuronnal
- `activation='relu'`: a ReLU (Rectified Linear Unit) aktivációs függvény – gyors éshatékony, ha pozitív, átengedi az értéket, ha negatív, akkor nullát ad vissza
- `Dense(9)`: 9 kimeneti neuron, mindegyik egy mezőhöz tartozó Q-értéket becsül
- `activation='linear'`: kimenet nincs torzítva, lehet negatív vagy pozitív
- `optimizer='adam'`: adaptív optimalizáló algoritmus, jól működik a legtöbb feladathoz
- `loss='mse'`: "mean squared error" – a háló azt tanulja, hogy a saját Q-becsléseimennyire térnek el a kívánt értékektől

5. Tanítás (train.py)

- **Epizód**: egy teljes játék lejátszása- **Tanulási ciklus**:
- véletlenszerű lépésekkel indul (felfedezés – "exploration")
- majd egyre jobban a tanult Q-értékek alapján dönt ("exploitation")
- **Epsilon**: annak valószínűsége, hogy az AI véletlenszerű lépést tesz
- magas eleinte (pl. 1.0), majd csökken (pl. 0.01)

6. Jutalmazás (reward rendszer)

- **+10**: ha az AI nyer
- **-10**: ha az AI veszít
- **0**: ha döntetlen
- A háló ezen értékek alapján frissíti a Q-becsléseket minden lépés után

7. GUI működés

- tkinter + ttkbootstrap
- Mezők háttere zöld → minél zöldebb, annál magasabb a Q-érték
- A mező alján megjelenik a Q-érték (pl. `0.45`)
- Konzol: kiírja az aktuális lépéshez tartozó összes Q-értéket

8. Modell értékelés

- AI-t tesztelünk egy véletlenszerű játékos ellen- Metrikák:
- győzelem/vereség/döntetlen arány
- learning curve: veszteség csökkenése az epizódok során

9. Eredmények

- Már néhány száz epizód után látványos tanulás
- A GUI vizualizációja világosan mutatja az AI gondolkodását
- A modell egyre következetesebben játszik jól

10. Fejlesztési lehetőségek

- AI vs AI játszmák, különböző generációk összevetése
- Replay mentés és visszajátszás
- Mélyebb tanulás memória-visszajátszással (Experience Replay)
- Webes frontend, REST API, multiplayer

11. Követelmények teljesítése

- Gépi tanulás/neuronhálós megoldás: ✅
- Saját ötlet, működő AI: ✅
- Open source kód + prezentáció: ✅
- Oktatási célra jól bemutatható, demonstratív: ✅

Melléklet: Használt fájlok

- `train.py` – AI tanítása
- `model.py` – neurális háló definíciója
- `game.py` – játék logikája
- `gui_tic_tac_toe.py` – grafikus felület, interakció és tanulás
- `evaluate_models.py` – különböző modellek összehasonlítása