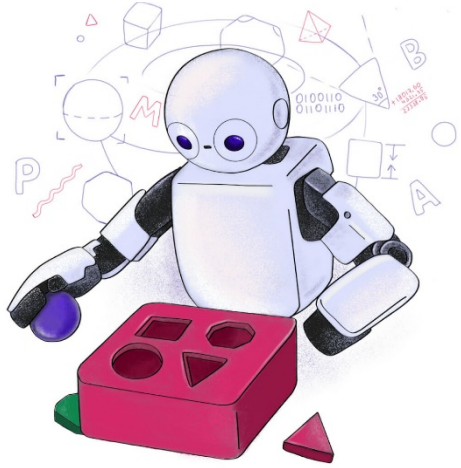
TP558 - Tópicos avançados em Machine Learning:

Algoritmo Reptile



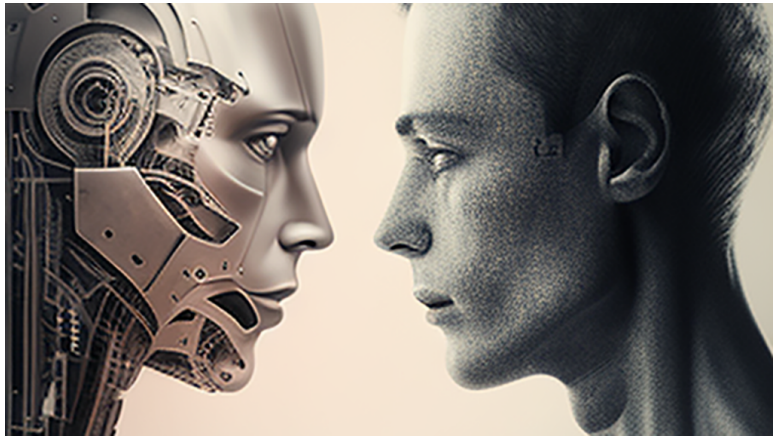
Inatel

Felipe Batista Faro Pinto
felipe.pinto@dtel.inatel.br

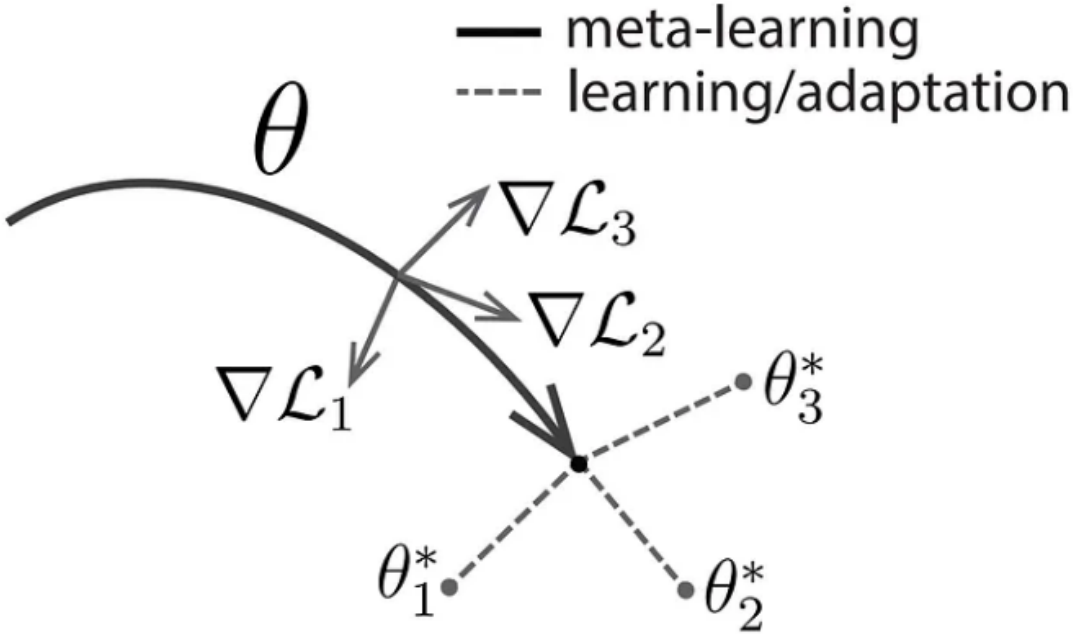
Agenda

- Introdução
- Fundamentação teórica
- Arquitetura e funcionamento
- Treinamento e otimização
- Vantagens e desvantagens
- Comparação com outro algoritmo
- Exemplo de aplicação
- Quiz
- Referências

