FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Universidade de Coimbra

Aprendizagem Profunda Aplicada

2021/2022

Practical Assignment 3

José Pedro Araújo Azevedo
uc2016236736@student.uc.pt

Rafael Alves Vieira
uc2017257239@student.uc.pt

29 de dezembro de 2021

1. Introdução

Este relatório descreve as análises e conclusões obtidas na realização trabalho prático nº3, que consiste na implementação de várias *networks DQN (Deep Q-Networks)* utilizando os conceitos leccionados de *Deep Reinforcement Learning* .
Primeiramente no *environment CartPole-v0* vão ser testadas quatro redes, nomeadamente *DQN*, *Double DQN*, *Dueling DQN* e *Double Dueling DQN* assim como no *enviroment CarRacing-v0*, for fim vai-se efetuar combinações de hiperparâmetros para tentar melhorar a rede e os resultados obtidos.

2. Métodos utilizados e informações sobre as arquiteturas

2.1 Replay buffer

Em ambas as versões o *replay buffer* não tem prioridades implementadas, por isso podemos ter esquecimento catastrófico a certo ponto do treino. Isto pode observar-se em alguns gráficos que mostramos no relatório (*CartPole* e *CarRacing*).

2.2 Double DQN

Analisando a equação *bellman* podemos ver que a *DQN* tradicional usa o seu próprio modelo para estimar valores de estados futuros. Por isso sempre que atualizamos os parâmetros para o modelo no nosso estado atual, estamos a influenciar os valores obtidos do estado futuro porque esta estimativa é feita com o mesmo modelo, e então o valor obtido da *target net* também muda para o próprio estado futuro.
Para resolver esta confusão, usa-se *Double DQN* em que temos dois modelos, o modelo normal *DQN* que treinamos a cada *step* e um segundo modelo (cópia do primeiro) que substitui a parte do cálculo do valor do proximo estado. A segunda *DQN* apenas serve para este fim, usa o valor máximo da *policy net* do próximo estado para calcular o valor da *target net* do próximo estado.
A primeira *DQN* ainda é a que se usa para calcular a *loss*, com o seu valor atual, mas com o *target value* da segunda *DQN*. Isto resolve o problema de *overconfident estimations*. Em muitas aplicações este método melhora os resultados e diminui o tempo de treino, mas nas nossas duas alicações (*Car pole* e *Car Racing*) não notámos essas diferenças.

2.3 Dueling

Aqui a arquitetura da *DQN* foi alterada de modo a se dividir no final (na última *FC layer*) entre uma layer que devolve o valor da *value funcion* e outra da *advantage function*. O valor da *value funcion*

representa o reward que se obtém de cada estado e os valores da *advantage function* representam o par de ação-estado. Para que a soma da advantage seja sempre zero, o valor médio é subtraido de todos os valores. Ou seja, em código (car racing) ficaria:

```
if(useDueling):
    x = self.features(x)
    x = x.view(x.size(0), -1)
    x = self.layersdueling(x)
    advantage = self.advantage(x)
    value     = self.value(x)
    return value + advantage  - advantage.mean()
```

Figura 1: Excerto do código aplicado na função *foward*.

Onde as layers/arquitetura da rede usadas podem ser, por exemplo como temos no car racing:

```
self.layers = nn.Sequential(
                #Add code here for DQN
    nn.Linear(in_features=18432 , out_features=256),#256
    nn.ReLU(),
    nn.Dropout(p=0.1),
    nn.ReLU(),
    nn.Linear(in_features=256, out_features=self.output_size),
    nn.ReLU(),
)


self.layersdueling = nn.Sequential(
            #Add code here for Dueling
    nn.Linear(in_features=18432 , out_features=128),#256
    nn.ReLU(),
)

self.advantage = nn.Sequential(
  nn.Linear(in_features=128, out_features=self.output_size),
  nn.ReLU(),
)

self.value = nn.Sequential(
  nn.Linear(in_features=128, out_features=1),
  nn.ReLU(),
 )
```

Figura 2: Excerto do código aplicado, arquitetura da rede *CarRacing*.

Este método será eficaz no *Car Racing*. Algumas vantagens são as seguintes: como estamos a separar o *value*, damos mais liberdade à nossa NN de aprender quais os estados é que possuem valor mais alto, ou seja, neste caso seriam os estados em que o carro está numa curva ou tem possibilidade de se despistar e sair da track onde obtém *rewards*. Estados menos importantes são aqueles em que o carro se movimenta apenas em frente e em linha reta. Ou seja, em estados menos importantes não precisamos de querer saber eminentemente, a cada step, quais as próximas ações a tomar.

3. CartPole-v0

Este *environment* consiste num poste que se move ao longo de uma trilha sem atrito. O sistema é controlado aplicando uma força de +1 ou -1 ao *cart*. O poste que o *cart* possui começa numa posição vertical, com o objetivo deste não cair.
Este *environment* é representado por quatro valores: posição do *cart*, velocidade do mesmo, ângulo do poste e velocidade do poste. Existem duas ações possíveis: mover para a esquerda ou direita, uma

recompensa de +1 é fornecida a cada *timestep* que o poste permaneça na vertical(podendo este ângulo variar até 12 graus para a esquerda ou direira), caso o poste se mova mais de 12 graus, ou se mova 2.4 unidades desde o centro, o episódio termina.

3.1. CartPole-v0 com network DQN

Inicialmente criou-se uma *network* simples com apenas 2 *layers fully connected*.

