A few meta learning papers

Guy Gur-Ari

Machine Learning Journal Club, September 2017

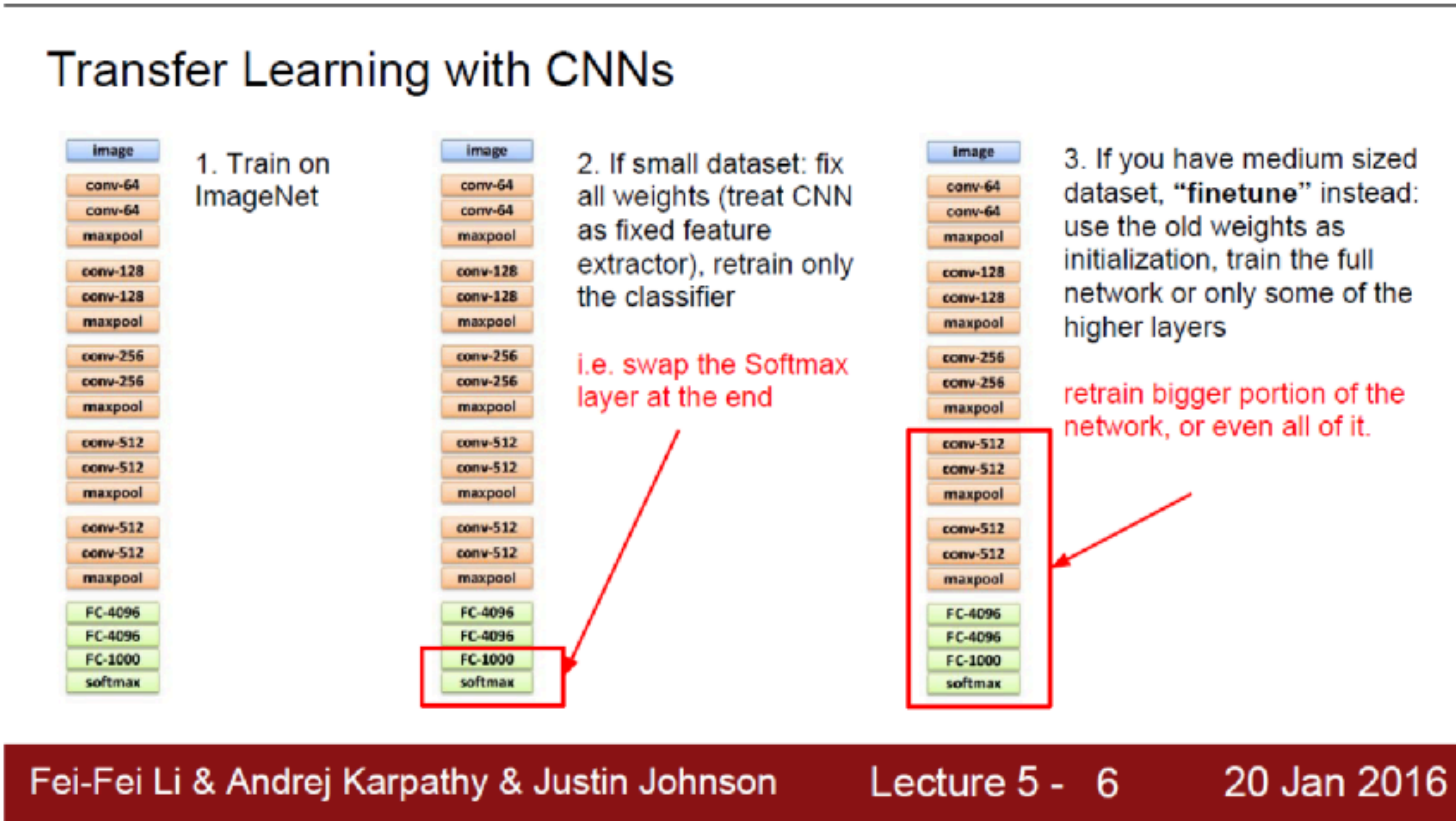
Meta Learning

- Mechanisms for faster, better adaptation to new tasks
  - ‘Integrate prior experience with a small amount of new information’
  - Examples: Image classifier applied to new classes, game player applied to new games, …
  - Related: single-shot learning, catastrophic forgetting
- Learning how to learn (instead of designing by hand)

Meta Learning

- Mechanisms for faster, better adaptation to new tasks
- Learning how to learn (instead of designing by hand)
- Each task is a single training sample
- Performance metric: Generalization to new tasks
- Higher derivatives show up, but first-order approximations sometimes work well

Transfer Learning (ad-hoc meta-learning)



Learning to learn by gradient descent by gradient descent

Andrychowicz et al.

