Jogando Breakout com Aprendizado por Reforço Profundo

Playing Breakout with Deep Reinforcement Learning

E. S. Ramos ; C. C. Aquino ; R. P. Leite ; L. S. Lima ; J. F. F. Neto

[1] *Departamento de Computação/Laboratório D02, Universidade Federal de Sergipe, 49100-000, São Cristóvão-Sergipe, Brasil*

{edcarlosufs , riquelmewin , cainacastro29 , falcao ,waynewyn } @ academico.ufs.br

O desenvolvimento de agentes inteligentes capazes de jogar videogames tem sido um tema amplamente investigado na área de Inteligência Artificial, impulsionado por técnicas como Aprendizado Profundo (Deep Learning) e Aprendizado por Reforço (Reinforcement Learning). Este estudo propõe a implementação de um agente inteligente para jogar Breakout utilizando a arquitetura Deep Q-Network (DQN). O DQN emprega redes neurais profundas para estimar os valores das ações e otimizar o processo de tomada de decisões com base em representações visuais do ambiente. O jogo Breakout, um dos benchmarks mais conhecidos no campo do Aprendizado por Reforço, apresenta desafios como o controle sequencial de ações, a interpretação de estados visuais e o equilíbrio entre exploração e explotação. Nossa abordagem utiliza o ambiente OpenAI Gym para o treinamento e avaliação do agente. Espera-se que os resultados demonstrem a eficiência do DQN na aprendizagem de estratégias otimizadas, contribuindo para o avanço no desenvolvimento de agentes autônomos em ambientes de jogos.

1. Introdução

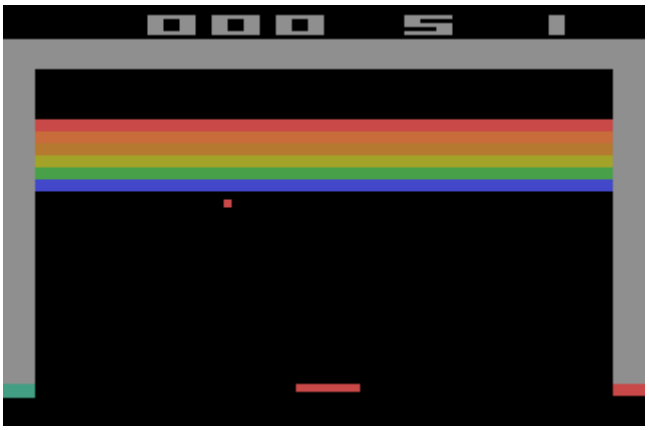
O desenvolvimento de agentes inteligentes capazes de jogar videogames tem sido um campo de pesquisa amplamente explorado na Inteligência Artificial (IA), especialmente após o avanço das técnicas de Aprendizado Profundo (Deep Learning) e Aprendizado por Reforço (Reinforcement Learning). Jogos clássicos, como Breakout, fornecem um ambiente ideal para testar e aprimorar essas técnicas, pois apresentam desafios que exigem planejamento, adaptação e aprendizado de estratégias otimizadas ao longo do tempo.

O uso de Redes Neurais Profundas (DNNs) em conjunto com o Aprendizado por Reforço resultou no desenvolvimento do Deep Q-Network (DQN), uma técnica introduzida por Mnih et al. (2015), que possibilitou que agentes treinados ultrapassassem o desempenho humano em vários jogos da plataforma Atari 2600. O DQN emprega uma rede neural para estimar a função de valor de ação (Q-learning), permitindo que o agente aprenda estratégias de tomada de decisão eficazes diretamente a partir de representações visuais do ambiente. Desde então, diversas melhorias foram sugeridas, como o Double DQN (van Hasselt et al., 2016), o Prioritized Experience Replay (Schaul et al., 2016) e o Dueling DQN (Wang et al., 2016), todas com o objetivo de aumentar a estabilidade e eficiência do processo de aprendizado.

