1 2  9 0 UNIVERSIDADE D COIMBRA

Aprendizegem Profunda Aplicada

Assignment 3

Autor:
Gonçalo Bastos
Leonardo Cordeiro

Numero de Estudante:
2020238997
2020228071

Dezembro 24, 2024

1 Introdução

Neste trabalho, pretendemos explorar a aplicação de técnicas de Aprendizagem por Reforço Profundo (Deep Reinforcement Learning - DeepRL) para controlar agentes em ambientes de simulação, com o foco principal no uso de representações de estado baseadas em imagens para treinar e otimizar o desempenho dos agentes em cenários desafiantes.

A tarefa inclui a implementação de diferentes variantes do algoritmo Deep Q-Learning, nomeadamente DQN, Double DQN e Dueling DQN. Estes métodos serão aplicados ao ambiente *CarRacing-v2*, fornecido pelo OpenAI Gym, e a um ambiente de simulação personalizado. No ambiente *CarRacing-v2*, o agente será desafiado a conduzir um carro numa pista gerada aleatoriamente, enquanto que, no ambiente personalizado, o objetivo será controlar um robô móvel que deve evitar obstáculos e alcançar um objetivo específico.

2 Parte I - CarRacing-v2

Começámos por analisar diferentes hiperparâmetros da rede e do modelo de *reinforcement learning*, utilizando inicialmente a estratégia DQN como ponto de partida. Foram realizados testes variando parâmetros como o número de iterações por acção (*ControlSteps*) e o número de frames consecutivos para representar o estado (*FrameStack*), que foram ajustados para 6 e 4, respectivamente, para melhorar a percepção do agente sobre o ambiente e estabilizar a execução das acções. Além disso, optimizámos o factor de desconto (*gamma*) para 0.99, incentivando o agente a priorizar recompensas a longo prazo, e ajustámos a taxa de decaimento da exploração (*exploration decay*) para 0.995, permitindo uma transição mais lenta da exploração para a exploração-exploração.

Com base nos resultados obtidos no DQN, implementamos os métodos de *reinforcement learning* avançados, nomeadamente o *Double DQN*, o *Dueling DQN* e o *Double Dueling DQN*. Estes métodos foram implementados e ajustados para aproveitar as vantagens de cada abordagem, como a redução de sobrestimativas (*overestimation bias*) no *Double DQN* e a separação entre valor e vantagem no *Dueling DQN*. Durante este processo, foram realizados testes em diferentes pistas geradas aleatoriamente, tanto estáticas como dinâmicas (activando a flag `UseRandomize`), para avaliar a robustez e a capacidade de generalização dos modelos.

Para avaliar as arquiteturas desenvolvidas e a performance de cada modelo foram utilizadas as seguintes métricas:

- Média dos 100 últimos episódios no final do treino;
- A pontuação obtida pelo modelo em 10 episódios, cada um com uma pista de formato aleatório. 5 apresentam as cores normais (estrada cinzenta com o ambiente verde, conforme utilizado no treino), os outros 5 utilizam cores aleatórias tanto para a pista como para o ambiente.

2.1 Deep Q-Learning

2.1.1 Arquitetura

Primeiramente, estabelecemos a arquitetura do nosso modelo, que foi encarregada de processar o estado do ambiente e decidir qual a ação óptima a tomar, com o intuito de maximizar a recompensa. A implementação foi desenvolvida de forma modular, permitindo alternar entre uma arquitetura standard (DQN/Double DQN) e uma versão avançada (*Dueling DQN* ou *Double Dueling DQN*). Abaixo descrevemos as duas abordagens:

A arquitetura base foi composta pelas seguintes componentes:

- **Camadas Convolucionais ('features'):** A rede começa com três camadas convolucionais para extrair características relevantes do estado do ambiente. As configurações foram:
  - Primeira camada: 32 filtros, *kernel*=5x5, *stride*=2.
  - Segunda camada: 64 filtros, *kernel*=3x3, *stride*=2.
  - Pooling: Max Pooling (*kernel*=2, *stride*=2).

  Cada camada é seguida por uma função de ativação GELU para adicionar não linearidade.
- **Camadas Lineares ('layers'):** A saída das camadas convolucionais é (*flattened*) e passada para:
  - Primeira camada linear: 7744 unidades para 256.
  - Segunda camada linear: 256 unidades para o número de ações (saídas).

Esta arquitetura retorna diretamente os valores Q para cada ação, que são usados para selecionar a ação com maior valor (*argmax*) e pode ser vista na figura 1.

Na arquitetura avançada *Dueling DQN* e *Double Dueling DQN* na Figura 10, a rede foi modificada para incluir a separação entre o cálculo do valor do estado (*value*) e a vantagem relativa das ações (*advantage*). Esta abordagem utiliza:

- **Camadas Convolucionais ('features'):** Iguais à configuração utilizada no DQN/Double DQN.
- **Camadas Lineares para Separação ('layersdueling'):** Após as camadas convolucionais, a saída é achatada e dividida em dois ramos:
  - **Valor do Estado (*value*):**
    - Primeira camada linear: 7744 para 256.
    - Segunda camada linear: 256 para 1.
  - **Vantagem das Ações (*advantage*):**
    - Primeira camada linear: 7744 para 256.
    - Segunda camada linear: 256 para o número de ações (saídas).
- **Combinação Final:** Os valores Q são calculados como:

$$Q(s,a) = V(s) + (A(s,a) - \text{mean}(A(s,a)))$$

O termo $\text{mean}(A(s,a))$ garante que a vantagem seja normalizada.

