COORDENADORIA DE ENGENHARIA DA COMPUTAÇÃO

AFFONSO BRIAN PEREIRA AZEVEDO

LEONARDO ZANONE

ANÁLISE ESTATÍSTICA DE MÉTODOS DE APRENDIZAGEM DE MÁQUINA APLICADOS AO DINO GAME

Sorocaba/SP

2021

AFFONSO BRIAN PEREIRA AZEVEDO

LEONARDO ZANONE

ANÁLISE ESTATÍSTICA DE MÉTODOS DE APRENDIZAGEM DE MÁQUINA APLICADOS AO DINO GAME

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do diploma de graduação em Engenharia da Computação.

Orientador: Prof. Dr. Johannes Von Lochter

Sorocaba/SP

2021

FICHA CATALOGRÁFICA ELABORADA PELA

“BIBLIOTECA FACENS”

Bibliotecária resp.
Eliane da Rocha - CRB 8062

A994a

Azevedo, Affonso Brian Pereira.

Análise estatística de métodos de aprendizagem de máquina aplicados ao Dino Game / por Affonso Brian Pereira Azevedo, Leonardo Zanone. – Sorocaba, SP: [s.n.], 2021.

60f.

Trabalho de Conclusão de Curso (Graduação) – Centro Universitário Facens – Curso de Engenharia de Computação, 2021.

Orientador: Prof. Dr. Johannes Von Lochter

1. Inteligência Artificial. 2. Aprendizado de Máquina. 3. Aprendizado por Reforço. 4. DQN. 5. SARSA. 6. One Ste Actor Critic. I. Zanone, Leonardo. II. Centro Universitário Facens. III. Título.

CDD 621.39

AFFONSO BRIAN PEREIRA AZEVEDO

LEONARDO ZANONE

ANÁLISE ESTATÍSTICA DE MÉTODOS DE APRENDIZAGEM DE MÁQUINA APLICADOS AO DINO GAME

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do diploma de graduação em Engenharia da Computação.

Orientador: Prof. Dr. Johannes Von Lochter

Sorocaba, 23 de Dezembro de 2021

BANCA EXAMINADORA

_____

Prof. Dr. Johannes Von Lochter

_____

Prof. Dr. Marc Gonzàlez Capdevila

_____

Prof. Kleber de Jesus Dias

AGRADECIMENTOS

Agradecemos primeiramente a Deus pela oportunidade de realizar este trabalho, em seguida à nossas famílias, que sempre nos motivaram, apoiaram e fizeram com que fosse possível termos recursos suficientes para a realização do curso e deste trabalho.

Aos amigos e colegas que passaram por nossa vida, pelos aprendizados que nos proporcionaram e os momentos de alegria que tivemos juntos durante esta jornada.

Ao nosso orientador Prof. Dr. Johannes Von Lochter pela orientação e auxílio necessário para que este projeto pudesse se tornar realidade.

Aos professores do curso de Engenharia de Computação pelos ensinamentos que nos fizeram crescer como profissionais e pessoas ao longo do curso.

Por fim, agradecemos a todos que de alguma forma colaboraram para nosso desenvolvimento intelectual e fizeram com que chegássemos ao fim deste trabalho.

“Se cheguei até aqui foi porque me apoiei no ombro dos gigantes.”

Isaac Newton

RESUMO

A utilização de métodos de aprendizado de máquina para solucionar problemas vem se popularizando cada vez mais nos dias de hoje, tarefas como separação automática de alimentos em indústrias, carros autônomos e robôs aspiradores podem utilizar como parte de suas tecnologias este tipo de algoritmo. Com a popularização desta abordagem surge a dúvida sobre quais algoritmos utilizar em determinados casos. Esta pesquisa tem como intuito auxiliar a sanar estas dúvidas aplicando três métodos de aprendizado por reforço a um estudo de caso utilizando o jogo Dino Game como ambiente de teste. Os métodos aplicados neste projeto são DQN, *One Step Actor Critic* e SARSA, métodos popularmente conhecidos e que podem facilmente ser aplicados a este problema. Com a aplicação destes métodos foi possível coletar dados de execução e gerar métricas para criar comparações e correlações entre os resultados e desempenho obtidos por cada um deles. Por fim foi possível observar em ambiente como o do Dino Game o método DQN possuí um potencial de acerto maior em sua tomada de decisões, embora acabe consumindo mais recursos para ser treinado.

Palavras-Chave: Inteligência Artificial. Aprendizado de Máquina. Aprendizado por Reforço. DQN. SARSA. One Ste Actor Critic.

ABSTRACT

The use of machine learning methods to solve problems has become more and more popular these days, tasks such as automatic food separation in industries, autonomous cars and vacuum robots can use this type of algorithm as part of their technologies. With the popularization of this approach, the question arises about which algorithms to use in certain cases. This research, in order to help resolve these doubts, applies three reinforcement learning methods to a case study using the Dino Game as a test environment. The methods applied in this project are DQN, One Step Actor Critic and SARSA, popularly known methods that can easily be applied to this problem. With the application of these methods, it was possible to collect execution data and generate metrics to create comparisons and correlations between the results and performance obtained for each one of them. Finally, it was possible to observe in an environment such as the Dino Game, the DQN method has a greater potential for success in its decision-making, although it ends up consuming more resources to be trained.

Keywords: Artificial Intelligence. Machine Learning. Reinforcement Learning. DQN. SARSA. One Ste Actor Critic.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AI	*Artificial Intelligence*
DQN	*Deep Q-Network*
IA	Inteligência Artificial
Mb	Megabytes
RAM	*Random Access Memory*
RL	*Reinforcement Learning*
SARSA	*State Action Reward State Action*

SUMÁRIO

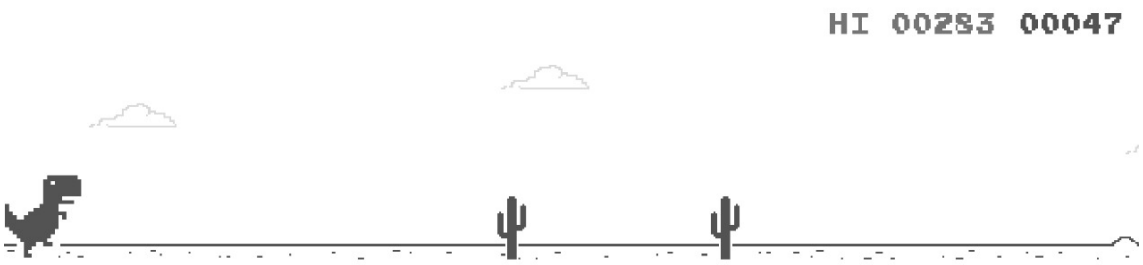
1 INTRODUÇÃO

Nos dias atuais, vem crescendo o número de aplicações que utilizam algum tipo de inteligência artificial em seus desenvolvimentos, por exemplo como mencionado por Fábio Caversan: "*In recent years, AI usage has exploded in pharma, health care and biotech*", (Caversan, 2020). A inteligência artificial é uma área da computação que abrange várias outras áreas, como visão computacional, processamento de linguagem natural, aprendizado de máquina, etc. Cada uma dessas áreas usa métodos para facilitar e automatizar algum processo que normalmente necessita de trabalho humano.

