Segundo Trabalho de Inteligência Artificial e Sistemas Inteligentes

Bruno Menegaz Acerbi

Departamento de Informatica – Universidade Federal do Espírito Santo (UFES)

bruno.m.acerbi@edu.ufes.br

Abstract

Nesse estudo foi realizada a implementação e a comparação entre diferentes técnicas de aprendizado de máquina por reforço para a construção de agentes inteligentes capazes de aprenderem a jogar o jogo do Dino da Google.

Keywords: Aprendizado de máquina, Metaheurística, Agente Inteligente, Árvore de Decisão, Otimização por enxame de partículas.

1. Introdução

Com o intuito de construir um agente inteligente capaz de jogar o jogo do Dino, utilizou-se um método de classificação acompanhando de um método de busca heurística, tais modelos escolhidos foram árvore de decisão[1] em conjunto a uma otimização por enxame de partículas [2].

Ao final da implementação, foram realizados testes e comparações entre as performances obtidas com o agente aqui produzido e de outra aplicação disponibilizada pelo professor. Para a visualização de tal disputa, produziu-se um diagrama de caixa e uma tabela com a contabilização dos dados pareados dos testes de hipótese entre os pares de cada método, com os resultados do *teste t* e resultado do teste não paramétrico de *wilcoxon*. As funções utilizadas nos cálculos dessas métricas estão disponíveis na biblioteca *scipy.stats* [3].

2. Descrição do Classificador

Para essa implementação, como já destacado anteriormente, o método de classificação escolhido foi a árvore de decisão[1]. Esse classificador se baseia em uma

estrutura de árvore para tomar decisões e classificar dados, comportando um conjunto de regras de decisão que ajudam a determinar a classe ou categoria de uma determinada instância de dados, no nosso caso, retornar a ação que o dinossauro deve tomar para ultrapassar um determinado obstáculo.

A construção da árvore mostrou-se uma tarefa bem simples, visto que poucos ramos bastaram para que houvesse resultados satisfatórios. A implementação pode ser visualizada na imagem 1.

```
def keySelector(self, distance, obHeight, speed, obType, nextObDistance, nextObHeight,nextObType):
    if distance <= up_metric*speed and distance >= down_metric*speed:
        if isinstance(obType, Bird) and obHeight < 83:
            return "K_UP"
        elif not isinstance(obType, Bird):
            return "K_UP"


    return "K_DOWN"
```

Figure 1: Árvore de Decisão

Nota-se a presença de um ramo principal que irá determinar se a distância atual do dinossauro para o próximo obstáculo é menor do que o limite de pulo e maior do que o limite para se abaixar, caso contrário o classificador sempre retorna a posição agachada visto que não há vantagem entre o estado em pé para o agachado. Caso os requisitos sejam satisfeitos, o próximo passo consiste na verificação do tipo de obstáculo, no qual realiza-se uma restrição da ação de pulo para as instâncias de obstáculos distintas aos pássaros que voam em médio e alta altura, visto que para esses casos o dinossauro deve se abaixar.

Por fim, destaca-se que os limites de distâncias, expressados nas variáveis globais *up_metric* e *down_metric* são ajustados pela metaheurística que será discutida futuramente.

3. Descrição da Meta Heurística

Dando prosseguimento, realizou-se uma implementação de uma otimização por enxame de partículas (PSO)[2] como metaheurística. Tal abordagem faz uso de uma busca em um espaço multidimensional para encontrar o melhor ajuste de solução para o problema em questão.

Como destacado anteriormente, a busca foi realizada sob métricas que ditam os limites de pulo e de agachamento do dinossauro, dessa forma cada partícula assume um valor distinto para tais variáveis e, a medida que rodadas de treino são realizadas, o conjunto de partículas irá se aproximar dos valores que obtiveram os melhores resultados registrados.

