AMC
AI-ML.Club

Optimization-Based Meta-Learning

ximingxing@gmail.com

Introduction

Deep learning models learn through backpropagation of gradients. However, the gradient-based optimization is neither designed to cope with a small number of training samples, nor to converge within a small number of optimization steps.

Is there a way to adjust the optimization algorithm so that the model can be good at learning with a few examples?

This is what optimization-based approach meta-learning algorithms intend for.

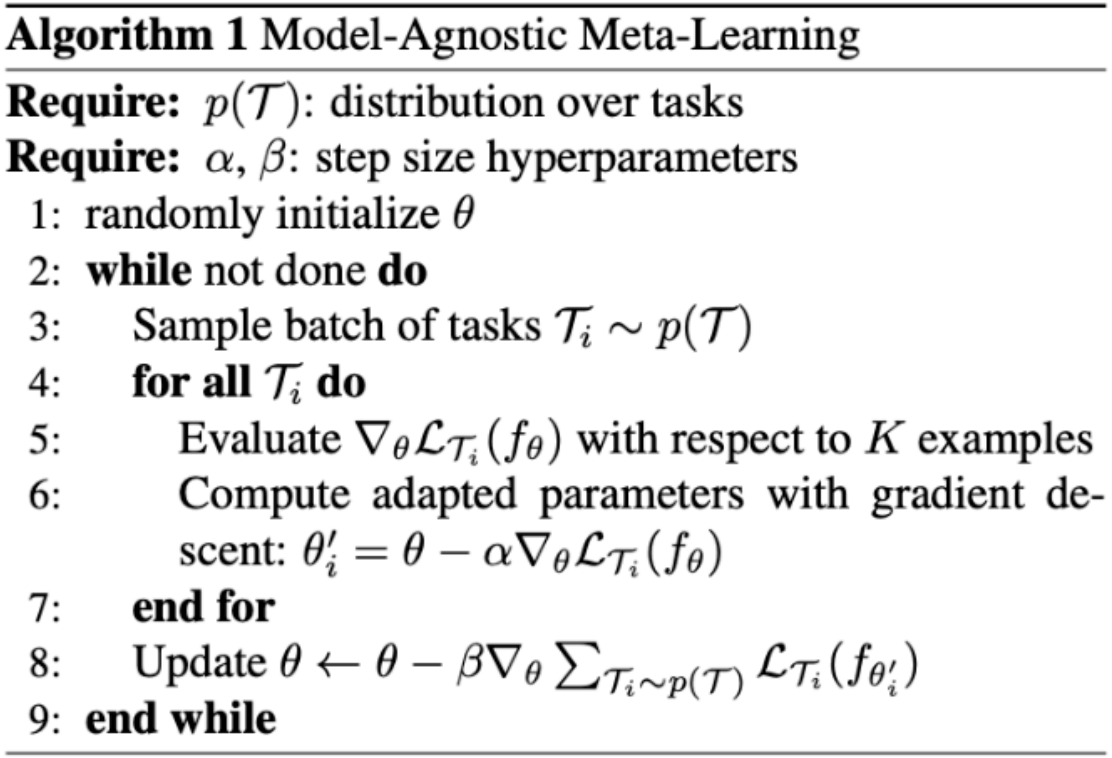
Background

“What if we directly optimized for an initial representation that can be effectively fine-tuned from a small number of examples?”

--- from author of MAML

This is exactly the idea behind our recently-proposed algorithm, model-agnostic meta-learning (MAML).

MAML: Model-Agnostic Meta-Learning



```
Algorithm 1 Model-Agnostic Meta-Learning
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
randomly initialize $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do
        Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples
        Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while
```