Deep Q-Network (DQN) é um algoritmo de aprendizado por reforço profundo que combina o Q-Learning com redes neurais profundas para aprender estratégias ótimas em ambientes complexos. O DQN revolucionou o campo da IA ao permitir que agentes superassem o desempenho humano em diversos jogos da Atari. O aprendizado por reforço consiste em um agente que interage com um ambiente, recebe recompensas e ajusta suas ações para maximizar ganhos futuros. O DQN resolve desafios como a alta dimensionalidade dos estados e ações usando Replay Buffer (armazenamento e reutilização de experiências) e Target Network (rede auxiliar para estabilidade). Além disso, extensões como Double Q-Learning, Prioritized Experience Replay e Dueling Network Architecture aprimoram a eficiência e estabilidade do aprendizado.

O jogo Breakout, disponível no conjunto de ambientes Atari Gym da OpenAI, é um dos benchmarks mais populares para experimentos em Aprendizado por Reforço. Lançado originalmente pela Atari em 1976, o jogo desafia o jogador a controlar uma barra móvel na

parte inferior da tela para rebater uma bola e destruir tijolos no topo. O objetivo é quebrar todos os tijolos sem deixar a bola cair.



No OpenAI Gym, o ambiente "Breakout-v0" (e suas variantes, como "BreakoutNoFrameskip-v4") fornece uma interface padronizada para experimentos de RL. O agente recebe como entrada imagens da tela (frames do jogo) e pode executar quatro ações discretas: manter-se parado, mover para a esquerda, mover para a direita e lançar a bola. O jogo retorna uma recompensa positiva quando um tijolo é destruído e termina quando todas as vidas são perdidas. Os principais desafios do Breakout para agentes de RL incluem: Controle sequencial de ações, O agente precisa planejar seus movimentos antecipadamente para rebater a bola corretamente. Observação baseada em pixels, como a entrada é a imagem do jogo, o modelo deve aprender a interpretar estados visuais e tomar decisões com base neles. Exploração e Exploração, O agente deve encontrar estratégias eficazes (como criar túneis nas laterais) enquanto explora novas abordagens.

Neste trabalho, propomos a criação de um agente inteligente que joga Breakout utilizando a arquitetura DQN. Nossa abordagem consiste em treinar um modelo de rede neural profunda para aprender estratégias eficazes por meio da interação com o ambiente.

2. METODOLOGIA

2.1 Detalhes do Ambiente

O treinamento do agente foi realizado no ambiente **Breakout-v4**, disponibilizado pelo **Arcade Learning Environment (ALE)**. O ALE fornece uma plataforma padronizada para a avaliação de agentes de aprendizado por reforço em jogos da plataforma Atari 2600.

2.1.1 Parâmetros do Ambiente

- **Versão do ambiente:** *Breakout-v4*
- **Frame Skip:** 3

O parâmetro **frame skip** determina a frequência com que o agente executa ações no ambiente. Em vez de processar cada quadro individualmente, o agente seleciona uma ação e a mantém por 3 frames consecutivos. Essa técnica reduz a complexidade computacional e melhora a estabilidade do aprendizado, pois evita que o agente precise decidir ações a cada quadro, o que pode resultar em variações desnecessárias e comportamentos ruidosos.

2.1.2 Formato do Estado

O estado do ambiente é representado por uma imagem do jogo renderizada a cada quadro. Cada estado é uma matriz tridimensional **(H,W,C)** onde:

- H representa a altura da imagem (originalmente 210 pixels).
- W representa a largura da imagem (originalmente 160 pixels).
- C corresponde ao número de canais da imagem (originalmente 3 canais RGB).

