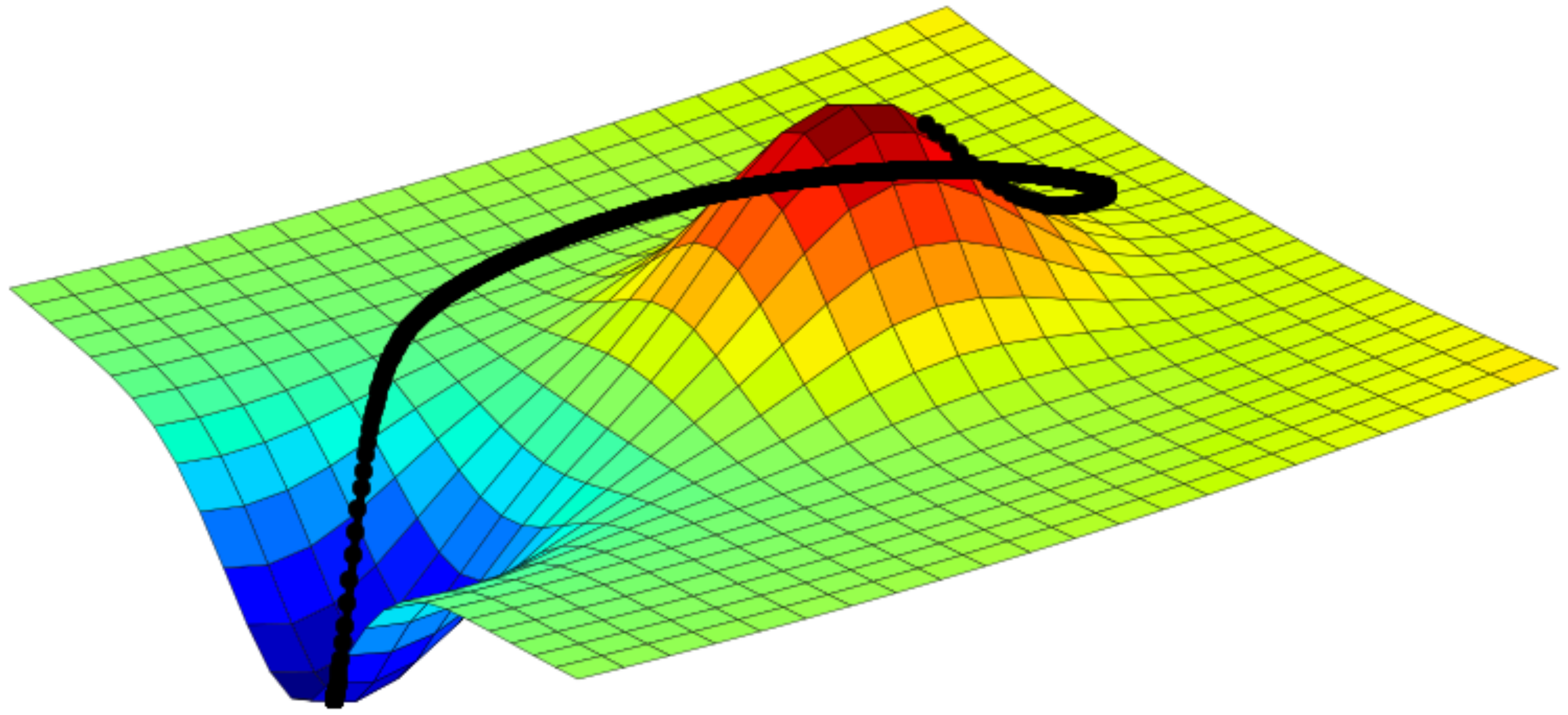
Learning to learn by gradient descent by gradient descent



Presentation by Quan Bach

Paper Authors: Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, Nando de Freitas

