Computação Natural

Deep Reinforcement Learning - Grupo 10

José Parpot Nº: PG41848, Raimundo Barros Nº: PG42814, Tiago Gonçalves Nº: PG42851, and André Soares Nº: A67654

Universidade do Minho, R. da Universidade, 4710-057 Braga, Portugal

Abstract. No presente trabalho encontra-se descrito os métodos e decisões que ocorreram ao longo do projeto que levaram a conclusão deste. Podemos visualizar a construção e treino da rede *Deep Q-Learning* assim como os passos que levaram para melhorar este modelo.

Keywords: Deep Reinforcement Learning · Machine Learning · Sistemas Inteligentes

1 Introdução

Este projeto consiste no desenvolvimento de um algoritmo de *Deep Reinforcement Learning*, para a criação de um agente capaz de jogar o jogo **Breakout**. Neste jogo da consola *Atari 2600*, o jogador controla uma plataforma e tem que destruir o máximo número de camadas de tijolos coloridos alinhadas no topo da tela, atingindo-as com uma bola que vai sendo refletida pela plataforma, paredes laterais e pelo tijolos. Sempre que o jogador falha em atingir a bola impedir que a mesma atinja o fundo da tela, perde uma vida de um total de três.

Recorrendo a um ambiente de desenvolvimento de *Pyhton*, em junção com *Tensorflow* e bibliotecas *OpenAI Gym*, o objetivo do trabalho passa primeiramente pelo treino de uma rede de *Deep Q-Learning* em que o modelo recebe como input o estado currente do jogo e tem como output os *Q-values* previstos pelo modelo para cada ação possível. Depois de treinar o modelo existe a fase de testagem e otimização da rede de *Deep Q-Learning* treinada, com a intenção de obter a melhor performance possível no jogo.

2 *Reinforcement Learning*

A Aprendizagem por Reforço (*Reinforcement Learning*) é uma abordagem computacional para entender e automatizar aprendizagem direcionada e tomada de decisão. Distingue-se de outras abordagens por sua ênfase na aprendizagem de um agente a partir da interação direta com seu ambiente, sem exigir supervisão ou modelos completos do ambiente [1].

O objetivo da Aprendizagem por Reforço é escolher a melhor ação para qualquer estado, o que significa que as ações devem ser classificadas e os valores devem ser atribuídos uma em relação a outra. Como essas ações dependem do estado, o que realmente estamos a medir é o valor dos pares de ação e estado, ou seja, uma ação tomada de um determinado estado, algo que o agente fez em algum lugar.

2.1 Componentes de Aprendizagem por Reforço

A Aprendizagem por Reforço pode ser entendido através de seus componentes: agente, ambiente, estados, ações e recompensas [4].

- **Agentes:** Executam ações. Operam sob controle autónomo, percebem seu ambiente, adaptam-se a mudanças e são capazes de assumir metas;
- **Ação (A):** É o conjunto de todos os movimentos possíveis que o agente pode fazer. Uma ação é quase auto-explicativa, mas deve notar-se que os agentes geralmente escolhem de uma lista de ações possíveis e discretas;
- **O Fator de Desconto:** É multiplicado por recompensas futuras, conforme descoberto pelo agente, a fim de amortecer o efeito dessas recompensas na escolha de ação do agente. Ele foi projetado para fazer com que as recompensas futuras valham menos que as recompensas imediatas. O agente deve escolher as ações que levam à melhor solução global possível, não apenas a melhor solução imediata;
- **Ambiente:** O mundo pelo qual o agente se move e que responde ao agente. O ambiente toma o estado atual e a ação do agente como entrada e retorna como saída a recompensa do agente e seu próximo estado;
- **Estado:** É uma situação concreta e imediata em que o agente se encontra; ou seja, um local e momento específico, uma configuração instantânea que coloca o agente em relação a outras coisas importantes, como ferramentas, obstáculos, inimigos ou prémios;
- **Recompensa (R):** É o feedback pelo qual medimos o sucesso ou o fracasso das ações de um agente em um determinado estado. A partir de qualquer estado, um agente envia a saída na forma de ações para o ambiente, e o ambiente retorna o novo estado do agente (que resultou da ação no estado anterior), bem como recompensas, se houver. As recompensas podem ser imediatas ou atrasadas.
- **Política :** É a estratégia que o agente emprega para determinar a próxima ação com base no estado atual. Ele mapeia estados para ações, as ações que prometem a maior recompensa.

