Optimization as a Model for Few-Shot Learning

Sachin Ravi
Princeton University & Twitter

In collaboration with

Hugo Larochelle

A RESEARCH AGENDA

- Deep learning successes have required a lot of labeled training data
  - collecting and labeling such data requires significant human labor
  - is that really how we'll solve AI ?
- Alternative solution : exploit other sources of data that are imperfect but plentiful
  - unlabeled data (unsupervised learning)
  - multi-modal data (multimodal learning)
  - multi-domain data (transfer learning)

A RESEARCH AGENDA

- One example of this problem: ***few-shot learning***
  - Defined as ***k-shot***, ***N-class*** classification: ***k*** examples for each of ***N*** classes
  - Model needs to generalize after seeing few examples from each class

META-LEARNING

- How to do well at few-shot training task?
  - Training algorithms such as SGD or ADAM prone to overfitting with random initialization
    - hard to know what good initialization is
  - We want to design a training algorithm for each small dataset
    - given training set with few examples
    - should output parameters θ for model M that generalize well to test set

- Idea: let's *learn* such a training algorithm, *end-to-end*
  - this is known as **meta-learning** or **learning-to-learn**

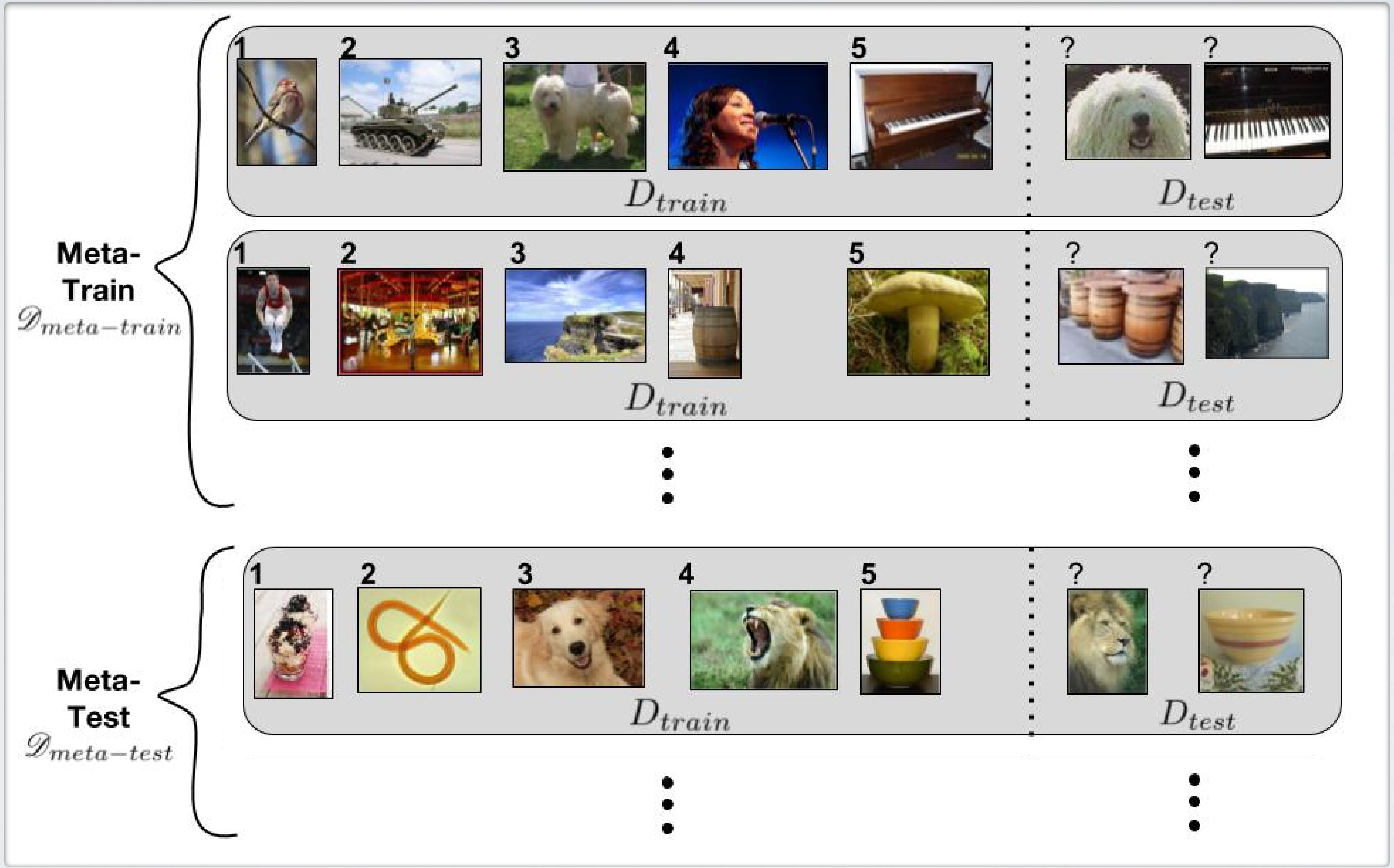
META-LEARNING

- Consider a training algorithm
  - *input*: training set $D_{train} = \{(\mathbf{X}_t, \mathbf{Y}_t)\}_{t=1}^{T}$
  - *output*: parameters θ of model M
  - *objective*: good performance on test set $D_{test} = (\mathbf{X}, \mathbf{Y})$

- Desire a meta-learning algorithm
  - *input*: meta-training set $\mathscr{D}_{meta-train} = \{(D_{train}^{(n)}, D_{test}^{(n)})\}_{n=1}^{N}$
  - *output*: parameters Θ representing a training algorithm
  - *objective*: good performance on meta-test set $\mathscr{D}_{meta-test} = \{(D_{train}^{(n')}, D_{test}^{(n')})\}_{n'=1}^{N'}$

META-LEARNING

META-LEARNING



A META-LEARNING MODEL

- How to parametrize training algorithms?
  - we take inspiration from the gradient descent algorithm:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

  - we parametrize this update similarly to LSTM state updates:

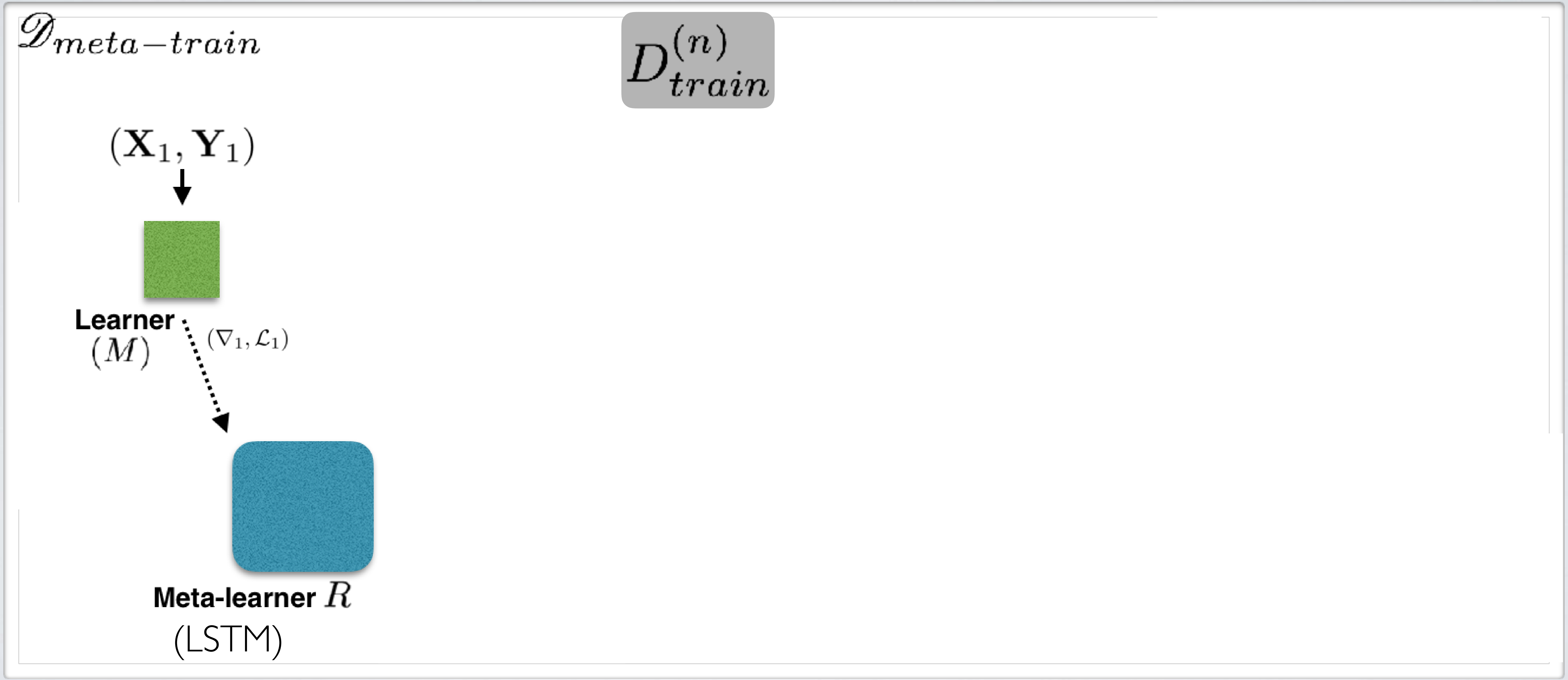
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

    - state c_t is model M's parameters
    - state candidate $\tilde{c}_t$ is the negative gradient
    - f_t and i_t are LSTM gates:

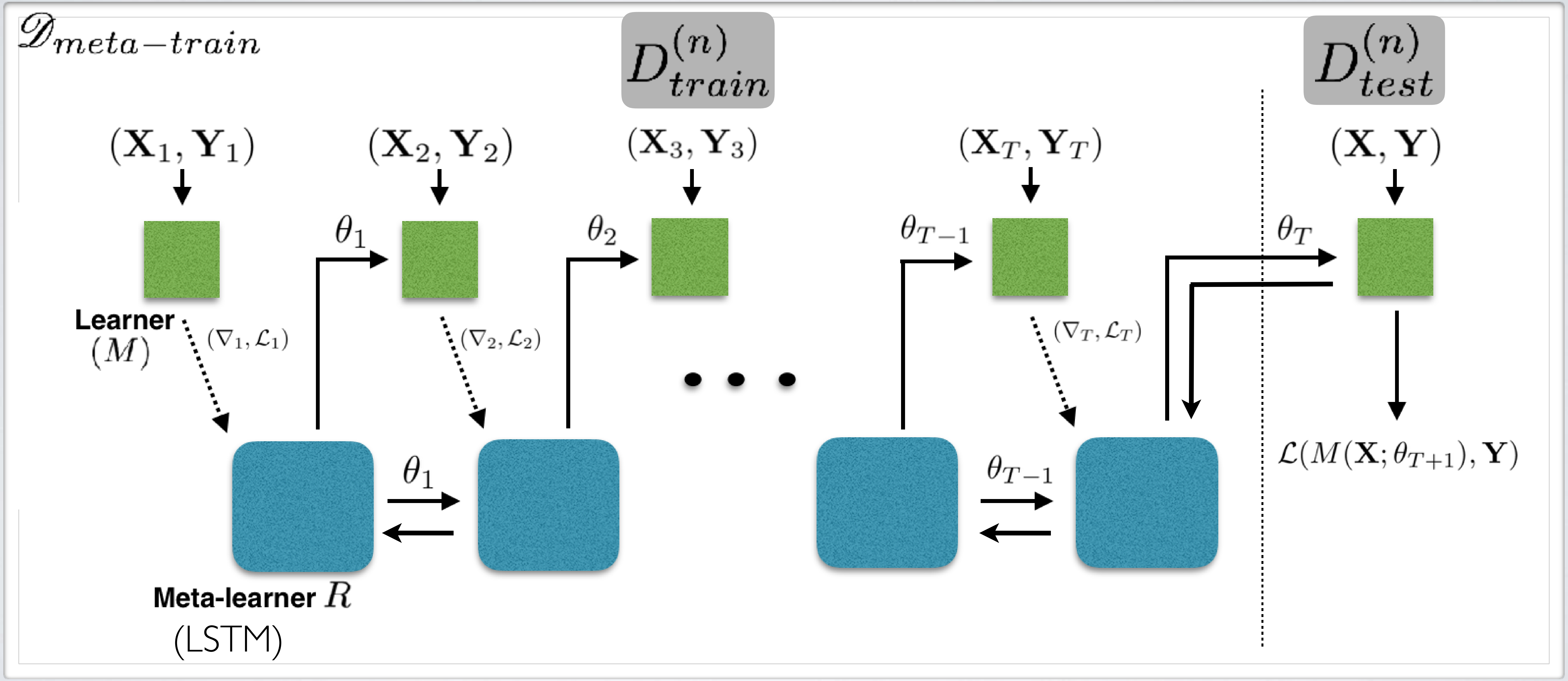
$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$

