Flávio Thiago Franco Vaz

Deep Reinforcement Learning para o Arcade Learning Environment

Relatório de Projeto Final II

DEPARTAMENTO DE INFORMÁTICA
Programa de Graduação em Ciência da Computação

Rio de Janeiro
Julho de 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Flávio Thiago Franco Vaz

Deep Reinforcement Learning para o Arcade Learning Environment

Relatório de Projeto Final II

Relatório de Projeto Final, apresentado ao Programa de Ciência da Computação, do Departamento de Informática da PUC-Rio como requisito parcial para a obtenção do titulo de Bacharel em Ciência da Computação.

Orientador: Prof. José Alberto Rodrigues Pereira Sardinha

Rio de Janeiro
Julho de 2024

Flávio Thiago Franco Vaz

Graduando em Ciência da Computação na PUC-Rio.

Ficha Catalográfica

Thiago Franco Vaz, Flávio

Deep Reinforcement Learning para o Arcade Learning Environment / Flávio Thiago Franco Vaz; orientador: José Alberto Rodrigues Pereira Sardinha. – 2024.

45 f: il. color. ; 30 cm

Projeto Final - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2024.

Inclui bibliografia

1. Informática – Trabalho de Conclusão de Curso. 2. Aprendizado Profundo. 3. Redes Neurais Convolucionais. 4. Aprendizado por Reforço. 5. Aprendizado Profundo por Reforço. 6. Atari. I. Pereira Sardinha, José Alberto. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Deep Reinforcement Learning para o Arcade Learning Environment.

CDD: 004

Dedico esse trabalho à minha mãe e à minha esposa por tanto...

Agradecimentos

Para meu orientador Alberto Sardinha, por aceitar me orientar.

Para minha amada mãe, razão de tudo.

Para meu pai e meus irmãos, por todo apoio incondicional.

Por fim, para minha esposa pelo companheirismo, carinho e compreensão nesses atarefados últimos anos.

Resumo

Thiago Franco Vaz, Flávio; Pereira Sardinha, José Alberto. **Deep Reinforcement Learning para o Arcade Learning Environment**. Rio de Janeiro, 2024. 45p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este estudo investiga a aplicação de técnicas de Deep Reinforcement Learning (DRL) no Arcade Learning Environment (ALE), com o objetivo de desenvolver agentes capazes de superar o desempenho humano em jogos do Atari 2600. A pesquisa foca na avaliação do desempenho e da convergência de técnicas de inicialização de pesos e bias consolidadas na literatura na arquitetura da Deep Q-Network (DQN). A análise inclui comparações entre diferentes estratégias de inicialização e suas implicações na eficiência de aprendizado e na robustez dos agentes treinados em um amplo conjunto de jogos.

Palavras-chave

Aprendizado Profundo; Redes Neurais Convolucionais; Aprendizado por Reforço; Aprendizado Profundo por Reforço; Atari.

Abstract

Thiago Franco Vaz, Flávio; Pereira Sardinha, José Alberto (Advisor). **Deep Reinforcement Learning for the Arcade Learning Environment**. Rio de Janeiro, 2024. 45p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This study investigates the application of Deep Reinforcement Learning (DRL) techniques in the Arcade Learning Environment (ALE), with the aim of developing agents capable of outperforming humans in Atari 2600 games. The research focuses on evaluating the performance and convergence of well-established weight and bias initialization techniques in the literature within the architecture of the Deep Q-Network (DQN). The analysis includes comparisons between different initialization strategies and their implications on the learning efficiency and robustness of agents trained across a range of games.

Keywords

Deep Learning; Convolution Neural Networks; Reinforcement Learning; Deep Reinforcement Learning; Atari.

Sumário

Lista de figuras

Lista de tabelas

Lista de algoritmos

Lista de Códigos

Lista de Abreviaturas

CNN – *Convolutional Neural Networks*

RL – *Reinforcement Learning*

DRL – *Deep Reinforcement Learning*

ALE – *Arcade Learning Environment*

RGB – *Red, Green & Blue*

ReLU – *Rectified Linear Unit*

TD – *Temporal Difference*

All grown-ups were once children... but only few of them remember it.

Antoine de Saint-Exupéry, *The Little Prince.*

1
Introdução

O campo de Reinforcement Learning (RL) (SUTTON; BARTO, 2018), e sua mais recente extensão, Deep Reinforcement Learning (DRL) (MNIH et al., 2013), representa uma dinâmica fronteira na pesquisa de inteligência artificial com o potencial de revolucionar como sistemas autônomos aprendem e interagem com seu ambiente. O principal atrativo de RL é sua versatilidade e generalidade, provendo um arcabouço para que agentes aprendam através de um processo de tentativa e erro sobre as consequências de suas ações a fim de atingir um objetivo. Esse paradigma, apesar de muito simples, é poderoso pois engloba qualquer problema que envolva uma sequência de tomadas de decisão sob incerteza (KOCHENDERFER, 2015).

A motivação para se estudar RL e DRL vem de sua ampla aplicabilidade e sucesso em diversas áreas. Algoritmos de RL vem se provando incrivelmente efetivos nos mais variados ramos, desde jogos, com o desempenho supra-humano do AlphaGo (SILVER et al., 2016) no complexo jogo de Go, passando por aplicações práticas como robótica (KOBER; BAGNELL; PETERS, 2013) e direção autônoma (XIANG, 2024; WANG et al., 2023) até controle em reatores nucleares (SEO et al., 2024; DEGRAVE et al., 2022). Essa versatilidade demonstra potencial para atacar problemas desafiadores, e de alto impacto, em setores como o industrial e de logística.

No entanto, a área é repleta de desafios (DULAC-ARNOLD et al., 2021). Um deles é a necessidade de grandes quantidades de dados para treinar modelos robustos, o que muitas vezes se mostra impraticável em ambientes do mundo real. Além disso, os modelos de DRL necessitam de muitos recursos computacionais para treinamento, enfatizando a necessidade de algoritmos mais eficientes em termos de *data-efficiency*. Outro desafio se apresenta na dificuldade de uma implementação segura de modelos de RL em aplicações do mundo real (THOMAS; LUO; MA, 2021), onde a incerteza pode levar a consequências indesejadas e potencialmente perigosas.

Portanto, a pesquisa em RL serve um propósito prático, permitindo a abordagem de problemas do mundo real que exigem soluções adaptáveis, escaláveis e inteligentes e apresentando oportunidade para liderar inovações que poderão agregar grande impacto na sociedade.

1.1 Definição do problema

O Arcade Learning Environment (ALE) (BELLEMARE et al., 2012) surgiu como uma plataforma para mensurar o progresso e a capacidade dos algoritmos de Reinforcement Learning. O ALE foi desenvolvido como um framework para facilitar a interação de algoritmos de RL com o variado conjunto de jogos do videogame Atari[1]. A plataforma não apenas replica as complexidades inerentes de cenários do mundo real, refletindo os desafios da tomada de decisão sob incerteza, um cenário comum em aplicações práticas, como também provê um ambiente controlado e reproduzível para avaliação rigorosa de métodos de RL.

No entanto, a aplicação de algoritmos de RL no ALE apresenta dificuldades. Um dos principais desafios é a elevada dimensionalidade do espaço de estados. Os jogos do Atari são representados por imagens o que implica no espaço de estados poder incluir todas as configurações possíveis de pixel na tela. Paralelamente, o espaço de ações é diverso, variando consideravelmente entre os diferentes jogos. Essa variabilidade adiciona uma camada de complexidade extra pois os algoritmos devem generalizar bem para diferentes tipos de tarefas e, ao mesmo tempo, ser robustos às particularidades de cada jogo.

Além disso, em muitos ambientes do ALE as recompensas podem ser esparsas e/ou atrasadas. Isso significa que as recompensas não são frequentemente fornecidas ao agente. Em vez de receber feedback contínuo sobre suas ações, o agente pode ter que realizar muitas ações antes de receber uma recompensa. Isso dificulta a identificação de quais ações são benéficas, pois as recompensas são raras o que faz com que ações não possam ser imediatamente associadas a resultados. Isso gera a necessidade de algoritmos que explorem de forma mais eficiente o espaço de estados e que consigam aprender mesmo com a esparsidade das recompensas. O jogo Montezuma's Revenge é um exemplo clássico desses desafios. Nele, o agente precisa realizar uma série de ações específicas e sequenciais para progredir e obter recompensas. O feedback no jogo não é imediato, e as ações que levam a uma recompensa muitas vezes precisam ser planejadas a muito longo prazo.