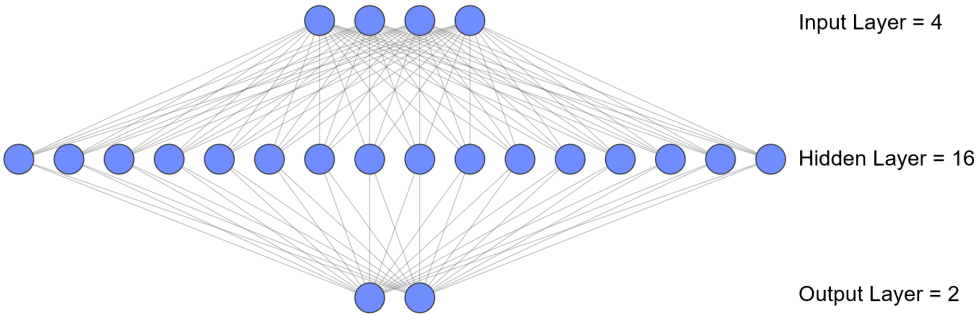

Figura 3: Ilustração da arquitetura utilizada.

Parâmetros utilizados:

- Total Episodes = 1000.
- Max Steps = 200.
- Batch Size = 128.
- Exploration Decay = 0.002.
- Discount Factor = 0.99.
- Learning Rate = 0.001

De salientar que estes parâmetros foram mantidos para todas as versões de *networks*. Deixando o *Exploration Decay* no valor *0.002* tem-se bastantes episódios em que agente realiza ações aleatórias, esta probabilidade vai descendo até chegar ao *treshold* de 0.01, na figura abaixo é possível observar este valor.

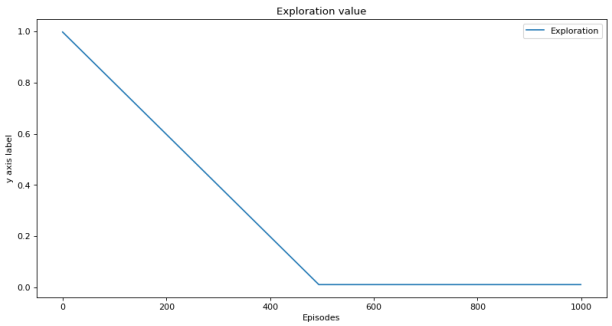

Figura 4: Gráfico do valor de *exploration*, em função do número de episódios.

Resultados obtidos

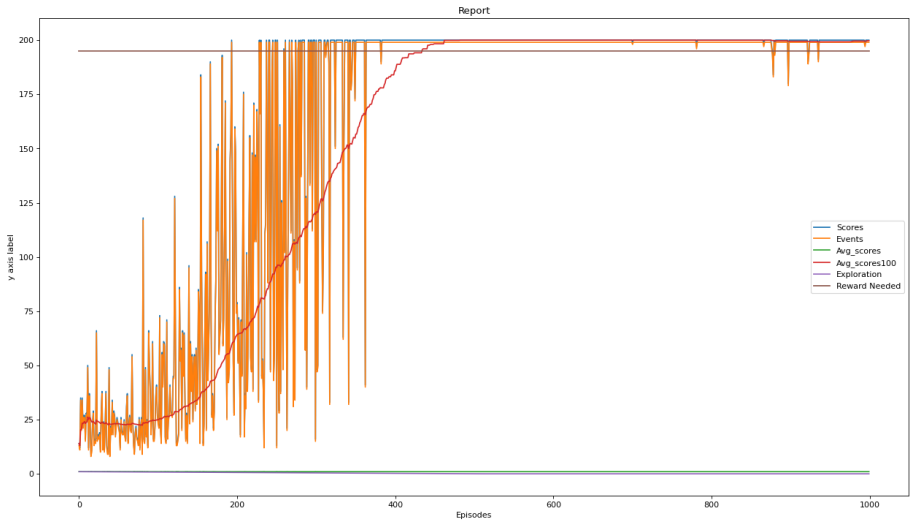

Figura 5: Gráfico de resultados obtidos para a rede DQN.

Tempo de execução: 8 minutos 51 segundos.
Primeiro episódio a obter reward necessária: 434.
Percentagem de Episódios com reward igual ou maior que 195: 56.60%.
Últimos 20 episódios com reward igual ou maior que 195: Sim.

3.2. CartPole-v0 com network Double DQN

Resultados obtidos

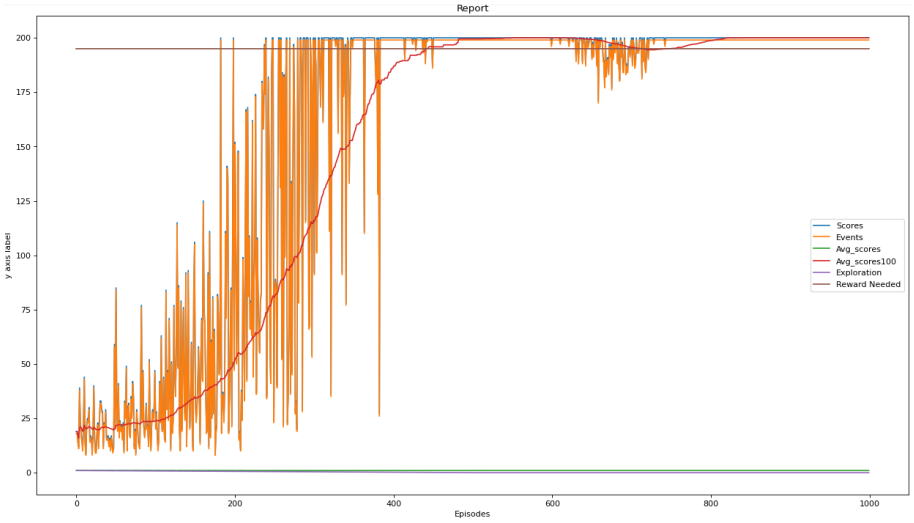

Figura 6: Gráfico de resultados obtidos para a rede Double DQN.

Tempo de execução: 16 minutos 29 segundos.
Primeiro episódio a obter reward necessária: 443.
Percentagem de Episódios com reward igual ou maior que 195: 52.40%.
Últimos 20 episódios com reward igual ou maior que 195: Sim.

