SCIENTIA PLENA VOL. 1, NUM. 1 2025
www.scientiaplena.org.br doi: 10.14808/sci.plena.2025.XXXXXX

Space Invaders com Aprendizado por Reforço - DQN e CNN

Space Invaders with Reinforcement Learning - DQN and CNN

B. F. Vieira[1]; D. S. F. Santos[2]; H. C. Júnior[3]

N. S. S. Júnior[4]; R. N. Andrade[5]

Departamento de Computação, Universidade Federal de Sergipe, 49107-230, Aracaju-SE, Brasil

hendrik@dcomp.ufs.br
(Recebido em 24 de fevereiro de 2025)

Resumo:
Este estudo explorou a aplicação do aprendizado por reforço profundo, utilizando Deep Q-Networks (DQN) e Redes Neurais Convolucionais (CNN), para treinar um agente no jogo Space Invaders dentro do Arcade Learning Environment (ALE). O modelo foi avaliado ao longo de 2440 iterações, demonstrando um aumento significativo na taxa de recompensa e na qualidade das decisões tomadas pelo agente. Os resultados indicam que a combinação de DQN com CNN permite uma percepção mais precisa do ambiente, resultando em uma melhor estratégia de jogo. No entanto, desafios como o tempo de treinamento e a convergência do modelo são pontos que devem ser aprimorados em pesquisas futuras.

Palavras-chave: Aprendizado por Reforço, Deep Q-Network, Redes Neurais Convolucionais, Arcade Learning Environment.

Abstract:
This study explored the application of deep reinforcement learning using Deep Q-Networks (DQN) and Convolutional Neural Networks (CNN) to train an agent in the game *Space Invaders* within the Arcade Learning Environment (ALE). The model was evaluated over 2,440 iterations, demonstrating a significant increase in the reward rate and the quality of the agent's decision-making. The results indicate that combining DQN with CNN enables a more accurate perception of the environment, leading to improved gameplay strategy. However, challenges such as training time and model convergence remain areas for improvement in future research.

Keywords: Reinforcement Learning, Deep Q-Network, Convolutional Neural Networks, Arcade Learning Environment.

1. INTRODUÇÃO

O desenvolvimento de agentes independentes a partir de dados provenientes de sensores de alta dimensão, como fala e visão, tem sido um grande desafio no campo do aprendizado por reforço. No entanto, com o avanço das redes neurais convolucionais, que permitem a extração eficiente [1] de informações úteis a partir de imagens, tornou-se possível avançar significativamente nessa área. O progresso das tecnologias de aprendizado profundo tem impulsionado melhorias notáveis nos problemas de percepção dentro da inteligência artificial [2]. Quando combinado com uma implementação eficiente de aprendizado por reforço, que tradicionalmente exigia dados rotulados manualmente, esse avanço permite tornar o desenvolvimento e treinamento de agentes mais automático e independente da intervenção humana.

O Arcade Learning Environment (ALE) é uma plataforma amplamente utilizada para testar agentes de aprendizado por reforço, tanto no aspecto de percepção quanto na escolha de política [2]. Essa ferramenta inclui uma biblioteca com diversos jogos do Atari 2600 e se destaca pela

praticidade, pois roda dentro do Python e facilita a definição de recompensas, tornando-se um ambiente ideal para o treinamento de agentes.

Neste trabalho, desenvolvemos um modelo de aprendizado por reforço que utiliza Deep Q-Network (DQN) para a seleção de política e redes convolucionais para a percepção do agente. O agente desenvolvido será aplicado ao jogo Space Invaders dentro do ALE, o que possibilita um acompanhamento detalhado do treinamento e da avaliação dos resultados.

Este artigo está estruturado da seguinte forma: a Seção 2 descreve os materiais e métodos utilizados; a Seção 3 apresenta os resultados e discussões; a Seção 4 traz as conclusões; e a Seção 5 inclui as referências bibliográficas.

