Learning to learn by gradient descent by gradient descent

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, Nando de Freitas
presented bt Francesco Demelas

January 4, 2024

Unrolling Methods

Question

How the design of an optimization problem can be cast as learning algorithm?

Answer

Authors develop a procedure to construct a learning algorithm that performs well on a particular class of optimization problem.

In this context **examples** are problem istances and **generalization** corresponds to the ability of transfer knowledge to different problems (transfer learning).

Setting

Machine Learning Task - Optimization Problem

$$\theta^* \in \arg\min_{\theta \in \Theta} f(\theta)$$

where θ are the **optimizee parameters**.
We can solve it using Stocastic Gradient Descent:

SGD - Standard Sequential Updates

$$\theta_{t+1} = \theta_t - \alpha_t \Delta f(\theta_t)$$

Main Idea

Learning Update Rules

Instead of considering

SGD - Standard Sequential Updates

$$\theta_{t+1} = \theta_t - \alpha_t \Delta f(\theta_t)$$

they propose

Learn Updates Rules

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

where g denotes the **optimizer** used as update rule and it depends by 9its own parameters ϕ.
g will be a Recurrent Neural Netowork (RNN)

(Preistorical) Related Works

1998: meta-learning (learning to learn) survey [1].

2016: building blocks in artificial intelligence [2].

2016: see multi-task learning as generalization, by directly train a base learner (rather than a training algorithm) [3].

1992,1993 the most general meta-learning approach: consider networks that are able to modify their own weigths. [4] [5] Differentiable end-to-end, but the learning rules are harder to train.

---

[1] S. Thrun and L. Pratt. Learning to learn. Springer Science & Business Media.

[2] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. arXiv.

[3] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In International Conference on Machine Learning.

[4] J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Computation.

[5] J. Schmidhuber. A neural network that embeds its own meta-levels. In International Conference on Neural Networks.

1997: use Success Story Algorithm (rather than gradient descent) to modify its search strategy.[6]

2016: reiforcement learning to schedule step sizes. [7]

1990, 1995 learn updates which avoids back-propagation by using simple parametric rules. Learning to learn **without** gradient descent by gradient descent. [8] [9]

1990,1999 fixed-weight RNN can exhibit dynamic behavior without need to modify their network weights [10] [11]

---

[6] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. Machine Learning.

[7] C. Daniel, J. Taylor, and S. Nowozin. Learning step size controllers for robust neural network training. In Association for the Advancement of Artificial Intelligence.

[8] S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for ANNs. Neural Processing Letters.

[9] Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. Université de Montréal.

[10] N. E. Cotter and P. R. Conwell. Fixed-weight networks can learn. In International Joint Conference on Neural Networks.

[11] A. S. Younger, P. R. Conwell, and N. E. Cotter. Fixed-weight on-line learning. Transactions on Neural Networks.

1998, 1992, 1999 Similar to the previous in a filtering context [12] which is directly related to simple multi-timescale optimizers [13] [14]

2001 connects these different threads of research by allowing for the output of backpropagation from one network to feed into an additional learning network, with both networks trained jointly. [15] [16]

This work: builds on the last work by modifying the network architecture of the optimizer in order to scale this approach to larger neural-network optimization problems.

---

[12] L. A. Feldkamp and G. V. Puskorius. A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering, and classification. Proceedings of the IEEE.

[13] R. S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In Association for the Advancement of Artificial Intelligence.

[14] N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. In International Conference on Artificial Neural Networks, volume 2, pages 569–574, 1999

[15] A. S. Younger, S. Hochreiter, and P. R. Conwell. Meta-learning with backpropagation. In International Joint Conference on Neural Networks, 2001

[16] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In International Conference on Artificial Neural Networks, pages 87–94. Springer, 2001.

Learning Task

Final **Optimizee** parameters

$$\theta^* \equiv \theta^*(f, \phi)$$

where ϕ are the **optimizer** parameters.

What does it means for an optimizer to be good?