3.3. CartPole-v0 com network Dueling DQN

Resultados obtidos

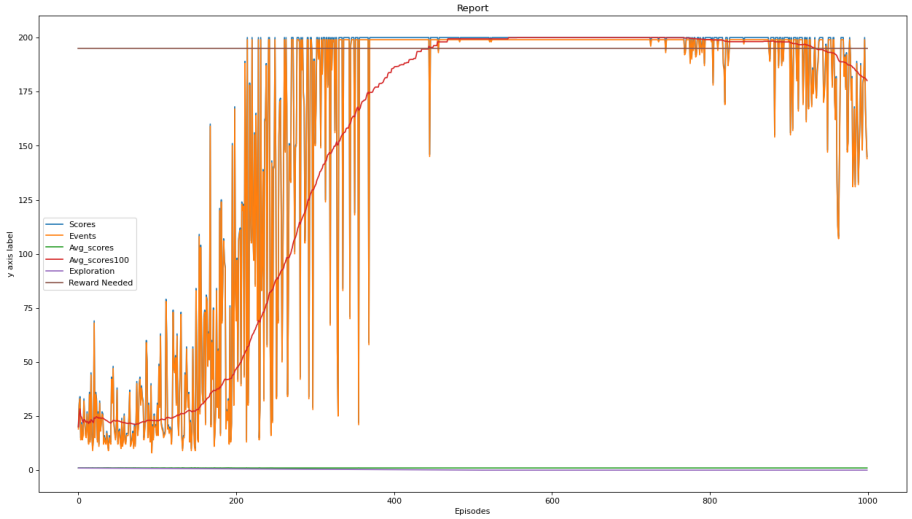

Figura 7: Gráfico de resultados obtidos para a rede Dueling DQN.

Tempo de execução: 15 minutos 48 segundos.
Primeiro episódio a obter reward necessária: 444.
Percentagem de Episódios com reward igual ou maior que 195: 49.00%.
Últimos 20 episódios com reward igual ou maior que 195: Não.

3.4. CartPole-v0 com network Double Dueling DQN

Resultados obtidos

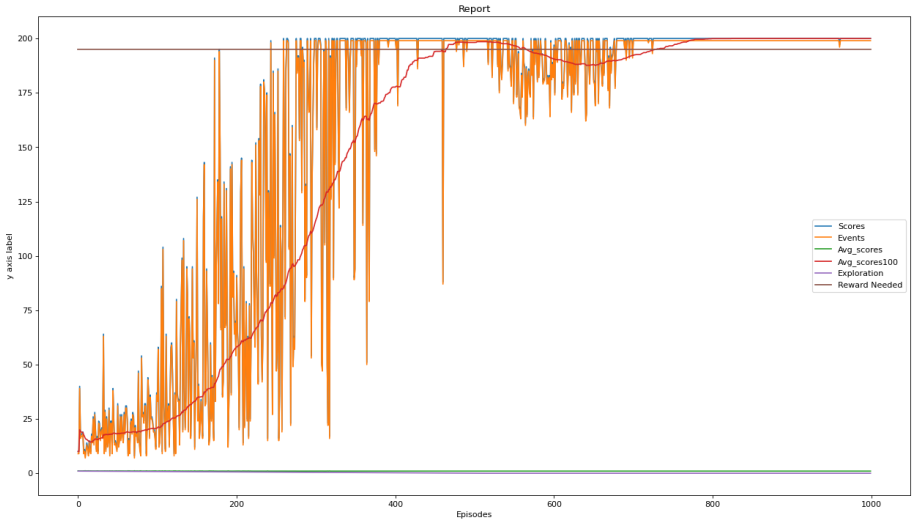

Figura 8: Gráfico de resultados obtidos para a rede Double Dueling DQN.

Tempo de execução: 19 minutos 7 segundos.
Primeiro episódio a obter reward necessária: 459.

Percentagem de Episódios com reward igual ou maior que 195: 47.20%.
Últimos 20 episódios com reward igual ou maior que 195: Sim.

3.5. Combinação de hiperparâmetros com network *DQN*

Tendo-se obtidos bons resultados com esta *network* e sendo esta com menor tempo de execução, irá-se combinar alguns hiperparâmetros na tentativa de melhorar o modelo e diminuir o tempo de execução. Observando a *Figura 5* e os resultados obtidos, sabe-se que no episódio 434 a *network* obtém a *reward* necessária, pode-se reduzir os episódios para 750 por exemplo, manipular a *Learning Rate*, *Batch size* e os valores de controlo(*max steps, exploration values , discount factor*) de forma a este treino ser mais rápido, no entanto, com a finalidade de obter 195 de *reward* nos últimos 20 episódios e o modelo não sofrer *Exploitation*.

Modificações

1 - Reduzir episódios para 600

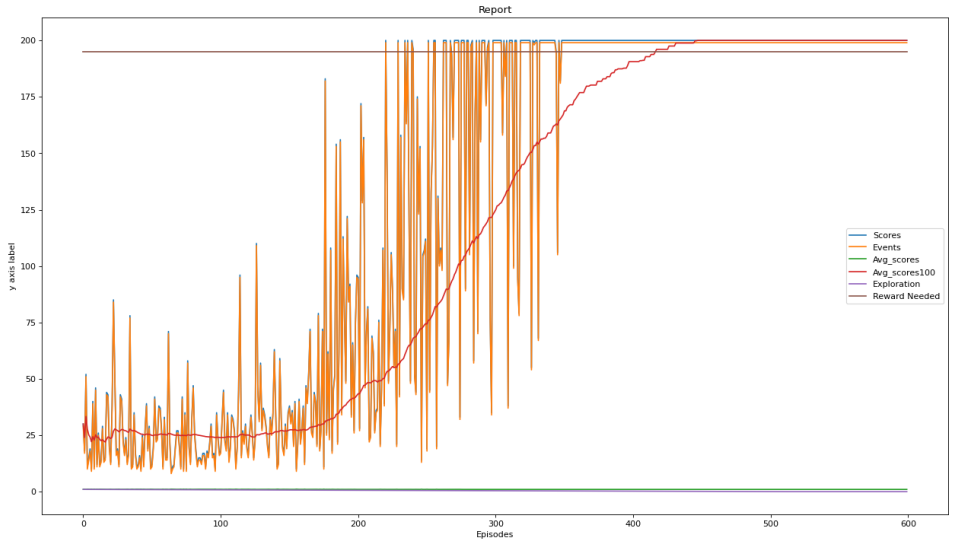

Figura 9: Gráfico de resultados obtidos para a rede DQN, 600 episódios.