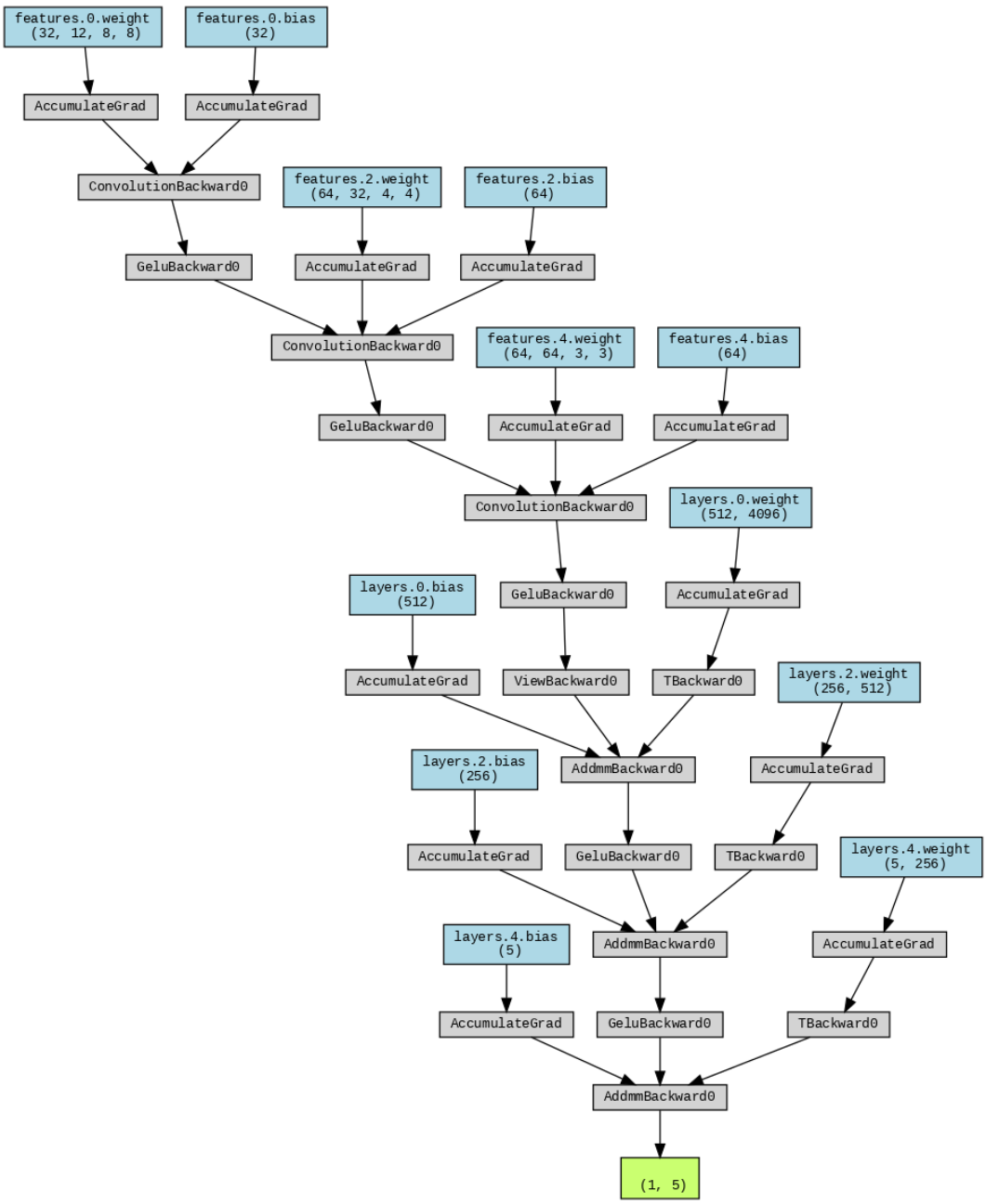

Figure 1: Arquitetura da rede DQN (sem *dueling*). A rede utiliza camadas convolucionais e lineares para calcular os valores Q diretamente.

