Agentes de Aprendizado por Reforço para Space Invaders com Redes Neurais Profundas

Luca de Paula Nascimento Lima
Universidade Federal de São Paulo
UNIFESP
São José dos Campos – SP Brasil
lukadp10@gmail.com

***Resumo*—aborda o tema da aprendizagem de máquina (machine learning) e, mais especificamente, o aprendizado por reforço (reinforcement learning). O aprendizado por reforço envolve treinar um agente para tomar uma sequência de decisões em um ambiente incerto e complexo, recebendo recompensas e penalidades com base em suas ações. O artigo discute o uso de agentes de aprendizado por reforço, especificamente o Deep Q-Learning, no ambiente do jogo Space Invaders e compara o seus resultados afim de determinar a eficácia do algoritmo.**

***Index Terms*—aprendizagem de máquina ,machine learning , aprendizado por reforço, reinforcement learning , Q-Learning, Deep Q-Learning, Space Invaders**

I. Introdução

A aprendizagem de maquina (machine learning) é um subcampo da inteligência artificial o qual se baseia no desenvolvimento de algoritmos e modelos que possibilitam que modelos computacionais aprendam apartir de dados sem que tenham uma programação explicita, são projetados para que possam aprender através de dados, reconhecer padrões e tomar decisões apartir desses padrões.

Podemos dividir a aprendizagem de maquina em 3 grandes classes, são aprendizagem supervisionada, aprendizagem não supervisionada e aprendizagem por reforço que será aplicada nesse trabalho. A aprendizagem supervisionada é a que um modelo será treinado usando dados que são rotulados, ou seja para cada exemplo de entrada teremos uma saida que é o conjunto verdade. O objetivo desse modelo é aprender a mapear as entradas para as saídas corretas, com base nos exemplos fornecidos.

Diferente da aprendizagem supervisionada, a não supervisionada treina seus modelos aparti de um conjunto de dados o qual não está rotulado, o objetivo desse modelo é encontrar padrões, estruturas ou representações sem nenhum conhecimento prévio das categorias das bases de dados.

Diferentemente dos anteriores, o aprendizado por reforço(Reinforcement Learning – RL) faz com que seu modelo(ou agente) interaja com um ambiente incerto e muitas vezes complexo e o treina para tomar uma sequência de decisões. Para que a esse agente aprenda a completar um objetivo, ele recebe penalidades se suas ações não forem boas e recompensas caso as ações o deixe mais próximo do objetivo. Dentre os diversos uso da aprendizado por reforço, a um que chama bastante atenção por mostrar a força que esses algoritmos tem, que é o uso de aprendizagem por reforço para ensinar uma maquina a jogar um jogo.

Identify applicable funding agency here. If none, delete this.

Jogos geralmente tem ambientes complexos e com inumeras possibilidades de ações, logo são perfeitos para o uso de modelos de aprendizagem por reforço, esses modelos são capazes de lidar com a incerteza desses ambientes e permitir que a maquina explore estratégias diferentes e aprendar a tomar as melhores decisões com base no feedback.

Para esse projeto, veremos como o algoritmo Deep q-Learning se sai no ambiente do jogo space invaders.

II. Conceitos Fundamentais

Como comentado anteriormente, o aprendizado de maquina é um subcampo da inteligência artificial que visa com que modelos computacionais aprendam a partir de dados reconhecer padrões e tomar decisões a partir desses padrões. Dentro do tópico de aprendizagem de maquina temos a aprendizagem por reforço.

A aprendizagem por reforço(Reinforcement Learning – RL) é o treinamento de modelos de aprendizado de máquina para tomar uma sequência de decisões. Nessa forma de aprendizagem temos um agente que irá aprender a completar um objetivo em um ambiente, no qual as interações do agente com o ambiente dão feedbacks para que o agente através da busca de maximização de recompensas(bom feedback) atinga a sua meta.

A aprendizagem por reforço tem os seguinte componentes, Agente, Ação, Fator de desconto, Ambiente, Estado, Recompensa, Política, Valor, Valor Q, Trajetória e Distinções principais.

Agente é o algoritmo, ele executa as ações, como por exemplo nesse trabalho o agente será a nave do jogo space invaders.

Ação é o conjunto de todos os movimentos possíveis do agente, no caso desse trabalho seria os movimentos da nave, como ir para direita, ir para esquerda e atirar.