Tempo de execução: 3 minutos 32 segundos.
Primeiro episódio a obter reward necessária: 417.
Número de episódios a obter reward igual ou maior que 195: 183.
Percentagem de Episódios com reward igual ou maior que 195: 30.50%.
Últimos 20 episódios com reward igual ou maior que 195: Sim.

Como o *Exploration Threshold* é atingido no episódio 500, temos ainda mais 100 episódios para concluir se o modelo se comporta da forma pretendida. Pela observação da *Figura 7*, conclui-se que 600 episódios são suficientes para a finalidade pretendida deste *environment*.

2 - Variáveis de controlo

Nesta etapa manteram-se todos os parâmetros mencionados em *3.1*, com a diferença do número de episódios que passou para 600 e a mudança de variáveis de controlo que se foram mudando.

Combinações de *Exploration Decay*

Exploration decay	**0.002**	**0.0025**	**0.005**	**0.1**
Primeiro episódio a obter +=195	**417**	364	236	287
Número de episódios com reward +=195	**183**	236	144	235
Percentagem de episódios com reward necessária	**30.50%**	39.33%	24.00	39.17%
Últimos 20 episódios com reward +=195	**Sim**	Sim	Não	Não
Tempo de execução	**3:32 minutos**	4:12 minutos	4:10 minutos	4:01 minutos

Figura 10: Tabela de combinações, *Exploration decay*.

Com o aumento do *Exploration Decay*, o agente deixa de tomar decições aleatórias mais rapidamente, o que pode necessitar um aumento no número de episódios para obter a *reward* necessária.
Desta forma o agente explora mais o *environment* o que em maioria dos casos é benéfico em termos de resolução do problema, porque este vai aprender o comportamento ideal do environment.
Por observação da *Figura 10*, para os valores *0.005* e *0.1* apesar de o modelo obter 195 de *reward*, mais tarde a *reward* desce. Como nestes casos obtém-se a *reward* mais cedo, futuramente a *network* "esquece-se", um destes motivos é o facto de não se implementar *priority replay buffer*.

Para as proximas etapas irá-se considerar o valor de 0.002 de *Exploration Decay*.

Combinações de *Discount Factor*

Discount Factor	**0.5**	**0.85**	**0.9**	**0.95**	**0.99**
Primeiro episódio a obter +=195	--	431	535	449	**417**
Número de episódios com reward +=195	--	118	65	151	**183**
Percentagem de episódios com reward necessária	--	19.67%	10.83%	25.16%	**30.50%**
Últimos 20 episódios com reward +=195	Não	Não	Sim	Sim	**Sim**
Tempo de execução	1:48 minutos	3:35 minutos	3:30 minutos	3:26 minutos	**3:32 minutos**

Figura 11: Tabela de combinações, *Discount Factor*.

Com o aumento do *Discount Factor*, o agente evita negligenciar *rewards* futuras, deste modo o *Discount Factor* deve ser o mais próximo de *1* para obter melhores resultados.
É possível observar este facto por observação da *Figura 9*.

Combinações de *Max Steps*

Max Steps	**200**	**225**	**500**
Primeiro episódio a obter +=195	**417**	426	431
Número de episódios com reward +=195	**183**	174	169
Percentagem de episódios com reward necessária	**30.50%**	29.00%	28.17%
Últimos 20 episódios com reward +=195	**Sim**	Sim	Sim
Tempo de execução	**3:32 minutos**	3:39 minutos	3:39 minutos

Figura 12: Tabela de combinações, *Max Steps*.

Efetivamente é necessário ter 200 ou mais steps para o agente obter a *reward* necessária, aumentado este valor não trás benefícios tanto no número de episódio em que obtém a *reward* de 195, tanto como em tempo de execução.

3 - Combinações de *Learning Rate*

Learning Rate	**0.002**	**0.0015**	**0.001**	**0.0005**
Primeiro episódio a obter +=195	571	561	**449**	477
Número de episódios com reward +=195	29	39	**151**	123
Percentagem de episódios com reward necessária	4.83%	6.50%	**25.16%**	20.50%
Últimos 20 episódios com reward +=195	Sim	Sim	**Sim**	Sim
Tempo de execução	3:19 minutos	3:28 minutos	**3:32 minutos**	3:17 minutos

Figura 13: Tabela de combinações, *Learning Rate*.

Implementar um valor de *Learning Rate* igual ou próximo do valor de *Exploration Decay*, poderia ser benéfico em termos de tempo de execução e qualidade do modelo. Por observação da *Figura 11*, conclui-se que com valores de *Learning Rate* mais baixos, o modelo obtém a *reward* necessária mais cedo.
Implementou-se ainda o *scheduler MultiStepLR* aumentando a *Learning Rate* em determinados episódios, no entanto, não se irá documentar essa parte, porque não trouxe resultados benéficos em termos de tempo de execução.

