

FGV EMAp

Al  x J  nio Maia, Jo  o Pedro Jer  nimo e Thal  s Ambrosim Falqueto

# Relat  rio de Desempenho em Ambiente Multi-Agente Speaker-Listener

Aprendizado por Refor  o

Rio de Janeiro

2025

# 1 Ambiente e Problema Inicial

O ambiente utilizado no projeto foi o ambiente Multi-Agent Speaker-Listener. Nessa configuração, dois agentes são colocados de forma a se comunicarem um com o outro para atingir um objetivo em comum. O **Speaker** possui visão direta do objetivo, porém, não pode interagir diretamente com o ambiente de forma a chegar no alvo. Já o **Listener** deve receber as informações do **Speaker** para tentar sair de um ponto **A** para um ponto **B**.

O problema passado foi um código já implementado envolvendo o ambiente Speaker-Listener utilizando o algoritmo MATD3. No entanto, o código passado estava com um desempenho relativamente baixo.

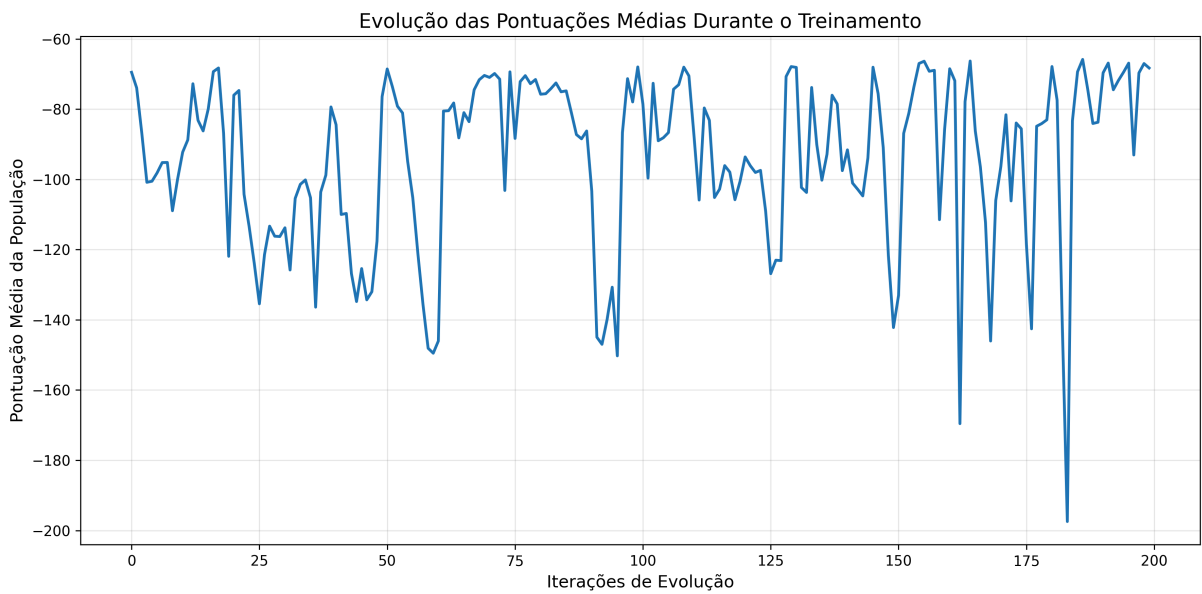


Figura 1: Desempenho do Algoritmo Padrão

## 2 Alterações

A primeira ideia para alteração do modelo foi a implementação de um algoritmo diferente, também fornecido pela biblioteca AgileRL. O novo algoritmo é o MADDPG (Multi-Agent Deep Deterministic Policy Gradient), porém, após a leitura da documentação do AgileRL e do paper sobre o MADDPG, a sua implementação foi descartada, tendo em vista que o algoritmo implementado por padrão (MATD3) é uma extensão, uma melhoria do MADDPG. Após essa decisão, o processo de melhoria do algoritmo foi feito com base na alteração dos parâmetros do algoritmo já implementado.

