# Learning optimization algorithms with neural networks

M.Sc. Data Science thesis presentation - AUEB

Nikolaos Stefanos Kostagiolas

Supervisor: Prof. Evangelos Markakis

## Table of contents

## Introduction

## Origins of machine learning

Learning: the ability of living organisms to acquire new, or modify existing knowledge, behaviors, skills, values or preferences.

Turing: need of a learning mechanism in computer systems as a key element towards emulating human intelligence.

*Machine Learning*: term coined by Arthur Samuel (1959).

Formal definition by Tom Mitchell (1998):
Machine learning is the study of algorithms that:

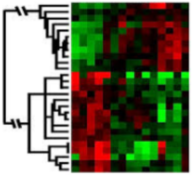
- **improve** their performance *P*
- at some **task** *T*
- with **experience** *E*

Therefore, a well-defined learning task is given by <*P*, *T*, *E*>.

## Applications of machine learning

Machine learning is used in settings where:

- Human expertise does not exist (navigating on Mars)
- Humans can't explain their expertise (speech recognition - natural language processing)
- Models must be customized (personalized medicine)
- Models are based on huge amounts of data (genomics)



Disclaimer: Machine learning is not a magic wand!

## A striking difference from human learning

**High performance** of machine learning model $\Leftrightarrow$ **abundance of data**

Machine learning $\neq$ human learning, latter is faster and way more sample-efficient:

- Kids are able recognize objects easily after being exposed to few examples
- People who know how to ride a bike are likely to discover the way to ride a motorcycle fast with little or even no demonstration

**Question**: is it possible for a machine learning model to exhibit similar properties, i.e. **learning new concepts and skills fast with a few training examples?**

**Spoiler**: yes indeed (especially if Turing's ”emulating human intelligence” is still the goal).

## Solution: ”learning to learn” or simply ”meta-learning”

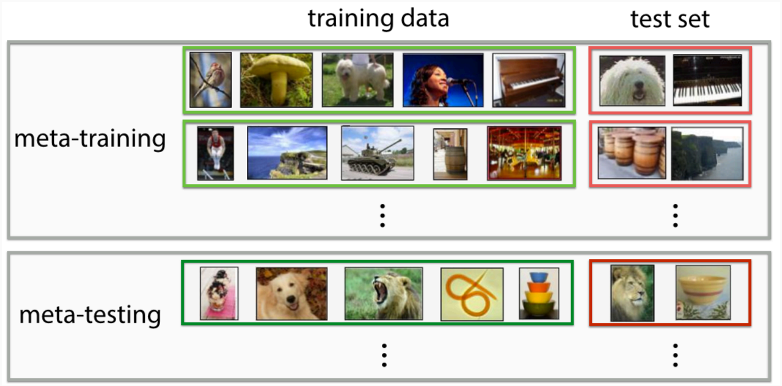
Meta-learning: aims to introduce generalization capabilities and sample efficiency to machine learning algorithms.

Generalization: the ability of adapting to new tasks and new environments that have never been encountered before during training time

Adaptation: mini learning session during test time but with limited exposure to the new task configurations.

Profit: The model can complete new tasks without the need of many training examples because its way of learning them is more efficient. Thus it has ”learned to learn”.

## Visualization example of a meta-learning setting



## Application examples

**More importantly**: meta-learning can be applied to a variety of machine learning problems, e.g. supervised learning, reinforcement learning etc.

Examples of meta-learning tasks include:

- A classifier trained on non-cat images can tell whether a given image contains a cat after seeing a handful of cat pictures.
- A game bot is able to quickly master a new game.
- A mini robot completes the desired task on an uphill surface during test even through it was only trained in a flat surface environment.
- An optimizer trained on a specific task by a gradient descent variant can perform efficiently tasks of the same family.

