Aprendizagem Profunda Aplicada – 2021/2022

Trabalho 3 de Aprendizagem Profunda Aplicada

Milena Mori
2016193815

Lucyanno Frota
2016116214

Esse trabalho teve como objetivo o estudo da técnica de double DQN, DQN e Dueling double DQN. Para esse efeito, nos estudamos dois problemas diferentes, um, o *CartPole-v0*, que consistia em uma haste presa em um carrinho, cujo movimento era influenciado pela posição dessa haste. O objetivo desse primeiro problema era tentar equilibrar a haste em cima do carrinho, mantendo o carrinho no centro da tela.

Já o segundo problema consistia em um carrinho de corrida sobre um percurso de asfalto que ele tem que aprender a percorrer, evitando a grama.

1 CartPole-v0

Para esse problema, como já mencionado, nosso objetivo era equilibrar a haste em cima do carrinho sobre o qual ela está apoiada. Para tal, nós testamos várias técnicas, que incluem *DQN*, *double DQN* e *double dueling DQN*.

1.1 Arquitetura

Em nossos testes, foram testadas 4 arquiteturas de rede que contêm somente a camada *fully connected*, as quais nos referimos relativamente à quantidade de *hidden layers*. As arquiteturas são muito simples, e a única diferença entre elas é o numero de camadas.

Todas essas redes possuem as mesmas quantidades de *inputs* (4) e *outputs* (2) da rede, devido ao fato de não termos feito alterações nos espaços de ações e observações. As arquiteturas que testamos foram:

1.1.1 1 Hidden Layers

- Input Layer, n = 4
- Hidden Layer 1, n = 256
- Output Layer, n = 2

1.1.2 2 Hidden Layers

- Input Layer, n = 4
- Hidden Layer 1, n = 128
- Hidden Layer 2, n = 256
- Output Layer, n = 2

1.1.3 4 Hidden Layers

- Input Layer, n = 4
- Hidden Layer 1, n = 32
- Hidden Layer 2, n = 64
- Hidden Layer 3, n = 128
- Hidden Layer 4, n = 256
- Output Layer, n = 2

1.1.4 6 Hidden Layers

- Input Layer, n = 4
- Hidden Layer 1, n = 8
- Hidden Layer 2, n = 16
- Hidden Layer 3, n = 32
- Hidden Layer 4, n = 64

- Hidden Layer 5, n = 128
- Hidden Layer 6, n = 256
- Output Layer, n = 2

Foi utilizado ReLU como função de ativação em todas as arquiteturas.

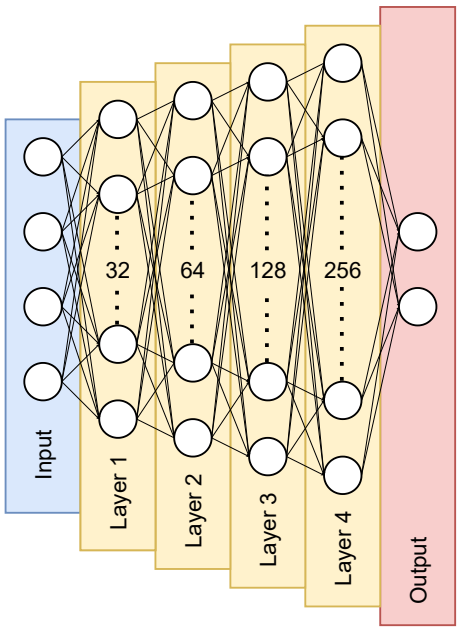

Figure 1: Diagrama da arquitetura de 4 Hidden layers. Os inputs/outputs estão associados com os espaços de observação e ação respectivamente.

A arquitetura que apresentou melhor desempenho em nossos testes for a com 4 *Hidden Layers* (2).

