Meta-learning and Inductive Bias

Learning to learn by gradient descent by gradient descent

Slides from Sergey Levine

ECE324, Winter 2023

Michael Guerzhoy

No Free Lunch Theorem

- (on the board)

Inductive bias

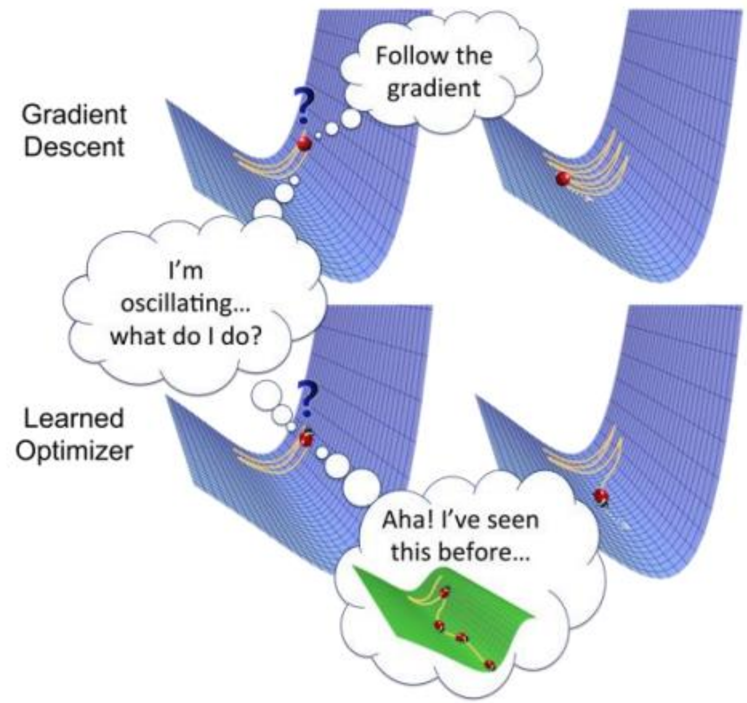
- The assumptions about the data that are built into a machine learning algorithm
- Linear/logistic regression
  - The output can be predicted as a linear function of the input
- k-NN with Euclidean distance
  - Nearby points in Euclidean space have similar labels
- ConvNet layers
  - Convolutions are useful for predicting the output
  - The output can be predicted from functions that have local support in the input
- L2 Regularization
  - The output is approximately linear in the input

Few shot learning

- Learning with few input examples
  - Few-shot learning: only a few examples are given per class
  - One-shot learning: only one example given per output class
  - (Zero-shot learning: classifying inputs without seeing examples of the class, but seeing some kind of description. E.g., finding zebras given the description “striped horse”)
- Requires strong (and appropriate) inductive bias

Meta-learning

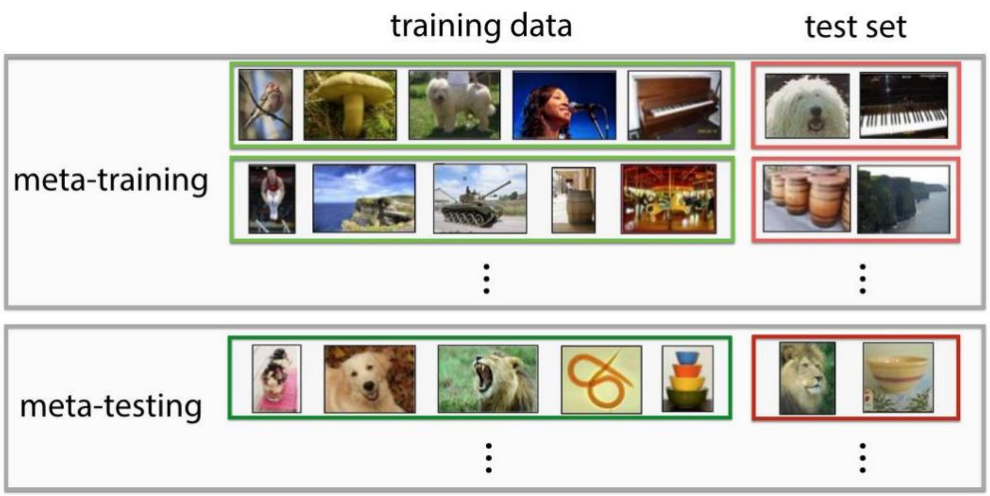
- Tuning the learning process by learning multiple related tasks
- Many formulations
    - Learning an optimizer
    - Learning an RNN that ingests experience
    - Learning a representation
- Tuning the inductive bias on the training set so as to well on the test set



Why is meta-learning a good idea?