1606.04474

Basic idea

Target ('optimizee') loss function

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \, .$$

Recurrent Neural Network *m* with parameters ϕ

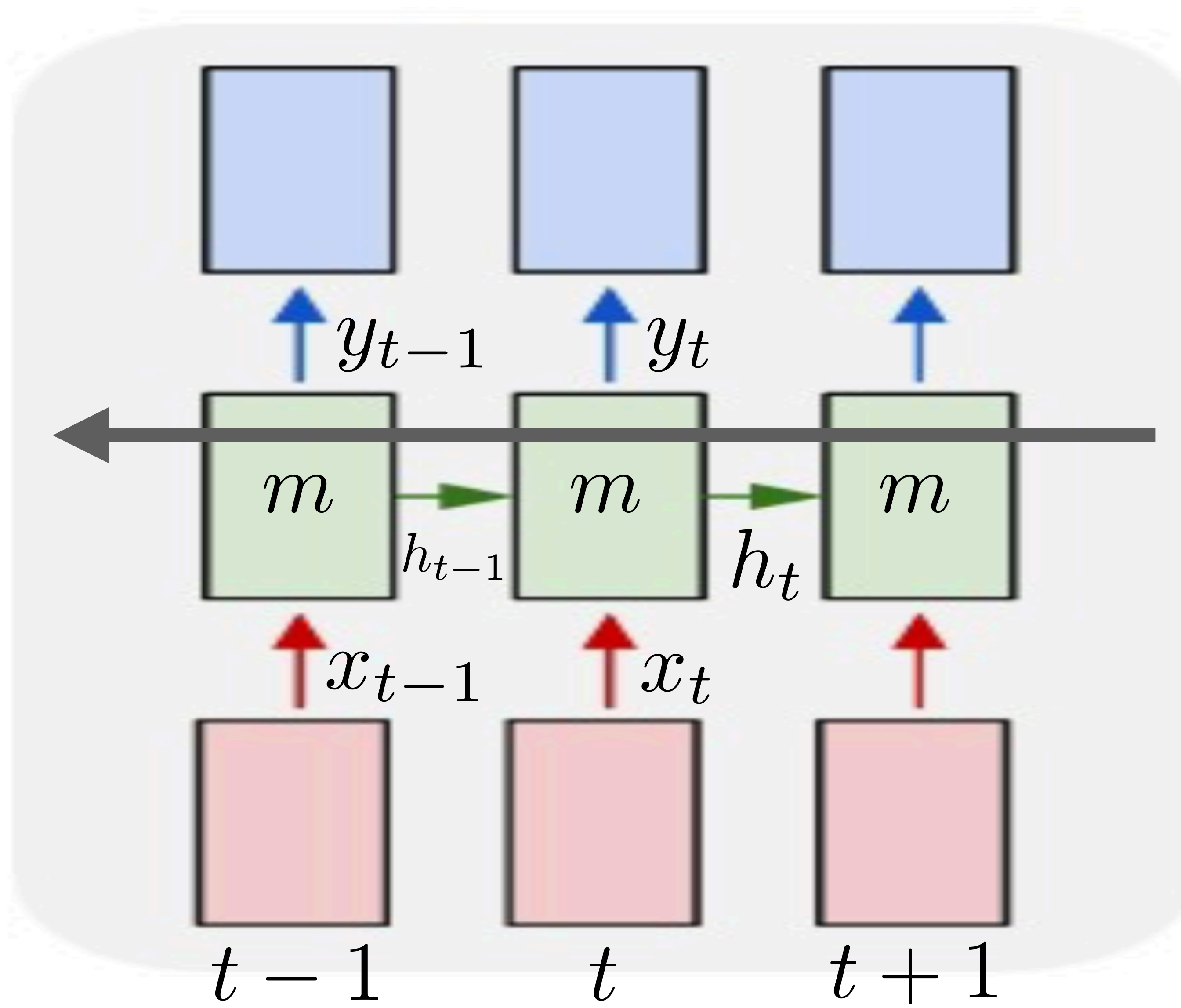
$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) \, .$$