META-LEARNING UPDATES



META-LEARNING UPDATES



PSEUDOCODE

Algorithm 1 Train Meta-Learner

Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner M with parameters θ, Meta-Learner R with parameters Θ.

$\Theta_0 \leftarrow$ random initialization
for $d = 1, n$ **do**
    $D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$
    $\theta_0 \leftarrow c_0$ ▷ Intialize learner parameters
    for $t = 1, T$ **do**
        $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}
        $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch
        $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ ▷ Get output of meta-learner using Equation 2
        $\theta_t \leftarrow c_t$ ▷ Update learner parameters
    end for
    $\mathbf{X}, \mathbf{Y} \leftarrow D_{test}$
    $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ ▷ Get loss of learner on test batch
    Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters
end for

TO SUM UP

- We use our meta-learning LSTM to model parameter dynamics during training
  - LSTM parameters are shared across M's parameters (i.e. treated like a large minibatch)
  - learns c_0, which is like learning M's initialization
- Inputs to meta-learning LSTM are the loss and gradient of learner
  - we use the preprocessing proposed by Andrychowicz et al. (2016)
- It is trained to produce parameters that have low loss *on the corresponding test set*
  - possible thanks to backprop (though we ignore gradients through the inputs of the LSTM)
- Model M uses batch normalization
  - we are careful to avoid "leakage" between and within meta-sets

RELATED WORK

- Learning to learn using gradient descent (2001)
  Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell
  - LSTM-based meta-learner that isn't using M's gradients and was applied to synthetic learning problems

- Gradient-based hyperparameter optimization through reversible learning (2015)
  Dougal Maclaurin, David Duvenaud, and Ryan P Adams
  - learns the learning rates of each time-step of minibatch SGD

- Learning to learn by gradient descent by gradient descent (2016)
  Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas
  - LSTM outputs the update, instead of using its cell state explicitly for that

- Matching networks for one shot learning (2016)
  Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra
  - learns a metric that generalizes well to new dataset with meta-learning