### 2.1 Primeira Alteração

As primeiras alterações foram feitas aumentando a quantidade de agentes escolhidos aleatoriamente dentro da população para o torneio. Depois, foram feitas alterações nos valores mínimos e máximo dos parâmetros de `learning_rate` tanto do **Listener** quanto do **Speaker**, permitindo que os ajustes de parâmetros fossem mais refinados. Porém, o resultado não foi muito satisfatório.

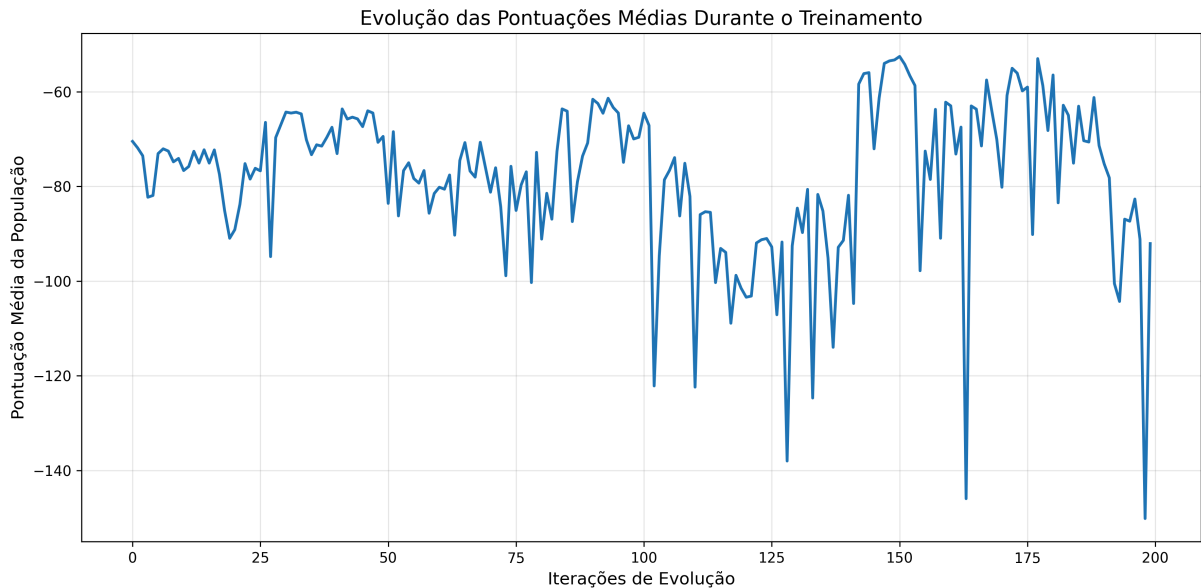


Figura 2: Desempenho do algoritmo após alterações fracas nos parâmetros

### 2.2 Alterações Posteriores

Outras alterações posteriores também foram feitas na tentativa de melhora do modelo. Algumas alterações feitas foram:

- Aumentar a dimensão das camadas das Redes Neurais dos agentes
- Aumentar a quantidade de experiências armazenadas (`batch_size`)
- Diminuir as probabilidades de exploração
- Aumentar a significância de recompensas a curto prazo ( $\text{Gamma}$ )
- Aumentar o refinamento dos passos realizados na atualização dos parâmetros ( $\text{Tau}$ )

porém, essas alterações geraram melhorias, mas não foram significativas. Foi notado um problema comum em todos os gráficos: a alta variância de recompensas de um passo para o outro.

