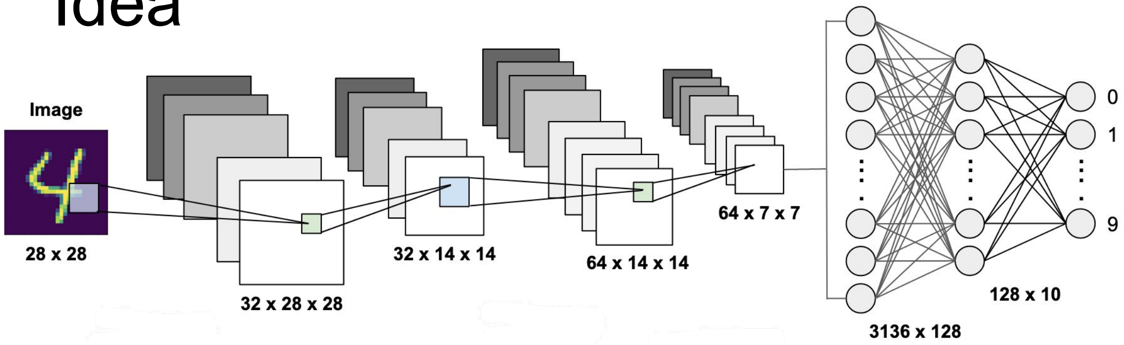
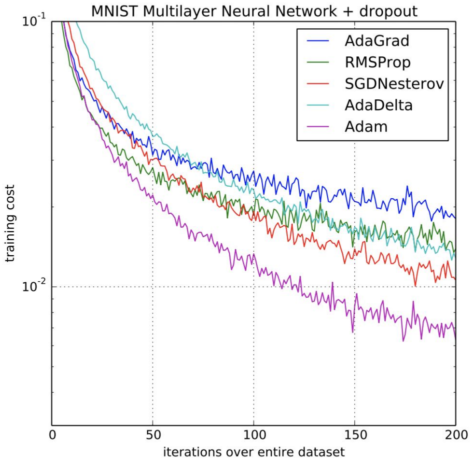
Learning to learn by gradient descent by gradient descent

Aleksandr Zuev
TUM SS21 [IN2107]
Recent trends in Automated Machine Learning
16.06.2021

Idea





- The move from hand-crafted features to learning features was very successful

- Why to design optimization algorithms by hand?

[MNIST Handwritten Digits Classification using a Convolutional Neural Network (CNN) | by Krut Patel | Towards Data Science]
[Adam: A Method for Stochastic Optimization]

Optimization problem

In ML setup it is mostly a problem of optimizing an **objective function** f(θ) defined over some **domain** θ ∈ Θ, and our goal is to find a **minimizer**:

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

The standard approach results in some sort of **gradient descent** with the following update rule:

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

No free lunch

No Free Lunch Theorems for Optimization [Wolpert and Macready, 1997] show that in the setting of combinatorial optimization, no algorithm is able to do better than a random strategy in expectation.

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

Learned update rule

We have the same optimization problem and our goal is to find a minimizer:

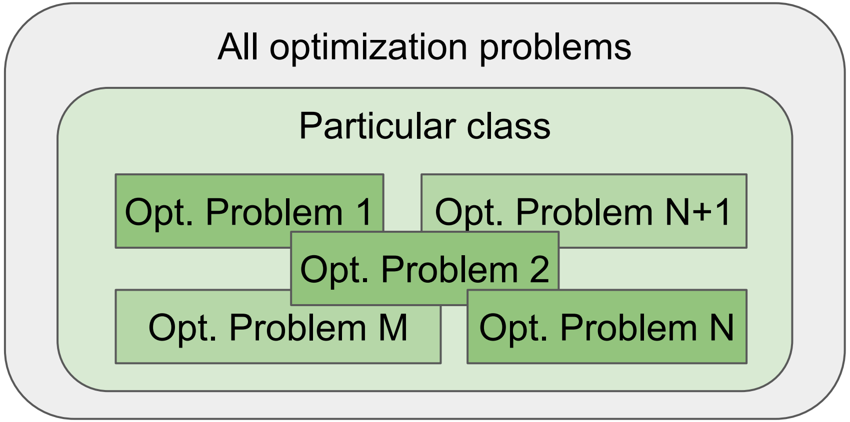
$$\theta^* = \arg\min_{\theta \in \Theta} f(\theta)$$