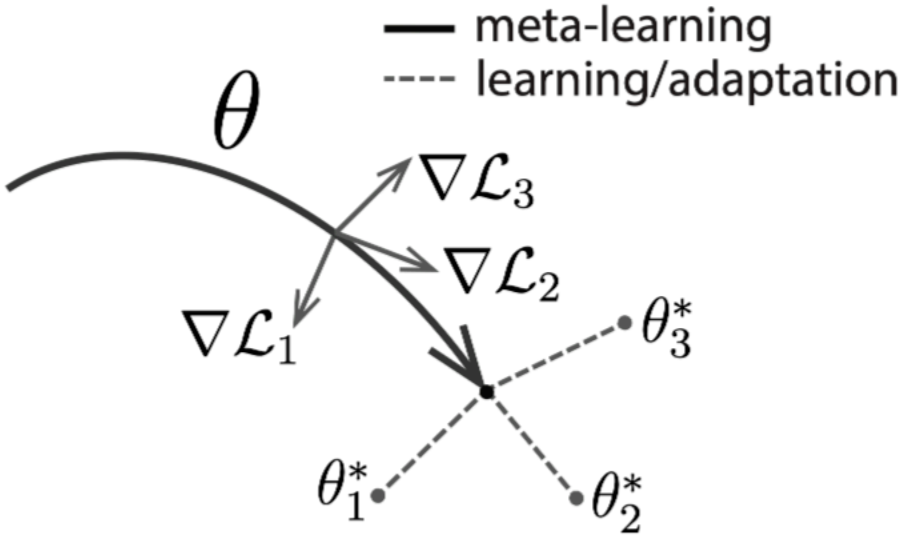
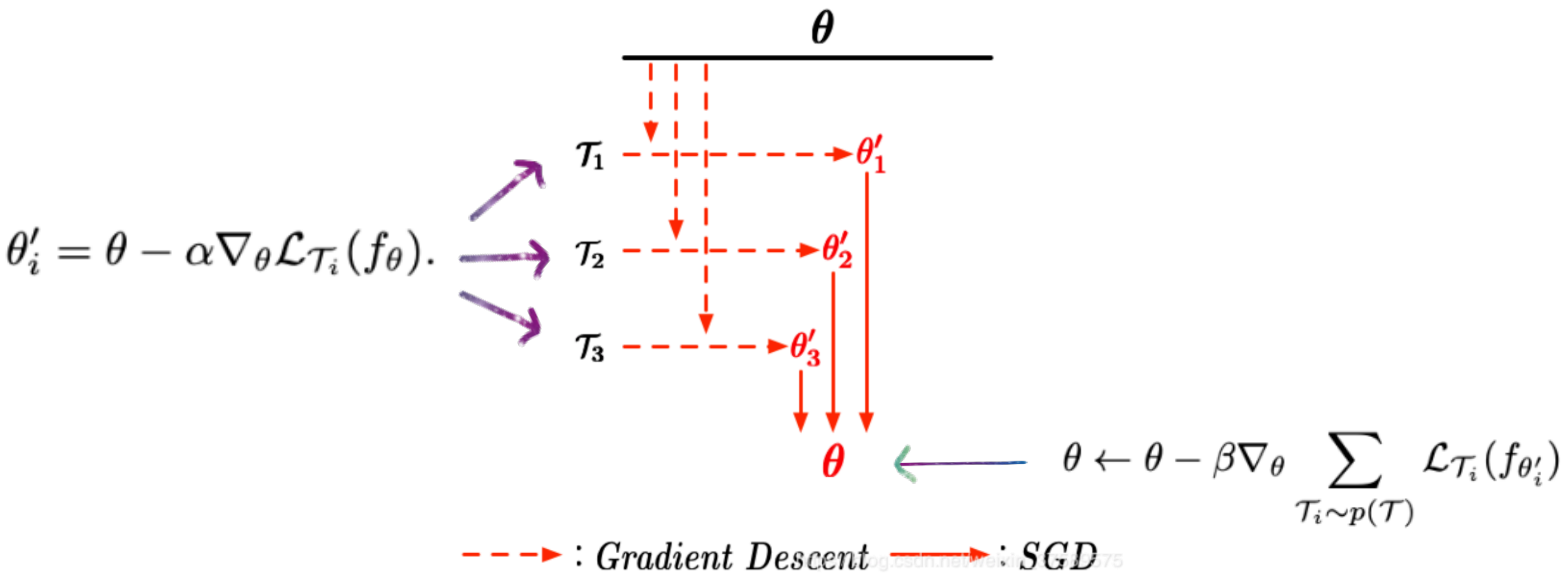


