Inteligência artificial para o jogo de Hex

Fábio Henrique Kiyoiti dos Santos Tanaka, Orientador: Denis Deratani Mauá

Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo

Introdução

A utilização de jogos de tabuleiro como objeto de testes para o estudo de inteligência artificial não é uma ideia nova, um dos exemplos mais famosos foi o computador *Deep Blue* da IBM, que em 1997 derrotou o campeão mundial de xadrez da época, Garry Kasparov, sendo este considerado um grande marco na história da computação.

O objetivo deste trabalho é desenvolver uma inteligência artificial para o jogo de Hex e com isso estudar diferentes conceitos da área de aprendizado de máquina. *Q-learning* e redes neurais foram algumas das principais estratégias utilizadas durante o desenvolvimento do programa.

Este projeto foi inspirado pelo *paper* "Neurohex: A Deep Q-learning Hex Agent"[1] e busca reproduzir seus resultados utilizando a biblioteca *pytorch*[2] do python.

O jogo de Hex

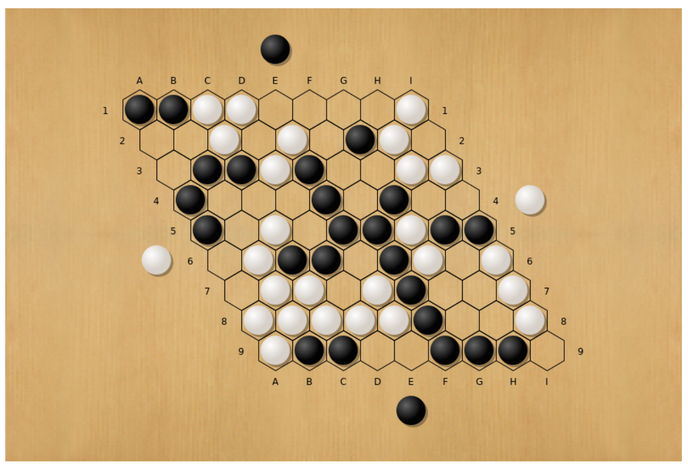

Figura: Jogo de Hex em progresso [9]

Hex[4] é um jogo de 2 participantes jogado em uma grade hexagonal tradicionalmente em forma de um losango com dimensões 11x11 ou 13x13 em que cada par de lados opostos recebe uma cor distinta. Durante a partida cada jogador recebe peças de uma dessas cores e eles se alternam colocando-as em espaços vazios do tabuleiro. O objetivo do jogo é criar um caminho utilizando suas peças de forma a conectar os lados opostos do tabuleiro que estejam marcados com a sua cor.

Apesar das regras simples, o jogo de Hex requer tanto planejamento estratégico quanto habilidades táticas tornando-o adequado para ser usado no estudo de algoritmos de aprendizado de máquina[3] [5].

Aprendizado por reforço

Aprendizado por reforço é um campo do aprendizado de máquina onde um agente aprende como interagir com o ambiente por meio de tentativa e erro, isso é, aplicando ações e observando os resultados.

Em cada passo o agente recebe do ambiente o estado em que se encontra e então seleciona uma ação para tomar. Após isso o ambiente retorna dois valores: o novo estado que o agente está e uma recompensa, que pode ser tanto positiva quanto negativa, que avalia o quão boa foi a ação realizada. Este processo então é repetido até que o ambiente devolva um estado terminal e assim finalize o episódio. Uma política π, é uma função que ao receber um estado determina uma ação que o agente deve tomar: $\pi(s_t) \rightarrow a_t$.

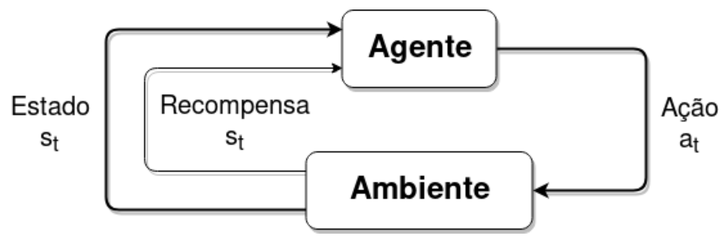

Figura: Ilustração de como funciona o aprendizado por reforço [7]

Deep Q-learning

Deep Q-learning é uma técnica de aprendizado por reforço onde busca-se aproximar o valor da função Q utilizando redes neurais. A função Q, por sua vez, é uma equação que busca avalizar os valores de pares estado, ação. Neste projeto foi utilizada uma rede neural convolucional como arquitetura para o aprendizado.

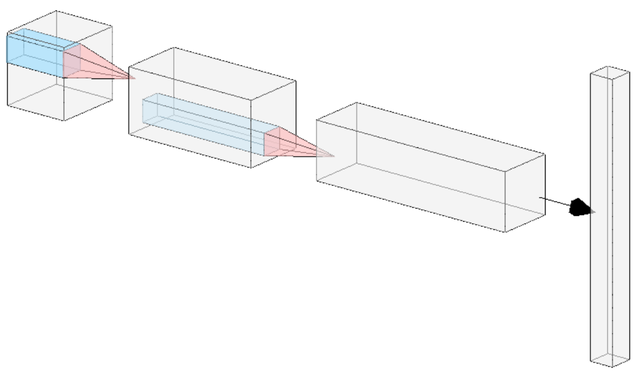

Figura: Representação visual de uma rede neural convolucional [6]