Ambiente é o mundo no qual o agente se move e interage, nesse trabalho o ambiente seria a fase do jogo que é dada pela biblioteca OpenAI Gym.

Estado é a situação concreta e imediata em que o agente se encontra no ambiente, no caso do jogo space invaders seria um momento especifico como no meio da fase.

Recompensa é o feedback que o nosso agente recebe e o que usamos para medir o sucesso ou fracasso das suas ações, exemplo desse trabalho seria quando a nave(agente) explode um dos inimigos ela ganha pontos.

Política é a estratégia do agente em definir sua próxima ação, usando de base o seu estado atual.

Valor é uma medida de retorno esperado ou da recompensa acumulativa que o agente pode esperar em um determinado estado ou ambiente.

Valor Q é o valor esperado futuramente como recompensa por uma ação do agente tomada em um estado s.

Trajetória é uma sequencia de estados e ações as quais influenciam esses estados.

Recompensa é um sinal imediato recebido em um determinado estado, enquanto valor é a soma de todas as recompensas que você pode antecipar desse estado.

Para entender como um modelo de RL(Reinforcement Learning) funciona temos que primeiro entender o processo Markov de decisão, MDPs (Markov Decision Processes).

MDPs é um modelo matemático usado para modelar problemas de tomada de decisões sequenciais, são bastante utilizadas em aprendizado por reforço, nele temos que o ambiente é modelado como um processo de decisão em etapas discretas, em que um agente toma ações com base no estado atual e recebe recompensas do ambiente. No MDP temos 5 componentes principais, um conjunto de estado (S) que representa todos os estados possíveis do ambiente, um conjunto de ações (A) que representa todas as ações que o agente pode tomar, um conjunto de recompensar (R) que tem o conjunto de recompensas que pode ser usado no modelo, temos os diferentes tempos que são representadas por (t = 0, 1, 2...). A cada tempo o agente recebera uma representação do estado do ambiente $S_t \in S$ e baseado nesse estado o agente escolhe uma ação $A_t \in A$ isso nos dará um par de estado e ação S_t, A_t. Quando o tempo for incrementado, t+1 e o estado ambiente transiciona para um novo estado $S_{t_1} \in S$, o agente recebera uma recompensa $R_{t+1} \in R$ que é dada por cima da ação antiga A_t do estado anterior S_t.

Podemos ver esse processo de recebimento de recompensa como uma função arbitrária, $f(S_t, A_t) = R_{t+1}$. podemos visualizar melhor como o processo funciona na figura 1.

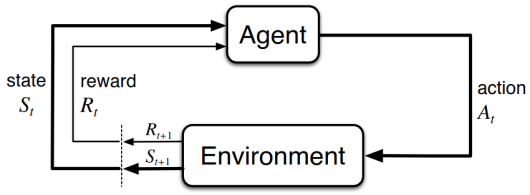

Figura 1. Legenda da imagem.

Em um MDP finito os conjuntos de estados, ações e recopensas S, A, R tem um valor finito de numero de elementos. Neste caso as variáveis aleatórias $R_t \in S_t$ êm distribuições de probabilidade discretas que dependem apenas do estado e ação anteriores. Logo para valores especificos dessas variavéis $s' \in S$ e $r \in R$ tem a probabilidade desses valores ocorrerem em um tempo t, dados os valores do estado e ação anteriores:

$p(s', r \mid s, a) = \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$.

Outro conceito importare para o entendimento desse trabalho são as redes neurais, redes neurais são modelos matemáticos inspirados no funcionamento do cérebro humano. Elas consistem em uma rede interconectada de unidades computacionais chamadas de neurônios artificiais, que processam informações e aprendem a partir dos dados. As redes neurais são usadas para tarefas de reconhecimento de padrões, classificação, previsão e tomada de decisões, sendo especialmente eficazes em lidar com dados complexos e não lineares.