Introduction

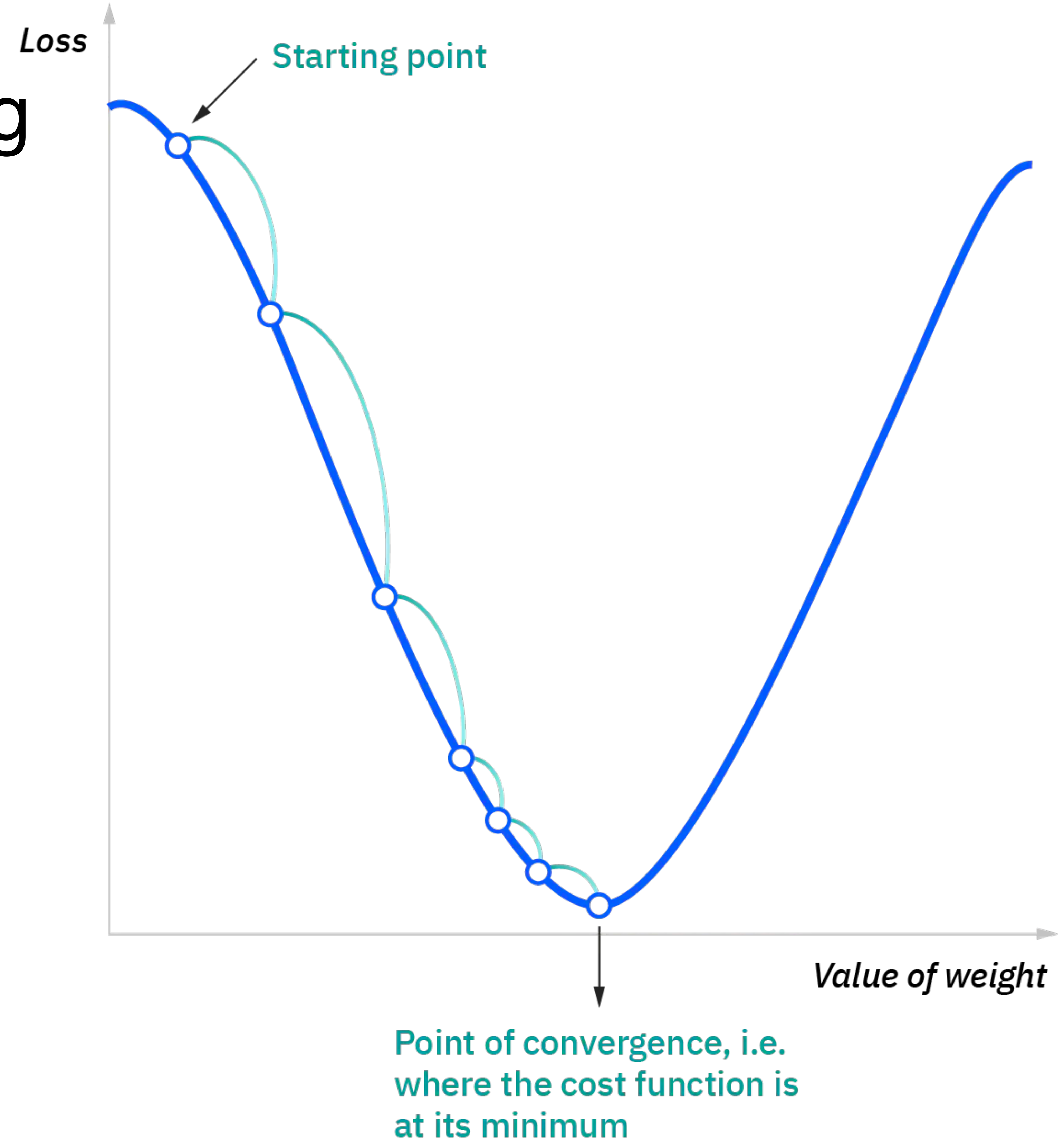
gradient descent

- Tasks in ML can be expressed as the problem of optimizing an objective function $f(\theta)$ defined over some domain $\theta \in \Theta$
- The goal is to find the minimizer $\theta^* = argmin_{\theta \in \Theta} f(\theta)$
- This can be solved by gradient descent if the function is differentiable, resulting in a sequence of updates:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