2.2 Pré-processamento das Imagens

Para garantir que as imagens do ambiente sejam adequadas como entrada para a rede neural, foi realizado um pré-processamento composto pelas seguintes etapas:

- **Conversão para escala de cinza:** A imagem original, capturada em formato RGB, foi convertida para escala de cinza. Essa conversão reduz a dimensionalidade da entrada, eliminando informações cromáticas desnecessárias, o que contribui para a eficiência computacional sem comprometer a capacidade do agente de interpretar a dinâmica do ambiente.
- **Redimensionamento:** As imagens foram redimensionadas para um formato fixo de 84 x 84 pixels. Essa padronização permite que a rede convolucional receba entradas de tamanho consistente, facilitando a extração de características visuais relevantes.
- **Conversão para tensor e normalização:** Após o redimensionamento, as imagens foram convertidas para um tensor do PyTorch e normalizadas. A normalização foi aplicada para estabilizar o processo de treinamento, garantindo que os valores dos pixels estejam dentro de uma faixa adequada para a propagação dos gradientes na rede neural.

2.3 Arquitetura da Rede Neural

A rede convolucional profunda utilizada é composta por três camadas convolucionais seguidas por duas camadas totalmente conectadas (**fully connected**, FC).

- **Camadas Convolucionais:**
  - **Camada 1:** 32 filtros de tamanho 8×8 com **stride** 4.
  - **Camada 2:** 64 filtros de tamanho 4×4 com **stride** 2.
  - **Camada 3:** 64 filtros de tamanho 3×3 com **stride** 1.
  - A saída das camadas convolucionais possui dimensões 64 x 7 x 7.
- **Camadas Totalmente Conectadas:**
  - **Flatten :** Transforma a saída da última camada convolucional de dimensão 64 × 7 × 7 em um vetor unidimensional de 3136 neurônios (64 × 7 × 7 = 3136), preparando os dados para as camadas totalmente conectadas.
  - **FC1:**
    - Entrada: 3136 neurônios.
    - Saída: 512 neurônios.
    - Função de ativação: **ReLU**.
  - **FC2:**
    - Entrada: 512 neurônios.
    - Saída: 4 neurônios, onde cada neurônio representa a estimativa da **função de valor de ação Q(s,a)** para uma das quatro ações possíveis do agente.

Essa arquitetura permite que o modelo aprenda uma representação eficiente do ambiente e tome decisões de ação baseadas na estimativa da função Q.

Pseudocódigo CNN:

```
Classe CNN:
    Entrada: action_size (Número de ações possíveis no ambiente)

    Definir Camadas:
        - Convolução 1: (canal_de_entrada1, canal_de_saída1, tamanho do kernel conv1, valor do stride conv1)
        - Convolução 2: (canal_de_saída1, canal_de_saída2, tamanho do kernel conv2, valor do stride conv2)
        - Convolução 3: (canal_de_saída2, canal_de_saída3, tamanho do kernel conv3, valor do stride conv3)
        - Camada totalmente conectada 1: (FC1 unidade de entrada, FC1 unidade de saída)
        - Camada totalmente conectada 2: (FC1 unidade de saída, ACTION_SIZE)

    Método forward(x):
        Aplicar função de ativação ReLU na saída da Convolução 1
        Aplicar função de ativação ReLU na saída da Convolução 2
        Aplicar função de ativação ReLU na saída da Convolução 3

        Achatar saída para um vetor (Flatten)

        Aplicar função de ativação ReLU na saída da Camada totalmente conectada 1

        Retornar saída da Camada totalmente conectada 2 (Q-values das ações)
```

2.4 Especificações do ambiente de hardware

Para a realização dos experimentos, foi utilizada a assinatura do Google Colab Pro, garantindo acesso a recursos computacionais aprimorados. O ambiente de execução foi configurado para utilizar uma **GPU NVIDIA T4** com 15 GB de memória VRAM, possibilitando um treinamento mais eficiente da rede neural profunda. Além disso, o sistema dispunha de 51 GB de memória RAM, proporcionando uma melhor manipulação de grandes conjuntos de dados e buffers de experiência. O armazenamento disponível no ambiente era de 235,7 GB de espaço em disco, sendo 15 GB dedicados ao armazenamento temporário de execução.

