

# Maximização de pontuação no jogo Freeway através de Aprendizado por Reforço

Vinicius Alves Matias<sup>1</sup>

<sup>1</sup>Escola de Artes, Ciências e Humanidades da Universidade de São Paulo  
Rua Arlindo Bettio, 1000, São Paulo/SP  
viniciusmatias@usp.br

## Abstract

Algoritmos de Aprendizado por reforço têm diversas aplicações na literatura, mas um tópico extremamente interessante é a utilização de jogos para avaliação do desempenho dos algoritmos. A utilização de games do Atari em um ambiente controlado podem servir de base para o estudo da generalização de algoritmos em inteligência artificial. Neste relatório será discutida a aplicação de dois algoritmos relativamente recentes (menos de 10 anos da primeira publicação), mas que em muitos casos permitem que um agente aprenda em um ambiente qualquer sem nenhuma grande adaptação ao método original proposto. Os algoritmos de Aprendizado por Reforço analisados foram o Proximal Policy Optimization (PPO) e o Deep Q-Learning Network (DQN). Para uma análise em um ambiente discretizado foram usados os algoritmos Value Iteration (VI) e Policy Iteration (PI). O jogo do Atari escolhido foi o Freeway, tendo este um ponto específico de recompensas esparsas.

## Introdução

Freeway é um jogo desenvolvido pela Activision e disponibilizado no Atari 2600 (Activision 1981). O objetivo do jogo é fazer uma galinha atravessar uma via expressa enquanto desvia de veículos, acumulando um ponto ao atravessar todas as faixas sem ser atingida. A pontuação máxima possível é igual à 34 pontos. Quando a galinha encontra um carro, ela volta duas posições com um delay curto, podendo ser atingida por outros carros nesse período. A interface do game que será usado no projeto pode ser vista na figura 1.

Pela análise da interface nota-se que o jogo permite dois usuários, contudo, para o desenvolvimento e análise dos algoritmos bastará usar apenas um. Importante notar que os carros se movimentam horizontalmente e as galinhas verticalmente. Os algoritmos que serão utilizados buscarão aumentar a pontuação que é exibida na barra superior.

Tendo o problema de maximização da pontuação no jogo em mente, este relatório discutirá a discretização do ambiente e consequente formulação do Processo de Decisão de Markov para o estudo de algoritmos ótimos de planejamento e a utilização de algoritmos complexos de apren-

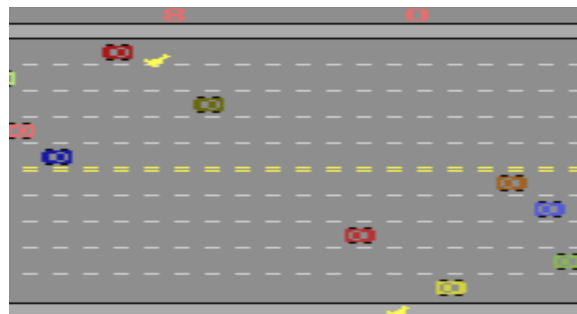


Figure 1: Interface do jogo Freeway

dizado por reforço para aplicação no jogo real. Serão abordadas as técnicas utilizadas, desempenho e peculiaridades do problema encontradas durante o estudo, com ênfase no tópico de otimizar a pontuação em um ambiente real de poucos feedbacks positivos - recompensas esparsas.

## Ambiente

Baseado no jogo Freeway, foi desenvolvida a discretização do ambiente por meio de uma matriz para a utilização dos algoritmos de planejamento, assim como foi possível emular o jogo pelo ambiente da Open AI Gym para utilizar os algoritmos de aprendizado por reforço profundo, ainda que com algumas adaptações de pré-processamento para otimizar os modelos.