α : *learning rate*

∇ : *gradient*



Introduction
problem statement

- Optimization algorithms are still designed by hand
- Modern work in optimization is fragmented; designing update rules tailored to specific classes of problems
    - Deep learning community: high-dimensional, non-convex optimization problems (momentum, Rprop, Adagrad, RMSprop)
    - Other communities favor other approaches
- *No Free Lunch Theorems for Optimization*

  specialization to a subclass of problems is in fact the only way that improved performance can be achieved in general

Proposal

Replace hand-designed update rules with a learned update rule

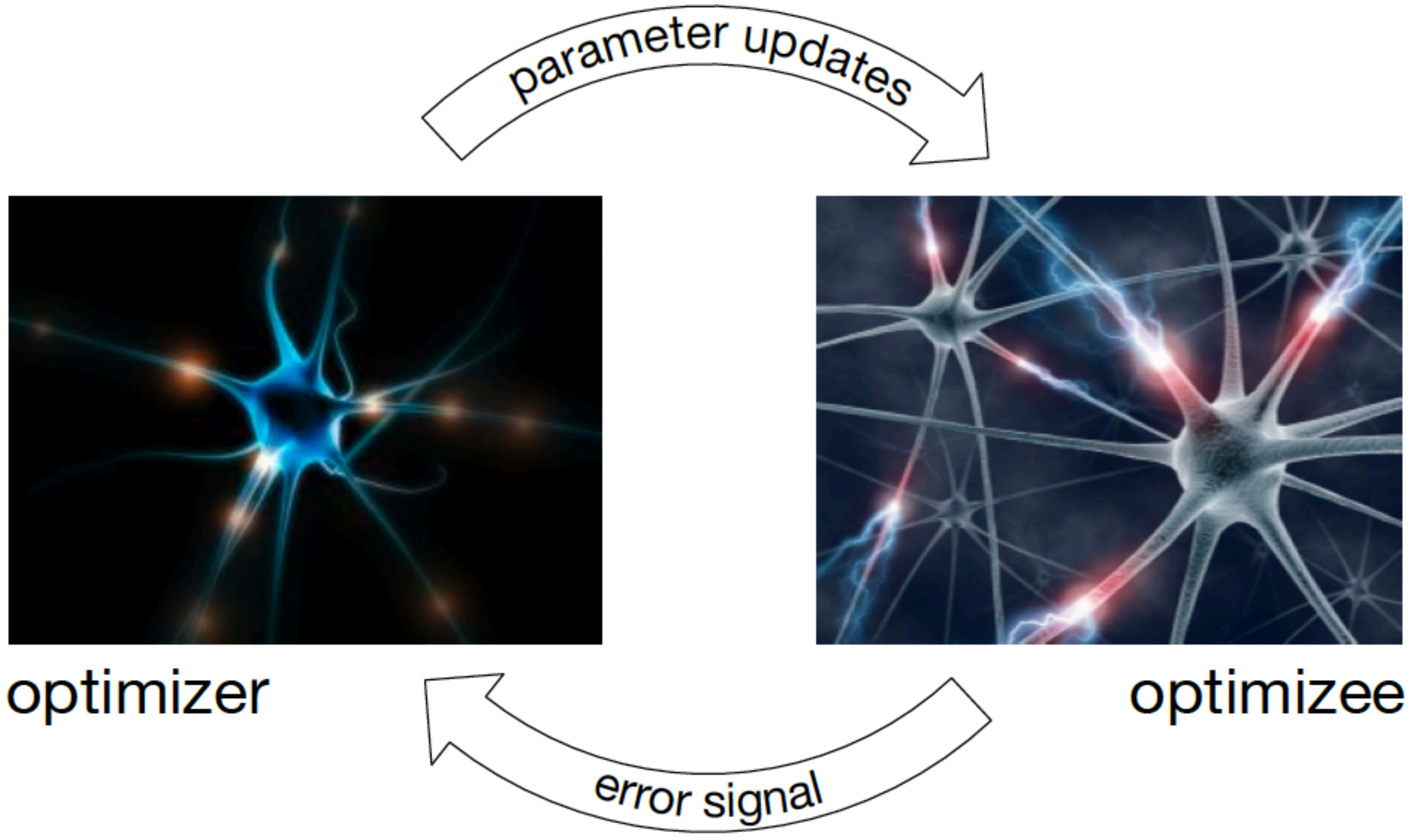


Figure 1: The optimizer (left) is provided with performance of the optimizee (right) and proposes updates to increase the optimizee's performance. [photos: Bobolas, 2009, Maley, 2011]

