Treinamento de Agentes de Aprendizado por Reforço no Pac-Man com DQN e MLP

Training of Reinforcement Learning Agents on Pac-Man with DQN and MLP

D. A. Sales; F. R. Santana; H. S. Siqueira; L. D. M. Cavalcanti; N. V. dos Santos

Departamento de computação, UFS, 49100-000, São Cristovão-SE, Brasil

Este estudo investiga o uso de uma abordagem de rede Q profunda (DQN) combinada com um perceptron multicamadas (MLP) para treinar um agente de aprendizagem por reforço (RL) para jogar o jogo Pac-Man. São aplicadas técnicas avançadas, como redes neurais convolucionais para extração de características e estratégias de estabilização de treinamento, incluindo repetição de experiência e redes alvo. As experiências mostram que, apesar das flutuações nas recompensas, o agente continua a melhorar ao longo do processo de treinamento. A análise comparativa de 200, 500 e 1000 episódios mostra que o aprendizado do agente se torna mais consistente, mas os hiperparâmetros ainda precisam ser ajustados para manter a estabilidade. O Monte Carlo Tree Search (MCTS) foi inicialmente considerado, mas abandonado devido ao alto custo computacional.
Palavras-chave: aprendizagem por reforço, rede Q profunda, Pac-Man.

This study investigates the use of a deep Q-network (DQN) approach combined with a multilayer perceptron (MLP) to train a reinforcement learning (RL) agent to play the Pac-Man game. Advanced techniques such as convolutional neural networks for feature extraction and training stabilization strategies including experience replay and target networks are applied. The experiments show that despite fluctuations in rewards, the agent continues to improve throughout the training process. Comparative analysis of 200, 500, and 1000 episodes shows that the agent's learning becomes more consistent, but hyperparameters still need to be adjusted to maintain stability. Monte Carlo Tree Search (MCTS) was initially considered but abandoned due to high computational cost.
Keywords: reinforcement learning, deep Q-network, Pac-Man.

1. INTRODUÇÃO

Os jogos Atari têm sido amplamente utilizados como benchmarks para testar algoritmos de aprendizagem por reforço (RL), proporcionando um ambiente desafiador para o desenvolvimento e avaliação de agentes inteligentes. A complexidade desses jogos combina insights visuais de alta dimensão com tomadas de decisão sequenciais, tornando-os uma área de pesquisa ideal para avanços em aprendizagem profunda aplicada à RL.

Um dos trabalhos pioneiros sobre o uso de aprendizagem profunda para jogar jogos Atari foi apresentado por Mnih et al. (2013), que propuseram a Deep Q Network (DQN), uma rede neural profunda capaz de aprender políticas diretamente a partir de pixels brutos do jogo (Mnih, 2013). Esta abordagem mostrou que agentes treinados com aprendizagem por reforço profundo podem alcançar desempenho equivalente ou melhor que o de jogadores humanos em uma variedade de jogos Atari (Mnih et al., 2015).

Outros métodos complementares são explorados para melhorar a eficiência do treinamento e a capacidade de generalização do agente. Por exemplo, Guo et al. (2014) usaram o Monte Carlo Tree Search (MCTS) offline para gerar dados de treinamento, permitindo que a rede neural aprendesse políticas eficazes sem interação direta com o ambiente (Guo et al., 2014). Trabalhos posteriores, como o de Silver et al. (2017) mostraram que técnicas avançadas de aprendizagem profunda combinadas com a busca em árvore de Monte Carlo podem atingir níveis de desempenho sobre-humanos, conforme demonstrado no jogo Go (Silver et al., 2017).

Neste trabalho, exploramos a construção e o treinamento de agentes baseados em aprendizagem por reforço para jogar Pac-Man usando uma abordagem híbrida que combina diferentes estratégias de aprendizagem. A implementação aproveita a biblioteca Gym, que fornece o ambiente de simulação Atari, bem como frameworks modernos para construção e treinamento de redes neurais, como PyTorch. O agente desenvolvido busca aprender estratégias eficientes para maximizar pontuações no jogo, explorando conceitos como aprendizagem por tentativa e erro, maximização de recompensas e otimização profunda de redes neurais.