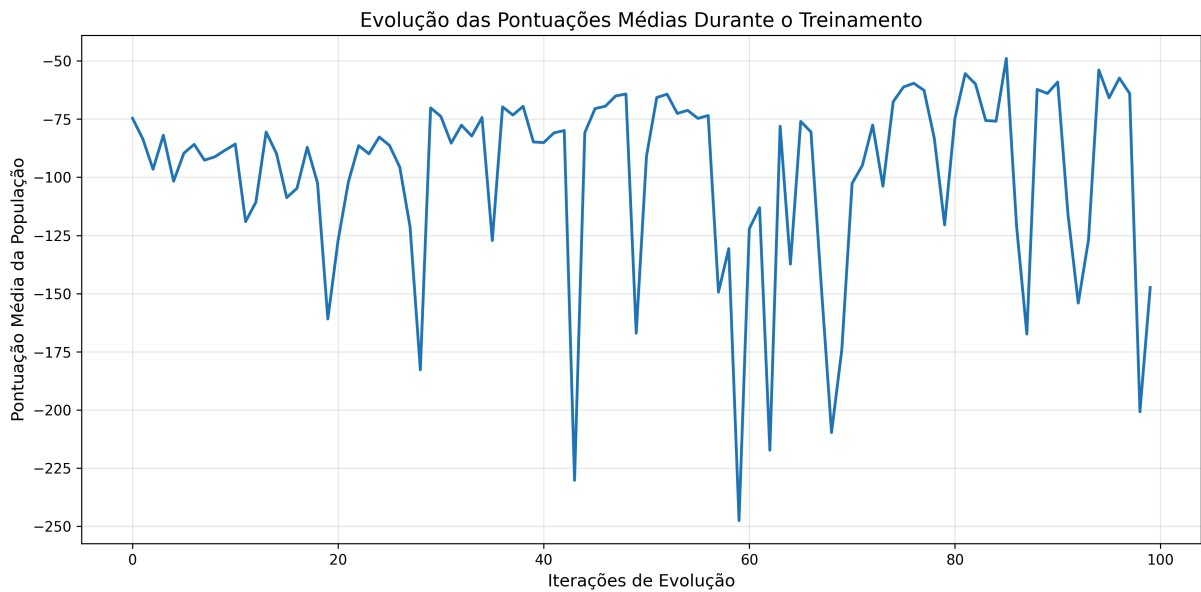


Figura 3: Pontuação após as alterações citadas anteriormente

O que estaria causando essa alta variância de pontuações entre uma geração e outra?

## 2.3 Alterações Finais

O fluxo do algoritmo faz com que, após os melhores agentes serem selecionados, todos os agentes são **mutados**, e a hipótese levantada foi que esse passo foi o principal causador da grande variância de pontuações no modelo. Então o seguinte trecho do algoritmo foi modificado,

Antes

py

```
...
268 elite,pop=tournament.select(pop)
269 pop = mutations.mutation(pop)
```

Depois

py

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

pois o agente de elite estava sendo modificado também. Após essa alteração, uma melhora significativa foi observada, porém, a variação alta continuou.