## Related Work

## Meta-learning: the initial steps

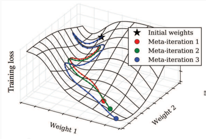
Initial studies in the field of meta-learning:

- Schmidhuber's legacy studies between late 1980s and early 1990s:
    - General idea: enhance recurrent neural networks with the ability to modify their own weights.
    - Downside: high computational requirements!
- Hochreiter et al., 2001:
    - General idea: Use an LSTM (Long Short-Term Memory) network as an optimizer in order to train multi-layer perceptrons.
- Bengio et al., 2002:
    - General idea: Replace backpropagation with a more biologically-plausible update algorithm

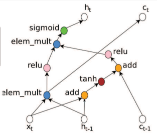
## Meta-learning: scaling up to form a field during the neural network resurgence

Despite initial plateaus, meta-learning has become a hot topic within the ML research community with several well known use cases:

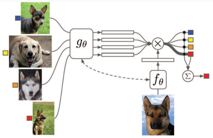
- Hyper-parameter and neural network optimization,
- Deep learning architecture search
- Few-shot learning
- Speeding-up reinforcement learning



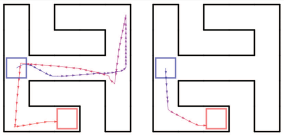
hyperparameter optimization
Maclaurin et al.'15



learned recurrent cell
Zoph & Le'17



few-shot image classifier
Vinyals et al.'16


learning to quickly navigate new mazes
Duan et al.'16

## Approaches to meta-learning: Optimizer learning

Work initiated with **Hochreiter et al.**'s work discussed earlier

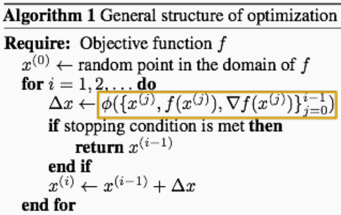
General idea

- Learn an **update rule** for the parameters of a neural network instead of devising it.
- **Meta-learner**: neural network, typically a recurrent neural network variant that learns the update rule
- **Learner**: another network that is updated by the meta-learner.
- The meta-learner network acts as the **optimizer**.
- Widely applied with general success [1], [2], [3], [4], [5], [6].
- Results in a more efficient update rule than that of a hand-written optimizer (?) while also saving time by bypassing hyperparameter tuning.

## Approaches to meta-learning: Optimizer learning

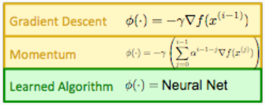
Motivation behind casting optimizer design as a learning problem: existing common elements among the different algorithms used for continuous optimization.

- Operation in an iterative fashion where the iterate is a single point in the domain of the objective function.
- Random initiation of the iterate point across the domain.
- Modification of the iterate at each iteration using a step vector update rule.
- An update rule that takes into account the previous or current gradients of the objective function.



**Algorithm 1** General structure of optimization algorithms

```
Require: Objective function $f$
  $x^{(0)} \leftarrow$ random point in the domain of $f$
  for $i = 1, 2, \ldots$ do
    $\Delta x \leftarrow \boxed{\phi(\{x^{(j)}, f(x^{(j)}), \nabla f(x^{(j)})\}_{j=0}^{i-1})}$
    if stopping condition is met then
      return $x^{(i-1)}$
    end if
    $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
  end for
```



| | |
|---|---|
| Gradient Descent | $\phi(\cdot) = -\gamma \nabla f(x^{(i-1)})$ |
| Momentum | $\phi(\cdot) = -\gamma \left( \sum_{j=0}^{i-1} \alpha^{i-1-j} \nabla f(x^{(j)}) \right)$ |
| Learned Algorithm | $\phi(\cdot) =$ Neural Net |

## Meta-learning with recurrent neural networks

## Problem definition

General practice in machine learning: express each task in the form of an optimization problem where the desired outcome is to optimize an objective function $f(\theta)$ over some domain $\theta \in \Theta$

Usually this is done by applying a form of gradient descent in a sequence of updates of the form:

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

Several optimization algorithms have surfaced in the process, namely momentum, Rprop, Adagrad, RMSprop and ADAM.