Loss function

Given a distribution of functions f we consider the loss:

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

Trajectory Dependent Loss

Loss function

Given a distribution of functions f we consider the loss:

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

for the *training optimizer* is convenient to have an objective that depends on the entire trajectory of optimization, for some **time horizon** T:

Trajectory Loss

$$\mathcal{L}(\phi) \equiv \mathbb{E}_f[\sum_{t=1}^{T} w_t f(\theta_t)]$$

where:

- $\theta_{t+1} = \theta_t + g_t$
- $\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\Delta_t, h_t, \phi)$
- $\Delta_t = \Delta_\theta f(\theta_t)$
- m will be a RNN model.

Structure of the network *m*

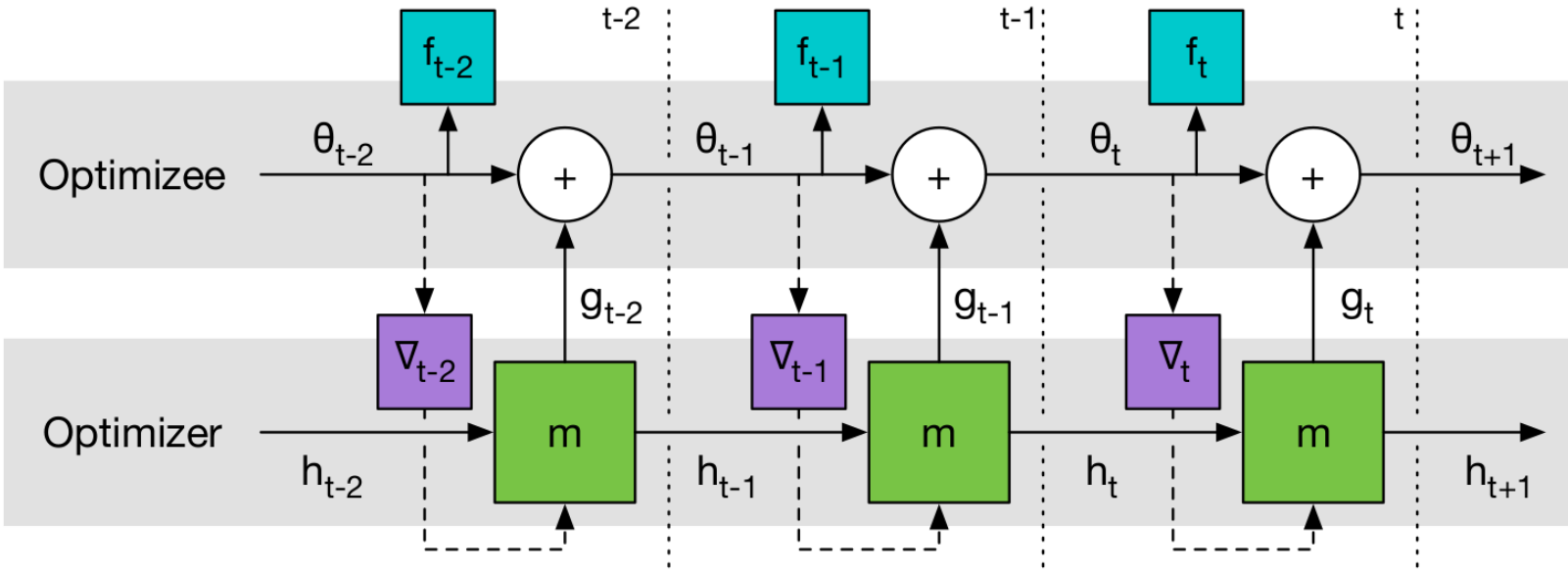

Figure 2: Computational graph used for computing the gradient of the optimizer.

LSTM will enable to automatically integrate the history of the gradients (like momentum).

Challenge

Using RNN in this setting we want to be able to optimize at least ten of thousand of parameters, but a fully-connected RNN require an enormous amount of parameters.

Proposed Solution:

Use an **optimizer** m which operates coordinate-wise on the parameters of the objective function. LSTM model in figure.

