Zeroth Order Policy Search Methods for Global Optimization Problems: An Experimental Study

Moésio W. da Silva Filho[1] **, Gabriel A. Barbosa**[1] **, Péricles B. C. Miranda**[1]**, André C. A. Nascimento**[1] **, Rafael Ferreira Mello**[1]

[1] Departamento de Computação
Universidade Federal Rural de Pernambuco (UFRPE) – Recife, PE

{moesio.wenceslau,gabriel.augusto,pericles.miranda}@ufrpe.br

Abstract. *Policy Search (PS) methods have been used to learn optimization algorithms, obtaining encouraging results. In this work, we used PS methods to learn optimization algorithms for global optimization problems, considering a little-studied scenario: high-dimensional functions and the optimization algorithms do not have access to the derivatives of the function to be optimized. The results indicate that, despite the difficulties, the learned optimization algorithms have promising performance in the studied scenario.*

Resumo. *Os métodos Policy Search (PS) vem sendo utilizados nos últimos anos para se aprender, automaticamente, algoritmos de otimização, obtendo resultados animadores. Neste trabalho, consideramos métodos PS para aprender algoritmos de otimização para problemas de otimização global, considerando um cenário pouco estudado: funções de alta dimensionalidade e os algoritmos de otimização não possuem acesso às derivadas da função a ser otimizada. Os resultados apontam, que apesar das dificuldades, os algoritmos de otimização aprendidos têm um desempenho promissor no cenário estudado.*

1. Introdução

A tarefa de encontrar o ótimo $\mathbf{x}^*$ de uma função f com relação a algum domínio $\mathcal{X}$ é um problema recorrente, conhecido como otimização global (quando $\mathbf{x}^*$ tem que ser um ótimo global) [Snyman and Wilke 2018]. Dessa forma, diferentes algoritmos de otimização foram desenvolvidos ao longo dos anos, considerando as peculiaridades dos problemas e/ou classe de problemas que visam resolver [Andrychowicz et al. 2016, Faury and Vasile 2018]. Entretanto, o processo de desenvolvimento é árduo e, por vezes, repetitivo, necessitando da participação de especialistas humanos [Li and Malik 2016] e grande cautela na escolha dos seus parâmetros [Faury and Vasile 2018]. Assim, a automatização do design de algoritmos de otimização é atrativa, sendo um tópico popular e recorrente [Faury and Vasile 2018, Zhang et al. 2020], onde diferentes métodos são utilizados, incluindo, os métodos de Aprendizagem por Reforço (do inglês, *Reinforcement Learning* - RL) [Andrychowicz et al. 2016].

Nesse contexto, os métodos de RL são utilizados para se aprender os algoritmos de otimização [Chen et al. 2017], representados pela política aprendida. Em específico, os métodos *Policy Search* (PS) vem sendo amplamente utilizados para esse problema e obtendo resultados positivos [Li and Malik 2016, Li and Malik 2017,

Faury and Vasile 2018, Zhang et al. 2020]. Todavia, existem alguns pontos que precisam ser investigados quanto ao uso desses métodos. Um deles é a capacidade de se aprender algoritmos de otimização que não dependem das derivadas da função a ser otimizada [Golovin et al. 2019], visto que a maioria dos trabalhos realizados com tais métodos aprendem algoritmos de otimização que necessitam da primeira e/ou segunda derivada (i. e., algoritmos de otimização de primeira e segunda ordem, respectivamente), como, por exemplo, em [Li and Malik 2016], [Li and Malik 2017] e [Zhang et al. 2020]. Outro aspecto que precisa ser melhor investigado é a dimensionalidade dos problemas de otimização testados, uma vez que, quando considerados problemas de otimização global, a dimensionalidade não é superior a 10 em muitos trabalhos, como, por exemplo, em [Li and Malik 2016], [Li and Malik 2017], [Faury and Vasile 2018] e [Zhang et al. 2020].

Dessa forma, neste trabalho, o objetivo é aprender algoritmos de otimização global que não necessitem das derivadas da função a ser otimizada (i. e., de ordem zero), e que possam ser aplicados para problemas com dimensionalidade superior a 10. Para isso, o problema de otimização global foi modelado matematicamente como um Processo de Decisão de Markov (MDP) e utilizamos 4 métodos PS populares (REINFORCE, TD3, SAC e PPO) para aprender algoritmos de otimização para 8 funções de otimização global distintas com alta dimensionalidade ($d = 30$) (provenientes da competição do *Evolutionary Multi-task Optimization IEEE WCCI 2020*), totalizando 32 algoritmos aprendidos. O desempenho dos algoritmos aprendidos foram comparados entre si, e também com 2 algoritmos clássicos de otimização de primeira ordem: *Gradient Descent* (GD) e *Nesterov's Accelerated Gradient* (NAG). Os resultados apontam que, apesar das dificuldades, a maioria dos métodos PS considerados foram capazes de aprender algoritmos de otimização com performance similar, ou melhor, que os algoritmos clássicos nos problemas considerados.