4 - Combinações de diferentes *Batch Sizes*

Batch Size	**32**	**64**	**128**	**256**
Primeiro episódio a obter +=195	433	**440**	449	496
Número de episódios com reward +=195	167	**160**	151	104
Percentagem de episódios com reward necessária	27.83%	**26.67%**	25.16%	17.33%
Últimos 20 episódios com reward +=195	Sim	**Sim**	Sim	Sim
Tempo de execução	3:37 minutos	**3:28 minutos**	3:32 minutos	3:40 minutos

Figura 14: Tabela de combinações, *Batch Sizes*.

Conclui-se que aumentar o *Batch size*, faz com que o episódio a ter a *reward* pretendida aconteça mais tarde, no entanto, usar *Batch size* de 32, 64 ou 128 tem-se bons resultados.
Irá-se considerar *batch size* de 64 para o melhor modelo.

Resultados do melhor modelo

Parâmetros utilizados:

- Total Episodes = 600.
- Max Steps = 200.
- Batch Size = 64.
- Exploration Decay = 0.002.
- Discount Factor = 0.99.
- Learning Rate = 0.001
- Otimizador = *Adam*.

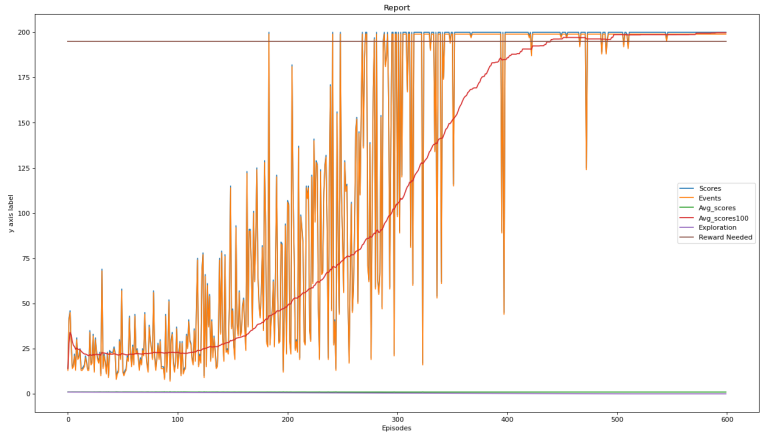

Figura 15: Gráfico de resultados obtidos, *melhor rede*.

Tempo de execução: 3 minutos 28 segundos.
Primeiro episódio a obter reward necessária: 440.
Percentagem de Episódios com reward igual ou maior que 195: 26.67%.
Últimos 20 episódios com reward igual ou maior que 195: Sim.

4. CarRacing-v0

4.1. Arquitetura da rede

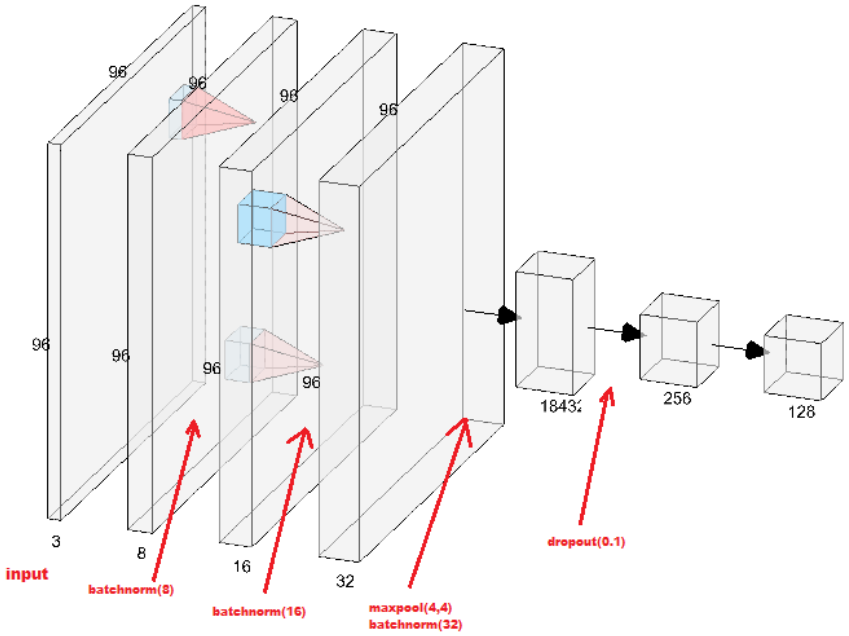

Figura 16: Ilustração da arquitetura utilizada.

4.2. Hiper-Parâmetros

- **Action Space:** Fizemos algumas experiências com o *action space* mas acabámos por não tirar muito benefício disso, e acabámos por deixar como estava.
- **Total Episodes:** Na *DQN* default chegámos a conseguir ter resultados quase máximos em score em 500 episodios, no entanto para a DDQN e Dueling precisámos de usar 1000 episódios.
- **MaxSteps:** 1000.
- **Control Steps:** Aqui entra o conceito de delayed rewards: não podemos esperar que a nossa rede reconheça decisões erradas se está sempre a fazer decisões novas a cada passo. Se deixarmos que a mesma ação seja repetida por exemplo entre 5 a 10 passos seguidos podemos concluir atravéz do reward obtido se essa ação foi boa ou má com mais certeza.
- **Frame Stack:** Os melhores valores obtidos para um frame stack foi 2 ou 3 frames. Como isto

representa também os estados futuros não queremos exagerar neste valor, mas também ter um valor pequeno como 1 ou 2 acaba por ser pouco e não tornar eficiente as nossas estimativas.