2. MATERIAL E MÉTODOS

O presente estudo utilizou um computador com GPU compatível com CUDA para acelerar o treinamento. O software empregado incluiu Python 3.11 e diversas bibliotecas especializadas, como PyTorch, Gymnasium para simulação de ambientes, Torchvision, NumPy, Scikit-Image, TorchSummary, ALE-Py para interação com o Arcade Learning Environment. O experimento consistiu no treinamento de um agente de aprendizado por reforço profundo utilizando uma rede neural convolucional baseada no algoritmo Deep Q-Network (DQN). Inicialmente, os frames do ambiente do jogo foram convertidos para escala de cinza e redimensionados para um tamanho padronizado, além da aplicação da técnica de frame stacking para fornecer contexto temporal ao modelo. A rede neural foi definida com uma arquitetura convolucional composta por três camadas convolucionais seguidas por camadas totalmente conectadas, utilizando a função de ativação ReLU. O treinamento do modelo envolveu a utilização do algoritmo DQN para aprendizado das melhores ações a serem tomadas pelo agente, implementando o mecanismo de Experience Replay com um buffer de transições para aumentar a estabilidade do treinamento. Foi empregada uma taxa de exploração epsilon-decay para equilibrar exploração e explotação, bem como o otimizador Adam com uma taxa de aprendizado ajustável. A avaliação do modelo foi conduzida por meio de testes periódicos para verificar a taxa de vitórias e pontuação do agente. Os experimentos foram realizados em ambientes da plataforma Gymnasium, incluindo testes em diferentes jogos da Atari para avaliar a capacidade de generalização do modelo.

2.1 Principais funções

Segue algumas das principais funções aplicadas no modelo:

```python
class DQN(nn.Module):
    def __init__(self, input_shape, num_actions):
        super(DQN, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(input_shape[0], 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()
        )
        self.fc = nn.Sequential(
            nn.Linear(self.feature_size(input_shape), 512),
            nn.ReLU(),
            nn.Linear(512, num_actions)
        )

    def feature_size(self, input_shape):
        return self.conv(torch.zeros(1, *input_shape)).view(1, -1).size(1)

    def forward(self, x):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        return self.fc(x)
```

Fonte: Própria

O código define uma rede neural profunda chamada DQN (Deep Q-Network), usada para aprendizado por reforço em ambientes que utilizam representações visuais, como jogos baseados em imagens.

A classe DQN herda de nn.Module e é composta por duas principais camadas:

1. **Camada convolucional (self.conv)**:
    a. Três camadas convolucionais processam a entrada de imagens, extraindo características relevantes.
    b. Cada camada é seguida por uma ativação ReLU, que introduz não linearidade para melhorar a capacidade de aprendizado da rede.
2. **Camada totalmente conectada (self.fc)**:
    a. Transforma as características extraídas pelas convoluções em representações vetoriais.
    b. Usa uma camada densa com 512 neurônios e ativação ReLU.
    c. A camada final tem um número de neurônios igual ao número de ações possíveis (num_actions), produzindo os valores Q para cada ação.

O método feature_size() calcula automaticamente o tamanho da saída da camada convolucional, garantindo que a transição para a camada densa seja compatível.

O método forward(x) executa a passagem direta dos dados pela rede, primeiro extraindo características com as convoluções e depois classificando as ações possíveis com as camadas totalmente conectadas.

```python
class DQNAgent:
    def __init__(self, state_shape, num_actions, learning_rate=0.00025, gamma=0.99, epsilon=1.0, epsilon_min=0.1, epsilon_d
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = DQN(state_shape, num_actions).to(self.device)
        self.target_model = DQN(state_shape, num_actions).to(self.device)
        self.target_model.load_state_dict(self.model.state_dict())  # Inicializa o target model com os mesmos pesos
        self.optimizer = optim.Adam(self.model.parameters(), lr=learning_rate)
        self.memory = ReplayBuffer(memory_size)
        self.gamma = gamma
        self.epsilon = epsilon
        self.epsilon_min = epsilon_min
        self.epsilon_decay = epsilon_decay
        self.batch_size = batch_size
        self.update_target_freq = update_target_freq
        self.train_step = 0
        self.num_actions = num_actions
        self.loss_fn = nn.MSELoss() # Salva o Loss
        self.episode_q_values = []  # Para a média dos Q-values


    def choose_action(self, state):
        if random.random() < self.epsilon:
            return random.randrange(self.num_actions)
        else:
            state = torch.FloatTensor(state).unsqueeze(0).to(self.device)
            q_values = self.model(state)
            self.episode_q_values.append(q_values.mean().item())  #  Salva o Q-value
            return q_values.argmax().item()

    def learn(self):
        if len(self.memory) < self.batch_size:
            return None, None

        states, actions, rewards, next_states, dones = self.memory.sample(self.batch_size)
        states = states.to(self.device)
        actions = actions.unsqueeze(1).to(self.device)
        rewards = rewards.to(self.device)
        next_states = next_states.to(self.device)
        dones = dones.to(self.device)

        q_values = self.model(states).gather(1, actions)
        next_q_values = self.target_model(next_states).max(1)[0].unsqueeze(1)
        target_q_values = rewards.unsqueeze(1) + self.gamma * next_q_values * (1 - dones.unsqueeze(1))

        loss = self.loss_fn(q_values, target_q_values)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        self.train_step += 1
        if self.train_step % self.update_target_freq == 0:
            self.target_model.load_state_dict(self.model.state_dict())

        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay

        return loss.item(), q_values.mean().item()

    def save(self, filepath):
        torch.save(self.model.state_dict(), filepath)
```