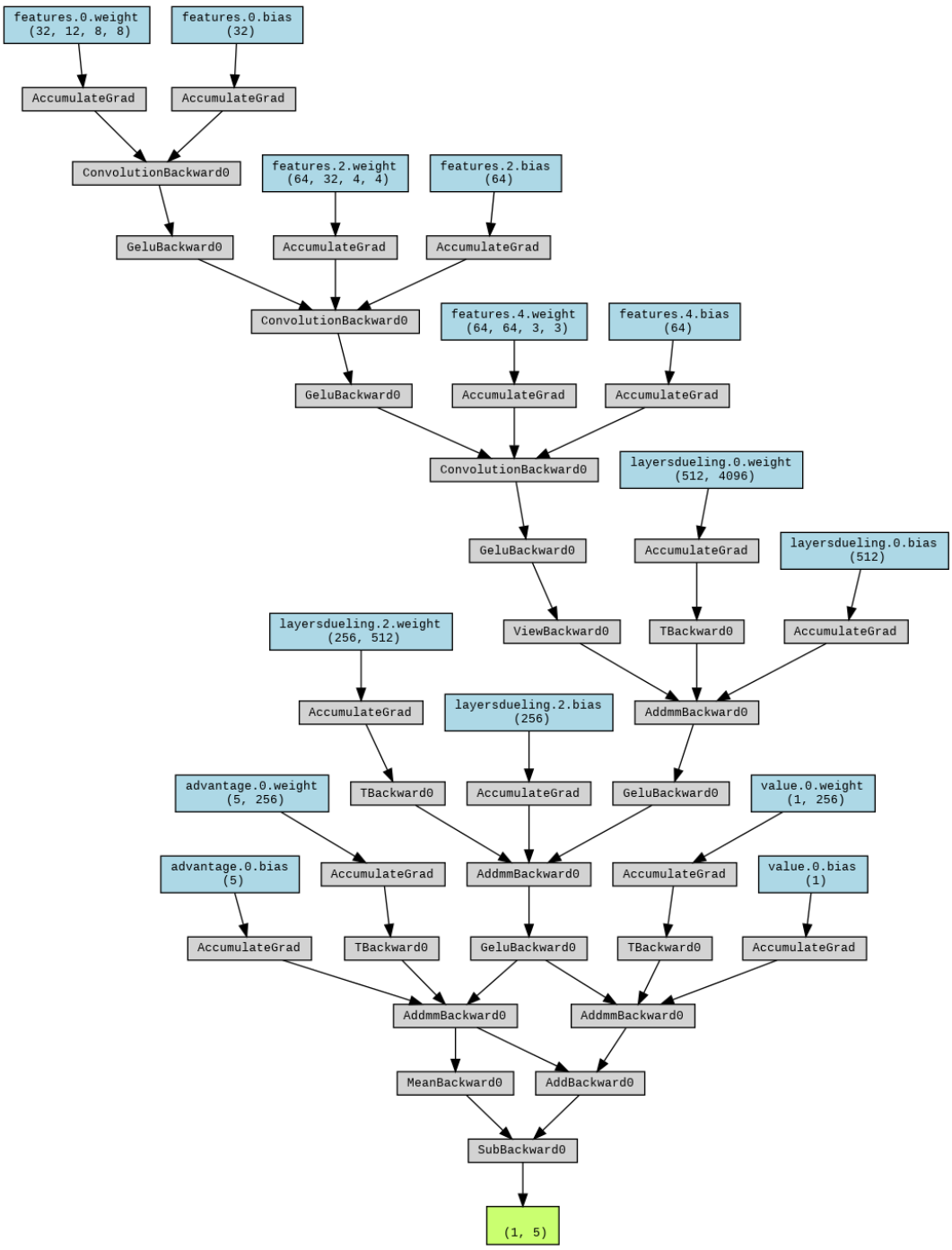

Figure 2: Arquitetura da rede DQN com *dueling*. A rede separa o cálculo do valor do estado (*value*) e da vantagem das ações (*advantage*), que são combinados para calcular os valores Q.

Os hiperparametros de treino já mencionados encontram-se aqui para efeitos de consulta 3

```
#hyper-parameters
TotalEpisodes=500;
MaxSteps=250;
ControlSteps=6; # number of iterations the same action is executed in the environment
FrameStack=4 # number of consecutive frames used to represent the state
FreezeCounter=5; # clone the model every X episodes
BatchSize=64;
exploration_threshold=1
exploration_threshold_min=0.01
exploration_decay=0.995
discount_factor=0.99
LearningRate=0.0001

SaveAtCounter=25 # save model or video at every X episodes
ResetCounter=25 # maximum number of experiences with bad performance (max=InitCounter+ResetCounter)
InitCounter=25 # minimum number of experiences per episode
FrameSize=3; # 3 if RGB 1 if grayscale
```

Figure 3: Hiperparametros usados

2.1.2 Resultados usando Controlo Contínuo

Os resultados obtidos com a implementação do modelo DQN em modo de controlo contínuo estão apresentados na Tabela 1 e na Figura 4. Neste modo, o agente tem acesso a um espaço de ações mais rico e detalhado, permitindo maior liberdade para ajustar o controlo em tempo real.

Pista de Teste	Recompensa Obtida
Normal 1	834.10
Normal 2	867.01
Normal 3	844.82
Normal 4	833.87
Normal 5	827.64
Aleatória 1	-74.39
Aleatória 2	-66.87
Aleatória 3	-23.03
Aleatória 4	-53.68
Aleatória 5	-68.59

Table 1: Recompensas obtidas pelo modelo DQN em diferentes tipos de pistas no modo contínuo.

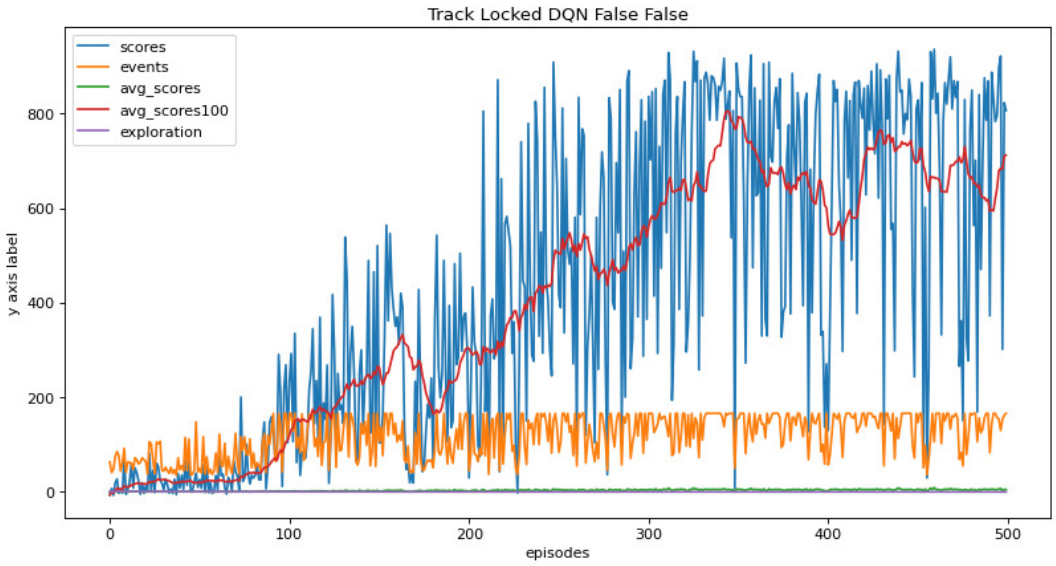

Figure 4: Desempenho do modelo DQN em controlo contínuo ao longo de 500 episódios. O gráfico mostra as recompensas obtidas, a média móvel de 100 episódios e a taxa de exploração (*epsilon*).