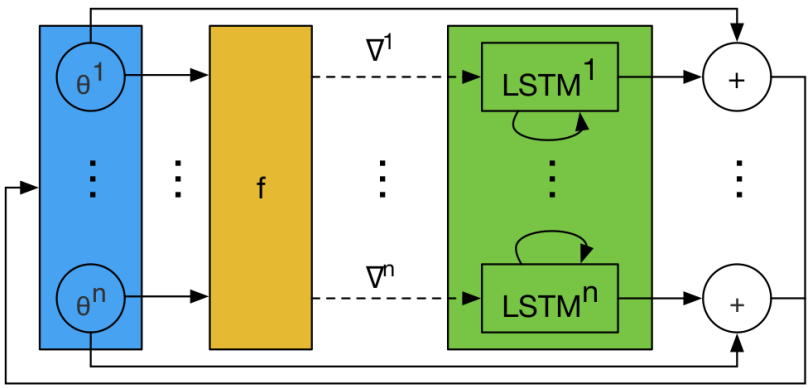

Figure 3: One step of an LSTM optimizer. All LSTMs have shared parameters, but separate hidden states.

- Different behavior on each coordinate using serparate activation functions for each objective function parameters.
- Use a small network has the effect of making the optimizer invariant to the order of parameters in the network.

Preprocessing and Postprocessing

- Rescale input and ouputs of LSTMS using a suitable constant
- In the Appendix A they present a more robust preprocessing technique which provides sligly better performances (but we'll not discuss this here).

In all the experiments

- LST with 2 hidden layers and 20 units in each layer
- ADAM with random search to choose the learning rate
- BASElines: SGD, RMSprop, ADAM, NAG, **LSTM**

Quadratic Functions

10-dimensional quadratic functions

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

where $y \in \mathbb{R}^{10}$ and $W \in \mathbb{R}^{10\times 10}$ are sampled using an IID Gaussian distribution.

- each function is optimized for 100 steps and unrolled for 20 steps.
- no preprocessing or post-processing

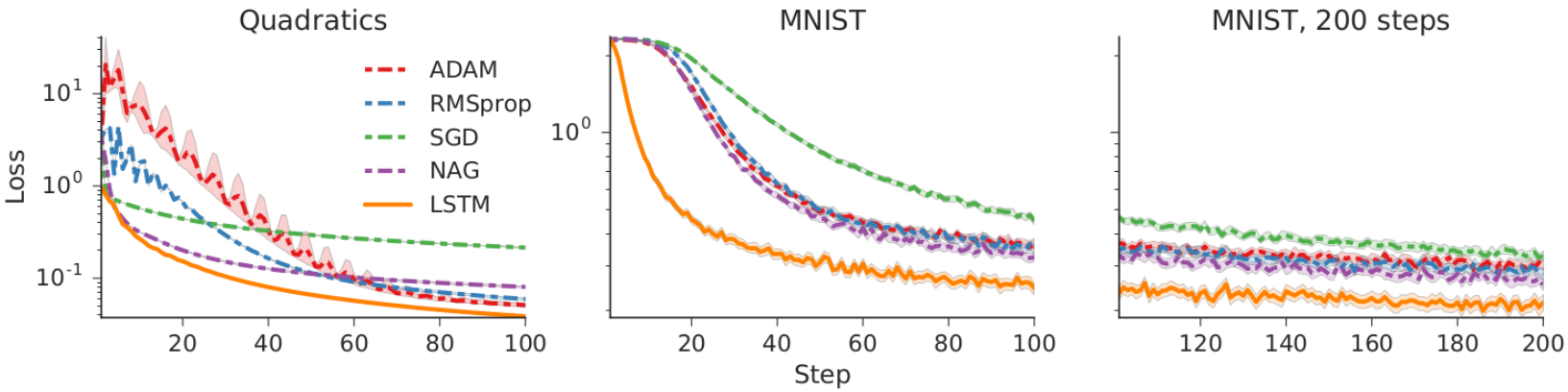

Figure 4: Comparisons between learned and hand-crafted optimizers performance. Learned optimizers are shown with solid lines and hand-crafted optimizers are shown with dashed lines. Units for the y axis in the MNIST plots are logits. **Left:** Performance of different optimizers on randomly sampled 10-dimensional 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

Small Network on MNIST

MNIST if a classic dataset used for the recognition of number from image. Here we use as $f(\theta)$ the cross-entropy.