Na maioria dos casos, existe mais de uma tecnologia possível para ser utilizada no desenvolvimento da aplicação, podendo ter resultados melhores ou piores dependendo do cenário. Esta pesquisa tem por objetivo mostrar algumas maneiras diferentes de solucionar um problema em um cenário específico e demonstrar as diferenças entre essas diferentes soluções, tentando encontrar alguma relação entre os métodos e os resultados.

O caso de estudo foi o "jogo do dinossauro", um jogo do tipo *arcade* que pode ser acessado através do navegador Google Chrome quando não há acesso à internet. No jogo há um dinossauro que ao começar uma partida, começa a correr para a direita que pode ser visto na Figura 1, porém na realidade quem se move é o cenário ao fundo dando a impressão de que o dinossauro está correndo. No percurso, aparecem obstáculos como cactos e pterodátilos e o objetivo do jogador é desviar desses obstáculos pulando-os ou agachando. Conforme o tempo vai passando, a velocidade do dinossauro aumenta, deixando mais difícil desviar dos obstáculos. O objetivo do jogo é fazer a maior pontuação possível, ou seja, chegar o mais longe possível, sendo que o jogo acaba quando o dinossauro toca em algum dos obstáculos.

Figura 1 – Dinossauro correndo no Dino Game



Fonte: autoria própria

Conhecendo o cenário, foram escolhidos métodos de aprendizado por reforço, que é um dos paradigmas de aprendizado de máquina. Para cada método foram coletados dados não só como pontuação máxima que é um dado que parece óbvio para dizer que um modelo é superior a outro. Foram também coletados dados como: tempo de treino, pontuação média, tamanho do modelo etc. Com esses dados é possível ter uma ideia mais abrangente sobre cada método para entender como cada uma dessas métricas influencia no resultado final.

Para alguém que está começando na área de inteligência artificial pode ser complicado saber qual método utilizar e por onde começar. Essa pesquisa pode ajudar essas pessoas a terem uma noção melhor de como cada método funciona e quais as vantagens e desvantagens nesse cenário, que pode se replicar em seu cenário.

Apesar desse cenário parecer muito específico, existem muitos casos de aplicação que utilizam a mesma lógica, onde é necessário tomar uma decisão a partir de algumas entradas dadas. Por exemplo, um carro autodirigido, que não pode bater em outros carros, pedestres e outros obstáculos na pista. Ou um aspirador automático que anda pela casa e deve ser capaz de identificar paredes, escadas e móveis para não ficar travado. Esses são alguns exemplos

que seguem uma lógica parecida com a do Dino Game e poderiam se beneficiar com essa pesquisa.

No segundo capítulo é tratado sobre a contextualização de inteligência artificial. No terceiro capítulo é abordado os três métodos de aprendizado por reforço estudados nesse trabalho, sendo eles: DQN ou *Deep Q-Network* (Rede-Q Profunda), SARSA ou *State Action Reward State Action* (Estado Ação Recompensa Estado Ação) e *One Step Actor Critic* (Um Passo Ator Crítico). No quarto capítulo é comentado sobre a implementação dos algoritmos. No quinto capítulo são exibidos os resultados obtidos. Por fim o capítulo 6 trás a conclusão e trabalhos futuros.

2 INTELIGENCIA ARTIFICIAL

Inteligência artificial pode ser definida como o esforço para a automatização de tarefas normalmente realizadas por seres humanos (CHOLLET, 2018).

A área de inteligência artificial é de certa forma genérica e engloba áreas mais específicas, tais como aprendizado de máquina e *deep learning* (aprendizado profundo), entretanto ela engloba também outras abordagens que não possuem nada relacionado a aprendizado (CHOLLET, 2018).

Através dos milênios filósofos se depararam e se atormentaram com algumas perguntas difíceis de serem respondidas, tais como: o que é a mente? O que é o pensamento? E com essas dúvidas e o avanço científico e computacional foi possível chegar a era da inteligência artificial, onde há um grande esforço para fazer com que as máquinas pensem e resolvam problemas (HAUGELAND, 1989).

2.1 História

O desenvolvimento da inteligência artificial iniciou-se nos anos 1950 quando um grupo de pioneiros de ciência de dados tiveram a ideia de fazer computadores "pensarem" (CHOLLET, 2018).

As primeiras implementações de inteligência artificial utilizadas em jogos de xadrez não utilizavam nenhum tipo de aprendizado, na verdade eram um conjunto de várias regras pré-definidas por programadores de forma a simular um comportamento que fizesse sentido do ponto de vista humano (CHOLLET, 2018)

Por muito tempo especialistas da área de inteligência artificial acreditaram que seria possível criar uma inteligência comparável à inteligência humana utilizando somente de um grande conjunto de regras pré-estabelecidas para a manipulação de conhecimento. Este tipo de abordagem ficou conhecido como *symbolic AI* (Inteligência Artificial simbólica) e foi predominante entre os anos 1950 e 1980 (CHOLLET, 2018).

Apesar da abordagem de *symbolic AI* ser bem efetiva em contextos bem definidos tais como jogos de xadrez, ela não se provou tão eficaz em contextos que não possuem uma regra tão clara como classificação de imagens, reconhecimento de linguagem falada e tradução de texto entre línguas, uma nova abordagem então surgiu para tomar o lugar da *symbolic AI*, o aprendizado de máquina (CHOLLET, 2018).

2.2 Áreas de aplicação de IA

Existem várias áreas em que a inteligência artificial pode ser aplicada, podendo ser implementados de maneiras diferentes, alguns exemplos de áreas são: robótica, visão computacional, aprendizado de máquinas, processamento de linguagem natural. (NEBEL; LAKEMEYER, 2003).

2.3 Aprendizado de máquina

Os algoritmos de *machine learning* (aprendizado de máquina) podem ser classificados em quatro categorias: aprendizado supervisionado, não supervisionado, semi-supervisionado e por reforço (KANG, 2018).

Para esse estudo o foco será o aprendizado por reforço, tendo em vista que para o cenário escolhido é o que faz mais sentido pela maneira como serão feitas as entradas e tomadas de decisões.

2.3.1 Algoritmos

Atualmente existem vários algoritmos para *machine learning*, cada um com suas vantagens e desvantagens.