Introdução



- Aprendizado humano vs aprendizado de máquina
- Limitação de dados
- Inferência Bayesiana
- **Meta-aprendizado**
- *Model Agnostic Meta Learning* (MAML)



Processo de treinamento e adaptação de um algoritmo de meta-aprendizado

Introdução

***Model Agnostic Meta Learning* (MAML)**

- Agnóstico ao tipo de modelo
- Desenvolvido para se adaptar rapidamente a diferentes tipos de tarefas
- Aprendizado com varias tarefas e poucas amostras
- Utiliza-se o gradiente descendente e cálculos de segunda ordem
- Existe variação como o *first-order* MAML (FOMAML)

Introdução

- Algoritmo de meta-aprendizado ***Reptile***
- Necessidade de algoritmos que aprendem rápido e se adapte a diferentes tarefas tendo um número de dados de entrada limitados para cada tarefa
- Tentar se aproximar da inferência Bayesiana
- Em essência, o *Reptile* opera treinando iterativamente um modelo em várias tarefas e atualizando os parâmetros do modelo a cada iteração de forma a minimizar a diferença entre os parâmetros finais e iniciais
- Utilização de gradiente descendente estocástico (SGD) de primeira ordem
- Aplicações de visão computacional, robótica, reconhecimento de fala, entre outras

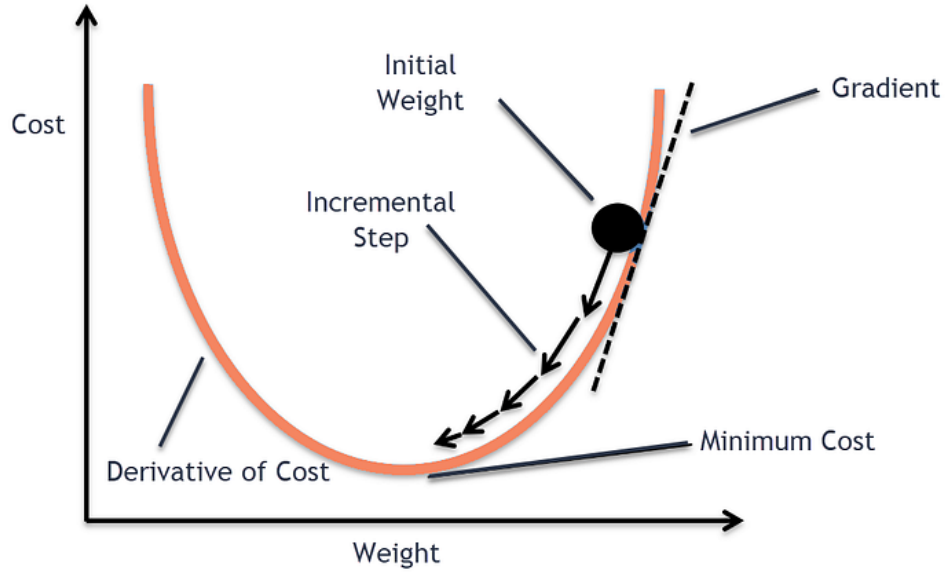
On First-Order Meta-Learning Algorithms

Alex Nichol and Joshua Achiam and John Schulman
OpenAI
{alex, jachiam, joschu}@openai.com

OpenAI

Fundamentação teórica

- Meta-aprendizado:
  - Algoritmos que são capazes de aprender a aprender
- Gradiente de primeira ordem
  - Encontra a inclinação da curva de custo
  - Computacionalmente mais eficiente que o de segunda ordem.
- Gradiente descendente estocástico (SGD)
  - Atualiza os passos do gradiente a cada amostra da tarefa



$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

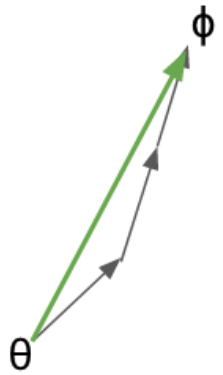
Fundamentação teórica

Por que utilizar o SGD?