- **Freeze Counter:** 25.
- **Batch Size:** 64.
- **Exploration vs Exploitation**: No enunciado pedia para implementarmos epsilon-greedy policy mas após analisar o código verificámos que isso já estava implementado. No entanto mudámos ligeiramente os hiper parametros que controlam a exploration:

  exploration threshold = 1

  exploration threshold minimum = 0.02

  exploration decay = 0.997

  Estes valores foram usados para 1000 episodios. Exprimentámos a diminuir o exploration decay para descermos mais rapido o exploration threshold e tomarmos decisoes menos aleatorias mais rapidamente. E com medo que isto trouxesse maus resultados no futuro por atingirmos um minimo mais rapidamente, aumentámos o minimo para haver mais decisoes aleatorias quando este minimo é atingido. No entanto depois voltámos a usar 0.997 porque o mínimo era atingido demasiado rápido, mas usámos 0.996 para 500 episódios.
  Valores adaptados para 500 episodios:

  exploration threshold = 1

  exploration threshold minimum = 0.02

  exploration decay = 0.996

  Este valor foi adaptado desta forma de modo a que durante 500 episodios atinjamos o mínimo de exploration threshold durante alguns episodios, o que nao seria possivel com 0.997.

- **Discount Factor** = 0.95
- **Learning Rate** = 0.0001 (no car racing nao usámos adaptive LR)
  Otimizador: Adam
- **Save Counter** = 50
- **Reset Counter** = 20 (foi alterado para 20 para nao perdermos muito tempo a andar na relva e poupar tempo de treino.)
- **Init Counter** = 20
- **Frame Size** = 3 (não usámos escala de cinzentos)

4.3. Resultados obtidos

4.3.1 DQN

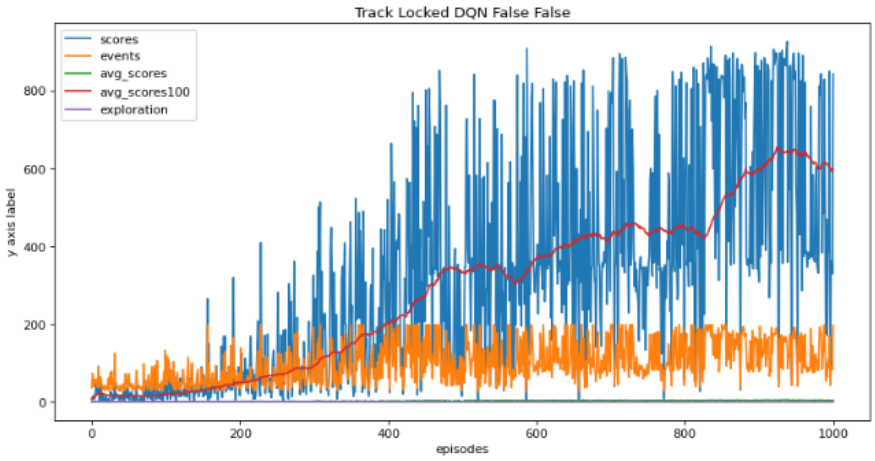

Figura 17: Resultados obtidos DQN

Podemos observar no gráfico alguns picos baixos na zona azul (e na linha vermelha) que podem indicar algum esquecimento. Como podemos ver conseguimos obter logo alguns concorrentes para a best net antes dos 500 episodios.

4.3.2 Dueling DQN

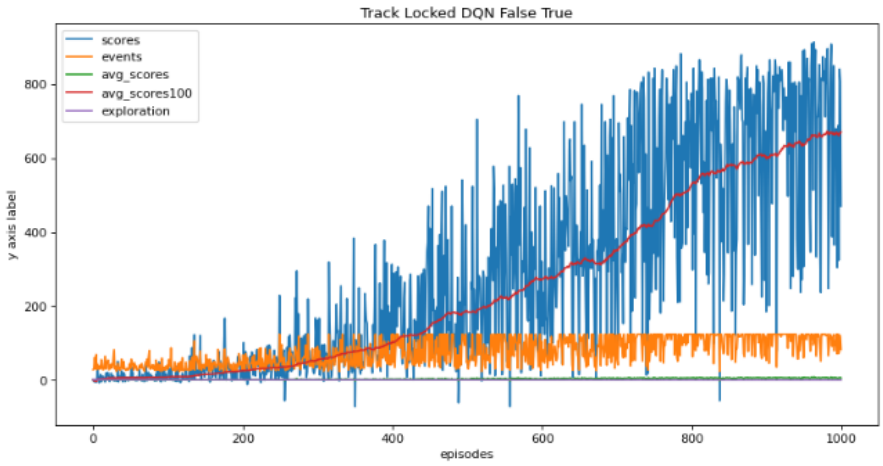

Figura 18: Por curiosidade exprimentámos um treino com dueling sem double Q network, apenas com DQN. Estes foram os resultados.

4.3.3 Double DQN e Double Dueling DQN

A implementação da DDQN em car racing foi com o seguinte código (seguindo o exemplo do código dado para DQN e dos slides das aulas teóricas):

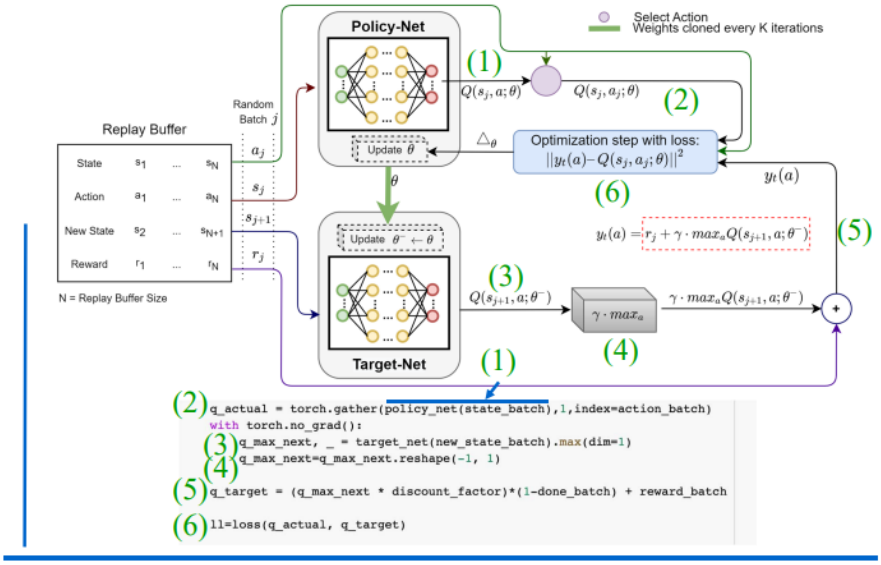

Figura 19: Esquema para o desenvolvimento do código da estimativa do próximo estado em DQN.