Key Main Ideas

- Casting algorithm design as a learning problem
- A learned update rule

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi), \qquad g : optimizer, f : optimizee$$

- Model the update rules g using a recurrent neural network (RNN) which maintains its own state and hence dynamically updates as a function of its iterates

Main Contribution of the Paper

Methods

- Allowing for the output of back-propagation from one network to feed into an additional *learning* network, with both network trained jointly.
- Modification to the network architecture of the optimizer to scale to larger neural-network optimization problem
- Learning to learn a.k.a meta-learning with recurrent neural networks

Main Contribution of the Paper

Algorithms & Models

Take the update step g_t to be the output of a RNN m, parameterized by ϕ, whose state will be denoted as h_t

For some horizon time T, given a distribution of functions f the expected loss is written as:

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

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

$w_t \in \mathbb{R}_{\geq 0}$: *are arbitrary weights associated with each time step*

$Note : \nabla_t = \nabla_\theta f(\theta_t)$

Minimizing the value of $\mathcal{L}(\phi)$ using gradient descent on ϕ

Main Contribution of the Paper

Computational Graph

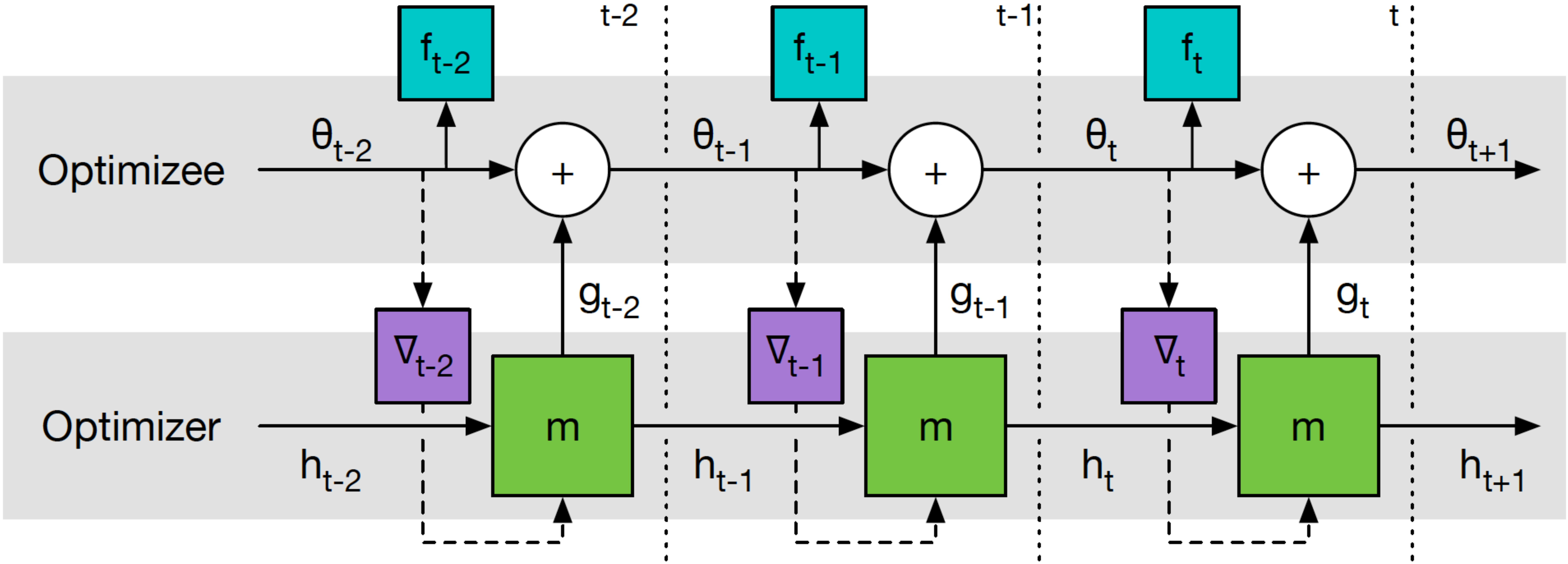


Figure 2: Computational graph used for computing the gradient of the optimizer.