O objetivo deste estudo é analisar o desempenho dos agentes, comparar diferentes métodos de RL e compreender os desafios e avanços na área de aprendizagem por reforço aplicada a jogos.

2. METODOLOGIA

A metodologia adotada para o desenvolvimento do agente de Aprendizado por Reforço (RL) no jogo Pac-Man foi estruturada em três etapas principais: configuração do ambiente, implementação do agente e treinamento e avaliação do modelo. Cada uma dessas etapas é descrita a seguir.

2.1 Configuração do Ambiente

Para a simulação do jogo, utilizamos o OpenAI Gym, uma biblioteca amplamente usada para experimentos em RL, que fornece uma interface padronizada para ambientes do Atari. O ambiente selecionado foi o MsPacman-v0, que apresenta desafios estratégicos como navegação no labirinto, coleta de pontos e evasão de inimigos.

Além disso, foram utilizadas as seguintes bibliotecas para suporte ao treinamento do agente:

- PyTorch: Para a implementação da rede neural do agente.
- NumPy e Matplotlib: Para manipulação de dados e visualização de estatísticas do treinamento.
- Gym Atari: Para carregamento do ambiente de jogo no OpenAI Gym.

A simulação foi conduzida em uma máquina com GPU NVIDIA para acelerar o treinamento da rede neural.

2.2 Implementação do agente

O agente foi desenvolvido utilizando uma abordagem baseada na Deep Q-Network (DQN) (Mnih et al., 2015), que combina Redes Neurais Profundas com o algoritmo Q-Learning para aprender uma política ótima a partir da interação com o ambiente. O modelo foi estruturado da seguinte forma:

- Entrada: Frames do jogo processados e convertidos em escalas de cinza, com redução de dimensionalidade para acelerar o aprendizado.
- Rede Neural: Uma CNN (Convolutional Neural Network) com três camadas convolucionais para extração de características visuais, seguida de camadas totalmente conectadas para estimativa das ações.
- Função de Recompensa: Recompensas positivas foram associadas à coleta de pontos e frutas, enquanto penalidades foram aplicadas quando o agente era capturado pelos fantasmas.
- Técnicas de Estabilização: Uso de um buffer de experiência para armazenar e amostrar interações passadas, além da técnica de Target Network para suavizar a atualização dos pesos.

2.2.1 Pré processamento dos frames

Nos jogos do Atari, as imagens brutas da tela precisam ser processadas antes de serem utilizadas pelo agente de Aprendizagem por Reforço Profundo (Deep RL). Neste trabalho, implementamos uma função de pré-processamento de quadros, que converte as imagens originais em um formato mais adequado para a entrada do modelo. Essa função segue abordagens comuns na literatura (Mnih et al., 2015), incluindo conversão para escala de cinza, redimensionamento para 84×84 pixels, normalização para o intervalo [-1,1], conversão para tensor e movimentação para GPU. O código a seguir apresenta a implementação da função de pré-processamento utilizada neste estudo.

```
# Outras importações

import os
import gym
import torch
import torch.nn as nn
import cv2

# Códigos de inicialização


def preprocess_frame(frame):
    # Converte de RGB para escala de cinza e redimensiona
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
    frame_resized = cv2.resize(frame_gray, (84, 84))
    frame_normalized = (frame_resized / 255.0 - 0.5) * 2  # Normaliza para [-1,1]


    # Converte para tensor e move para GPU
    frame_tensor = torch.tensor(frame_normalized, dtype=torch.float32, device=device)
    return frame_tensor


# Restante do código
```

2.2.2 Arquitetura da rede neural

Para que o agente possa tomar decisões no ambiente do Atari Pac-Man, utilizamos uma rede neural baseada no Deep Q-Network (DQN) com melhorias estruturais. A arquitetura implementada combina camadas convolucionais para extração de características da imagem com uma MLP (Multi-Layer Perceptron) mais profunda para modelagem das decisões do agente.