Target parameters

Optimizer parameters

[1606.04474]

Vanilla RNN refresher



Backpropagation through time

$$h_t = \tanh\left(W_h h_{t-1} + W_x x_t\right)$$

$$y_t = W_y h_t$$

[Karpathy]

Meta loss function

Ideal

$$\mathcal{L}(\phi) = \mathbb{E}_f\left[f\left(\theta^*(f, \phi)\right)\right]$$

Optimal target parameters
for given optimizer

In practice

$$\mathcal{L}(\phi) = \mathbb{E}_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right] \qquad \text{where} \qquad \theta_{t+1} = \theta_t + g_t\,,$$

$$\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi)$$

RNN
(2-layer LSTM)

RNN hidden
state

$$\nabla_t = \nabla_\theta f(\theta_t) \qquad w_t \equiv 1$$

[1606.04474]

Meta loss function

$$\mathcal{L}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t f(\theta_t) \right] \qquad \text{where} \qquad \theta_{t+1} = \theta_t + g_t \,,$$

$$\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi)$$

$$\nabla_t = \nabla_\theta f(\theta_t) \qquad\qquad w_t \equiv 1$$

- Recurrent network can use trajectory information, similar to momentum

- Including historical losses also helps with backprop through time

[1606.04474]

Training protocol

- Sample a random task *f*

- Train optimizer on *f* by gradient descent
  (100 steps, unroll for 20)

$$\mathcal{L}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t f(\theta_t) \right] \quad \text{where} \quad \begin{aligned} \theta_{t+1} &= \theta_t + g_t \,, \\ \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} &= m(\nabla_t, h_t, \phi) \end{aligned}$$

- Repeat

[1606.04474]

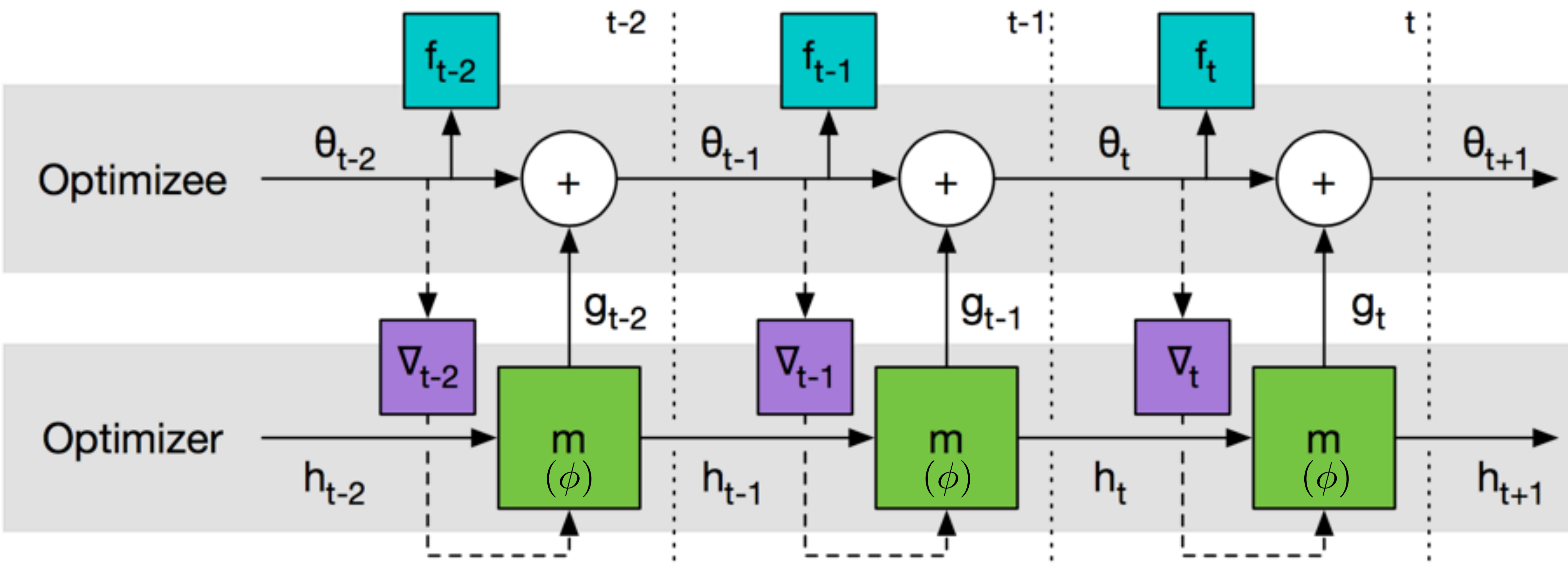
Test optimizer performance

- Sample new tasks
- Apply optimizer for some steps, compute average loss
- Compare with existing optimizers (ADAM, RMSProp)