– **Valor (V):** É o retorno esperado a longo prazo com desconto, em oposição à recompensa de curto prazo R. V (s) é definido como o retorno esperado a longo prazo do estado atual sob a política . Descontamos as recompensas ou diminuímos seu valor estimado, quanto mais futuro elas ocorrerem.
– **Valor Q ou Valor da Ação (Q):** É semelhante ao Valor (V), exceto pelo fato de ser necessário um parâmetro extra, a ação atual **a**. Q (s, a) refere-se ao retorno a longo prazo de uma ação que executa uma ação sob política do estado atual **s**. Q mapeia pares de ação e estado para recompensas
– **Trajetória:** Uma sequência de estados e ações que influenciam esses estados.

Diferente de outras formas de *Machine Learning*, como aprendizagem supervisionada e não supervisionada,a aprendizagem por reforço só pode ser pensada sequencialmente em termos de pares de ação de estado que ocorrem um após o outro.

A aprendizagem por reforço julga as ações pelos resultados que elas produzem. É orientado a objetivos, e seu objetivo é aprender sequências de ações que levarão um agente a atingir seu objetivo ou maximizar sua função objetivo.

Nos videojogos, o objetivo é terminar o jogo com mais pontos, para que cada ponto adicional obtido ao longo do jogo afete o comportamento subsequente do agente; ou seja, o agente pode aprender que deve atirar em navios de guerra, tocar em moedas ou desviar de meteoros para maximizar sua pontuação.

Considerando o jogo **Breakout**, neste jogo o utilizador controla uma plataforma na parte inferior da tela e tem que saltar a bola de volta para quebrar todos os tijolos na metade superior da tela. Cada vez que você bate em um tijolo, ele desaparece e sua pontuação aumenta - recebe-se uma recompensa.

Para treinar a rede neuronal, a entrada na rede seria imagens de tela, e a saída seria de três ações: esquerda, direita ou fogo (para iniciar a bola). Teria sentido tratá-lo como um problema de classificação - para cada tela de jogo você deve decidir, se você deve mover para a esquerda, direita ou pressionar o fogo. Porém, é necessário muitos exemplos de treinamento, e para tal se faz necessário gravar sessões de jogos usando jogadores experientes, mas isso não é realmente como aprendemos. Precisamos apenas de *feedback* ocasional que fizemos a coisa certa e podemos descobrir todos os outros nós mesmos.

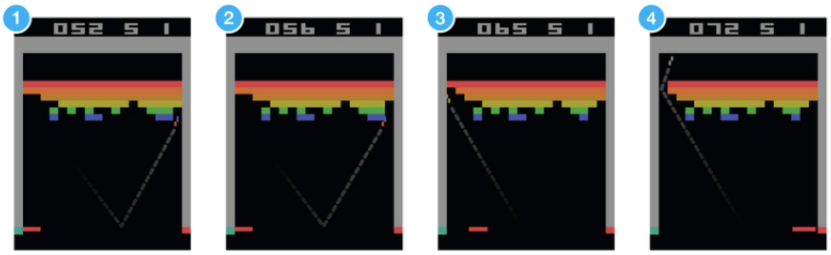


Fig. 1: Representação de um ciclo de *feedback*

Na aprendizagem por reforço, temos rótulos dispersos e atrasados, as recompensas. Com base apenas nas recompensas, o agente deve aprender a comportar-se no ambiente.

Embora a ideia seja bastante intuitiva, na prática há inúmeros desafios. Por exemplo, quando você bate em um tijolo e ganha uma recompensa no jogo Breakout, muitas vezes não tem nada a ver com as ações (movimentos de remo) que você fez antes de obter a recompensa. Todo o trabalho pesado já foi feito, quando se colocou a plataforma corretamente e saltou a bola de volta. Isso é chamado de problema de atribuição de crédito - ou seja, qual das ações anteriores foi responsável por receber a recompensa e até que ponto.