- Deep learning works well, but requires large datasets
- In many cases, we have little data available for a specific task, but have more data for other, related tasks

Meta-learning with supervised learning



supervised learning: $f(x) \rightarrow y$

input (e.g., image)   output (e.g., label)

supervised meta-learning: $f(\mathcal{D}^{\mathrm{tr}}, x) \rightarrow y$

training set

Supervised meta-learning with RNNs

"Generic" learning:

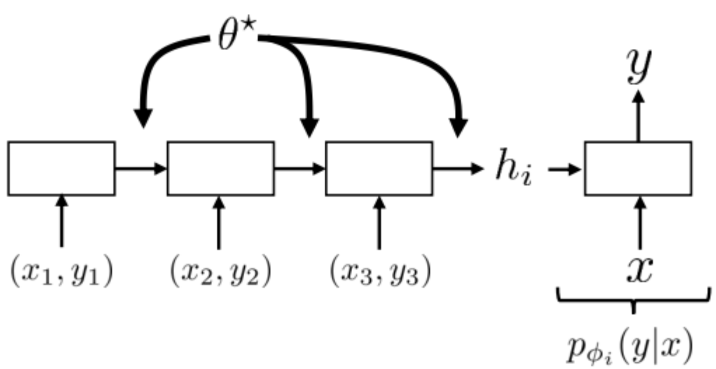
$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

$$= f_{\text{learn}}(\mathcal{D}^{\text{tr}})$$

"Generic" meta-learning:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$



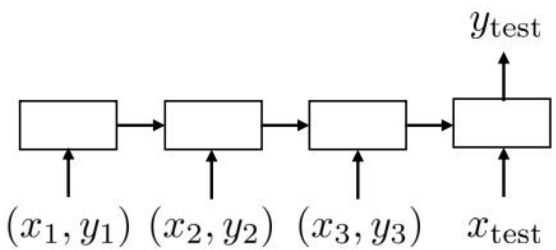
RNN hidden state

meta-learned weights

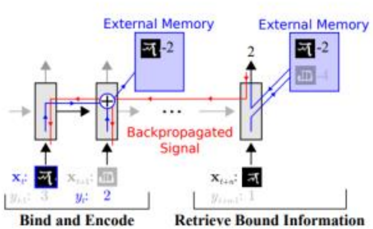
$$\phi_i = [h_i, \theta_p]$$

Meta-learning methods

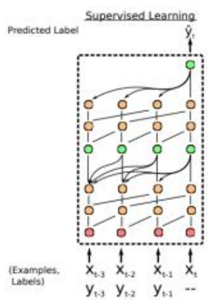
black-box meta-learning



some kind of network that can read in an entire (few-shot) training set

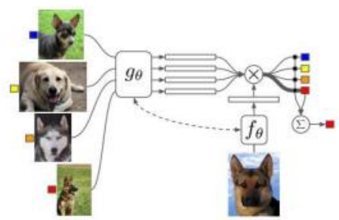


Santoro et al. **Meta-Learning with Memory-Augmented Neural Networks**. 2016.

