Maximização de pontuação no jogo Freeway através de Aprendizado por Reforço

Vinicius Alves Matias[1]
[1]Escola de Artes, Ciências e Humanidades da Universidade de São Paulo
Rua Arlindo Bettio, 1000, São Paulo/SP
viniciusmatias@usp.br

Abstract

A aplicação de algoritmos de Aprendizado por Reforço em diversos problemas do mundo real está sendo cada vez mais possível devido os avanços feitos na literatura da área, tendo essa área também uma particularidade que não é tão vista em outras ramificações de Aprendizado de Máquina: a utilização de jogos para avaliação do desempenho dos algoritmos. Neste artigo será discutida a aplicação de dois algoritmos publicados em há menos de 10 anos, mas que em muitos casos permitem que um agente aprenda em uma ambiente sem nenhuma grande adaptação ao método original proposto. Os algoritmos de Aprendizado por Reforço analisados foram o Proximal Policy Optimation (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 para o Atari 2600 (Activision 1981). O objetivo é fazer uma galinha atravessar uma via expressa enquanto desvia de veículos, acumulando um ponto ao atravessar todas as faixas sem ser atingida. Quando a galinha encontra um carro, ela volta duas posições com um delay curto, podendo ser atingida por outros carros nesse período. A pontuação máxima atingida até o momento neste jogo foi de 34 pontos (Tang et al 2017), por meio do algoritmo *TRPO-hash*.

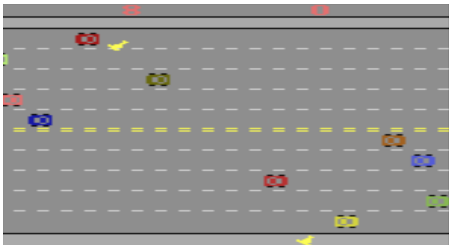


Figure 1: Interface do jogo Freeway

Pela análise da interface da figura 1 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 artigo 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 aprendizado por reforço para aplicação no jogo real. Serão abordadas as técnicas utilizadas, desempenho e pecularidades do problema encontradas durante o estudo, com ênfase no tópico de otimizar a pontuação em uma ambiente real de poucos feedbacks positivos.

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 (Brockamn et al 2016) para utilizar os algoritmos de aprendizado por reforço profundo.

Discretização

Para representar o jogo de uma maneira discreta foi desenvolvida uma matriz de 10 linhas e 9 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), inserido de forma aleatória. As outras posições da matriz são representadas pelo símbolo $-$, demonstrando que estão vazias.

Como o jogo tem um tempo limite de 136 segundos (Weiss 2007), o ambiente também terá um tempo delimitado especificado. Para este artigo segue-se que haverão

272 iterações (uma referência à possibilidade ocorrem duas mudanças no ambiente por segundo), isto é, para cada iteração t o agente poderá tomar uma ação, assim como os carros se moverão para a próxima posição válida na sua linha - à esquerda ou na primeira posição à direita quando chega ao limite da matriz. 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. A transição de um estado para outro atualiza o ambiente, tal como os veículos.

I	I	I	I	A	I	I	I	I
o	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	o
-	-	-	-	-	-	-	o	-
-	-	o	-	-	-	-	-	-
-	-	-	-	o	-	-	-	-
o	-	-	-	-	-	-	-	-
-	-	-	o	-	-	-	-	-
-	-	-	-	o	-	-	-	-
F	F	F	F	F	F	F	F	F

(a) Ambiente em t

I	I	I	I	A	I	I	I	I
-	-	-	-	-	-	-	-	o
-	-	-	-	-	-	-	o	-
-	-	-	-	-	-	o	-	-
-	o	-	-	-	-	-	-	-
-	-	-	o	-	-	-	-	-
-	-	-	-	-	-	-	-	o
-	-	o	-	-	-	-	-	-
-	-	-	o	-	-	-	-	-
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

Pelo ambiente de emulação do Atari (Bellemare et al 2013), a velocidade para execução de algoritmos que têm relação à redes neurais muitas vezes é dependente da entrada recebida. Para otimizar os algoritmo DQN foi realizado um redimensionamento nos frames enviados aos jogos. A dimensão de 210px (altura) por 160px (largura) passou para 84X84. Esse redimensionamento influenciou em uma redução de 34 milhões de parâmetros possíveis na rede neural para 5 milhões.