Existia um gap significativo na literatura em relação a técnicas eficientes para lidar com a questão da alta dimensionalidade do espaço de estados no ALE antes do surgimento da Deep Q-Network (DQN) (MNIH et al., 2013; MNIH et al., 2015). Para tentar superar esses desafios eram usadas técnicas clássicas de RL. No entanto, esses métodos falhavam em capturar o nível de abstração necessário para obter generalização no amplo e diverso

[1]Atari 2600 é um videogame lançado em 1977.

conjunto de jogos presentes. A introdução da DQN foi um marco revolucionário no endereçamento dessas questões. Ao integrar Deep Learning (LECUN; BENGIO; HINTON, 2015) com Q-Learning (WATKINS; DAYAN, 1992), a DQN foi capaz de aprender e generalizar através dos inputs visuais em grande parte dos jogos do Atari de forma sem precedentes. Esse passo não só alavancou a capacidade dos métodos de RL de realizar extração de características de forma autônoma como também se adaptar dinamicamente as variadas necessidades de diferentes jogos. O sucesso da DQN destacou o potencial de Deep Reinforcement Learning em superar desafios significativos presentes no framework, definindo uma nova direção para futuras pesquisas na área.

1.2 Contribuição

Esse trabalho apresenta uma implementação do algoritmo da Deep Q-Network (DQN) assim como proposto no artigo publicado na revista Nature (MNIH et al., 2015). Ao construir a DQN do zero e de forma a ser facilmente adaptada buscamos prover um framework para fins educacionais que pode ser utilizado para exploração e realização de experimentos que ajudem a desenvolver uma maior compreensão das técnicas de Deep Reinforcement Learning. A implementação feita busca ser fiel a metodologia e arquitetura original proposta numa tentativa de obter resultados similares aos apresentados no artigo.

No entanto, a principal contribuição desse trabalho é uma análise de técnicas de inicialização de pesos e vieses e seu impacto na convergência e performance no algoritmo da DQN. A inicialização de pesos (GLOROT; BENGIO, 2010; HE et al., 2015; LECUN et al., 1989) em redes neurais é um fator que influencia na dinâmica do treinamento e no eventual sucesso de modelos de Deep Learning. Enquanto no artigo original são apresentados detalhes e intuições sobre a arquitetura da rede, processo de treinamento e escolha de hiper parâmetros não é informado como é feita a inicialização dos pesos e vieses da rede neural proposta. Além disso, não é feita nenhuma discussão sobre como diferentes estratégias de inicialização podem afetar o processo de aprendizado e resultados obtidos. Nosso trabalho propõe preencher parcialmente esse vazio ao avaliar algumas técnicas de inicialização tradicionais da literatura. Para isso, conduzimos experimentos usando dois conjuntos distintos de jogos do ALE onde pudemos avaliar a robustez dos métodos de inicialização de pesos e obter insights sobre a consistência deles em diferentes cenários.

2
Background

Reinforcement Learning é aprender como mapear situações para ações de forma a maximizar um sinal numérico de recompensa sem que em nenhum momento o agente em aprendizado seja informado sobre que ações tomar (SUTTON; BARTO, 2018). Ao invés disso o agente deve descobrir quais ações levam a uma maior recompensa as tomando, num processo de tentativa e erro. Em alguns casos as ações tomadas podem afetar não só a recompensa imediata recebida pelo agente, mas também quais serão as próximas situações e, consequentemente, todas as recompensas posteriores.

2.1
Markov Decision Process e Reinforcement Learning

Markov Decision Processes (MDP) (BELLMAN, 1957) é um modelo fundamental no campo de Reinforcement Learning. Os MDPs são utilizados para caracterizar ou descrever ambientes onde agentes tomam decisões sequenciais com o objetivo de maximizar recompensas ao longo do tempo. Pode ser definido por uma tupla $< s_t, a_t, r_{t+1}, s_{t+1} >$ como mostrado na Figura 2.1. Esta seção apresenta os conceitos principais de um MDP, incluindo estados, ações, recompensas e políticas, além de explorar as funções de Valor e Q e a Equação de Bellman (BELLMAN, 1957).

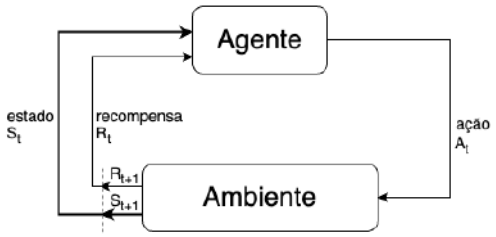

Figura 2.1: A interação Agente - Ambiente em um MDP

Um estado em um MDP representa uma configuração completa do ambiente em um dado momento. É denotado por $s \in S$, onde S é o conjunto de todos os estados possíveis. A principal característica dos estados em um MDP é a propriedade de Markov, que diz que a probabilidade de transição para um próximo estado depende exclusivamente do estado atual e da ação escolhida, e não de estados anteriores. Essa propriedade pode ser expressa como:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_1, a_1, s_2, a_2, \ldots, s_t, a_t)$$

Esta propriedade simplifica a modelagem e a solução de problemas de decisão sequencial, permitindo que se foque apenas no estado atual e na ação presente.

As ações, denotadas por $a \in A$, representam as escolhas que um agente pode fazer a partir de um estado, sendo A o conjunto de todas as ações possíveis em um ambiente. A escolha de uma ação desse conjunto determina a transição de um estado para outro, seguindo a distribuição de probabilidade de transição do ambiente.

Recompensas são valores numéricos recebidos após a transição de um estado para outro como consequência da tomada de uma ação. Denotada por $R(s, a, s')$, a recompensa reflete o valor imediato da transição, incentivando o agente a perseguir estados e ações que maximizem a soma total das recompensas ao longo do tempo.

2.1.1 Política

Uma política π é uma estratégia que define a ação a ser tomada em cada estado. Pode ser determinística, onde uma ação específica é selecionada para cada estado, ou estocástica, onde ações são escolhidas de acordo com uma distribuição de probabilidade. A política é essencialmente o plano que o agente segue para interagir com o ambiente.

No contexto Reinforcement Learning, a política desempenha um papel fundamental na determinação do comportamento do agente. Uma política ótima, denotada por π^*, maximiza a recompensa esperada ao longo do tempo. Existem diferentes abordagens para representar e aprender políticas:

Políticas Tabulares: Utilizadas em ambientes discretos e de tamanho reduzido, onde a política pode ser representada explicitamente em uma tabela que mapeia estados a ações.

Políticas Baseadas em Funções: Em ambientes com espaços de estado ou ação contínuos ou de grande dimensão, utiliza-se uma função parametrizada (como redes neurais) para aproximar a política.

Aprendizado de Política Direta vs. Indireta:

- Aprendizado Direto de Política: Técnicas como Gradiente de Política (SUTTON et al., 1999) ajustam diretamente os parâmetros da política de modo a maximizar a recompensa esperada.
- Aprendizado Indireto de Política: Métodos como Q-Learning (WATKINS; DAYAN, 1992) aprendem uma função chamada Q que é utilizada para derivar uma política ótima. Veremos mais sobre a função Q nas próximas seções.

Exploração vs. Explotação: A política deve equilibrar a exploração de novas ações e a explotação de ações conhecidas que fornecem alta recompensa. Este equilíbrio é crucial para o sucesso do aprendizado por reforço.

2.1.2 Funções de Valor

A função de valor de estado $V^{\pi}(s)$ (SUTTON; BARTO, 2018) mede o valor esperado descontado de recompensa de estar em um estado s e seguir a política π. Já a função de valor de ação-estado, ou função $Q^{\pi}(s, a)$ (SUTTON; BARTO, 2018), mede o valor esperado descontado de recompensa de executar a ação a no estado s e depois seguir a política π. A Equação de Bellman para a função Q pode ser expressa como:

$$Q^{\pi}(s, a) = E\left[R_{t+1} + \gamma Q^{\pi}\left(s_{t+1}, \pi\left(s_{t+1}\right)\right) | s_t = s, a_t = a\right]$$

No contexto de RL, o objetivo é encontrar a política ótima π^* que maximiza o retorno esperado, definido pela soma das recompensas descontadas. Os algoritmos de RL, como a DQN que será avaliada nesse trabalho, utilizam a estrutura de um MDP para iterativamente melhorar a estimativa das funções Q de forma a maximizar a recompensa obtida em um determinado ambiente.