2.1.3 Resultados usando Controlo Discreto

Os resultados obtidos com a implementação do modelo DQN em modo de controlo discreto estão apresentados na Tabela 2 e na Figura 5. Neste modo, o agente utiliza cinco ações disponíveis (‘nothing‘, ‘left‘, ‘right‘, ‘gas‘, ‘brake‘) para navegar pela pista.

Pista de Teste	Recompensa Obtida
Normal 1	640.41
Normal 2	445.25
Normal 3	319.65
Normal 4	853.03
Normal 5	868.22
Aleatória 1	422.62
Aleatória 2	-86.57
Aleatória 3	-93.01
Aleatória 4	-29.49
Aleatória 5	-72.25

Table 2: Recompensas obtidas pelo modelo DQN em diferentes tipos de pistas no modo discreto.

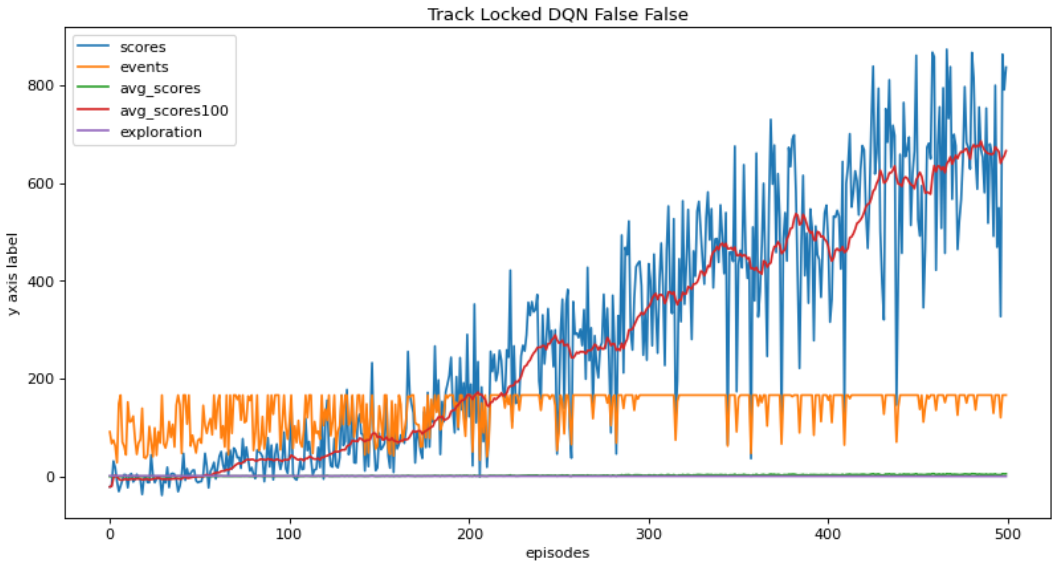

Figure 5: Desempenho do modelo DQN em controlo discreto ao longo de 500 episódios. O gráfico mostra as recompensas obtidas, a média móvel de 100 episódios e a taxa de exploração (*epsilon*).

2.2 Double DQN

Uma das limitações do algoritmo DQN é a sua tendência para sobrestimar os valores Q associados às ações, o que pode levar a decisões subótimas, especialmente durante as fases iniciais de treino, onde as estimativas são menos precisas. Estas sobrestimativas podem prejudicar o desempenho geral do modelo, desvalorizando experiências relevantes e priorizando ações erradas.

O algoritmo *Double DQN* foi desenvolvido para mitigar este problema, utilizando duas redes neuronais. Uma rede principal é responsável por selecionar as ações, enquanto uma segunda rede, denominada rede alvo, avalia os valores Q das ações selecionadas. Esta separação permite corrigir o viés de sobrestimação, resultando em decisões mais estáveis e precisas ao longo do treino.

2.2.1 Resultados usando Controlo Contínuo

Os resultados obtidos no modo de controlo contínuo utilizando Double DQN estão apresentados na Tabela 3 e nas Figuras 6 e **??**. Neste modo, o espaço de ações mais detalhado permitiu ao modelo obter melhores resultados em pistas normais, mas dificuldades em pistas aleatórias.

Pista de Teste	Recompensa Obtida
Normal 1	408.16
Normal 2	848.25
Normal 3	264.58
Normal 4	718.24
Normal 5	605.25
Aleatória 1	129.80
Aleatória 2	-43.72
Aleatória 3	-53.68
Aleatória 4	-38.37

Table 3: Recompensas obtidas pelo modelo Double DQN em diferentes tipos de pistas no modo contínuo.

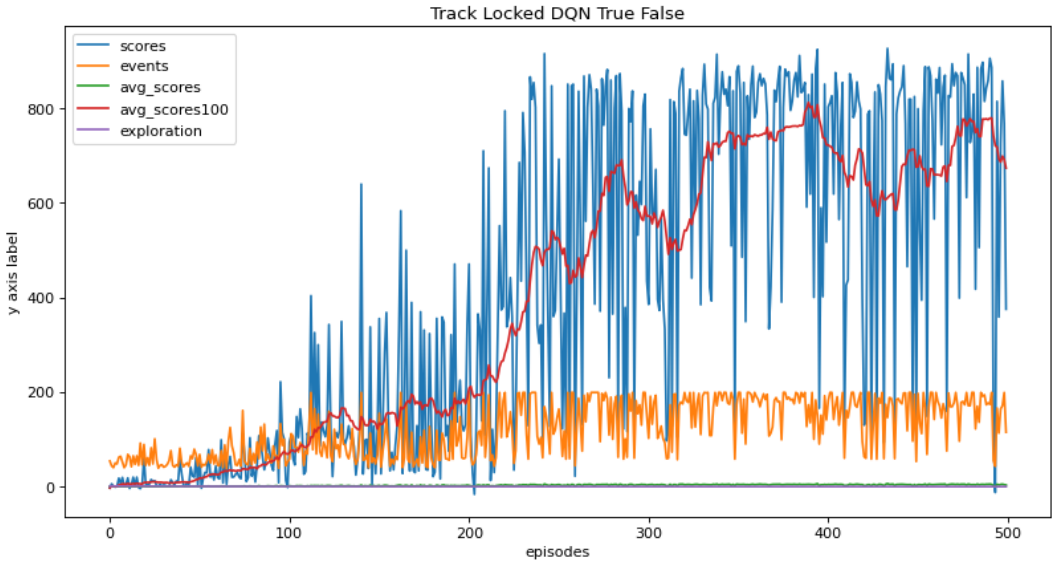

Figure 6: Desempenho do modelo Double DQN em controlo contínuo (primeira configuração).