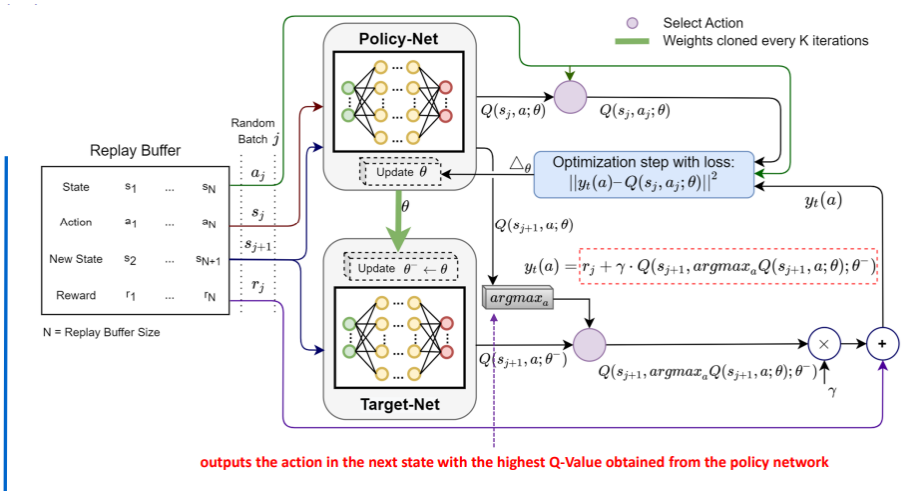

Figura 20: Esquema da DDQN

```
if usedoubleDQN:
    q_actual        = torch.gather(policy_net(state_batch),1,index=action_batch)

    next_policy_vals = policy_net(new_state_batch)
    target_vals      = target_net(new_state_batch)

    with torch.no_grad():
      q_next =  torch.gather(target_vals,1,torch.max(next_policy_vals,1)[1].unsqueeze(1))
      q_next.reshape(-1, 1)

    q_target = (q_next * discount_factor)*(1-done_batch) + reward_batch

    ll=loss(q_actual, q_target)
```

Figura 21: Código para a estimativa do proximo estado em DDQN. Ou seja, tivemos de alterar os passos 3 e 4. E antes de ir buscar o valor Q do proximo estado precisamos de ter uma segunda dqn que usa o new state batch em vez de state batch onde vamos buscar o valor usado no calculo do target.

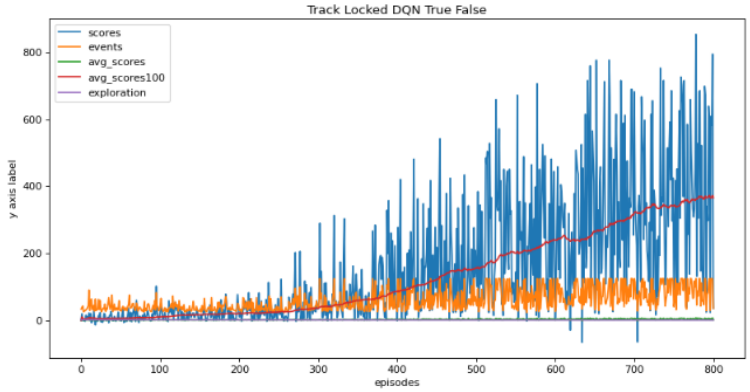

Figura 22: Resultados obtidos DDQN

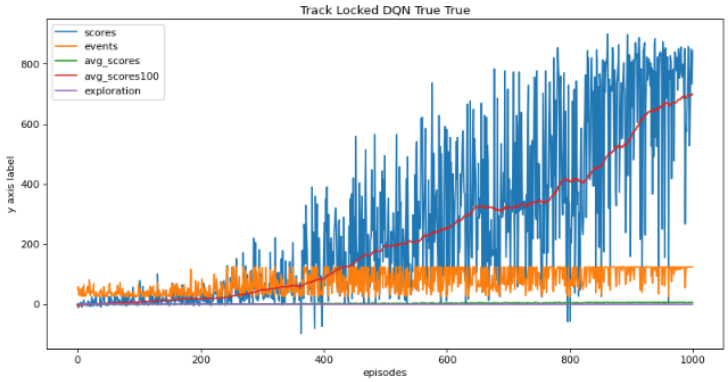

Figura 23: Resultados obtidos DDQN dueling

Quer em DDQN quer em DDQN dueling conseguimos reparar que o processo de treino é mais consistente do que na DQN. E que, de facto, ao contrário do que é previsto teóricamente, usar DDQN piorou os resultados finais. Ficámos por saber uma explicação exata para este fenómeno, mas acreditamos que em DDQN possamos precisar de mais tempo de treino do que na DQN. Mas, por sua vez Dueling melhora os resultados que obtivemos na DDQN.

5. Respostas , no que diz respeito ao Reinforcement Learning.

Compare the performance of the Deep Q-Learning architecture in both state representations.Compare also the training time. Assess the drawbacks of each representation from the observed performances. O cartpole tem mais simplicidade quer na sua arquitetura da rede (apenas usa FC layers enquanto o car racing usa Conv. layers também), quer no seu enviroment. Portanto todo o processo de observação do estado é o que trás mais complexidade ao car racing, e cria uma diferença enorme no tempo de treino.