2.1.3 Equação de Bellman

A Equação de Bellman (BELLMAN, 1957) é uma formulação matemática que relaciona o valor de um estado com os valores dos estados subsequentes. Ela é usada para calcular a função de valor, que estima o valor esperado de recompensas futuras a partir de um estado seguindo uma política π. A Equação de Bellman para a função de valor de estado, $V^{\pi}(s)$, é dada por:

$$V^{\pi}(s) = E^{\pi}\left[R_{t+1} + \gamma V^{\pi}\left(s_{t+1}\right) | s_t = s\right]$$

onde γ é o fator de desconto usado para ponderar a importância das recompensas futuras. O uso de valores de γ próximos a zero implicam em agentes que aprendem a maximizar somente a próxima recompensa no MDP, enquanto valores próximos a um implicam em agentes que tentarão maximizar a recompensa total obtida no longo prazo.

2.2 Model-Based e Model-Free Reinforcement Learning

No campo do Reinforcement Learning, as abordagens podem ser classificadas em duas categorias principais: Model-Based e Model-Free. Ambas têm suas próprias características, vantagens e desvantagens, e a escolha entre elas depende do problema específico.

Model-Based Reinforcement Learning envolve a construção de um modelo explícito do ambiente. Este modelo é utilizado pelo agente para prever o resultado das suas ações e planejar seu comportamento com base nessas previsões. O modelo pode ser representado por uma função de transição que descreve a probabilidade de transitar de um estado para outro dado uma ação, e uma função de recompensa que prevê a recompensa esperada ao tomar uma ação em um estado específico. Esse tipo de abordagem permite que o agente planeje suas ações podendo potencialmente levar a soluções mais inteligentes e informadas. No entanto, também traz consigo uma complexidade de modelagem: construir um modelo preciso do ambiente pode ser desafiador, especialmente em ambientes complexos ou com alta dimensionalidade.

Model-Free Reinforcement Learning, por outro lado, não tenta construir um modelo explícito do ambiente. Em vez disso, o agente aprende diretamente uma política ou uma função de valor através da interação com o ambiente. Os métodos Model-Free podem ser divididos em duas subcategorias: aprendizado baseado em valor e aprendizado baseado em política. Esses métodos podem simplificar o processo de desenvolvimento já que não requerem a construção de um modelo do ambiente e também aumentam a aplicabilidade geral dos algoritmos pois podem ser aplicados a uma ampla variedade de problemas sem a necessidade de modelagem específica. No entanto, em geral, esses algoritmos requerem mais interações com o ambiente para aprenderem uma política ou função de valor eficaz.

2.3 Temporal Difference Learning

Temporal Difference (TD) Learning (SUTTON, 1988) é uma abordagem que combina ideias de aprendizado supervisionado e aprendizado não supervisionado para estimar funções de valor. TD Learning é particularmente eficiente pois utiliza o conceito de *bootstrap*, atualizando suas estimativas de valor a partir de outras estimativas, o que permite que o aprendizado ocorra a partir de uma única interação com o ambiente e sem supervisão humana.

O TD Learning envolve a atualização da função de valor utilizando a diferença temporal, que é a diferença entre a estimativa de valor atual e a

estimativa de valor para o próximo estado. Essa diferença é utilizada para corrigir a estimativa atual, melhorando gradualmente a precisão da função de valor.

A atualização da função de valor de estado $V(s)$ utilizando TD Learning é dada por:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

TD(0) é a forma mais simples de TD Learning, onde a atualização ocorre a cada passo, utilizando apenas o próximo estado para calcular a diferença temporal. A atualização é imediata e não considera recompensas futuras além da recebida no próximo estado.

2.4 Q Learning

Q-Learning é um algoritmo de TD Learning fundamental em Reinforcement Learning. Ele se baseia na ideia de aprender a função de valor de ação-estado, conhecida como função Q, que estima a recompensa esperada ao executar uma ação em um estado específico e seguir a melhor política futura a partir desse estado. A função Q é atualizada iterativamente com base nas experiências do agente.

A atualização da função Q em Q-Learning é baseada na Equação de Bellman para a função de valor de ação-estado. A equação de atualização de Q-Learning pode ser expressa como:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

onde s é o estado atual, a é a ação atual tomada, r é a recompensa recebida após tomar a ação a no estado s, s' é o próximo estado resultante da ação a, α é a taxa de aprendizado, que controla a velocidade de atualização dos valores Q, γ é o fator de desconto, que pondera a importância das recompensas futuras e $\max_{a'} Q(s',a')$ é o valor máximo da função Q para o próximo estado s' considerando todas as ações possíveis a'.

A ideia principal é ajustar iterativamente a estimativa da função Q para que ela se aproxime do valor verdadeiro das recompensas futuras esperadas.

Apesar de sua simplicidade e eficácia, o Q-Learning enfrenta desafios em ambientes com grandes espaços de estado-ação. Em tais casos, o uso de tabelas para armazenar valores Q se torna impraticável. Para lidar com esses desafios, várias extensões e técnicas foram desenvolvidas, incluindo o uso de

aproximação de função com redes neurais, resultando em algoritmos como o da DQN.

2.4.1 Política ϵ-gananciosa

A política ϵ-gananciosa é uma estratégia utilizada para balancear exploração e explotação durante o aprendizado. Ela é definida da seguinte forma:

- Com probabilidade ϵ, selecionar uma ação aleatória (exploração).
- Com probabilidade $1 - \epsilon$, selecionar a ação que maximiza o valor da função Q no estado atual (exploração).

O valor de ϵ é geralmente reduzido ao longo do tempo, começando com um valor alto para incentivar a exploração inicial e diminuindo gradualmente para favorecer a explotação de ações que se mostraram promissoras.

Algoritmo 1: O Algoritmo do Q-Learning

```
para cada episódio do treinamento faça
    inicializar estado inicial s
    para cada passo do episódio faça
        selecionar uma ação a com base na política ε-gananciosa.
        executar ação a, observar a recompensa r e o próximo estado s'.
        atualizar a função Q usando a equação de atualização do
         Q-Learning.
        atualizar o estado s ← s'.
```

2.5 Redes Neurais

Na última década, redes neurais profundas emergiram como ferramentas poderosas e versáteis para resolver uma ampla variedade de problemas em diversas áreas, incluindo reconhecimento de padrões (HE et al., 2016) , processamento de linguagem natural (VASWANI et al., 2017) e aprendizado por reforço (MNIH et al., 2015). Esta seção fornece uma visão geral das redes neurais, com ênfase particular nas Redes Neurais Convolucionais (CNNs), destacando seu papel no aprendizado com imagens e sua aplicação na superação das limitações dos métodos tradicionais de aprendizado por reforço em ambientes complexos, como os jogos do Atari.

Redes neurais são modelos computacionais inspirados na estrutura e no funcionamento do cérebro humano. Elas consistem em camadas de neurônios artificiais interligados, onde cada neurônio realiza uma combinação linear dos

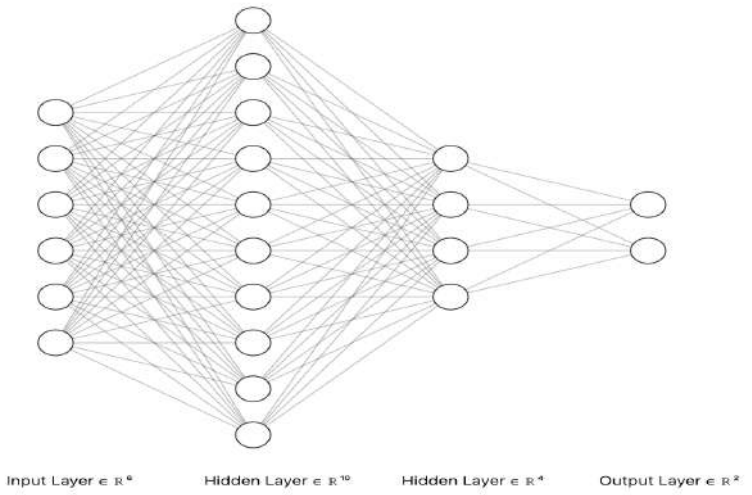

Figura 2.2: Uma Rede Neural com duas camadas ocultas

sinais fornecidos por suas entradas seguida por uma transformação não-linear. Um exeplo de uma rede neural é apresentado na Figura 2.2. Podemos definir o funcionamento de um neurônio através da equação:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

onde f é uma função não linear, chamada função de ativação, w_i e b são os parâmetros responsáveis pelo aprendizado do neurônio, chamados pesos e bias, e x_i são os sinais de entrada do neurônio.