Temos tambem saber como o algoritmo Deep Q-Learning funciona. Para explicar o DQL(Deep Q-Learning) precisamos explicar primeiramento sobre seu modelo mais básico o Q-Learning. Q-Learning é baseado em um modelo chamado de tabela Q, a qual armazena valores Q para cada par estado-ação. O algoritmo funciona da seguinte forma, primeiro temos a inicialização da tabela Q com valores arbitrários, depois no passo 2 o agente irá interagir com o ambiente a partir de uma politica de exploração-exploração, a ação escolhida pode ser aleatória ou baseada nos valores Q. Depois no passo 3 o agente irá realizar uma ação e irá receber uma recompensa do ambiente, logo depois no passo 4 ele irá atualizar o valor de Q para todos os estados-ação através da formula: $Q(S_t, A_t) \leftarrow (1-\alpha) Q(S_t, A_t) + \alpha (R(S_t, A_t) + \gamma \max_a Q(S_{t+1}), A)$ na qual α representa a taxa de aprendizagem, γ representa o fator de desconto. $Q(S_t, A_t)$ representa a qualidade da ação A_t no estado S_t. $R(S_t, A_t)$ representa o valor do reforço da ação A_t no estado S_t depois devemos repetir os passos 2,3 e 4 até que a tabela Q tenha convergido para valores ótimos e o agente tenha explorado suficientemente o ambiente.

No Deep Q-Learnig ao invés de uma tabela Q, utilizamos uma rede neural profunda para aprender uma representação das ações e estados. No Deep Q-Learning o agente irá receber como entrada o estado do ambiente, e alimenta-o na rede neural profunda para obter uma estimativa dos valores Q para cada ação possível. O agente, então, seleciona a ação com o maior valor Q estimado (ou usa uma estratégia de exploração) e executa a ação no ambiente. Durante o treinamento, através do algoritmo de retropropagação do erro a rede neural será atualizada, nesse algoritmo a diferença entre os valores Q estimados e os Q reais será minimizada.

Logo, vemos que o Deep Q-Learning é uma versão do Q-Learning onde aplicamos uma rede neural profunda para poder ter resultados melhores, nesse trabalho foi escolhido o algoritmo DQN pois como o nosso ambiente será o joogo space invaders, o qual o agente precisa saber a localização de inimigos, dos ataques dos inimigos e de sua própria nave, como o DQN usaremos redes convolucionais para que o agente aprenda esses padrões espaciais complexos, além disso como o Q-Learning usa uma tabela para armazenar os valores de Q para cada par estado-ação, no casso de nosso ambiente não seria viavel com um grande espaço de estados e ações, já o DQN usa de redes profundas para aproximar a função Q, o que permite que o agente generalize e tome decisões eficientes

em um espaço de ações complexo.

III. Trabalhos Relacionados

Para a realização desse trabalho, foi precisso estudar o assunto a fundo, com a utilização de vários livros e trabalhos da área. O livro "Reinforcement Learning - Richard S. Sutton and Andrew G. Barto"foi de bastante ajuda em entender mais sobre Reinforcement Learning e seus fundamentos. O trabalho de "Análise de Políticas de Exploração no Aprendizado por Reforço aplicado a Jogos de Atari"pelo Antônio Carlos de Souza Junior e o trabalho de "Desenvolvimento de um Agente Inteligente para Jogar Jogos Genéricos de Atari 2600"por Lucas Antunes de Almeida, me ajudou a entender melhor como reinforcement learning é usado para aprender a jogar jogos de atari, e também falaram bastante sobre o Deep Q-Learning que é o algoritmo que está sendo implementado nesse trabalho. Porém um trabalho que realmente me chamou atenção para o Deep Q-Learning foi o trabalho de "Comparação de Desemoenho do Algoritmo Deep Q-Learning em Ambientes Simulados com Estados Contínuos", do Gabriel Colombo onde ele fala sobre o algoritmo Q-Learning e como a aplicação de redes neurais artificiais é usado com esse algoritmo formando o Deep Q-Learning.

Além desses trabalhos tiveram outros que me ajudaram a ter uma base para a aplicação desse trabalho, o site "Deep Learnig Book"me ajudou a entender melhor a história do Reinforcment Learning e sobre os algoritmos Q-Learning e Deep Q-Learning, também foi de bastante ajuda nesse quesito o trabalho "Deep Reinforcement Learning to play Space Invaders"do Nihit Desai e do Abhimanyu Banerjee e o trabalho "Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos"do Ícaro da Costa Mota. Para a aplicação do código eu me baseei no projeto do nicknochnack em que ele aplica o Deep Q-Learning, o titulo do projeto é "KerasRL-OpenAI-Atari-SpaceInvadersv0"no github e também me baseei nos artigos da Chloe Wang, o artigo "Using Deep Reinforcement Learning To Play Atari Space Invaders"o "Deep Learning At A Surface Level"e por fim o "A Look Into Neural Networks and Deep Reinforcement Learning". Através dos conhecimentos obtidos por esses trabalhos, foi possivel a implementação desse trabalho.

