Speaker Listener Multi Agent Reinforcement Learning Task

Bryan Monteiro

Nicholas Costa

Sofia Monteiro

SPEAKER LISTENER MULTI AGENT REINFORCEMENT LEARNING

1. Problema inicial

O problema de aprendizado por reforço multi agente, *speaker-listener* é um problema de comunicação entre dois agentes em que o agente *speaker* envia sinais para um agente *listener* que deve interpretar os sinais e tomar uma decisão de movimentação de forma a diminuir a distância entre os dois agentes. O cerne do problema está em que os dois agentes devem encontrar uma "língua comum" para que a comunicação seja efetiva.

Tomando como base um script python que implementa a lógica estrutural do treinamento de um modelo para resolver o problema de *speaker-listener*, foram implementadas diferentes abordagens com o intuito de aumentar o desempenho final do modelo ao final do treinamento.

2. Modelo Baseline

Nas configurações iniciais, o modelo apresentou um desempenho longe do satisfatório, sendo bastante inconstante entre as iterações e com um score máximo ruim.

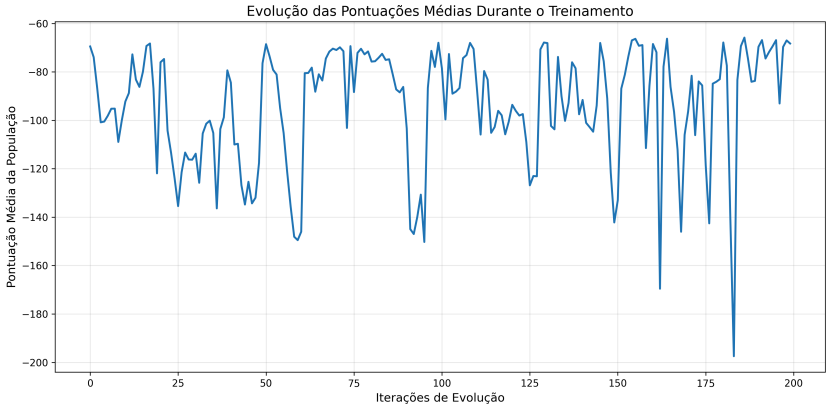

Figura 1. Desempenho do modelo sem modificações.

3. Primeiras alterações

O primeiro conjunto de mudanças afetou alguns hiper parâmetros iniciais do modelo, sendo eles:

`INIT_HP`

`population_size: 4 -> 6` : Aumentar tamanho da população para variabilidade maior no treinamento.

`gamma: 0.95 -> 0.99` : Aumentar penalização de erros para evitar mínimos locais.

`MUTATIONS`

`architecture: 0.2 -> 0.1` : Diminuir as mutações de arquitetura, com o intuito de reduzir a instabilidade no treinamento.

`new_layer_prob: 0.2 -> 0.1` : Diminuir mutações de novas camadas, para permitir especialização do modelo, sem grandes mudanças estruturais.

`rl_hp: 0.2 -> 0.4` : Aumentar probabilidade de mudanças nos valores da rede neural, permitindo ajustes mais precisos para o problema.

`TRAINING_LOOP:`

`evo_steps: 10.000 -> 50.000` : Aumentar a quantidade de passos para uma evolução, fazendo com que o modelo tenha mais tempo para realmente aprender, sem ser atualizado antes de uma melhora na performance

3.1. **Resultado.**

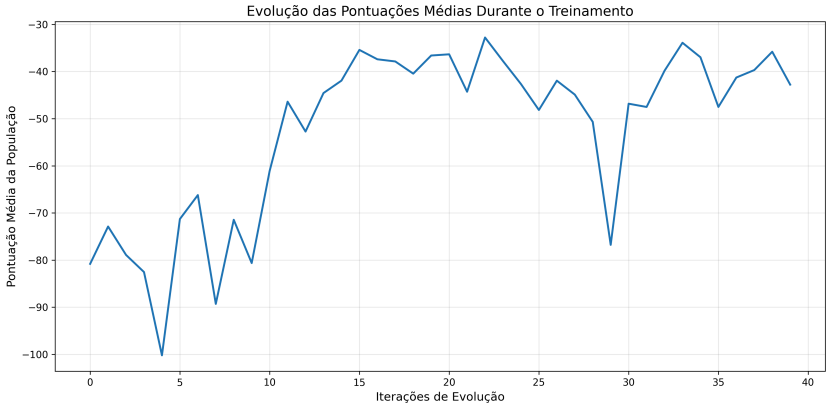

Figura 2. Desempenho do modelo após modificações.