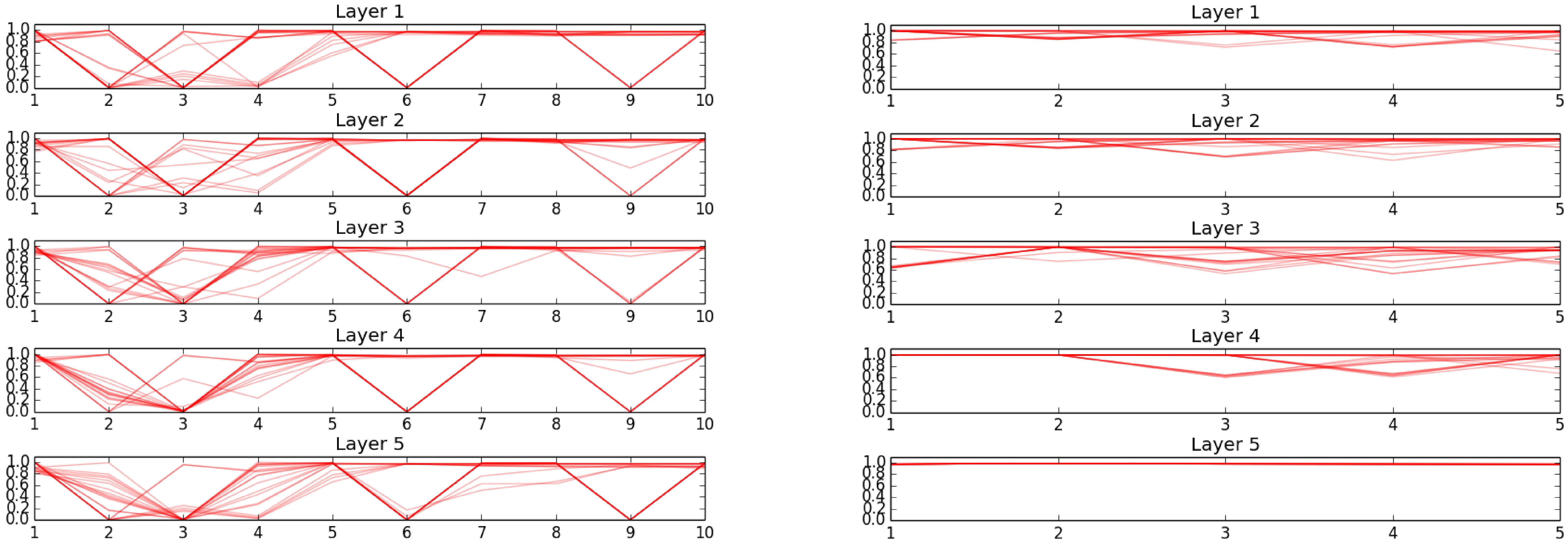
EXPERIMENT

- Mini-ImageNet
  - random subset of 100 classes (64 meta-training, 16 meta-validation, 20 meta-testing)
  - random sets D_{train} are generated by randomly picking 5 classes from class subset
  - model M is a small 4-layer CNN; meta-learner LSTM has 2 layers

Model	**5-class**	
	1-shot	5-shot
Baseline-finetune	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
Baseline-nearest-neighbor	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
Matching Network	$\mathbf{43.40 \pm 0.78\%}$	$51.09 \pm 0.71\%$
Matching Network FCE	$\mathbf{43.56 \pm 0.84\%}$	$55.31 \pm 0.73\%$
Meta-Learner LSTM (OURS)	$\mathbf{43.44 \pm 0.77\%}$	$\mathbf{60.60 \pm 0.71\%}$

EXPERIMENT

- Learned input gates



1-shot learning 5-shot learning

IN CONCLUSION

- We consider learning on multi-domain data, in the form of few-shot learning problem
  - rather than usual train/test dataset split, each dataset consists of a set of datasets
- We parameterize a training algorithm in the form of a LSTM
  - we train the meta-learner LSTM end-to-end on few-shot learning task
  - parameters of LSTM represent both the training algorithm and initialization of model M
- We evaluate our meta-learner model on mini-ImageNet dataset
  - the meta-learner model is competitive with state-of-the-art metric-learning methods

THANKS!