IV. Objetivos

O objetivo geral desse trabalho, é a implementação de um algoritmo Deep Q-Learning em um ambiente do jogo space invaders do videogame atari, verificar sua capacidade de adaptação nesse ambiente e comparar sua eficácia e comparar o resultado com o de dados de jogadores aleatórios.

V. Metodologia Experimental

Para a implementação desse trabalho será usado a a biblioteca OpenAI Gym para a criação de um ambiente para a utilização dos algoritmos de aprendizado por reforço, será usado um arquivo ROM(read only memory) para a emulação do ambiente do jogo space invaders. Usaremos o Jupyter notebook como ambiente de codificação desse trabalho, devido

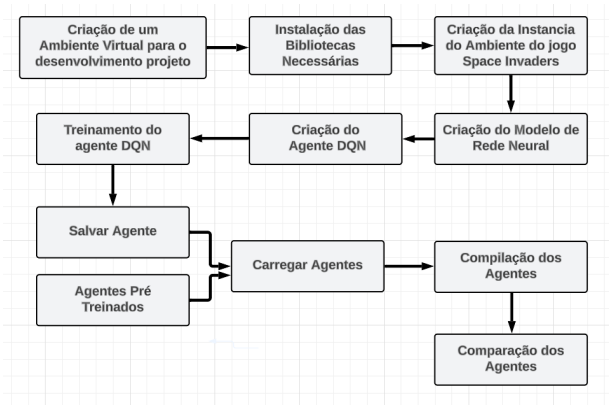

Figura 2. Diagrama de Blocos do Projeto.

sua facilidade de execução sem configuração e por sua divisão de codigos em blocos. Será usada também as bibliotecas numpy, random, tensorflow e keras-rl2.

Foi criado um ambiente virtual através do Anaconda, para que pudéssemos trabalhar com o python 3.7.15, o qual tem compatibilidades com as bibliotecas que desejavamos, que seriam o tensorflow 2.3.1, keras-rl2 1.0.4 e a gym[atari] 0.18.0, como a OpenAI Gym não inclue os roms, foi baixado o roms de um repositório online, impoortante lembrar que esse Rom está sendo usada para fins de pesquisa e não lucrativos.

A. Criação do Ambiente

Utilizando a biblioteca gym criamos o ambiente do jogo 'SpaceInvaders-v0', salvamos as dimensões para um uso posterior e as possiveis ações que o agente pode tomar no ambiente. .

B. Criação da Rede Neural

Foi criada uma função para criar nossa rede neural convolucional e densa, a qual será usada para o nosso agente, nossa função recebe como paramêtros as dimensões do ambiente e as possiveis ações do agente no ambiente, dados que salvamos no momento em que criamos o ambiente. Através da função Sequential da biblioteca Keras, para definir que nosso modelo de CNN(Convolutional Neural Network) será sequencial, depois começamos criando 3 camadas convolucionais. A primeira camada convolucional 2d com 32 filtros de tamanho 8x8 e com strides 4x4(para se mover 4 pixels na altura e largura a cada passo) definimos que a função de ativação é a ReLU (Rectified Linear Unit) especifica o formato da entrada da camada convolucional. Neste caso, temos uma entrada com 3 canais (correspondendo a uma imagem colorida RGB) e altura, largura e canais definidos pelas variáveis height, width e channels, respectivamente. Adicionamos a segunda camada que tem 64 filtros de tamanho 4x4 e com os strides de 2x2 e para nossa terceira camada convolucional adicionada tem 64 filtros de tamanho 3x3. Depois colocamos uma camada de achatamento usando a função Flatten(), ela transforma a saída da camada convolucional anterior em um vetor unidimensional, permitindo a passagem dos dados para as camadas densas.

Depois começamos a criar as camadas densas usadas para mapear os recursos extraídos pelas camadas convolucionais em ações ou em valores Q. Nossa primeira camada densa adicionada tem 512 neurônios e usamos a mesma função de ativação ReLU. Na segunda camada adiciona diminuimos os neurónios pela metada totalizando 512 e em nossa ultima camada, temos uma camada densa que será nossa camada de saida onde o total de neurônios é igual ao total de ações que o agente pode tomar, em nosso ambiente o total de ações são 6. No final da função retornamos a nossa rede neural profunda.