A abordagem segue as diretrizes propostas por Mnih et al. (2015) (Human-level control through deep reinforcement learning), que demonstraram que redes convolucionais podem aprender representações visuais eficientes diretamente dos pixels do jogo. O código a seguir apresenta a implementação da DQNWithEnhancedMLP, que incorpora essas melhorias.

```
//Importações
//Outras partes do código

DQNWithEnhancedMLP(nn.Module):
    def __init__(self, action_size):
```

```
        // parte convolucional
        super(DQNWithEnhancedMLP, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(4, 32, kernel_size=8, stride=4),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Flatten()
        )

       //utilizando uma MLP
        self.mlp = nn.Sequential(
            nn.Linear(64 * 7 * 7, 512),
            nn.ReLU(),
            nn.Linear(512, 256),  # Camadas extras
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Linear(128, action_size)
        )

    def forward(self, x):
        x = self.cnn(x)
        return self.mlp(x)

//Restante do código
```

A parte convolucional (CNN) do modelo é responsável por processar os estados de alta dimensão (por exemplo, as imagens do ambiente Atari) e extrair automaticamente características relevantes.

Após extrair as características com a CNN, o modelo utiliza uma MLP (camadas totalmente conectadas) para mapear essas características em *Q-values* – valores que indicam a qualidade de cada ação em determinado estado.

2.2.3 Por que não implementamos MCTS

Para complementar o DQN, e antes de implementá-lo com uma MLP (Multi-Layer Perceptron), consideramos o uso do MCTS (Monte Carlo Tree Search) para selecionar as melhores ações exploratórias por meio de simulações. Chegamos a desenvolver uma implementação inicial, mas rapidamente percebemos que essa abordagem não seria viável.

No contexto do jogo Atari Pac-Man, enfrentamos desafios que tornaram o MCTS menos eficaz. Esse método baseia-se em simulações repetidas a partir do estado atual para estimar o valor das ações. Em jogos como Go, onde os estados são bem definidos e as simulações são rápidas, o MCTS é altamente eficiente. No entanto, no Pac-Man, onde os estados são representados por imagens e o ambiente possui uma dinâmica complexa, cada simulação exigia um alto custo computacional.

Isso resultou em um tempo de simulação excessivamente longo e em um aumento significativo no uso de memória, tornando a abordagem inviável dentro das nossas limitações computacionais. Com base nesses desafios, optamos por seguir diretamente com a implementação do DQN, que se mostrou mais adequado para lidar com o problema.

2.3 Treinamento e Avaliação

O agente foi treinado utilizando o algoritmo DQN com Replay Memory em uma série de episódios, ajustando hiperparâmetros como taxa de exploração (ε-greedy), taxa de aprendizado (α) e fator de desconto (γ). Durante o treinamento, as seguintes métricas foram monitoradas:

- Score médio por episódio: Mede a eficiência do agente ao longo do tempo.
- Taxa de exploração vs. exploração: Avalia o equilíbrio entre tentativa de novas ações e exploração de ações já conhecidas.
- Convergência da função Q: Verifica a estabilização da política aprendida.

A cada nova sessão de treinamento o modelo era salvo e na próxima seção o modelo mais recente era carregado automaticamente, atualizando os pesos atuais com o do modelo que fica em memória e continuando o treinando a partir do ponto parado no antigo.