Alguns exemplos são as redes neurais e algoritmos genéticos.

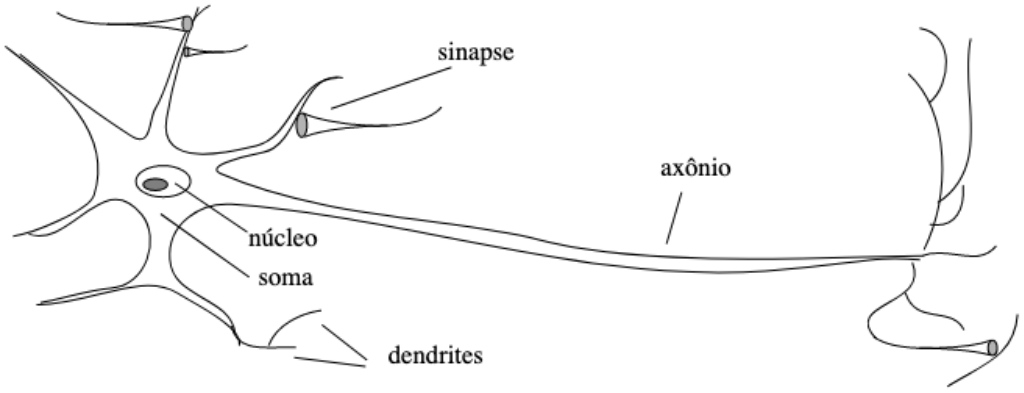
2.3.1.1 Redes Neurais

Como muitas das invenções humanas, as redes neurais artificiais utilizam como base a natureza, neste caso a natureza das redes neurais presentes no cérebro do ser humano.

Observando o cérebro humano, é possível visualizar certas qualidades que o auxiliam na resolução de problemas, sendo algumas delas a tolerância a falhas, permitindo que a eliminação de alguns neurônios não comprometa a funcionalidade total do cérebro, a capacidade de aprendizagem de tarefas nunca antes realizadas, o entendimento de informações mesmo quando não estão completas e o paralelismo de processamento das informações (RAUBER, 2005).

O neurônio é uma célula onde reações químicas e elétricas realizam o processamento de informação. O neurônio é constituído por um núcleo e um corpo chamado de soma. A saída da informação de um neurônio ocorre através de impulsos elétricos propagados através do axônio, no fim do axônio existem diversas ramificações que distribuem a informação para outros neurônios através das sinapses. Sinapses são as ligações realizadas entre o axônio do neurônio que transmite a informação e o dendrito do neurônio que recebe a informação. A Figura 2 apresenta de forma simplificada a estrutura de um neurônio biológico (RAUBER, 2005).

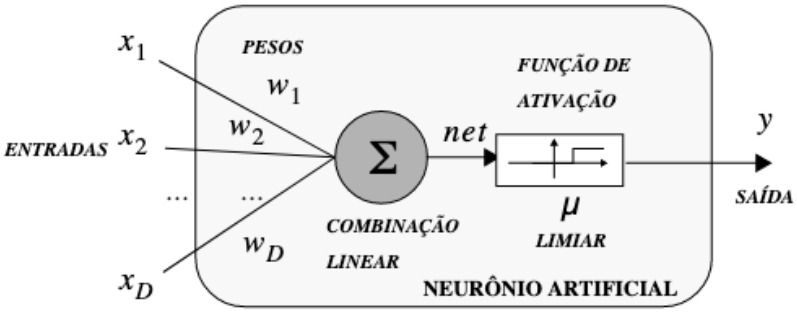
Figura 2 – Neurônio biológico



Fonte: RAUBER, (2005, p. 4).

Um modo de representação de um neurônio artificial é o modelo descrito por McCulloch e Pitts no ano de 1943 no livro *A Logical Calculus of the Ideas Immanent in Nervous Activity* (Cálculo Lógico de Ideias Inerentes a Atividade Nervosa) conforme a Figura 3. Este modelo tenta simular as atividades de um neurônio biológico. Seu funcionamento é bem simples, o neurônio possui *D* entradas *X* e *D* pesos *W*. Sendo assim a somatória da multiplicação das entradas por seus respectivos pesos gera o valor *net*. O valor *net* por sua vez é inserido em uma função de ativação e caso o valor de *net* seja superior ao limiar *μ* a função de ativação irá retornar com o valor 1, caso contrário o valor de y será 0 (RAUBER, 2005).

Figura 3 – Modelo de um neurônio de McCulloch e Pitts



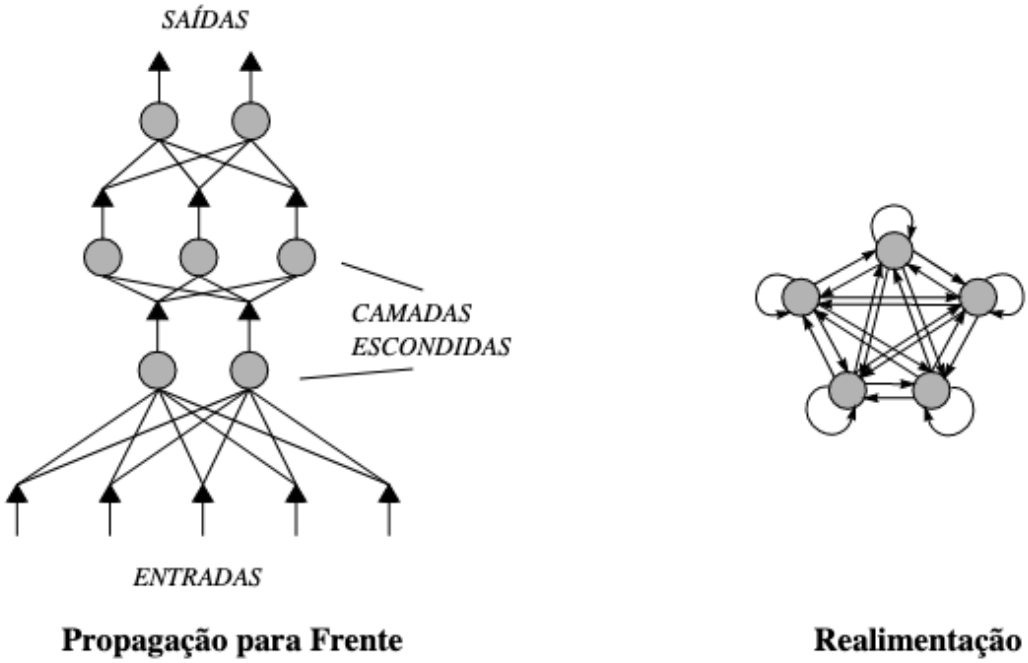
Fonte: RAUBER, (2005, p. 6)

Uma rede neural é o conjunto de vários neurônios interligados entre si, formando assim camadas de processamento.