C. função para criar o Agente

Criamos uma função para criar o nosso agente com a utilização da função DQNAgent da biblioteca keral-rl2, ela recebe como parâmetro o modelo de rede neural que criamos mais cedo e o conjunto de ações. Em nossa primeira linha de comando da função temos a definição da politica de exploração, usamos a Epsilon-Greedy, onde o valor de Epsilon (eps) controla a taxa de exploração versus exploração, depois usamos a LinearAnnealedPolicy para reduzir gradualmente o valor de Epsilon à medida que o agente ganha experiência. Na segunda linha criamos uma memória sequencial para armazenar as transições do agente, ela é usada posteriormente no treinamento do agente para relembrar experiências passadas. Em nossa "Ultima linha"temos a criação do agente DNQ, usamos a função DQNAgent e passamos commo parâmetro o modelo de rede neural, nossa memória e a politica de exploração, também nessa função definimos que o agente jogara 10000 vezes com ações aleatórias antes de começar a aprender, essas "rodadas de aquecimento"servem para iniciar o nosso agente, ele usa esses episódios para conhecer o ambiente e coletar informações sobre recompensas associadas a diferentes ações, também no preenchimento da memória e na inicialização dos pesos de nossa rede. NO final da função retornamos o nosso agente.

D. Criando nosso Agente e Começando o Treinando

através da nossa função de criação de rede neural, criamos uma e a nomeamos de Model, logo depois criamos nosso agente DQN usando como parametro o Model e o número de parâmetros, denominamos o agente de dqn. Antes de darmos inicio ao treinamento do agente, usamos o método compile() para configurar o processo de treinamento do agente, usamos o otimizador Adam com uma taxa de aprendizado de 1e-4, esse otimizador irá ajustar o os pesos da rede neural apartir da função de perda MSE(Mean Squared Error) que é utilizada para calcular a diferença entre os valores Q estimados pelo agente e os valores Q-alvo durante o treinamento. Assim, damos inicio ao nosso treinamento usando o método fit, esse método recebe como parâmetro nosso ambiente e o número de episodios que nosso agente deve treinar, foi definido 1000000 episódios para que o nosso agente treine, além disso definimos os parâmetos visualize como falso pois assim o treinamento acontece de forma mais rápida, e verbose como 2 para termos mais detalhamentos de cada episódio treinado, quando terminado o treinamento do modelo usamos o método save.weights para salvar nosso agente e poder carregalo novamente depois.

VI. Estimação dos Resultados

Este projeto tem como objetivo implementar o algoritmo Deep Q-Learning para ensinar um agente a jogar o jogo Space Invaders. Espera-se que, à medida que o agente treinar em mais episódios (rodadas do jogo), ele melhore seu desempenho. Nosso objetivo é acompanhar e estimar a evolução do agente ao longo do tempo.

VII. Resultados/Discussões

O projeto foi implementado principalmente em um notebook do Jupyter e em um arquivo py no vscode, o jupyter foi usado por sua facilidade de executar os códigos em células individuais, porém por não ter uma interface gráfica foi usado o vscode para a visualização do jogo. É importante destacar que, devido à falta de uma placa gráfica dedicada no meu notebook, houve uma restrição no número de episódios de treinamento que pude realizar. A capacidade de treinar o agente por um grande número de episódios é geralmente beneficiada pelo uso de uma GPU, que acelera o processamento e permite uma iteração mais rápida do algoritmo de Deep Q-Learning. No entanto, devido às limitações do hardware, fui capaz de treinar o agente apenas até 64 episódios. A fim de realizar uma análise mais aprofundada e abrangente do algoritmo DQN no contexto do jogo Space Invaders, optei por utilizar pesos pré-treinados disponíveis publicamente. utilizaremos dois pesos pré-treinados nesse trabalho, um em que o agente treinou por 10 mil episódios e outro que treinou por 1 milhão de episódios.

Para esse trabalho, comparamos o desempenho do agente tomando ações aleatórias, o agente treinado por 64 episódios, pelo de 10.000 episódios e por fim pelo de 1.000.000 episódios, gravamos as pontuaçãoes de 100 partidas de cada e os analisamos. Podemos ver na figura 3 que assim como esperado, o algoritmo que passou mais tempo treinando obteve as maiores pontuações, é interessante ver também que nosso modelo treinado por 64 episódios teve um desempenho quase totalmente linear, isso acontece devido ao agente sempre ter as mesmas ações, como nosso algoritmo definou o número de "aquecimentos"(warm-up) como 10000, siginica que durante 10 mil episódios o nosso agente irá tomar ações aleatórias antes de começar a aprender com o agente DQN, durante as etapas de aquecimento, o agente DQN explora o ambiente executando ações aleatórias para coletar uma quantidade inicial de experiências. Isso é importante para preencher a memória do agente com exemplos iniciais e permitir que ele tenha uma base de dados suficiente para aprender a tomar decisões melhores. Devido ao agente ter treinado apenas por 64 episódios, ele não começou a aprender a jogar, está apenas tomando algumas ações aleatórias iguais em todas as partidas.