Main Contribution of the Paper

Long Short Term Memory (LSTM) optimizer

- RNNs have at least tens of thousands of parameters
- Solution: optimizer m operates coordinate-wise on the parameters (*coordinatewise network architecture*)
- LSTM: is a RNN which has feedback connections

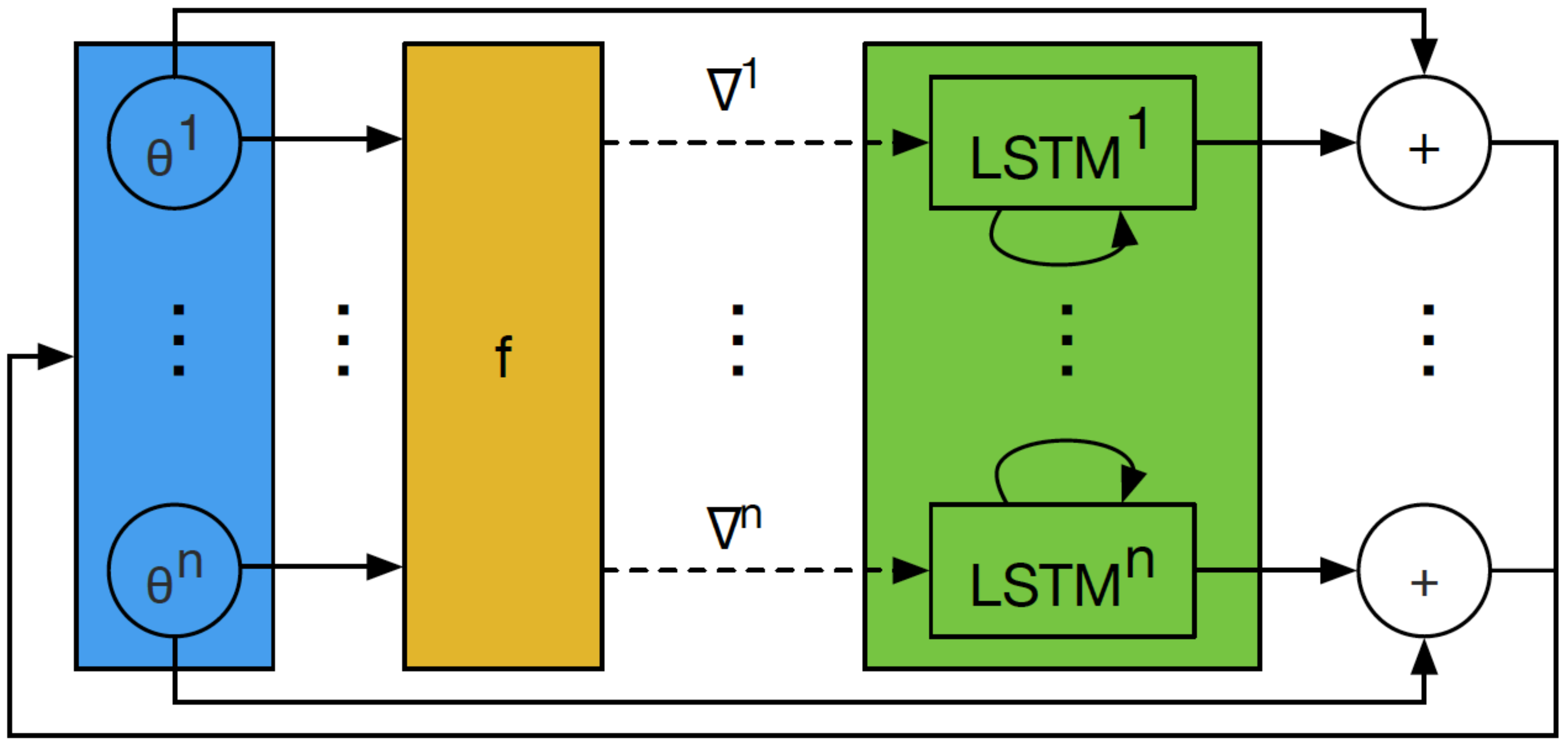


Figure 3: One step of an LSTM optimizer. All LSTMs have shared parameters, but separate hidden states.