1.2 Observações e Ações

1.2.1 Observações

- Posição do carrinho
- Velocidade do carrinho
- Angulo da haste
- Velocidade angular da haste

1.2.2 Ações

- Mover o carrinho para esquerda
- Mover o carrinho para direita

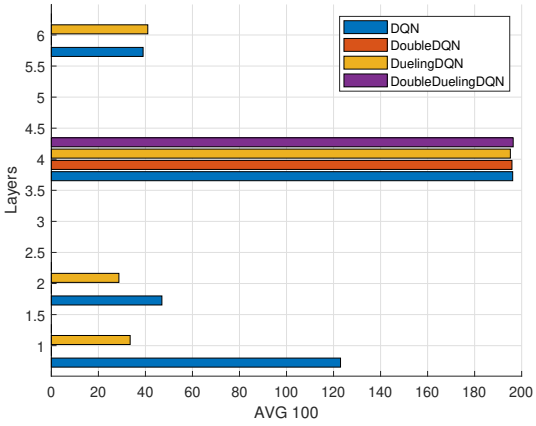

Figure 2: Comparação do desempenhos no quesito media dos últimos 100 episódios dos melhores testes feitos por Layers e Métodos.

1.3 Reward

O mecanismo de *reward* proposto é inteiramente baseado na haste que existe sobre o carrinho, e a posição que o carrinho ocupa sobre a tela não tem qualquer influencia sobre o *reward*.

Muitas vezes, o carrinho acaba por sair de vista e a simulação acaba. Portanto, seria interessante que o carrinho ficasse o maior tempo possível no centro da tela. Uma estratégia que poderia ser utilizada para melhorar o modelo de *reward* seria somar pontos quando ele estivesse próximo do centro e subtrair quando ele estivesse distante, sendo que a quantidade de pontos a retirar seria maior conforme a distancia do centro da tela aumentasse.

1.4 Resultados

Como podemos observar na tabela 1, o melhor resultado que obtivemos foi com a estratégia *double dueling DQN*, com os hiperparâmetros da tabela 2. Ela precisou somente de 126 épocas para atingir o *score* médio de 195, média essa que foi calculada entre os scores das últimas 100 épocas.

O gráfico que ilustra a evolução do score médio com o passar dos episódios na figura 2. Pode-se observar também a média dos 100 últimos valores e o score bruto em cada episódio. Podemos perceber que o score começa baixo, e começa a ter picos cada vez maiores quando ele começa a exploração. Depois disso, o score começa a

Eps Trained	Layers	Method	LR	Exp Decay	Best Score	AVG 20	AVG 100
126	4	DoubleDuelingDQN	0.0001	0.02	200	200	195.77
128	4	DoubleDQN	0.0001	0.02	200	200	195.15
262	4	DoubleDuelingDQN	0.0001	0.004	200	200	196.41
318	4	DoubleDQN	0.0001	0.004	200	200	195.27
409	4	DoubleDuelingDQN	0.0001	0.002	200	200	195.37
449	4	DoubleDuelingDQN	0.001	0.002	200	200	196.02
472	4	DuelingDQN	1e-05	0.002	200	199.6	195.02
548	4	DoubleDQN	0.001	0.002	200	200	195.88
1723	4	DQN	0.001	0.002	200	200	196.19
2000	1	DQN	0.01	0.002	200	141.85	122.97
2000	2	DQN	1e-05	0.002	72	47	47
2000	6	DQN	0.1	0.002	83	39	39
2000	1	DuelingDQN	1	0.002	141	42.4	33.5
2000	2	DuelingDQN	0.01	0.002	80	28.71429	28.71429
2000	6	DuelingDQN	0.1	0.002	108	41	41

Table 1: Comparação das arquiteturas baseado nos métodos de treino

Technique	Double DQN
Episodes Needed	126
Learning rate	0.0001
Learning rate decay	1%
Exploration Threshold	1
Exploration Threshold Min	0.01
Exploration Decay	2%
Discount Factor(γ)	0.99

Table 2: Hiperparâmetros do nosso melhor modelo

aumentar, e o score fica quase sempre em alta, até que o score médio dos últimos 100 episódios ultrapassa 195 no episódio 126.

