Instituto Tecnológico de Aeronáutica

CT-213: Inteligência Artificial para Robótica Móvel

Lab 12: *Deep Q-Learning*

Bruno Benjamim Bertucci - Turma 23.2

1 Introdução

Na área de aprendizado por reforço livre de modelo, uma abordagem possível é realizar a estimação da função ação-valor ou função valor com uma tabela contendo o valor de cada ação para cada estado possível. Porém, a aplicabilidade dessa abordagem é limitada, pois requer que os estados e ações sejam discretos e, além disso, ela tem a sua viabilidade comprometida quando o espaço de estados ou de ações é muito grande, ou mesmo, contínuo. Ela também considera que estados próximos podem ser muito diferentes entre si, o que não é o caso de muitas aplicações.

Para solucionar esse problema, é possível realizar uma aproximação da função ação-valor ou da função valor. Existem vários métodos capazes de fazer isso, sendo um deles o uso de redes neurais profundas, o que deu origem ao denominado *Deep Reinforcement Learning*. Um algoritmo desse tipo é o *Deep Q-Learning*, baseado no *Q-Learning* e que utiliza uma rede neural profunda para estimar uma função ação-valor.

Algumas estratégias são utilizadas para estabilizar o treinamento desse algoritmo. Uma delas é o *Experience replay*, que armazena um número fixo das últimas transições de estado e ação em um *replay buffer*, e amostra aleatóriamente um *mini-batch* dessas transições para realizar a atualização da rede neural. Isso garante que os dados usados para treinamento não sejam sequenciais. Em outras palavras, esses dados tornam-se descorrelacionados, o que auxilia na convergência da rede neural.

Outra estratégia empregada é a de *Fixed Q-targets*, que fixa os pesos da rede neural durante a obtenção dos novos valores da função ação-valor, os quais são posteriormente usados para a atualização da rede. Isso garante que os dados sejam estacionários, uma vez que eles próprios são gerados por predições da rede, sendo que isso também auxilia na convergência dela.

2 Implementação

A implementação de *Deep Q-Learning* teve como componente fundamental a rede neural usada para aproximar uma função ação-valor, a qual foi implementada usando a plataforma *Keras*. A arquitetura dela é constituída de três camadas completamente conectadas, sendo que as primeiras duas têm 24 neurônios cada e usam função de ativação *ReLU*, enquanto a última camada usa função de ativação linear. O *input* da rede é dado por um vetor unidimensional que representa o estado do agente, enquanto o *output* corresponde à função-valor aproximada para cada ação possível, a partir do estado fornecido.

Para aplicar essa rede neural, foi criada uma classe denominada *DQNAgent*. Já no construtor, ela recebe parâmetros relevantes para o aprendizado, como os tamanhos dos espaços de estados e ações, um fator de desconto γ, um parâmetro ε para ser utilizado na política ε-greedy que o agente segue, dois parâmetros para regular o decaimento e um valor mínimo para ε, uma taxa de aprendizado para o otimizador da rede, e um tamanho máximo para o *replay buffer*.

Ainda no construtor da classe, o *replay buffer* é criado na forma de um *Double-ended queue (deque)*, e a rede neural é criada e compilada, usando como função de custo o erro quadrático médio e com um otimizador *Adam*, com a taxa de aprendizado mencionada anteriormente. Um sumário da arquitetura da rede também é impresso na tela, o qual encontra-se na Figura 1.