Evaluation Design

Comparison with standard optimizers:

- Stochastic Gradient Descent
- RMSprop
- ADAM
- Nesterov's accelerated gradient (NAG)

Datasets:

- Quadratic functions: $f(\theta) = ||W\theta - y||_2^2$; with 10x10 matrices W and 10-dimensional vector y
- MNIST
- CIFAR-10
- ImageNet

Experimental Results

Quadratic functions & MNIST

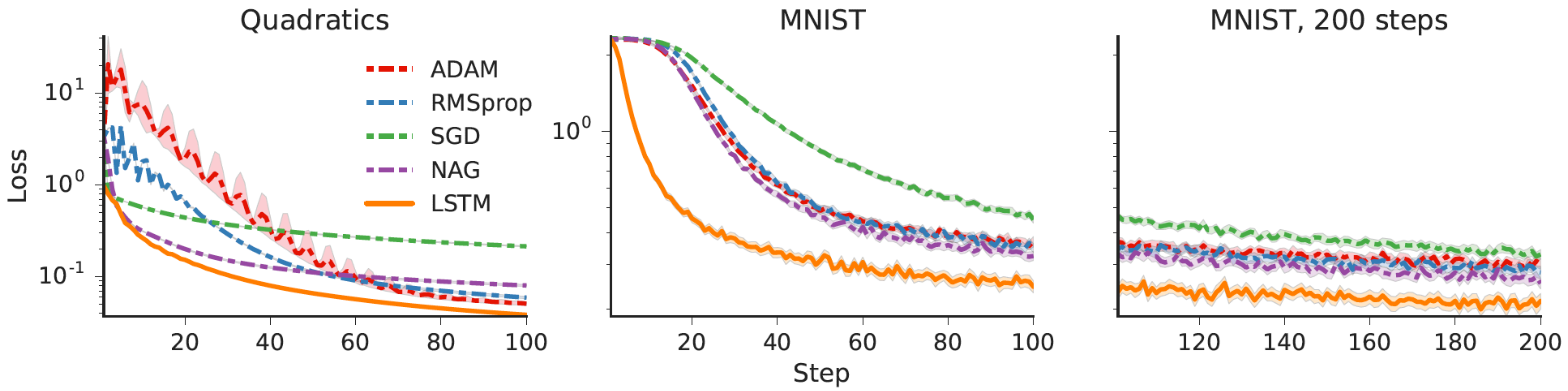


Figure 4: Comparisons between learned and hand-crafted optimizers performance. Learned optimizers are shown with solid lines and hand-crafted optimizers are shown with dashed lines. Units for the y axis in the MNIST plots are logits. **Left:** Performance of different optimizers on randomly sampled 10-dimensional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