Fonte: Própria

A classe DQNAgent representa um agente de aprendizado por reforço baseado em Deep Q-Networks (DQN). Esse agente é projetado para interagir com um ambiente, aprender políticas ótimas e melhorar sua tomada de decisão ao longo do tempo.

2.2 Principais Componentes

1. **Inicialização (__init__)**
    a. Cria dois modelos DQN: um para aprendizado (self.model) e outro para previsão de valores futuros (self.target_model), atualizado periodicamente.
    b. Utiliza ReplayBuffer para armazenar experiências passadas e amostragem eficiente.

   c. Define hiperparâmetros como taxa de aprendizado (learning_rate), fator de desconto (gamma), taxa de exploração (epsilon e seu decaimento) e frequência de atualização da rede-alvo (update_target_freq).
   d. Usa nn.MSELoss() para calcular a diferença entre os valores Q previstos e os alvos.
2. **Escolha de Ação (choose_action)**
   a. Implementa uma estratégia ε-greedy para equilibrar exploração e exploração:
      i. Com probabilidade epsilon, escolhe uma ação aleatória.
      ii. Caso contrário, seleciona a ação com o maior valor Q previsto pelo modelo.
   b. Salva os valores Q médios para monitoramento do aprendizado.
3. **Aprendizado (learn)**
   a. Obtém uma amostra de experiências do ReplayBuffer.
   b. Calcula os valores Q atuais e os valores Q-alvo usando a rede-alvo.
   c. Minimiza a diferença entre os valores Q preditos e os valores Q-alvo com MSELoss.
   d. Atualiza os pesos da rede com Adam.
   e. Periodicamente copia os pesos da rede principal para a rede-alvo (update_target_freq).
   f. Reduz gradualmente epsilon para diminuir a exploração ao longo do tempo.
4. **Salvamento do Modelo (save)**
   a. Permite salvar os pesos do modelo para uso futuro ou continuação do treinamento.

Esse agente é adequado para problemas de aprendizado por reforço, como jogos e controle de robôs, onde o agente deve aprender estratégias eficientes através de tentativa e erro.

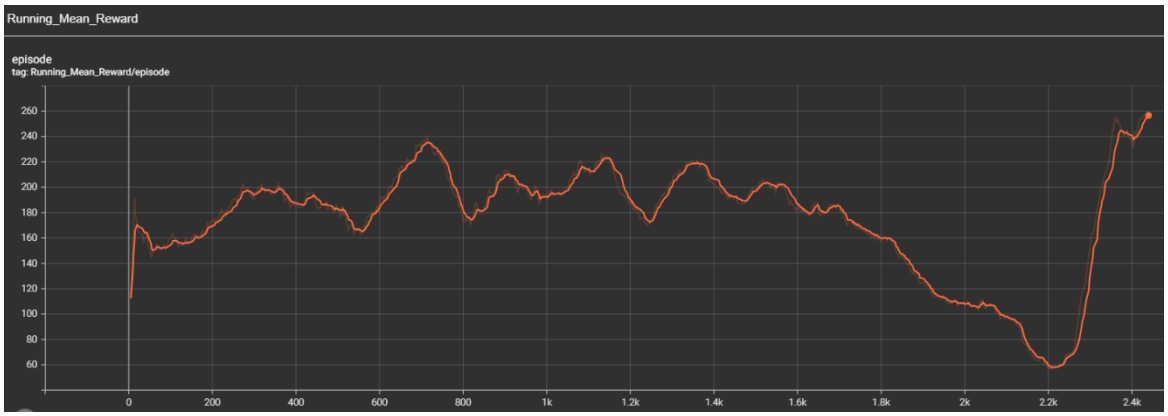
3. RESULTADOS E DISCUSSÃO

Nesta seção, apresentamos os resultados obtidos durante os experimentos realizados com o modelo de aprendizado por reforço baseado em DQN e CNN. Os testes foram conduzidos utilizando o jogo *Space Invaders* no Arcade Learning Environment (ALE). O modelo foi treinado por 2440 interações.