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.

Em ambientes reais, contudo, a definição das recompensas deve estar atrelada ao que se tem disponível do ambiente, e por vezes essa recompensa não é frequente. Ambientes desse tipo se enquadram no problema de recompensas esparsas (Hare 2019).

Freeway tem apenas uma recompensa bem definida, que é chegar ao fim da via expressa (aumentar o score do jogo em 1). 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 tentam contornar esses problemas por intermédio de um certo controle de aleatoriedade - nem sempre utilizar a melhor ação para um estado, que tipicamente seria a de ficar confortável sem receber nenhuma recompensa

Ainda que para o *Freeway* as recompensas esparsas não sejam muito problemáticas, 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.

Markov Decision Process

O ambiente discretizado é a base para a aplicação dos algoritmos de planejamento, que são úteis para analisar o comportamento do problema em um ambiente controlado. Antes de iniciar esse estudo, note que no ambiente discreto temos 9 posições possíveis para o agente (chegar ao estado meta implica retornar à posição inicial) e 272 iterações, assim, a máxima pontuação possível será de $\lceil \frac{272}{9} \rceil = 31$ pontos. Com isto dito, podemos definir o MDP (*Markov Decision Process*), que podese ser definido como uma tupla $< S, A, D, T, R >$ (Mausam and Kolobov, 2012), onde:

- $s \in S$ são os estados. O agente explora uma sequência de 10 estados (tendo no máximo três visíveis diretamente) e cada um, por sua vez, leva informações coluna adjacente ao agente (de onde vêm os obstáculos). Note que o estado inicial s_0 é fixo e há um conjunto de estados metas G (posição final da matriz) em que o agente chega ao fim da via expressa. Havendo então a possibilidade de um agente transitar em 10 estados (não se mantém no estado meta, mas transita para ele), e que cada estado pode ser ocupado por um obstáculo, agente ou espaço nulo com exceção do estado meta (que só pode ser ocupado pela referência de meta) e o estado inicial (que terá ou um agente ou espaço vazio), temos que existem $3^{10} + 1 + 2 = 59052$ composições de estado diferentes;
- $a \in A$ são as ações, tendo um total de 3 possíveis no conjunto $A = \{UP, DOWN, NOOP\}$;
- $d \in D$ é o tempo (ou época) que ocorre a decisão, proveniente de um conjunto finito (não poderão ser mapeados estados inifinatamente);
- T é uma função de transição que mapeia o estado atual, época e ação em um estado s_{t+1} para então atualizar o ambiente. A transição entre estados segue uma política derivada dos algoritmos, mas vale notar que a transição é determinista - não segue uma distribuição de probabilidade para definir qual estado ir;
- R é uma função de recompensa. Os algoritmos de planejamento entregam uma recompensa positiva igual à 1 para o estado meta, -1 para a colisão com os obstáculos, e nula para qualquer outra posição.

Problemas de Horizonte indeterminado englobam o conceito de estados meta, e de fato à um estado que o agente não sabe que deve chegar sem auxílio de um algoritmo. Esse ponto fica muito claro na necessidade de explorar o ambiente na fase de aprendizado por reforço até encontrar um *score*. Em planejamento, podemos aproveitar o fato de delimitarmos o limite do processo (estado com recompensa positiva), e executar os algoritmos à partir dele como um algoritmo para Horizonte Finito.

Visto que queremos maximar a pontuação, devemos buscar uma política π markoviana (decisão depende apenas do estado inicial em planejamento, apesar de poder se aproveitar de um *buffer* com o histórico, em aprendizado por reforço), estacionária (não há uma relação entre tempo e transição/recompensa) e determinista (sempre será feita a mesma ação para uma configuração de estado).

Algoritmos de Planejamento Probabilístico

Esta seção visa trazer um embasamento teórico dos dois algoritmos ótimos para planejamento que serão utilizados nos experimentos.