1.5 DQN, Double DQN e Dueling Double DQN

Como o primeiro problema era mais simples, e, portanto, mais leve computacionalmente, nos decidimos que seria mais adequado testar as diferentes técnicas nesse problema, já que o segundo demoraria muito mais para treinar todas essas diferentes variações no modelo.

Assim, ao observar os dados da tabela 1, podemos claramente perceber que as melhores estratégias são, em ordem:

$$DoubleDuelingDQN > DoubleDQN > DuelingDQN >> DQN$$

O *double dueling DQN* (*DDDQN*) se sai muito melhor que o simples *DQN* por ter uma melhor "politica de seleção" das ações anteriores. Essa simples diferença pode levar a muitas melhorias no desempenho geral do treino. É importante tomar atenção que o gráfico dos scores do *DDDQN* é muito menos ruidoso que o do *DQN*, o que garante maior estabilidade e facilita a convergência do modelo.

2 CarRacing-v0

O objetivo desse problema era fazer com que um carrinho corra sobre uma pista. Para isso, nós utilizamos a estratégia DQN.

2.1 Arquitetura

A arquitetura da rede que criamos para resolver o problema foi semelhante a que criamos para o problema anterior, com a adição de camadas de *feature extraction*. A rede final ficou então:

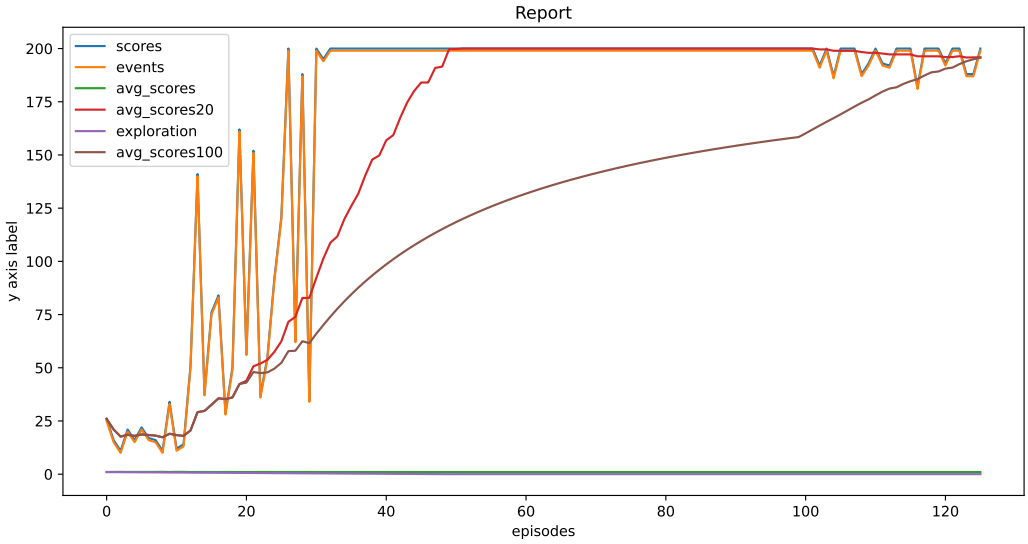

Figure 3: Gráfico que mostra a evolução do score ao longo dos episódios com treino em Double Dueling DQN.