As redes neurais são capazes de aprender representações dos dados por meio de um processo de treinamento que ajusta os pesos das conexões entre os neurônios, geralmente utilizando gradiente descendente e retropropagação:

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

onde α é a taxa de aprendizado, L é uma função de perda e $\frac{\partial L}{\partial w}$ é o gradiente da função de perda em relação aos pesos.

As Redes Neurais Convolucionais (CNNs) (LECUN et al., 1989) são uma classe específica de redes neurais particularmente eficazes no processamento de dados visuais como vídeos e imagens. Introduzidas inicialmente para tarefas de reconhecimento de imagem, as CNNs têm uma arquitetura que explora a estrutura espacial dos dados, utilizando camadas de convolução para capturar características locais das imagens, como bordas, texturas e formas nas camadas iniciais, e características mais complexas nas camadas finais. Um exemplo de CNN é apresentado na Figura 2.3.

A principal vantagem das CNNs é sua capacidade de aprender características relevantes de maneira hierárquica. Em camadas iniciais, as CNNs capturam características de baixo nível, como bordas e texturas, enquanto em camadas mais profundas, elas reconhecem estruturas mais complexas, como objetos e cenas completas. Isso é particularmente importante em problemas onde a entrada são imagens, permitindo que as CNNs extraiam representações significativas dos dados de entrada sem a necessidade de criar características manualmente.

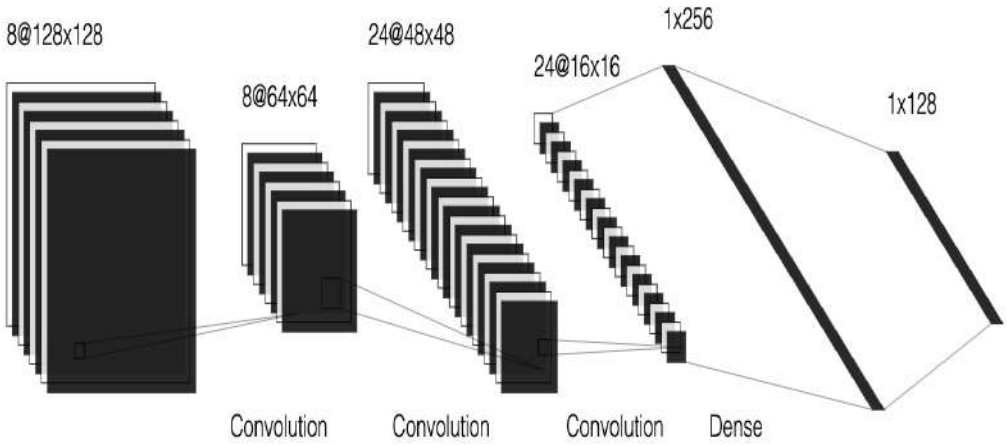

Figura 2.3: Uma Rede Neural Convolucional com 4 camadas

Tradicionalmente, os métodos de aprendizado por reforço enfrentavam desafios significativos quando aplicados a problemas com espaços de estado de alta dimensão, como os jogos do Atari. Esses desafios incluíam a dificuldade de representar e generalizar os estados do ambiente a partir dos frames do jogo, bem como a necessidade de uma exploração eficiente em espaços de ação complexos.

Os jogos do Atari apresentam um ambiente complexo, onde a entrada é constituída por pixels que mudam rapidamente. Métodos tradicionais, como Q-Learning, baseados em tabelas de valores, não conseguem lidar com essa alta dimensionalidade e complexidade visual. A integração de CNNs no framework de Deep Reinforcement Learning transformou a abordagem desses problemas. As CNNs são usadas para processar diretamente os frames do jogo, extraindo

características importantes e reduzindo a dimensionalidade do espaço de entrada. O algoritmo apresentado a seguir combina essa capacidade de representação das CNNs com o Q-Learning, permitindo que o agente aprenda a função Q diretamente a partir de dados visuais brutos. Esse avanço permitiu que agentes de DRL superassem o desempenho humano em vários jogos do Atari 2600, demonstrando a eficácia das CNNs em capturar a complexidade dos ambientes visuais e facilitando a aprendizagem de estratégias complexas e eficazes previamente desconhecidas.

3
Deep Q Network

A DQN é um algoritmo de Reinforcement Learning que combina Q-learning com redes neurais profundas. Foi apresentado pelo trabalho de Mnih et al. em 2015, onde foi usado para jogar jogos de Atari com desempenho sobre humano na maioria dos casos. Q-learning é um método de RL que visa aprender a função $Q(s, a)$, que representa o valor esperado da recompensa descontada total ao tomar a ação a no estado s e seguir a política ótima a partir de então. A ideia principal por trás da DQN é usar uma rede neural profunda para aproximar a função Q. Essa rede neural toma um estado s como entrada e produz uma estimativa dos valores Q para todas as ações possíveis para o ambiente em que está sendo treinada. O Algoritmo 2 apresenta a DQN.

Algoritmo 2: DQN

Inicializar Buffer de Replay
Inicializar θ
$\theta' \leftarrow \theta$
Inicializar $s_1 = \{x_1\}$
Pre-processar $\phi_1 = \phi(s_1)$
para *t = 1, T* **faça**
    Com probabilidade ϵ *selecionar uma ação aleatória* a_t *caso contrário selecionar a ação* $a_t = max_a Q(\phi(s_t), a; \theta)$
    Executar ação a_t *no emulador e observar recompensa* r_t*, imagem* x_{t+1} *e se o jogo acabou* (d_t)
    $s_{t+1} \leftarrow s_t, a_t, x_{t+1}$
    Pre-processar $\phi_{t+1} = \phi(s_{t+1})$
    Salvar transição $< \phi_t, a_t, r_t, phi_{t+1}, d_t >$ no buffer de replay
    Recuperar batch de transições aleatórias do buffer de replay $< \phi_k, a_k, r_k, phi_{k+1}, d_k >$
    se d_k *é terminal* **então**
        $y_k \leftarrow r_k$;
    senão
        $y_k \leftarrow r_k + \gamma max_{a'} Q(\phi_{k+1}, a'; \theta')$;
    $q_k \leftarrow Q(\phi_k, a_k; \theta)$
    Gradient Descent com $L(y_k, q_k)$
    A cada C passos $\theta' \leftarrow \theta$

3.1 Rede Q Alvo (Target Network)

Uma das principais dificuldades em treinar redes neurais profundas para RL é a instabilidade causada pelas constantes atualizações dos valores de Q. Para mitigar isso, a DQN utiliza uma rede Q alvo (target network) (MNIH et al., 2015), que é uma cópia da rede Q original, mas com pesos θ^- que são atualizados menos frequentemente. A rede Q alvo serve para fornecer uma estimativa mais estável dos valores de Q durante o treinamento. Sem ela, as atualizações do Q-learning poderiam resultar em grandes oscilações nos valores estimados devido à constante mudança dos pesos da rede neural, o que pode dificultar a convergência do algoritmo. A rede Q alvo é atualizada periodicamente com os pesos da rede Q atual, mas não é atualizada a cada passo de tempo, proporcionando assim um "norte" mais estável para as estimativas de Q.