Propose a new reward model (if necessary) for the CartPole-v0 environment. You do not need to implement the new reward model but should clearly compare and highlight the advantages and disadvantages of both models. If you think there is no need for a new model you should explain your reasoning.
O *reward model* fornecido, não é um "bom"exemplo de ser aplicado num problema de *DeepRL*, isto porque não existe penalização quando o poste atinge os máximos valores indicados(ângulo de 12 graus, o *cart* mover-se mais de 2.4 unidades), para tal deveria ser aplicada uma *reward* de -1 nestes casos. No entanto, decidiu-se não alterar o *reward model*, visto que não foi necessário.

Compare the performance of the Deep Q-Learning architecture with DQN, Double DQN and Double Dueling DQN. How can you explain the gap in performance (if any)? As comparações de performance e explicaçao de resultados foi realizada durante a avaliaçao dos resultados obtidos nas páginas anteriores.

Experiment with Q-learning hyperparameters (Deep layers, action replay memory, batch size, gamma, epsilon, episode, etc). Discuss the effect of the Deep Q-learning hyperparameters on the methods performance. Estas combinações de hiperparâmetros foram implementadas e discutidas no ponto 3.5.

You should find a parameter that makes a nontrivial difference on the performance. Encontrámos parâmetros como o control steps, frame stack e parâmetros relacionados com exploration que fazem toda a diferença no treino. Estas diferenças na performance estão detalhadas nas secções anteriores.

Training for a larger number of episodes would likely yield further improvements in performance? Explain your answer. Talvez sim, se tivessemos um priority replay buffer, porque aumentar o número de episódios significa que vamos estar a prolongar mais o treino para obter resultados ideais, e durante este processo pode ocorrer esquecimento catastrófico e em vez de ter melhores resultados acabamos por ter o efeito contrário, e depois estamos a perder tempo a aprender algo que ja tinhamos aprendido. No entanto, chegámos a tentar treinos para mais de 1000 episodios no *CarRacing*, porque ao analisar as nossas curvas dos scores, viu-se que em ddqn e dueling ainda estavam com tendência a crescer nos 1000 episodios, mas nunca conseguimos acabar porque ficámos sempre sem *GPU* do google collab.

Run the best model obtained in the OpenAI CarRacing-v0 environment using only one track in an unmodified version (one track per episode). Comment the results

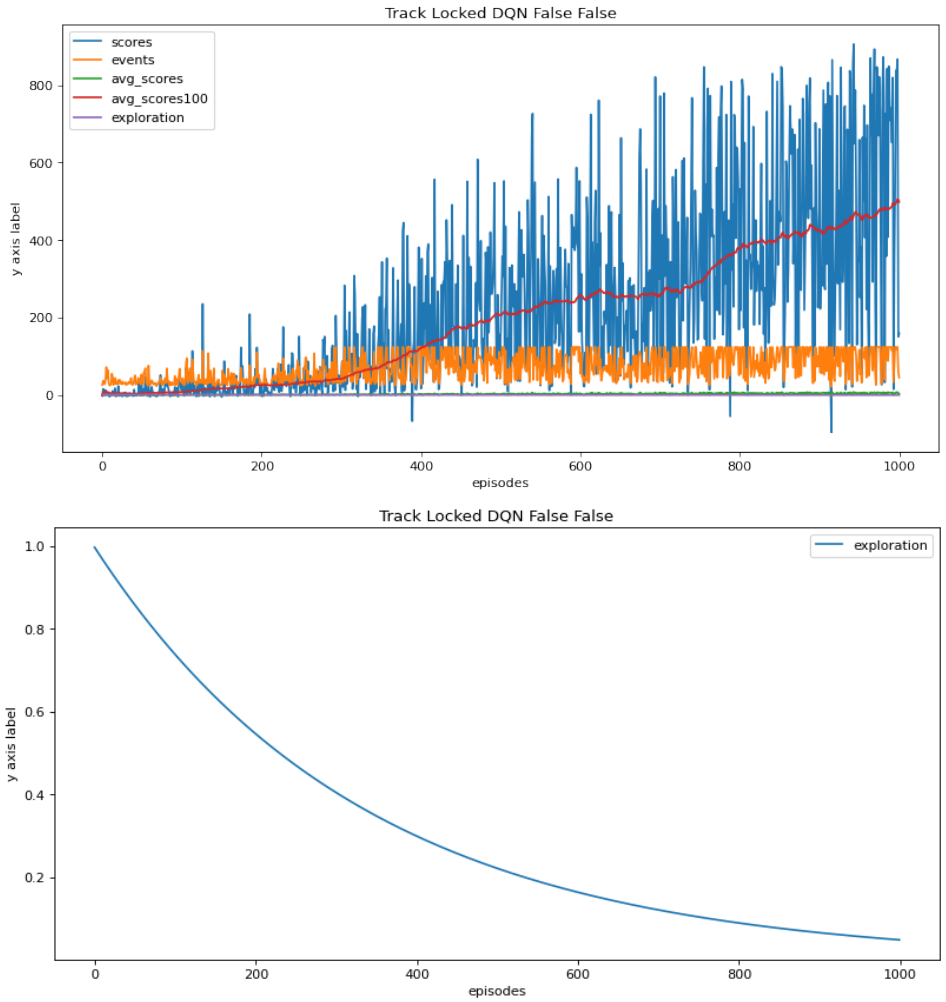

Figura 24: Resultados obtidos, e valor de *Exploration* em função do número de episódios.

O modelo que pensamos ser o melhor é onde obtivemos melhores scores que apesar de o processo de aprendizagem nao ser tão consistente e confiável, é o DQN. E como era de esperar, se usarmos uma pista diferente em todos os episodios em vez de usar sempre a mesma pista, vamos obter scores inferiores.

Webgrafia

https://pytorch.org/
https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-nois
https://towardsdatascience.com/dueling-deep-q-networks-81ffab672751

Bibliografia

Deep Learning with PyTorch by Eli Stevens