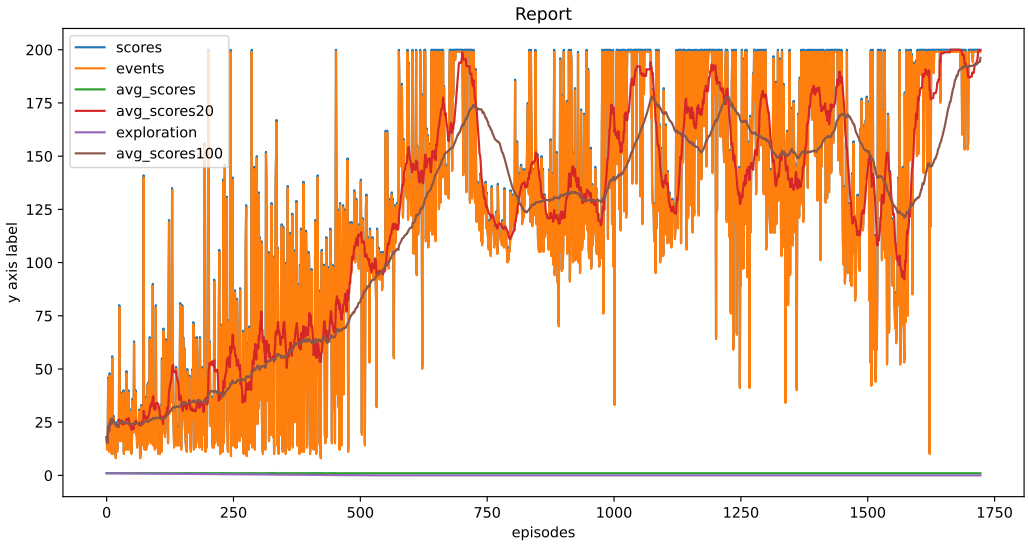

Figure 4: Gráfico que mostra a evolução do score ao longo dos episódios com treino em DQN.

2.1.1 Feature Extraction Layer

- Convolutional Layer 1, n = 9
- Convolutional Layer 2, n = 64
- Convolutional Layer 3, n = 8

Depois dessas duas camadas, e entre elas, aplicava-se a função de ativação.

2.1.2 Fully Connected Layers

- Fully Connected Layer 1, n = 73728
- Fully Connected Layer 2, n = 64
- Output Layer, n = 12

Entre as camadas *fully connected*, é inserida a função de ativação.

2.2 Reward

O modelo de *reward* presente nesse problema é muito simples, ele consiste em +1000/N para cada *tile* da pista que o carrinho percorre, em que N é o numero de tiles no trajeto, -0.1 por *frame*. Sua simplicidade tende a fazer com que a rede precise de mais tempo de treino.

Uma boa forma de melhorar esse modelo seria adicionado uma componente de penalização, somando um valor negativo ao *reward* quando o carro passasse pelo gramado. Outra alternativa que poderíamos utilizar seria adicionar uma componente de bonificação inversamente proporcional à distância ao centro do trajeto.

2.3 Resultados

Para esse exercício, fizemos três experimentos. De inicio treinamos com caminho fixo por aproximadamente 600 episódios, em seguida, utilizamos os pesos gerados para treinar em um cenário com caminhos aleatórios por mais 300 episódios. O resultado que obtivemos, no geral, foi bastante bom, mas observamos um comportamento estranho quando testamos essa rede na mesma pista que utilizamos para treiná-la inicialmente: o carrinho começa a andar e de repente para, e não anda mais até o fim da simulação. Em nossos testes, utilizamos apenas o *DQN*.

3 Respostas

Proponha um modelo novo de *reward* para o primeiro exercício. Você não precisa implementar o novo modelo, mas deve claramente comparar e destacar as vantagens e desvantagens de cada um. Se você acha que não precisa de um modelo novo, justifique.

Secção 1.3. Para o segundo exercício, verificar secção 2.2.

Compare a *performance* do algoritmo de Deep Q-Learning nos dois problemas. Compare também o tempo de treino. Analise as desvantagens de cada tipo de *state representation* das *performances* observadas.

Podemos perceber que o problema 2 demorou, no geral, muito mais tempo para treinar. Uma explicação para isso é que, como o segundo problema baseia-se em imagem para obter informações do ambiente, é necessário ter uma camada de extração de *features*, que é composta por camadas convolucionais, e que torna o processo muito mais laborioso, se comparado com o primeiro problema.

Um problema de *affordance-based state representation* é que, em aplicações reais, não temos sempre os parâmetros necessários para representar os estados, sendo em muitos casos necessário fazer a extração dos parâmetros de imagens.

Treinar por um número maior de épocas melhoraria os resultados? Explique. Analisando o gráfico 7, podemos perceber que a pontuação ainda não parece ter estabilizado, e o comportamento da curva ainda é muito ruidoso. Logo, mais episódios poderiam ser bastante benéficos ao modelo, principalmente no treino em caminho aleatório.