2.2.2 Resultados usando Controlo Discreto

Os resultados obtidos no modo de controlo discreto utilizando Double DQN estão apresentados na Tabela 4 e nas Figuras 7. Neste modo, o agente teve acesso a um espaço de ações discretas mais limitado, mas foi capaz de obter melhores resultados em algumas pistas normais.

Pista de Teste	Recompensa Obtida
Normal 1	351.33
Normal 2	198.66
Normal 3	695.99
Normal 4	212.31
Normal 5	772.41
Aleatória 1	-28.81
Aleatória 2	-32.14
Aleatória 3	-61.30
Aleatória 4	-77.69
Aleatória 5	13.47

Table 4: Recompensas obtidas pelo modelo Double DQN em diferentes tipos de pistas no modo discreto.

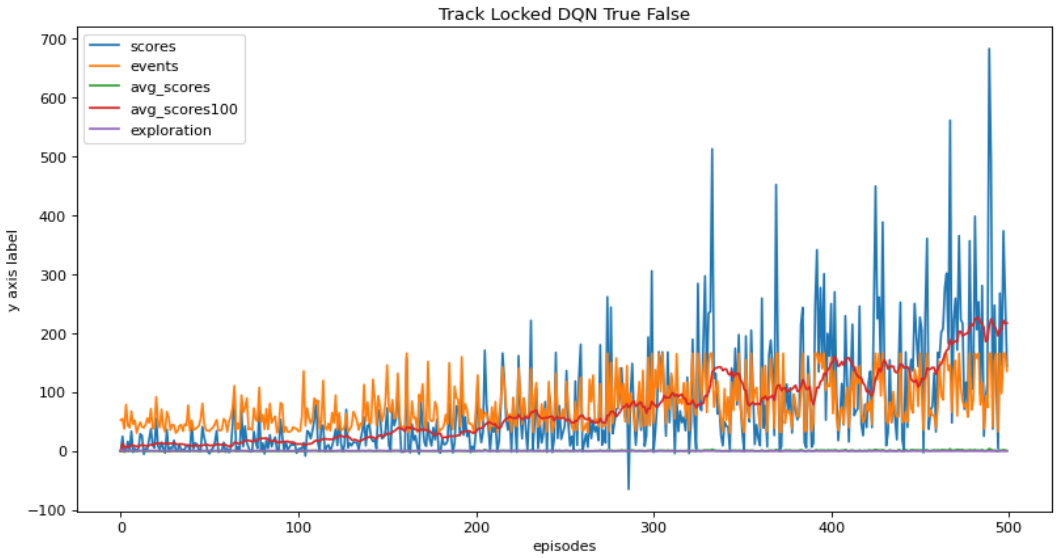

Figure 7: Desempenho do modelo Double DQN em controlo contínuo (segunda configuração).

No modo de controlo discreto, o modelo Double DQN demonstrou desempenho moderado em pistas normais, mas ainda apresentou dificuldades em generalizar para pistas aleatórias, especialmente nas configurações mais desafiantes.

2.3 Duelling

Nesta implementação, a arquitetura é modificada ao dividir a penúltima camada em dois ramos: o *Value*, que calcula o valor do estado independentemente das ações, e o *Advantage*, que estima a vantagem relativa de realizar uma determinada ação. Esta separação permite que o modelo aprenda a identificar estados valiosos sem necessariamente associá-los diretamente a uma ação específica, algo que não é possível no DQN tradicional.

Esta abordagem apresenta vantagens em cenários onde algumas ações não alteram significativamente o ambiente, evitando que o modelo atribua prioridade desnecessária a ações que têm pouco impacto. Para o ambiente *CarRacing-v2*, espera-se que esta separação mais clara entre o valor do estado e a vantagem das ações ajude o modelo a priorizar de forma mais consistente comportamentos que mantêm o carro na pista.

Durante a implementação deste método, foram utilizados os mesmos hiperparâmetros que demonstraram bom desempenho no DQN, garantindo uma base de comparação justa entre as duas arquiteturas.

2.3.1 Resultados usando Controlo Contínuo

Os resultados do modelo *Dueling DQN* no modo de controlo contínuo estão apresentados na Tabela 5 e na Figura 8. Neste modo, o modelo mostrou boa capacidade de controle em pistas normais, mas ainda encontrou dificuldades em pistas aleatórias.

Pista de Teste	**Recompensa Obtida**
Normal 1	875.90
Normal 2	921.99
Normal 3	870.67
Normal 4	879.46
Normal 5	872.11
Aleatória 1	-100.53
Aleatória 2	-67.53
Aleatória 3	-34.84
Aleatória 4	-65.83

Table 5: Recompensas obtidas pelo modelo Dueling DQN em diferentes tipos de pistas no modo contínuo.