No jogo Breakout ilustrado na Fig. 1, uma estratégia simples é mover-se para a margem esquerda e esperar lá. Quando lançado, a bola tende a voar para a esquerda com mais frequência do que a direita e você conseguirá marcar com facilidade cerca de 10 pontos antes de morrer. Contudo o seu objetivo não é esse, pois deve-se explorar outras estratégias e ações possíveis que sejam melhores. Isso é chamado de dilema *explore-exploit*.

2.2 Modelos de Aprendizagem por Reforço

Existem alguns modelos de aprendizagem importantes e amplamente utilizados em aprendizagem por Reforço.

Processos de Decisão de Markov (MDP) É um framework matemático para o mapeamento do processo de tomada de decisões, ou seja, é utilizado para definir a interação entre um agente de aprendizagem e seu ambiente em termos de estados, ações e recompensas. Todos os problemas de *Reinforcement Learning* podem ser considerados um MDP, pois o MDP é usado para resolver problemas de otimização. A ideia fundamental é encontrar uma política que maximize a recompensa.

Os conceitos de valor e função de valor são essenciais para a maior parte dos métodos de aprendizagem por Reforço. Assumimos a posição de que a função de valor é importante para a pesquisa eficiente no espaço das políticas. O uso de funções de valor distingue os métodos de aprendizagem por Reforço dos métodos evolutivos que buscam diretamente no espaço de políticas orientado por avaliações de políticas inteiras.

Os seguintes parâmetros são usados para obter uma solução:

- Conjunto de ações – A
- Conjunto de estados – S
- Recompensa – R
- Política – n
- Valor – V

Em *Reinforcement Learning*, MDPs são usados para modelar o ambiente. MDPs usam "*timesteps*", e mapeiam o estado corrente a cada timestep. O agente escolhe qualquer ação disponível no estado corrente em um dado *timestep*.

No *loop* de *feedback* (Fig. 2), os subscritos indicam as etapas de tempo t e t + 1, cada uma das quais se refere a estados diferentes: o estado no momento t e o estado no momento t+.

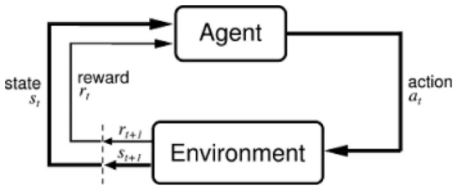


Fig. 2: Representação de um ciclo de *feedback*

Um exemplo de uma função objetiva para a aprendizagem por reforço; ou seja, a maneira como define seu objetivo:

$$\sum_{t=0}^{t=\infty} \gamma^t r(x(t), a(t))$$

Fig. 3: Função Objetivo

Soma-se a função de recompensa "r" sobre "t", que significa etapas de tempo. Portanto, essa função objetivo calcula toda a recompensa que pode-se obter executando, um jogo. O "x" é o estado em um determinado momento, "a" é a ação executada nesse estado e "r" é a função de recompensa para "x" e "a".

Q-Learning É um método baseado em valor de fornecer informações para informar qual ação um agente deve executar. É considerado *off-policy* porque a função *q-learning* aprende com ações que estão fora da política atual, como executar ações aleatórias. Funções de valor são funções de par de ação de estado que estimam quão boa será uma ação específica em um determinado estado ou qual o retorno esperado para essa ação.

O 'q' no *q-learning* significa qualidade. A qualidade, neste caso, representa a utilidade de uma determinada ação para obter alguma recompensa futura.

Como existem várias possibilidades para as ações, o algoritmo deve recompensar aquelas que levam ao destino da forma mais rápida e penalizar aquelas que não levam. O agente então vai experimentando as possibilidades e criando uma tabela com o que traz recompensa e o que não traz. Se a aprendizagem for bem sucedida o agente aprenderá o melhor conjunto de ações que leva ao destino.

3 *Deep Reinforcement Learning*

Embora o *Q-learning* seja um algoritmo muito poderoso, sua principal fraqueza é a falta de generalidade. Se você visualizar o *Q-learning* como números de atualização em uma matriz bidimensional (Espaço de Ação * Espaço de Estado), ele se parecerá com a programação dinâmica. Isso indica que, para os estados que o agente de *Q-learning* não viu antes, não tem ideia de qual ação executar. Em outras palavras, o agente de *Q-learning* não tem a capacidade de estimar valor para estados invisíveis. Para lidar com esse problema, o DQN se livra da matriz bidimensional introduzindo a Rede Neuronal Artificial Profunda (*Deep Learning*).