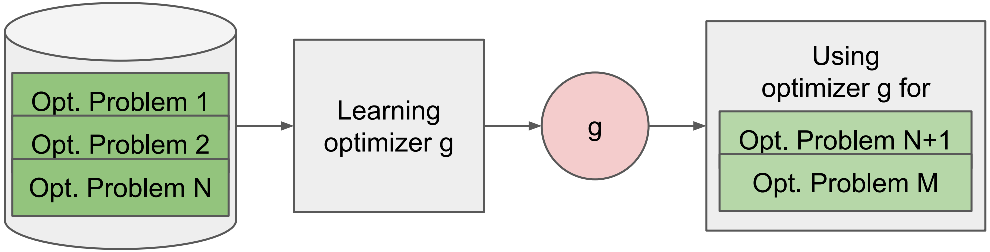
But now, let's learn **update rule *g*** specified by its own set of parameters ϕ:

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

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

Transfer learning and generalization





Goal: develop a procedure for constructing a learning algorithm which performs well on a particular class of optimization problems.

Casting construction of a learning algorithm as a learning problem itself allows to specify a class of optimization problems by examples.

Learning to learn with RNNs

Final parameters: optimizer parameters ϕ and the optimizee *f*:

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

Loss, given the distribution of functions *f*:

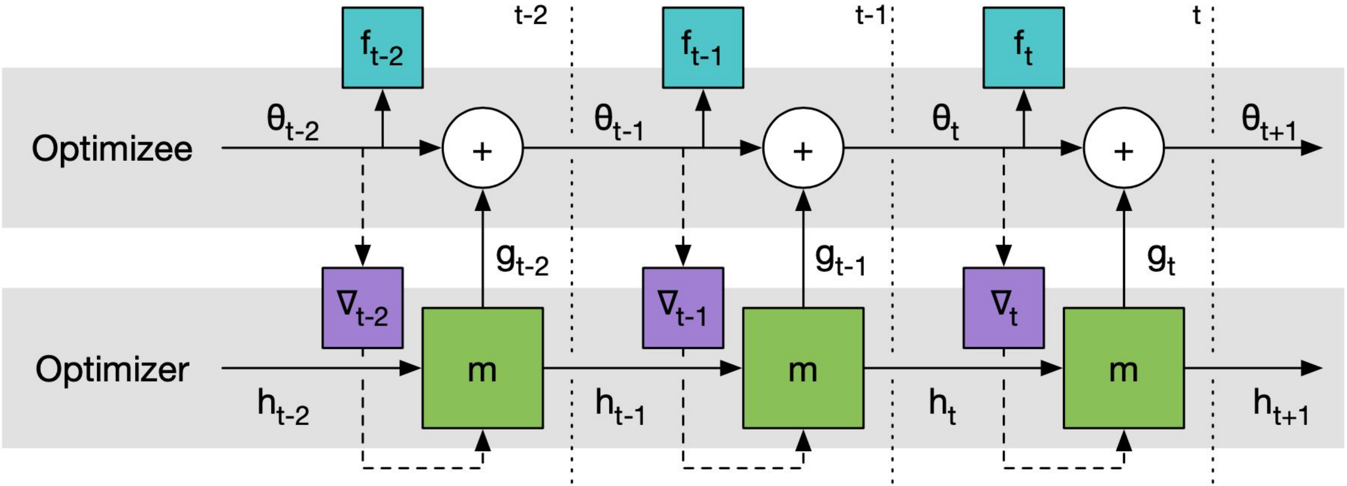
$$\mathcal{L}(\phi) = \mathbb{E}_f\left[f\left(\theta^*(f, \phi)\right)\right]$$

Using *m* – RNN, $w_t \in \mathbb{R}_{\geq 0}$, short notation $\nabla_t = \nabla_\theta f(\theta_t)$
and depending on trajectory of optimization for some horizon T:

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

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

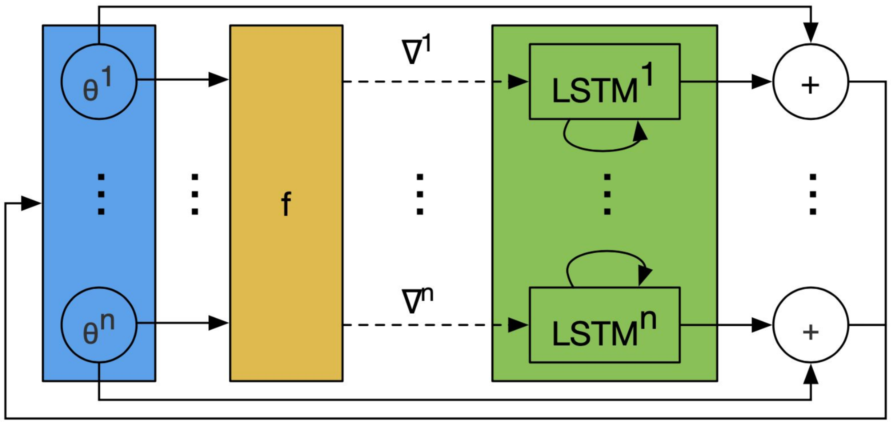
Minimizing loss



Gradient descent on ϕ with the assumption of $\partial\nabla_t/\partial\phi = 0$

(gradients along the dashed lines are dropped)