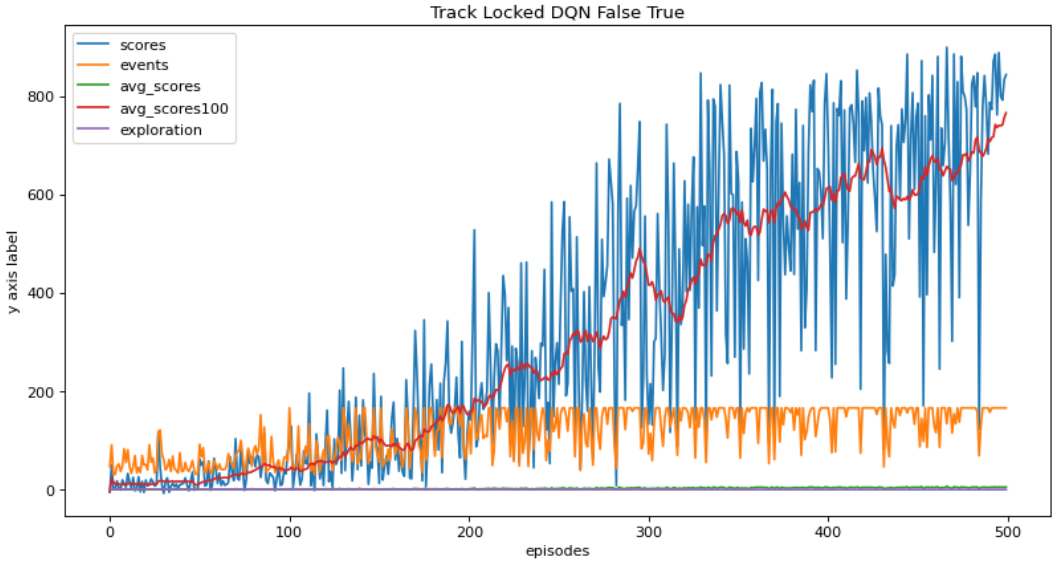

Figure 8: Desempenho do modelo Dueling DQN em controlo contínuo (configuração inicial).

Os resultados indicam que o *Dueling DQN* conseguiu obter altos desempenhos em pistas normais, com recompensas consistentes acima de 870. Entretanto, o desempenho em pistas aleatórias foi limitado, com várias recompensas negativas.

—

2.3.2 Resultados usando Controlo Discreto

No modo de controlo discreto, os resultados do *Dueling DQN* estão apresentados na Tabela 6 e nas Figuras 9 e **??**. O espaço de ações limitado impactou o desempenho, mas o modelo ainda conseguiu obter boas recompensas em algumas pistas normais.

Pista de Teste	**Recompensa Obtida**
Normal 1	855.43
Normal 2	838.90
Normal 3	679.46
Normal 4	831.20
Normal 5	764.15
Aleatória 1	-257.36
Aleatória 2	-79.42
Aleatória 3	-29.84
Aleatória 4	198.19

Table 6: Recompensas obtidas pelo modelo Dueling DQN em diferentes tipos de pistas no modo discreto.

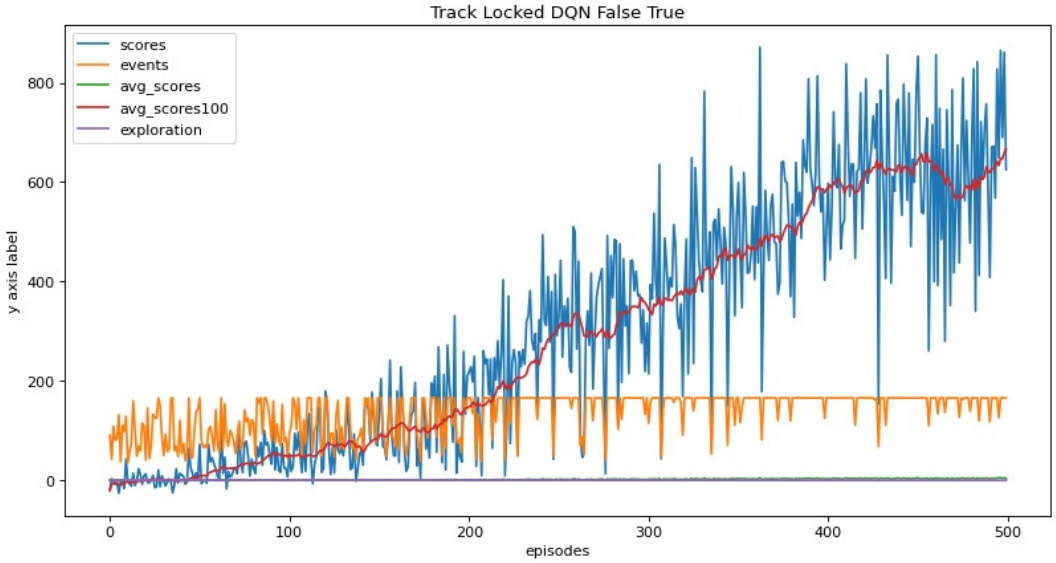

Figure 9: Desempenho do modelo Dueling DQN em controlo discreto (configuração inicial).

Apesar das limitações do espaço de ações discretas, o modelo conseguiu atingir altas recompensas em pistas normais. No entanto, em pistas aleatórias, o desempenho foi inconsistente, com variações significativas nas recompensas.

O *Dueling DQN* mostrou ser eficaz em identificar estados valiosos, apresentando melhorias em pistas normais em comparação com o DQN e o Double DQN. No entanto, as dificuldades em generalizar para pistas aleatórias destacam a necessidade de maior diversidade nos dados de treino.

