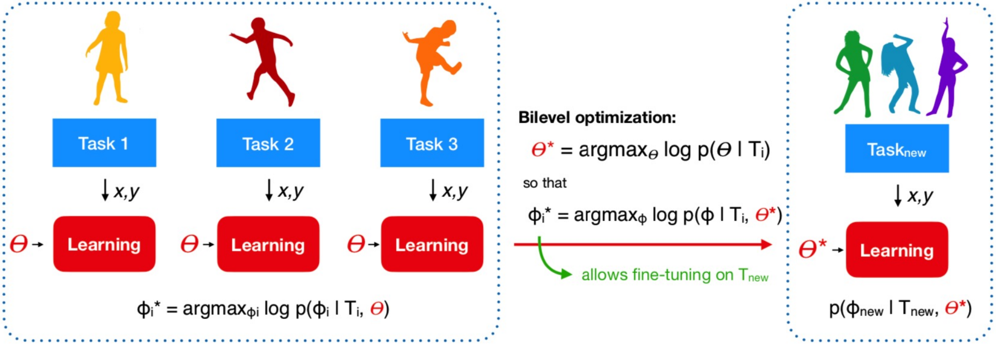
AutoML: Meta-Learning

Gradient-based meta-learning

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Gradient-based meta-learning

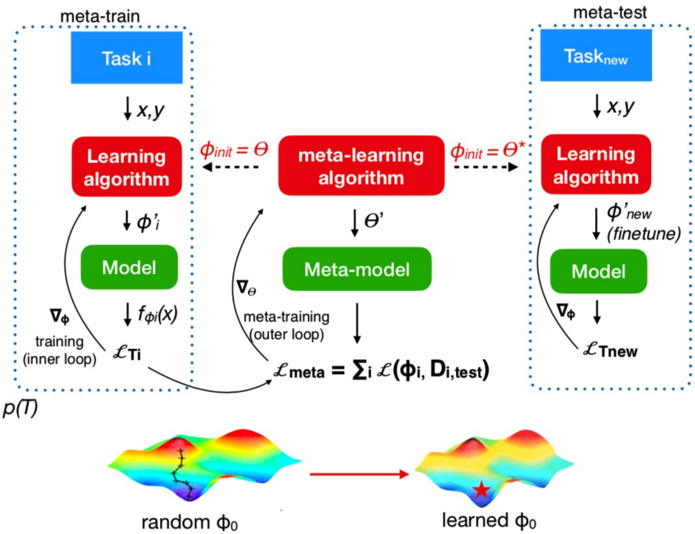
parameterize some aspect of the learner that we want to learn as meta-parameters θ meta-learn θ across tasks



$\Theta(Prior)$, could encode an initialization ϕ, the hyperparameters λ, the optimizer,...

Learned θ^* should learn T_{new} from small amount of data, yet generalize to a large number of tasks

Gradient-based methods: learning ϕ_{init}



- Θ (prior): model initialization ϕ_{init}
  - learn representation suitable for many tasks (e.g. pretrained CNN)
  - maximize rapid learning
- Each task i yields task-adapted ϕ_i
  - Update algorithm u

    $\phi_i = u(\Theta, D_{i,train})$
- Finetune Θ^* on T_{new} (in few steps)

  $\phi'_{new} = u(\Theta^*, D_{new,train})$

Can be seen as bilevel optimization:

$$\Theta^* = \mathrm{argmin}_{\Theta} \sum_i \mathcal{L}_i(\phi_i, D_{i,test})$$

$$\phi_i = u(\Theta, D_{i,train})$$

Model agnostic meta-learning (MAML)

Meta-training

- Current initialization Θ, model f_Θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$ — derivative of test-set loss
- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update Θ to minimize sum of per-task losses, repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

α, β: *learning rates*

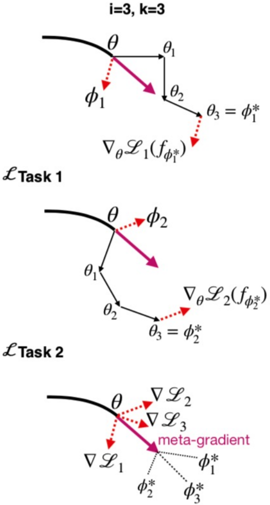
$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

meta-gradient: *second-order gradient* + backpropagate

compute how changes in Θ affect the gradient at new Θ

Meta-testing

- Training data of new task D_{train}
- Θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$



Model agnostic meta-learning (MAML)

Meta-training

- Current initialization Θ, model f_Θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$

derivative of test-set loss

- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update Θ to minimize sum of per-task losses, repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