3.1 Visualização gráfica

Após o fim do treinamento, foram gerados alguns gráficos para permitir uma melhor análise dos resultados. Segue alguns exemplos:

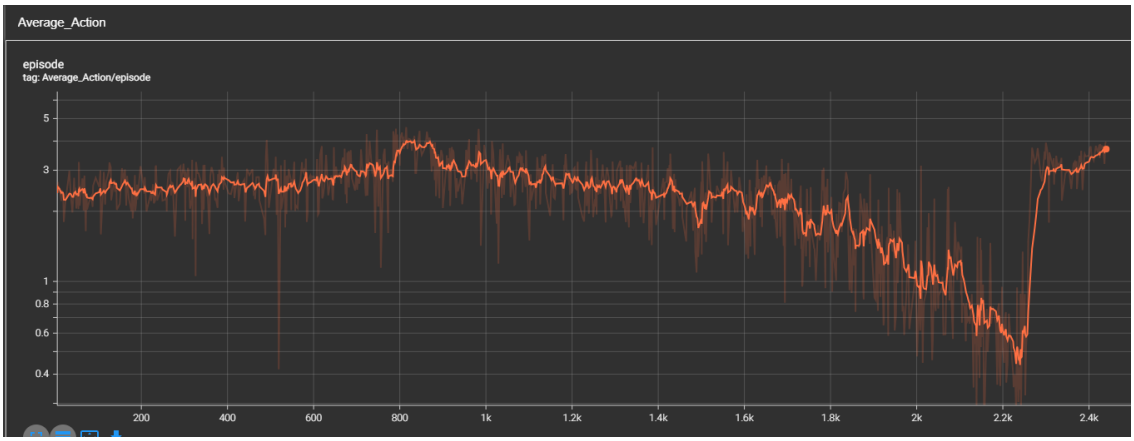
3.1.1 Média de recompensa pelo corredor



Fonte: Própria

Os resultados indicam uma melhora progressiva no desempenho do agente ao longo do treinamento. Inicialmente, a recompensa média por episódio era de aproximadamente **160 pontos**, enquanto no final do treinamento essa média subiu para **260 pontos**. Esse crescimento evidencia o aprendizado eficaz da política ideal.

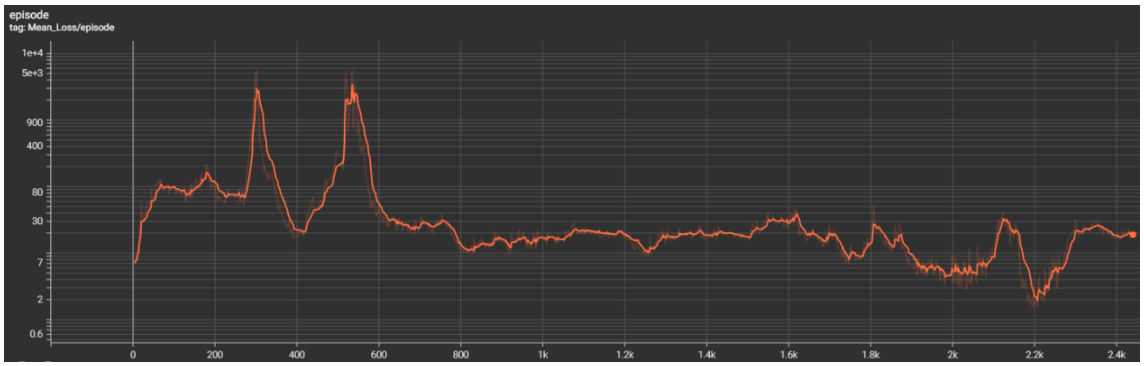
3.1.2 Média de ações por episódio:



Fonte: Própria

Pelo gráfico, é evidente, que com o avanço dos treinamentos, o agente passou a executar mais ações durante a execução da partida. Esse fato isolado não diz muito a respeito da qualidade do agente. Contudo, inicialmente, o agente conseguia em média menos 50 pontos por execução de partidas, no fim do treinamento, passados as 2440 iterações, a média subiu consideravelmente. Isso mostra que além de tomar mais ações, o agente busca maximizar a pontuação obtida durante a partida através das melhores ações.