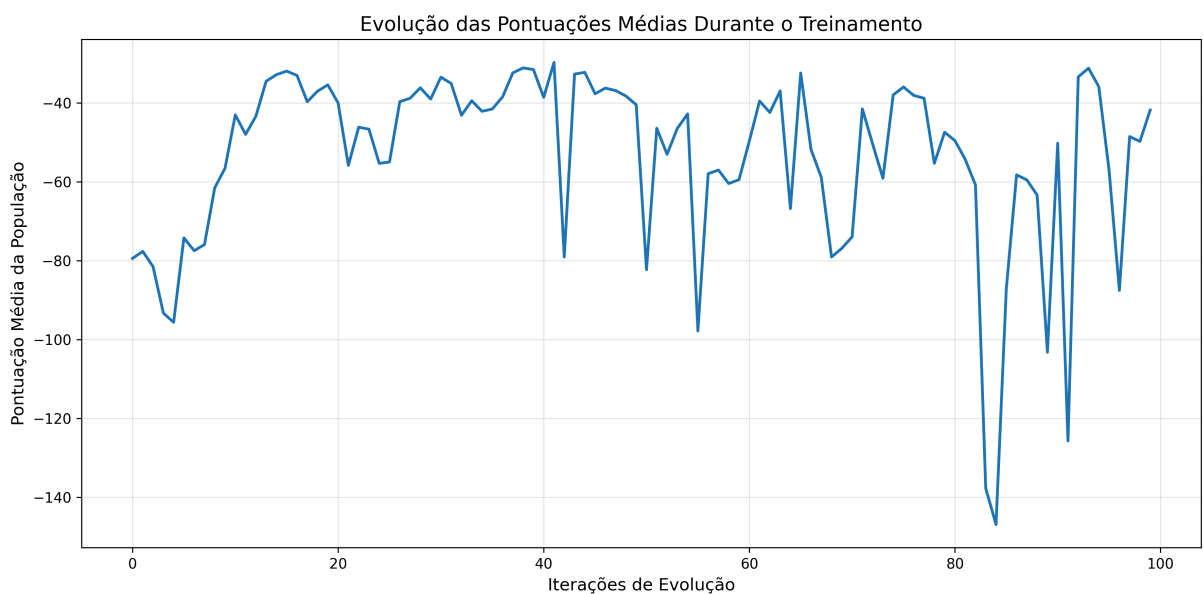


Figura 4: Desempenho após a preservação do agente de elite de cada torneio

Foi levantado, então, que o modelo poderia não estar aprendendo da forma que deveria, principalmente quanto aos picos variados, então adicionamos tanto Tau quanto Policy\_Freq aos parâmetros aprendidos pelo modelo:

...

py

```
81 hp_config = HyperparameterConfig(
82     tau=RLParameter(min=1e-4, max=1e-2),
83     policy_freq=RLParameter(min=1, max=5, dtype=int),
84     ...
```

e obtivemos uma melhoria significativa também, porém, a alta variância ainda permanecia, mesmo que em menor quantidade.

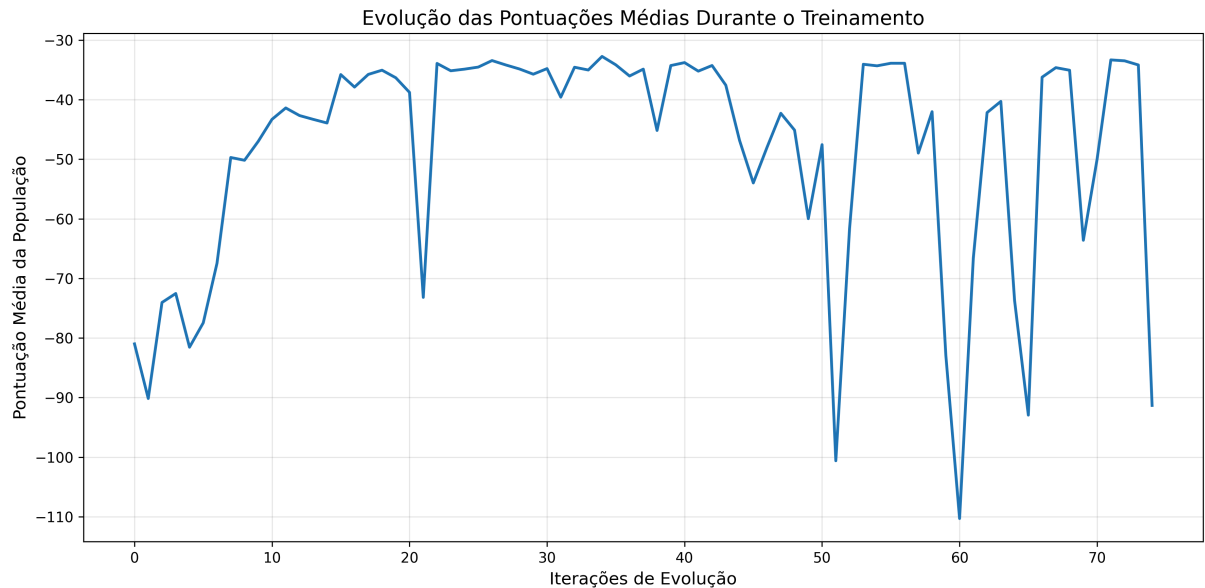


Figura 5: Desempenho após permitir que o modelo aprendesse os parâmetros tau e policy\_freq