O restante do artigo é organizado da seguinte forma: A Seção 2 introduz os conceitos básicos. A Seção 3 discute os trabalhos relacionados sobre o uso dos métodos Policy Search para se aprender algoritmos de otimização. As seções 4 e 5 apresentam, respectivamente, o arranjo experimental e os resultados. Por último, a Seção 6 resume o artigo e discute trabalhos futuros.

2. Background

Nessa seção, serão abordados os conceitos básicos de Aprendizagem por Reforço e a classificação dos diferentes métodos utilizados nessa área.

2.1. Aprendizagem por Reforço

RL é uma das áreas de Aprendizado de Máquina responsável por lidar com o problema de tomada de decisões sequenciais, onde, normalmente, um agente aprende a se comportar visando melhorar uma *recompensa* numérica obtida ao interagir com o ambiente [Sutton and Barto 2018, François-Lavet et al. 2018]. Nesse contexto, o problema de RL pode ser formalizado com o uso de uma interface agente-ambiente [Sutton and Barto 2018]. O *agente* é o aprendiz, sendo responsável pela tomada de ações. O *ambiente* é o ente com o qual o agente interage. Essa interface pode ser representada matematicamente como um Processo de Decisão de Markov (MDP).

Um MDP é descrito como um processo de controle estocástico em tempo discreto, que pode ser definido pela tupla $\langle \mathcal{S}, \mathcal{A}, R, p, \gamma \rangle$ onde: $\mathcal{S}$ é conjunto de estados; $\mathcal{A}$ é o conjunto de ações; $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R} \subset \mathbb{R}$ é a função recompensa e $\mathcal{R}$ é o conjunto de possíveis recompensas; $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,\ 1]$ é a função que descreve a dinâmica do MDP; $\gamma \in [0, 1]$ é um fator de desconto utilizado para o cálculo do retorno esperado.

2.2. Métodos de Aprendizagem por Reforço

Existem diversos métodos clássicos para resolver os problemas de RL que podem ser formulados como um MDP finito, como, por exemplo, *Programação Dinâmica*, *Métodos de Monte Carlo* e *Aprendizagem por Diferença Temporal* [Sutton and Barto 2018]. Dessa forma, é importante classificar os diferentes métodos existentes. A Figura 1 contém o sistema de classificação adotado neste trabalho. As categorias foram escolhidas seguindo as sugeridas em [Zhang and Yu 2020], [Sutton and Barto 2018] e [Arulkumaran et al. 2017].

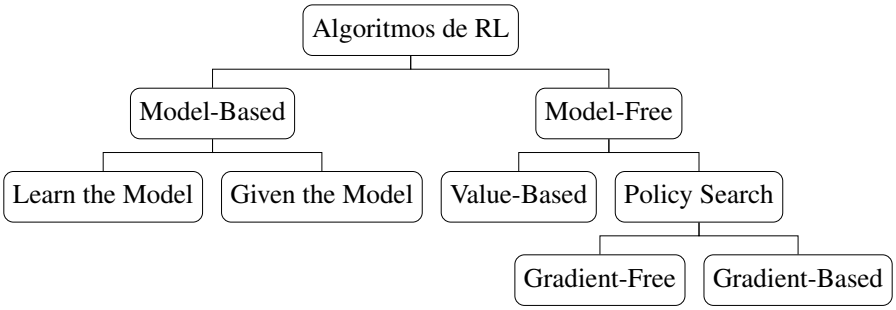


Figura 1. Classificações dos Métodos de RL.

De forma resumida, os métodos *Model-Based* aprendem e/ou fazem uso de alguma função que descreve as dinâmicas do MDP. Diferentemente, os métodos *Model-Free* não fazem uso e nem aprendem tais funções. Os métodos *Model-Free* podem, ainda, ser classificados em *Value-Based*, onde o agente aprende alguma *value function* que vai ser utilizada para a escolha das ações, e *Policy Search*, onde o agente aprende diretamente uma política responsável pela escolha das ações nos diferentes estados.

Nesse trabalho consideramos os métodos *Policy Search*, onde uma política parametrizada é aprendida e aprimorada buscando maximizar o retorno esperado [Arulkumaran et al. 2017, Zhang and Yu 2020], para aprender algoritmos de otimização global. Essa família de métodos foi escolhida, principalmente, pela aplicabilidade em diversos problemas, inclusive quando o espaço de ações é contínuo, e pelos bons resultados obtidos nos últimos anos em diferentes domínios, como, por exemplo, na Robótica [Arulkumaran et al. 2017] e para se aprender algoritmos de otimização [Li and Malik 2016, Faury and Vasile 2018, Zhang et al. 2020].