3.2 Buffer de Replay

Outra técnica importante utilizada pela DQN é o buffer de experiência (MNIH et al., 2015). Este buffer armazena um conjunto de transições $(s_t, a_t, r_t, s_{t+1}, d)$ experienciadas pelo agente durante a interação com o ambiente. O último elemento d informa se um episódio terminou após a interação t. O buffer de replay ajuda a quebrar a correlação entre as transições consecutivas no processo de aprendizado. Ao armazenar experiências passadas e amostrar batches aleatórios dessas experiências para realizar o treinamento, o buffer de replay garante que as atualizações dos pesos da rede neural sejam feitas com base em um conjunto mais diversificado e representativo de experiências. Isso melhora a eficiência do treinamento e a estabilidade do algoritmo.

3.3 Pré-processamento

Antes de ser processados pela rede Q os estados do Atari passam por um processo de pré-processamento ϕ que visa reduzir a dimensionalidade dos dados. Inicialmente os frames do jogo tem dimensão 210 x 160. O primeiro passo de pré-processamento toma o valor máximo de cada pixel para cada canal RGB entre dois frames consecutivos do ambiente. Posteriormente a imagem gerada é convertida para escala de cinza e redimensionada para 84 x 84. Por fim, os quatro frames mais recentes são empilhados. Esse passo é necessário a fim de produzir estados que atendam a propriedade dos estados de Markov

uma vez que um só frame não é suficiente para que o agente consiga tomar uma decisão.

3.4 Arquitetura

A rede neural profunda $Q(s, a; \theta)$ tem parâmetros θ que são ajustados para minimizar a diferença temporal (TD error):

$$\delta = \gamma Q\left(s_{t+1}, a; \theta^{-}\right) \; + \; r_t \; - \; Q\left(s_t, a_t; \theta\right)$$

onde θ^{-} são os parâmetros da rede Q alvo, que são atualizados em momentos específicos do treinamento para igualar θ.

A arquitetura da rede neural utilizada consiste em três camadas convolucionais com ativações ReLU (FUKUSHIMA, 1969) (Conv1: 32 filtros de tamanho 8 x 8 e stride 4; Conv2: 64 filtros de tamanho 4 x 4 e stride 2; Conv3: 64 filtros de tamanho 3 x 3 e stride 1), seguida por uma camada linear com 512 neurônios, uma camada ReLU e uma camada de saída linear final com N neurônios, onde N é o número de ações válidas para o ambiente atual. A Figura 3.1 ilustra a DQN.

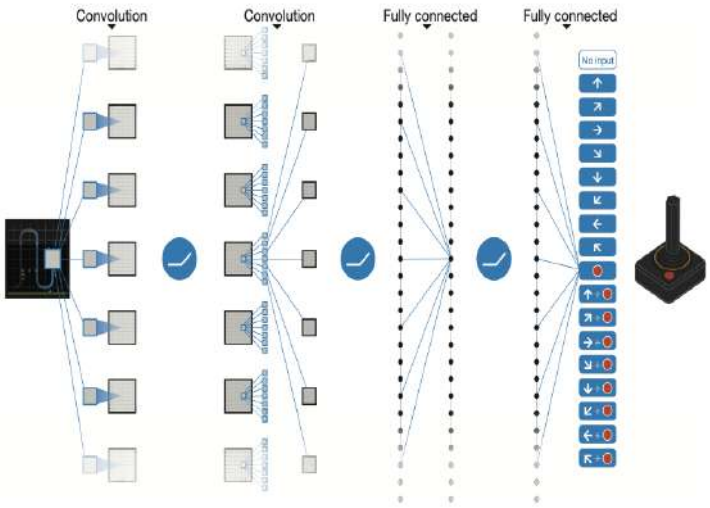

Figura 3.1: Ilustração da Rede Neural Convolucional da DQN. Retirado de (MNIH et al., 2015)

4
Experimentos e Resultados

4.1
O ambiente de desenvolvimento

Os experimentos realizados no projeto foram desenvolvidos usando Python versão 3.9.7. Todos os agentes foram treinados em uma instância virtual Linux (Ubuntu) com 10 cores de CPU, 50GB de memória RAM e uma GPU RTX 3090 com 24GB de memória interna. Os principais pacotes usados e suas versões são apresentados abaixo:

- gym: 0.26.2
- hydra: 1.3.2
- opencv-python: 4.8.1.78
- torch: 2.1.1+cu118
- torchrl: 0.2.1
- wandb: 0.16.1

O pacote Hydra é um pacote desenvolvido pelo Facebook para facilitar a administração de configurações em experimentos ao criar arquivos de configuração hierárquicos, possibilitando o desacoplamento, facilitando a organização, a alteração de parâmetros e a execução de múltiplos experimentos com configurações diferentes em uma única execução do código.

A plataforma *Weights and Biases* oferece uma ferramenta para *experiment tracking* com uma interface intuitiva, de fácil aprendizado e uso. Todos os experimentos descritos nesse capítulo fizeram uso essa plataforma para reportar métricas, gerar gráficos e salvar modelos. Além disso a plataforma foi usada posteriormente para comparar resultados de diferentes execuções.

Todos os agentes foram treinados a partir de interações diretas com os jogos do Atari em um ambiente emulado fornecido pelo pacote Gym. Em outras palavras, os agentes aprenderam a jogar os jogos os jogando.

4.2 Experimentos

Os procedimentos adotados na pesquisa foram divididos em três etapas: seleção da inicialização de pesos, seleção da inicialização de bias e avaliação final das técnicas selecionadas.

Nas etapas de seleção de pesos e seleção de bias foram usados cinco jogos como base para os experimentos realizados: Breakout, Space Invaders, Seaquest, Enduro e River Raid. Para cada um desses jogos, agentes foram treinados usando 40 milhões de frames do jogo. Já na etapa de Avaliação final foram usados quatro jogos: Alien, Crazy Climber, Tennis e Time Pilot e os agentes foram treinados usando 200 milhões de frames do jogo. A diferenciação entre os conjuntos de jogos de seleção de pesos e bias e avaliação final foi feita com intuito de remover qualquer tipo de viés causado pelo processo de seleção. Frames desses jogos podem ser encontrados na Figura 4.1. A cada um milhão de frames, os modelos foram avaliados em 540 mil frames do jogo, e o resultado médio por episódio obtido nessa etapa, chamada de validação, é reportado. Os pesos do modelo com melhor resultado encontrado na etapa de validação são salvos e posteriormente usados para uma etapa de testes, em que o agente é avaliado em 30 episódios e o resultado médio por episódio é reportado. O score final de cada agente foi definido como o resultado obtido durante a etapa de testes, enquanto o score final de cada técnica de inicialização de pesos e bias foi a média dos scores dos quatro agentes associados a cada jogo.

Em cada experimento foram treinados quatro agentes usando seeds diferentes, sendo essas 1, 2, 3 e 4, afim de trazer mais robustez para os resultados encontrados. Assim como na DQN (MNIH et al., 2015) foram usadas inicializações de ambiente com um número aleatório entre 0 e 30 de ações *no-op*, onde o agente toma a ação não fazer nada. Esse processo é realizado com intuito de gerar novas possíveis configurações iniciais para o ambiente, oque nem sempre ocorre pois existem jogos em que é necessário tomar uma ação específica para que o jogo se inicie, como no Breakout e Tennis. Todos os episódios foram truncados em 10800 frames.

É importante ressaltar que o intuito desse trabalho não é realizar um ajuste de hiper-parâmetros ou tentar obter a combinação de pesos e bias que gere os melhores resultados possíveis e sim analisar como a inicialização de pesos pode afetar o desempenho da DQN. Além disso, seria inviável por questões de recursos e tempo realizar todos os experimentos necessários para um ajuste de hiper-parâmetros apropriado dado o longo tempo de treinamento da DQN (12 horas/40 milhões de frames em uma máquina com 50GB de memória RAM e uma GPU RTX 3090).

Figura 4.1: Frames dos jogos usados nos experimentos. Fonte: site da Gym[1]

4.2.1 Estratégias de inicialização de pesos

Foram usadas quatro técnicas diferentes de inicialização de pesos foram utilizadas, sendo essas: He Uniform (HE et al., 2015), Xavier Uniform (GLOROT; BENGIO, 2010), Truncated Uniform e Xavier Normal (GLOROT; BENGIO, 2010). O uso adequado destas técnicas pode reduzir problemas de gradientes desaparecendo ou explodindo, proporcionando um início de treinamento mais estável e eficaz.