O processo de *Q-Learning* cria uma matriz (tabela) exata para o agente a qual ele "consulta" para maximizar sua recompensa a longo prazo durante a sua aprendizagem. Embora essa abordagem não seja errada por si só, é prática apenas para ambientes muito pequenos e rapidamente perde a viabilidade quando o número de estados e ações no ambiente aumenta.

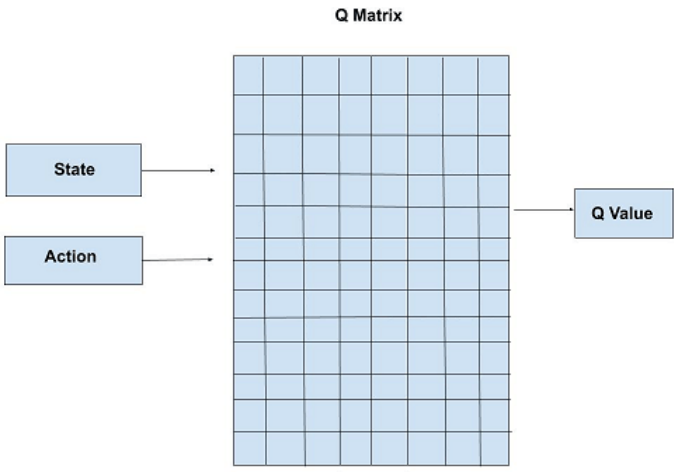


Fig. 4: Q-Learning Matriz (Q Table)

3.1 *Deep Q Network* (DQN)

O DQN utiliza uma rede neuronal para estimar a função de valor Q. A entrada para a rede é a corrente, enquanto a saída é o valor Q correspondente a cada ação.

A solução para o problema acima vem da constatação de que os valores na matriz têm apenas importância relativa, ou seja, os valores têm importância apenas em relação aos outros valores. Assim, esse pensamento nos leva a *Deep Q-Network*, que usa uma rede neuronal profunda para aproximar os valores.

Essa aproximação de valores não prejudica desde que a importância relativa seja preservada. Ou seja, substituímos a *Q Table* no processo *Q-Learning* por um modelo de *Deep Learning* para a aprendizagem dos valores Q. Por isso *Deep Q-Network* também é chamada de *Deep Q-Learning*.

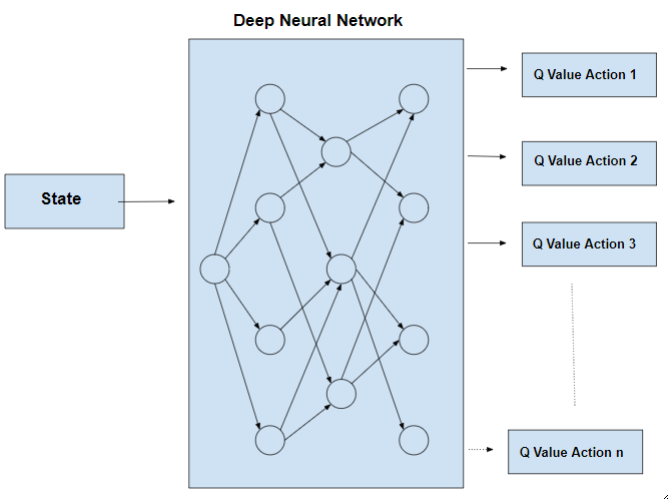


Fig. 5: Substituição do Q Matriz por uma *Deep Learning*

Equação de Bellman Afirma que o valor Q produzido por estar no estado **s** e selecionar a ação **a** é a recompensa imediata recebida, **r (s, a)**, mais o valor Q mais alto possível do estado **s'** (que é o estado em que chegamos depois de executar a ação a dos estados). Receberemos o valor Q mais alto de **s'** escolhendo a ação que maximiza o valor Q. Também apresentamos , geralmente chamado de fator de desconto, que controla a importância das recompensas de longo prazo versus as imediatas.