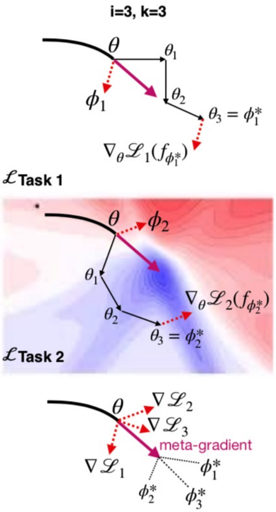
α, β: *learning rates*

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

meta-gradient: *second-order gradient* + backpropagate
compute how changes in Θ affect the gradient at new Θ

Meta-testing

- Training data of new task D_{train}
- Θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$



Model agnostic meta-learning (MAML)

Meta-training

- Current initialization Θ, model f_Θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$

derivative of test-set loss

- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update Θ to minimize sum of per-task losses, repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \textstyle\sum_i \mathcal{L}_i(f_{\phi_i})$$

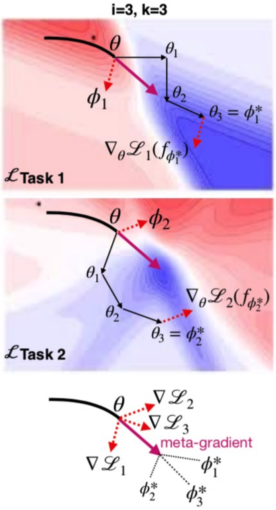
α, β: learning rates

$$\theta \leftarrow \theta - \beta \nabla_\theta \textstyle\sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

meta-gradient: <u>*second-order gradient*</u> + backpropagate
compute how changes in Θ affect the gradient at new Θ

Meta-testing

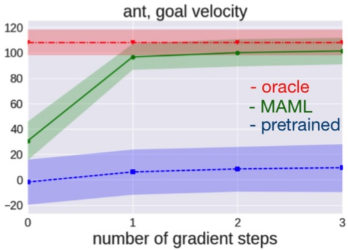
- Training data of new task D_{train}
- Θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$



Model agnostic meta-learning (MAML)

Example of reinforcement learning:

- Goal: reach certain velocity in certain direction



Other gradient-based techniques

- Changing update rule yield different variants:
  - MAML [1,6]

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

$$\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$$

  - MetaSGD [2]

$$\phi_i = \theta - \alpha diag(w) \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$$

w: weight per parameter

  - Tnet [3]

$$\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*}, w)$$

$$\phi_i = \theta - \alpha B(\theta, w) \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$$

All use second-order gradients, Meta-learn a transformation of the gradient for better adaptation

  - Meta curvature [4]

$$\phi_i = \theta - \alpha P(\theta, \phi_i) \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$$

- Online MAML (Follow the Meta Leader) [7]
  - WarpGrad [5]
  - Minimizes regret
  - Robust, but computation costs grow over time

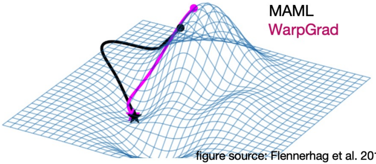
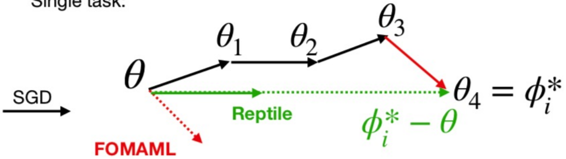


figure source: Flennerhag et al. 2019

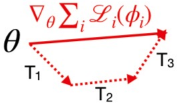
Scalability

- Backpropagating derivates of ϕ_i wrt θ is compute + memory intensive (for large models)
- First order approximations of MAML:
  - First order MAML [1] uses only the last inner gradient update: $\theta \leftarrow \theta - \beta \sum_i \mathcal{L}_i(f_{\phi_i^*})$
  - Reptile [2]: iterate over tasks, update θ in direction of ϕ_i^*: $\theta \leftarrow \theta - \beta\,(\phi_i^* - \theta)$

Single task:



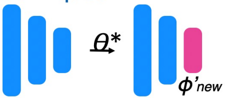
FOMAML meta-gradient:



Generalizability

[1] Finn et al. 2018
[2] Raghu et al. 2020
[3] Tian et al. 2020
[4] Stadie et al. 2019

- MAML is more resilient to overfitting than many other meta-learning techniques [1]
- Effectiveness seems mainly due to feature reuse (finding θ) [2]
  - Fine-tuning *only the last layer* equally good



- On few-shot learning benchmarks, a good embedding outperforms most meta-learning [3]
  - Learn representation on entire meta-dataset (merged into single task)
  - Train a linear classifier on embedded few-shot D_{train}, predict D_{test}
- For meta-RL, also learn how to *explore* (how to sample new environments)
  - E-MAML: add exploration to meta-objective (allows longer term goals) [4]

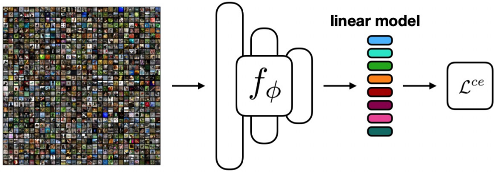
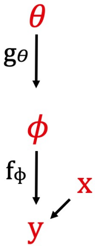


figure source: Tian et al. 2020

Bayesian meta-learning

Can meta-learning reason about uncertainty in the task distribution?



