Talita Halboth Cunha Fernandes
GRR20165399
talitahalboth@gmail.com

Trabalho Prático 4
Inteliência Artificial 2020/2021

16 de fevereiro de 2021

1 Q-Learning

O algoritmo de Q-Learning é um algoritmo de aprendizado por reforço relativamente simples de ser implementado. Neste trabalho, foi implementado o algoritmo para resolver o problema ”Frozen Lake”, como está implementado na biblioteda Gym.

1.1 Alterando o valor de ϵ e número máximo de episódios de treinamento

Foram realizados testes utilizando diferentes valores para o parâmetro ϵ e do número máximo de episódios de treinamento para se entender seus efeitos sobre a taxa de sucesso do algoritmo. A figura 1 mostra o boxplot os resultados obtidos para valores de ϵ 0.0, 0.5 e 1.0, e número máximo de episódios variando entre 1000, 5000, 10000 e 20000, rodando cada algoritmo 30 vezes, usando a versão “vanilla“ do q-learning.

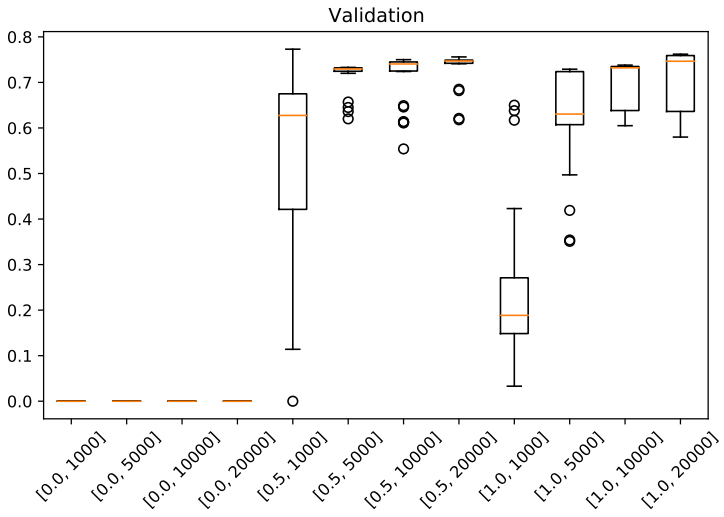

Figura 1: Box-plot da taxa de acerto do algoritmo vanilla

A primeira coisa que observamos é que, para $\epsilon = 0$, não houve acertos. Isto é de se esperar. O algoritmo de Q-learning implementado usa a política epsilon-gulosa, o que permite a exploração de outros caminhos que não o “caminho ótimo“ encontrado até o momento, baseando-se no valor de ϵ definido. Quando maior o ϵ, maiores as chances de o algoritmo explorar outros caminhos (*exploration*) ao invés de tomar o caminho ótimo (*exploitation*). No caso de um valor de $\epsilon = 0$, o algoritmo nunca tentará explorar caminhos novos. Isto significa que não haverá aprendizado, e o caminho a ser seguido será sempre determinístico. Como o caminho inicial encontrado acaba caindo em um buraco, todas as interações seguintes de teste e validação também acabarão no buraco.
A seguir podemos avaliar os resultados para $\epsilon = 0.5$. Neste caso, a taxa de acerto ficou entre 0.6 e 0.8, sendo que houve uma grande variação nos resultados das validações para treinamentos com 1000 episódios, e essa variação diminui drasticamente para treinamentos com mais episódios. Isto porque com 1000 episódios, nem sempre o algoritmo encontra uma solução boa, mas aumentando os episódios fica mais fácil encontrar melhores resultados. Porém, o aumento na taxa de acertos se torna cada vez menor com o aumento até estagnar, mas em contrapartida o tempo de execução vai se extendendo, portanto é importante limitar o número de episódios quando o tempo é um fator importante. Além disso, o algoritmo pode encontrar o melhor caminho que é capaz e não conseguir melhorar a partir daí. A figura 2 mostra a média da taxa de acerto com $\epsilon = 0.5$ para diferentes valores máximos de episódios.

Partindo para os resultados encontrados com $\epsilon = 1.0$, podemos ver que a variância nos resultados foi maior que com $\epsilon = 0.5$, mas houveram taxas relativamente altas. Muitas vezes, para problemas probabilísticos, soluções aleatórias conseguem encontrar resultados bons, pos isso vemos esse resultado.

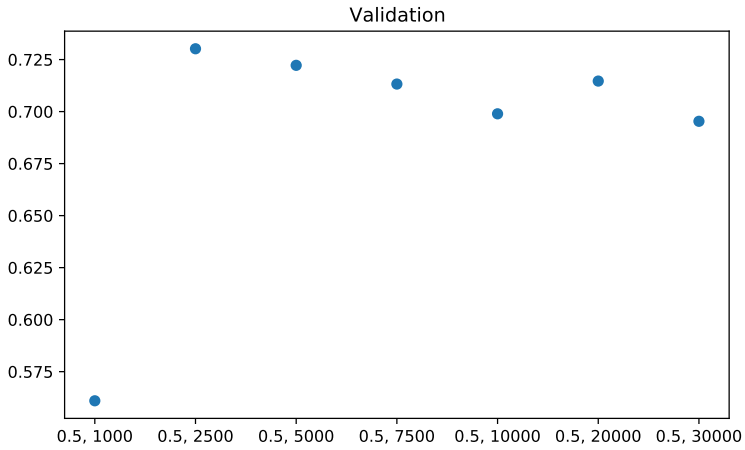

Figura 2: Médias de taxa de acerto

1.2 Testes sem fator probabilístico

Foram realizados também testes usando a opção "-ds", que cria um ambiente onde o jogador não escorrega. Os resultados encontrados podem ser vizualizados na figura 3. Novamente, para $\epsilon = 0$, o algoritmo não é capaz de encontrar nenhum caminho até o fim. Porém, em quase todos os outros testes q têm $\epsilon > 0$, o algoritmo encontrou o caminho ótimo em 100% das vezes, exceto para $\epsilon = 0.5$ e máximo de 1000 episódios de treinamento. Isto porque, em um ambiente sem a possibilidade de escorregar, tomar uma ação em um determinado estado sempre leva para o mesmo estado seguinte, então, a menos que o agente realize uma ação aleatória depois de encontrar o caminho até o fim, ele sempre será capaz de chegar no destino.