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

MAML: Model-Agnostic Meta-Learning



$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta).$$

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

AMC
AI-ML.Club

LEO: Latent Embedding Optimization

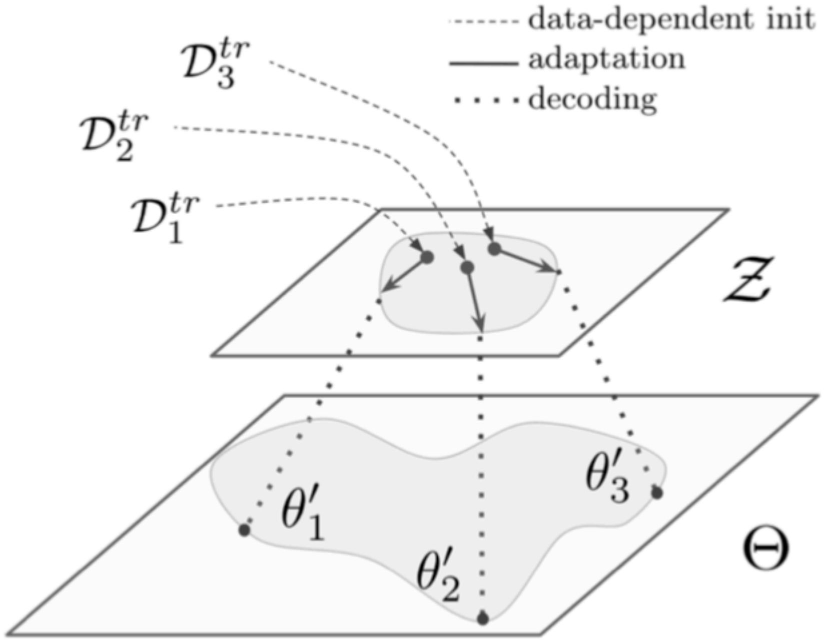


Figure 1: High-level intuition for LEO. While MAML operates directly in a high dimensional parameter space Θ, LEO performs meta-learning within a low-dimensional latent space $\mathcal{Z}$, from which the parameters are generated.

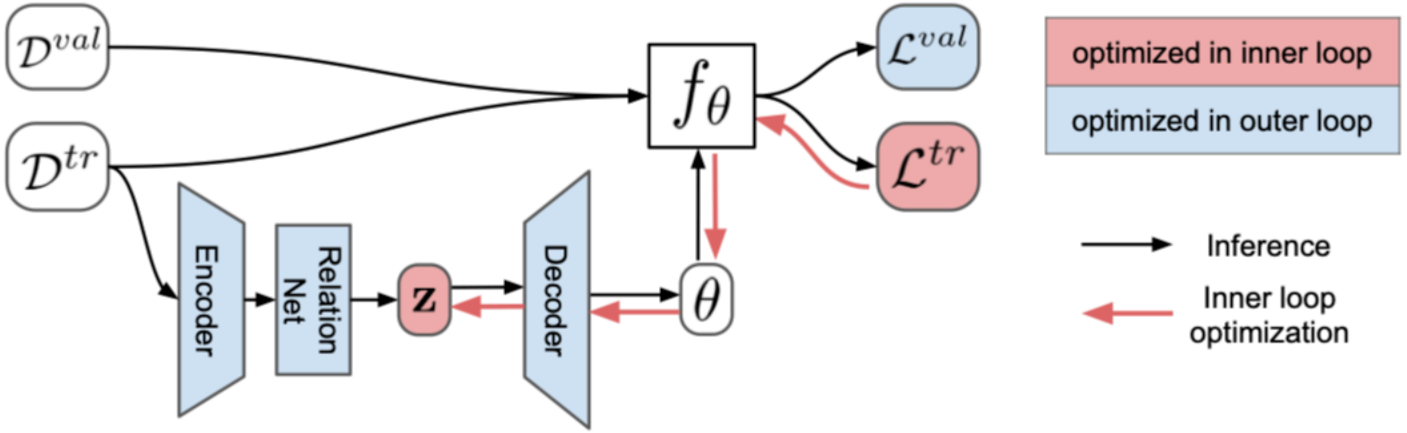
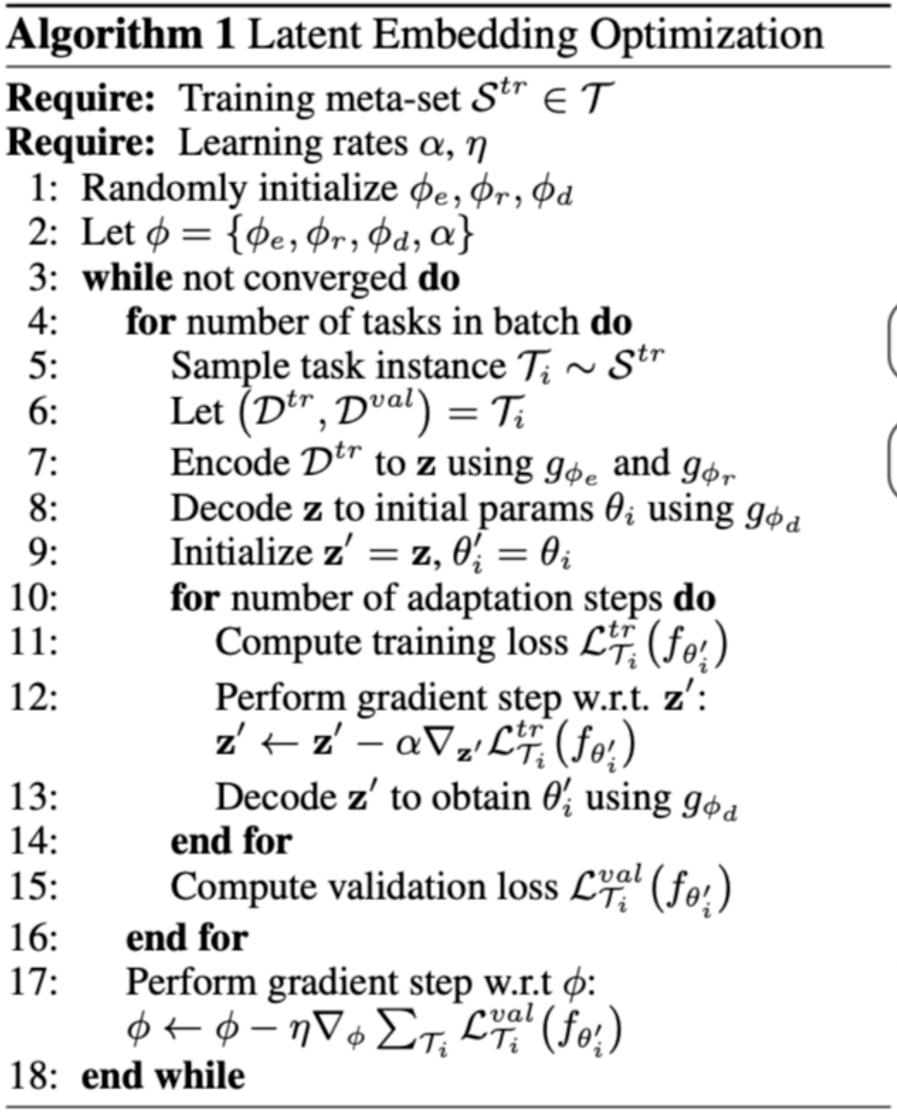