## Discretização

Para representar o jogo de uma maneira discreta foi desenvolvida uma matriz de 10 linhas e 10 colunas. A primeira linha contém letras *I* (início) em cada posição, com exceção da posição central, que inicia com o agente representado por *A*. O objetivo é atravessar verticalmente a matriz até a décima linha, chegando à letra *F* que representa o fim da linha, e consequentemente aumentando o score do jogo. Em cada uma das linhas centrais encontra-se em uma das posições um carro representado pela letra *O* (obstáculo). As outras posições da matriz são representadas pelo símbolo *-*, demonstrando que estão vazias.

Como o jogo tem um tempo limite de 136 segundos, o ambiente também terá um tempo delimitado especificado. Para este artigo seguimos que existirão 272 iterações, isto é, para milissegundo o agente poderá tomar uma ação, assim como os carros se moverão para a próxima posição válida na sua linha - da primeira à quarta linha vão para a direita, e da quinta à nona vão para a esquerda. Para o ambiente discreto não foi considerada uma velocidade específica para cada veículo, mas sim uma mesma para todos. A figura 2 exemplifica essa mudança do ambiente para uma próxima iteração.

I	I	I	I	A	I	I	I	I
0	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	0
-	-	-	-	-	-	-	0	-
-	-	0	-	-	-	-	-	-
-	-	-	-	0	-	-	-	-
0	-	-	-	-	-	-	-	-
-	-	-	0	-	-	-	-	-
-	-	-	-	0	-	-	-	-
F	F	F	F	F	F	F	F	F

(a) Ambiente em  $t$

I	I	I	I	A	I	I	I	I
-	0	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	0	-
-	-	-	0	-	-	-	-	-
-	-	-	0	-	-	-	-	-
-	-	-	-	-	-	-	0	-
-	-	0	-	-	-	-	-	-
-	-	-	0	-	-	-	-	-
F	F	F	F	F	F	F	F	F

(b) Ambiente em  $t + 1$

Figure 2: Matriz que representa o ambiente discreto em um momento  $t$  (a) e  $t + 1$  (b)

### Pré-Processamento no Atari

A velocidade para execução de algoritmos que têm relação à redes neurais muitas vezes é dependente da entrada recebida. Para otimizar os algoritmos PPO e DQN foi realizada um redimensionamento nos frames enviados aos jogos. A dimensão de 210px (altura) por 160px (largura) passou para 84X84. Esse redimensionamento influenciou em mais agilidade para realizar os testes, visto que para o DQN (que será visto logo) sem nenhuma adaptação da imagem resultou uma rede neural com mais de 34 milhões de parâmetros possíveis, enquanto a adaptação resultou em cerca de 5 milhões de pesos possíveis. Para este momento do projeto não foram realizadas outras adaptações como na luminância, ajustes para escala de cinza ou junção de dimensões da imagem em um único canal, mencionados no artigo que propôs o DQN.

### Recompensas esparsas

Em planejamento, quando discretizamos o ambiente podemos facilmente determinar recompensas para incentivar o aprendizado de um agente. Isso implica que não teremos apenas recompensas positivas, mas também negativas facilmente detectáveis (chegar ao fim do trajeto é algo positivo e ser atingido por um obstáculo é negativo).

Em ambientes reais, contudo, a definição das recompensas deve estar atrelada ao que se tem disponível do ambiente, não sendo viável realizar processamentos à mais para adicionar outros tipos de recompensas (como uma definição prévia de processamento para identificar veículos e então atualizá-los como recompensas negativas). Além de aumentar a complexidade do método, processamentos adicionais

se afastariam da tentativa de um algoritmo de aprendizado por reforço aprender sem nenhuma interferência externa.

Isso leva à notar que nosso problema terá apenas uma recompensa bem definida à priori, que é chegar ao fim da via expressa (aumentar o score do jogo). Para adquirir essa recompensa, contudo, o agente deve passar por todos os obstáculos e então receber o feedback.

Isso não é tão incomum em jogos do Atari, tanto que os algoritmos PPO e DQN conseguem contornar esses problemas por intermédio da "janela" (buffer) que terão acesso para definir as melhores ações para um problema. O DQN em especial tem algo que salva muitos algoritmos do caso de recompensas esparsas: o fato de ser *off-policy*, logo, ainda que busque a ação que otimize a recompensa em um estado, há uma probabilidade de desvirtuar dessa suposta melhor recompensa e, eventualmente, tornar o processo mais eficiente.