## Problem definition

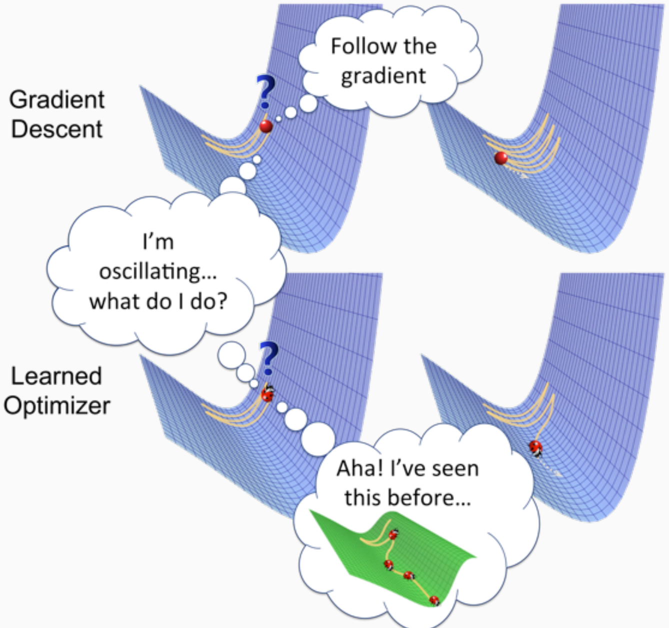
**Problem**: each of the aforementioned algorithms performs well by exploiting problem-specific structures at the expense of generalization

**Solution**: learn the update rule by using an optimizer model $m_\phi$, specified by parameters $\phi$ in order to update the parameters of our loss function (also referred to as the optimizee) in the following form:

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

Optimizer is constantly informed about the performance of the optimizee $f$ and, by updating its parameters $\phi$ it varies its update rule proposals in order to infer the optimal update rule for updating the optimizee's parameters $\theta$, thus maximizing its performance.

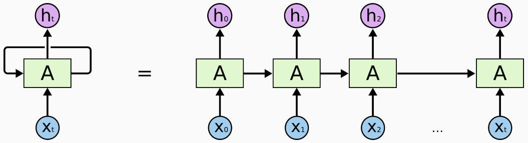
## Intuitive visualization



## Architecture details

In our case, the learned update will take the form of a LSTM (a recurrent neural network variant). Why?

- **Requirement #1**: a sequential learning architecture is required - gradient descent is, at its very essence, a sequence of updates on separate states in-between.
- **Reason for requirement #1**: reasoning about previous events that occurred during the optimization process in order to shape better future update rules.
- **Architectural solution #1**: employ a recurrent neural network architecture that allows information from earlier points in time to persist to later ones.



## Architecture details

But why do we specifically need LSTMs instead of a plain recurrent neural network?

- **Requirement #2**: information about previous states, not only has to affect later ones but it also has to be maintained.
- **Reason for requirement #2**: information regarding how previous suggested updates of our optimizer affected the performance of the optimizee has to be accessed in some way.
- **Architectural solution #2**: employ a Long Short-Term Memory network (or simply LSTM) with a memory-like feature called the *cell state*.
- Disclaimer: Not novel ideas, introduced earlier mainly by **Andrychowicz et al.**, 2016 and widely-applied later.

## Learning phase details

So how is the optimizer trained?

- Let the final *optimizee parameters* be written as $\theta^*(f, \phi)$ where $\phi$ are the optimizer parameters and $f$ is the function we are trying to optimize (or simply the optimizee).
- Thus, the expected loss with relation to the optimizer can be written as:
$$L(\phi) = E_{f \longleftarrow D}\left[f(\theta^*(f, \phi))\right]$$
where $f$ is drawn **according to some distribution $D$ of functions**.
- As it is already mentioned, our LSTM optimizer network $m$ outputs the update steps $g_t$ and is parameterized by $\phi$, while its state at time $t$ can be denoted as $h_t$.
- But what if we wanted to relate the expected loss with respect to the parameter values **throughout the whole optimization trajectory**?