Vale destacar que, com base nos treinos e testes implementados ao longo do desenvolvimento do trabalho, notou-se que esse método de busca, para essa aplicação, é extremamente sensível a inicialização, ou seja, caso a primeira atribuição de valores às partículas seja muito desfavorável, dificilmente o agente irá conseguir convergir para resultados positivos.

4. Resultados

A princípio, realizou-se o treinamento do agente inteligente para o ajuste das métricas, devido a sensibilidade destacada anteriormente, os intervalos de inicialização das partículas restringiram-se de [**5, 10**] para a *down_metric* e [**20,30**] para a *up_metric*. Foram criadas *20* partículas distintas que realizaram *30* rodadas.

A partir da imagem2, nota-se que o modelo convergiu com, aproximadamente, *15* iterações e uma pontuação próxima à *3100*, a partir daí os resultados mantiveram-se similares, com pequenas variações positivas ou negativas.

Com o melhor ajuste obtido, inicia-se as rodadas de teste, onde outras *30* rodadas são realizadas, porém apenas a partícula de melhor desempenho é executada. As pontuações obtidas podem ser visualizadas no diagrama de caixas3, no qual também exibe-se os resultados registrados pelo agente inteligente do professor em contraste.

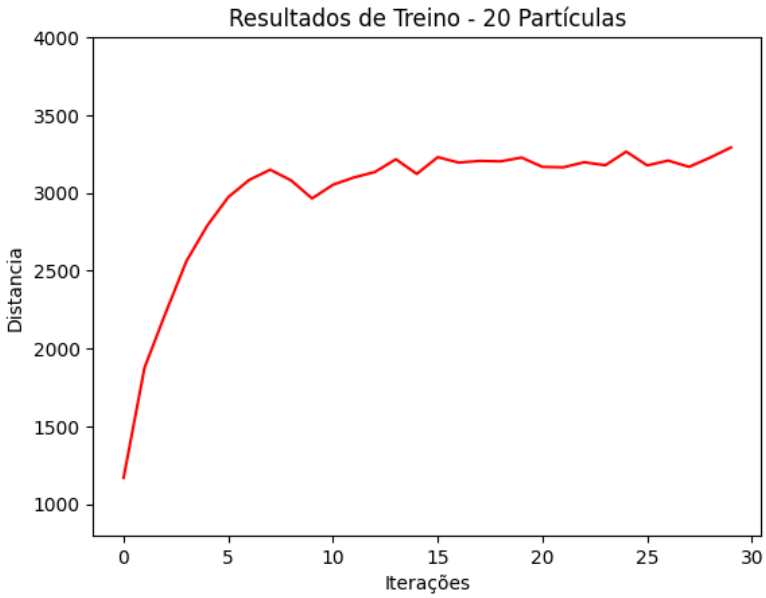

Figure 2: Treinamento do Agente

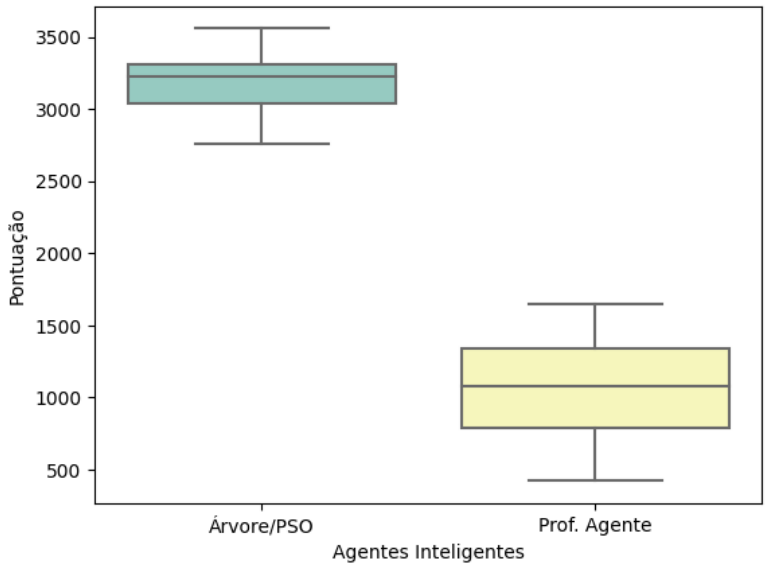

Figure 3: Teste e Comparação dos Agentes

Segue-se as pontuações registradas para cada agente inteligente, em conjunto a uma tabela1 com as médias e desvios padrões registrados.