2.4 Teste do melhor modelo com UseRandomize=True

Conforme pedido segue o teste do nosso melhor modelo com a 'flag' UseRandomize=True

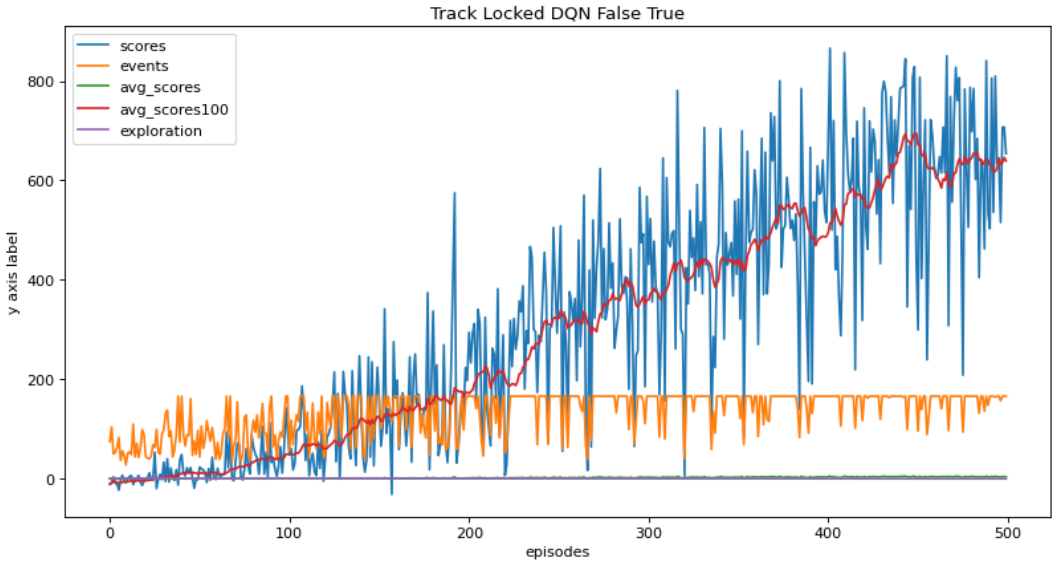

Figure 10: Arquitetura da rede DQN com *dueling*. A rede separa o cálculo do valor do estado (*value*) e da vantagem das ações (*advantage*), que são combinados para calcular os valores Q.

2.5 Conclusão

Com base nos testes realizados a melhor arquitetura foi utilizar Dueling DQN com controlo Discreto. Obtiveram maiores scores, atingindo o objetivo prentendido e além disso também teve melhor desempenho nos testes de avalização. Para melhorar a preformance do modelo seriam necessários mais episódios.

3 Parte II

3.1 Arquitetura

Nesta segunda parte do trabalho utilizámos uma arquitetura semelhante à da primeira parte, alterando a extração de features de camadas convolucionais por camadas lineares. Optámos por uma arquitetura simples que obteve resultados variados, muitos valores negativos, contudo, foi possível alcançar os objetivos, mesmo com alterações das posições de partida e de chegada. Baseados nos testes da primeira parte utilzámos o modelo Dueling DQN com controlo em espaço discreto.

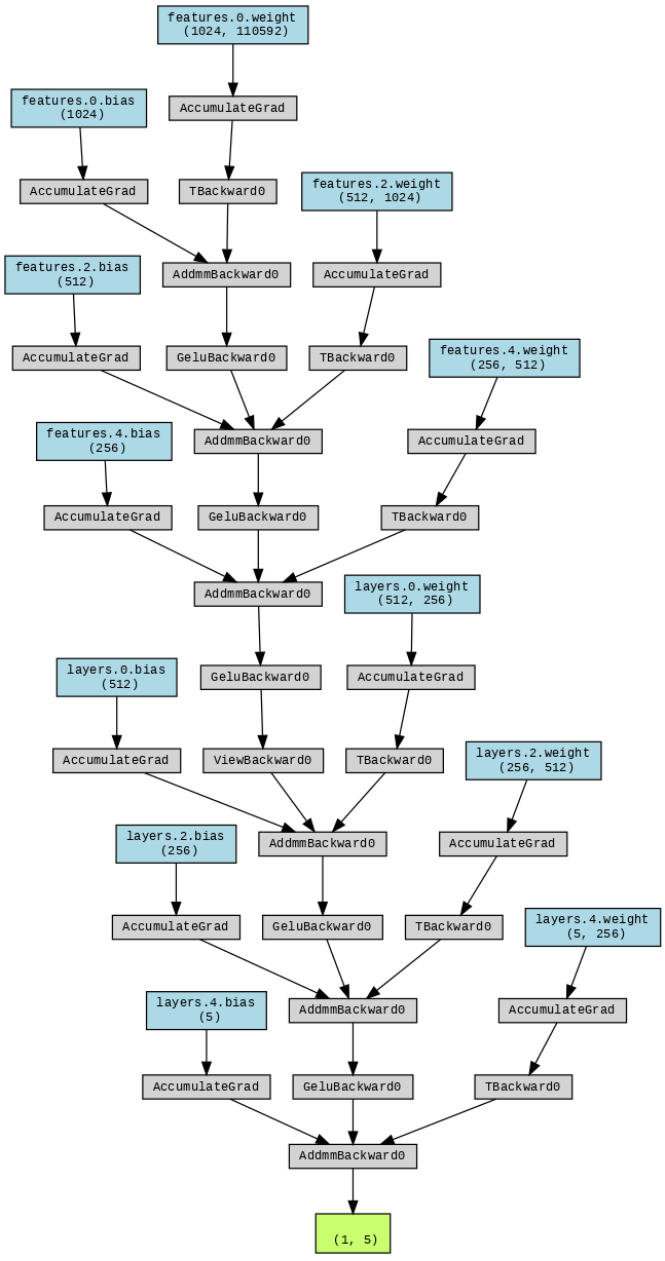

Figure 11: Dueling DQN - Arquitetura

A arquitetura utilizada no Dueling DQN é composta por camadas lineares organizadas em três blocos principais. O bloco inicial de **extração de características** transforma a entrada em uma representação densa através de três camadas lineares com ativações GELU, reduzindo progressivamente as dimensões ($inputs \rightarrow 1024 \rightarrow 512 \rightarrow 256$). A seguir, a rede é dividida em dois *streams*: o **stream de valor**, que calcula o valor intrínseco do estado ($V(s)$) utilizando uma única camada linear ($256 \rightarrow 1$), e o **stream de vantagem**, que estima a vantagem relativa de cada ação ($A(s,a)$) com outra camada linear ($256 \rightarrow outputs$). Finalmente, os dois *streams* são combinados para calcular os valores $Q(s,a)$, permitindo que o agente escolha as melhores ações com base no estado atual.