Coordinatewise LSTM optimizer



We want to optimize tens of thousands of parameters → fully connected RNN is not feasible

We will use optimizer RNN which operates <u>coordinatewise</u> (similar to Adam)

This results in:

- small network
- invariant to the order of parameters

All LSTMs have:

- shared parameters
- separate hidden states

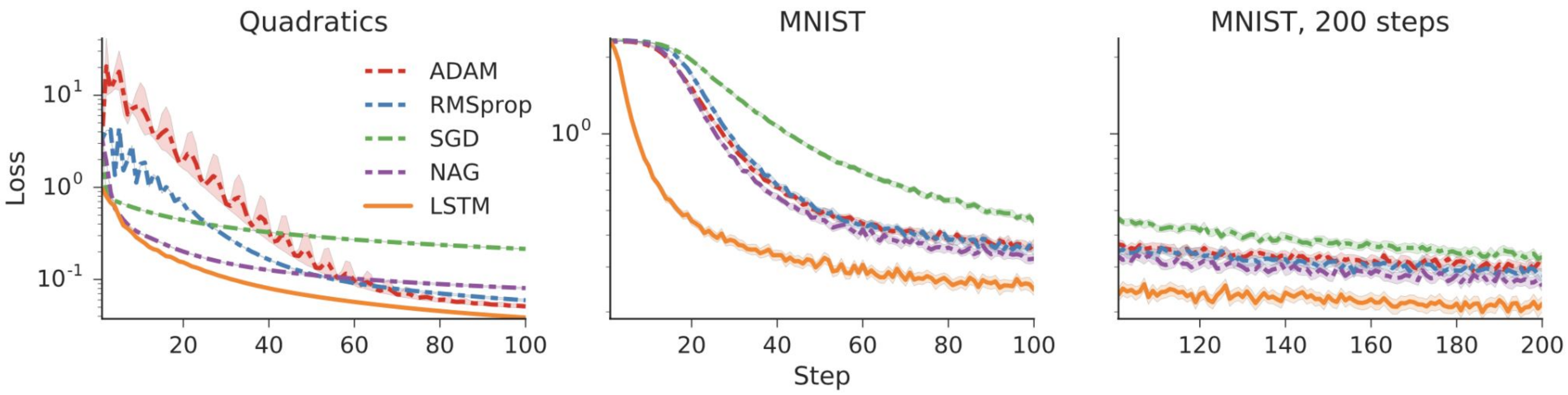
Preprocessing and postprocessing

Optimizer inputs and outputs can have very different magnitudes

$$\nabla^k \rightarrow \begin{cases} \left(\frac{\log(|\nabla|)}{p}, \operatorname{sgn}(\nabla)\right) & \text{if } |\nabla| \geq e^{-p} \\ (-1, e^p \nabla) & \text{otherwise} \end{cases}$$

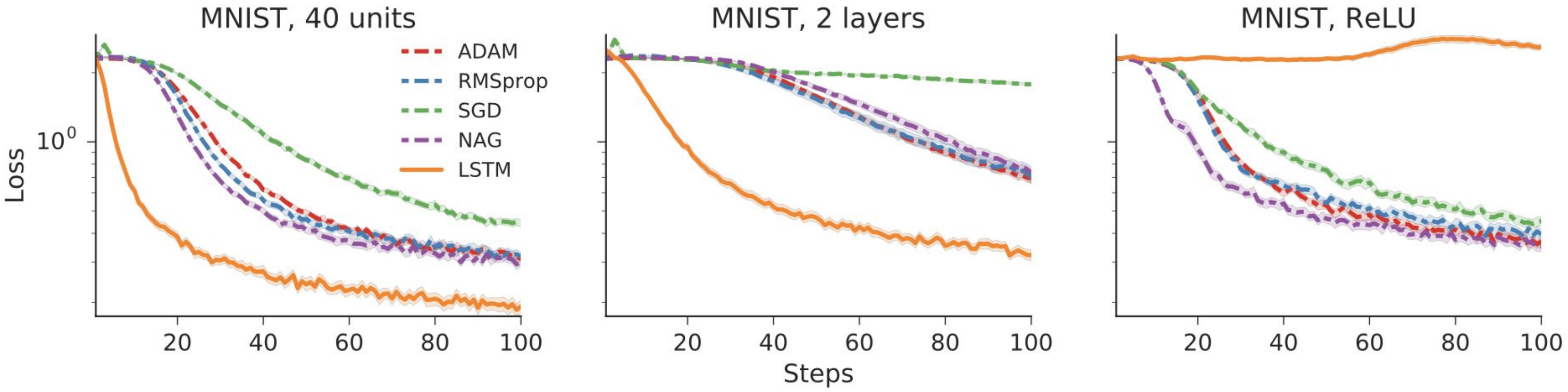
In practice rescaling inputs and outputs using suitable constants is sufficient