É importante notar que ainda que para o *Freeway* as recompensas esparsas não sejam problemas muito grandes por haver um bom tratamento delas, em alguns problemas pode ser mais interessante buscar algoritmos que buscam explorar muito o ambiente em busca de informações que possam ser úteis posteriormente, isto é, algoritmos que "dirigidos à curiosidade" (Pathak et al. 2017), que possivelmente teriam desempenho decente sob o jogo em questão

## Planejamento Probabilístico

O ambiente discretizado mencionado no item anterior é a base para a aplicação dos algoritmos de planejamento. Antes de iniciar esse estudo, note que a pontuação máxima para essa versão do jogo não é 34 pontos, pois temos 10 posições possíveis para o agente e 272 iterações, assim, a máxima pontuação possível será de  $\lfloor \frac{272}{10} \rfloor = 27$  pontos. Com isto dito, podemos definir o MDP (*Markov Decision Process*).

Um MDP é uma tupla ...

### Value Iteration

### Policy iteration

### Aprendizado por Reforço

### Proximal Policy Optimization

### Deep Q-Learning Network

O algoritmo Deep Q-Learning Network foi proposto por uma equipe da DeepMind (Minth et al. 2013) em um artigo que ficou amplamente conhecido na área pela robustez do algoritmo. Com poucas alterações na estrutura descrita pelos autores o algoritmo pode ser adaptado à diferentes problemas e, muitas vezes, com desempenho sobre-humano quando falamos de jogos. Tal como o nome diz, esse algoritmo parte da definição do *Q-Learning*:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$$

Ainda que com certas semelhanças com o *Value Iteration*, o *Q-Learning* segue abordagens diferentes e entra no escopo de aprendizado por reforço, e não planejamento probabilístico. Na equação,  $Q(s, a)$  é a soma da recompensa de se chegar em um estado  $s$  com a ação  $a$  somada ao próximo  $Q(s', a)$  para uma ação  $a$  que maximiza esse valor, descontada por um valor pré-definido  $\gamma$ . A recursão causada em

$Q(s', a)$  é o que causa a convergência do *Q-Learning*. A semelhança com o *Value Iteration*, portanto, vem da questão da convergência, ou seja, da equação de *Bellman*.

Uma coisa é convergência em ambientes discretos, em que a recorrência consiga ser definida, outra coisa totalmente diferente é a aplicação da recorrência em um ambiente que não temos conhecimento da quantidade de estados à priori, e nem temos ideia do fim do algoritmo. Para aplicar esse conceito em um ambiente real desconhecido que foi proposta a adição de redes neurais com uma maneira de armazenar *Q* valores anteriores nesse problema.

A rede neural servirá para estimar o *Q-value* para um estado e ação (com pesos aleatórios no início, mas se adaptando conforme o treinamento), esse *Q* valor predito será batido com o valor real de se tomar essa ação e, assim, podemos computar a perda na rede neural como o quadrado da diferença entre o predito e a ação tomada. Uma abordagem fantástica que faz toda a diferença nesse processo é que nós armazenamos os últimos *n* valores de *Q* para um um tupla estado-ação-estado predito, e durante a iteração da rede neural haverá uma probabilidade  $1 - \epsilon$  que normalmente é grande no início de tomarmos uma ação aleatória, mas que decresce no decorrer das iterações, e isso implica em escolhermos uma ação que maximize o *Q-value* dentro da rede neural para um par estado-ação muitas vezes, mas com um probabilidade baixa de procurarmos outra ação.