Experimental Results

MNIST

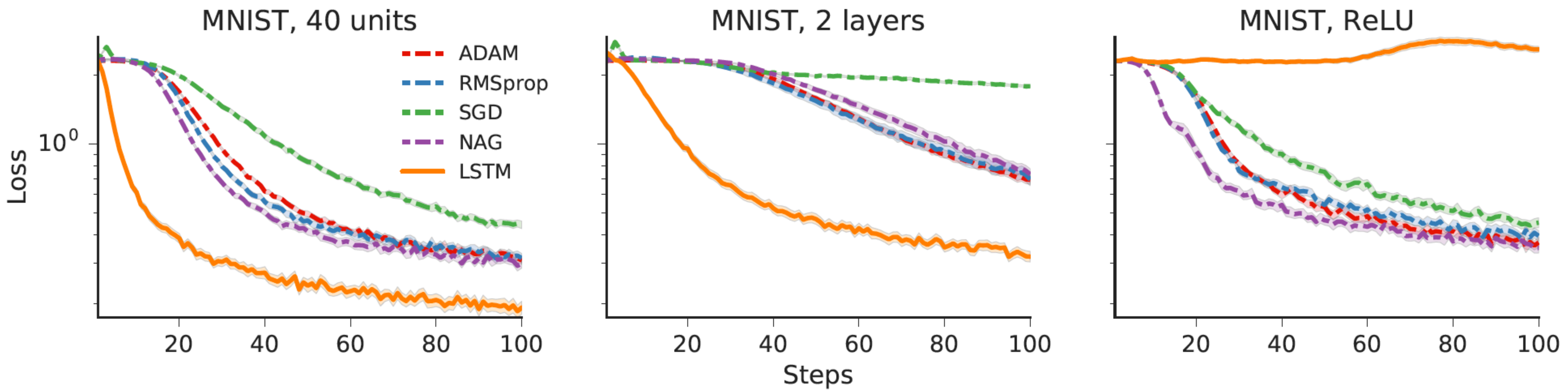


Figure 5: Comparisons between learned and hand-crafted optimizers performance. Units for the y axis are logits. **Left:** Generalization to the different number of hidden units (40 instead of 20). **Center:** Generalization to the different number of hidden layers (2 instead of 1). This optimization 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.

Experimental Results
CIFAR

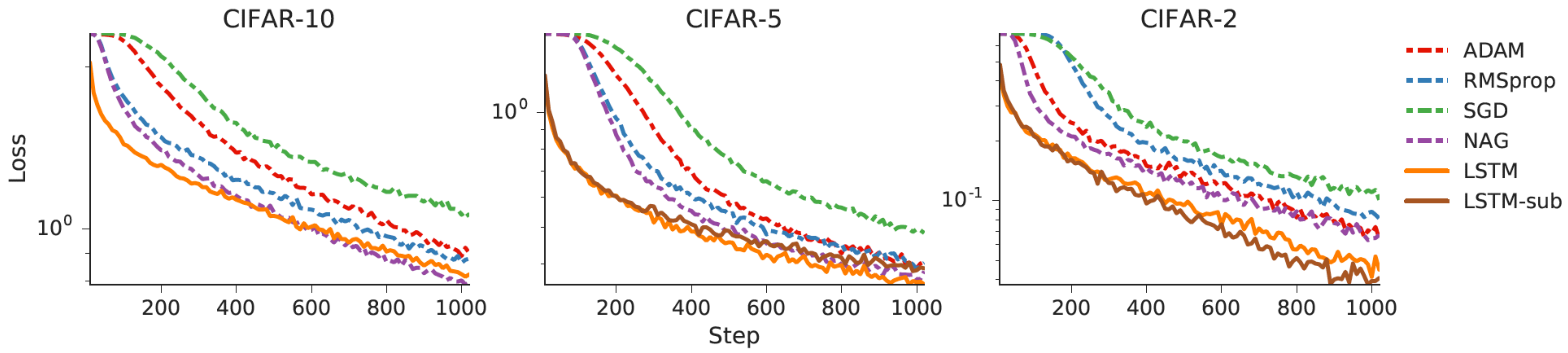


Figure 7: Optimization performance on the CIFAR-10 dataset and subsets. Shown on the left is the LSTM optimizer versus various baselines trained on CIFAR-10 and tested on a held-out test set. The two plots on the right are the performance of these optimizers on subsets of the CIFAR labels. The additional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