Experiments



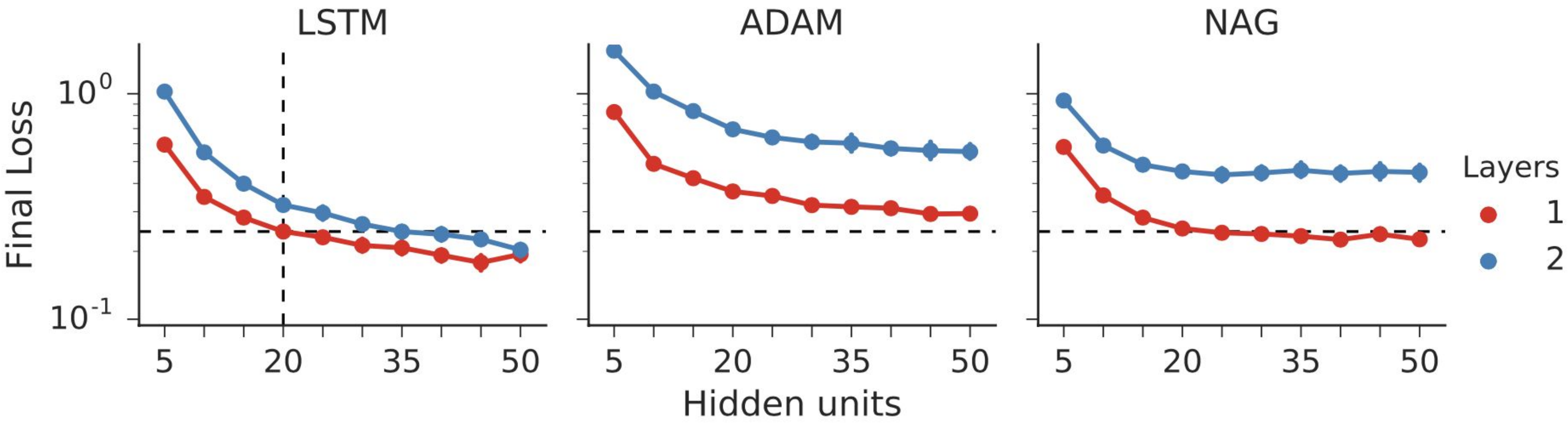
Optimizer RNN	2-layer LSTM, 20 hidden units, trained on 100 epochs using Adam, learning rate found by random search		Reused from MNIST ←
Optimizee	$f(\theta) = \lVert W\theta - y \rVert_2^2$ for 10x10 W matrices	Cross entropy error of NN, 20 hidden units, sigmoid	Reused from MNIST ←

Generalization to different architectures



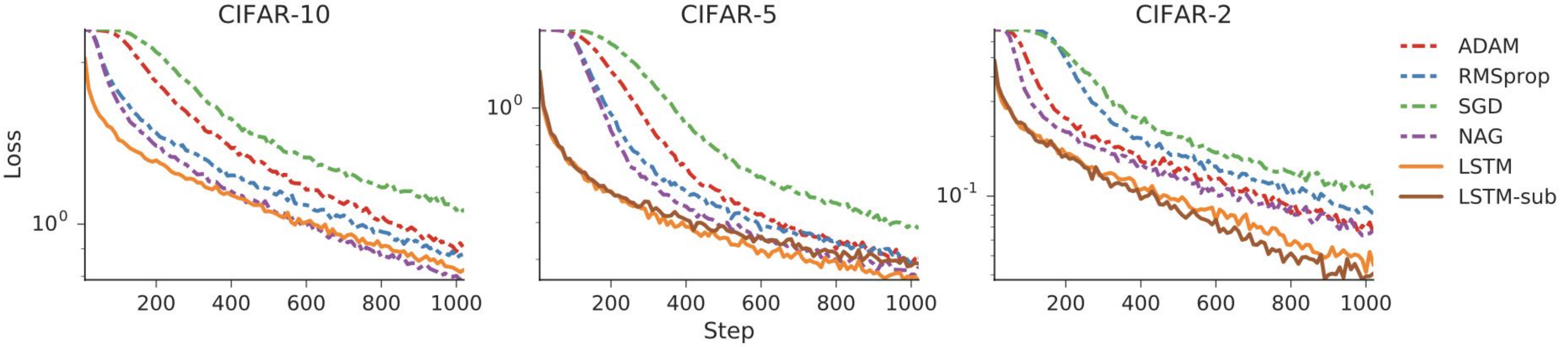
Optimizer RNN	Reused same from MNIST 2-layer LSTM, 20 hidden units, trained on 100 epochs using Adam, learning rate found by random search		
Optimizee	Cross entropy error of NN, 40 hidden units, sigmoid	Cross entropy error of NN, 20+20 hidden units, sigmoid	Cross entropy error of NN, 20 hidden units, ReLU