3. EXPERIMENTOS

O treinamento foi realizado em algumas fases com diferentes parâmetros, conforme percebemos necessidade de ajuste. Dentre os parâmetros temos:

- **Número de episódios:** Define o número total de episódios que o agente irá executar durante o treinamento, onde cada episódio corresponde a uma partida completa (do início ao fim) do jogo.
- **Fator Gamma**: Esse é o fator de desconto. Ele determina o quanto as recompensas futuras são valorizadas em relação às recompensas imediatas.
- **Taxa de Aprendizad**o: A taxa de aprendizado usada pelo otimizador. Responsável por controlar o tamanho dos passos dados na atualização dos pesos da rede.
- **Epsilon**: É o valor inicial de epsilon na estratégia epsilon-greedy, que controla a exploração.
- **Epsilon min**: Esse é o valor mínimo para epsilon. Conforme o treinamento avança, o epsilon decai (diminuindo a exploração) até atingir esse valor mínimo.
- **Epsilon decay**: Fator é usado para reduzir (decair) o valor de epsilon a cada episódio.
- **Buffer size:** Define o tamanho máximo do replay buffer, que é a memória onde o agente armazena suas experiências (estados, ações, recompensas, próximos estados, e se o episódio terminou).
- **Batch size:** Indica quantas amostras serão retiradas (randomicamente) do replay buffer a cada atualização da rede.

3.1 Parâmetros nas fases principais

Abaixo segue uma tabela que indica os parâmetros utilizados em cada fase principal.

Tabela 1: Parâmetros por fase.

Parâmetros	Fase 1	Fase 2	Fase 3
nº de episódios	200	500	1000
fator gamma	0.99	0.99	0.99
learning rate	0.0001	0.00005	0.00005

epsilon	1.0	1.0	1.0
epsilon min	0.1	0.2	0.2
epsilon decay	0.995	0.98	0.98
buffer size	100000	100000	100000
batch size	32	64	256

Durante o treinamento, o agente armazenou experiências em um buffer e realizou aprendizado por lotes utilizando a função de perda MSELoss().

Embora cada fase na tabela conta com um número de episódios fixo, treinamentos com números de episódios menores (ou maiores) dentro dos intervalos de cada Fase, e respeitando os outros parâmetros, foram realizados para aumentar a interação do agente. Ainda, a divisão em várias fases de treinamento também se deve ao fato da nossa limitação de acesso a GPU nas contas do Google Colab.

4. RESULTADOS E DISCUSSÃO

4.1 Resultado do Treinamento na Fase 1

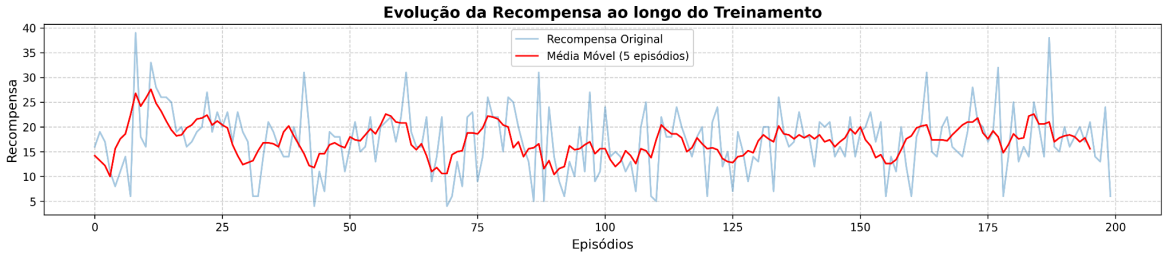

Figura 4.1: Gráfico de Evolução da Recompensa da Fase 1

Na figura 4.1 a evolução da recompensa (linha azul) representa a recompensa em cada episódio, e a linha vermelha é a média móvel a cada 5 episódios. Há uma oscilação na recompensa episódio a episódio, o que indica que o agente ainda não estabilizou completamente seu desempenho. A média móvel (linha vermelha) sugere que, embora haja episódios com recompensas mais altas, não há uma tendência de crescimento clara. Ela oscila ao redor de um nível intermediário.