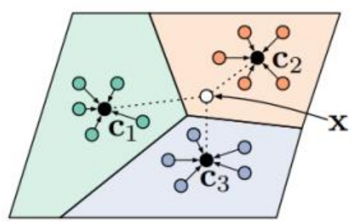


Mishra et al. **A Simple Neural Attentive Meta-Learner**. 2018.

non-parametric meta-learning

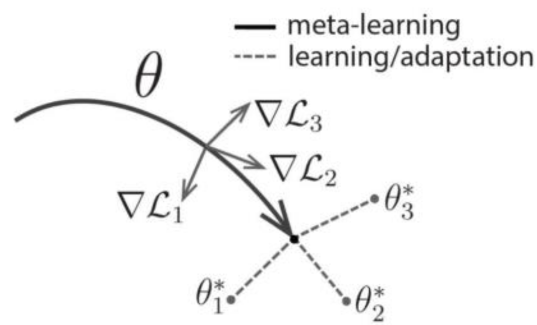


Vinyals et al. **Matching Networks for One Shot Learning**. 2017.



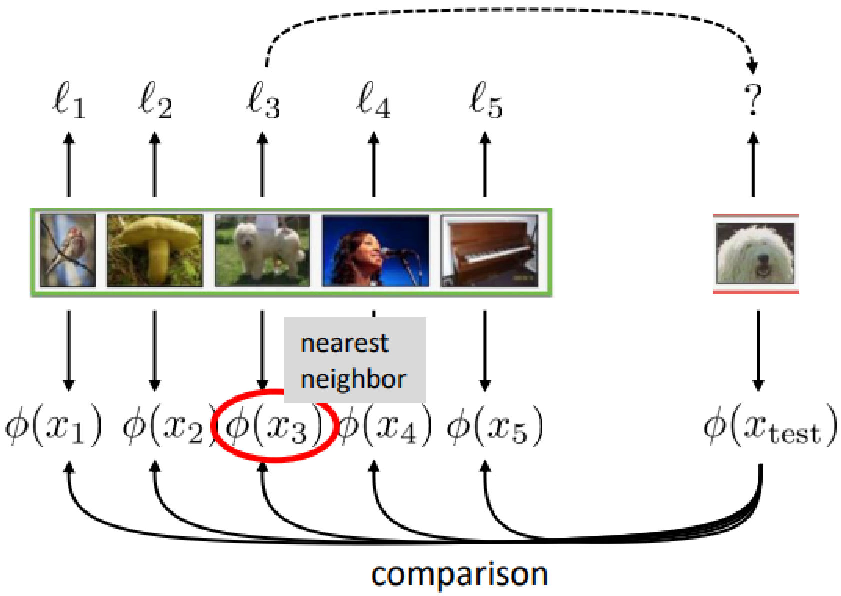
Snell et al. **Prototypical Networks for Few-shot Learning**. 2018.

gradient-based meta-learning



Finn et al. **Model-Agnostic Meta-Learning**. 2018.

Basic idea: Nearest Neighbours



why does this work?

that is, why does the nearest neighbor have the right class?

because we **meta-train** the features so that this produces the right answer!

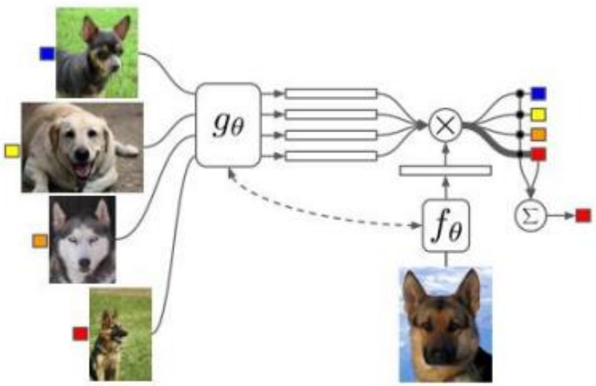
$\phi = \phi_\theta$

$$p_{\text{nearest}}(x_k^{\text{tr}}|x_j^{\text{ts}}) \propto \exp(\phi(x_k^{\text{tr}})^T \phi(x_j^{\text{ts}}))$$

$$p_\theta(y_j^{\text{ts}}|x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}) = \underbrace{\sum_{k:y_k^{\text{tr}}=y_j^{\text{ts}}} p_{\text{nearest}}(x_k^{\text{tr}}|x_j^{\text{ts}})}_{\text{all training points that have this label}}$$