3. Trabalhos Relacionados

Nessa seção, serão aprofundados alguns dos trabalhos que utilizaram os métodos Policy Search no contexto de otimização global e aprendizado de algoritmos de otimização.

Em [Williams and Peng 1991] ocorre uma das primeiras investigações sobre o uso de RL como método de otimização de funções. Nele são consideradas variantes

do algoritmo REINFORCE para a otimização de 6 problemas distintos: 4 funções matemáticas, que devem ser maximizadas, e 2 problemas de otimização combinatória. As 4 funções matemáticas consideradas mapeavam um hipercubo n-dimensional em números reais, $f : \{0, 1\}^n \rightarrow \mathbb{R}$, onde $n = 20$ foi adotado. Os resultados obtidos indicaram uma boa performance para os problemas considerados. Todavia, diversos problemas de otimização atuais consideram o domínio como um subconjunto do $\mathbb{R}^d$, sendo necessário novas investigações envolvendo o REINFORCE, e suas variantes, nesses cenários.

Em [Li and Malik 2016] os métodos de RL são utilizados para aprender algoritmos de otimização de forma automática, partindo da observação que a execução de um algoritmo de otimização pode ser vista como a execução de um MDP, onde a política representa o algoritmo de otimização. O método utilizado foi o Guided Policy Search (GPS), os experimentos consideraram a minimização do erro de modelos de regressão logística, regressão linear robusta e classificação, considerando o $\mathbb{R}^3$ e $\mathbb{R}^4$ como espaços de busca. Os resultados apontaram que o algoritmo aprendido performava melhor, em termos de convergência e valor objetivo final, que algoritmos de otimização clássicos, como GD e L-BFGS. Entretanto, não são considerados problemas com maior dimensionalidade e outros algoritmos de RL além do GPS, fazendo-se necessário mais investigações nesse sentido.

Em [Li and Malik 2017] foram feitas mudanças com relação ao framework proposto em [Li and Malik 2016], e novos testes foram realizados considerando a minimização do erro de redes neurais totalmente conectadas para classificação de imagens. As mudanças introduziram o uso de um Processo de Decisão de Markov Parcialmente Observável (POMDP), estratégias para a redução da dimensionalidade do espaço de estados e ações, levando em conta peculiaridades do problema em questão, e uma versão adaptada do GPS, chamada de Convolutional GPS. Os resultados demonstraram que o algoritmo aprendido performou melhor que outros algoritmos populares (Adam, AdaGrad, RMSProp, e outros) e conseguiu generalizar para outros conjuntos de dados. Todavia, visto que o único problema considerado é o da minização do erro de redes neurais, fazem-se necessárias novas pesquisas considerando estratégias para a redução da dimensionalidade e o uso de POMDPs em outros problemas de otimização.

Mais recentemente, em [Faury and Vasile 2018] é proposto um novo algoritmo, chamado *Rover Descent*, para se aprender algoritmos de otimização de ordem zero. O Rover Descent aprende um algoritmo de otimização por meio da aprendizagem de 3 sistemas independentes: Um preditor para o ângulo de movimentação, aprendido com o uso de *imitation learning*; Um preditor para a magnitude (i. e., taxa de aprendizagem), aprendido com o Deterministic Policy Gradient (DPG); e um preditor da superfície da função objetiva, também aprendido com o DPG. Os experimentos são realizados com o algoritmo de otimização aprendido para funções matemáticas de 2 dimensões e redes neurais para classificação binária com 10, 20 e 50 dimensões, obtendo bons resultados quando comparados com outros algoritmos de otimização (GD, NAG, e outros). Apesar de considerar o problema de se aprender um algoritmo de otimização de ordem zero, o Rover Descent não é testado em funções matemáticas com maiores dimensões, necessitando de novos estudos nessas condições.

Em [Zhang et al. 2020] é considerado aprender um algoritmo de otimização, de ordem dois, com duas fases. A primeira fase é de minimização, onde um algoritmo de

otimização é utilizado para encontrar um mínimo local. Já a segunda é de *escape*, onde um algoritmo é utilizado para escolher uma nova posição que permita que a próxima fase minimização escape do mínimo local. O algoritmo para a fase de escape é aprendido utilizando um método Policy Gradient, enquanto o algoritmo para a fase de minimização é aprendido utilizando um método model-based. O algoritmo de otimização aprendido, constituído pelos algoritmos das duas fases, é testado para funções com diferentes dimensionalidades entre 2 e 10 e para a otimização de redes neurais, obtendo resultados melhores que outros algoritmos clássicos (GD e BFGS).