[1606.04474]

Computational graph

$$\mathcal{L}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t f(\theta_t) \right] \qquad \text{where} \qquad \begin{aligned} \theta_{t+1} &= \theta_t + g_t \, , \\ \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} &= m(\nabla_t, h_t, \phi) \end{aligned}$$



Graph used for computing the gradient of the **optimizer** (with respect to ϕ)

[1606.04474]

Simplifying assumptions

- No 2nd order derivatives: $\nabla_\phi \nabla_\theta f = 0$
- RNN weights shared between target parameters
  - Result is independent of parameter ordering
  - Each parameter has separate hidden state

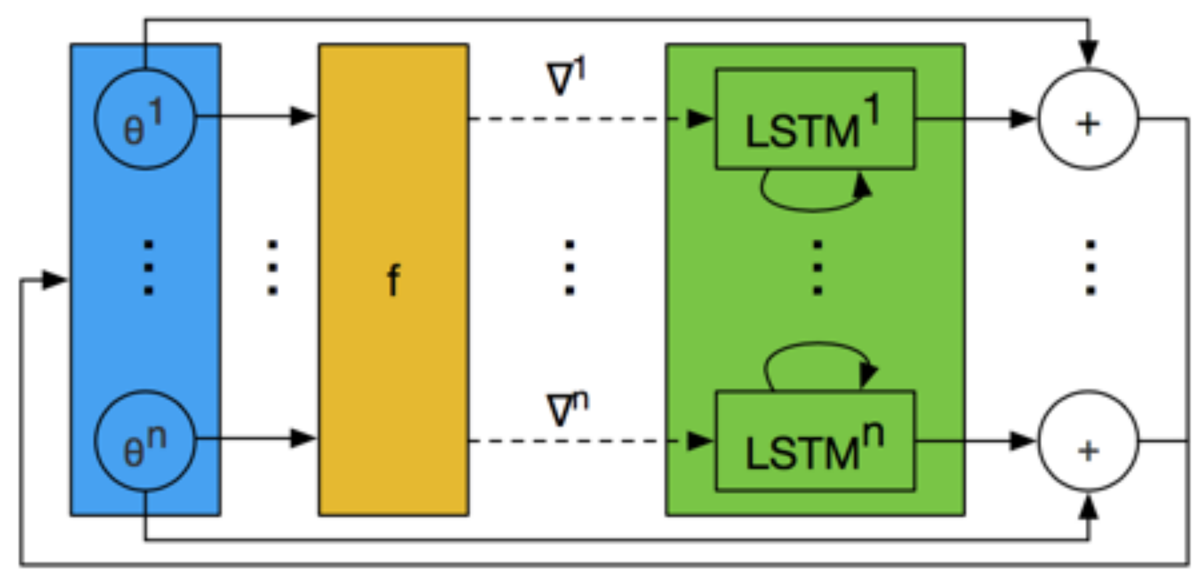


Figure 3: One step of an LSTM optimizer. All LSTMs have shared parameters, but separate hidden states.

[1606.04474]