Então a hipótese de que as **mutações** estariam afetando o modelo foi levantada, de forma que elas estavam ocorrendo com muita frequência, o que deixa o modelo instável (principalmente por conta da alta quantidade de iterações que são rodadas). Então foi feita uma alteração nas probabilidades de mutações:

```
...                                     ...
123 mutations = Mutations(
124     no_mutation=0.99, # Probability of no mutation
125     architecture=0.1, # Probability of architecture mutation
126     new_layer_prob=0.05, # Probability of new layer mutation
127     parameters=0.05, # Probability of parameter mutation
128     activation=0, # Probability of activation function mutation
129     rl_hp=0, # Probability of RL hyperparameter mutation
130     mutation_sd=0.01, # Mutation strength
131     rand_seed=1,
132     device=device,
133 )
```

e um resultado **excelente** foi obtido.

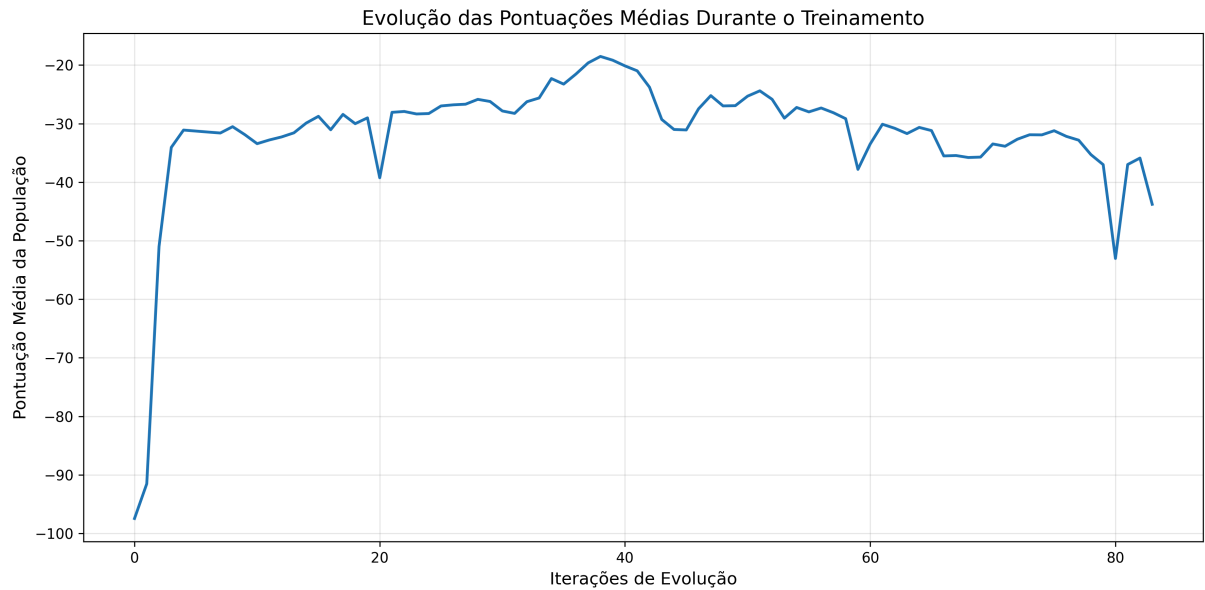


Figura 6: Desempenho após a diminuição significativa nas probabilidades de mutação

### 3 Conclusão

O trabalho apresentou um desafio muito interessante, sendo possível desafiar a interpretação dos modelos através dos gráficos de desempenho e das etapas do algoritmo, de forma a mostrar como um mesmo algoritmo pode apresentar diversos desempenhos dependendo dos fatores envolvidos. Esse projeto apresenta a importância da avaliação rigorosa de um modelo sobre um problema, tendo em vista que vários fatores podem o levar a ter uma performance ruim, mas isso não significa que ele não pode gerar bons resultados para o problema em que está sendo aplicado.