Diferentemente dos trabalhos anteriores, o presente trabalho aborda sobre a aprendizagem de algoritmos de otimização de ordem zero, por meio de métodos Policy Search, para a otimização global de funções matemáticas com dimensionalidade superior às consideradas anteriormente.

4. Materiais e Métodos

Nessa seção será apresentado o arranjo experimental. Primeiro, o problema de otimização global foi modelado como um MDP. Em seguida, listamos as funções de benchmark consideradas nos testes e os algoritmos utilizados. Por último, discutimos a implementação e execução dos experimentos.

4.1. Modelagem

Para utilizar os algoritmos de RL para otimização de funções, adequamos o problema de otimização global como um MDP seguindo o framework proposto por [Li and Malik 2016]. Assim, definimos o MDP: o conjunto de estados $\mathcal{S} \subseteq \mathbb{R}^d$ como o domínio da função em questão; o conjunto de ações $\mathcal{A} \subseteq \mathbb{R}^d$ como o conjunto de possíveis passos; a função recompensa $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R} \subset \mathbb{R}$ como $R(S_t, A_t, S_{t+1}) = -f(S_{t+1})$ onde f é a função objetiva em questão; o fator de desconto $\gamma \in [0, 1]$ foi adotado como 0.99 para todos algoritmos; A função que descreve as dinâmicas do MDP não é utilizada pelos agentes, assim não será descrita aqui.

4.2. Funções de Benchmark

As 8 funções de benchmark consideradas são conhecidas pela literatura e são provenientes da competição *Evolutionary Multi-task Optimization IEEE WCCI 2020*[1]. A implementação de cada uma das funções está disponível no site da competição. As funções escolhidas são classificadas em: unimodal (F_1—F_4) e multimodal (F_5—F_8). A Tabela 1 possui detalhes sobre cada uma delas. Adotamos $d = 30$ para todos os testes e comparamos os diferentes algoritmos aprendidos utilizando o mesmo arranjo experimental.

4.3. Algoritmos e Hiperparâmetros

Para os testes, selecionamos os algoritmos REINFORCE [Sutton and Barto 2018], Proximal Policy Optimization (PPO) [Schulman et al. 2017], Twin Delayed DDPG (TD3) [Fujimoto et al. 2018] e Soft Actor-Critic (SAC) [Haarnoja et al. 2018]. As escolhas se

[1] `http://www.bdsc.site/websites/MTO_competition_2020/MTO_Competition_WCCI_2020.html`

Tabela 1. Funções adotadas para os testes, $F(x^*)$ é a solução ótima.

Tipo	No.	Função	Espaço de Busca	$F(x^*)$
Unimodal	1	Sphere	$x \in [-5.12, 5.12]^d$	0.0
	2	Rosenbrock	$x \in [-5, 10]^d$	0.0
	3	Sum Squares	$x \in [-10, 10]^d$	0.0
	4	Rotated Hyper-Ellipsoid	$x \in [-10, 10]^d$	0.0
Multimodal	5	Ackley	$x \in [-32.768, 32.768]^d$	0.0
	6	Levy	$x \in [-10, 10]^d$	0.0
	7	Griewank	$x \in [-10, 10]^d$	0.0
	8	Rastrigin	$x \in [-5.12, 5.12]^d$	0.0

basearam na popularidade, aplicabilidade em problemas de diferentes domínios, e facilidade de implementação desses algoritmos.

A Tabela 2 contém os valores dos principais hiperparâmetros de cada algoritmo. As configurações gerais adotadas foram: 2000 episódios de treino; 250 interações por episódio; Batch de 256 amostras (quando aplicável); e redes neurais com duas camadas escondidas (256 unidades em cada). As configurações foram escolhidas com base nas adotadas pelos trabalhos que propuseram esses algoritmos [Sutton and Barto 2018, Schulman et al. 2017, Fujimoto et al. 2018, Haarnoja et al. 2018], não sendo realizados ajustes finos, visando manter as configurações similares para todos eles. Para mais detalhes, acessar o repositório desta pesquisa disponibilizado na Subseção 4.4.

Além de serem comparados entre si, os algoritmos de otimização aprendidos pelos métodos PS foram comparados com dois baselines, o Gradient Descent (GD) e o Nesterov's Accelerated Gradient (NAG). Os parâmetros do GD e NAG foram otimizados para cada uma das funções. Em específico, consideramos as possíveis taxas de aprendizagem como $\eta \in \{0.1, 0.01, 0.001, 0.0001\}$ e os possíveis *momentum* como $\upsilon \in \{0.5, 0.8, 0.9\}$, sendo realizada uma busca exaustiva para selecionar as configurações que obtiveram os melhores resultados em 20 execuções distintas.

Tabela 2. Hiperparâmetros utilizados.