Value Iteration

A Iteração de Valor (Sutton and Barto 2020) é um método usado em planejamento quando busca-se identificar qual estado de todo seu ambiente discreto é o que terá a melhor "recompensa" total, ou seja, maior valor. O Valor de cada estado é calculado mediante a aplicação do princípio de otimilidade de *Bellman* para a incerteza de um ambiente (O'Donoghue et al 2018). Independente do estado inicial que se aplica a equação de Bellman, este deverá identificar o valor ótimo para no ambiente. A função valor ótima $V^*(s, n)$ para um estado s em um passo n de um problema de Horizon Finito é dada pela equação 1:

$$V^*(s,n) = \max_{a \in A} \{R(s,a) + \sum_{s' \in S} T(s,a,s')V(s', n-1)\} \tag{1}$$

Essa equação é aplicada no problema de maneira recursiva à partir do estado meta do ambiente discretizado, e vai calculando o valor de todos os estados mediante a recompensa do estado s' e a probabilidade de alcançá-lo com uma das três ações possíveis.

Como a iteração de valor não retorna uma política, é necessária uma adaptação ao método para, ao final, coletarmos a ação que proporcione o maior resultado para as ações possíveis em um estado s (equação 2).

$$\pi(s,n) = arg\max_{a \in A} \{R(s,a) + \sum_{s' \in S} T(s,a,s')V(s', n-1)\} \tag{2}$$

A recorrência permite a convergência do algoritmo à política ótima, mas ainda assim ele depende da definição de um ambiente que possa coletar alguma pontuação, caso contrário, pontuará zero.

Policy Iteration

O algoritmo *Value Iteration* retorna o valor ótimo para os estados de uma ambiente discreto, mas a política ótima para um estado deve ser computada após essa informação se seguirmos o algoritmo original proposto. O algoritmo *Policy Iteration* visa suprir essa falta da obtenção da política ótima do *Value Iteration*. Nesse algoritmo não itera-se a função valor, mas sim uma política inicial arbitrária (Sutton and Barto 2020).

Esse método de iteração consiste de duas partes essenciais: *improvement* (melhoria) e *evaluation* (avaliação). A fase de melhoria parte de uma política π e aplica um algoritmo adaptado da forma de iteração de valor (equação 3).

$$V^{\pi}(s) = R(s, \pi(s)) + \sum_{s' \in S} T(s, \pi(s), s')V^{\pi_i}(s') \tag{3}$$

Essa equação pode ser vista como o valor de um estado s pela política π - inicializada com ações randomicas. Note que não usamos mais a ação de maneira pré-definida, mas uma ação proveniente da política que queremos melhorar.

A etapa de avaliação da política, por sua vez, recebe a função valor da política para calcular uma nova política π' (equação 4).

$$\pi'(s) = arg\max_{a \in A} \{R(s,a) + \sum_{s' \in S} T(s,a,s')V^{\pi}(s')\} \tag{4}$$

Executando essas iterações de política até a convergência, ou seja, $\pi = \pi'$.

Algoritmos de Aprendizado por Reforço

Esta seção aborda também sob viés teórico, a estrutura dos algorimos de aprendizado por reforço profundo utilizados no *Freeway*.

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. Tal como o nome sugere, esse algoritmo parte da definição do *Q-Learning* (Szepesvári 2009).

$$Q(s,a) = R(s,a) + \gamma \max_{a \in A} Q(s', a) \tag{5}$$

Na equação 5, $Q(s, a)$ 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 γ. A recursão causada em $Q(s', a)$ é o que causa a convergência do *Q-Learning*. É perceptível a aplicação da equação de *Bellman* para gerar a convergência.

Pelo fato de a aplicação da recorrência não ser muito viável em problemas complexos e reais que o DQN propôs 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). O Q valor predito será comparado 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 o retorno real. Um diferencial nesse processo é que armazena-se os últimos n valores de Q para um um tupla estado-ação-estado predito, e durante a iteração da rede neural haverá uma probabilidade $1 - \epsilon$ (alta no início) de tomarmos uma ação aleatória, mas que decresce no decorrer das iterações, e isso implica que tendemos à escolhermos uma ação que maximiza o *Q-value* dentro da rede neural para um par estado-ação, mas com um probabilidade baixa de procurarmos outra ação.