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

Fig. 6: Equação de Bellman

A equação de Bellman (Fig. 6) possui duas características importantes:

– Enquanto ainda mantemos as suposições dos estados de Markov, a natureza recursiva da Equação de Bellman permite que as recompensas dos estados futuros se propaguem para estados passados longínquos;

– Não é necessário saber realmente quais são os verdadeiros valores Q quando começamos; Desde a sua recursividade, podemos adivinhar algo, e eventualmente convergirá para os valores reais.

Sendo assim, o DQN visa substituir a tabela de valores Q por uma rede neuronal que tente aproximar os valores Q. É geralmente referido como o aproximador ou a função de aproximação e indicado como Q (s, a;), em que representa os pesos treináveis da rede.

Sabendo disto, só faz sentido usar a Equação de Bellman como a função de custo.

$$Cost = \left[Q(s,a;\theta) - \left(r(s,a) + \gamma \max_{a} Q(s',a;\theta) \right) \right]^2$$

Fig. 7: Equação para obert o valor do custo

Por fim, temos é a função erro do quadrado médio (função usada em regressão linear, um dos modelos mais básicos em *Machine Learning*), onde o valor Q atual é a previsão (y) e as recompensas imediatas e futuras são o destino (y').

$$MSE = \frac{1}{n} \sum_{1}^{n} (y_i - y_i')^2$$

Fig. 8: Função do Erro Quadrático Médio

Na Aprendizagem por Reforço, o conjunto de treino é criado à medida que que o agente avança; pedimos ao agente para tentar selecionar a melhor ação usando a rede atual – e registamos o estado, a ação, a recompensa e o próximo estado em que ele terminou. Decidimos o tamanho de um lote e, toda vez que novos registos forem gravados, selecionamos o lote de registos aleatoriamente na memória e treinamos a rede. Os *buffers* de memória usados geralmente são chamados de *Experience Replay*. Existem vários tipos de tais memórias – uma muito comum é um *buffer* de memória cíclico. Isso garante que o agente continue treinando sobre seu novo comportamento, em vez de coisas que podem não ser mais relevantes.

Embora correta, essa arquitetura é muito ineficiente do ponto de vista técnico. Observe que a função de custo requer o valor Q máximo futuro, portanto, precisaremos de várias previsões de rede para um único cálculo de custo. Então, em vez disso, podemos usar a seguinte arquitetura:

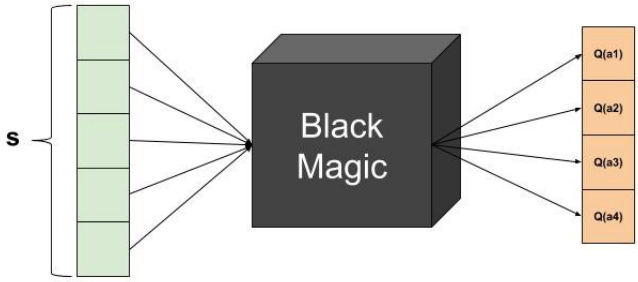


Fig. 9: Arquitetura DQN

Aqui, fornecemos à rede apenas os estados como entrada e recebemos valores Q para todas as ações possíveis de uma só vez.

3.2 Implementação DQN - *Breakout*

As etapas a seguir se referem as partes do código foram extraídas de um repositório GitHub implementado por GiannisMitr [8].

Processar Imagem do Jogo As imagens do Atari brutas são grandes, 210x160x3 por padrão. No entanto, não precisamos desse nível de detalhe para aprendê-los.

Podemos, portanto, economizar muito tempo com o pré-processamento da imagem do jogo, incluindo:

- Redimensionamento para uma forma menor, 64 x 64
- Convertendo para tons de cinza
- Cortando partes irrelevantes da imagem (parte superior e inferior)

Frame buffer Nosso agente só pode processar uma observação por vez, então temos que nos certificar de que contém informações suficientes para encontrar as ações ideais. Por exemplo, o agente tem que reagir a objetos em movimento, então ele deve ser capaz de medir a velocidade do objeto.

Para isso, introduzimos um buffer que armazena as 4 últimas imagens.