Figure 2: Overview of the architecture of LEO.

LEO: Latent Embedding Optimization



Algorithm 1 Latent Embedding Optimization

```
Require: Training meta-set $\mathcal{S}^{tr} \in \mathcal{T}$
Require: Learning rates $\alpha$, $\eta$
Randomly initialize $\phi_e, \phi_r, \phi_d$
Let $\phi = \{\phi_e, \phi_r, \phi_d, \alpha\}$
while not converged do
    for number of tasks in batch do
        Sample task instance $\mathcal{T}_i \sim \mathcal{S}^{tr}$
        Let $(\mathcal{D}^{tr}, \mathcal{D}^{val}) = \mathcal{T}_i$
        Encode $\mathcal{D}^{tr}$ to $\mathbf{z}$ using $g_{\phi_e}$ and $g_{\phi_r}$
        Decode $\mathbf{z}$ to initial params $\theta_i$ using $g_{\phi_d}$
        Initialize $\mathbf{z}' = \mathbf{z}$, $\theta_i' = \theta_i$
        for number of adaptation steps do
            Compute training loss $\mathcal{L}^{tr}_{\mathcal{T}_i}(f_{\theta_i'})$
            Perform gradient step w.r.t. $\mathbf{z}'$:
            $\mathbf{z}' \leftarrow \mathbf{z}' - \alpha \nabla_{\mathbf{z}'} \mathcal{L}^{tr}_{\mathcal{T}_i}(f_{\theta_i'})$
            Decode $\mathbf{z}'$ to obtain $\theta_i'$ using $g_{\phi_d}$
        end for
        Compute validation loss $\mathcal{L}^{val}_{\mathcal{T}_i}(f_{\theta_i'})$
    end for
    Perform gradient step w.r.t $\phi$:
    $\phi \leftarrow \phi - \eta \nabla_{\phi} \sum_{\mathcal{T}_i} \mathcal{L}^{val}_{\mathcal{T}_i}(f_{\theta_i'})$
end while
```