- $\partial f(\theta)/\partial\theta$ is estimated using random batches of 128 samples.
- as (optimizee) model we consider a MLP with 1 hidden layer, 20 units and sigmoid activation function
- run for 100 steps and unrolled for 20 steps.

Also considering the (optimizer) model trained on the previous problem (quadratic loss) it performs well on this problem, outperforming hand-crafted methods.

Generalization to different architectures

Consider (for MNIST) a model trained with the previous model and test on the following models:

1. MLP with 40 units (instead of 20)
2. MLP with 2 hidden layers (instead of 1)
3. Relu activation function (instead of sigmoid)

LSTM still performs well, but for the Relu it is not able to generalize as the learning is associated to a different dynamic.

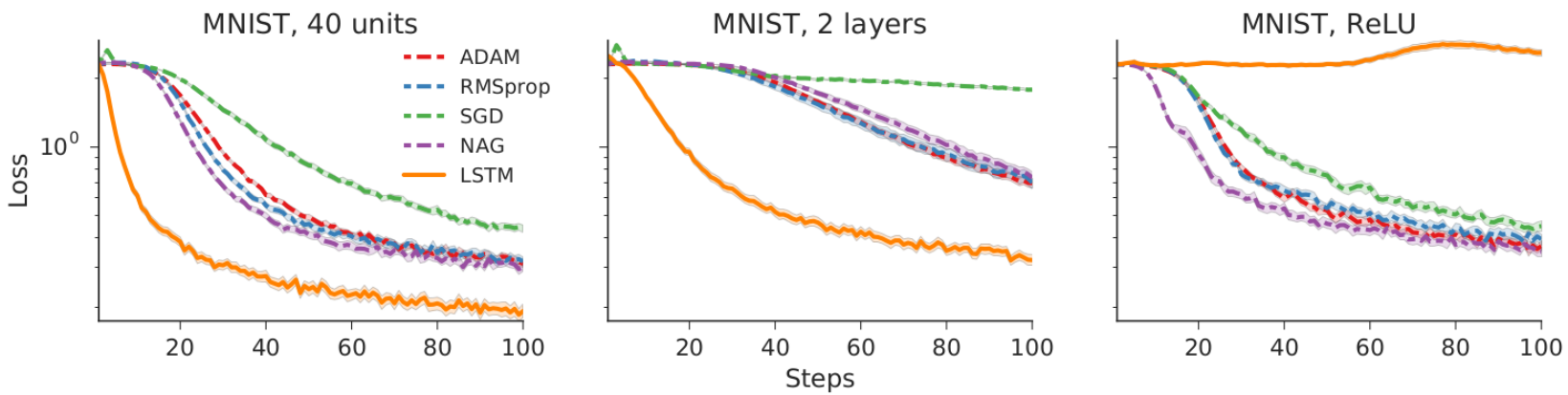

Figure 5: Comparisons between learned and hand-crafted optimizers performance. Units for the y axis are logits. **Left:** Generalization to the different number of hidden units (40 instead of 20). **Center:** Generalization to the different number of hidden layers (2 instead of 1). This optimization problem is very hard, because the hidden layers are very narrow. **Right:** Training curves for an MLP with 20 hidden units using ReLU activations. The LSTM optimizer was trained on an MLP with sigmoid activations.