Um episódio de treinamento ocorre da seguinte forma, para cada passo: Uma ação é escolhida seguindo uma política ε-greedy através do método *act* da classe, que recebe o estado atual, e

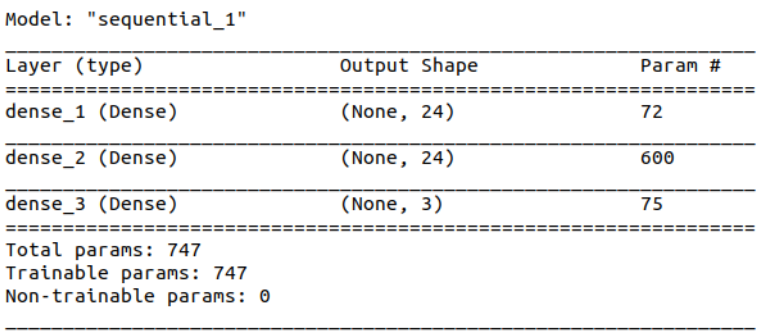

Figura 1: Sumário da arquitetura da rede neural.

tem probabilidade ε de retornar uma ação aleatória, e probabilidade de $1 - \varepsilon$ de retornar uma ação gulosa, ou seja, aquela que tem o maior valor na predição da rede neural para esse estado.

Em seguida, o agente realiza a ação escolhida, e encontra o próximo estado, bem como a recompensa por essa transição, adicionando-a ao *replay buffer* através do método *append_experience*. Uma vez que o *buffer* for grande o suficiente, no caso, o dobro do tamanho do *mini-batch* adotado, o *experience replay* começa a ser realizado através do método *replay*. Este amostra o *mini-batch*, o qual nessa implementação tem tamanho de 32 transições, e itera sobre ele.

Para cada iteração, é feita a predição da rede neural para o estado correspondente. Se na transição atual o episódio chega ao fim, o valor da ação correspondente dado pela predição é alterado para ser igual à recompensa da transição. Caso contrário, ele será equivalente a essa mesma recompensa somado com a multiplicação do fator de desconto pelo maior elemento da predição da rede para o estado final dessa transição, de forma semelhante ao cálculo usado no algoritmo *Q-Learning*. Esses novos valores da função ação-valor aproximada são armazenados, bem como os estados correspondentes a eles. Dado que a rede neural não é atualizada nesse momento, essa implementação segue a estratégia dos *Fixed Q-targets*.

Após a análise de todo o *mini-batch*, a rede neural é finalmente atualizada, tendo como *inputs* os estados armazenados nele e como *output* esperado os valores alterados da função ação-valor para cada um desses estados. O método *replay* retorna, por fim, o valor da função de custo proveniente da atualizaçao.

Ao término de cada episódio, o parâmetro ε é atualizado através do método *update_epsilon*, o qual, quando executado, multiplica ε por um fator de decaimento. Esse decaimento, no entanto, é limitado por um valor mínimo, de tal forma que, ao longo dos episódios, ε decai exponencialmente até atingir esse valor mínimo, e após isso, permanece constante. Isso é feito de tal forma a tornar a escolha de ações gradativamente mais gulosa.

Por fim, a classe *DQNAgent* também incorpora métodos para salvar e carregar os pesos da rede neural encontrados durante o treinamento, para que seja possível reutilizar a rede treinada ou continuar seu treinamento.

3 Aplicação no problema do *Mountain Car*

A implementação de *Deep Q-Learning* foi testada usando a simulação *Mountain Car*, da *OpenAI Gym*. O desafio é otimizar uma política de movimento para que um carro consiga subir um vale e alcançar o ponto mais alto do ambiente bi-dimensional. Para isso, o agente que controla o carro dispõe da posição horizontal do carro e da sua velocidade como espaço de estados, bem como três ações possíveis: empurrar o carro para a esquerda, empurrar para a direita, e não empurrar o carro. As posições possíveis para o carro estão no intervalo [-1,2 , 0,6], e as velocidades, no intervalo [-0,7 , 0,7], sendo o ponto mais baixo do vale situado na posição -0,5.

3.1 Treinamento do agente

No teste, foram realizados 300 episódios de treinamento. A instância do agente criada para ele usou os parâmetros padrão da classe, ou seja, $\gamma = 0,95$, $\varepsilon = 0,5$, $\varepsilon\_\text{min} = 0,01$, $\varepsilon\_\text{decay} = 0,98$, learning_rate $= 0,001$, e buffer_size $= 4098$.