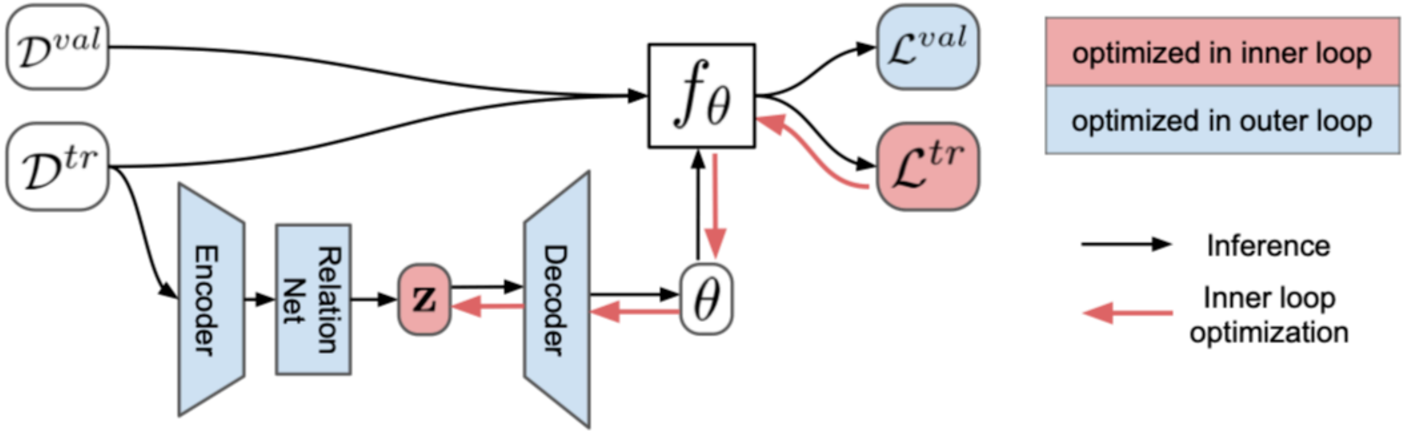
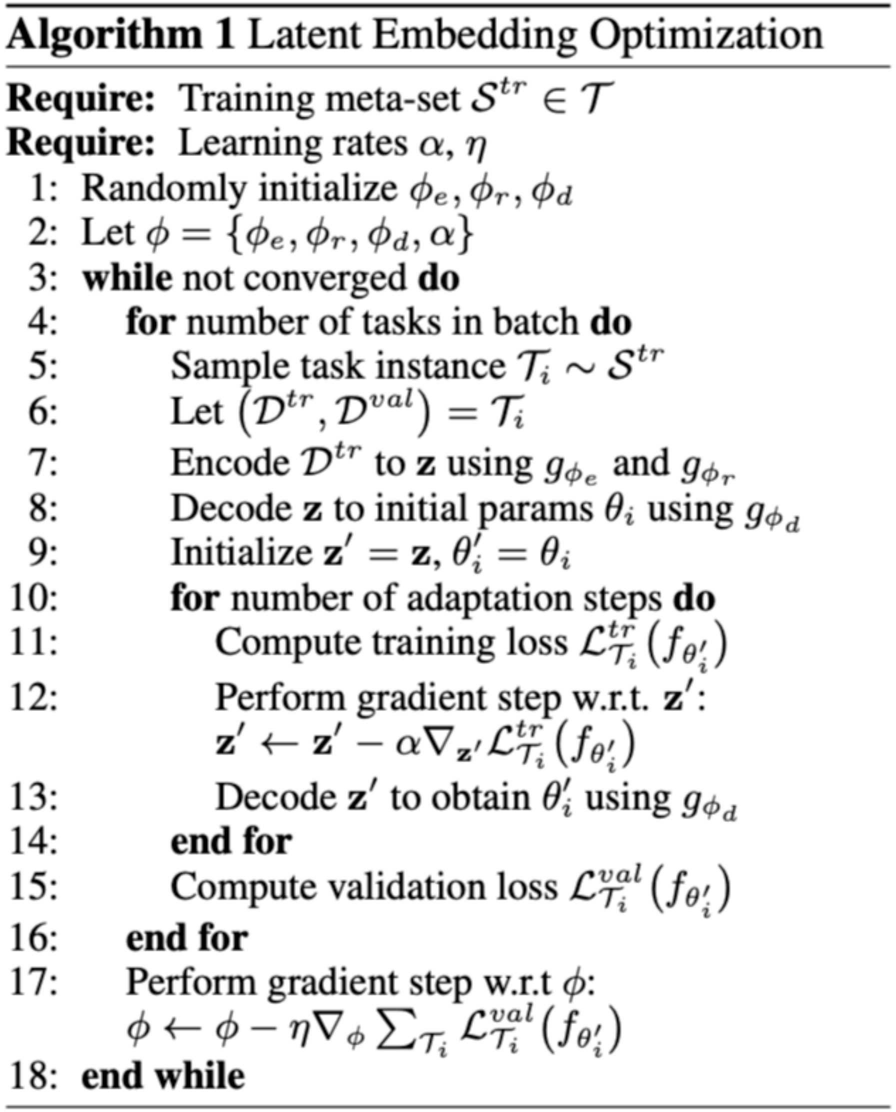