- Convergência mais rápida que o gradiente descendente
- Mais eficiente em questão do uso da memória computacional
- Estes fatores são de grande importância para o propósito do algoritmo *Reptile*

Fundamentação teórica

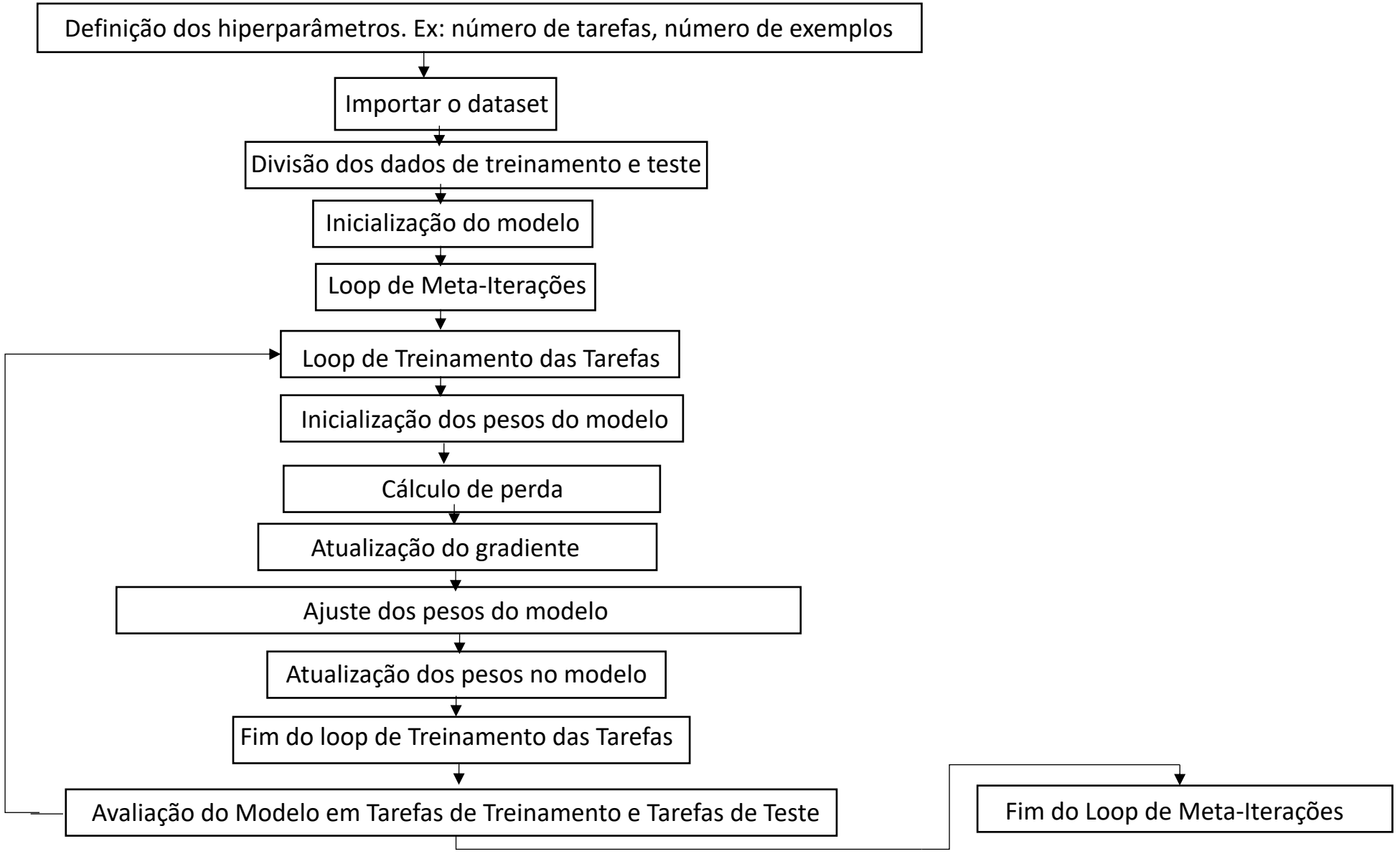
- Tarefas (*tasks*), classes, exemplos (*shots*), parâmetros iniciais e finais (depois de k *shots*).
- Vários passos no gradiente descendente estocástico
- As tarefas fazem parte do conceito de *meta-learning*
- Múltiplas tarefas fazem parte da otimização do modelo, de modo a se adaptar melhor para diversas outras tarefas



Algorithm 1 Reptile, serial version

Initialize ϕ, the vector of initial parameters
for iteration $= 1, 2, \ldots$ **do**
  Sample task τ, corresponding to loss L_τ on weight vectors W
  Compute $W = \mathrm{SGD}(L_\tau, \phi, k)$
  Update $\phi \leftarrow \phi + \epsilon(W - \phi)$
end for