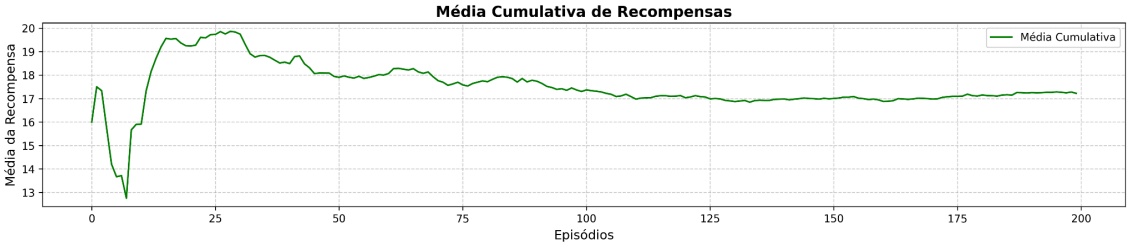

Figura 4.2: Gráfico de Média Cumulativa de Recompensas da Fase 1

Inicialmente, a média cumulativa de recompensas começa relativamente alta e, conforme o treinamento avança, cai para um patamar mais baixo e depois se estabiliza. Essa estabilização pode indicar que o agente estava adotando uma estratégia que não melhora a longo prazo, mas também não degradava a performance de forma severa.

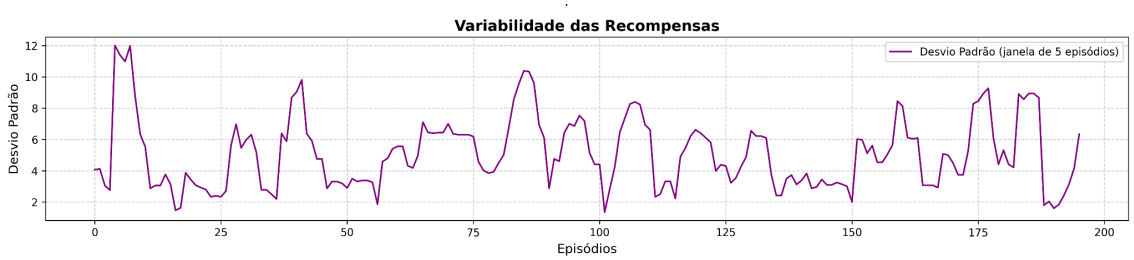

Figura 4.3: Gráfico de Variabilidade de Recompensas da Fase 1

A linha roxa exibe o desvio padrão das recompensas em uma janela de 5 episódios (ou seja, quão dispersas estão as recompensas dentro de cada janela). Vê-se que em alguns pontos há picos de variabilidade, sugerindo que o agente às vezes tem episódios muito bons e, logo depois, episódios ruins. Uma variabilidade menor, acompanhada de recompensas mais altas, seria o cenário ideal, pois indicaria consistência no desempenho.

4.2 Resultado do Treinamento no Fase 2

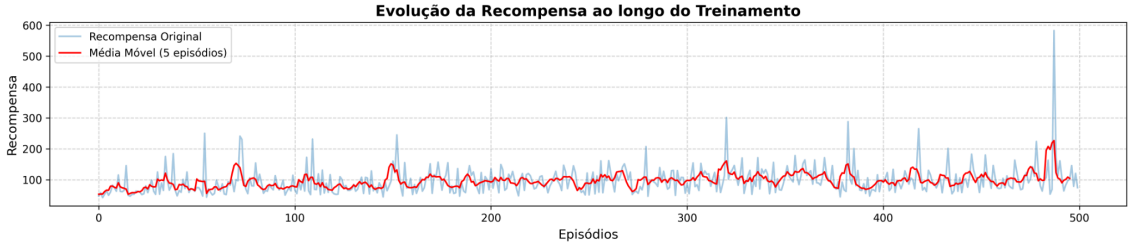

Figura 4.4: Gráfico de Evolução da Recompensa da Fase 2

Apesar de haver picos em alguns episódios, há uma tendência de crescimento ao longo do tempo, embora com oscilações. Isso sugere que o agente ocasionalmente obtém recompensas mais altas, mas ainda passa por episódios com recompensas baixas ou nulas.