Experimente com os *Q-Learning hyperparameters*. Discuta o efeito destes na *performance*. Vocês devem modificar um hiperparâmetro que faça uma diferença não-trivial na performance.

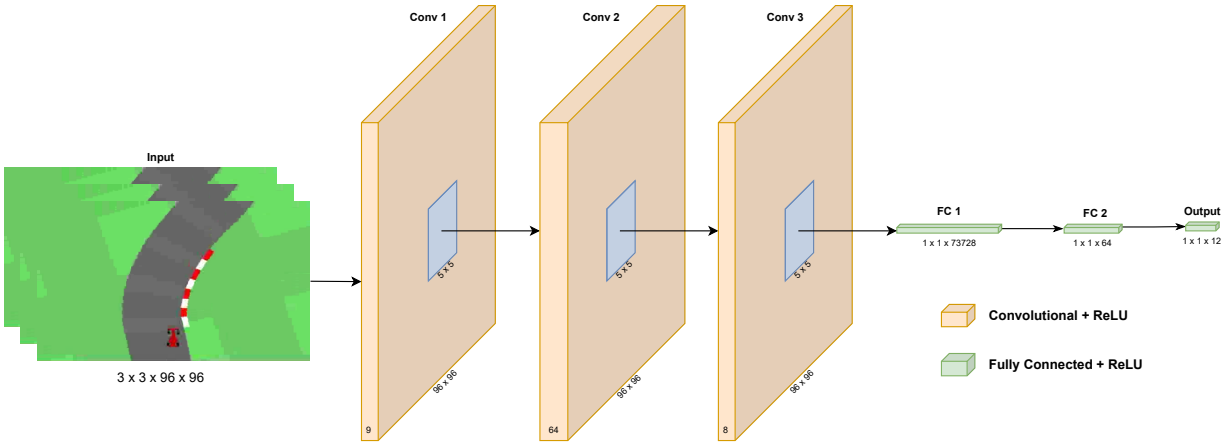

Figure 5: Caption

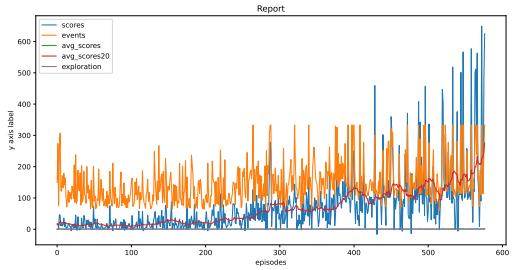

Figure 6: Desempenho do treino realizado em percurso fixo.

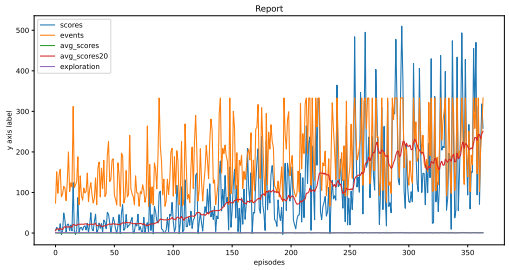

Figure 7: Desempenho do treino a partir do pesos do treino representado na fig:7 realizado em percurso aleatório.

Nós tentamos modificar a profundidade da rede do primeiro exercício, testando 1, 2, 4 e 6 *hidden layers*. Como podemos observar na figura 2, o melhor resultado que observamos foi com a rede que era composta por 4 *hidden layers*. Talvez 1 ou 2 camadas não fossem suficientes para obter um bom resultado na rede, e 6 camadas fossem demais para resolver um exercício tão simples. Outra explicação pode ser dada pelo número de camadas ser igual ao número de *state representations* (secção 1.2.1).

4 Links relevantes

Vídeo demonstrativo : LINK

Pasta com melhores resultados + vídeos : LINK

Link github : LINK

Bibliografia

[1] "Solving car racing with proximal policy optimisation." [Online]. Available: https://notanymike.github.io/Solving-CarRacing/

[2] "Deep q networks (dqn)." [Online]. Available: https://nn.labml.ai/rl/dqn/