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\underbrace{f_\theta(\mathcal{D}_i^{\text{tr}})}_{\text{learned (soft) nearest neighbor classifier}}, \mathcal{D}_i^{\text{ts}}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} \log p_\theta(y_j^{\text{ts}}|x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}})$$

Matching networks



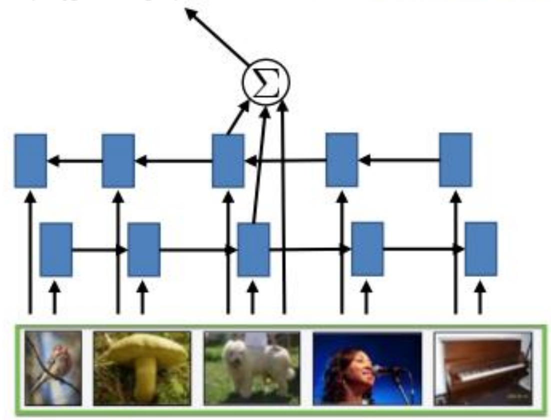
$$p_\theta(y_j^{\text{ts}} | x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}) = \sum_{k: y_k^{\text{tr}} = y_j^{\text{ts}}} p_{\text{nearest}}(x_k^{\text{tr}} | x_j^{\text{ts}})$$

$$p_{\text{nearest}}(x_k^{\text{tr}} | x_j^{\text{ts}}) \propto \exp(g(x_k^{\text{tr}}, \mathcal{D}_i^{\text{tr}})^T f(x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}))$$

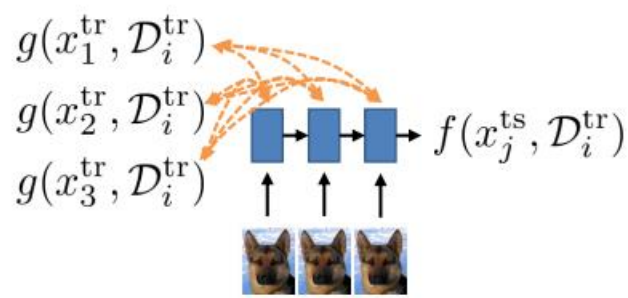
different nets to embed x^{tr} and x^{ts}

both f and g conditioned on entire set $\mathcal{D}_i^{\text{tr}}$

$g(x_k^{\text{tr}}, \mathcal{D}_i^{\text{tr}})$ bidirectional LSTM embedding

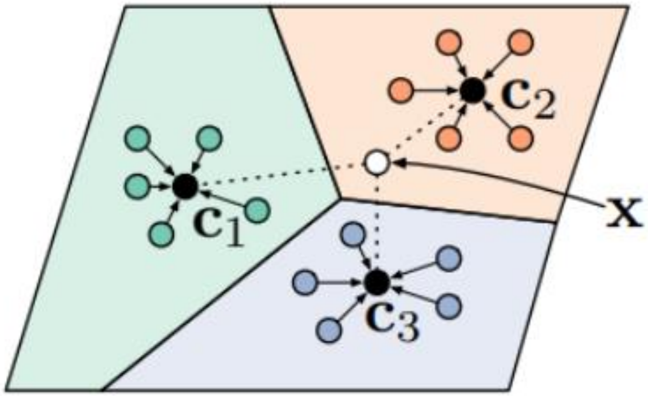


$f(x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}})$ attentional LSTM embedding



Vinyals et al. **Matching networks for few-shot learning**. 2016.