O modelo conseguiu uma melhora significativa em comparação ao modelo inicial, se mantendo a maior parte do tempo com score médio acima de −60.

4. Novas alterações

Mudanças mais refinadas foram feitas na segunda iteração, sendo as principais a preservação do agente de elite numa população e o fine tuning de mais hiper parâmetros:

`INIT_HP`

`population_size: 6 -> 4` : Como essa mudança não apresentou impacto muito positivo e aumentou o tempo de treinamento consideravelmente, esse parâmetro voltou ao padrão original.

`batch_size: 128 -> 1024` : Aumentar a estabilidade do treinamento considerando mais memórias de experiência, evitando exemplos ruins ao acaso.

`learn_step: 100 -> 128 :` Outra mudança com o intuito de manter uma maior estabilidade, ativando o aprendizado de forma mais espaçada e permitindo mais experiências entre eles.

`policy_freq: 2 -> 3 :` Mudança relacionada ao "juiz" do modelo, *critic agent* que avaliará a performance de outro agente, aumentar esse parâmetro dá mais tempo para o juíz ter uma forma correta de julgamento

`LEARN_LR_PARAMS` parâmetros aprendidos pelo modelo durante treinamento, foram adicionados ao processo os parâmetros `tau` e `policy_frequency` para não ter escolhas "arbitrárias" de parâmetros, mas ter um processo de treinamento que avalia qual opção é a melhor dentre as disponíveis.

Este bloco de código foi alterado:

```
elite, pop = tournament.select(pop)
pop = mutations.mutation(pop)
pop[0] = elite
```

O objetivo dessa mudança é preservar o melhor agente de cada iteração, impedindo que ele sofra mutações. Isso garante que o melhor desempenho da população nunca piore. Um novo agente só se tornará o "elite" se, após o processo de mutação, ele conseguir um resultado superior ao do elite anterior.

4.1. **Resultado.**

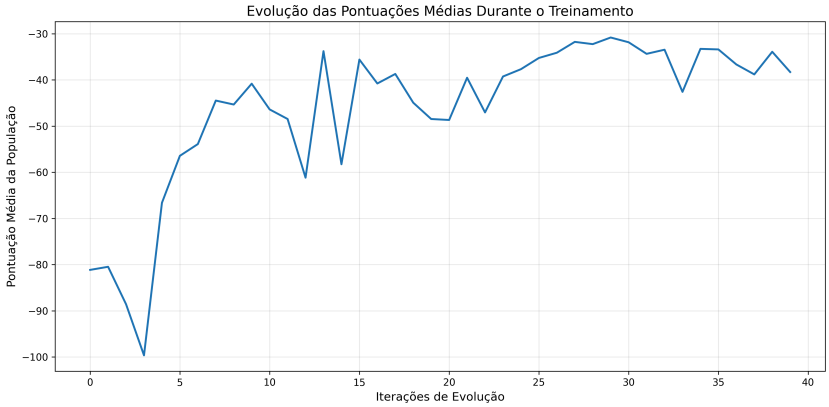

FIGURA 3. Desempenho do modelo após modificações.

O modelo se apresentou mais estável e manteve a maior parte das iterações com score médio muito próximo de -30, indicando que as alterações feitas foram eficientes para aumentar a estabilidade durante o treinamento.

5. PENÚLTIMAS ALTERAÇÕES

`HyperparameterConfig:` Mudanças no range em que os hiper parâmetros `lr_actor` e `lr_critic` podem atuar, indo de 10^{-4} para 10^{-7}, permitindo um ajuste mais preciso desses parâmetros durante o treinamento.

`max_steps: 2.000.000 -> 5.000.000 :` Aumentando o número de passos totais do modelo, permitimos que ele consiga um desempenho melhor ao final do treinamento.

5.1. **Resultado.**

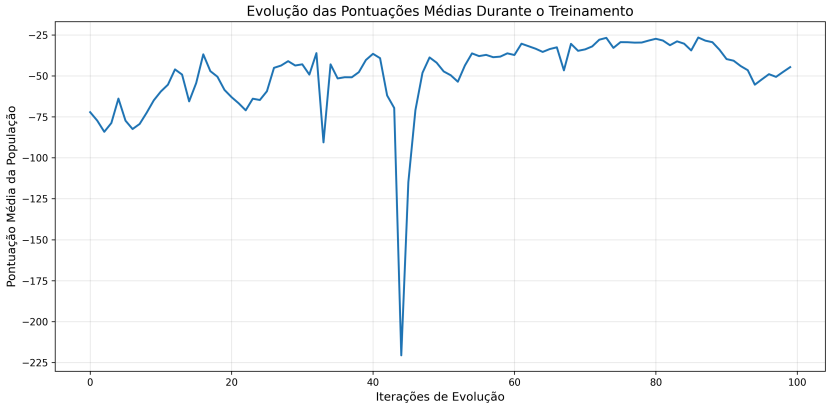

FIGURA 4. Desempenho do modelo após modificações.