Experiments

Variability is in initial target parameters
and choice of mini-batches

$$f(\theta) = \|W\theta - y\|_2^2$$

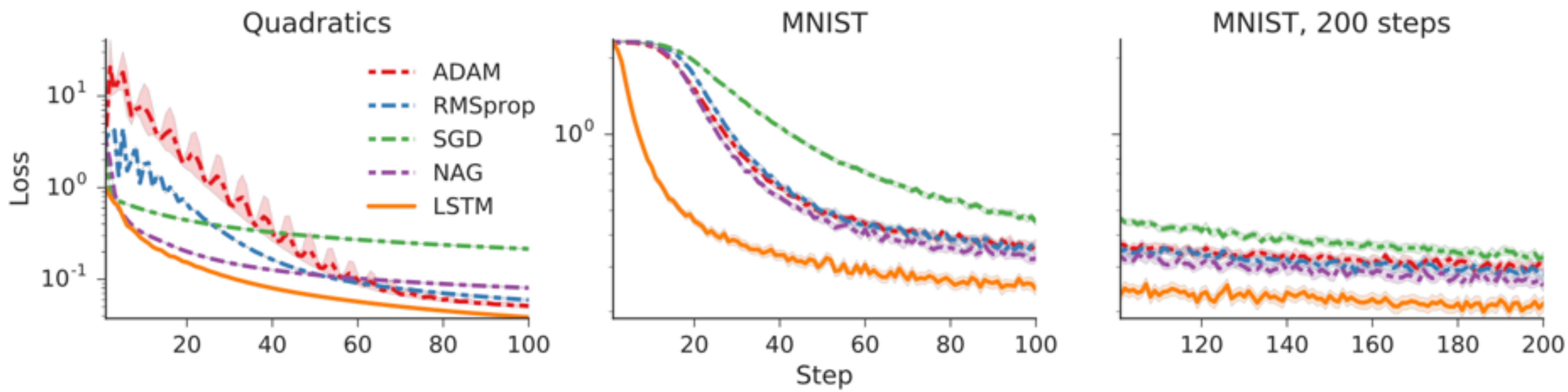


Figure 4: Comparisons between learned and hand-crafted optimizers performance. Learned optimizers are shown with solid lines and hand-crafted optimizers are shown with dashed lines. Units for the y axis in the MNIST plots are logits. **Left:** Performance of different optimizers on randomly sampled 10-dimensional quadratic functions. **Center:** the LSTM optimizer outperforms standard methods training the base network on MNIST. **Right:** Learning curves for steps 100-200 by an optimizer trained to optimize for 100 steps (continuation of center plot).

[1606.04474]

Experiments

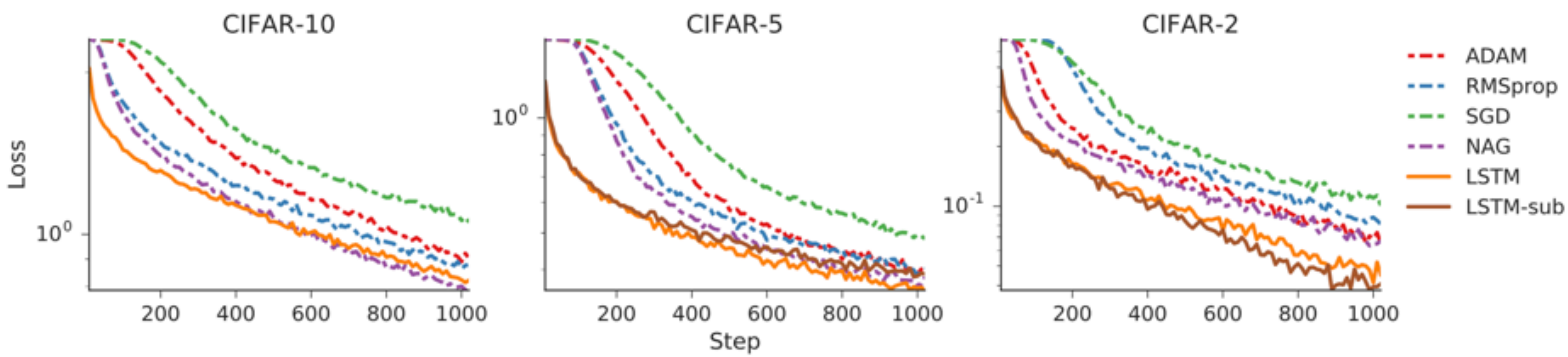


Figure 7: Optimization performance on the CIFAR-10 dataset and subsets. Shown on the left is the LSTM optimizer versus various baselines trained on CIFAR-10 and tested on a held-out test set. The two plots on the right are the performance of these optimizers on subsets of the CIFAR labels. The additional optimizer *LSTM-sub* has been trained only on the heldout labels and is hence transferring to a completely novel dataset.