$T_i = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-train}$

$\hat{\phi} \leftarrow g_\theta((x,y)_{train})$

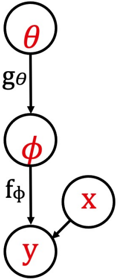
$y_{test} \leftarrow f_{\hat{\phi}}(x_{test})$

$\theta \leftarrow \theta - \nabla_\theta \mathcal{L}(y_{test})$

$T_j = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-test}$

$\phi_j \leftarrow g_\theta((x,y)_{train})$ *finetuning*

$y_{test} \leftarrow f_{\phi_j}(x_{test})$



$T_i = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-train}$

$p(\phi_i | (x,y)_{train})$

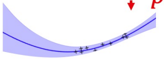
$p(y_{test} | x_{test})$

$p(\theta) \leftarrow p(\theta | (x,y)_{train})$

$T_j = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-test}$

$p(\phi_j | (x,y)_{train}, \theta)$ *finetuning*

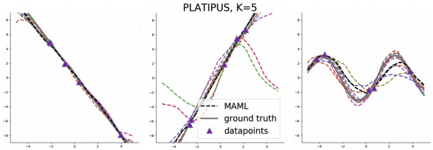
$p(y_{test} | x_{test})$

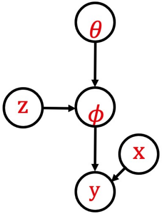


Fully Bayesian meta-learning

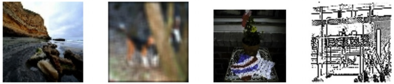
[1] Finn et al. 2019
[2] Kim et al. 2018
[3] Ravi et al. 2019
[4] Grant et al. 2018
[5] Edwards et al. 2017
[6] Garnelo et al. 2018
[7] Jerfel et al. 2019

- Use approximation methods to represent uncertainty over ϕ_i
  - Sampling technique + variational inference: PLATIPUS [1], BMAML [2], ABML [3]
  - Laplace approximation: LLAMA [4]
  - Variational approximation of posterior:
    - Neural Statistician [5], Neural Processes [6]
- What if our tasks are not IID?
  - Impose additional structure, e.g. with task-specific variable z [7]





mini-ImageNet with filters for non-homogeneous tasks:



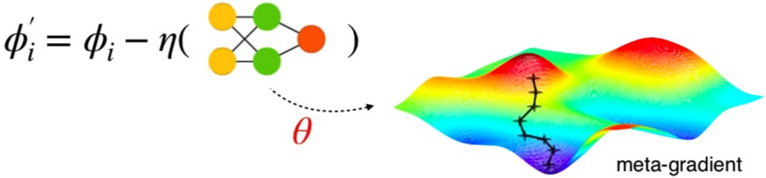
(a) plain **(b)** blur **(c)** night **(d)** pencil

Figure source: Jerfel et al. 2019

Meta-learning optimizers

Our brains probably don't do backprop, instead:

- weights: networks that continuously modify the weights of another network
- Gradient based: parameterize the update rule using a neural network
  - Learn meta-parameters across tasks, by gradient descent



- Represent update rule as an LSTM, hierarchical RNN

Meta-learning optimizers

- Meta-learned (RNN) optimizers ‘rediscover’ momentum, gradient clipping, learning rate schedules, learning rate adaptation,. . .
- RL-based optimizers: represent updates as a policy, learn using guided policy search
- Combined with MAML:
  - learn per-parameter learning rates
  - learn precondition matrix (to ‘warp’ the loss surface)
- Black-box optimizers: meta-learned with an RNN, or with user-defined priors
- Speed up backpropagation by meta-learning sparsity and weight sharing

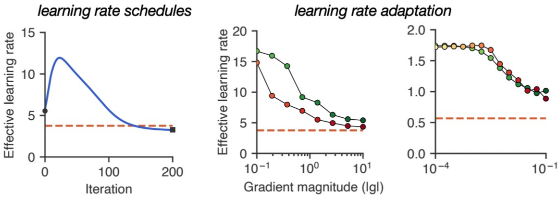


Figure source:
Maheswaranathan et al. 2020