Prototypical networks



Two simple ideas compared to matching networks:

1. Instead of "soft nearest neighbor," construct prototype for each class

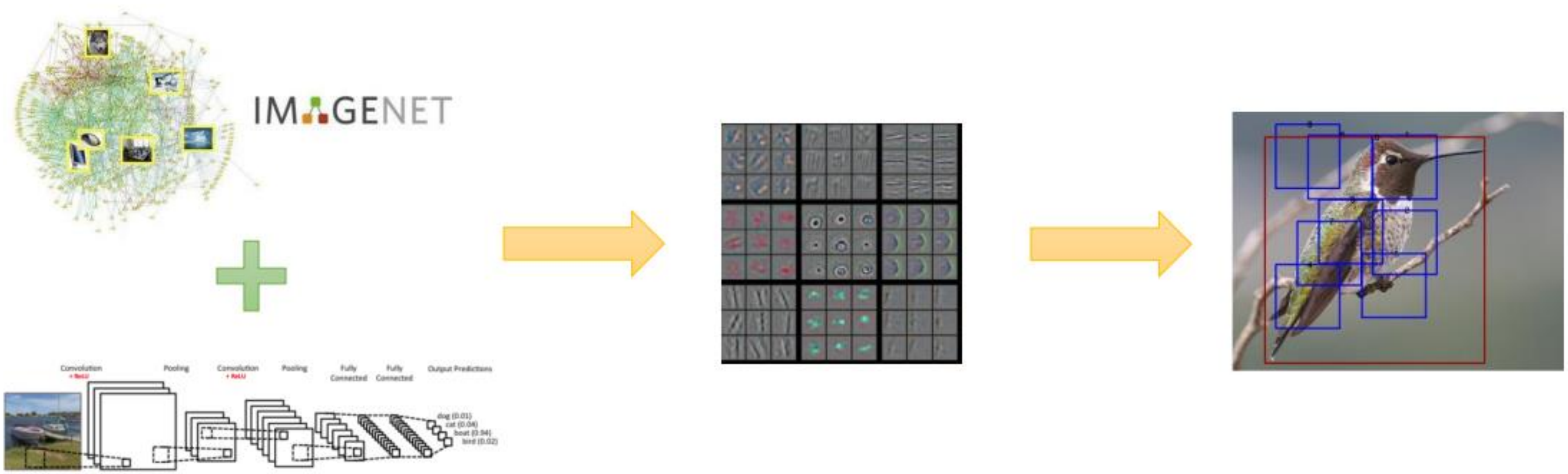
$$p_\theta(y|x_j^{\text{ts}}, \mathcal{D}_i^{\text{tr}}) \propto \exp(c_y^T f(x_j^{\text{ts}})) \quad c_y = \frac{1}{N_y} \sum_{k:y_k^{\text{tr}}=y} g(x_k^{\text{tr}})$$

2. Get rid of all the complex junk ~~bidirectional LSTM embedding~~

~~attentional LSTM embedding~~

Snell et al. **Prototypical networks for few-shot learning**. 2017.

Representation



is pretraining a *type* of meta-learning?

better features = faster learning of new task!

Meta-learning as an optimization problem

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$

what if $f_\theta(\mathcal{D}_i^{\text{tr}})$ is just a *finetuning* algorithm?