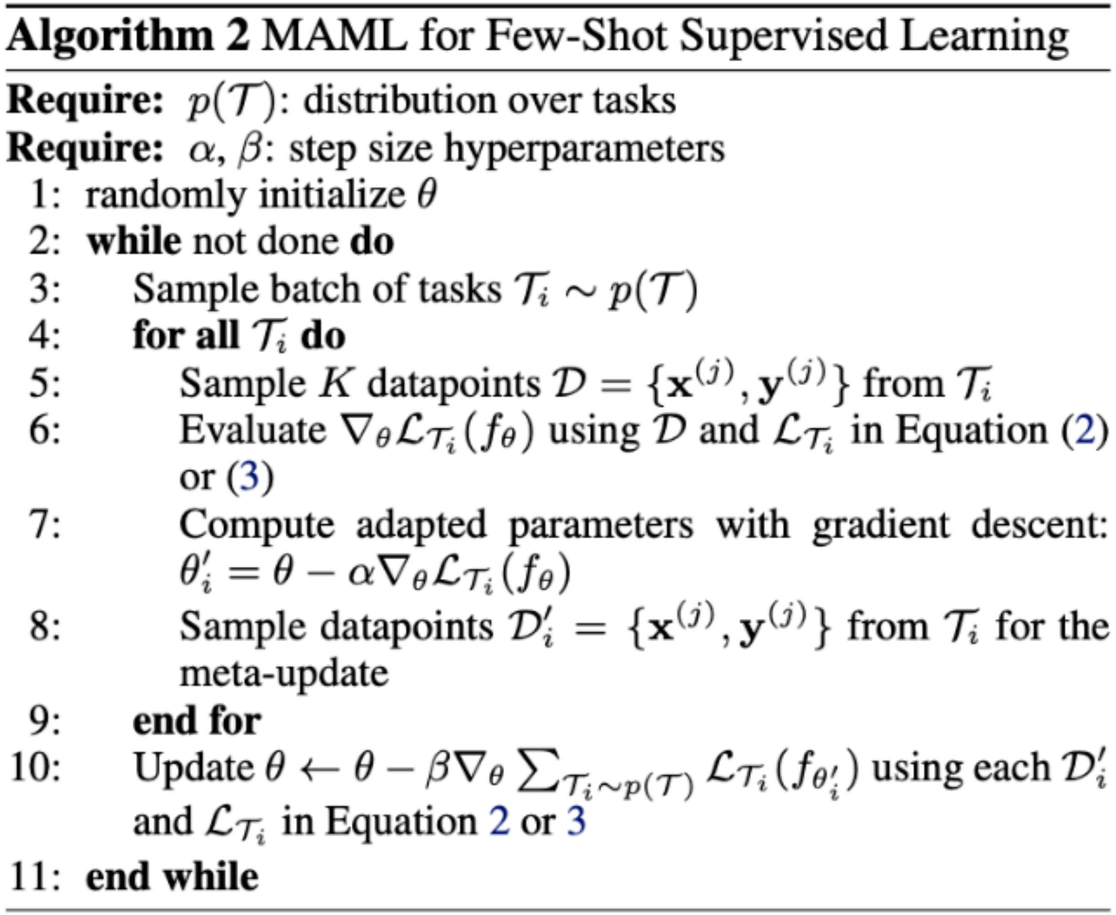
Figure 2: Overview of the architecture of LEO.