Generalization to different architectures



Optimizer RNN	Reused same from MNIST 2-layer LSTM, 20 hidden units, trained on 100 epochs using Adam, learning rate found by random search
Optimizee	Cross entropy error of NN, hidden units and number of hidden layers systematically vary, sigmoid

Convolutional network on CIFAR-10



Optimizer RNN	For fully-connected layer: separate LSTM trained on train set, LSTM-sub trained on held-out set		
	For convolutional layers: separate LSTM		
Optimizee	Cross entropy error of 3x(Conv2d → MaxPool) → Fully-connected(32), ReLU, BatchNorm used		
Dataset	All labels	5 of 10 labels	2 of 10 labels

Neural art

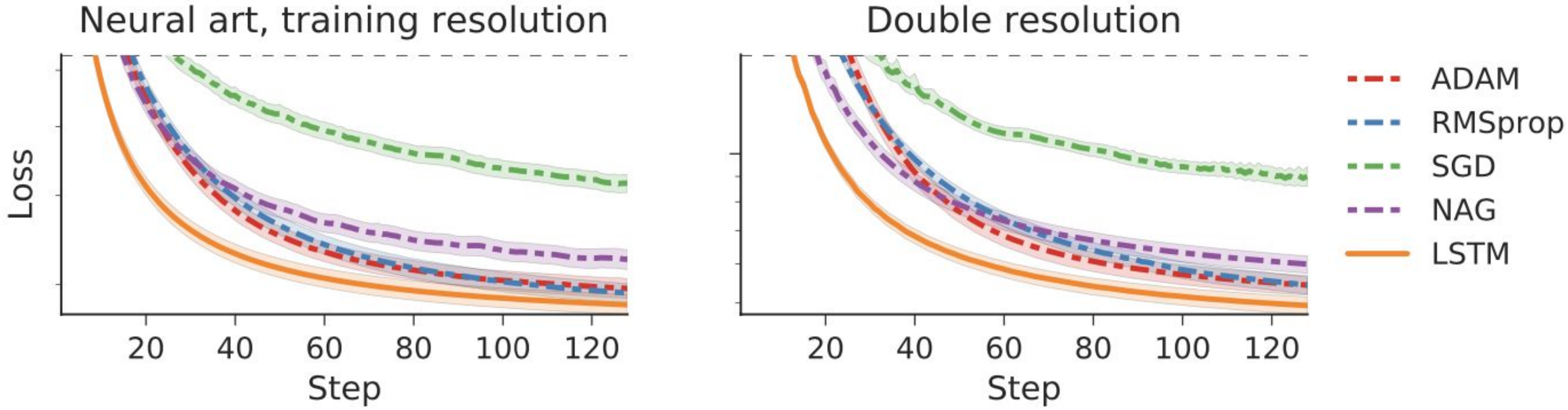
Each content and style image pair results to a different optimization problem

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

content image *styled image* style image

content image *styled image* style image

Neural art



<table>
<tr><td colspan="2">Same optimizer,
trained on 1 fixed style image and 1800 content images (64x64) from ImageNet</td></tr>
<tr><td>Same style image,
same resolution (64x64)</td><td>Different style image,
double resolution (128x128)</td></tr>
</table>

Conclusion

- Casting the design of optimization algorithms as a learning problem
- Learned optimizers perform comparably well
- Some degree of generalization
  (trained on 12,888, generalized to 49,152 parameters in Neural art)

– Problematic to generalize to different activation functions (Sigmoid, ReLU) and layers (Conv2d, Fully-connected)
– Scalability
– Proof of concept

Thank you for attention!

If you have any questions feel free to ask