3.2 Resultados

3.2.1 Primeiro Cenário

Neste primeirto cenário é usada apenas a flag PoseOnly, onde apenas a posição do robô em relação ao objeto é fornecida como entrada. Este cenário é mais simples pois o agente não precisa de lidar com informações sobre os obstáculos, o foco está em aprender a trajetória direta para alcançar o objetivo.

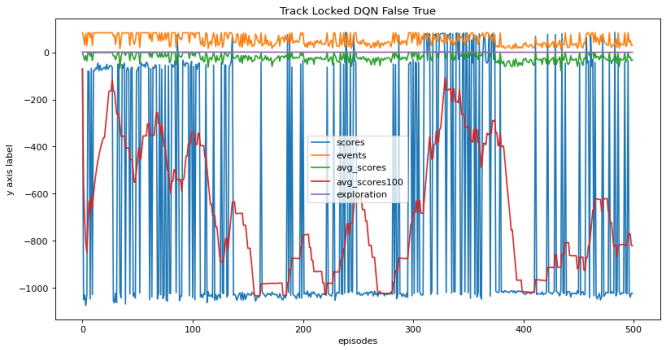

Figure 12: Resultados Obtidos Primeiro Cenário

Os resultados no gráfico refletem o desempenho do agente neste cenário, com a abordagem Dueling DQN com controlo discreto:

- O agente demonstra uma melhoria progressiva nas recompensas médias após aproximadamente 200 episódios, evidenciando aprendizagem da tarefa.
- Após 300 episódios, o agente mostra maior consistência nas recompensas e eventos, mas ainda com oscilações em alguns episódios.

Apesar das melhorias nos últimos episódios as recompensas médias permanecem negativas indicando possibles otimizações. O modelo alcançou o objetivo com alguma frequência porém é necessário adicionar robustez ao modelo e consistência por exemplo aumentando o número de episódiosm, diminuir o learning rate, e possívelmente adicionar uma camada de dropout no modelo.

3.2.2 Segundo Cenário

Neste segundo cenário o robô recebe informações adicionais, incluíndo um mapa local representando os obstáculos. Este cenário é mais desafiante pois o agente precisa de processar informações espaciais e lidar com obstáculos para encontrar o caminho ideal até ao objetivo.

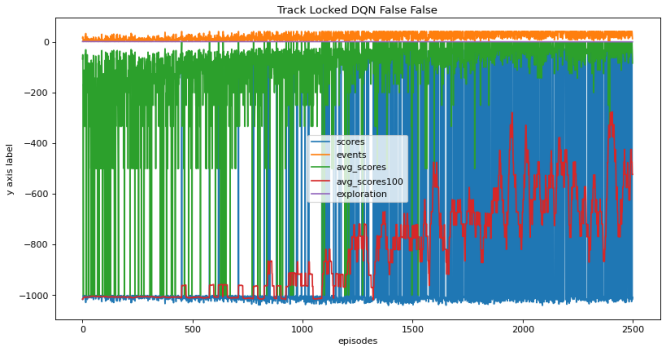

Figure 13: Resultados Obtidos Segundo Cenário

Comparando com o cenário 1 o agente requer mais episódios para estabilizar o seu desempenho devido ao aumento no espaço de estados. Por volta do episódio 1000, a recompensa média começa a estabilizar em valores significativamente melhores do que os episódios iniciais. Após 1500 episódios a recompensa média estabiliza acima de -200 demonstrando que o robo está a aprender a evitar obstáculos.

No cenário 2, o agente demonstrou uma capacidade significativa de lidar com a complexidade, mas precisou de mais episódios para alcançar um desempenho consistente. Contudo alguns testes adicionais poderiam ser efetuados para otimizar a performance do modelo como expandir as dimenões das camadas lineares para lidar com maior quantidade de informação, adicionar uma camada de dropout e ainda testar algumas modificações nos hiperparâmetros.

3.2.3 Conclusão

No cenário PoseOnly, o agente convergiu rapidamente devido ao espaço de estados reduzido e decisões menos complexas, alcançando recompensas médias estáveis em menos episódios. Já no cenário PoseAndLocalMap, a maior dimensionalidade do espaço de estados e a necessidade de lidar com obstáculos aumentaram significativamente a variabilidade nos resultados e o tempo necessário para convergência, embora o modelo tenha demonstrado boa adaptação com treinamento prolongado. Contudo, a variabilidade nas recompensas indica a necessidade de melhorias, como ajuste fino de hiperparâmetros e aumento do número de episódios de treino, para melhorar a consistência e desempenho do agente em cenários de maior complexidade. Na visulaização o resultado obtido para o segundo cenário mostra a capacidade do robo a aprender a evitar resultados.

4 Conclusão Final

Este trabalho explorou a aplicação de redes neurais profundas no contexto de Reinforcement Learning, utilizando o DQN e suas variantes em diferentes cenários. Na **Parte I**, a implementação de uma CNN no DQN foi aplicada ao ambiente CarRacing-v2, demonstrando melhorias no aprendizado ao longo dos episódios, com a variante Dueling DQN contribuindo para maior estabilidade e maiores scores. Já na **Parte II**, a arquitetura Dueling DQN foi avaliada em dois cenários: *PoseOnly*, onde o agente mostrou aprendizado rápido devido à simplicidade do espaço de estados, e *PoseAndLocalMap*, um ambiente mais complexo que exigiu mais treino devido à inclusão de mapas locais com obstáculos. A capacidade do Dueling DQN de separar o valor do estado ($V(s)$) das vantagens de ação ($A(s, a)$) mostrou-se crucial para lidar com cenários complexos, destacando-se pela robustez e capacidade de generalização