Compare



Algorithm 1 Latent Embedding Optimization

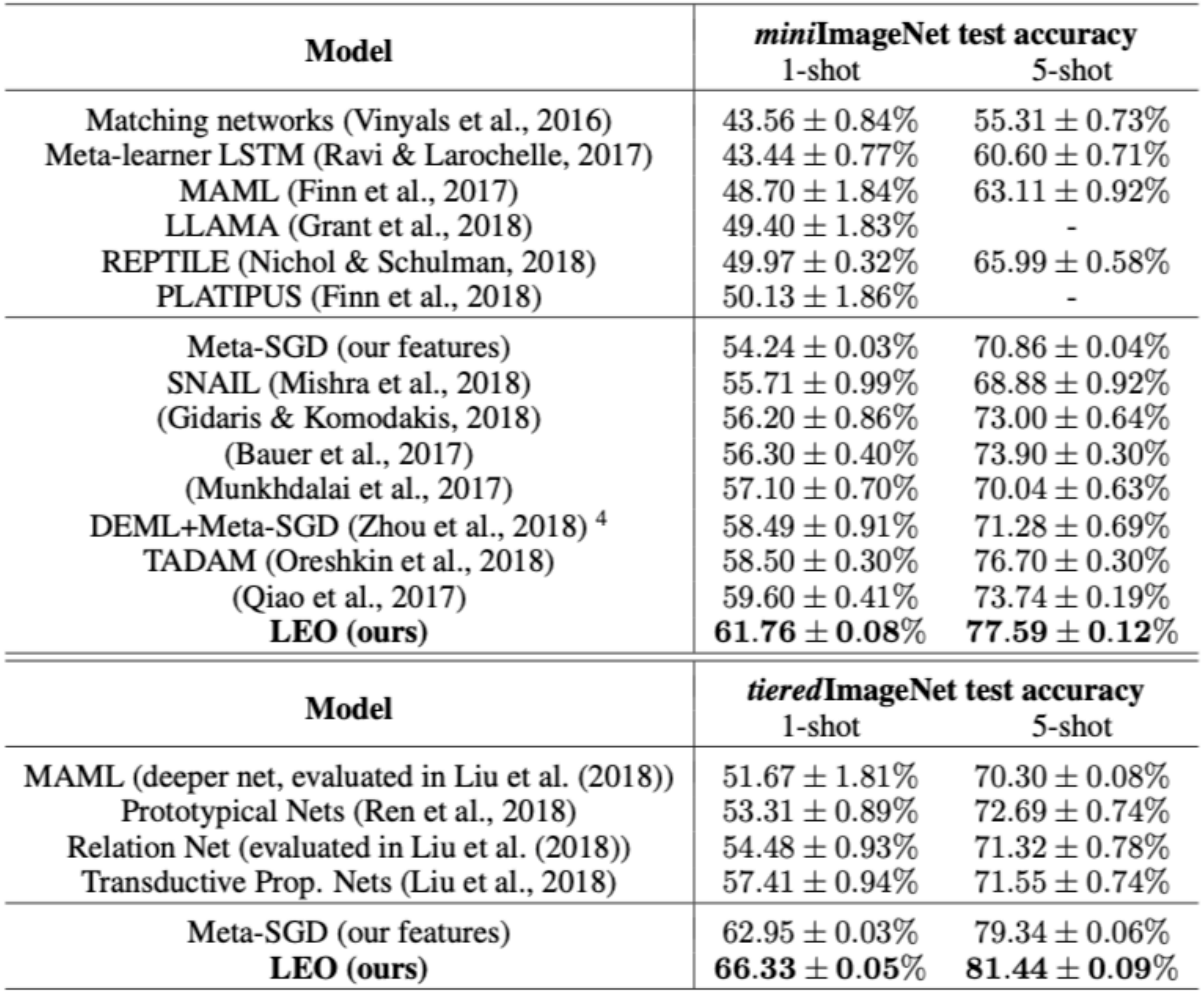
```
Require: Training meta-set $\mathcal{S}^{tr} \in \mathcal{T}$
Require: Learning rates $\alpha$, $\eta$
Randomly initialize $\phi_e, \phi_r, \phi_d$
Let $\phi = \{\phi_e, \phi_r, \phi_d, \alpha\}$
while not converged do
    for number of tasks in batch do
        Sample task instance $\mathcal{T}_i \sim \mathcal{S}^{tr}$
        Let $\left(\mathcal{D}^{tr}, \mathcal{D}^{val}\right) = \mathcal{T}_i$
        Encode $\mathcal{D}^{tr}$ to $\mathbf{z}$ using $g_{\phi_e}$ and $g_{\phi_r}$
        Decode $\mathbf{z}$ to initial params $\theta_i$ using $g_{\phi_d}$
        Initialize $\mathbf{z}' = \mathbf{z}$, $\theta'_i = \theta_i$
        for number of adaptation steps do
            Compute training loss $\mathcal{L}^{tr}_{\mathcal{T}_i}(f_{\theta'_i})$
            Perform gradient step w.r.t. $\mathbf{z}'$:
            $\mathbf{z}' \leftarrow \mathbf{z}' - \alpha \nabla_{\mathbf{z}'} \mathcal{L}^{tr}_{\mathcal{T}_i}(f_{\theta'_i})$
            Decode $\mathbf{z}'$ to obtain $\theta'_i$ using $g_{\phi_d}$
        end for
        Compute validation loss $\mathcal{L}^{val}_{\mathcal{T}_i}(f_{\theta'_i})$
    end for
    Perform gradient step w.r.t $\phi$:
    $\phi \leftarrow \phi - \eta \nabla_{\phi} \sum_{\mathcal{T}_i} \mathcal{L}^{val}_{\mathcal{T}_i}(f_{\theta'_i})$
end while
```