Hiperparâmetro	REINFORCE	SAC	TD3	PPO
α	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
β	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
γ	0.99	0.99	0.99	0.99
τ	—	$5 \cdot 10^{-3}$	$5 \cdot 10^{-3}$	—
Frequência de Atualização Target Networks	—	2	2	—
Capacidade da Memória de Replay	—	10^6	10^6	—

4.4. Implementação e Execução

O repositório da presente pesquisa encontra-se em `https://github.com/rl-opt/rlopt`. O código foi feito em Python 3.8, utilizando o TensorFlow 2.5.0 e TF-Agents 0.8 (`https://www.tensorflow.org/agents`). Os agentes foram treinados utilizando o Google Colaboratory seguindo as especificações dessa seção. Detalhes da implementação:

- Todas as ações tomadas pelos agentes resultam em nova posição dentro do espaço de busca da função em questão, impondo que os agentes só explorem o espaço de busca definido.
- As funções de perda das redes neurais foram minimizadas utilizando o algoritmo ADAM, presente no TF-Agents, com as taxas de aprendizado (α e β) da Tabela 2.
- O estado inicial, $S_0 = (x_1, \ldots, x_d)$, foi inicializado aleatoriamente, seguindo uma distribuição uniforme, ao início de cada episódio e execução de teste.
- Durante o treino, todos os episódios possuíram 250 interações agente-ambiente sendo esse o único critério de parada adotado. Durante a avaliação, foram consideradas 500 interações agente-ambiente como critério de parada.
- Para os algoritmos que fazem uso de uma memória de replay (*Replay Buffer*), foram executados 20 episódios iniciais para geração das primeiras amostras.
- Para todos os algoritmos, as ações $A_t = (a_1, \ldots, a_d) \in \mathcal{A}$ foram limitadas em $A_t \in [-1, 1]^d$, visando manter um valor comum que não extrapola os espaços de busca considerados.

Como resultado do treinamento dos agentes, foram aprendidos 32 algoritmos de otimização distintos e foram realizados testes comparativos entre eles e os baselines (GD e NAG), no Google Colaboratory, com 100 execuções independentes em cada uma das funções. Dessa forma, os resultados são apresentados em termos da média dessas execuções.

5. Resultados e Discussão

Nessa seção, serão discutidos e apresentados os resultados dos testes, e, salvo quando explícito o contrário, os algoritmos de otimização aprendidos serão identificados pelo nome do método PS utilizado para aprendê-lo.

5.1. Análise das Melhores Soluções

A Tabela 3 contém a média das soluções finais, o desvio padrão dessas soluções e a quantidade média de iterações necessárias para alcançá-las. Os resultados são apresentados com precisão de 3 casas decimais e o algoritmo que obteve a melhor solução é destacado em negrito.

Pela Tabela 3, pode-se perceber que ao menos um dos algoritmos de otimização aprendidos alcançou melhores soluções finais que os baselines em 5 das 8 funções consideradas, sendo 1 unimodal (F_2) e todas multimodais (F_5–F_8). Isso se deve ao fato do GD e NAG serem algoritmos de otimização de primeira ordem, dessa forma, podem ficar presos em mínimos locais ou não conseguir alcançar os mínimos globais quando os gradientes da função não são suficientes para encontrá-lo (e. g., quando os gradientes são muito próximos de 0). Entretanto, visto que os algoritmos de otimização aprendidos não se restringem a seguir uma direção dada pelos gradientes da função, é possível superar esses problemas. Uma outra vantagem dos algoritmos de otimização aprendidos, com relação aos baselines, é que eles não necessitam de otimização nos parâmetros para serem utilizados. Entretanto, é perceptível que os métodos PS considerados tiveram dificuldades para aprender algoritmos de otimização global capazes de alcançar o mínimo global com precisão, visto que o menor erro observado (TD3 na F_7) foi 0.026. De qualquer forma, os resultados são animadores e apontam que os algoritmos aprendidos são robustos à posição inicial e conseguem escapar dos mínimos locais, diferentemente do NAG e GD.

Tabela 3. Comparação estatísticas dos agentes nas funções de benchmark.