Systematical Variation of the tested architecture

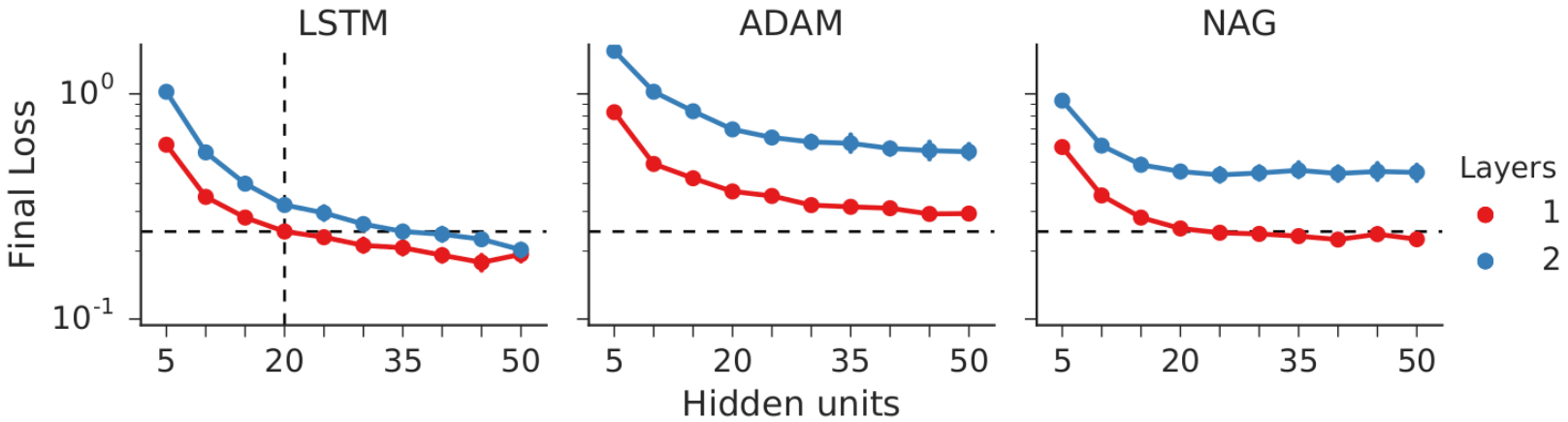

Figure 6: Systematic study of final MNIST performance as the optimizee architecture is varied, using sigmoid non-linearities. The vertical dashed line in the left-most plot denotes the architecture at which the LSTM is trained and the horizontal line shows the final performance of the trained optimizer in this setting.

as the models are sufficiently similars we can see that some models outperforms the baseline model.

Convolutional Network on CIFAR-10

The CIFAR model is qan images dataset used to recognize clothest.

- as (optimizee) model we consider a model with 3 hidden convolutional layers with max-pooling, followed by a fully connected layer with 32 units.
- All non.linearities are: Relu activation and batch normalization.
- as (optimizer) model the LSTM used before was not sufficient

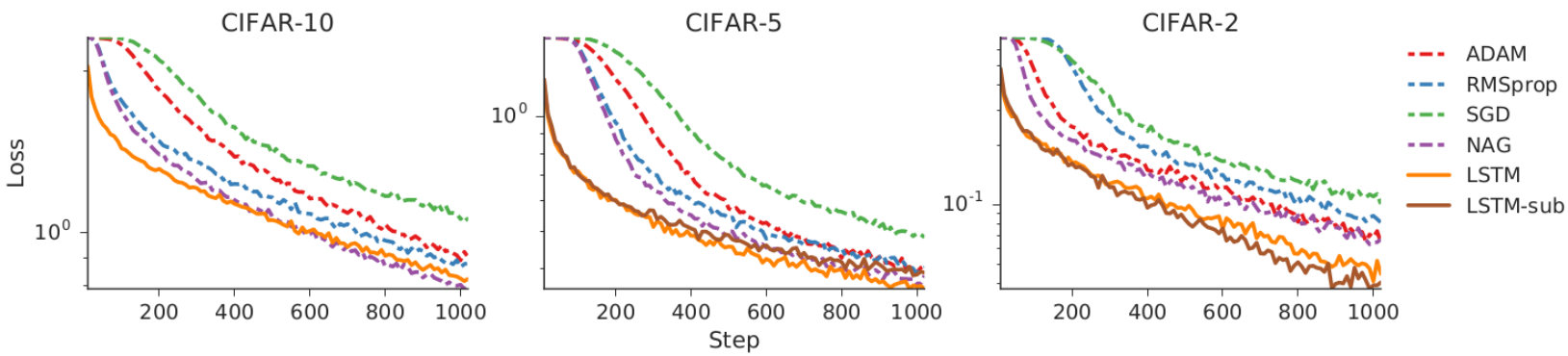

Figure 7: Optimization performance on the CIFAR-10 dataset and subsets. Shown on the left is the LSTM optimizer versus various baselines trained on CIFAR-10 and tested on a held-out test set. The two plots on the right are the performance of these optimizers on subsets of the CIFAR labels. The additional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