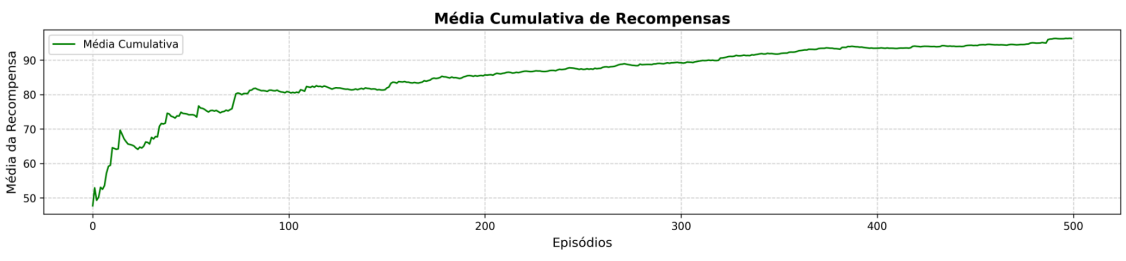

Figura 4.5: Gráfico de Média Cumulativa de Recompensas da Fase 2

Esse gráfico sobe de forma relativamente consistente, o que indica melhora geral do desempenho do agente ao longo dos episódios. Quando a curva se estabiliza ou cresce lentamente, significa que o agente está se aproximando de uma política mais consistente. Nesse caso, vemos um crescimento praticamente constante, o que é um sinal positivo de aprendizado.

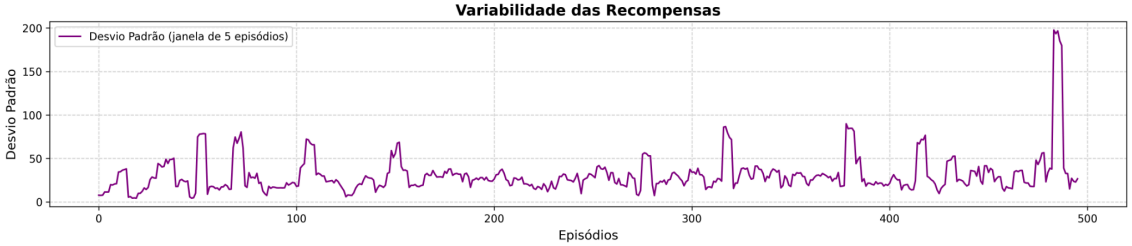

Figura 4.6: Gráfico de Variabilidade de Recompensas da Fase 2

Na maior parte do tempo, a variabilidade permanece em um patamar relativamente baixo, mas com alguns picos. Esses picos sinalizavam episódios "fora da curva" — em que o agente teve desempenho muito melhor (ou pior) do que a média. A tendência de manter o desvio padrão geralmente baixo (exceto em picos esporádicos) sugere que o agente está adquirindo alguma estabilidade na maneira de jogar, mas ainda há certa imprevisibilidade em algumas situações.

4.3 Resultado do Treinamento no Fase 3

Figura 4.7: Gráfico de Evolução da Recompensa da Fase 3

Há episódios em que o agente obtém recompensas mais altas, intercalados com episódios de recompensas baixas. A média móvel vermelha não apresenta uma tendência de crescimento muito acentuada, mas parece manter-se em um patamar ligeiramente crescente ou quase estável.