Outro artigo publicado pela Deep Mind trazendo uma abordagem possível para treinamento de jogos no Atari (Mnih et al 2015), 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. Baseado neste artigo treinamos nosso agente em uma 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)

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

Proximal Policy Optimization

Problemas de apendizado por reforço profundo atribuem à redes neurais a tarefa de atuar como política em um ambiente, sendo seus neurônios de saída a melhor ação à ser tomada pelo que se tem conhecimento até então. Métodos *Policy Gradient* seguem essa abordagem (Sutton et al 2000). Algoritmos dessa linha, como o *Proximal Policy Optimization* (Schulman et al 2017) seguem uma abordagem mais conservadora para estimar o valor de um estado.

O PPO é um algoritmo *on-policy*, logo, aprende de acordo com a experiência atual do agente, e por isso precisam ter uma função *Loss* à ser aplicada na rede neural que otimize as recompensas à serem obtidas, sendo definida para métodos de gradiente conforme a equação 6.

$$Loss^{PG}(\theta) = \hat{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{6}$$

Sendo ela basicamente é uma estimativa ($\hat{E}$) proveniente da multiplicação entre o logaritmo da probabilidade de se tomar uma ação a em um estado s_t pela política π_θ (a saída da rede neural) por uma estimativa Advantage $\hat{A}$.

Existem várias maneiras de se calcular a estimativa *Advantage*. Para a implementação deste artigo é utilizada a Generalized Advantage Estimation (GAE) descrita por Schulman et al (2018). Essa estimativa leva em conta o valor do estado predito pela rede neural (*Critic*, como será visto na sequência) $V(s_{t+1})$, o valor do estado atual $V(s_t)$, a recompensa real r_t de se tomar a ação predita e chegar no estado previsto, e se esse estado é visível (m_t) no tempo t. Para evitar que se ocupe tempo demais nesse processamento, captura-se apenas as últimas 128 iterações armazenadas.

Dado o conhecimento das variáveis, o GAE pode ser calculado a partir da iteração mais recente em um *loop* até a iteração armazenada mais antiga (de $t = 127$ à $t = 0$) performando as equações 7,8 e 9 na sequência. E Para gerar um escalar $\hat{A}$ foi considerado o desvio padrão de $Retornos_t(s_t, a_t)$, logo, $\hat{A} = \sigma(Retornos_t(s_t, a_t))$

$$\delta = r_t + \gamma V(s_{t+1})m_t - V(s_t) \tag{7}$$

$$gae_t = \delta + \gamma.\lambda.m_t.gae_{t+1} \tag{8}$$

$$Retornos_t(s_t, a_t) = gae_t + V(s_t) \tag{9}$$

A função *Loss* é aplicada em um modelo *Actor - Critic*, que resumidamente, consiste de dois modelos onde um gera uma saída (*Actor*) e ou outro controla a variância dessa saída (*Critic*). Uma maneira para evitar que enviesemos demais a função $Loss$ pela variância da estimativa *Advantage* (algo bem comum quando utiliza-se a abordagem *Actor-Critic*) é utilizando a política anterior para normalizar os resultados, consequentemente limitá-los, como visto naequação 10 (Schulman et al 2015).

$$r_t(\theta) = maximize_\pi \ \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A} \right] \tag{10}$$

Garantindo, portanto, que a política antiga não irá diferir muito da política atual. Finalmente, a função $Loss$ do PPO proposta em 2017 (equação 11) consiste da aplicação destes conceitos, além da adição de um corte entre valores para evitar mais ainda que a atualização de uma política não irá diferir muito de um momento para outro, ainda que influênciadas pelo valor de ϵ (neste artigo é 0.2).

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{11}$$

Experimentos e Resultados

Esta seção discute os experimentos e resultados da aplicação dos quatro algoritmos estudados.