Algorithm 2 MAML for Few-Shot Supervised Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
randomly initialize $\theta$
while not done do
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ do
        Sample $K$ datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
        Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2)
        or (3)
        Compute adapted parameters with gradient descent:
        $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
        Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the
        meta-update
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each $\mathcal{D}'_i$
    and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
end while
```

Experiments



Model	*mini*ImageNet test accuracy	
	1-shot	5-shot
Matching networks (Vinyals et al., 2016)	43.56 ± 0.84%	55.31 ± 0.73%
Meta-learner LSTM (Ravi & Larochelle, 2017)	43.44 ± 0.77%	60.60 ± 0.71%
MAML (Finn et al., 2017)	48.70 ± 1.84%	63.11 ± 0.92%
LLAMA (Grant et al., 2018)	49.40 ± 1.83%	-
REPTILE (Nichol & Schulman, 2018)	49.97 ± 0.32%	65.99 ± 0.58%
PLATIPUS (Finn et al., 2018)	50.13 ± 1.86%	-
Meta-SGD (our features)	54.24 ± 0.03%	70.86 ± 0.04%
SNAIL (Mishra et al., 2018)	55.71 ± 0.99%	68.88 ± 0.92%
(Gidaris & Komodakis, 2018)	56.20 ± 0.86%	73.00 ± 0.64%
(Bauer et al., 2017)	56.30 ± 0.40%	73.90 ± 0.30%
(Munkhdalai et al., 2017)	57.10 ± 0.70%	70.04 ± 0.63%
DEML+Meta-SGD (Zhou et al., 2018) [4]	58.49 ± 0.91%	71.28 ± 0.69%
TADAM (Oreshkin et al., 2018)	58.50 ± 0.30%	76.70 ± 0.30%
(Qiao et al., 2017)	59.60 ± 0.41%	73.74 ± 0.19%
LEO (ours)	**61.76 ± 0.08%**	**77.59 ± 0.12%**

Model	*tiered*ImageNet test accuracy	
	1-shot	5-shot
MAML (deeper net, evaluated in Liu et al. (2018))	51.67 ± 1.81%	70.30 ± 0.08%
Prototypical Nets (Ren et al., 2018)	53.31 ± 0.89%	72.69 ± 0.74%
Relation Net (evaluated in Liu et al. (2018))	54.48 ± 0.93%	71.32 ± 0.78%
Transductive Prop. Nets (Liu et al., 2018)	57.41 ± 0.94%	71.55 ± 0.74%
Meta-SGD (our features)	62.95 ± 0.03%	79.34 ± 0.06%
LEO (ours)	**66.33 ± 0.05%**	**81.44 ± 0.09%**

Conclusion

We have introduced Latent Embedding Optimization (LEO), a meta-learning technique which uses a parameter generative model to capture the diverse range of parameters useful for a distribution over tasks, and demonstrated a new state-of-the-art result on the challenging 5-way 1- and 5-shot miniImageNet and tieredImageNet classification problems.

LEO achieves this by learning a low- dimensional data-dependent latent embedding, and performing gradient-based adaptation in this space, which means that it allows for a task-specific parameter initialization and can perform adaptation more effectively.

Code

- https://github.com/deepmind/leo
- https://github.com/cbfinn/maml
- https://github.com/dragen1860/MAML-Pytorch