AI for Ultimate TicTacToe

Cameron Cianci, Muhamad Faraj, Christopher Davies

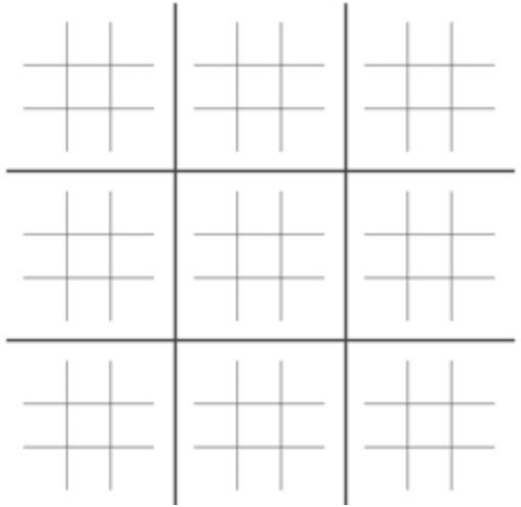
Our Plan:

- Ultimate Tic Tac Toe Gameplay
  - Create a Ultimate Tic Tac Toe game in python
  - All game functions should be clear to the player
- Trainable AI
  - Have a trainable AI to play against
  - Player decides to train the AI or play against the AI
  - Should be able to be saved
  - Previously saved AI can be loaded inside the code

What is Ultimate TicTacToe

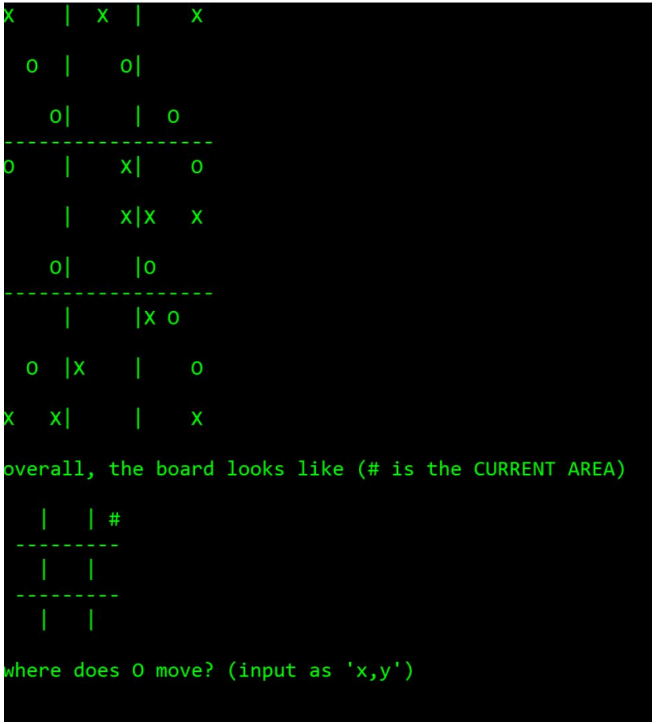
- TicTacToe boards within a TicTacToe board
- Each players move decides which board the next player uses
- Complicated strategy

- If you're interested, https://bejofo.net/ttt



Our Approach

1. Model the game in command line
2. Create a neural network which can play the game
3. Train the network through a genetic algorithm
4. Allow for users to play against the NN



The Neural Network and Training

- Split into 3 networks
  - Global Sentiment Net.
  - Local Sentiment Net.
  - Move Decision Net.
- Trained by Tensorflow's Evolution function, `tfp.optimizer.differential_evolution_minimize()`

```
fungus = tf.keras.Sequential([#fungus does whole board sentiment
    layers.Dense(8, activation='relu', name="layer1", input_shape=(18,), dtype='float32'),
    layers.Dense(2, activation='sigmoid', name="layer2")
])
fungus.compile(optimizer=tf.keras.optimizers.Adam(0.01),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

chungus = tf.keras.Sequential([#chungus does local sentiment
    layers.Dense(12, activation='sigmoid', name="layer1", input_shape=(18,), dtype='float32'),
])
chungus.compile(optimizer=tf.keras.optimizers.Adam(0.01),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

amongus = tf.keras.Sequential([#amongus chooses
    layers.Dense(10, activation='relu', name="layer1", input_shape=(36,), dtype='float32'),
    layers.Dense(9, activation='sigmoid', name="layer2")
])
amongus.compile(optimizer=tf.keras.optimizers.Adam(0.01),
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

Forking and Multithreading

- `tfp.optimizer.differential_evolution_minimize()` automatically creates two thread pools, with threads for each network
  - Therefore training is automatically parallelized
- Also the AI network is forked during play (right)
  - Could allow for augmentations of the game for more than 2 players. Ex(4 players running on different sub-boards)

```
printBoard()
printBigBoard()
print("Where does O play?(AI)")
pr = os.fork()
if pr is 0:
    AIchildprocess()
else:
    pro= os.wait()
if(wincon() != 0):
    break
```

Results

- The Artificial Intelligence could play the game and make moves
- AI is a challenging opponent, but very possible to defeat
- Parallelism/Forking accelerated training and could allow for 4 player variants