## Learning phase details

The previous expected loss definition is equal to the following:

$$L(\phi) = E_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right]$$

where

$$\theta_{t+1} = \theta_t + g_t$$

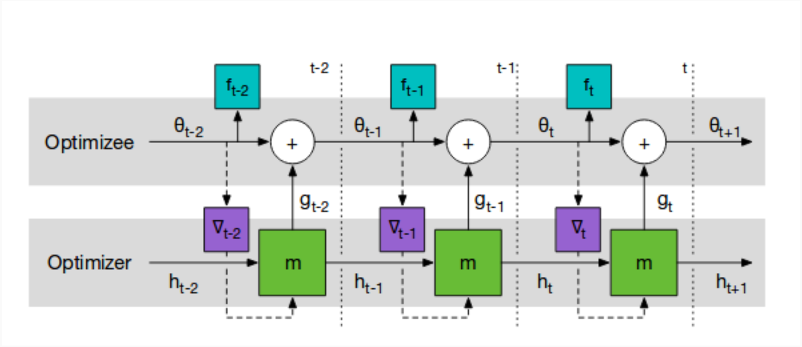
$$\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi)$$

We can, thus, perform gradient descent on $\phi$ in order to minimize $L(\phi)$ which should give us an optimizer that is capable of optimizing f efficiently.

## Learning phase details

Basically this is nothing we wouldn't expect: the loss of the optimizer neural net (i.e. our LSTM) is simply **the summed training loss of the optimizee** as it is trained by the optimizer.

The optimizer takes in the gradient of the optimizee as well as its previous state, and outputs a suggested update that we hope will reduce the optimizee's loss as fast as possible.



## Experiments & Results

## Experimental setup

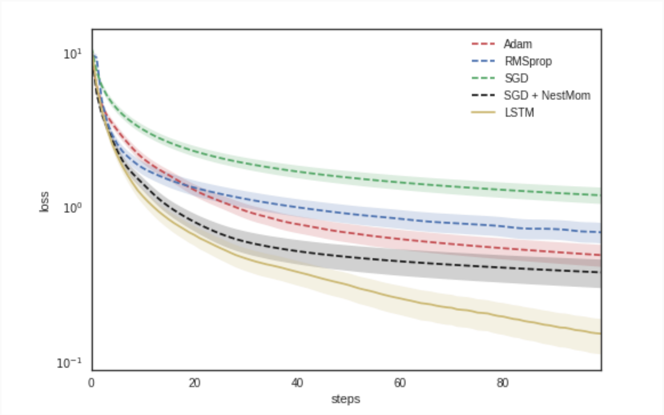
- Optimizer architecture employed: two LSTM layers with 20 hidden units per layer.
- Optimizing the optimizer: ADAM optimizer with learning rate selected by random search.
- Each epoch consists of 100 iterations. After each epoch the performance of the optimizer is evaluated (no training during evaluation phase). Every 20 iterations, the parameters of the optimizer are updated using backpropagation.
- Best optimizer is chosen based on its final test loss on a novel sampled test set.
- Comparison with well-known and widely-used optimizers in literature: SGD, RMSprop, ADAM and SGD with Nesterov Momentum.
- Experiment setting identical to the one used in the study of Andrychowicz et al., 2016.