Neural Art

Each Neural Art problem starts from a content image c and a style image s and is given by:

$$f(\theta) = \alpha \mathcal{L}_{content}(c, \theta) + \beta \mathcal{L}_{style}(s, \theta) + \gamma \mathcal{L}_{reg}(\theta)$$

the minimizer of f is called *styled image*.[17]

- 1 style and 1800 content images taken from ImageNet
- select 100 content images for testing and 20 content images for validation of trained optimizers
- train the optimizer on 64x64 content images from ImageNet and one fixed style
- test how well it generalizes to a different style image and higher resolution (128x128)
- Each image was optimized for 128 steps and trained optimizers were unrolled for 32 steps
- preprocessing described in Appendix A and no postprocessing

[17] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. arXiv Report 1508.06576, 2015.

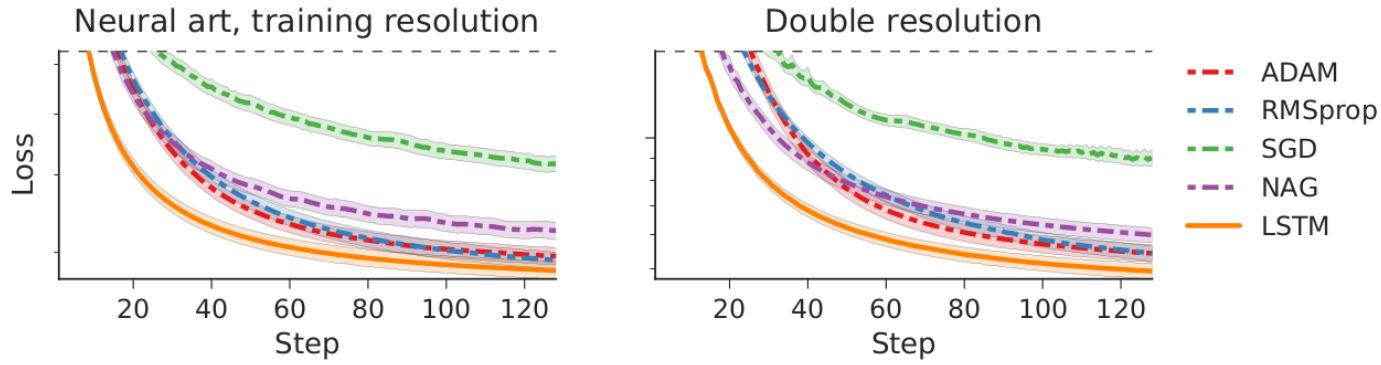

Figure 8: Optimization curves for Neural Art. Content images come from the test set, which was not used during the LSTM optimizer training. Note: the y-axis is in log scale and we zoom in on the interesting portion of this plot. **Left:** Applying the training style at the training resolution. **Right:** Applying the test style at double the training resolution.

The LSTM optimizer outperforms all standard optimizers if the resolution and style image are the same as the ones on which it was trained. Moreover, it continues to perform very well when both the resolution and style are changed at test time.

Figure 9: Examples of images styled using the LSTM optimizer. Each triple consists of the content image (left), style (right) and image generated by the LSTM optimizer (center). **Left:** The result of applying the training style at the training resolution to a test image. **Right:** The result of applying a new style to a test image at double the resolution on which the optimizer was trained.

Conclusions

- cast the design of optimization algorithms as a learning problem, which enables us to train optimizers that are specialized to particular classes of functions.
- learned neural optimizers compare favorably against state-of-the-art optimization methods used in deep learning.
- remarkable degree of transfer:
  - LSTM optimizer trained on 12,288 parameter neural art tasks being able to generalize to tasks with 49,152 parameters, different styles, and different content images all at the same time
  - transferring to different architectures in the MNIST task.
  - on the CIFAR image labeling task show that the LSTM optimizers outperform hand- engineered optimizers when transferring to datasets drawn from the same data distribution.