Função	Algoritmo	Solução Final	Desvio Padrão	Iterações
F_1	GD	0.000	0.000	202
	NAG	0.000	0.000	94
	REINFORCE	253.588	40.538	0
	SAC	1.218	0.140	247
	TD3	0.559	0.010	8
	PPO	0.078	0.007	23
F_2	GD	3 598 321.500	1 043 694.500	0
	NAG	3 731 307.500	1 048 323.813	1
	REINFORCE	3 351 963.500	929 725.560	1
	SAC	5786.663	1773.539	239
	TD3	1560.544	31.156	223
	PPO	119.182	0.000	249
F_3	GD	805.385	161.740	499
	NAG	5.501	4.563	499
	REINFORCE	14 483.139	3084.537	1
	SAC	99.671	10.209	262
	TD3	17.912	2.199	44
	PPO	227.700	11.217	345
F_4	GD	802.411	165.477	499
	NAG	5.899	4.266	499
	REINFORCE	14 777.384	2624.734	1
	SAC	174.400	26.272	165
	TD3	20.268	3.130	58
	PPO	18.590	1.211	26
F_5	GD	19.500	0.193	332
	NAG	19.542	0.158	25
	REINFORCE	21.188	0.203	4
	SAC	8.550	5.090	262
	TD3	3.847	0.066	60
	PPO	20.929	0.190	17
F_6	GD	35.752	19.414	60
	NAG	63.676	18.743	495
	REINFORCE	361.244	83.365	0
	SAC	2.499	0.245	250
	TD3	3.393	0.785	11
	PPO	6.515	0.549	35
F_7	GD	1.235	0.034	499
	NAG	1.156	0.025	499
	REINFORCE	1.243	0.040	1
	SAC	1.023	0.005	266
	TD3	0.026	0.001	15
	PPO	1.223	0.045	1
F_8	GD	269.493	43.991	277
	NAG	259.683	40.345	134
	REINFORCE	531.943	59.066	1
	SAC	220.637	12.698	209
	TD3	236.925	7.749	247
	PPO	499.854	50.890	2

5.2. Análise de Convergência

A Figura 2 apresenta os gráficos de convergência média de todos os algoritmos nas funções F_1–F_8 após 100 execuções independentes com $d = 30$ e o estado inicial S_0, inicializado de forma aleatória, sendo compartilhado entre todos os algoritmos.

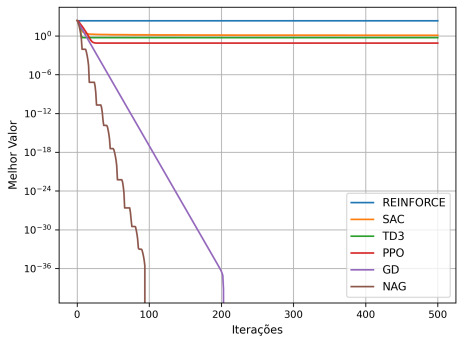
Pelos gráficos, temos que para a função F_1 o SAC, TD3 e PPO se assemelharam com o GD e NAG em um problema simples para algoritmos de otimização de primeira ordem. Já para a função F_2, o SAC, TD3 e PPO convergiram melhor que o GD e NAG, principalmente por essa função possuir gradientes muito pequenos em posições próximas do mínimo global. Para a F_3, o SAC, TD3 e PPO conseguiram encontrar melhores soluções mais rapidamente que o GD e NAG, todavia o NAG alcançou melhores soluções finais. De forma similar, na F_4 o SAC, TD3 e PPO alcançaram melhores soluções mais rapidamente, todavia o NAG alcançou melhores soluções finais. Para a F_5, o SAC e TD3 apresentaram melhor convergência, enquanto o NAG e GD não conseguiram escapar dos mínimos locais. De forma similar, para a F_6 o SAC, TD3 e PPO conseguiram encontrar melhores soluções mais rapidamente, enquanto o GD e NAG não conseguiram escapar dos mínimos locais. Para a F_7, apenas o TD3 conseguiu escapar dos mínimos locais. Por último, para a F_8 o SAC, TD3, GD e NAG convergiram de forma similar, todavia o SAC e TD3 alcançaram melhores resultados finais.

Todavia, é importante compreender as dificuldades dos algoritmos aprendidos para encontrar o mínimo global das funções consideradas. Em particular, conjecturamos que, para os algoritmos de otimização aprendidos pelo método REINFORCE, não houve uma exploração suficiente no espaço de busca da função durante o treinamento. Já para os demais métodos (TD3, SAC e PPO) conjecturamos que a modelagem do MDP pode ter dificultado substancialmente o aprendizado, visto que, para algumas funções, as recompensas adquirem valores muitos grandes ($> 10^5$). Entretanto, apesar dessas dificuldades, os gráficos de convergência apontam que foram aprendidos algoritmos de otimização competitivos ao GD e NAG, inclusive, obtendo melhores soluções em algumas funções. Portanto, tais métodos conseguiram aprender algoritmos de otimização de ordem zero em grandes dimensões.