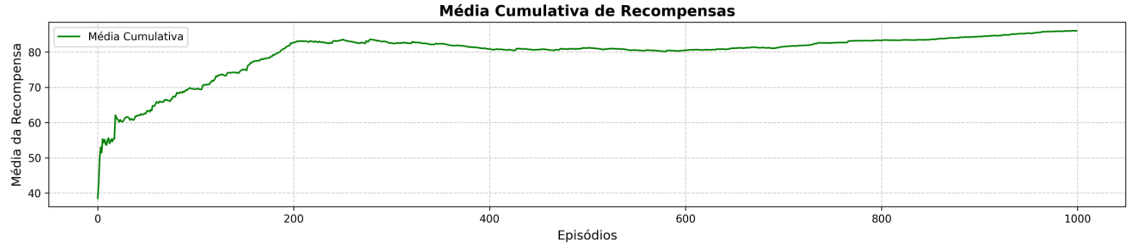

Figura 4.8: Gráfico de Média Cumulativa de Recompensas da Fase 3

A curva sobe de forma relativamente consistente, indicando melhora geral ao longo dos episódios. Em torno de 300 a 400 episódios, há um ganho mais rápido, e depois o crescimento desacelera, mas continua positivo. A média cumulativa está subindo, mostrando que, de modo geral, o agente tem se tornado melhor ao longo do tempo.

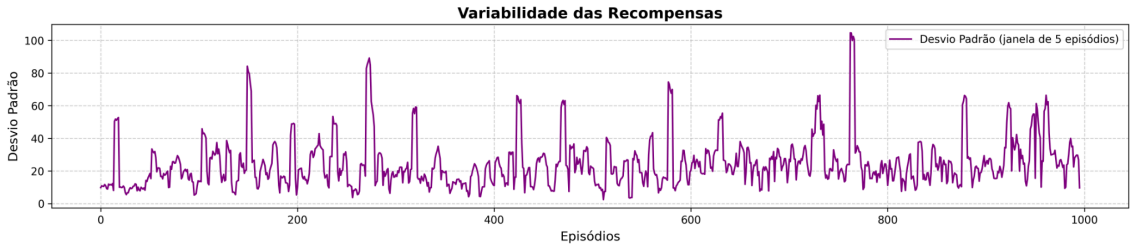

Figura 4.9: Gráfico de Variabilidade de Recompensas da Fase 3

Nesta Fase há vários picos de variabilidade, mostrando que, ocasionalmente, a performance varia muito entre episódios próximos. Mesmo com esses picos, não há uma tendência clara de aumento ou diminuição sustentada da variabilidade, indicando que o agente ainda alterna entre bons e maus episódios. Isso pode ocorrer se o agente ainda estiver explorando bastante, o que pede um ajuste no esquema de exploração (*epsilon decay*) .

4.4 Análise Geral:

As três séries de resultados evidenciam um progresso gradativo do agente. De 200 para 500 episódios, há sinais mais claros de melhora, e de 500 para 1000 episódios a média cumulativa sobe ainda mais, embora com picos e oscilações. O agente não está 100% consistente, mas está aprendendo cada vez mais e obtendo recompensas médias superiores ao longo do tempo. No geral, o agente está aprendendo, porém de forma relativamente lenta e com episódios bons e ruins intercalados. Essa dinâmica é comum em aprendizado por reforço; com mais tempo de treino e ajustes, é possível que o agente estabilize em um patamar de recompensas mais elevadas.

5. CONCLUSÃO

Neste trabalho, exploramos a construção e o treinamento de agentes de aprendizagem por reforço (RL) para jogar Pac-Man usando uma abordagem de rede Q profunda (DQN) aumentada por perceptron multicamadas (MLP). Implementamos técnicas avançadas de aprendizagem, incluindo redes neurais convolucionais para extrair recursos visuais e treinar estratégias de estabilização, como repetição de experiência e redes alvo.

Experimentos mostram que ao longo da fase de treinamento, o agente consegue aprender gradativamente como jogar, melhorar seu desempenho e acumular recompensas maiores. No entanto, observamos mudanças de desempenho ao longo dos episódios, sugerindo que o ajuste de hiperparâmetros, como taxa de exploração e fator de desconto, pode ajudar a melhorar a estabilidade e a eficiência do aprendizado.

