Most important features

Grupo 3

UC-Laboratório de inteligência artificial e ciência de dados

Alphazero e o seu funcionamento:

Na nossa implementação do alphazero existe:

Uma classe **ResNet**, que representa a neural network utilizada para avaliar a posição atual do jogo e estimar as probabilidades de vitória para as diferentes ações possíveis. A sua entrada corresponde ao estado do jogo (game_state), e as suas saídas principais saídas são uma estimativa da probabilidade de vitória para o jogador atual e uma estimativa do quão favorável é o estado para o jogador atual.

O **Monte Carlo Tree Search**, que simula múltiplos jogos para encontrar a melhor jogada em um determinado estado do jogo. O critério que nós optámos por utilizar é o Upper Confidence Bound (UCB), de modo a equilibrar a exploração. De seguida é feita uma simulação de Monte Carlo, simulando-se o jogo a partir do novo nó. Os resultados das simulações são propagados de volta através da árvore.

Assim, durante a fase de seleção do MCTS, a probabilidade de seleção de cada ação é influenciada pelas saídas da rede neuronal.

A classe **node** representa um nó na árvore de busca MCTS.

A classe **AlphaZero** coordena o processo de treino do modelo, usando o MCTS para gerar dados de treino através de jogos contra si próprio. Utilizamos na implementação uma componente de “temperatura” que introduz alguma variabilidade às possíveis jogadas do nosso modelo.

É importante notar que implementámos também uma função **augment_state**, que permite realizar a aplicação de transformações de simetria, sendo possível aumentar o conjunto de dados de treino.

Treino dos modelos

O treino dos modelos ocorre com o modelo jogando contra si próprio:

No treino do **Go**, tivemos de eliminar a opção de passar, jogando o modelo até não haver mais posições possíveis. Tomámos esta decisão pois quando testávamos o resultado do treino, o modelo limitava-se a passar e demonstrava uma enorme preferência por essa ação quando não era o suposto.

No **Attaxx**, definimos um limite de movimentos máximo pois o modelo entrava num loop infinito de jogadas que, apesar de válidas, impediam o treino do modelo.

Teste dos modelos

O teste dos modelos é feito jogando **2N jogos** com turno alternado, N contra um algoritmo **Greedy** e N contra um algoritmo **Random**.

O algoritmo **Greedy** usa duas heurísticas diferentes dependendo do jogo:

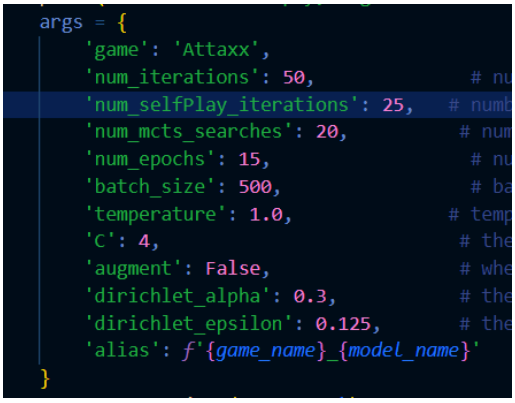
Go: A diferença do número de peças + territórios conquistados, para cada jogador.

Attaxx: A diferença do número de peças, para cada jogador

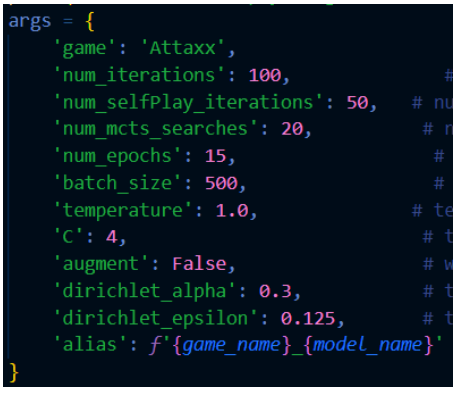
Parâmetros utilizados

Para treinar os nossos modelos usamos dois conjuntos diferentes, um para o **Go** e outro para o **Attaxx**:

Go:



Ataxx:



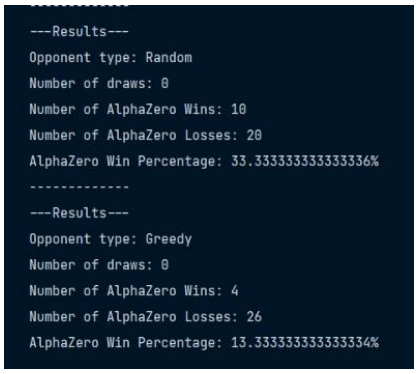
Devido à demora que o treino apresentou desde inicio não conseguimos testar todos os parâmetros possiveis. Chegamos a este conjunto de valores e a partir daí colocamos o número de selfPlays e de iterações ajustado para cada jogo.

Inicialmente colocamos o número de iterações mais elevado que o número de jogos. Com base nos testes invertemos isso pois concluimos que embora com menos jogos o modelo se torna melhor se treinar mais vezes.

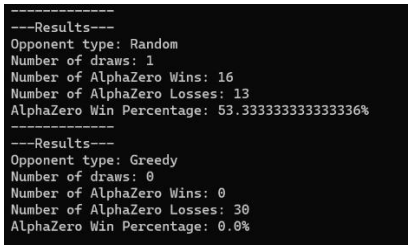
Resultados:

Ataxx 4x4:

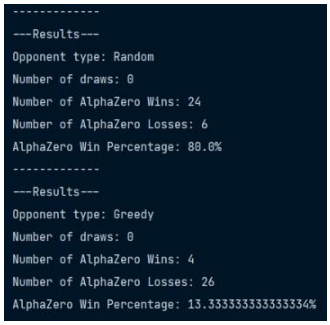
Ataxx 5x5:



Attax 6x6:



Go 7x7:



Go 9x9:

```
-------------
---Results---
Opponent type: Random
Number of draws: 0
Number of AlphaZero Wins: 4
Number of AlphaZero Losses: 6
AlphaZero Win Percentage: 40.0%
-------------
---Results---
Opponent type: Greedy
Number of draws: 0
Number of AlphaZero Wins: 5
Number of AlphaZero Losses: 5
AlphaZero Win Percentage: 50.0%
```

Análise dos resultados:

Ataxx:

4x4: Como o tabuleiro é pequeno o primeiro jogador ganha sempre se jogar otimamente. Por esse motivo, o nosso modelo consegue aprender rapidemente a vencer quando joga em primeiro, impossibilitando a capacidade de aprender a jogar um segundo, pois ganha contra ele mesmo em apenas duas jogadas. A prova disso é que quando realizamos os testes com o modelo sendo sempre o primeiro obtemos 100% de vitória em ambos os testes. Pelo contrário, quando joga sempre em segundo perde quase 100% dos jogos, pois não “sabe” jogar em segundo.

5x5/6x6: Os resultados ficaram ambos longe do esperado, sendo o Attaxx um jogo com um espaço de procura grande. As nossas capacidades computacionais não são suficientes para criar um modelo capaz de obter um melhor score contra o greedy.

Go:

7x7/9x9: Os resultados ficaram ambos longe do esperado, sendo o Go um jogo com um espaço de procura ainda maior que o Attaxx. As nossas capacidades computacionais ou mesmo a nossa implementação podem não ser suficientes para criar um modelo capaz de obter um melhor score contra o greedy.