Todas as técnicas usadas foram projetadas para preservar a variância das ativações ao longo das camadas da rede, promovendo um treinamento mais estável e eficiente durante o treinamento. A seguir, detalhamos cada uma dessas técnicas:

4.2.1.1 He Uniform

A técnica de inicialização He Uniform, também conhecida como Kaiming Uniform, desenvolvida por Kaiming He et al., é especialmente eficaz para redes neurais profundas utilizando funções de ativação ReLU (Rectified Linear Unit). Nesta abordagem, os pesos são inicializados com valores uniformemente distribuídos dentro do intervalo $[-a, a]$, onde $a = \sqrt{\frac{6}{fan_{in}}}$. Aqui, fan_{in} representa o número de unidades de entrada para os neurônios na camada.

[1]Disponível em <https://www.gymlibrary.dev/environments/atari/complete_list/>. Acesso em 10 jul. 2024

4.2.1.2
Xavier Uniform

A inicialização Xavier Uniform, também conhecida como Glorot Uniform, foi introduzida por Xavier Glorot e Yoshua Bengio. Esta técnica é amplamente utilizada em redes com funções de ativação sigmoid ou tanh. Os pesos são inicializados de forma que os valores sejam distribuídos uniformemente no intervalo $[-a, a]$, com $a = \sqrt{\frac{6}{fan_{in}+fan_{out}}}$, onde fan_{in} e fan_{out} representam o número de unidades de entrada e saída, respectivamente.

4.2.1.3
Truncated Uniform

A técnica de inicialização Truncated Uniform é a padrão do PyTorch. Nesta abordagem, os pesos são inicializados com valores uniformemente distribuídos dentro do intervalo $[-a, a]$, onde $a = \frac{1}{\sqrt{fan_{in}}}$. Aqui, fan_{in} representa o número de unidades de entrada para os neurônios na camada.

4.2.1.4
Xavier Normal

Assim como a Xavier Uniform, a técnica Xavier Normal também foi proposta por Xavier Glorot e Yoshua Bengio. Porém, nesta abordagem, os pesos são inicializados a partir de uma distribuição normal com média zero e variância $\sigma^2 = \frac{2}{fan_{in}+fan_{out}}$.

Estas técnicas de inicialização de pesos foram selecionadas e aplicadas em nossos experimentos para explorar e comparar a eficácia de cada abordagem no contexto de diferentes arquiteturas de redes neurais.

4.2.2
Estratégias de inicialização de bias

Após a conclusão da etapa inicial, a inicialização de pesos que obteve o melhor score foi fixada para a próxima fase de Seleção da inicialização do bias. Duas opções de inicialização de bias foram testadas: 0 e Truncated Uniform. O mesmo procedimento de treinamento e avaliação foi repetido para determinar a melhor inicialização de bias usando 4 seeds.

4.2.3
Avaliação final

Nessa parte foi medida a performance das inicializações escolhidas em um conjunto de jogos diferente do usado nos processos de seleção feitos anteriormente. O mesmo procedimento foi seguido: quatro modelos foram

treinados para cada jogo, cada um com uma seed diferente, seguindo os mesmos métodos de avaliação utilizados nas etapas anteriores.

4.3 Resultados

Essa seção apresenta os resultados obtidos nos experimentos listados na seção anterior. Em cada Tabela, os valores em destaque em verde representam os melhores desempenhos entre todas as seeds. Ao fim de cada seção são apresentadas Tabelas consolidadas para comparação direta entre as diferentes técnicas utilizadas.

4.3.1 Resultados das inicializações de pesos

Os resultados obtidos com as diferentes inicializações de pesos nos jogos selecionados são apresentados nas Tabelas 4.1, 4.2, 4.3, 4.4. A Tabela 4.5 consolida os resultados obtidos.

Jogo	Seed 1	Seed 2	Seed 3	Seed 4	Average	Std-Dev
Breakout	248	242	258	287	259	17
Seaquest	902	2051	1552	883	1347	488
Space Invaders	907	792	818	861	845	44
Enduro	723	681	756	673	708	33
River Raid	4795	5022	4658	5218	4923	214

Tabela 4.1: Inicialização Truncated Uniform

Jogo	Seed 1	Seed 2	Seed 3	Seed 4	Average	Std-Dev
Breakout	243	135	239	75	173	71
Seaquest	787	1188	1241	774	998	218
Space Invaders	767	704	753	795	755	33
Enduro	604	581	648	623	614	25
River Raid	3741	3436	3649	4140	3741	255

Tabela 4.2: Inicialização Xavier Uniforme

Jogo	Seed 1	Seed 2	Seed 3	Seed 4	Average	Std-Dev
Breakout	223	145	154	176	175	30
Seaquest	828	862	851	1188	932	148
Space Invaders	700	715	725	739	720	14
Enduro	643	716	758	698	704	41
River Raid	3837	3533	2874	3053	3324	381

Tabela 4.3: Inicialização Xavier Normal

Jogo	Seed 1	Seed 2	Seed 3	Seed 4	Average	Std-Dev
Breakout	100	10	80	40	58	35
Seaquest	432	776	576	306	523	175
Space Invaders	663	560	590	638	613	40
Enduro	479	424	17	483	351	194
River Raid	2452	1793	2474	2650	2342	326

Tabela 4.4: Inicialização He Uniforme

Jogo	Truncated Uniform	Xavier Uniform	Xavier Normal	He Uniform
Breakout	259	173	175	58
Seaquest	1347	998	932	523
Space Invaders	845	720	590	613
Enduro	708	614	704	351
River Raid	4923	3741	3324	2342

Tabela 4.5: Consolidação das inicializações de peso

4.3.2 Resultados das inicializações de bias

Os resultados obtidos com as diferentes inicializações de bias nos jogos selecionados são apresentados nas Tabelas 4.7 e 4.1, uma vez que a inicialização de bias usada na seção passada é Truncated Uniform. A Tabela 4.8 consolida os resultados obtidos. Usaremos como base para para essa seção a inicialização de pesos com melhor desempenho nos experimentos da seção anterior, presentes na Tabela 4.6.

Jogo	Truncated Uniform
Breakout	259
Seaquest	1347
Space Invaders	845
Enduro	708
River Raid	4923

Tabela 4.6: Resultado da melhor inicialização de pesos

Jogo	Seed 1	Seed 2	Seed 3	Seed 4	Average	Std-Dev
Breakout	292	264	167	298	255	53
Seaquest	972	1417	893	1357	1160	230
Space Invaders	899	880	949	863	898	32
Enduro	769	628	747	831	744	74
River Raid	5116	5440	4973	4935	5116	199

Tabela 4.7: Inicialização zero

Jogo	Truncated Uniform	Zero
Breakout	259	255
Seaquest	1347	1160
Space Invaders	845	898
Enduro	708	744
River Raid	4923	5116

Tabela 4.8: Consolidação das inicializações bias

4.3.3 Resultados da avaliação final

Por fim, nessa seção apresentamos na Tabela 4.10 os resultados dos testes finais com as melhores inicializações encontradas nas duas etapas anteriores, sendo essas Truncated Uniform para pesos e zero para bias. Em conjunto, é apresentado na Tabela 4.11 uma comparação dos resultados obtidos com os resultados apresentados no paper da Deep Mind (MNIH et al., 2015). Usaremos como base para para essa seção as inicializações de pesos de pesos e bias com melhor desempenho nos experimentos das seções anteriores, presentes na Tabela 4.9.