Importante notar que na aplicação dos algoritmos de planejamento alguns comportamentos que poderiam parecer à priori contraditórios ao método, são porém justificáveis pela composição do ambiente. Como mencionado na discretização do ambiente, os obstáculos são postos em posições aleatórias da matriz, e essa composição pode, em muitos casos, gerar um ambiente cujo agente não consiga chegar ao estado meta visto que cada estado inclui informações apenas da coluna adjacente às posições possíveis para deslocamente, justamente para reduzir a complexidade do método.

Optou-se por manter essa composição no ambiente discreto. A Activision, para evitar esse problema, inicializa, em toda a partida os agentes na mesma posição, mas com velocidades diferentes.

Para criar resultados que facilitem a comparação com algoritmos de aprendizado por reforço, cada *frame* (instante t

do ambiente desenvolvido) pode ser analisado como um ambiente discreto, ou seja, uma representação do jogo que pode ter algoritmos de planejamento probabilístico aplicáveis e, portanto, ter uma ação ótima de um estado para outro. Ao fim são esperadas $n\_epochs * 272$ aplicações do método de iteração de valor.

Cada iteração de valor teve em média 300 equações de *Bellman* executadas. A divergência na pontuação entre os jogos pode ser visto de maneira resumida pelo boxplot da figura 3. Por ele é possível notar que uma pontuação maior que 15 é bem mais provável que uma menor.

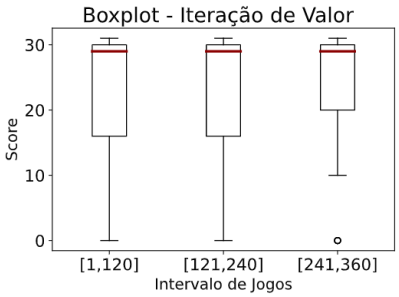


Figure 3: Execuções do Value Iteration. Cada intervalo compreende os scores finais em 120 ambientes de 10 estados.

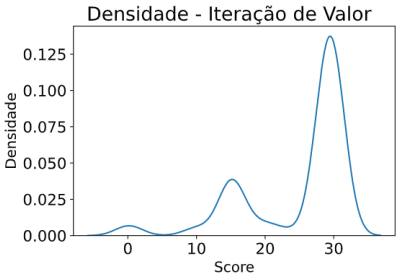


Figure 4: Gráfico de densidade de probabilidade para os scores obtidos em 360 "jogos" executando Value Iteration. Cada jogo é composto por 272 iterações de valor seguidas em um ambiente de 10 estados.

Como explicado neste tópico, o resultado um tanto quanto contra intuitivo de pontuações nulas mostrada nas figuras 3 e 4 tem uma explicação: a aleatoriedade da composição dos ambientes, e a sequência de ações ótima possível não necessariamente resultar em alcançar a meta.

Ainda assim, a maioria das pontuações finais foram maiores que 15 pontos, com a maior densidade nos valores entre 28 e 30 pontos (figura 4), implicando em uma média de 24.89 pontos. Vale notar que para atingir 31 pontos, os estados de maior valor deveriam sempre estar atrelados à ação de ir para frente, pois essa pontuação só é possível se, durante toda a execução do jogo, for possível ir para frente (após um longo período de treinamento, os algoritmos de aprendizado por reforço apresentam essa mesma tática, de ir sempre que possível à frente, mesmo que colidam com obstáculos e voltem duas casas).

O *Policy Iterarion* teve um score médio de 23.83 em 360 jogos, que é próximo ao *Value Iteration*, mas gerado atráves de comportamentos diferentes pelo agente atuando no ambiente. Na implementação do algoritmo, a convergência tipicamente demorou à ocorrer, e por isso delimitou-se um valor máximo de 100 iterações de política, caso não tenha convergido até então. Uma forte influência para o tempo foi a fase de avaliação da política. Nesse método o score final do jogo foi sempre de 0, 1, 30 ou 31 pontos, por isso um boxplot ou outro método de visualização dos dados conforme o tempo não são muito informativos.