Segundo RAUBER (2005, p. 6): "O potencial e flexibilidade do cálculo baseado em redes neurais vêm da criação de conjuntos de neurônios que estão interligados entre si. Esse paralelismo de elementos com processamento local cria a “inteligência” global da rede. Um elemento da rede recebe um estímulo nas suas entradas, processa esse sinal e emite um novo sinal de saída para fora que por sua vez é recebido pelos outros elementos".

Podemos visualizar as principais topologias de redes neurais na Figura 4, uma delas propagando as entradas de uma camada para outra e a outra fazendo a própria realimentação entre seus neurônios.

Figura 4 – Principais topologias de redes neurais artificiais



Fonte: RAUBER, (2005, p. 7)

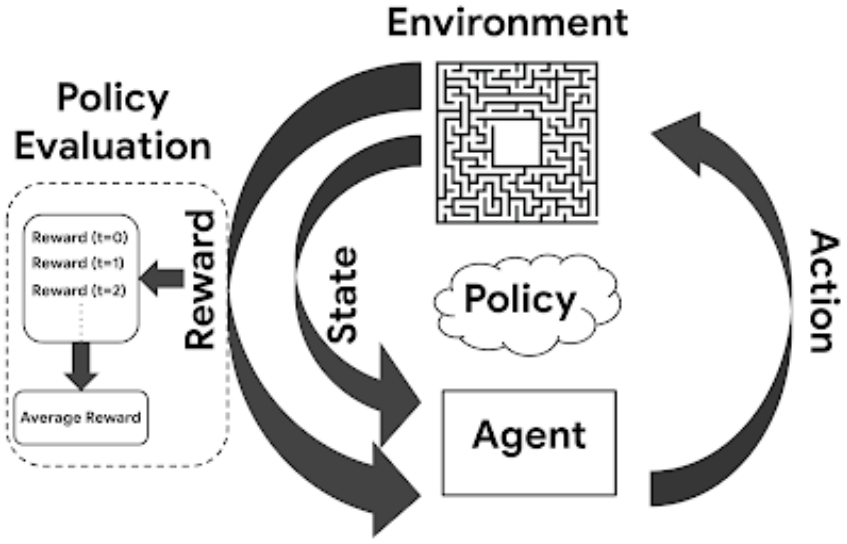
2.3.2 Aprendizado por reforço

O aprendizado por reforço tem mostrado resultados impressionantes em uma variedade de domínios como jogos de estratégia, robótica, etc. (YAGHMAIE, 2021).

No *reinforcement learning* (aprendizado por reforço) ou *RL* típico, o modelo do sistema é desconhecido e foca em aprender como reagir com o sistema e otimizar a performance (YAGHMAIE, 2021).

Ela se baseia na ideia de um agente que realiza ações no ambiente que podem mudar o seu estado e para cada ação, esse agente recebe uma recompensa podendo ser positiva ou negativa, para que o agente saiba que está tomando as decisões mais adequadas para a solução do problema, como ilustrado na Figura 9.

Figura 5 - Ilustração de aprendizado de máquina



[1]Fonte: Disponível em: https://ai.googleblog.com. Acesso em: 2 maio 2021

Existem três maneiras de abordar um problema de *RL*: *Dynamic programming* (programação dinâmica), *policy gradient* (gradiente de política) e *model-building RL* (construção de modelo) (YAGHMAIE, 2021).

[1] Imagem retirada de: https://ai.googleblog.com/2020/04/off-policy-estimation-for-infinite.html

3 MÉTODOS DE APRENDIZADO POR REFORÇO

O processo de aprendizado para agentes que lidam diretamente com entradas de sensores de larga escala como visão e reconhecimento de voz é um dos maiores desafios para o estudo de aprendizado por reforço (*reinforcement learning*). (Volodymyr et al. 2013).

Neste capítulo serão apresentados três diferentes métodos de aprendizado por reforço que serão utilizados no Dino Game. Os métodos apresentados terão acesso às ações que podem ser utilizadas no jogo, a imagem capturada da tela e a recompensa obtida após uma determinada ação, assim como aconteceria com um jogador humano.

Nos algoritmos apresentados há a existência de uma rede neural (agente) que faz escolhas e se comunica com o ambiente (jogo), desse modo aprendendo com a tomada de decisões ao longo do tempo.

3.1 SARSA

Esse algoritmo foi chamado inicialmente de *Modified Connectionist Q-Learning* (Aprendizado-Q “Coneccionista” Modificado) também ficou conhecido como *State Action Reward State Action* (Estado Ação Recompensa Estado Ação) ou apenas SARSA, sugerido por Rich Sutton, pois é exatamente o processo realizado pelo algoritmo a cada atualização (Rummery et al. 1994).

Como o nome original dizia, esse algoritmo é uma versão modificada do algoritmo *Q*-Learning, seguindo a mesma estrutura de algoritmo e apenas modificando a forma de atualizar o estado a cada iteração. No algoritmo de *Q-Learning* é utilizado o valor máximo entre os estados.

No início do treinamento, os valores de ações da função Q que ainda não foram exploradas são muito prováveis de estarem erradas e mesmo nos estágios finais do treino, utilizar o valor máximo é uma superestimação do valor real de retorno (Rummery et al. 1994).

A principal diferença desse método é a lógica de atualização, que segue a fórmula $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$. No Algoritmo 1 é possível verificar um pseudocódigo de implementação do algoritmo SARSA.

Algoritmo 1 – Pseudocódigo da implementação do SARSA

```
inicialize a tabela de estado-valor
para cada episódio de 1 até M faça:
    inicialize a sequencia S1
    escolha uma ação At do estado S1 usando uma política de seleção
    para cada t de 1 até T faça:
        execute a ação At
        armazene a recompensa Rt e o novo estado St+1
        escolha uma ação At+1 do estado St+1 usando uma política de seleção
        atualize a tabela de estado-valor seguindo a fórmula do SARSA
        armazene o novo estado obtido em St
        armazene a nova ação obtida em At
    fim para
fim para
```

Outro fator importante desse algoritmo é a política de seleção, podendo mudar de implementação para implementação. Rummery e Niranjan citam o método *ε-greedy* (guloso) onde na maioria dos casos a ação de maior recompensa é escolhida, porém há uma ε chance de outra ação ser escolhida aleatoriamente.

3.2 DQN

DQN, conhecido também como *Deep Q-Network* (Rede-Q Profunda) é um método de RL que utiliza a técnica conhecida como *experience replay* (experiência de repetição) armazenando as experiências obtidas pelo agente em uma *replay memory* (memória de repetição) que é utilizada para realizar futuras decisões do agente. (Volodymyr et al. 2013).