Jogo	Truncated Uniform e Zero	Deep Mind	Humano
Breakout	255	401	32
Seaquest	1160	5286	20182
Space Invaders	898	1976	1652
Enduro	744	301	309
River Raid	5116	8316	13513

Tabela 4.9: Resultados das melhores inicializações de pesos e bias comparados com o desempenho humano e o da DQN treinada em 200 milhões de frames

Jogo	Seed 1	Seed 2	Seed 3	Seed 4	Average	Std-Dev
Crazy Climber	99543	112090	116923	115510	111017	6853
Tennis	-23	-23	-23	-23	-23	0
Time Pilot	4387	3113	1113	2590	2801	1173
Alien	2019	1782	1707	1612	1780	151

Tabela 4.10: Resultados obtidos na etapa final de testes

Jogo	Nosso	Deep Mind	Humano
Crazy Climber	111017	114103	35411
Tennis	-23	-2.5	-8.9
Time Pilot	2801	5947	5925
Alien	1780	3069	6875

Tabela 4.11: Comparação entre os resultados obtidos em nosso trabalho, no da Deep Mind e o desempenho humano

Figura 4.2: Curvas de Recompensa ao longo do Treinamento na etapa de seleção de pesos

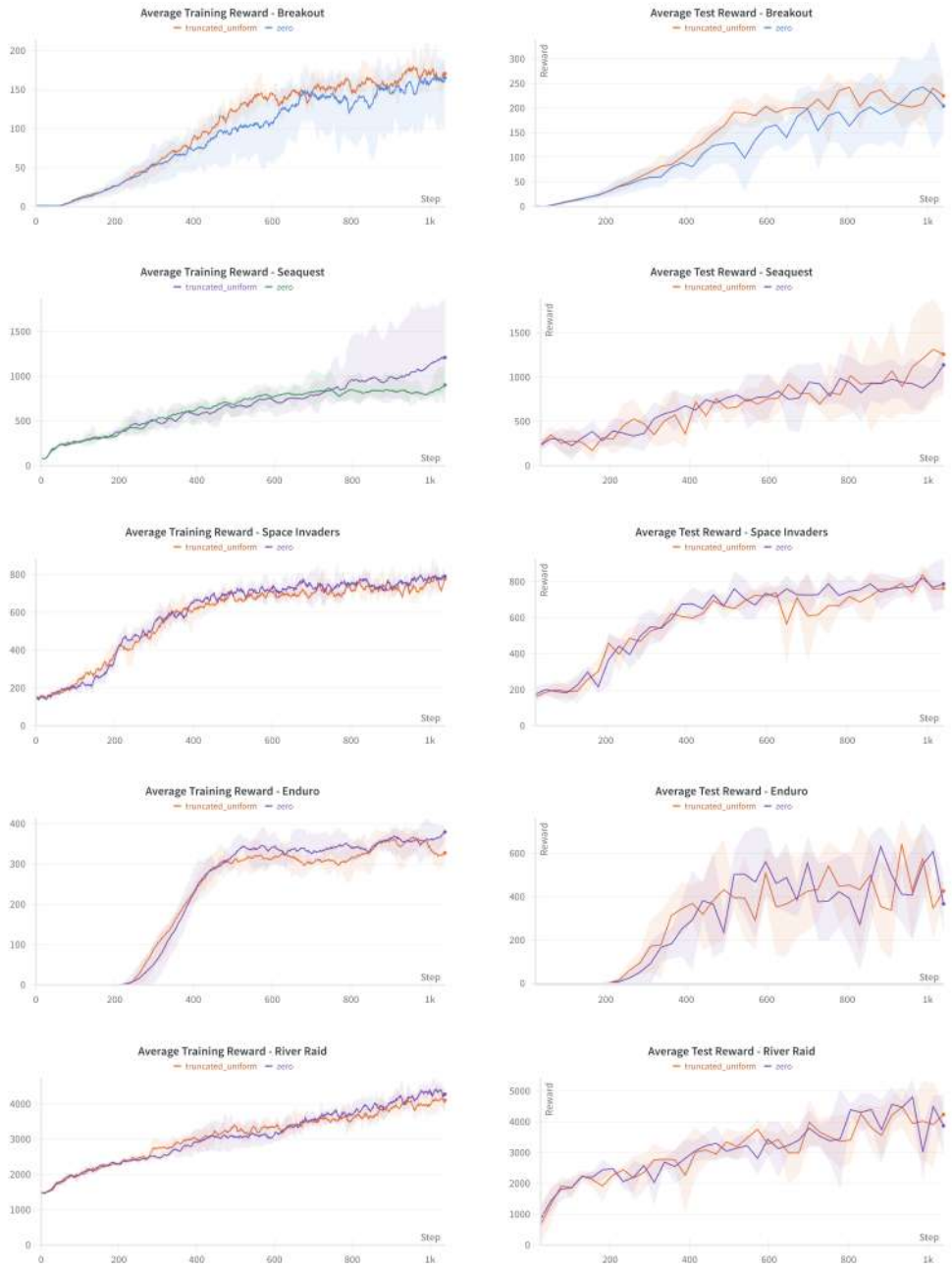

Figura 4.3: Curvas de Recompensa ao longo do Treinamento na etapa de seleção de bias

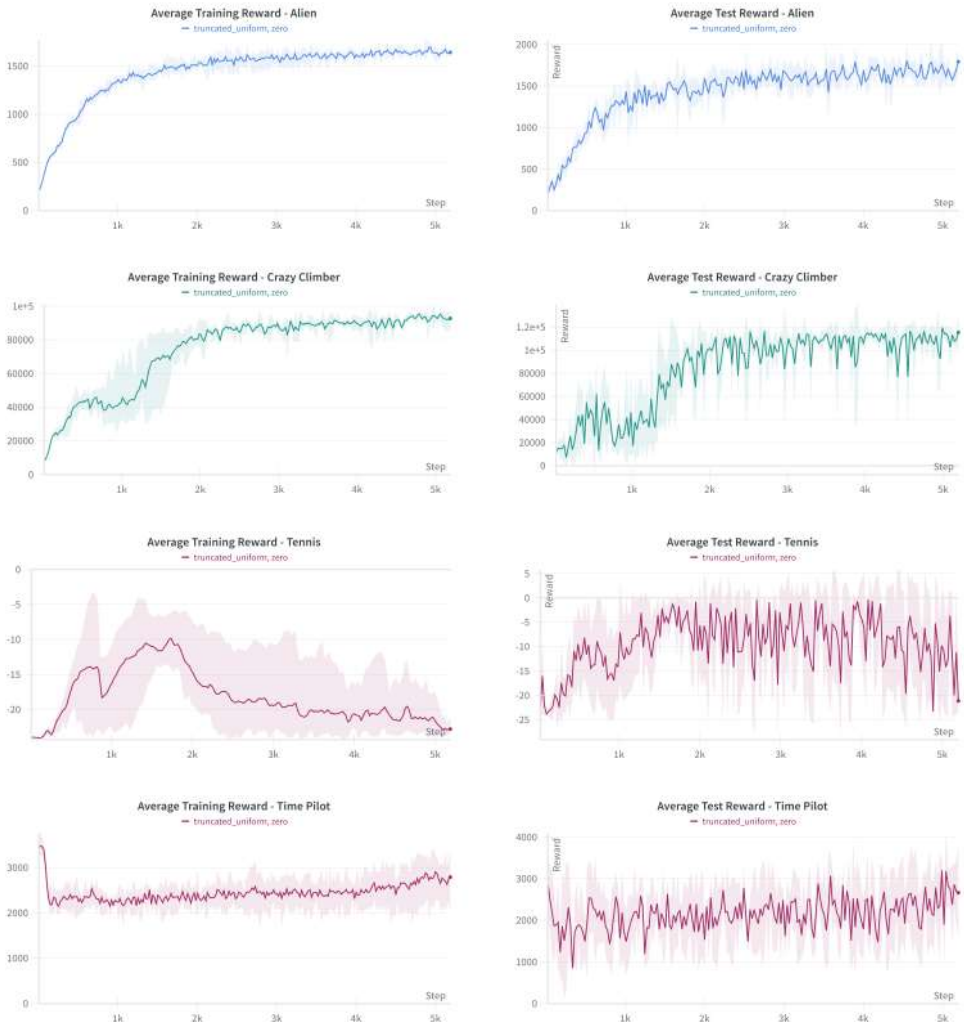

Figura 4.4: Curvas de Recompensa ao longo do Treinamento na etapa de avaliação final

Hiper-parâmetro	**Valor**	**Descrição**
minibatch size	32	Número de casos de treinamento sobre os quais cada passo de otimização é realizado.
replay memory size	1000000	Capacidade do buffer de replay que é usada para salvar experiências do agente.
agent history length	4	O número de frames mais recentes experienciados pelo agente que são dados como input à rede Q.
target network update frequency	10000	A frequência com a qual a rede alvo é atualizada.
discount factor	0.99	Fator de desconto γ usado na atualização do Q-learning.
action repeat	4	Número de repetições da ação selecionada pelo agente.
update frequency	4	O número de ações selecionadas pelo agente entre atualizações sucessivas de pesos.
learning rate	0.00025	A taxa de aprendizado usada pelo otimizador.
gradient momentum	0.95	Momentum do gradiente usado pelo otimizador.
squared gradient momentum	0.95	Momentum do gradiente ao quadrado (denominador) usado pelo otimizador.
min squared gradient	0.01	Constante adicionada ao quadrado do gradiente no denominador na equação do otimizador.
initial exploration	1	Valor inicial de ϵ na exploração ϵ-gananciosa.
final exploration	0.1	Valor final de ϵ na exploração ϵ-gananciosa.
final exploration frame	4000000	O número de frames sobre os quais o valor inicial de ϵ é linearmente reduzido ao seu valor final.
replay start size	200000	Uma política aleatória é executada para este número de frames antes que o aprendizado comece.
no-op max	30	Número máximo de ações "não fazer nada"a serem executadas pelo agente no início de um episódio.