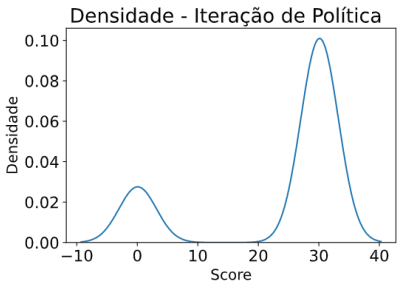
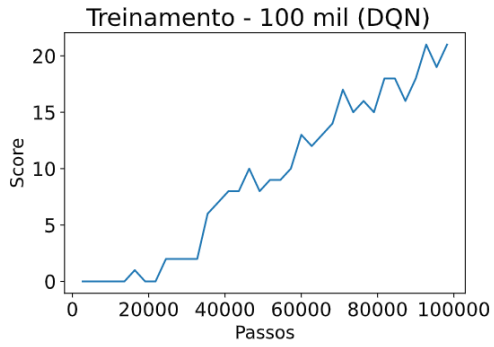


Figure 5: Gráfico de densidade de probabilidade para os scores obtidos em 360 "jogos" executando Policy Iteration. Cada jogo é composto por 272 iterações de valor seguidas em um ambiente de 10 estados.

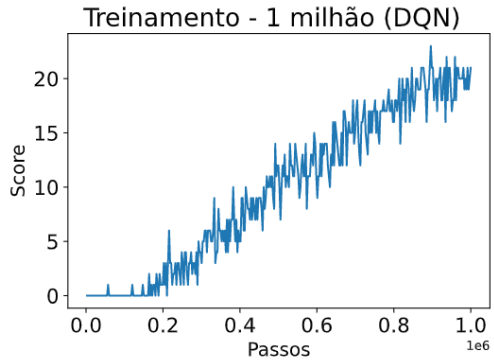
Ainda que a maior parte dos jogos em 30 ou 31 pontos (Figura 5), na média o score cai para acompanhar os outros valores. Isso implica que o método escolhe a melhor ação para o conjunto de estados visível, mas não necessáriamente será a ação que, dada uma nova configuração do ambiente, resulte em uma otimização do resultado.

Em aprendizado por reforço foram relizados testes para diferentes valores de gamma γ, *learning rate* α, quantidade de épocas (jogos) mediante o número de passos, e tamanho do *buffer* para analisar o DQN. Dois testes interessantes que sintetizam a ideia por trás do algoritmo aplicado no *Freeway* serão mencionados aqui: ambos mantém o mesmo modelo sequêncial mencionado anteriormente, $\alpha = 0.01$ e $\gamma = 1$ incialmente (descrescendo até $\gamma = 0.1$. 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 6.

Vemos a mesma tendência nas duas curvas. Ambas prezam a exploração do ambiente no início devido o γ 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

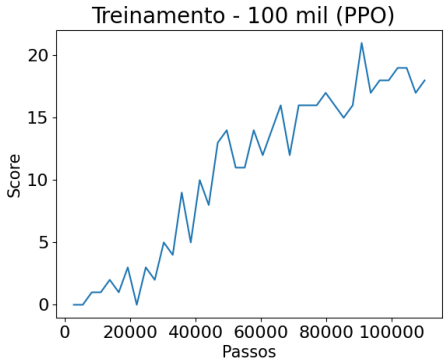


(a) 100 mil passos

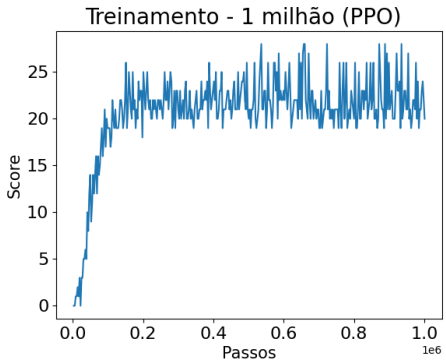


(b) 1 milhão de passos

Figure 6: Aumento na pontuação para 100 mil passos (a) e 1 milhão de passos (b) utilizando o algoritmo DQN



(a) 100 mil passos



(b) 1 milhão de passos

Figure 7: Aumento na pontuação para 100 mil passos (a) e 1 milhão de passos (b) utilizando o algoritmo PPO

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.