2.5 Sistema de recompensas:

2.5.1 Recompensa Base do Ambiente

O agente recebe recompensas positivas ao obter bons resultados no ambiente do jogo. A recompensa original é multiplicada por uma constante. Após um número de passos constante, a recompensa é multiplicada novamente, incentivando a longevidade no jogo.

2.5.2 Penalidade por Perda de Vida

O código verifica se o número de vidas mudou. Se o agente perde uma vida, ele recebe uma penalidade. Isso incentiva o modelo a evitar ações que resultem em perdas de vida.

2.6 Hiperparâmetros

Tabela 1:

	Parâmetros de Treinamento
Batch_Size	16
Gamma	0.99
Epsilon	1.0
Epsilon_min	0.05
Epsilon_decay	0.001
Learning_rate	0.00025
Buffer_size	80000
Target_update	2000

Tabela 2:

	Parâmetros de Recompensa
Recompensa	Recompensa bruta oriunda do ambiente
Penalidade por perder vida	-1
Multiplicador recompensa	2
Passos para dobrar recompensa	500

Pseudocódigo AgentDQN:

```
Clase treinamentoAgentDQN:
    Para cada episódio em NUM_EPISODES:
        Inicializar frame_stack (buffer para armazenar os últimos 4 frames)
        Reiniciar o ambiente e processar o primeiro frame
        Obter o número inicial de vidas do jogador
        Preencher o frame_stack com o primeiro frame 4 vezes
        Criar o estado empilhando os 4 frames
        Inicializar total_reward, action_noop_count e pontuacao

        Para cada passo no episódio (PASSOS_POR_EPISODIO):
            Selecionar uma ação com base na política atual e estado empilhado
            Executar a ação no ambiente e obter o próximo estado, recompensa e status do jogo
            Ajustar a recompensa:
                - Dobrar a recompensa básica
                - Se passaram mais de 400 passos, dobrar a recompensa novamente
                - Se a recompensa for positiva, incrementar a pontuação
                - Se a ação for 'no-op' (ação 0), incrementar action_noop_count
                - Se o número de vidas diminuir, aplicar penalidade por perda de vida

            Atualizar o estado empilhado com o próximo frame processado
            Armazenar a transição no buffer de experiência
            Atualizar o estado atual
            Treinamento da DQN:
                Se o buffer de replay tiver amostras suficientes:
                    - Amostrar um mini-batch de transições
                    - Calcular os Q-values da política atual para as ações escolhidas
                    - Calcular os Q-values esperados usando a rede-alvo
                    - Calcular a perda (SmoothL1Loss)
                    - Atualizar os pesos da rede neural
            Atualizar a rede-alvo periodicamente
            Se o jogo terminou, exibir o número de passos e sair do loop
        Reduzir o valor de epsilon para controle da exploração
```

3. EXPERIMENTOS

Para avaliar o desempenho do agente treinado utilizando a abordagem Deep Q-Network (DQN), realizamos experimentos no ambiente Breakout-v4 do Gymnasium. O agente foi treinado ao longo de 5.000 episódios, utilizando os hiperparâmetros previamente definidos. Durante o treinamento, a política de exploração-explotação foi ajustada por meio de um decaimento ε-greedy, permitindo que o agente equilibrasse a exploração de novas ações e a exploração das melhores estratégias aprendidas.

A função de perda foi minimizada utilizando o otimizador Adam, com uma taxa de aprendizado ajustada para favorecer a estabilidade do treinamento. Além disso, implementamos Replay Buffer para armazenar e amostrar experiências passadas, reduzindo a correlação entre

atualizações consecutivas e melhorando a convergência do modelo. Para mitigar a instabilidade na estimativa da função de valor, utilizamos uma rede-alvo (target network) atualizada periodicamente.