Aumentando o número de iterações e a capacidade do modelo de fazer ajustes mais finos aos parâmetros, o modelo conseguiu um desempenho excelente, se aproximando de scores como -25 e ficando acima de -50 por praticamente todo o treinamento.

6. Últimas alterações

A última mudança foi apenas uma mudança na lógica de plot de erros do modelo. Anteriormente o gráfico mostrava o desempenho médio de toda a população, o que levava a interpretações erradas do treinamento, já que poderíamos ter um agente de elite muito bem treinado, mas agentes na população que não obtiveram bons resultados e isso colocava o desempenho da iteração muito mais baixo do que deveria ser. O modelo final entrega o melhor agente do treinamento e, por conta disso, vamos mostrar apenas o histórico de treinamento do elite de cada iteração.

6.1. **Resultado.**

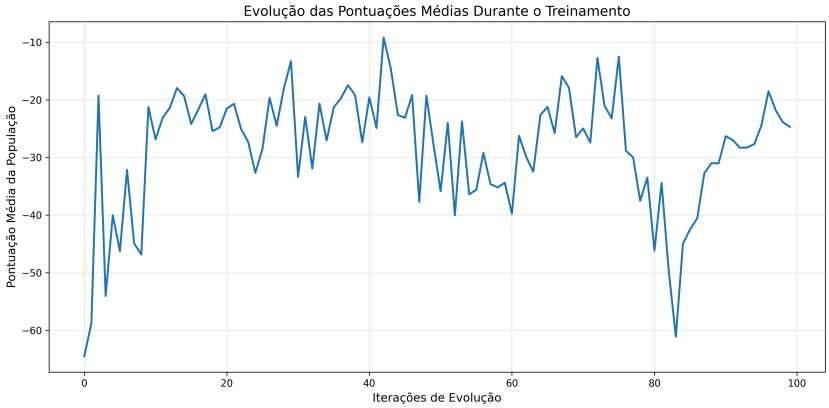

FIGURA 5. Desempenho do modelo após modificações.

A figura mostra que o desempenho do agente final é realmente muito superior ao desempenho médio da população, tendo pontuações menores do que -10 e se mantendo próximo de -20 durante o treinamento. O gráfico pode parecer mais instável, porém é apenas por que nos gráficos anteriores, por ser plotado uma média da população, era muito menos provável de existirem desempenhos drasticamente diferentes.

7. Possíveis modificações

Outras abordagens que poderiam ser exploradas para tentar melhorar o desempenho são:

Utilizar o algoritmo MADDPG poderia trazer resultados diferentes, porém, segundo a documentação da biblioteca AgileRL, MATD3 é uma forma mais estável do MADDPG, então sua utilização foi descartada.

Utilizar o algoritmo IPPO, que tem uma abordagem on-policy, sem depender necessariamente do experience replay e de memórias passadas, poderia trazer novos resultados possivelmente melhores, mas exigiriam uma nova e completa implementação do problema dado, não foi implementado por poder estar muito fora do escopo da atividade.

Alterar o ruído de Ornstein Uhlenbeck para o ruído Gaussiano, porém, após ler sobre os dois tipos de ruído, o primeiro pareceu ter uma proposta mais alinhada com o problema inicial, já que ele tem as mesmas propriedades do ruido Gaussiano, mas com uma variância controlada ao longo do tempo, que converge para 0.

Implementar Redes neurais mais profundas, com mais camadas e mais neurônios. Essa abordagem não foi explorada principalmente por conta do tempo de execução do código, aumentar a complexidade tornaria impraticável o treinamento do modelo com máquinas locais.

8. Conclusão

O trabalho apresentou um desafio interessante que foi abordado de forma iterativa com verificações e validações das modificações por meio do plot de erro histórico, cada modificação gerou discussões que definiram como elas deveriam ser feitas. Após mudanças principais, como alteração de hiper parâmetros, a preservação do melhor agente, uma correção no plot de erros e mais parâmetros dentro do treinamento, o modelo atingiu um resultado satisfatório de desempenho.

Escola de Matemática Aplicada, Fundação Getulio Vargas, Rio de Janeiro