O método utiliza uma implementação do *Q-Learning* (aprendizado-Q) na qual existe uma função de aproximação para calcular a melhor ação para determinado estado. Normalmente algoritmos que utilizam o método de Q-Learning implementam uma matriz de estados e ações, onde é calculado as melhores ações para cada estado, entretanto em problemas que possuem uma grande quantidade de possíveis estados e ações o método de armazenar os valores em uma matriz torna-se custoso, portanto, a utilização de uma função de aproximação que possua um resultado próximo a uma matriz é necessária (Volodymyr et al. 2013).

No pseudocódigo apresentado no algoritmo 2 é possível observar (de forma abstraída) o funcionamento do algoritmo DQN. O primeiro passo para o funcionamento do algoritmo é a inicialização da memória de replay e da rede neural. Em seguida deve ser inicializado um *loop* (laço) externo que será executado M vezes, sendo M o número de episódios de treinamento.

Dentro do *loop* externo é inicializado a sequência de imagens S_1, contendo somente a imagem inicial X_1. Em seguida é realizado o pré-processamento da sequência de imagens S_1 e o resultado é armazenado em Q_1. Logo após é criado um *loop* interno que é executado T vezes, sendo T o número de iterações de tomadas de decisão que serão realizadas até a finalização daquele episódio (este número geralmente é desconhecido, sendo assim geralmente este *loop* interno é implementado como um *while*), dentro do *loop* é gerado um valor aleatório V em um intervalo de zero à um, se o valor for maior ou igual à E (sendo E a uma constante para definir a probabilidade de decisões aleatórias para a exploração do algoritmo) é selecionado uma ação aleatória, caso contrário é utilizado o método de predição da rede neural para escolher a melhor ação para aquele estado, essa variação entre utilizar um valor conhecido e explorar uma nova possibilidade possibilita que o algoritmo aprenda novos modos de enfrentar um estado específico. Após selecionada a ação é então executada no ambiente e armazena-se a imagem de retorno em X_t e a recompensa em R_t. A sequência S_{t+1} é criada, pré-processada e armazenada em Q_{t+1}. Em seguida a transição composta por Q_t, A_t, R_t, e Q_{t+1} é armazenada na memória de repetição D. Por fim é calculado o método do gradiente de forma a

otimizar a rede neural para as futuras escolhas de ações. O processo se repete até que ambos os *loops* (laços) terminem.

Algoritmo 2 – Pseudocódigo da implementação do DQN

```
inicialize a memória de replay D com capacidade N
inicialize a rede neural de função Q com pesos aleatórios
para cada episódio de 1 até M faça:
    inicialize a sequencia S1 como sendo um conjunto contedo X1
    preprocesse a sequencia S1 e armazene o resultado em Q1
    para cada t de 1 até T faça:
        gere um valor aleatório de 0 à 1 e armazene em V
        se o valor calculado V for maior ou igual a probabiolidade E:
            selecione uma ação At
        senão:
            selecione a ação At de acordo com o resultado da rede
        execute a ação At
        armazene a nova imagem da tela em Xt
        armazene a recompensa obtida em Rt
        salve o estado St+1 como sendo (St, At, Xt+1)
        preprocesse St+1 e armazene em Qt+1
        armazene a transição (Qt, At, Rt, Qt+1) na memória de replay D
        calcule o método do gradiente (função de otimização)
    fim para
fim para
```

3.3 One Step Actor Critic

Esta implementação do algoritmo *One Step Actor Critic* (Um Passo Ator Crítico) está utilizando um método do tipo *off-policy* (política desligada).

Os métodos que utilizam a estratégia *off-policy* tem suas ações escolhidas seguindo uma política não relacionada a rede que está sendo calculada ou até mesmo de forma aleatória. Diferentemente, nos métodos *on-policy* (política

ligada) as ações tomadas pelo ator são escolhidas necessariamente pela política adotada pelo algoritmo. (SAGAR, 2020).

Primeiramente o algoritmo inicia a rede de política com valores aleatórios, em seguida inicializa a função de estado-valor, ambas com valores aleatórios. Em seguida é criado um laço de repetição que vai de 1 à M, sendo M o valor de episódios de treino, dentro deste loop o estado S_1 é inicializado com o valor presente na tela. Em seguida o loop interno é criado, este loop vai de 1 até T, sendo T o número de ações tomadas até o fim daquele episódio, como este valor geralmente é desconhecido geralmente implementa-se este loop como sendo um *while* (enquanto) até que o estado seja terminal. Dentro do loop interno é executada a função de estado-valor passando o estado S_t, o resultado obtido é então armazenado na ação A_t. Em seguida a ação A_t é executada no ambiente e o novo estado é salvo em S_{t+1}, enquanto a recompensa é salva em R_t.

Os valores obtidos durante essa iteração (S_t, S_{t+1} e R_t) são então utilizados para atualizar a rede de política e a rede de estado valor, os *loops* então se repetem até que sejam terminados todos os episódios necessários.

A Implementação do algoritmo em pseudocódigo pode ser vista no algoritmo 3.

Algoritmo 3 – Pseudocódigo da implementação do *One Step Actor Critic*

```
inicialize a rede de política com valores aleatórios
inicialize a função de estado-valor com valores aleatórios
para cada episodio de 1 até M faça:
    inicialize o estado S1
    para cada t de 1 até T:
        execute a função estado-valor passando o estado St
        armazene o resultado na ação At
        execute a ação At no ambiente
        armazene o novo estado em St+1
        armazene a recompensa em Rt
        atualize a função de estado-valor
        atualize a rede de política
    fim para
fim para
```

