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

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 servidr 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 uma ambiente qualquer 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 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 aprendizado por reforço para aplicação no jogo real. Serão abordadas as técnicas utilizadas, desempenho e pecularidades do problema encontradas durante o estudo, com enfase no tópico de otimizar a pontuação em uma ambiente real de poucos feedbacks positivos - recompensas esparsas.

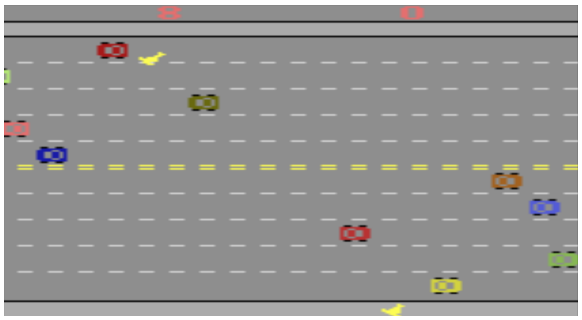


Figure 1: Interface do jogo Freeway

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), 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, 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 - à direita ou na primeira posição à esquerda quando chega ao limite do campo. 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
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)

A maneira que permite que esse ambiente funcione de forma não dinâmica para a aplicação dos algoritmos serão melhor esclarecidos no subtópico de Planejamento Probabilístico *Value Iteration*

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 redimenstionamento 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 propos 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 reallizar 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, que são úteis para analisar o comportamento do problema em um ambiente controlado. Antes de iniciar esse estudo, note que a pontuação máxima para essa versão do jogo não é 34 pontos, pois temos 9 posições possíveis para o agente (chegar ao estado meta implica em retorn à posição inicial) e 272 iterações, assim, a máxima pontuação possível será de $\lfloor \frac{272}{9} \rfloor = 30$ pontos. Com isto dito, podemos definir o MDP (*Markov Decision Process*). Para este MDP consideramos 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 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 ações possíveis. A é representado como um conjunto $A = \{UP, DOWN, NOOP\}$;

- $d \in D$ é o tempo (ou época) que ocorre a decisão. Visto que há um limite de tempo para finalizar o jogo igual à 136 segundos (Weiss 2007), D é um conjunto finito (isto é, não poderão ser mapeados estados inifinatamente);
- T é uma função de transição que mapeia os estado corrente, época e ação em um estado s_{t+1}. A transição entre estados segue uma política (que cabe aos algoritmos definirem), 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 estado meta, -1 para a colisão com os obstáculos, e nula para qualquer outra posição. O estado meta com recompensa +2 foi testado também, mas não trouxe nenhuma grande diferença para várias execuções dos algoritmos (variação de menos de um ponto na pontuação média).

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 de indeterminação fica muito claro na necessidade de explorar o ambiente na fase de aprendizado por reforço, pois não recebe nenhum retorno positivo até encontrar o que poderia ser definido como meta e aumentar seu *score*. Em planejamento, podemos aproveitar o fato de delimitarmos o limite do processo (estado com recompensa positiva), e executar os algoritmos à partir dele.

Visto que queremos maximar a pontuação e a formulação do jogo, 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, como será visto em aprendizado por reforço), estacionária (não há uma relação entre tempo e transição/recompensa) e determinista (sempre será feita a mesma ação para um mesmo estado).

Value Iteration

A Iteração de Valor é um método usado em planejamento quando busca-se identificar qual estado de todo seu ambiente discreto é o que terá a melhor "recompensa", ou seja, maior valor. O Valor de cada estado é calculado mediante a aplicação do prinicípio de otimilidade de *Bellman*. Independente do estado inicial que se aplica a equação de Bellman, este deverá identificar uma política ótima para o estado em questão. A questão da definição da política ótima é deixada em segundo plano no *Value Iteration*, isto porque, como dito, ele visa retornar o valor de um estado, e não a melhor ação à tomar. A função valor ótima $V^*(s, n)$ para um estado s em um passo n de um problema de Horizon Finito é dada por:

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

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, ou seja:

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

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. Esse detalhe é importante pois, 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. O exemplo mais extremo desse caso, mas não restrito somente à ele, é de todos os obstáculos entrarem na mesma coluna, implicando que em algum momento o agente colidirá com um obstáculo, e em uma reação de cadeia levará ele de volta ao início. Como sempre há três ações possíveis para o agente, ele escolherá a melhor para o momento, mas no futuro essa ação ótima poderá sofrer influência da constituição do ambiente e levâ-lo de volta ao início de maneira constante.

Optamos por manter essa composição no ambiente discreto. A Activision, para evitar esse problema, inicializa, em toda a partida, os agentes na mesma posição e com mesma velocidade (portanto, não será um detalhe relevante para a etapa de aprendizado por reforço).