Separate optimizers for convolutional and fully-connected layers

[1606.04474]

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Finn, Abbeel, Levine

1703.03400

Basic idea

- Start with a class of tasks $\mathcal{T}_i$ with distribution $p(\mathcal{T})$
- Train one model θ that can be quickly fine-tuned to new tasks ('few-shot learning')

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

- How? Explicitly require that a single training step will significantly improve the loss
- Meta loss function, optimized over θ:

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

[1703.03400]

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update (to avoid overfitting?)
9: **end for**
10: 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
11: **end while**

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1 - \mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)})) \tag{3}$$

[1703.03400]

Comments

- Can be adapted to any scenario that uses gradient descent (e.g. regression, reinforcement learning)

- Involves taking second derivative

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}\right)$$

- First-order approximation still works well

[1703.03400]

Regression experiment

Single task = compute sine with given underlying amplitude and phase

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)},\mathbf{y}^{(j)} \sim \mathcal{T}_i} \| f_\phi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)} \|_2^2, \qquad (2)$$

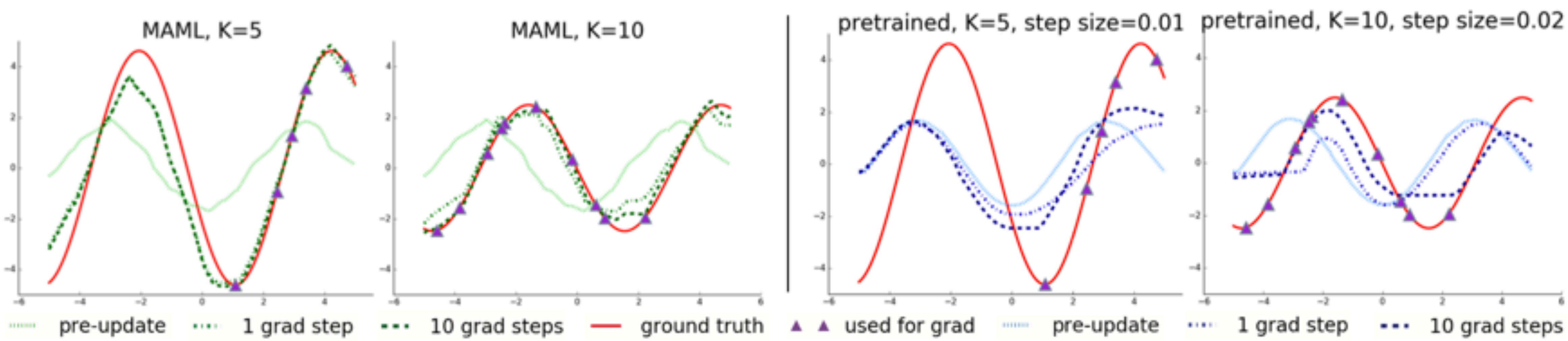


Figure 2. Few-shot adaptation for the simple regression task. Left: Note that MAML is able to estimate parts of the curve where there are no datapoints, indicating that the model has learned about the periodic structure of sine waves. Right: Fine-tuning of a model pretrained on the same distribution of tasks without MAML, with a tuned step size. Due to the often contradictory outputs on the pre-training tasks, this model is unable to recover a suitable representation and fails to extrapolate from the small number of test-time samples.

Model is FC network with 2 hidden layers

Pretrained = compute a single model on many tasks simultaneously

[1703.03400]

Classification experiment

Table 1. Few-shot classification on held-out Omniglot characters (top) and the MiniImagenet test set (bottom). MAML achieves results that are comparable to or outperform state-of-the-art convolutional and recurrent models. Siamese nets, matching nets, and the memory module approaches are all specific to classification, and are not directly applicable to regression or RL scenarios. The $\pm$ shows 95% confidence intervals over tasks. Note that the Omniglot results may not be strictly comparable since the train/test splits used in the prior work were not available. The MiniImagenet evaluation of baseline methods and matching networks is from Ravi & Larochelle (2017).

	5-way Accuracy		20-way Accuracy	
Omniglot (Lake et al., 2011)	1-shot	5-shot	1-shot	5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	**89.7 ± 1.1%**	**97.5 ± 0.6%**	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	**98.7 ± 0.4%**	**99.9 ± 0.1%**	**95.8 ± 0.3%**	**98.9 ± 0.2%**

	5-way Accuracy	
MiniImagenet (Ravi & Larochelle, 2017)	1-shot	5-shot
fine-tuning baseline	28.86 ± 0.54%	49.79 ± 0.79%
nearest neighbor baseline	41.08 ± 0.70%	51.04 ± 0.65%
matching nets (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, first order approx. (ours)	**48.07 ± 1.75%**	**63.15 ± 0.91%**
MAML (ours)	**48.70 ± 1.84%**	**63.11 ± 0.92%**