4 IMPLEMENTAÇÃO

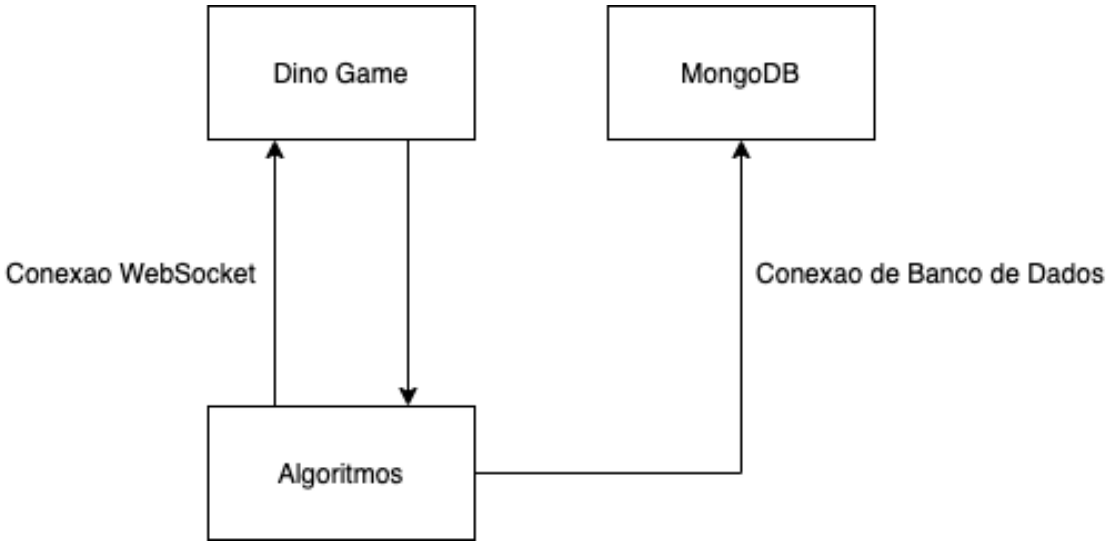
Para o desenvolvido deste projeto de pesquisa foi utiliza a linguagem de programação Python em sua versão 3.8.10. As bibliotecas utilizadas para a implementação dos algoritmos de aprendizado de máquina foram PyTorch 1.9.0 para criação das redes neurais, Pillow 8.3.1 para o processamento de imagens, PyMongo 3.12.0 para acessar leitura e escrita ao banco de dados e Pandas 1.3.2 para salvar os dados brutos em CSV.

Foi utilizada uma cópia do jogo Dino Game retirada da implementação presente no navegador Chromium, foram então adicionados alguns métodos para que fosse possível estabelecer a comunicação entre as redes neurais e o ambiente através de uma conexão WebSocket.

A figura 10 mostra como foi feita as conexões entre as partes, os algoritmos de aprendizado de máquina se comunicam com o ambiente do Dino Game através de uma conexão WebSocket, podendo assim executar ações no ambiente e coletar informações do jogo.

Para a extração de dados durante a execução os algoritmos criam uma coleção dentro do banco de dados MongoDB salvando as informações de cada iteração até o fim da execução do algoritmo, estes dados são coletados para a utilização em analises posteriormente.

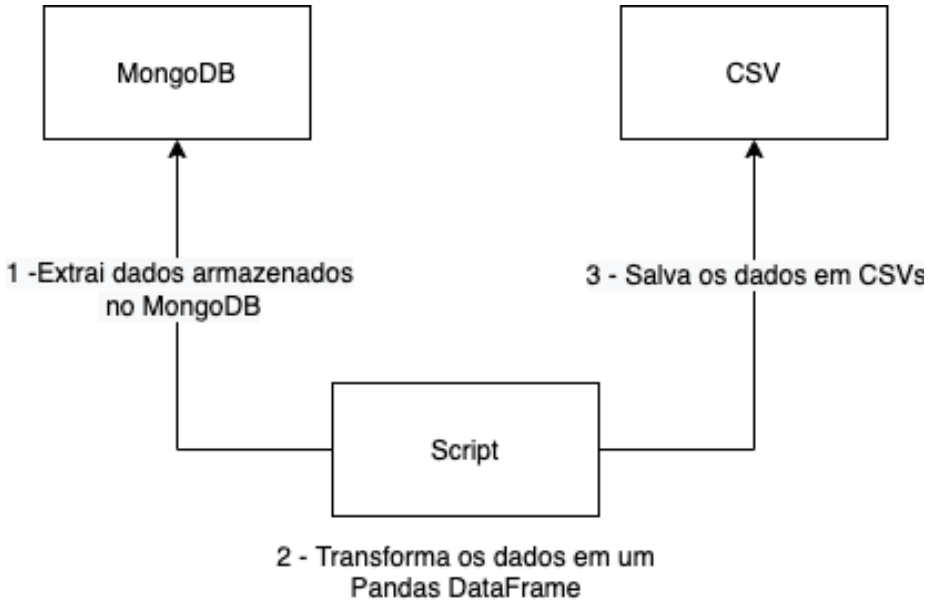
Figura 6 – Arquitetura de conexões do projeto



Fonte: autoria própria

Utilizando Python e pandas foi implemento um script que se conecta ao MongoDB, extrai os dados de cada coleção e transforma as tabelas em arquivos do tipo CSV de forma a possibilitar a visualização dos dados no Excel para realizar analises como pode se ver na Figura 11.

Figura 7 - Funcionamento do script de extração de dados



Fonte: autoria própria

5 RESULTADOS

Os resultados foram obtidos seguindo o protocolo de utilizar a mesma máquina para ter uma comparação justa de tempo, bem como a quantidade de episódios, para comparação justa de desempenho de aprendizado.

Cada algoritmo foi executado por 1000 episódios. O ambiente utilizado tem as seguintes configurações:

- Intel Core i5-9300H (2,40GHz x 8)

- 8 Gb Memoria RAM (DDR 4)

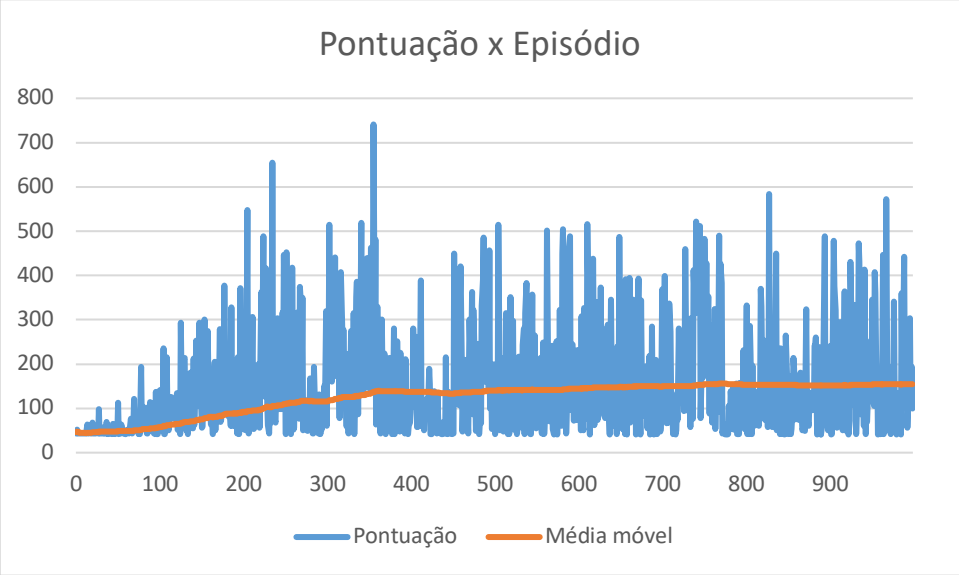
- Linux Ubuntu 20.04.3 LTS

O primeiro parâmetro analisado foi a pontuação a cada episódio, ou seja, quantos pontos o algoritmo conseguiu fazer antes que o dinossauro colidisse com um objeto e obtivesse *game over* (fim de jogo).