Durante o treinamento, para iteração de cada episódio, o agente escolhe a ação a ser tomada e observa o novo estado e a recompensa pela transição, que, a princípio, é de -1 para cada passo de tempo decorrido. Os passos são executados até que se atinja um limite de 200 passos ou até que o carro alcance o ponto mais alto do ambiente, situado na posição 0,5. A recompensa também é complementada com um valor de 50 caso o carro atinja o ponto 0,5. São inclusos também na recompensa indicativos de que o agente está se aproximando do seu objetivo, para auxiliar o seu aprendizado durante todo o treinamento. Esses indicativos são dados pelo quadrado do deslocamento horizontal com relação à posição inicial do episódio e do quadrado da velocidade do carro.

A execução da lógica do *experience replay* descrita anteriormente é aplicada usando o valor dessa recompensa modificada para cada transição de estados. Além disso, para cada episódio de treinamento, uma recompensa acumulada é calculada, iterativamente multiplicando o valor acumulado anterior pelo fator de desconto γ e somando a recompensa modificada para cada passo.

Realizando o teste dessa forma, foi possível observar que o agente gradativamente aprendeu a mover o carrinho de tal forma a se distanciar mais do vale a cada episódio, e eventualmente conseguiu alcançar o seu objetivo com uma alta velocidade, aprendendo, inclusive, a mover-se inicialmente para trás para ganhar impulso.

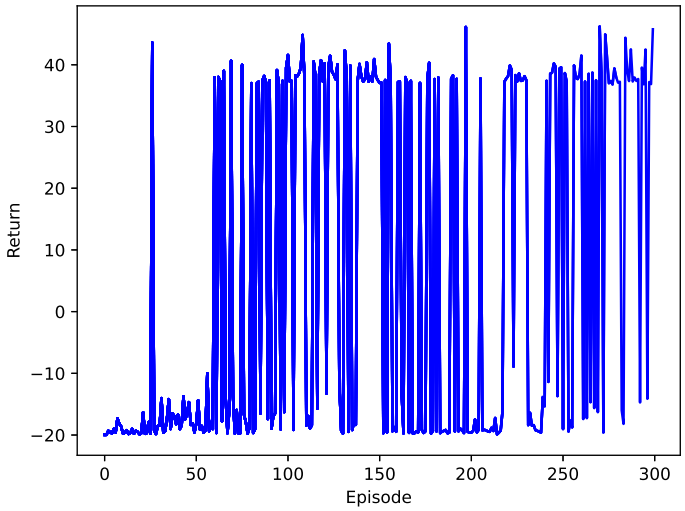

Figura 2: Recompensa acumulada para cada episódio de treinamento.

O gráfico da Figura 2 mostra a evolução da recompensa acumulada de cada episódio. A recompensa por alcançar o objetivo é bem maior, em módulo, que as demais, e como esse alcance provoca o fim do episódio, essa recompensa em particular não sofre decaimento. Portanto, já era esperado que o gráfico tivesse uma disparidade grande entre os episódios nos quais o objetivo foi alcançado, e aqueles nos quais não foi alcançado.

Além disso, é possível perceber que, durante os 50 primeiros episódios, o agente não era capaz de levar o carro ao objetivo consistentemente, tendo feito isso uma única vez. Depois disso, ele passou a ser capaz de obter sucesso com frequência, apesar de que, até o fim do treinamento, ainda eram frequentes os episódios nos quais o agente não teve sucesso.

3.2 Avaliação da política

Para avaliar a política aprendida através dos pesos da rede neural salvos ao término do teste de treinamento, foi executado um segundo teste. Dessa vez, foi feita uma instância do agente semelhante à do teste anterior, porém com $\varepsilon = 0$ e $\varepsilon\_\text{min} = 0$, de tal forma que a política seguida nesse momento de execução era gulosa. Isso se alinha com o algoritmo *Q-Learning*, que aprende seguindo uma política ε-greedy, mas executa uma política gulosa. Por isso, tratam-se de algoritmos *off-policy*.