Arquitetura e funcionamento



Arquitetura e funcionamento

Rede neural convolucional

- 4 camadas de convolução e normalização
- 1 camada para achatamento do vetor
- 1 camada densa

```
def conv_bn(x):
    x = layers.Conv2D(filters=64, kernel_size=3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    return layers.ReLU()(x)


inputs = layers.Input(shape=(28, 28, 1))
x = conv_bn(inputs)
x = conv_bn(x)
x = conv_bn(x)
x = conv_bn(x)
x = layers.Flatten()(x)
outputs = layers.Dense(classes, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile()
optimizer = keras.optimizers.SGD(learning_rate=learning_rate)
```

Treinamento e otimização

- Utilização do **gradiente descendente estocástico** como otimizador
- Como função de erro é utilizado a **entropia cruzada categórica esparsa**

```
def conv_bn(x):
    x = layers.Conv2D(filters=64, kernel_size=3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    return layers.ReLU()(x)


inputs = layers.Input(shape=(28, 28, 1))
x = conv_bn(inputs)
x = conv_bn(x)
x = conv_bn(x)
x = conv_bn(x)
x = layers.Flatten()(x)
outputs = layers.Dense(classes, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.compile()
optimizer = keras.optimizers.SGD(learning_rate=learning_rate)
```

```
for images, labels in mini_dataset:
    with tf.GradientTape() as tape:
        preds = model(images)
        loss = keras.losses.sparse_categorical_crossentropy(labels, preds)
    grads = tape.gradient(loss, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
new_vars = model.get_weights()
```

```
        # Train on the samples and get the resulting accuracies.
        for images, labels in train_set:
            with tf.GradientTape() as tape:
                preds = model(images)
                loss = keras.losses.sparse_categorical_crossentropy(labels, preds)
            grads = tape.gradient(loss, model.trainable_weights)
            optimizer.apply_gradients(zip(grads, model.trainable_weights))
        test_preds = model.predict(test_images, verbose=0)
        test_preds = tf.argmax(test_preds).numpy()
        num_correct = (test_preds == test_labels).sum()
        # Reset the weights after getting the evaluation accuracies.
        model.set_weights(old_vars)
        accuracies.append(num_correct / classes)
    training.append(accuracies[0])
    testing.append(accuracies[1])
    if meta_iter % 100 == 0:
        print(
            "batch %d: train=%f test=%f" % (meta_iter, accuracies[0], accuracies[1])
        )
```

Treinamento e otimização

Entropia cruzada categórica esparsa

- Usada para modelos com problemas de classificação onde as classes são mutuamente exclusivas
- Fornece a medida da diferença entre a probabilidade da predição e dos rótulos
- Neste caso, os vetores de entrada (classes) são representados como valores inteiros. Exemplo: [1], [2], [3]...
- Compatível com a função de ativação ***softmax***

Treinamento e otimização

Entropia cruzada categórica esparsa

$$L_{CE} = -\sum_{i=1}^{n} t_i \log p_i, \text{ para n classes,}$$

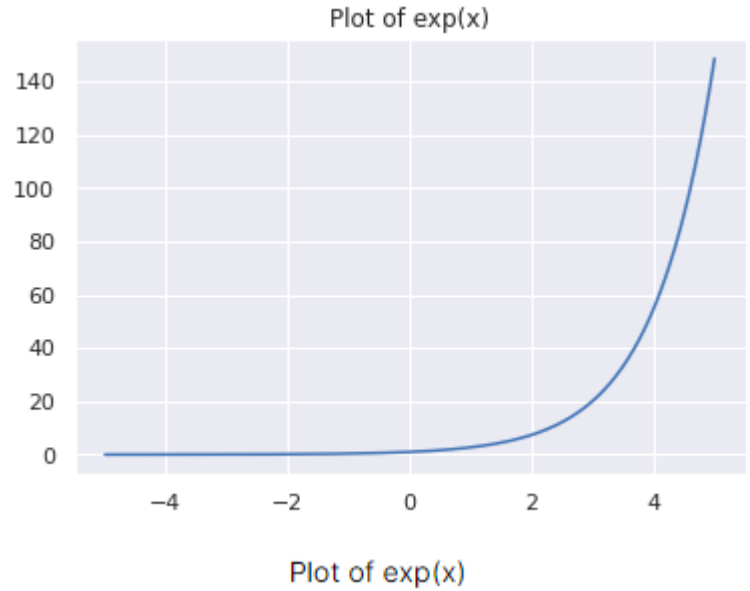
onde t_i é o rótulo verdadeiro e p_i é a probabilidade do rótulo prevista pelo modelo