A média cumulativa aumenta após comparar a recompensa para cada uma das três etapas (200, 500 e 1000 episódios), o que significa que o agente está aprendendo algo útil no ambiente. Em todas as etapas, há oscilações nas recompensas de um episódio para outro. Este é um comportamento padrão para algoritmos de Aprendizado por Reforço quando estão em ambientes complexos. No entanto, aumentando os episódios, os agentes começam a explorar menos (se ε decresce) e a consolidar a política aprendida. Para alcançar políticas genuinamente robustas, políticas para jogos de Atari, como Pac-Man, por exemplo, são conhecidas por necessitar de milhares ou até milhões de etapas. Portanto, mesmo 1000 episódios podem não ser suficientes para se chegar a um ótimo local. Porém, a evolução dos gráficos mostra que cada fase adiciona aprendizado.

A abordagem inicial do MCTS foi considerada inviável em decorrência do custo computacional e do longo tempo de simulação, confirmando que o DQN era adequado para este tipo de problema. Um segundo elemento da análise dos resultados foi entender que, mesmo que

o agente tenha dado passos importantes, ainda pode ser aprimorado com técnicas mais avançadas, como Prioritized Experience Replay ou dueling DQN, para aprimorar sua tomada de decisão e eficiência de aprendizado.

Este estudo reforça a importância da combinação entre redes neurais profundas e aprendizado por reforço na criação de agentes inteligentes capazes de aprender em ambientes complexos.

6. PRÓXIMOS PASSOS

Os próximos passos incluem a ampliação e o refinamento do treinamento do agente, visando uma convergência mais robusta e desempenho superior. Primeiramente, será necessário aumentar significativamente o número de episódios de treinamento – considerando que ambientes complexos como o Pac-Man no Atari frequentemente exigem milhares de episódios para que a política se consolide. Além disso, os hiperparâmetros relacionados à exploração, como o valor inicial e mínimo de epsilon e a taxa de decaimento, deverão ser ajustados para reduzir gradualmente a exploração e favorecer a utilização da política aprendida.

Outros pontos a serem otimizados incluem a função de recompensa, por meio de ajustes no bônus por sobrevivência e na penalidade de término, para incentivar episódios mais longos e minimizar terminações prematuras. A capacidade do replay buffer também pode ser revista para garantir a diversidade necessária de experiências. A possibilidade de explorar variações na arquitetura da rede, especialmente na parte da MLP, poderá contribuir para uma melhor aproximação da função de valor. Por fim, o monitoramento contínuo dos gráficos de recompensas e variabilidade servirá para avaliar o impacto desses ajustes e orientar futuras modificações.

7. REFERÊNCIAS BIBLIOGRÁFICAS

1. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al. OpenAI Gym [Internet]. arXiv preprint arXiv:1606.01540; 2016 [citado em 24 fev 2025]. Disponível em: https://arxiv.org/abs/1606.01540
2. Guo X, Singh S, Lee H, Lewis R, Wang X. Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In: Advances in Neural Information Processing Systems (NIPS); 2014. [citado em 24 fev 2025]
3. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning [Internet]. arXiv preprint arXiv:1312.5602; 2013 [citado em 24 fev 2025]. Disponível em: https://arxiv.org/abs/1312.5602
4. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015 Feb;518(7540):529-33, doi:10.1038/nature14236 [citado em 24 fev 2025]
5. OpenAI. ChatGPT: modelo de linguagem baseado em IA [Internet]. OpenAI; 2025 [citado em 24 fev 2025]. Disponível em: https://chat.openai.com
6. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Mastering the game of Go without human knowledge. Nature. 2017 Oct;550(7676):354-9, doi:10.1038/nature24270 [citado em 24 fev 2025]
7. Russell S, Norvig P. Artificial Intelligence: A Modern Approach [Internet]. Fourth Edition. Pearson; 2020 [citado em 24 fev 2025]. Disponível em: https://aima.cs.berkeley.edu