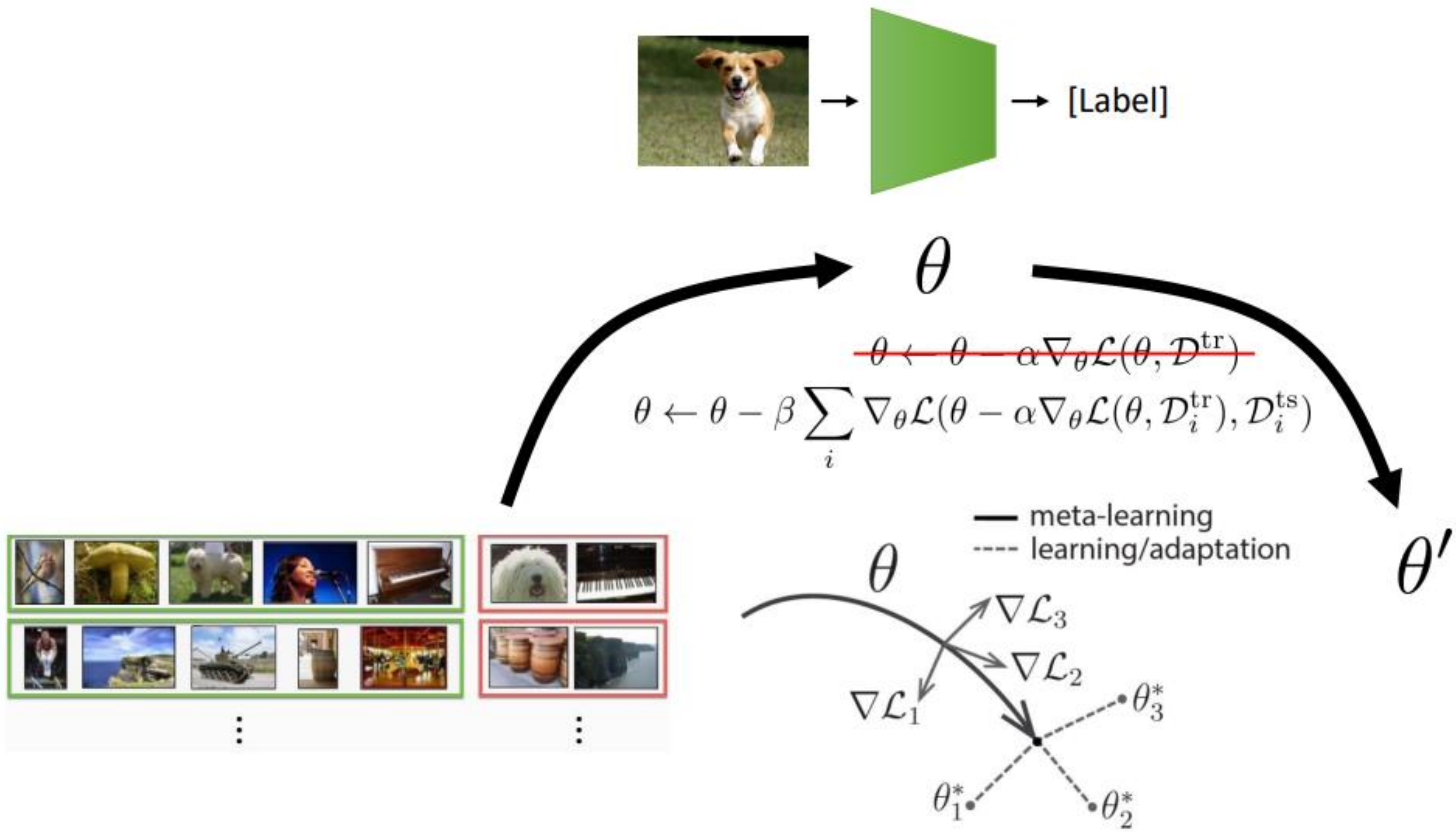
$$f_\theta(\mathcal{D}_i^{\text{tr}}) = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}})$$

(could take a few gradient steps in general)

This can be trained the same way as any other neural network, by implementing gradient descent as a computation graph and then running backpropagation *through* gradient descent!

Finn, Abbeel, Levine. **Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks**.

MAML in pictures



What did we just do?

supervised learning: $f(x) \to y$

supervised meta-learning: $f(\mathcal{D}^{\mathrm{tr}}, x) \to y$

model-agnostic meta-learning: $f_{\mathrm{MAML}}(\mathcal{D}^{\mathrm{tr}}, x) \to y$

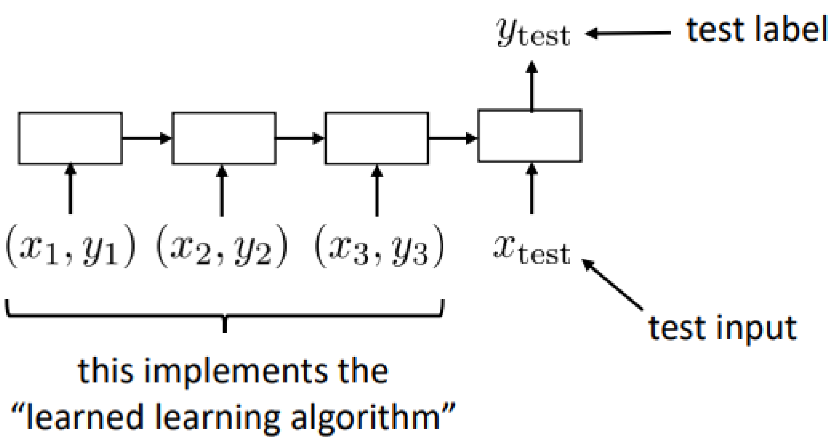
$$f_{\mathrm{MAML}}(\mathcal{D}^{\mathrm{tr}}, x) = f_{\theta'}(x)$$