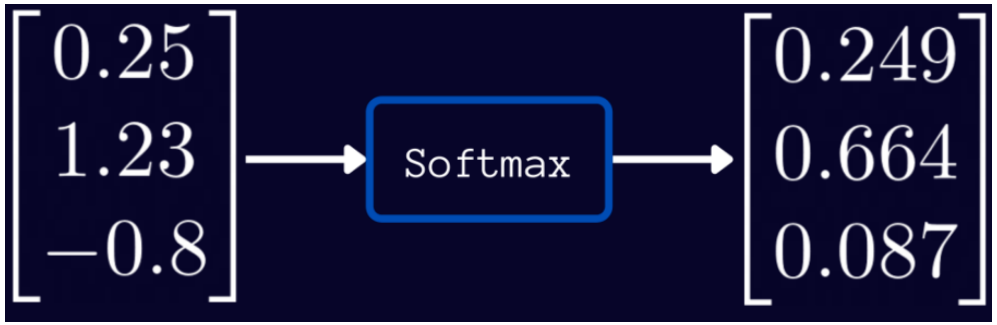
Treinamento e otimização

Função *Softmax*

- Função de ativação que atribui a cada classe de entrada uma distribuição de probabilidade.

$$S(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$





Vantagens e desvantagens

- **Vantagens:**

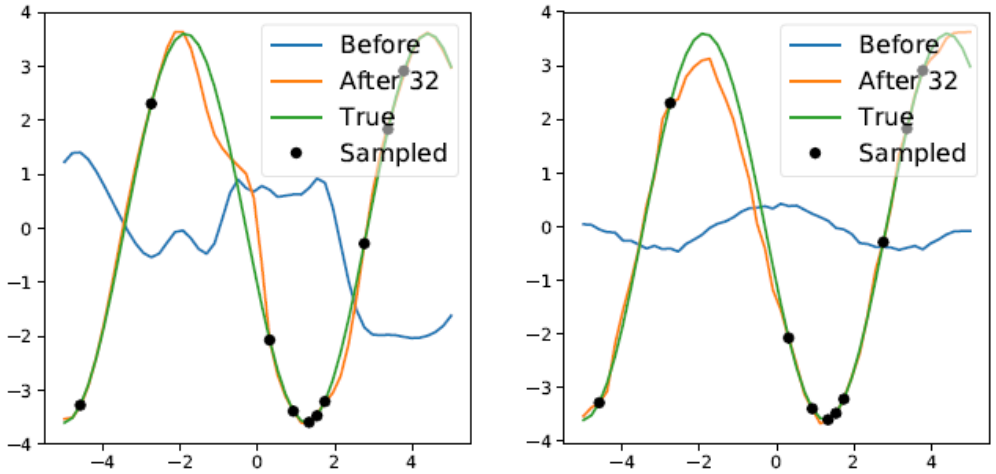
1. **Adaptação Rápida a Novas Tarefas**: Reptile é projetado para aprender rapidamente com poucos exemplos de novas tarefas ou ambientes, o que pode ser útil em cenários de transferência de aprendizado ou meta-aprendizado.
2. **Simplicidade**: O algoritmo Reptile é relativamente simples de implementar, o que o torna acessível e fácil de entender em comparação com algumas abordagens mais complexas de meta-aprendizado.
3. **Eficiência Computacional**: Reptile requer menos computação do que métodos de otimização mais intensivos, como gradient-based meta-learning (aprendizado por gradientes), tornando-o mais eficiente em termos de recursos computacionais.

- **Desvantagens:**

1. **Sensibilidade à Inicialização**: A eficácia do Reptile pode depender fortemente da inicialização do modelo e da escolha dos hiperparâmetros. Uma inicialização inadequada ou hiperparâmetros mal ajustados podem prejudicar o desempenho do algoritmo.
2. **Requer Muitas Tarefas de Treinamento**: Embora o Reptile seja projetado para adaptação rápida com poucos exemplos, ele ainda pode exigir um conjunto relativamente grande de tarefas de treinamento para garantir uma boa generalização e evitar o overfitting.
3. **Dependência da Similaridade das Tarefas**: A eficácia do Reptile pode ser limitada em cenários onde as tarefas de teste são muito diferentes das tarefas de treinamento. Nesses casos, pode ser necessária uma técnica de meta-aprendizado mais avançada ou uma abordagem diferente.