Tabela 4.12: Hiper-parâmetros usados nos experimentos

Parâmetro	**Valor**	**Descrição**
otimizador	RMSProp	Otimizador usado para atualização de pesos no *Gradient Descent*.
função de perda	Huber [2]	Função usada para calcular a perda que o agente busca minimizar em cada passo de otimização.
delta da função de Huber	1	Parâmetro da função de Huber [3].

Tabela 4.13: Detalhes do extras do treinamento

5
Conclusão

Reinforcement Learning tem ganhado destaque como uma abordagem para a resolução de problemas complexos de tomada de decisão e controle. A capacidade dos algoritmos de RL de aprender comportamentos ótimos através de interações não supervisionadas com o ambiente os torna adaptáveis e eficazes em diversas tipos de aplicações. Este estudo investigou o impacto das técnicas de inicialização de pesos e bias no desempenho dos agentes de RL, demonstrando que um início bem planejado pode melhorar a eficiência do aprendizado e a qualidade dos resultados finais.

Nossa pesquisa focou na influência das principais técnicas de inicialização de pesos e bias descritas na literatura sobre o desempenho de agentes de Deep Q-Networks (DQN) em jogos de Atari. Os resultados apresentados mostram a importância de uma escolha adequada dessas técnicas para o desempenho dos agentes. Observamos que, embora algumas técnicas de inicialização possam melhorar a eficiência e eficácia do treinamento, outras podem ter um impacto muito negativo, ressaltando a necessidade de uma seleção criteriosa.

A inicialização de pesos é um fator crucial no treinamento de Redes Neurais, pois uma boa inicialização pode prevenir problemas de gradientes desaparecendo ou explodindo, proporcionando uma base estável para o aprendizado. Este trabalho demonstra que a escolha da técnica de inicialização pode alterar drasticamente o desempenho dos agentes treinados com DQN.

Como trabalho futuro, sugerimos explorar uma variedade mais ampla de técnicas de inicialização de pesos, incluindo o desenvolvimento de novas metodologias ou ajustes nas técnicas existentes. A identificação de combinações mais eficazes para o Q-Learning pode representar um avanço no campo do RL. Além disso, recomendamos uma investigação mais profunda sobre os aspectos do treinamento e as configurações usadas, buscando otimizar ainda mais o desempenho dos agentes de RL.

6
Referências bibliográficas

BELLEMARE, M. et al. The Arcade Learning Environment: An Evaluation Platform for General Agents. **Journal of Artificial Intelligence Research**, v. 47, jul. 2012.

BELLMAN, R. A Markovian Decision Process. **Journal of Mathematics and Mechanics**, v. 6, n. 5, p. 679–684, 1957. ISSN 0095-9057. Publisher: Indiana University Mathematics Department. Disponível em: <https://www.jstor.org/stable/24900506>.

DEGRAVE, J. et al. Magnetic control of tokamak plasmas through deep reinforcement learning. **Nature**, v. 602, n. 7897, p. 414–419, fev. 2022. ISSN 1476-4687. Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/s41586-021-04301-9>.

DULAC-ARNOLD, G. et al. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. **Machine Learning**, v. 110, n. 9, p. 2419–2468, set. 2021. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/s10994-021-05961-4>.

FUKUSHIMA, K. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. **IEEE Transactions on Systems Science and Cybernetics**, v. 5, n. 4, p. 322–333, out. 1969. ISSN 2168-2887. Conference Name: IEEE Transactions on Systems Science and Cybernetics. Disponível em: <https://ieeexplore.ieee.org/document/4082265>.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. JMLR Workshop and Conference Proceedings, 2010. p. 249–256. ISSN: 1938-7228. Disponível em: <https://proceedings.mlr.press/v9/glorot10a.html>.

HE, K. et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: **2015 IEEE International Conference on Computer Vision (ICCV)**. [s.n.], 2015. p. 1026–1034. ISSN: 2380-7504. Disponível em: <https://ieeexplore.ieee.org/document/7410480>.

HE, K. et al. Deep Residual Learning for Image Recognition. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Las Vegas, NV, USA: IEEE, 2016. p. 770–778. ISBN 978-1-4673-8851-1. Disponível em: <http://ieeexplore.ieee.org/document/7780459/>.

KOBER, J.; BAGNELL, J. A.; PETERS, J. Reinforcement Learning in Robotics: A Survey. 2013.

KOCHENDERFER, M. J. **Decision making under uncertainty: theory and application**. Cambridge, Massachusetts: The MIT Press, 2015. (Lincoln Laboratory series). ISBN 978-0-262-02925-4.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, maio 2015. ISSN 0028-0836, 1476-4687. Disponível em: <https://www.nature.com/articles/nature14539>.

LECUN, Y. et al. Handwritten Digit Recognition with a Back-Propagation Network. In: **Advances in Neural Information Processing Systems**. Morgan-Kaufmann, 1989. v. 2. Disponível em: <https://proceedings.neurips.cc/paper/1989/hash/53c3bce66e43be4f209556518c2fcb54-Abstract.html>.

MNIH, V. et al. Playing Atari with Deep Reinforcement Learning. 2013. Disponível em: <https://arxiv.org/abs/1312.5602>.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, v. 518, n. 7540, p. 529–533, fev. 2015. ISSN 1476-4687. Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/nature14236>.

SEO, J. et al. Avoiding fusion plasma tearing instability with deep reinforcement learning. **Nature**, v. 626, n. 8000, p. 746–751, fev. 2024. ISSN 1476-4687. Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/s41586-024-07024-9>.

SILVER, D. et al. Mastering the game of Go with deep neural networks and tree search. **Nature**, v. 529, n. 7587, p. 484–489, jan. 2016. ISSN 1476-4687. Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/nature16961>.

SUTTON, R.; BARTO, A. **Reinforcement Learning, second edition: An Introduction**. 2nd. ed. [S.l.]: Bradford Books, 2018.

SUTTON, R. S. Learning to predict by the methods of temporal differences. **Machine Learning**, v. 3, n. 1, p. 9–44, ago. 1988. ISSN 0885-6125, 1573-0565. Disponível em: <http://link.springer.com/10.1007/BF00115009>.

SUTTON, R. S. et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: **Advances in Neural Information Processing Systems**. MIT Press, 1999. v. 12. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html>.

THOMAS, G.; LUO, Y.; MA, T. Safe Reinforcement Learning by Imagining the Near Future. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2021. v. 34, p. 13859–13869. Disponível em: <https://proceedings.neurips.cc/paper/2021/hash/73b277c11266681122132d024f53a75b-Abstract.html>.

VASWANI, A. et al. Attention is All you Need. In: **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2017. v. 30. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.

WANG, L. et al. **Efficient Reinforcement Learning for Autonomous Driving with Parameterized Skills and Priors**. arXiv, 2023. ArXiv:2305.04412 [cs]. Disponível em: <http://arxiv.org/abs/2305.04412>.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, v. 8, n. 3, p. 279–292, maio 1992. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00992698>.

XIANG, D. Reinforcement learning in autonomous driving. **Applied and Computational Engineering**, v. 48, p. 17–23, mar. 2024.