Experimental Results

Neural Art

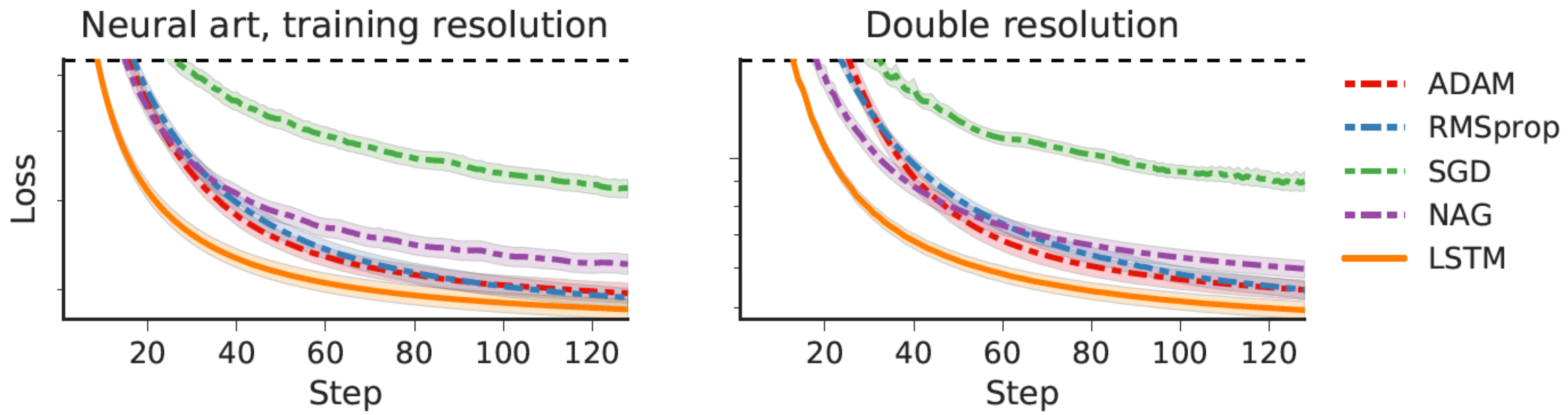


Figure 8: Optimization curves for Neural Art. Content images come from the test set, which was not used during the LSTM optimizer training. Note: the y-axis is in log scale and we zoom in on the interesting portion of this plot. **Left:** Applying the training style at the training resolution. **Right:**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

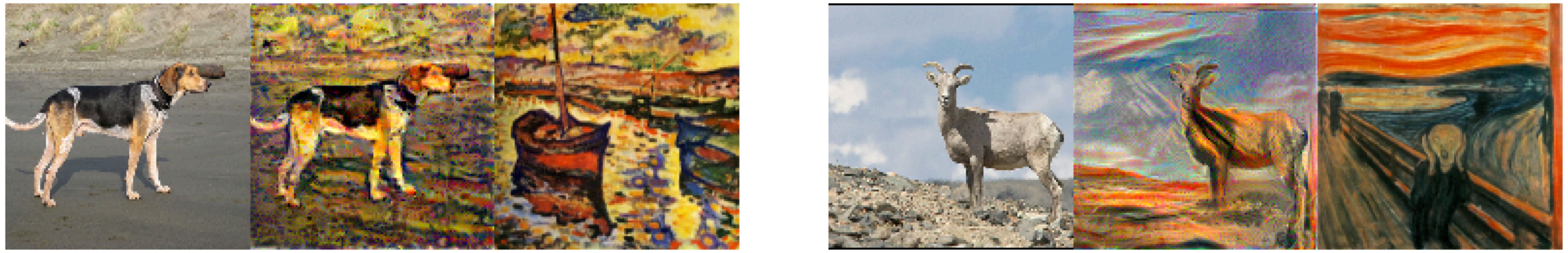


Figure 9: Examples of images styled using the LSTM optimizer. Each triple consists of the content image (left), style (right) and image generated by the LSTM optimizer (center). **Left:** The result of applying the training style at the training resolution to a test image. **Right:** The result of applying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

Critique the main contribution

Challenges with RNNs

- Computation being slow
- Difficulty accessing information from a long time ago

Critique:

- No comparison in the time it takes to train the network
- No comparison in the time to run the network (only how the loss function change each step)

Critique the main contribution

Method of training

Early stopping is implemented while training; the authors would stop after each epoch, freeze the parameters and evaluate its performance.

Critique:

- This method of training is very time consuming
- Required human evaluation

Critique the main contribution

Models to compare

The authors tuned the learning rate for SGD, RMSProp, ADAM, and NAG

Critique:

- These standard optimizers might not be at their optimal performance

Discuss Questions

Update Rule

Standard optimizer's sequence update:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t) \text{ (1)}$$

LSMT optimizer's sequence update:

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) \text{ (2)}$$

What is the significant of the (-) in equation (1)?

Why does it change to the (+) in (2)?