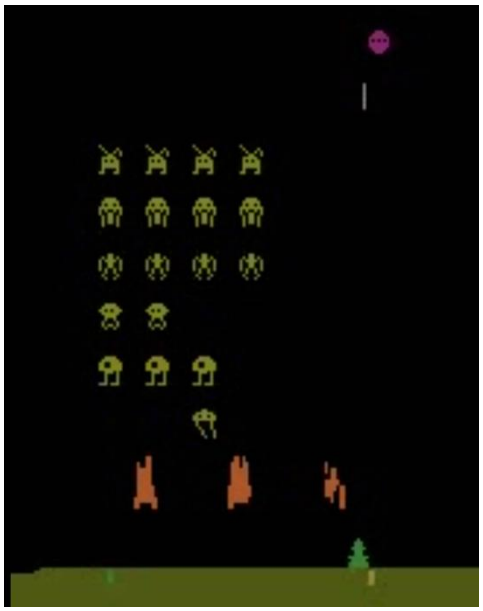
3.1.3 Perda média por episódio



Fonte: Própria

A imagem acima mostra a diminuição do parâmetro loss que mede que o quanto as predições da rede estão erradas em relação aos valores reais. Isso evidência que o agente está aprendendo a como realizar a tarefa da melhor forma a cada etapa do treinamento.

Imagem 1: Modelo 2440 Episódios



Fonte: Própria

Imagem 2: Modelo 2440 Episódios

Fonte Própria

No final do treinamento do modelo, com um total de 2440 episódios executados, é possível notar que o modelo treinado começou a ter uma tendência a ficar no lado direito da tela, buscando sempre atingir a nave inimiga mais distante, por conta da grande quantidade de recompensas que ela proporciona. Isso levou a maioria dos episódios executados com essa rede treinada a seguir esse comportamento, que, mesmo podendo levar à perda de vidas, ainda compensaria pela pontuação recebida.

Nesse contexto, é possível observar que ocorreu o overfitting do modelo, em que ele tendeu a um comportamento previsível e específico, por notar que isso proporcionaria uma maior recompensa, ao invés de aprender padrões gerais do jogo, como focar mais em desviar dos tiros e sobreviver por mais tempo. Em próximos trabalhos, uma abordagem para diminuir a ocorrência dessa situação seria realizar dropout de alguns neurônios, com o intuito de reduzir o aprendizado mais específico da rede e deixá-la mais generalizada, além de aumentar os dados de treinamento.

3.2 Discussão

Por fim, é nítido que a abordagem híbrida com DQN e CNN apresenta bons resultados dentro dos testes apresentados. Contudo, o número baixo de iterações durante o treinamento influencia negativamente no desempenho do agente. Durante a simulação com o modelo mais treinado, é notável que o agente ainda não atingiu o máximo que é possível, ou seja, não está nem próximo do estado da arte desse campo.

4. CONCLUSÃO

Os experimentos conduzidos neste estudo demonstraram que a integração entre Deep Q-Networks (DQN) e Redes Neurais Convolucionais (CNN) proporciona um significativo aumento no desempenho de agentes de aprendizado por reforço em ambientes de alta dimensionalidade visual, como Space Invaders. O agente conseguiu melhorar sua taxa de recompensa ao longo das iterações, evidenciando o impacto positivo da abordagem utilizada.

Apesar dos avanços, ainda existem desafios a serem enfrentados, como a necessidade de um maior tempo de treinamento para atingir um desempenho próximo ao estado da arte e a possibilidade de exploração de arquiteturas mais sofisticadas, como Dueling DQN e Double DQN. Para trabalhos futuros, sugerimos a implementação de técnicas de aprendizado distribuído e o teste do modelo em diferentes ambientes e jogos para avaliar sua capacidade de generalização.

Assim, este estudo contribui para o avanço do aprendizado por reforço profundo, fornecendo insights valiosos sobre a combinação de DQN com CNN e destacando caminhos para futuras melhorias e aplicações.

5. REFERÊNCIAS BIBLIOGRÁFICAS

1. Mnih, Volodymyr. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
2. Guo, Xiaoxiao, et al. "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning." Advances in neural information processing systems 27 (2014).
3. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.