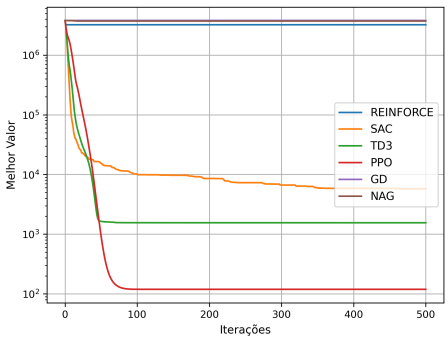
5.3. Discussão

Visto que os resultados apresentados na Subseção 5.1 e Subseção 5.2 apontam que os métodos PS considerados enfrentaram dificuldades para aprender algoritmos de otimização, abordamos sobre as possíveis causas e soluções para essas dificuldades nessa subseção.

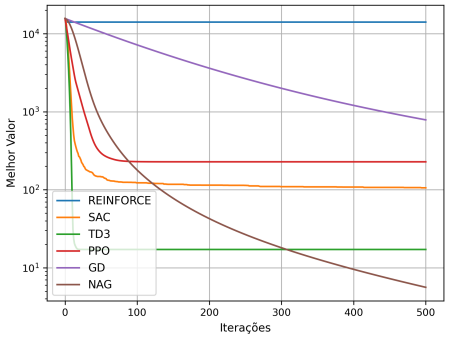
Dimensionalidade do espaço de ações: Uma das possíveis causas para essa dificuldade é a alta dimensionalidade do espaço de ações, onde cada ação possui 30 dimensões (Subseção 4.1), quando comparado com problemas clássicos de RL. Por exemplo, a dimensionalidade das ações para os ambientes presentes na biblioteca *OpenAI Gym* [Brockman et al. 2016] é inferior a 30. A Tabela 4, que compara a média das melhores soluções de diferentes algoritmos de otimização aprendidos para a função F_5 com diferentes dimensões ($d = 5$, $d = 10$, $d = 30$), corrobora com o argumento que alta dimensionalidade torna o aprendizado mais complicado. Os resultados do comparativo com as demais funções pode ser encontrado no repositório disponibilizado na Subseção 4.4, todavia são similares a Tabela 4.



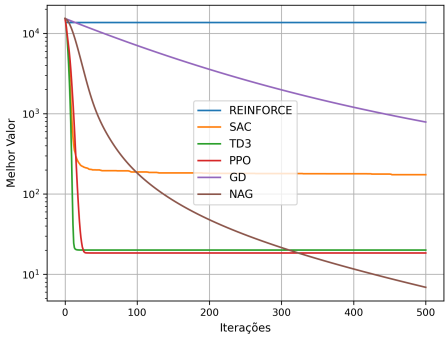
(a) F_1



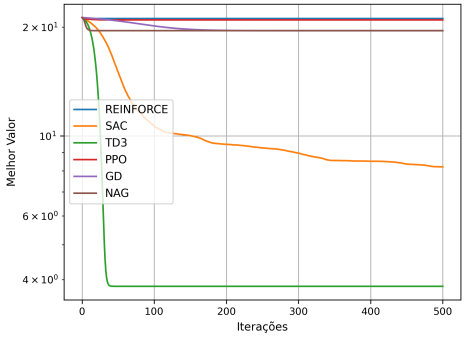
(b) F_2



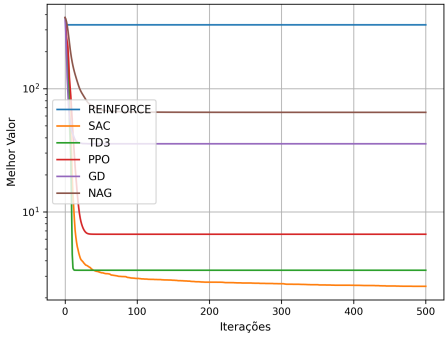
(c) F_3



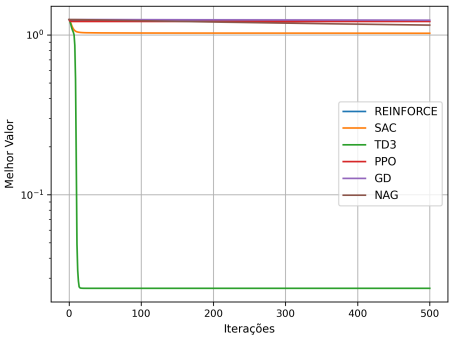
(d) F_4



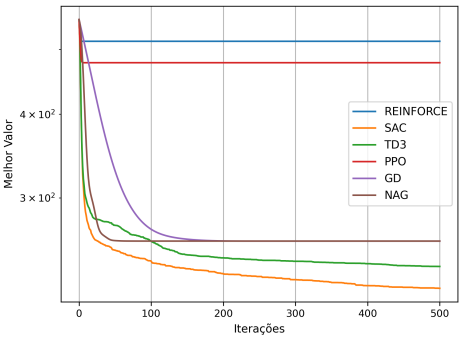
(e) F_5



(f) F_6



(g) F_7



(h) F_8

Figura 2. Gráficos de convergência dos algoritmos aprendidos.