## Experiment #1 - Quadratic loss functions

Motivation: most loss functions are in quadratic form

Task: optimization of synthetic quadratic functions defined in the 10-dimensional space (here minimization of squared error loss):

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

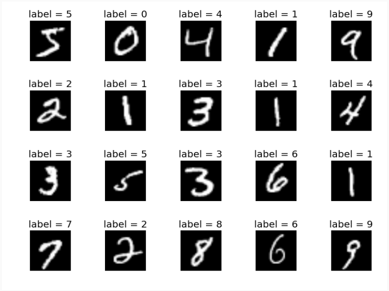


## Experiment #2 - MNIST

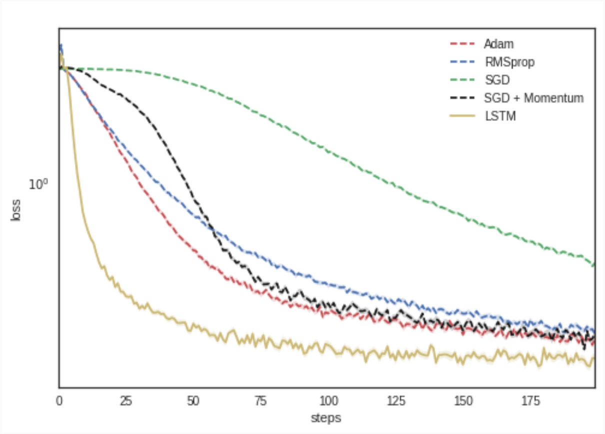
**Motivation**: baseline for most computer vision systems.

**Task**: optimization of a single-hidden-layer neural network consisting of 20 units on the MNIST handwritten digits dataset, used Multi-layer Perceptron crossentropy loss as the minimization objective.

**Dataset description**: 60.000 training examples and 10.000 test examples of 28x28 greyscale images, each one displaying a handwritten number digit.



## Experiment #2 - MNIST



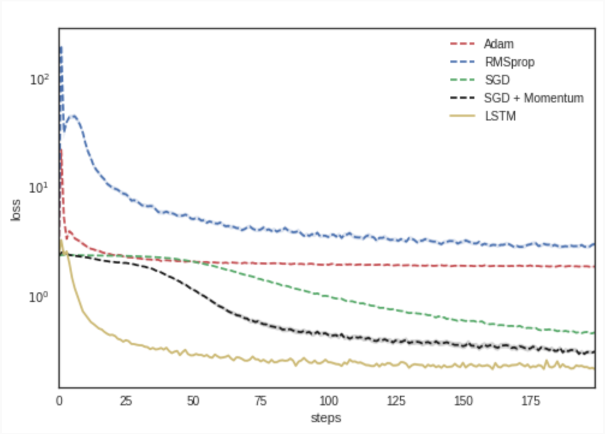
## Experiment #3 - generalization to different model architectures

Motivation: test the generalization capabilities of the learned optimizer on MNIST on different model architectures, albeit being similar to the one it was originally trained on.

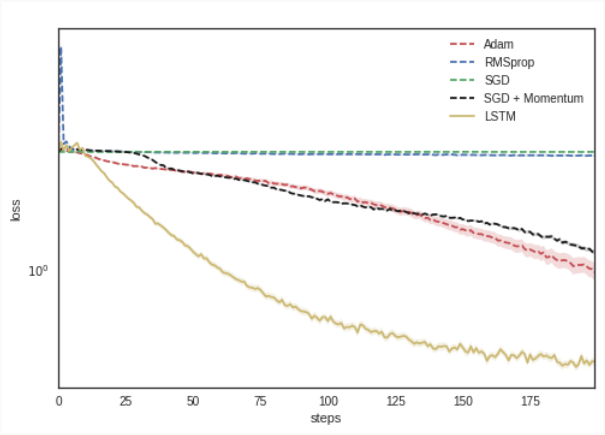
Task:

- alter the model architecture by:
  - introducing more hidden units (40)
  - adding more layers (2).
- check if the learned optimizer can adapt to it by comparing its performance with optimizers that were trained on these very architectures.
- stick to MNIST for the time being.