Na figura 3 podemos ver o desempenho dos agentes durante 100 episódios, para ter uma melhor visualização calculamos as médias das pontuações obtidas por cada um e as demostramos na figura 4, a primeira vista, podemos até pensar que nosso

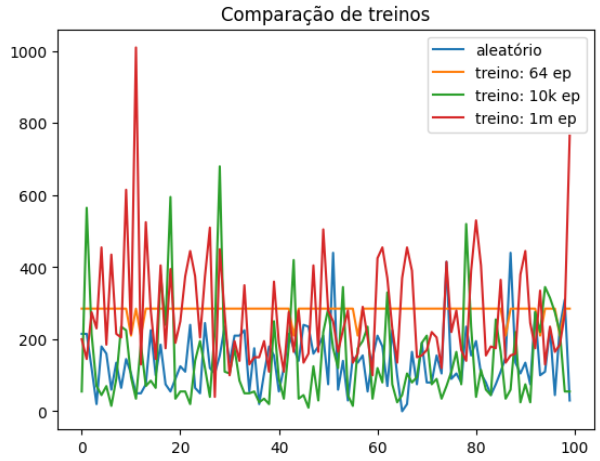

Figura 3. desempenho dos agente.

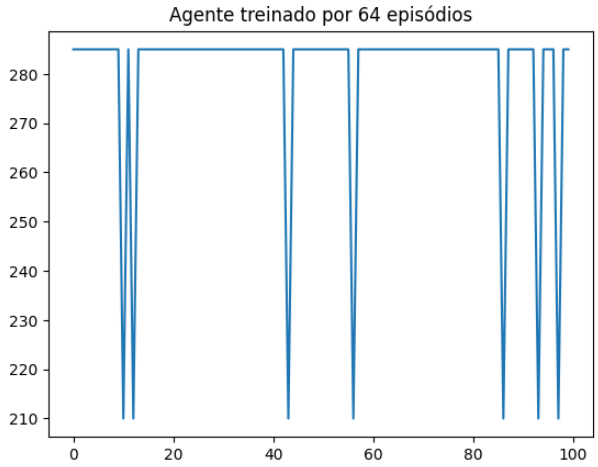

Figura 5. agente treinado por 64 episódios.

modelo treinado por menos tempo teve melhores resultados, porém isso se dá devido a constancia de suas pontuações, como esse agente sempre toma as mesmas ações logicamente ele na maioria das vezes manteria a mesma pontuação, diferente dos outros modelos que em algumas partidas tem resultados bastantes ruins e outros muito bons, logo não podemos afirmar que esse agente é o melhor pois ele tem um limite em suas pontuações, como podemos observar na imagem 5. Esse agente nunca passa de 285 pontos.

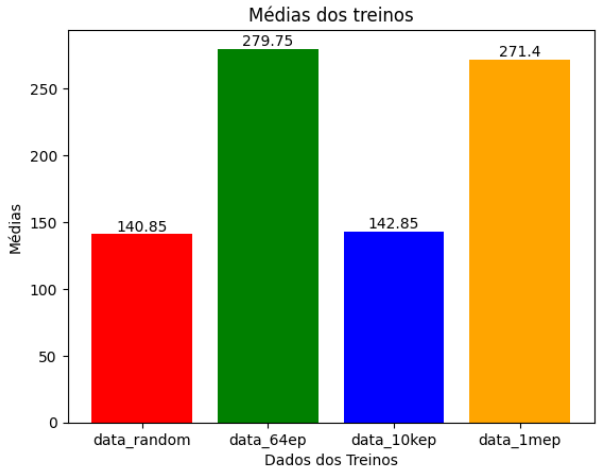

Figura 4. Média dos agentes.