Each classification class is a single task

[1703.03400]

RL experiment

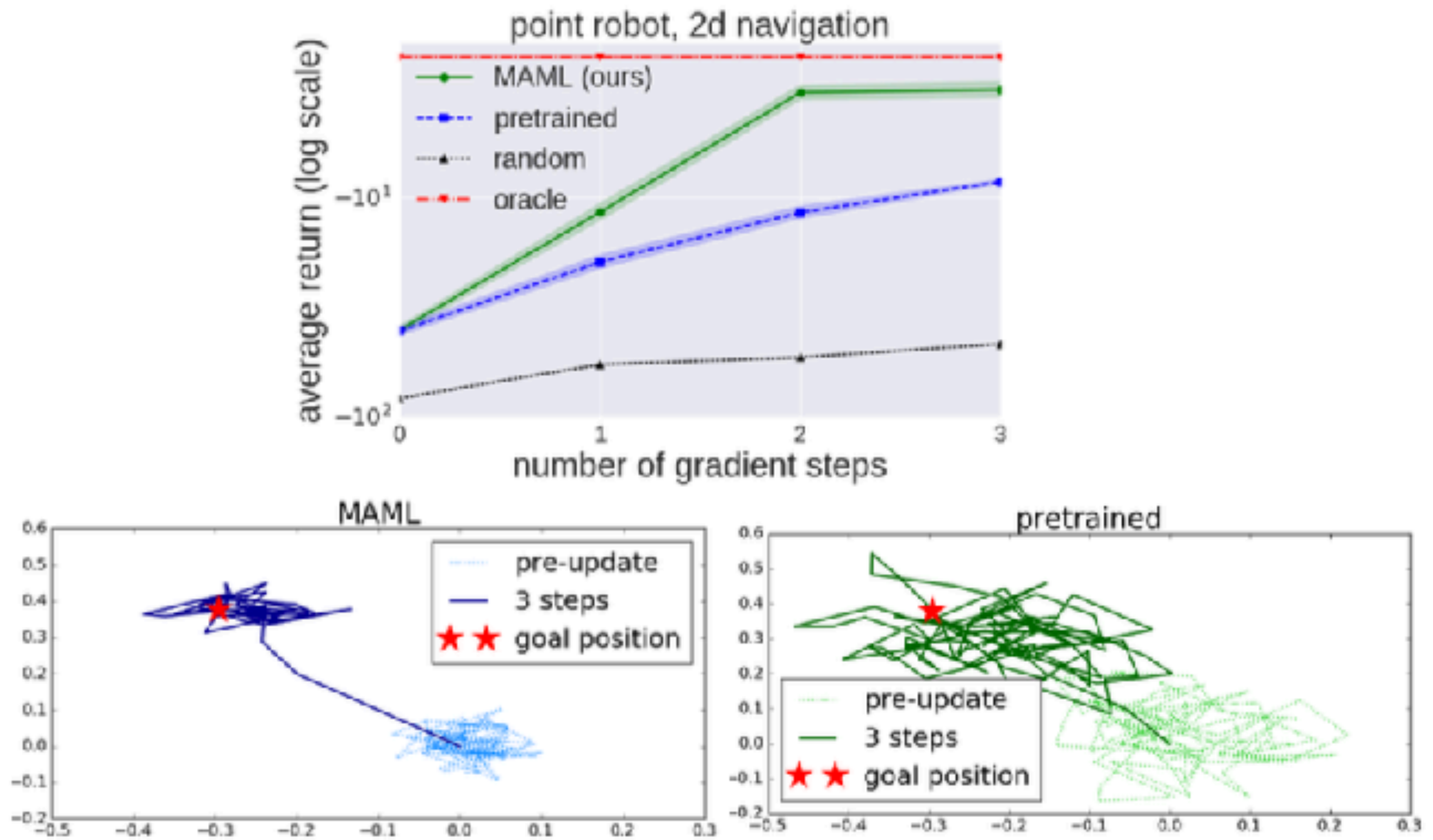


Figure 4. Top: quantitative results from 2D navigation task, Bottom: qualitative comparison between model learned with MAML and with fine-tuning from a pretrained network.

Reward = negative square distance from goal position.
For each task, goal is placed randomly.

[1703.03400]

Overcoming catastrophic forgetting in neural networks

Kirkpatrick et al.

