AGENTES AUTÔNOMOS E REINFORCEMENT LEARNING

ELETIVA

Projeto Final - Entrega Final

Caio Emmanuel

Entrega final contendo implementação e resultados finais sobre um agente autônomo desenvolvido para o jogo Doom.

São Paulo

Junho de 2022

Sumário

1 Introdução

1.1 Contexto

Este trabalho foi realizado sobre o ambiente da biblioteca *Python* chamada *ViZDoom*[1], uma biblioteca criada para integrar os cenários recriados de *ZDoom*[2] com a estrutura de *environments* da biblioteca *gym*[3].

Os agentes criados foram treinados sobre os cenários da categoria *basics*, ou seja, eles possuem apenas uma sala pequena, com um ou alguns poucos monstros e a movimentação é limitada a um único grau de liberdade (esquerda-direita).

1.2 Objetivo

O objetivo desse trabalho é implementar e analisar a performance de um agente autônomo atuando sobre os cenários *basics* da biblioteca *ViZDoom*.

2 Implementação

O código principal para implementar o agente está no arquivo *agent.py* no repositório que foi enviado junto desse arquivo. Os parâmetros importantes que forem citados estarão configurados no arquivo *constants.py* e algumas funções que forem citadas estarão no arquivo *utils.py*.

2.1 Agente

O agente desenvolvido utiliza o algoritmo *Double Deep Q-Learning*[4] que consiste de utilizarmos duas redes neurais para simular os valores produzidos por uma *Q-Table* no algoritmo clássico *Q-Learning*. Onde uma das redes gera os valores da *Q-Table* mas seus pesos são atualizados com menos frequência que a outra rede, que é atualizada a cada iteração do treinamento, mas produz valores com uma maior variação por conta disso.

Para isso foi utilizado o módulo *PyTorch*.

2.1.1 Rede neural

A topologia da rede neural utilizada foi uma topologia convolucional, aqui não houve tanta preocupação em construir essa topologia visto que o foco era a performance do algoritmo de treinamento do agente e não da rede neural e, portanto, foi utilizada a topologia padrão dos exemplos da documentação do *Pytorch* sobre *Convolutional Networks* adaptada para os frames do jogo.

A rede neural consiste de quatro camadas de convolução e duas *fully-connected* ao fim.

2.1.2 Hiperparâmetros

Sobre os hiperparâmetros, eles podem ser vistos em ‘constants.py‘ como dito, mas não há nada de especial sobre eles e foram escolhidos os mesmos que o tutorial do *ViZDoom*.

O único hiperparâmetro que é importante comentar é o número de *epochs* de treinamento. Segundo os próprios desenvolvedores, e conforme verificado durante o treinamento, um número muito alto de *epochs*, superior a 20 (vinte), reduz a capacidade de aprendizado do agente e este até mesmo começa a regredir. Para 20 *epochs* o *score* do agente converge para algo entre 70 e 80.

2.2 Pré-processamento

Os *frames* do jogo são preprocessados para diminuir a dimensão do *input* da rede neural, necessitando de menos pesos para serem atualizados e, consequentemente, acelerando o treinamento. O pré-processamento é simples e constitui basicamente de diminuir as dimensões da imagem para outra dimensão especificada e retirar as três dimensões de cores, visto que as cores não traduzem nenhuma informação relevante.

2.3 Treinamento

O treinamento é semelhante ao proposto pelo *paper* original sobre *Double Deep Q-Learning*, ou seja, são gerados um número de *batch frames*, em seguida os pesos da rede principal são treinados tendo como entrada esses *frames* e os respectivos valores da *Q-Table* produzidos pela rede *target* que só será atualizada após algumas épocas com os pesos da rede principal.

Nesse caso em especial, foi adicionado um teste após cada treinamento, para verificar-se com novos *frames* o desempenho do agente.

Os valores dos *scores* são salvos para então produzir um gráfico com a evolução destes entre as *epochs*.

3 Resultados

Os resultados podem ser verificados de duas formas.

Ao fim do treinamento, o agente irá jogar um número de jogos determinado em ‘constants.py‘ para verificar visualmente o comportamento deste.

Ainda, será gerado uma imagem com a evolução dos scores no tempo como a Figura 1.

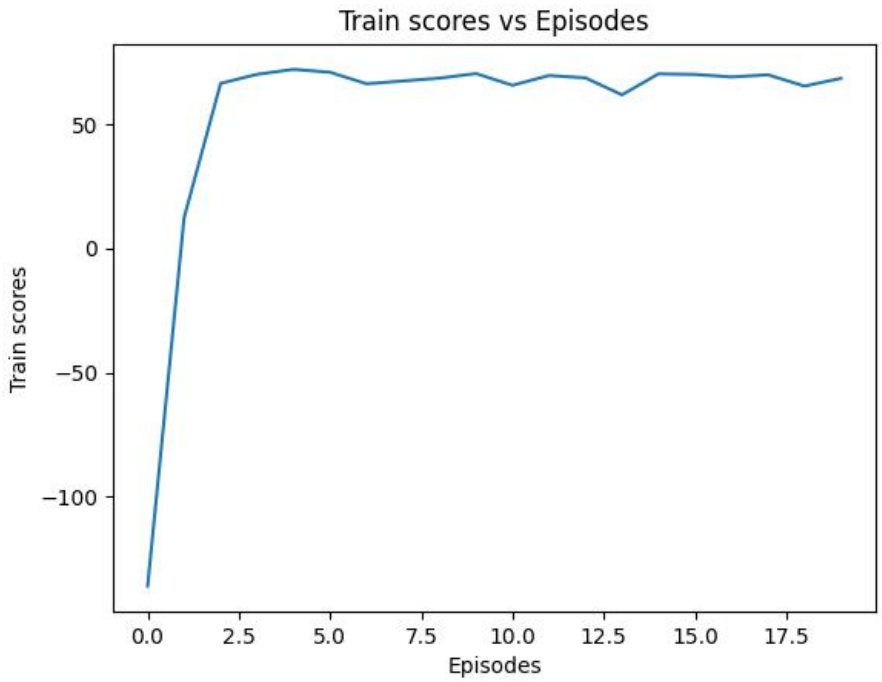

Figura 1: Average Score per Steps - Doom Agent

Em geral, o resultado foi satisfatório e cumpriu ao *score* esperado segundo os comentários dos desenvolvedores que diziam que o *score* deveria fixar entre 70 e 80 no caso ideal.

Referências

[1] *ViZDoom*. https://github.com/mwydmuch/ViZDoom. Accessed: 2022-05-31.

[2] *ZDoom*. https://github.com/rheit/zdoom. Accessed: 2022-05-31.

[3] *gym-library*. https://www.gymlibrary.ml/. Accessed: 2022-05-31.

[4] Hado van Hasselt, Arthur Guez e David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].