2 anos depois a DeepMind publicou outro artigo trazendo uma abordagem possível para treinamento de jogos no Atari, consistindo da utilização de redes convolucionais para tratar a entrada (e processá-la na rede) e por fim passá-la para uma camada profunda que terá como saída as ações possíveis de serem tomadas no jogo. Para o problema de maximização no *Freeway*, fizemos algumas adaptações à fim de criar um rede com a seguinte configuração:

- Uma entrada de frames 84X84X3 (3 canais);
- Uma camada oculta de 32 filtros 8X8 com stride 4 e função de ativação ReLU (convolucional);
- Uma camada oculta de 64 filtros 4X4 com stride 2 e função de ativação ReLU (convolucional);
- Uma camada oculta de 64 filtros 3X3 com stride 1 e função de ativação ReLU (convolucional);
- Uma camada *Flatten* para intermediar as camadas convolucionais e densas
- Uma camada oculta de 512 perceptrons e função de ativação ReLU (densa)
- Uma camada oculta de 256 perceptrons e função de ativação ReLU (densa)
- Uma camada oculta de 3 perceptrons (quantidade de ações) e função de ativação ReLU (densa)

Para esclarecer, filtro é o "tamanho" da janela em pixels que serão considerados, e *strides* é em quantos pixels a janela será movimentada.

Foram realizados testes para diferentes valores de  $\gamma$ , *learning rate*  $\alpha$ , quantidade de épocas (jogos) mediante

o número de passos, e tamanho do *buffer*. Dois testes interessantes que sintetizam a ideia por trás do algoritmo aplicado no *Freeway* serão mencionados aqui: ambos mantêm o mesmo modelo sequencial mencionado anteriormente,  $\alpha = 0.01$ ,  $\gamma = 1$  inicialmente (descrescendo até  $\gamma = 0.1$ , *buffer* com capacidade para armazenar 10 mil iterações. A diferença entre um teste e outro é dada pela quantidade de passos: o primeiro treinamento foi feito em 100 mil passos (mais de 36 jogos) e o segundo em 1 milhão de passos (mais de 360 jogos). A curva de aprendizado (aumento do *score*) conforme aumenta a quantidade de passos pode ser vista na figura à seguir.

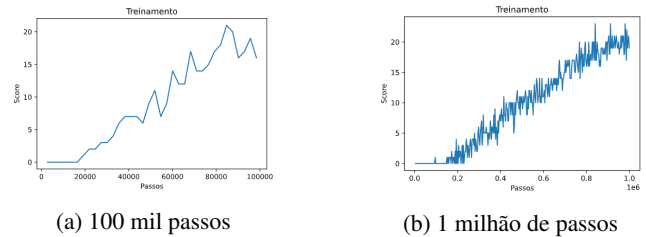


Figure 3: Aumento na pontuação para 100 mil passos (a) e 1 milhão de passos (b)

Vemos a mesma tendência nas duas curvas. Ambas prezam a exploração do ambiente no início devido o  $\gamma$  alto, mas vão aumentando o score após aproximadamente 20% das iterações. Durante esse treinamento o maior score para 100 mil iterações foi de 21 pontos, e 23 pontos para 1 milhão. Há uma variação maior nos resultados do teste do algoritmo para o modelo de menos iterações. O teste foi realizado em 10 épocas, tendo um score médio de 22.5 para 100 mil iterações (com resultados variando de 20 à 25 pontos) e score médio de 21.1 para 1 milhão de iterações (variando de 21 à 22 pontos). O primeiro modelo teve uma variância maior, mas conseguiu uma pontuação maior na média também, o modelo com mais tempo de treinamento teve uma pontuação menor, mas com menos variância entre os resultados. Quando se avalia a visualização dos modelos, algo que o segundo faz com mais frequência é evitar carros, podendo ser uma influência para ter mais segurança nos passos dados após um longo treinamento. Algo que também influencia a diferença é o tamanho do *buffer*, pois ele equivale à 10% do total de iterações para 100 mil dados, e 1% para 1 milhão. Um *buffer* de tamanho 1000 para o primeiro modelo não permitia um aprendizado nas 100 mil iterações, visto que não conseguia alcançar a recompensa em todo o período de treinamento.

## Resultados

## Conclusão

## Referências