Outro detalhe de ambientes discretos é que eles não devem dinâmicos. Para criar resultados que facilitem a comparação com algoritmos de aprendizado por reforço, utilizamos cada *frame* (instante t do ambiente desenvolvido) 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. A transição de um *frame* para outro implica em mais uma execução do *Value Iteration*. Ao fim são esperadas $n\_epochs * 272$ cálculos do método de iteração de valor. Cada iteração teve em média 300 equações de *Bellman* executadas. Os resutados de 360 jogos podem ser vistos na figura a seguir.

Como explicado neste tópico, o resultado um tanto quanto contra intuitivo de pontuações nulas mostrada na figura 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 de 15 pontos, com a maior densidade nos valores entre 28 e 30 pontos, implicando em uma média de 25.67 pontos. Vale notar que para atingir 30 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).

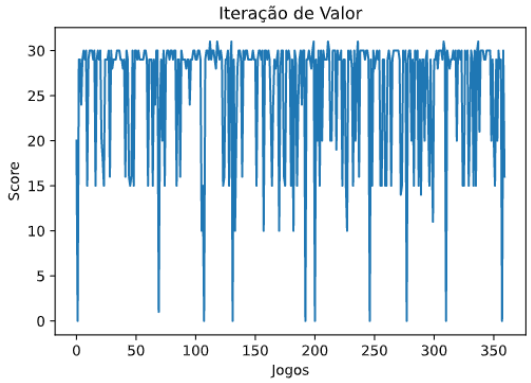


Figure 3: Execuções do Value Iteration. Cada ”jogo” é composto por 272 iterações de valor seguidas em um ambiente de 10 estados.

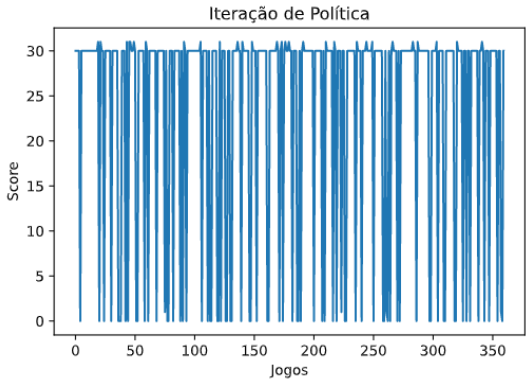


Figure 4: Execuções do Policy Iteration. Cada ”jogo” é composto por 272 iterações de valor seguidas em um ambiente de 10 estados.

Policy Iteration

O algoritmo *Value Iteration* retorna o valor ótimo para os estados de uma ambiente discreto (no caso, cada *frame* do ”jogo”), 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 iteramos mais a função valor, mas sim uma política inicial arbitrária.

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

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

Essa equação pode ser vista como o valor de um estado s tomando uma política π. 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' pela fórmula:

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

Executando essas iterações de política até a convergência, ou seja, $\pi = \pi'$. Na implementação do algoritmo, a convergência tipicamente demorou à ocorrer, e por isso delimitou-se um valor máximo de 100 iterações, caso não tenha convergido até então.

A aplicação do *Policy Iteration* teve melhores resultados comparados à execução única do *Value Iteration* com a aplicação da ação posteriormente, porém, o tempo de execução do algoritmo também foi maior, sendo a fase de avaliação da política crucial para esses dois pontos.

Aprendizado por Reforço

Proximal Policy Optimization

Problemas de apendizado por reforço profundo tipicamente 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

Limitar a atualização da política na rede update -¿ novo/velho advantage -¿ quão bom é o estado clip -¿ menor entre epsilon e atualização da rede multiplas redes neurais minibatch estocástico memória -¿ 20 transições probs por softmax

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*:

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

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 γ. 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.

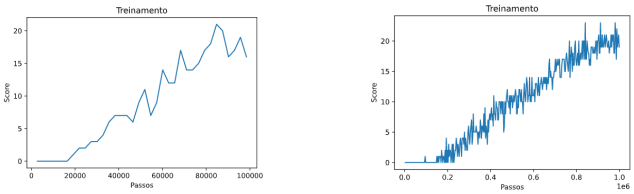
A rede neural servirá para estimar o *Q-value* para um estado e ação (com pesos aleatórios no início, mas se adaptando conforme o treinamento), esse Q valor predito será batido com o valor real de se tomar essa ação e, assim, podemos computar a perda na rede neural como o quadrado da diferença entre 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 relizados testes para diferentes valores de gamma γ, *learning rate* α, 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 mantém o mesmo modelo sequêncial mencionado anteriormente, $\alpha = 0.01$, $\gamma = 1$ incialmente (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.



(a) 100 mil passos (b) 1 milhão de passos

Figure 5: 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 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 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.

Comparações entre os algoritmos

Conclusão

Referências