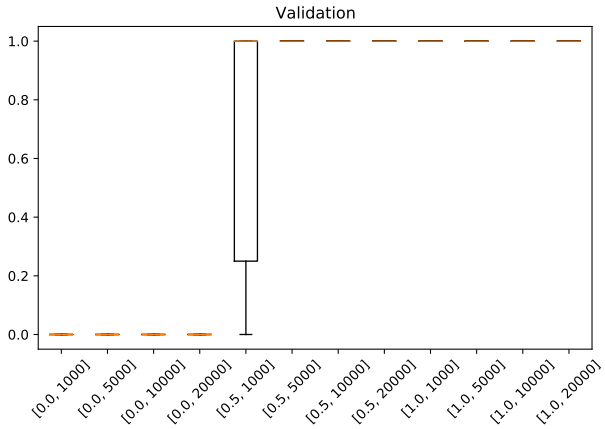

Figura 3: Caption

1.3 Análise de correlação entre taxa de acertos durante treinamento e validação

A figura 4 Mostra a relação entre a taxa de acertos durante o treinamento e durante a validação. Podemos notar que não há um valor alto de correlação entre os valores. O eixo X representa as taxas obtidas nos treinamentos, e o eixo Y representas as taxas obtidas na validação.

1.4 Otimização de Hiperparâmetros

Os testes realizados até aqui foram feitos usando valores arbitrários para as variáveis ϵ, γ e α. Porém, há formas para otimizar estes parâmetros, também chamados de "hiperparâmetros", para melhorar o resultado do algoritmo de Q-learning. Há várias formas de se fazer isso, como descrito por [1], cada uma com seus pontos positivos e negativos. Para este trabalho, foi realizada a otimização de parâmetros usando um algortimo

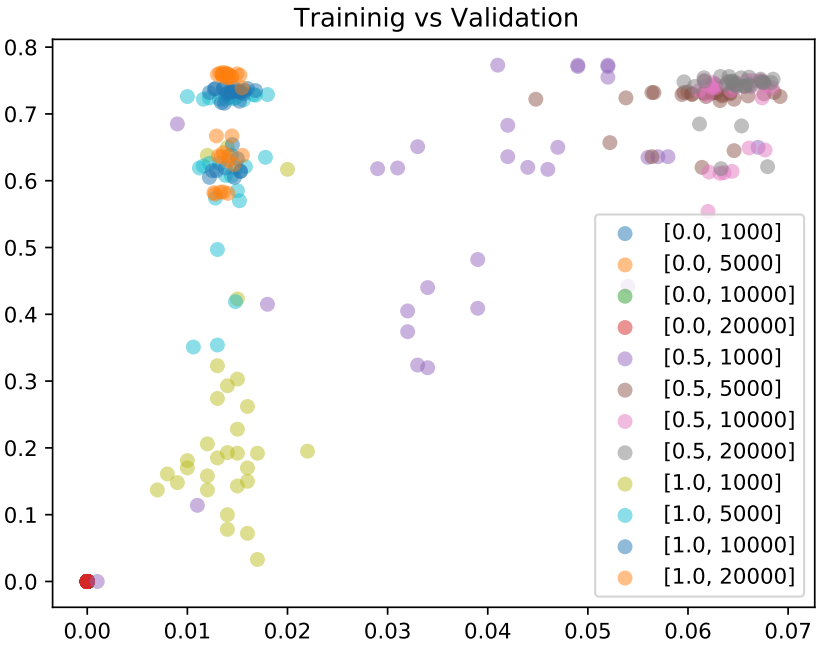

Figura 4: Gráfico de Dispersão

genético [2], [3]. Foi usada a biblioteca pyeasyga [1] que implementa um algoritmo genético genérico para realizar a otimização dos parâmetros. Esta otimização é realizada no arquivo optimizeHyperparam.py. É importante que o algoritmo de Q-learning não use sempre as mesmas seeds para inicialização dos elementos aleatórios, usando ao invés um valor diferente a cada iteração, para que se encontrem resultados diferentes a cada vez que o algoritmo roda que representam melhor os quão bom são os parâmetros escolhidos.

Referências

[1] Timo Böhm. *How to optimize Hyperparameters of Machine Learning Models*. Set. de 2018. URL: https://towardsdatascience.com/how-to-optimize-hyperparameters-of-machine-learning-models-98baec703593.

[2] Marcos del Cueto. *Genetic Algorithm to Optimize Machine Learning Hyperparameters*. Set. de 2020. URL: https://towardsdatascience.com/genetic-algorithm-to-optimize-machine-learning-hyperparameters-72bd6e2596fc.

[3] Laurits Tani et al. *Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics*. 2020. arXiv: 2011.04434 [hep-ex].

[1] https://github.com/remiomosowon/pyeasyga