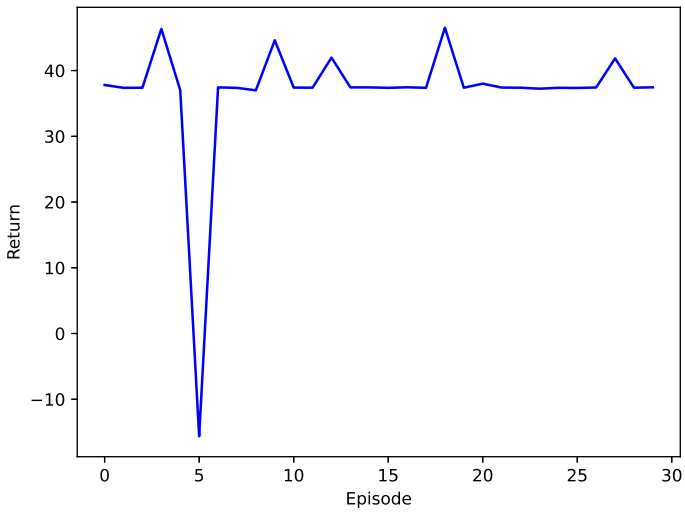

Figura 3: Avaliação da política.

O teste, então, carregou os pesos da rede neural e executou 30 episódios de avaliação da política aprendida. A execução dos episódios é semelhante ao que foi feito no teste anterior, porém sem o passo de aprendizado da rede neural através de *experience replay*. O sucesso do treinamento foi dado pela porcentagem de sucesso do agente no alcance do objetivo, o que pode ser observado pelo retorno acumulado do episódio. A Figura 3 mostra os retornos de cada episódio. O agente não foi capaz de atingir o ponto mais alto do ambiente em apenas um dos 30 episódios, mostrando que o aprendizado foi bastante eficaz para esse problema.

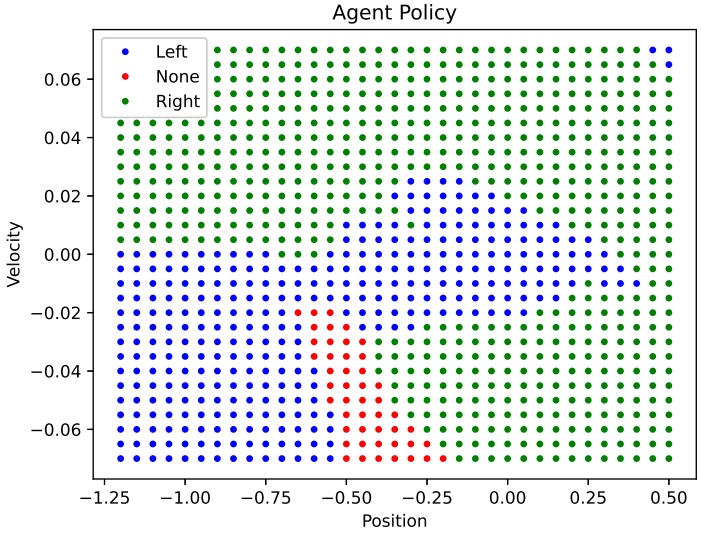

Figura 4: Política aprendida.

Finalmente, o teste gerou um gráfico (Figura 4) ilustrando a política aprendida pela rede neural, realizando predições para pontos discretos uniformemente distribuídos pelo espaço de estados e classificando-os segundo a ação escolhida para cada um. Observando o gráfico, percebeu-se algumas particularidades da política aprendida, como a tendência do agente de empurrar o carro para a esquerda quando ele está com velocidade para a esquerda e com posição menor que -0,5, o que evidencia o seu comportamento de ganhar impulso para conseguir alcançar seu objetivo.