Comparação com outros algoritmos

Comparação com o **MAML**



(b) After MAML training (c) After Reptile training

Figure 1: Demonstration of MAML and Reptile on a toy few-shot regression problem, where we train on 10 sampled points of a sine wave, performing 32 gradient steps on an MLP with layers $1 \to 64 \to 64 \to 1$.

Comparação com outros algoritmos

Comparação com o **MAML e MAML de primeira ordem**

Algorithm	1-shot 5-way	5-shot 5-way
MAML + Transduction	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
1^{st}-order MAML + Transduction	$48.07 \pm 1.75\%$	$63.15 \pm 0.91\%$
Reptile	$47.07 \pm 0.26\%$	$62.74 \pm 0.37\%$
Reptile + Transduction	$49.97 \pm 0.32\%$	$65.99 \pm 0.58\%$

Table 1: Results on Mini-ImageNet. Both MAML and 1^{st}-order MAML results are from [4].

Algorithm	1-shot 5-way	5-shot 5-way	1-shot 20-way	5-shot 20-way
MAML + Transduction	$98.7 \pm 0.4\%$	$99.9 \pm 0.1\%$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$
1^{st}-order MAML + Transduction	$98.3 \pm 0.5\%$	$99.2 \pm 0.2\%$	$89.4 \pm 0.5\%$	$97.9 \pm 0.1\%$
Reptile	$95.39 \pm 0.09\%$	$98.90 \pm 0.10\%$	$88.14 \pm 0.15\%$	$96.65 \pm 0.33\%$
Reptile + Transduction	$97.68 \pm 0.04\%$	$99.48 \pm 0.06\%$	$89.43 \pm 0.14\%$	$97.12 \pm 0.32\%$

Table 2: Results on Omniglot. MAML results are from [4]. 1^{st}-order MAML results were generated by the code for [4] with the same hyper-parameters as MAML.

Exemplo(s) de aplicação

- Este exemplo demonstra a aplicabilidade do Reptile para estimação de diferentes letras de diversos alfabetos (dataset Omniglot), utilizando poucas amostras de entrada.
- Link para o colab: https://colab.research.google.com/drive/11EoaMtPHC-9gnTytEtU3Zpi2Bl_XXZlg?usp=sharing

Quiz

- Link para acesso ao quiz: https://forms.gle/vxTCzKVqUnaasD6eA

Referências

- Nichol, Alex, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms." *arXiv preprint arXiv:1803.02999* (2018).
- Nichol, Alex, and John Schulman. "Reptile: a scalable metalearning algorithm." arXiv preprint arXiv:1803.02999 2.3 (2018): 4.
- Admoreu, “Few-Shot learning with Reptile”. Keras. Disponível em: <https://keras.io/examples/vision/reptile/>, 2023.
- Wild, Cody Marie. “A Search for Efficient Meta-Learning: MAMLs, Reptiles, and Related Species”. Medium. Disponível em: <https://towardsdatascience.com/a-search-for-efficient-meta-learning-mamls-reptiles-and-related-species-e47b8fc454f2>, 2020.
- Abacus.AI. “A Beginner’s Guide to Meta-Learning”. Medium. Disponível em: <https://medium.com/abacus-ai/a-beginners-guide-to-meta-learning-73bb027007a>, 2020
- Kapil, Divakar. “Stochastic vs Batch Gradient Descent”. Medium. Disponivel em: <https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1>, 2019.
- Rahman, Moklesur. “What You Need to Know about Sparse Categorical Cross Entropy”. Medium. Disponivel em: <https://rmoklesur.medium.com/what-you-need-to-know-about-sparse-categorical-cross-entropy-9f07497e3a6f>, 2024
- Mulller, Luis *et. al.* “An Interactive Introduction to Model-Agnostic Meta-Learning”. Disponível em: <https://interactive-maml.github.io/maml.html>
- Caron, Paul. “Model Agnostic Meta-Learning made simple”. Medium. Disponível em: <https://medium.com/instadeep/model-agnostic-meta-learning-made-simple-3c170881c71a>, 2021.
- Vungarala, Seshu Kumar. “Stochastic gradient descent vs Gradient descent — Exploring the differences”. Medium. Disponível em: <https://medium.com/@seshu8hachi/stochastic-gradient-descent-vs-gradient-descent-exploring-the-differences-9c29698b3a9b#:~>, 2023.
- Priya C., Bala. “Softmax Activation Function: Everything You Need to Know”. Pinecone. Disponível em: <https://www.pinecone.io/learn/softmax-activation/>, 2023.

Obrigado!