1612.00796

Basic idea

- Catastrophic forgetting: When a model is trained on task A followed by task B, it typically forgets A

- Idea: After training on A, freeze the parameters that are important for A

hyperparameter

optimal parameters for task A

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta^*_{A,i})^2$$

diagonal of Fisher information matrix

$$F_i \approx \frac{\partial^2 \mathcal{L}_A}{\partial \theta_i^2}$$

[1612.00796]

Why Fisher information?

$$\begin{aligned}\mathcal{L}(\theta) &= -\log(\theta|D_A, D_B) \\ &= -\log p(D_B|\theta) - \log p(\theta) - \log p(D_A|\theta) + \log p(D_A, D_B) \\ &\sim \mathcal{L}_B(\theta) - \log p(D_A|\theta)\end{aligned}$$

$$-\log p(D_A|\theta) = -\sum_i \log p_\theta(x_i) \sim -\sum_x p_A(x) \log p_\theta(x)$$

now suppose $p_{\theta_*} = p_A$ **then**

$$-\sum_x p_{\theta_*}(x) \log p_{\theta_* + d\theta}(x) = S(p_{\theta_*}) + \frac{1}{2} d\theta^T F d\theta + \cdots$$

$$F_{ij} = E_{x\sim p_\theta}[\nabla_{\theta_i} \log p_\theta(x) \nabla_{\theta_j} \log p_\theta(x)]$$

Why Fisher information?

$$\mathcal{L}(\theta) \sim \mathcal{L}_B(\theta) + \frac{1}{2} d\theta^T F d\theta$$

$$d\theta = \theta - \theta_A^*$$

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta_{A,i}^*)^2$$

MNIST experiment

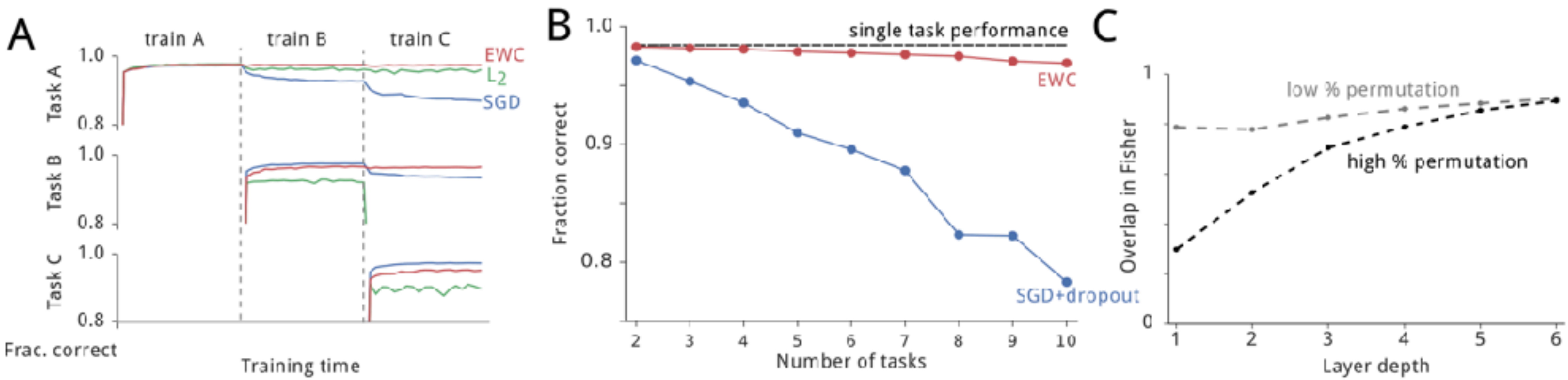


Figure 2: Results on the permuted MNIST task. A: Training curves for three random permutations A, B and C using EWC(red), L_2 regularization (green) and plain SGD(blue). Note that only EWC is capable of mantaining a high performance on old tasks, while retaining the ability to learn new tasks. B: Average performance across all tasks using EWC (red) or SGD with dropout regularization (blue). The dashed line shows the performance on a single task only. C: Similarity between the Fisher information matrices as a function of network depth for two different amounts of permutation. Either a small square of 8x8 pixels in the middle of the image is permuted (grey) or a large square of 26x26 pixels is permuted (black). Note how the more different the tasks are, the smaller the overlap in Fisher information matrices in early layers.