Após o término do treinamento, realizamos testes com o agente treinado ao longo de 112 episódios para avaliar seu desempenho em um cenário de execução sem exploração adicional. O gráfico apresentado na Figura 1 ilustra a evolução da pontuação do agente ao longo desses episódios. Observamos uma tendência de melhora na pontuação média, indicando progresso no aprendizado. No entanto, a pontuação ainda apresenta variabilidade significativa, sugerindo possíveis instabilidades no processo de treinamento, o que pode estar relacionado a fatores como escolha dos hiperparâmetros ou necessidade de um tempo de treinamento mais prolongado.

Para uma análise quantitativa mais detalhada, a Tabela 3 apresenta a maior pontuação registrada durante os experimentos, representando o número máximo de blocos destruídos em uma única partida.

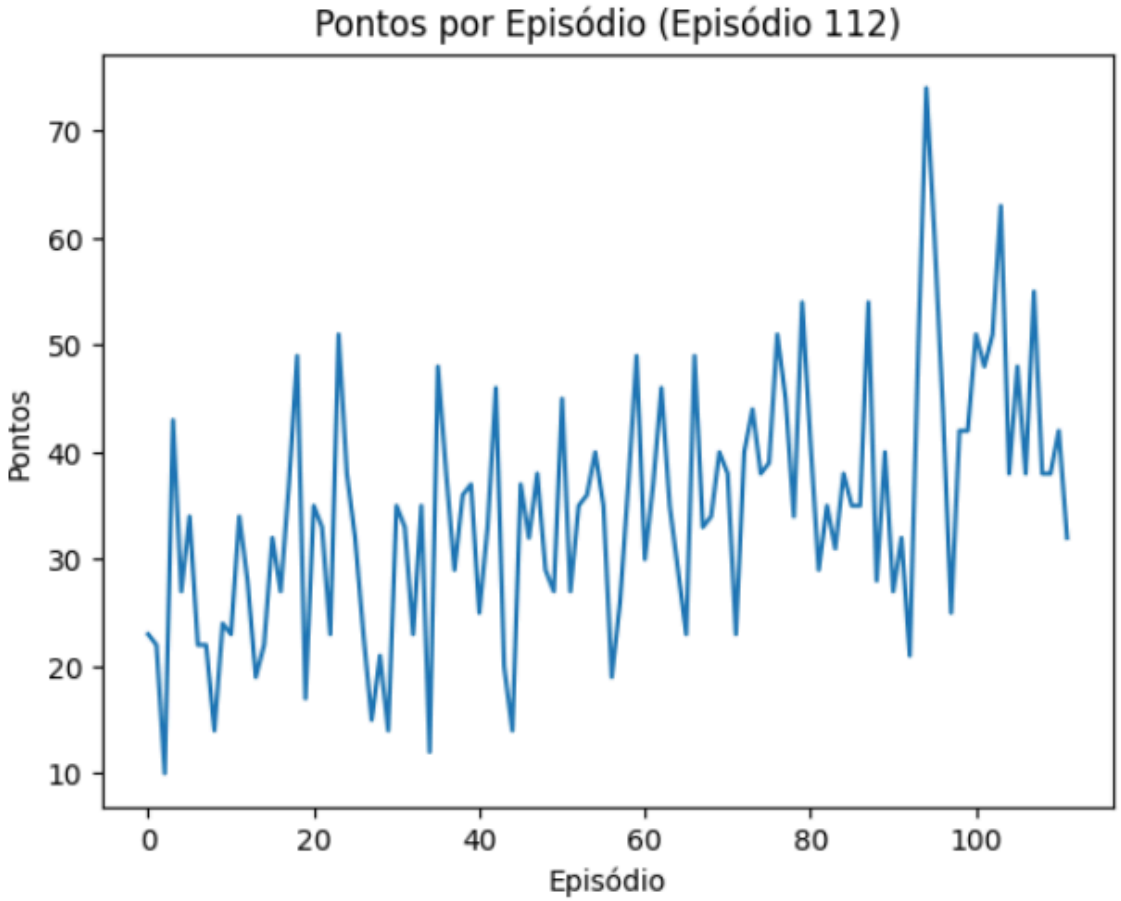


Tabela 3:

	Tipos de Pontuações máximas
Recorde	**Breakout-v4**
Blocos destruídos	41
Por pontos (do jogo)	132

4. Resultados e Discussão

O agente treinado com DQN alcançou um máximo de 41 blocos destruídos, um desempenho razoável, mas ainda inferior ao ótimo esperado para esse ambiente. Essa limitação pode estar relacionada à dificuldade do agente em aprender estratégias avançadas, como controle refinado da bola para otimizar a destruição dos blocos superiores.

Os resultados obtidos indicam que, embora o DQN tenha sido capaz de aprender padrões básicos do jogo, ele apresenta limitações na aquisição de estratégias mais sofisticadas. Isso pode ser atribuído a diferentes fatores:

- Exploração limitada: O algoritmo DQN depende de uma estratégia ε-greedy, que pode não ser suficiente para garantir uma exploração eficiente do espaço de ações. Métodos mais avançados, como Exploração com Prioritized Experience Replay (PER), poderiam mitigar esse problema.
- Oscilações no aprendizado: A variação na pontuação sugere que o agente pode estar sofrendo com instabilidades na função de valor, algo que poderia ser minimizado com abordagens como Double DQN ou Dueling DQN.
- Eficiência da função de recompensa: O agente pode estar otimizando sua recompensa sem desenvolver táticas de longo prazo, como manter a bola presa na parte superior do cenário para destruir múltiplos blocos sem interferência. A reformulação da função de recompensa, incluindo incentivos para atingir regiões superiores, poderia aprimorar esse comportamento.

5. Conclusão

Durante a fase de testes, foi constatada uma instabilidade no desempenho do agente final, associada à emergência de comportamentos viciantes indesejáveis no modelo treinado. Tais comportamentos podem comprometer a capacidade de generalização e eficiência do agente em diferentes situações dentro do ambiente.

Para mitigar esses problemas, propõem-se as seguintes melhorias:

- **Aplicação de técnicas de regularização na rede neural**: A introdução de métodos como dropout, weight decay ou batch normalization pode ajudar a reduzir o overfitting e melhorar a estabilidade do aprendizado.
- **Aumento do número de episódios de treinamento**: Treinar o agente por mais episódios permitirá uma melhor convergência dos parâmetros da rede, reduzindo oscilações e garantindo um comportamento mais robusto.
- **Modificar a política de exploração**: Tentar trocar a política de exploração pode evitar que o agente fique preso em comportamentos viciantes que limitam seu desempenho.

6. Referências Bibliográficas

1. BELLEMARE, M. G.; NADDAF, Y.; VENESS, J.; BOWLING, M. *The Arcade Learning Environment: An Evaluation Platform for General Agents*. Journal of Artificial Intelligence Research, v. 47, p. 253-279, jun. 2013.
2. MNIH, Volodymyr. *Playing atari with deep reinforcement learning*. arXiv preprint, arXiv:1312.5602, 2013. Disponível em: https://tinyurl.com/atari-rl. Acesso em: 14 Fev. 2025.

3. MNIH, Volodymyr; et al. *Human-level control through deep reinforcement learning.* Nature, v. 518, n. 7540, p. 529-533, 2015. Disponível em: https://tinyurl.com/human-level-nature. Acesso em: 14 Fev. 2025.
4. SILVER, David; et al. *Mastering the game of Go without human knowledge.* Nature, v. 550, n. 7676, p. 354-359, 2017. Disponível em: https://tinyurl.com/mastering-go. Acesso em: 14 Fev. 2025.
5. GYMNASIUM. *Basic Usage.* Disponível em: https://gymnasium.farama.org/introduction/basic_usage/. Acesso em: 14 Fev. 2025.

6. GOOGLE. *Google Colaboratory.* Disponível em: https://colab.google/. Acesso em:20 Fev. 2025