- **Árvore/PSO - Pontuações por Round**: 1214.0, 759.5, 1164.25, 977.25, 1201.0, 930.0, 1427.75, 799.5, 1006.25, 783.5, 728.5, 419.25, 1389.5, 730.0, 1306.25, 675.5, 1359.5, 1000.25, 1284.5, 1350.0, 751.0, 1418.75, 1276.5, 1645.75, 860.0, 745.5, 1426.25, 783.5, 1149.75, 1482.25

- **Prof. Agente - Pontuações por Round**: 3301.83 3556.92 2913.42 3053.50 2815.67 3008.25 2762.17 3036.00 3235.92 3313.17 3267.83 3341.50 3342.42 3276.25 3086.75 3207.75 3117.75 3163.92 2964.67 3345.92 3310.00 3336.92 2983.83 3262.92 3336.42 3254.67 3173.92 3264.17 2968.50 3180.42

	Árvore/PSO	**Prof. Agente**
Média	3172.78	1068.18
Desvio Padrão	177.63	304.04

Table 1: Tabela com o pareamento entre os modelos

Por fim, vemos os testes de hipótese *teste t* e teste de *wilcoxon* pareados entre os modelos na tabela2, onde a matriz triangular superior apresenta os resultados do *teste t* e a matriz triangular inferior os resultado de teste de *wilcoxon*. Vale destacar que para se rejeitar a hipótese nula, foi escolhido um nível de significância de pelo menos 95%, dessa forma, caso $p < 0.05$, descarta-se a equivalência dos métodos. Tais valores estão destacados em negrito.

Árvore/PSO	**0.0**
0.0	Prof. Agente

Table 2: Tabela com o pareamento entre os modelos

5. Conclusões

5.1. Análise geral dos resultados

Com os resultados coletados, destaca-se que a produção desse modelo foi satisfatória, visto que o agente produzido neste estudo conseguiu convergir para um resultado que demonstrou-se superior aos testes realizados com o agente disponibilizado pelo professor.

Nota-se que as médias de resultados diferem em mais de *2000* pontos, um valor bem significativo. Outro ponto a se destacar é a facilidade de treino do modelo produzido, visto que, com uma inicialização ideal, bastaram 15 rodadas para que o agente convergisse a resultados positivos.

Por fim, analisando os testes de hipótese *teste t* e teste de *wilcoxon* pareados, conclui-se a disparidade clara entre os modelos aqui comparados.

5.2. Contribuições do Trabalho

Pode-se destacar que as principais contribuições deste trabalho acabaram sendo voltadas ao desenvolvimento pessoal, com o estudo e a implementação de um método de classificação que faz uso uma de busca heurística para a otimização de um agente capaz de jogar um jogo, tendo assim uma visualização prática e interessante de como um algoritmo pode aprender a realizar uma tarefa.

5.3. Melhorias e trabalhos futuros

Para trabalhos futuros, a implementação de uma árvore de decisão mais complexa, a qual faz uso de mais variáveis, seria interessante para possíveis resultados mais expressivos.

References

[1] C. Kingsford, S. L. Salzberg, What are decision trees?, Nature biotechnology 26 (9) (2008) 1011–1013.

[2] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948 vol.4. doi:10.1109/ICNN.1995.488968.

[3] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van

Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272. doi:10.1038/s41592-019-0686-2.