$$\theta' = \theta - \alpha \sum_{(x,y)\in\mathcal{D}^{\mathrm{tr}}} \nabla_\theta \mathcal{L}(f_\theta(x), y)$$

Just another computation graph

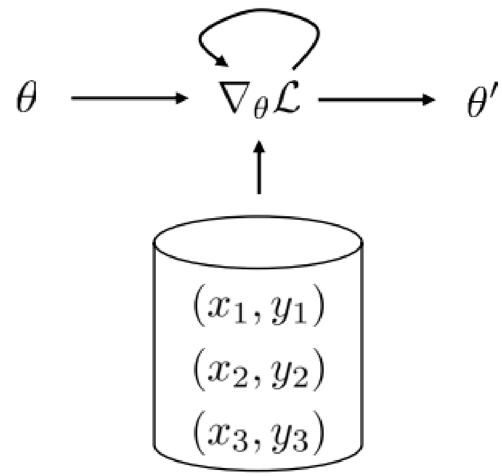
Why does it work?

box
black-based meta-learning



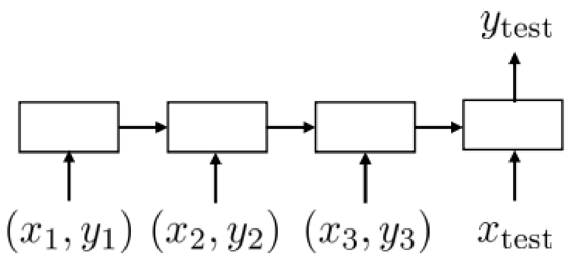
- Does it converge?
  - Kind of?
- What does it converge to?
  - Who knows…
- What to do if it's not good enough?
  - Nothing…

MAML



- Does it converge?
  - Yes (it's gradient descent…)
- What does it converge to?
  - A local optimum (it's gradient descent…)
- What to do if it's not good enough?
  - Keep taking gradient steps (it's gradient descent…)

black-box meta-learning



some kind of network that can read in an entire (few-shot) training set

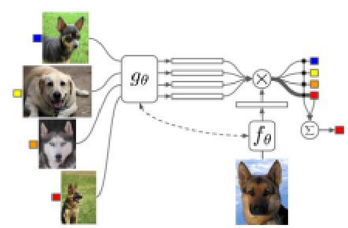
+ conceptually very simple

+ benefits from advances in sequence models (e.g., transformers)

- minimal inductive bias (i.e., *everything* has to be meta-learned)

- hard to scale to "medium" shot (we get long "sequences)

non-parametric meta-learning



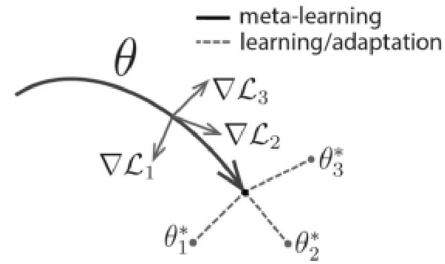
Vinyals et al. **Matching Networks for One Shot Learning**. 2017.

+ can work very well by combining some inductive bias with easy end-to-end optimization

- restricted to classification, hard to extend to other settings like regression or reinforcement learning

- somewhat specialized architectures

gradient-based meta-learning



Finn et al. **Model-Agnostic Meta-Learning**. 2018.

+ easy to apply to any architecture or loss function (inc. RL, regression)

+ good generalization to out-of-domain tasks

- meta-training optimization problem is harder, requires more tuning

- requires second derivatives