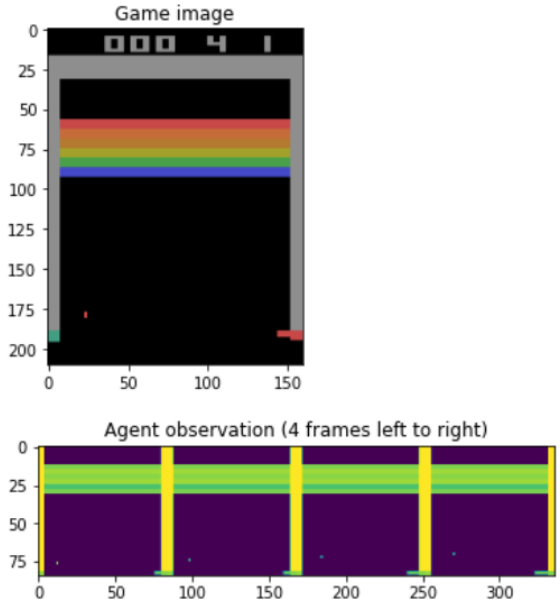


Fig. 10: observação real do Agente após o processamento

Montar a rede Será construída uma rede neuronal para mapear imagens para definir valores-q. Essa rede será chamada em cada etapa do agente utilizando convoluções ampliadas com um pequeno número de recursos para economizar tempo e memória.

Experience Replay A interface é bastante simples, o código utilizado é da própria *OpenAI* e consiste nos seguintes parâmetros:

Para criar um buffer de reprodução devem informar o número máximo de transições para armazenar no buffer. Quando o buffer transborda os registos antigos são abandonados.

- exp_replay.add() - salva (s, a, r, s', done) tupla no buffer;
- exp_replay.sample (batch_size) - deve informar a quantidade de transações para amostrar e retorna os seguintes valores para amostras aleatórias batch_size:
  - obs_batch: Lote de observações;
  - act_batch: lote de ações executadas dado obs_batch;
  - rew_batch: recompensas recebidas como resultado da execução de act_batch;
  - next_obs_batch: próximo conjunto de observações visto após a execução de act_batch;
  - done_mask: 1 se a execução de act_batch resultou no final de um episódio e 0 caso contrário.
- len (exp_replay) - retorna o número de elementos armazenados no buffer de reprodução.

A função play_and_record é utilizada para o agente jogar o jogo por exactamente **n** passos e grave cada cada estado, ação e recompenas no *buffer* de repetição. Sempre que o jogo terminar, os valores são registados e jogo é reiniciado. No final, a função retorna a soma das recompensas ao longo do tempo.

Redes Alvo A rede alvo, é uma cópia dos pesos da rede neuronal a ser usada para valores Q de referência.

A própria rede é uma cópia exata da rede do agente, mas seus parâmetros não são treinados. Em vez disso, eles são movidos aqui da rede real do agente de vez em quando.

$$Q_{reference}(s,\ a) = r + \gamma \cdot \max_{a'} Q_{target}(s',\ a')$$

Fig. 11: Função para a cópia dos pesos para *Q reference*

Q-Learning Aqui, escrevemos uma função semelhante a *agent.update* do *Q-learning* tabular, onde são criados marcadores de posição que serão alimentados com exp_replay.sample (batch_size).

Em seguinda, os valores-q serão utilizados para as ações que o agente acabou de realizar.

$$L = \frac{1}{N}\sum_i [Q_\theta(s,\ a) - Q_{reference}(s,\ a)]^2$$

Fig. 12: Erro TD do Q-Learning

Aprendizagem de diferença temporal (TD) ajustam a função de predição com o objetivo de fazer com que seus valores sempre satisfaçam esta condição. O erro TD (Fig. 12 indica o quanto a função de previsão atual se desvia dessa condição para a entrada atual, e o algoritmo atua para reduzir esse erro.

Conforme definido na função *Q-reference* (Fig. 11) temos os seguintes parâmetros:

- **Q_target(s',a')** denota o valor **q** do próximo estado e próxima ação prevista por target_network
- **s, a, r, s'** são o estado atual, ação, recompensa e próximo estado, respectivamente
- **gamma** é um fator de desconto definido duas células acima.