Os resultados para o PPO podem ser vistos na figura 7. O modelo *Actor-Critic* utiliza a mesma configuração de *perceptrons* que o DQN, com o acréscimo da função *Loss* descrita anteriormente. A taxa de aprendizado α se manteve como 0.01. Para o cálculo de $\hat{A}$ é utilizado o fator de desconto $\gamma = 0.99$ e lambda $\lambda = 0.95$.

O comportamento para 1 milhão de passos foi um complemento ao de 100 mil, visto que não foi necessária nenhuma adaptação nos parâmetros que enviesasse a simulação (como o tamanho do buffer, que foi primordial no DQN). Pela análise da curva para 1 milhão de passos, percebemos que o algoritmo chegou ao seu limiar - não há uma otimização da pontuação, mas sim a manutenção dela pelas melhores ações possíveis encontradas.

Ainda que o PPO tenha convergido antes que o DQN, o *score* médio atingido não foi muito diferente, ainda que ligeiramente maior. Assim como no DQN, o modelo treinado em 1 milhão de passos foi testado também em 10 configurações de jogos e obteve um score médio de 22.7 pontos, com destaque de que a máxima pontuação adquirida foi de 28 pontos.

Conclusão

O jogo *Freeway*, em aprendizado por reforço, demonstra a necessidade de explorar ambientes para resultados melhores. Algoritmos ótimos usados em planejamento também são suscetíveis à uma pontuação reduzida mediante a configuração do ambiente, apontando que nem sempre a escolha ótima pode ser capaz de otimizar a pontuação, quando simulamos vários ambientes discretos de uma maneira dinâmica para produzir esse resultado. Algo notável tanto em Planejamento quanto Aprendizado por Reforço é que a sequência de ações não necessariamente chegará à pontuação máxima, mesmo que seja ótima para um estado. A configuração do jogo permitiu uma leve melhoria no desempenho do PPO, porém, ambos os algoritmos de Aprendizado por Reforço atingiram um score positivo e por vezes superior à média humana.

Referências

Activision, Inc. 1981. Atari 2600 Instructions Archive. Santa Clara, CA. Activision AG-009-03 Rev. 2.

Bellemare, MG.; Naddaf, Y; Veness, J; and Bowlingm. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. Journal of Artificial Intelligence Research 47: 253-279.

Brockman G.; Cheung, V.; Petterson, L.; Schneider, J; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAi Gym.

O'Donoghue, B; Osband, I.; Munos, R; and Mnih, V. 2018. The Uncertainty Bellman Equation and Exploration.

Hare, J. 2019. Dealing with Sparse Rewards in Reinforcement Learning.

Mausam and Kolobov, A. 2012. Planning with Markov Decision Processes: An AI Perspective. Morgan and Claypool Publishers.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning.

Mnih, V.; Kavukcuoglu, K.; Silver, D. et al. 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Pathak, D; Agrawal, P; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven Exploration by Self-supervised Prediction.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, A. 2015. Trust Region Policy Optimization.

Schulman, J.; Wolski, F.; Dhariwal, P.; and Radford, A. and Klimov, O. 2017. Proximal Policy Optimization Algorithms. CoRR, abs/1707.06347.

Schulman, J; Moritz, P.; Sergey L.; Jordan, M; and Abbeel, P. 2018. High-Dimensional Continuous Control Using Generalized Advantage Estimation.

Szepesvári, C. 2009. Algorithms for Reinforcement Learning (Synthesis Lectures on Artificial Intelligence and Machine Learning). Morgan and Claypool Publishers.

Sutton, R. S.; and Barto, A. G. 2020. Reinforcement Learning: An Introduction. The MIT Press.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. Advances in Neural Information Processing Systems 12, 1057-1063. The MIT Press

Tang, H.; Houthooft, R.; Foote, D.; Stooke, A.; Chen, X; Duan, Y; Schulman, J; Turck, F.; and Abbeel, P. 2017. # Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. Advances in Neural Information Processing Systems 30, 2753-2762. Curran Associates, Inc.

Weiss, B. Classic Home Video Games, 1972-1984: A Complete Reference Guide. 2007. McFarland and Company.