Visto que a média das 100 partidas não é uma boa forma de medir a eficácia, pegamos a melhor pontuação de cada agente durante as 100 partidas, podemos visualizar no gráfico da imagem 6, nela podemos visualizar que o agente que treinou por 1 milhão de episódios conseguiu passar de 1000 pontos, algo que os outros agentes não foram capazes de fazer, podemos ver que o pior agente foi o treinado por 64 episódios. Podemos focar na grande diferença que temos do agente que treinou por 10 mil episódios para o que treinou por 1 milhão, mostra que o agente dado um maior numero de treinamento tende a melhorar muito, podemos imaginar que modelos treinados por 10 milhões ou 50 milhões de episódios poderiam ter resultados bem melhores, no site Papers with code, podemos ver um ranking com outros algortmos no jogo space invaders, nos mostra o quão longe o reinforcement learning pode chegar em jogos de Atari.

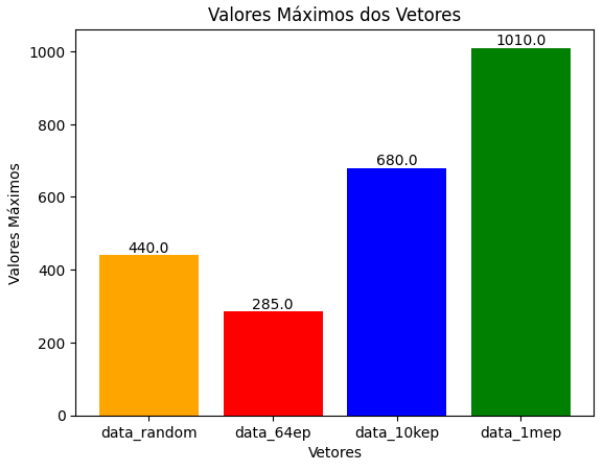

Figura 6. Melhores Pontuações.

REFERÊNCIAS

[1] SOUZA JUNIOR, Antônio Carlos. Análise de políticas de exploração no aprendizado por reforço aplicado a jogos de Atari. 2022.

[2] Livro Reinforcement Learning - Richard S. Sutton and Andrew G. Barto

[3] ICOLOMBO, Gabriel. Comparação de desempenho do algoritmo Deep Q-Learning em ambientes simulados com estados contínuos. 2022. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

[4] ALMEIDA, Lucas Antunes de. Desenvolvimento de um agente inteligente para jogar jogos genéricos de Atari 2600. 2019.

[5] ZIELINSKI, Kallil Miguel Caparroz et al. Controle flexível de sistemas a eventos discretos utilizando simulação de ambiente e aprendizado por reforço. 2021. Dissertação de Mestrado. Universidade Tecnológica Federal do Paraná.

[6] MNIH, V. et al. Human-level control through deep reinforcement learning. Nature, Nature Publishing Group, v. 518, n. 7540, p. 529, 2015.

[7] MOTA, Ícaro da Costa. Aprendizagem por reforço utilizando Q-Learning e redes neurais artificiais em jogos eletrônicos. 2018.
[8] OTTONI, André Luiz Carvalho et al. Análise da influência da taxa de aprendizado e do fator de desconto sobre o desempenho dos algoritmos Q-learning e SARSA: aplicação do aprendizado por reforço na navegação autônoma. Revista Brasileira de Computação Aplicada, v. 8, n. 2, p. 44-59, 2016.
[9] DESAI, Nihit; BANERJEE, Abhimanyu. Deep Reinforcement Learning to Play Space Invaders. Technical Report, 2017.
[10] Deep Learnig Book: https://www.deeplearningbook.com.br/.
[11] Nicknochnack. (Ano). KerasRL-OpenAI-Atari-SpaceInvadersv0. Recuperado de https://github.com/nicknochnack/KerasRL-OpenAI-Atari-SpaceInvadersv0
[12] Wang, C. (Ano). A Look into Neural Networks and Deep Reinforcement Learning. Recuperado de https://chloeewang.medium.com/a-look-into-neural-networks-and-deep-reinforcement-learning-2d5a9baef3e3
[13] Wang, C. (Ano). Deep Learning at a Surface Level. Recuperado de https://chloeewang.medium.com/deep-learning-at-a-surface-level-8565878d8b5c
[14] Wang, C. (Ano). Using Deep Reinforcement Learning to Play Atari Space Invaders. Recuperado de https://chloeewang.medium.com/using-deep-reinforcement-learning-to-play-atari-space-invaders-8d5159aa69ed