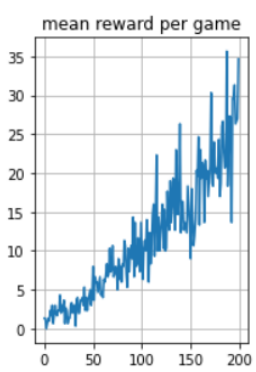
Treino Com as funções criadas e os parâmetros definidos é o momento de treinar o agente.

1. Criar o *buffer* e preenchê-lo. O valor do *buffer* no primeiro ciclo foi 100K;

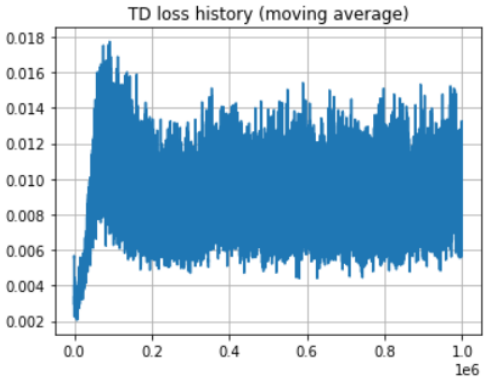
2. Fazer um lote de amostra de observações do *buffer*;
3. Criar vetores vazios para salvar o histórico dos valores médios da recompensa e para salvar o histórico dos valores de perda do TD;
4. Configurar o *epsilon* igual 1 para encorajar o agente a explorar o ambiente;
5. Criar um ciclo onde o processo abaixo irá se repetir 1 milhão de vezes:
   - Inserir o agente configurado no ambiente do **Breakout**, onde este irá jogar 10 partidas e a soma do seus valores médios serão armazenados;
   - Treinar a rede:
     - Formato da imagem igual 64x64;
     - Quatro camadas convolucionais intermediárias;
     - Camada *Flatten* para a transformação da matriz da imagem para um formato de um *array*;
     - Camada *Dense*, onde é informado o número de *frames* e ativação igual a "linear".
     - Perda de TD, cujo valor é o quadrado do valor de Q da ação atual menos o valor Q de referência.
   - Ajustar os parâmetros a cada 500 iterações com o jogo, onde o valor de *epsilon* é reduzido a cada iteração;
   - Salvar os pesos do agente a cada 5000 iterações em um ficheiro "*model_X.h5*", para posteriormente ser utilizado no treino da rede a medida que for diminuindo o *epsilon* inicial do agente;
   - Plotar os resultados para acompanhamento e análise (Fig. 13 e Fig. 14)

Conforme os gráficos gerados para o modelos 1 (*buffer* = 100K, *trange* = 1M e *epsilon* = 1) e modelo 2 (*buffer* = 70K, *trange* = 500k e *epsilon* = 0.5) que foram testados ao longo deste trabalho, podemos inferir que:

- A perda TD (MSE entre os valores Q atuais do agente e os valores Q alvo), oscilam ao longo do tempo, o que é perfeitamente normal, pois o agente está a ser recompensando ou penalizado com base nas suas ações. O modelo 1, nas primeiras 10K iterações oscila de forma ascendente, pois o agente está a explorar o ambiente e está a cometer mais erros, porém após 20K iterações sua perda diminui e mantém oscilando entre uma taxa de 0.005 - 0.015 durante todo período de treinamento. O modelo 2, tem um comportamento diferente, começa em ciclo descendente, depois, após as 20K iterações começa a oscilar de forma ascendente, registando picos de 0.018 de perda.
- A recompensa média, que é a soma esperada das recompensas para o conjunto de estados e ações que o agente recebe durante a sessão do jogo. Pode-se observar que o Modelo 1 e 2 a vai obtendo médias maiores a medida que o agente é treinado, pois, a cada ciclo de 500 iterações o agente vai deduzindo o valor de *epsilon*, fazendo com o agente assuma menos riscos e seja mais preciso no seu movimento.