Tabela 4. Comparação das melhores soluções na F_5 em diferentes dimensões.

Dimensões	REINFORCE	SAC	TD3	PPO
$d = 5$	20.614	0.518	0.092	0.012
$d = 10$	21.065	3.177	0.685	20.430
$d = 30$	21.188	8.550	3.847	20.929

Ações Limitadas: Outra possível causa para a dificuldade dos algoritmos PS é a limitação das ações que podem ser tomadas pelo agente em $[-1, 1]^d$, visto que remove a possibilidade de se aprender algoritmos com maior grau de liberdade. Na realidade, a única limitação das ações do agente deve ser diretamente relacionada com o estado $S_t = (x_1, \ldots, x_d)$ no qual o agente se encontra, já que o agente deveria ser capaz de tomar qualquer ação $A_t = (a_1, \ldots, a_d)$ desde que $S_t + A_t \in \mathcal{S}$. Todavia, implementar essa possibilidade requer que os métodos PS sejam modificados de forma a compelir que os agentes só tomem ações que sejam possíveis para o estado atual, logo que os algoritmos escolhidos consideram o conjunto de ações possíveis como sendo o mesmo para todos estados.

Ajuste fino dos hiperparâmetros: Uma outra possível causa é a sensibilidade dos algoritmos de RL aos hiperparâmetros escolhidos, posto que diferentes configurações podem afetar drasticamente o desempenho [François-Lavet et al. 2018]. Mesmo que os algoritmos de otimização aprendidos não necessitem de ajustes de parâmetros, os métodos de RL utilizados para aprendê-los precisam. Dessa forma, é crucial considerar outras configurações de hiperparâmetros e aferir a influência dessas configurações nos algoritmos de otimização aprendidos.

6. Conclusão e Trabalhos Futuros

Nesse trabalho foi realizado um estudo experimental com os métodos *Policy Search*, uma das família de métodos de Aprendizagem por Reforço, para aprender algoritmos de otimização de ordem zero de forma automática. Foram usadas 8 funções de benchmark conhecidas pela literatura e realizados diferentes avaliações, adotando $d = 30$, com 4 algoritmos dessa família (REINFORCE, SAC, TD3 e PPO). Os resultados mostraram que o TD3, PPO e SAC conseguiram aprender algoritmos de otimização que performaram melhor, em termos da solução final e convergência, que o GD e NAG, enquanto o REINFORCE não conseguiu. Pelas análises realizadas, foram discutidas possíveis causas para essas dificuldades: A alta dimensionalidade das ações quando comparado a outros problemas típicos de RL; a limitação das ações em $[-1, 1]^d$; e a sensibilidade dos algoritmos de RL com relação aos hiperparâmetros. Portanto, os resultados desse trabalho, somado às outras contribuições, demonstram que os métodos de RL, especialmente métodos da família Policy Search, se põem como alternativas para resolver problemas de otimização global, sendo capazes de aprender algoritmos de otimização global que não dependem das derivadas da função objetiva. Entretanto, adaptações se mostram necessárias para obter melhores resultados. Assim, iremos realizar investigações futuras considerando as mudanças propostas, outras modelagens do problema de otimização, outras funções de benchmark, e métodos PS multi-agente.

7. Agradecimentos

Os autores agradecem à FACEPE pelo apoio financeiro para a realização desse trabalho.

Referências

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2017). Learning to learn without gradient descent by gradient descent.

Faury, L. and Vasile, F. (2018). Rover descent: Learning to optimize by learning to navigate on prototypical loss surfaces. *CoRR*, abs/1801.07222.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). *An Introduction to Deep Reinforcement Learning*. Now Foundations and Trends.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th ICML 2018*, volume 80, pages 1582–1591.

Golovin, D., Karro, J., Kochanski, G., Lee, C., Song, X., and Zhang, Q. (2019). Gradientless descent: High-dimensional zeroth-order optimization. *CoRR*, abs/1911.06317.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905.

Li, K. and Malik, J. (2016). Learning to optimize. *CoRR*, abs/1606.01885.

Li, K. and Malik, J. (2017). Learning to optimize neural nets. *CoRR*, abs/1703.00441.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Snyman, J. A. and Wilke, D. N. (2018). *Practical Mathematical Optimization*. Springer International Publishing.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction (Second Edition)*. MIT Press.

Williams, R. J. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268.

Zhang, H., Sun, J., and Xu, Z. (2020). Learning to be global optimizer. *CoRR*, abs/2003.04521.

Zhang, H. and Yu, T. (2020). *Taxonomy of Reinforcement Learning Algorithms*, pages 125–133. Springer Singapore, Singapore.