Para treinar a rede e assim aproximar melhor o valor das ações em cada estado foi utilizada a seguinte função de erro:

$$loss = 0.5[(r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1})) - Q(s, a)]^2$$

Testes e resultados

Após ser verificado que o programa estava de fato funcionando da maneira desejada, foram realizados testes com diversas configurações com o objetivo de estimar os melhores valores para diferentes atributos. Abaixo se encontram os melhores destes resultados.

classificação	lr	gamma	aleatoriedade	# de camadas	% de vitórias
1	0.01	0.8	0.1	2	69%
2	0.01	0.9	0.1	1	26%
3	0.1	0.9	0.1	1	14%
4	0.1	0.8	0.1	1	9%
5	0.1	0.8	0.25	1	9%
6	0.01	0.8	0.4	2	9%

Tabela: Testes variando diferentes atributos

Utilizando a melhor destas configurações foram feitos testes com mais exemplos com o objetivo de obter resultados mais precisos. Alguns dos resultados podem ser conferidos abaixo.

lr	gamma	aleatoriedade	# de camadas	% de vitórias
0.01	0.8	0.1	2	77%

Tabela: Resultados após 300000 jogos

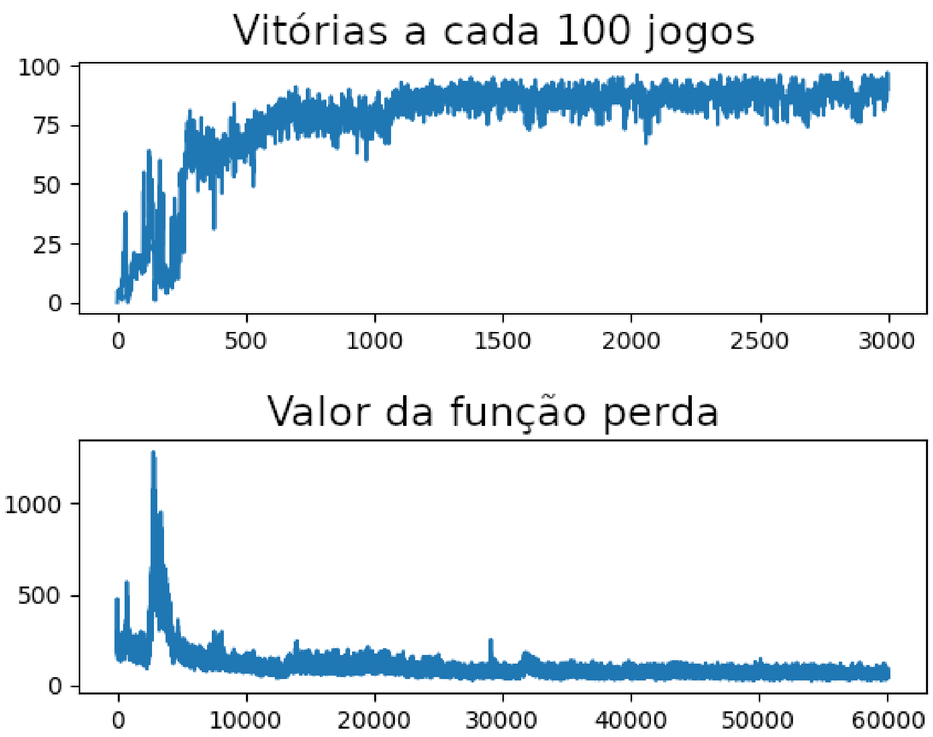

Figura: Gráficos da quantidade de vitórias e da função de erro. [8]

Conclusão

Para este projeto foi construído com sucesso uma inteligência artificial para o jogo de Hex. Apesar dela ter sido trabalhada em um tabuleiro pequeno e portanto mais simples, seus resultados foram positivos e sua implementação é suficientemente genérica de forma que com mais testes e aperfeiçoamentos individuais, seria possível jogar em tabuleiros maiores.

A maior dificuldade encontrada no desenvolvimento do programa foi identificar a origem de diversos problemas pois é complicado diferenciar se um erro ocorreu devido à falta de conhecimento teórico, erro na implementação de um algoritmo ou falta de otimização para o problema em específico.

Referências

[1] Gautham Vasan Kenny Young, Ryan Hayward. Neurohex: A deep q-learning hex agent. **https://arXiv:1604.07097v2**, 2016.

[2] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga e Adam Lerer. Automatic differentiation in pytorch. Em NIPS-W, 2017

[3] Vadim V. Anshelevich. The game of hex: An automatic theorem proving approach to game programming. **http://vanshel.com/Hexy/Publications/VAnshelevich-ARTINT.pdf**, 2000

[4] John Nash. Some games and machines for playing them. **https://www.rand.org/content/dam/rand/pubs/documents/2015/D1164.pdf**, 1952.

[5] Stefan Reisch. Hex is PSPACE-complete. Acta Informatica, 15(2):167–191, 1957

[6] Diagrama construído pela ferramenta: **http://alexlenail.me/NN-SVG/AlexNet.html**

[7] Diagrama construído pela ferramenta: **http://draw.io**

[8] Gŕafico construído pela ferramenta: **https://matplotlib.org/**

[9] Tabuleiro construído pela ferramenta: **https://github.com/ryanbhayward/hexgui**