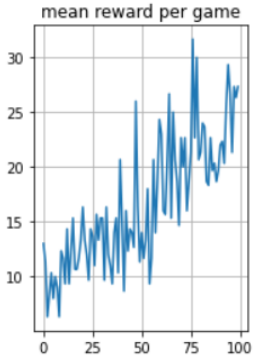


(a) Média de Recompensas por Jogo

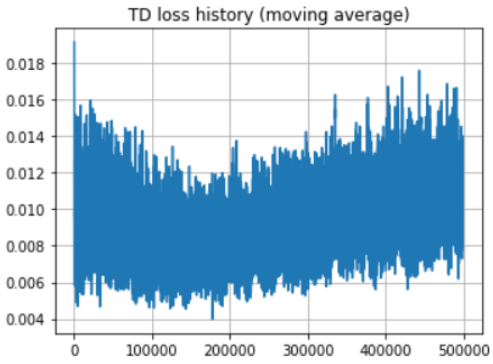


(b) Histórico *TD Loss*

Fig. 13: Resultados obtidos do Modelo 1



(a) Média de Recompensas por Jogo

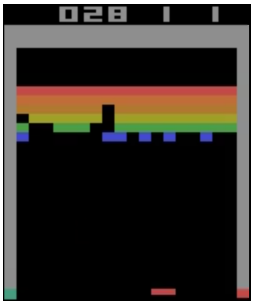


(b) Histórico *TD Loss*

Fig. 14: Resultados obtidos do Modelo 2

O treinamento dos agentes leva tempo. Por conta disso, foram testados e validados apenas 2 (dois) modelos (1 e 2), cujo estes possuem o valor de *epsilon* igual 1 e 0.5 respetivamente, sendo que este último foi gerado a partir do modelo 1 que foi salvo e carregado previamente. Ao gerar um novo modelo, com *epsilon* igual 0.4, observamos que o modelo não convergiu bem.

Na Fig. 15 podemos verificar a pontuação dos respetivos modelos em que o agente jogou 3 partidas.



(a) Modelo 1 (*epsilon*=1)

(b) Modelo 1 (*epsilon*=0.5)

Fig. 15: *Game Score (n_game = 3)*

4 Conclusão

Os algoritmos de *Deep Reinforcement Learning* são bastante competentes ao nível do desenvolvimento de modelos *AI* de grande variedade capazes de executar diferentes tarefas, assim provando a sua grande utilidade no ambiente da Inteligência Artificial. Com este projeto pudémos constatar este facto, ao atingir resultados e performances bastante satisfatórias na tarefa destinada ao modelo desenvolvido.

De notar no entanto o considerável peso computacional que acaba por ser a principal barreira na criação de um melhor modelo e de uma mais profunda otimização. Foi também esta a maior dificuldade que o grupo atravessou no desenvolvimento do projeto, sendo a principal razão da existência de alguns pontos a melhorar, particularmente ao nível da otimização da rede.

Ao longo do projeto chegamos à conclusão que ao diminuir o valor do *epsilon* iterativamente, seria possível atingir melhores resultados. Caso tivéssemos a oportunidade de continuar o trabalho, esta solução seria um dos passos para melhorar o resultado.

References

1. Deep Learning Book, https://www.deeplearningbook.com.br/algoritmo-de-agente-baseado-em-ia-com-reinforcement-learning-parte-1/. Último acesso 04 Jun 2021.
2. Deep Learning Book, https://www.deeplearningbook.com.br/componentes-do-aprendizado-por-reforco-reinforcement-learning/. Último acesso 04 Jun 2021.
3. Temporal-Difference (TD) Learning, https://towardsdatascience.com/temporal-difference-learning-47b4a7205ca8. Último acesso 05 Jun 2021.
4. Deep Learning Book, https://www.deeplearningbook.com.br/algoritmo-de-agente-baseado-em-ia-com-reinforcement-learning-parte-2/. Último acesso 05 Jun 2021.
5. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. DeepMind Technologies, (2013)
6. OpenAI, https://github.com/openai/gym. Último acesso 06 Jun 2021.
7. DeepMind, https://deepmind.com/blog/article/deep-reinforcement-learning. Último acesso 06 Jun 2021.
8. DQN-Atari-Breakout, https://github.com/GiannisMitr/DQN-Atari-Breakout. Último acesso 06 Jun 2021.