5.1 Pontuação

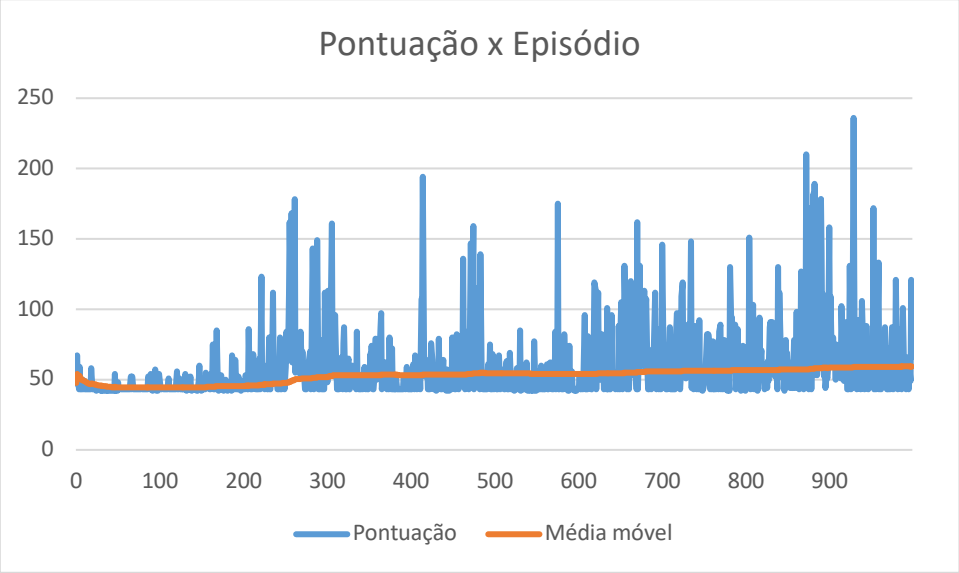
As Figuras 12, 13 e 14 exibem respectivamente a pontuação obtida por episodio dos algoritmos DQN, *One Step Actor Critic* e SARSA.

Figura 8 - Pontuação do DQN por episódio



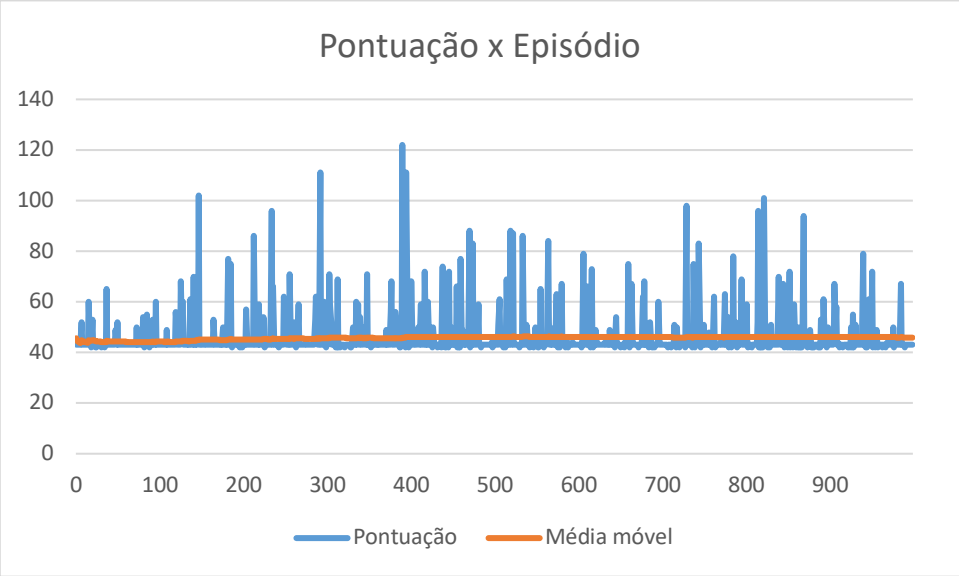
Fonte: autoria própria

Figura 9 - Pontuação do *One Step Actor Critic* por episódio



Fonte: autoria própria

Figura 10 - Pontuação do SARSA por episódio



Fonte: autoria própria

Para facilitar a compreensão destes resultados, as informações obtidas foram condensadas na Tabela 1 que contém as pontuações mínimas, médias e máximas de cada algoritmo.

Tabela 1 - Pontuação dos algoritmos

Algoritmo	**Pontuação mínima**	**Pontuação média**	**Pontuação máxima**
DQN	41	154	741
One Step Actor Critic	42	59	236
SARSA	42	45	122

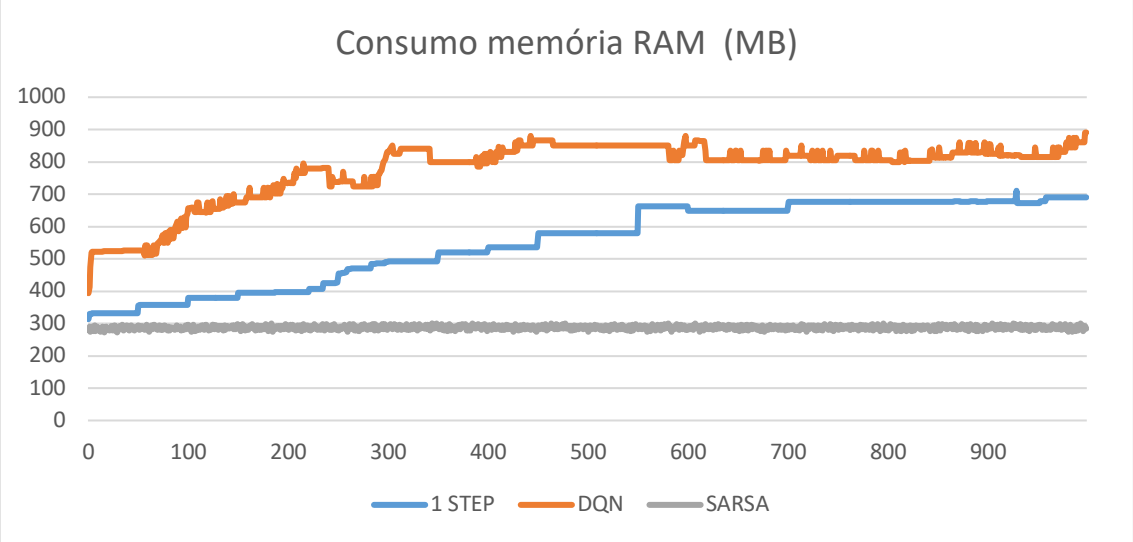
Fonte: autoria própria

Utilizando a Tabela 1 é possível observar que o DQN conseguiu uma pontuação máxima maior que os outros algoritmos, além disso sua pontuação média também é superior em relação aos algoritmos testados.

5.2 Memória Consumida

O próximo parâmetro analisado foi a memória consumida durante a execução dos algoritmos, o gráfico da Figura 15 mostra o consumo de memória RAM medida em megabytes para os três algoritmos utilizados.

Figura 11 - Gráfico de consumo de memória RAM



Fonte: autoria própria

Para facilitar o entendimento, os dados da Figura 15 foram transformados na Tabela 2 que sumariza os principais pontos de consumo de memória.

Tabela 2 - Consumo de RAM em Mb

Algoritmo	**RAM mínima (Mb)**	**RAM média (Mb)**	**RAM máxima (Mb)**
DQN	297	776	891
One Step Actor Critic	290	554	711
SARSA	272	288	300

Fonte: autoria própria

Utilizando os dados apresentados na Figura 15 e na Tabela 2 é evidente que o algoritmo que necessitou de menos memória para sua execução foi o SARSA tendo uma média de uso de 288Mb. Além disso o SARSA apresentou um uso de memória mais constante, sendo próximo de uma linha reta, enquanto os outros dois algoritmos apresentaram um crescimento considerável no uso de memória durante a execução.

O algoritmo que mais utilizou memória RAM foi o DQN, com uma média de 776Mb e um pico de 891Mb.

O aumento de memória RAM observado dá-se por conta da memória de *replay*, esse aumento é observado no DQN e no *One Step Actor Critic* por conta da estratégia *off-policy*. Já o SARSA por utilizar uma estratégia *on-policy*, não precisa manter os estados anteriores e assim mantendo o consumo de memória muito mais estável.

5.3 Tempo de Execução

O último parâmetro observado foi o tempo de execução, entre os dados obtidos estão o tempo total, o tempo efetivamente utilizado para jogar, o tempo não jogado (que inclui todo tempo gasto fora do jogo) e a proporção de tempo não jogado em relação ao tempo total. Os dados obtidos estão sintetizados na Tabela 3.

Tabela 3 - Tempos de execução

Algoritmo	**Tempo total (min)**	**Tempo jogado (min)**	**Tempo não jogado (min)**	**Proporção de tempo não jogado (%)**
DQN	276	256	20	7%
One Step Actor Critic	130	106	25	19%
SARSA	101	83	18	18%

Fonte: autoria própria

Utilizando a Tabela 3 é possível observar que o algoritmo que passou a menor quantidade de tempo fora jogo em proporção ao tempo total foi o DQN, enquanto os outros algoritmos apresentaram tempos proporcionais muito próximos. Além disso os tempos total e jogado do DQN são expressivamente maiores que dos outros algoritmos.

Contudo, é possível perceber uma grande vantagem na utilização do DQN pois o mesmo apresenta um resultado muito superior que os outros em questão de pontuação e eficiência no tempo de treino. O único ponto observado como negativo é o seu maior consumo de memória em relação aos outros dois algoritmos.

6 CONCLUSÃO

O objetivo do projeto de realizar a comparação entre diferentes métodos de aprendizado por reforço foi bem-sucedido. Implementando diferentes algoritmos (DQN, SARSA e *One Step Actor Critic*) e coletando dados durante suas execuções, foi possível obter as métricas necessárias para compará-los e chegar a uma conclusão de qual algoritmo obteve o melhor desempenho no problema proposto, o Dino Game.

Apesar de utilizar pouca memória RAM durante sua execução, uma média de 288Mb e uma máxima de 300Mb, o SARSA apresentou um baixo desempenho em sua pontuação se comparado com os outros algoritmos, tendo uma média de 45 pontos e máxima de 122 pontos. O *One Step Actor Critic* apesar de conseguir uma pontuação maior, obteve uma média de 59 pontos e máxima de 236 pontos, entretanto o algoritmo acaba gastando consideravelmente mais RAM do que o SARSA, com uma média de 554Mb e uma máxima de 711Mb. O DQN acabou apresentando-se como a melhor opção no estudo de caso realizado, tendo um consumo de RAM um pouco maior que o do *One Step Actor Critic*, com uma média de 776Mb e uma máxima de 891Mb, entretanto esse consumo maior de RAM teve bom proveito, já que o algoritmo obteve uma pontuação consideravelmente maior que outros, tendo uma média de 154 de pontos e máxima de 741 pontos, além disso o aproveitamento de tempo, ou seja o tempo gasto fora das tomadas de decisão no jogo foi menor no DQN, com 7% de tempo de processamento fora do jogo, enquanto SARSA e *One Step Actor Critic* apresentam respectivamente 18% e 19% de tempo de processamento fora do jogo.

Para pesquisas futuras é possível analisar outros algoritmos como A2C, A3C e algoritmos genéticos. É possível alterar os tipos de entradas de dados, métricas ou utilizar GPU no processamento para verificar diferentes comportamentos. Também pode-se aplicar os mesmos algoritmos em outros cenários para observar novos resultados.

REFERÊNCIAS

CHOLLET, François. **Deep Learning with Python.** 1. Ed. Shelter Island: Manning, 2018.

HAUGELAND, John. **Artificial Intelligence:** The Very Idea. 1. ed. [*S. l.*]: MIT Press, 1989.

KANG, Myeongsu; JAMESON, Noel Jordan. Machine Learning: Fundamentals. **Prognostics and Health Management of Electronics: Fundamentals, Machine Learning, and the Internet of Things**, p. 85-109, 2018.

MNIH, Volodymyr et al. Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013.

NEBEL, Bernhard; LAKEMEYER, Gerhard (ed.). **Exploring Artificial Intelligence in the New Millennium**. 1. ed. São Francisco, CA, Estados Unidos: Morgan Kaufmann Publishers, 2003.

RAUBER, Thomas Walter. Redes neurais artificiais. **Universidade Federal do Espírito Santo**, 2005.

RUMMERY, Gavin A.; NIRANJAN, Mahesan. **On-line Q-learning using connectionist systems**. Cambridge, England: University of Cambridge, Department of Engineering, 1994.

SAGAR, Ram. **On-Policy VS Off-Policy Reinforcement Learning**. [*S. l.*], 11 abr. 2020. Disponível em: https://analyticsindiamag.com/reinforcement-learning-policy/. Acesso em: 2 set. 2021.

YAGHMAIE, Farnaz Adib; LJUNG, Lennart. A Crash Course on Reinforcement Learning. **arXiv preprint arXiv:2103.04910**, 2021.