## Experiment #3 - generalization to different model architectures



## Experiment #3 - generalization to different model architectures



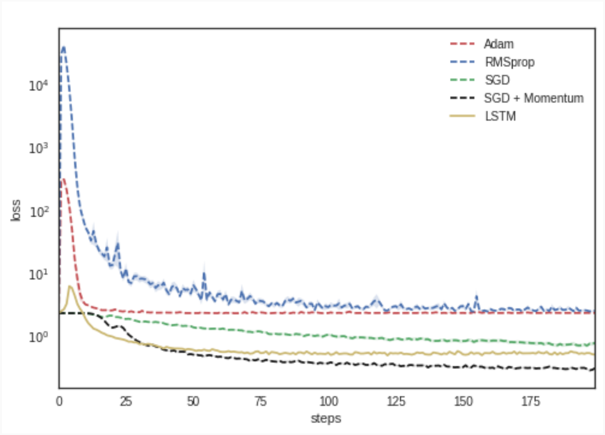
## Experiment #4 - generalization to different learning dynamics

Motivation: test the generalization capabilities of the learned optimizer on MNIST on different learning dynamics, where it is expected to not perform very well due to differences in the optimization landscapes.

Task:

- alter the model architecture by changing the activation function from sigmoid to ReLU.
- check if the learned optimizer can adapt to it by comparing its performance with optimizers that were trained on these very architectures.
- stick to MNIST for the time being.

## Experiment #4 - generalization to different learning dynamics



## Experiment #5 - generalization to different MNIST-like datasets

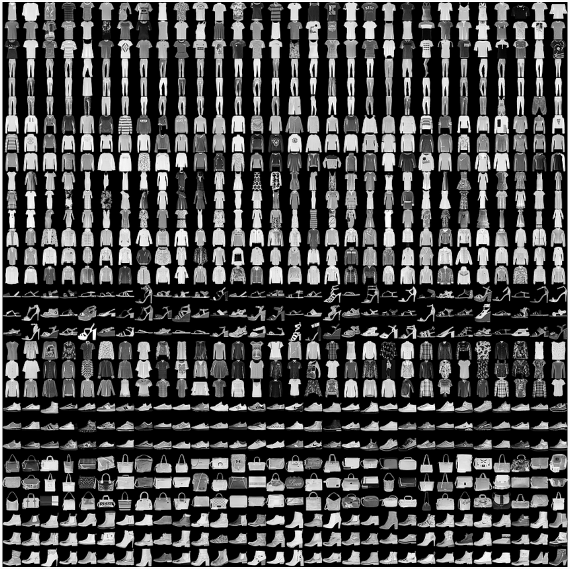
**Motivation**: learned optimizer performs reasonably well $\rightarrow$ generalization to similar tasks to those it was trained on.

**Task**: compare the performance of the optimizer trained on the MNIST dataset on similar datasets like FashionMNIST and EMNIST against its standard hand-written counterparts trained on these very datasets.

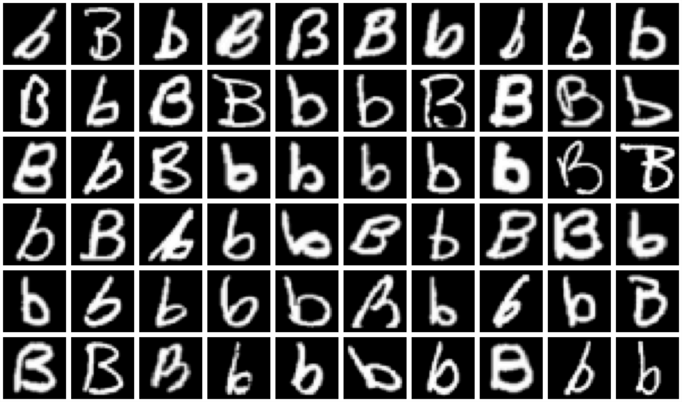
**Dataset description**:

- FashionMNIST: 60.000 training examples and 10.000 test examples in the form of 28x28 greyscale images, each one of which belongs to 10 categories, encapsulates a harder problem than simply recognizing digits.
- EMNIST Letters: 145.600 28x28 greyscale images depicting characters belonging to 26 letter classes.

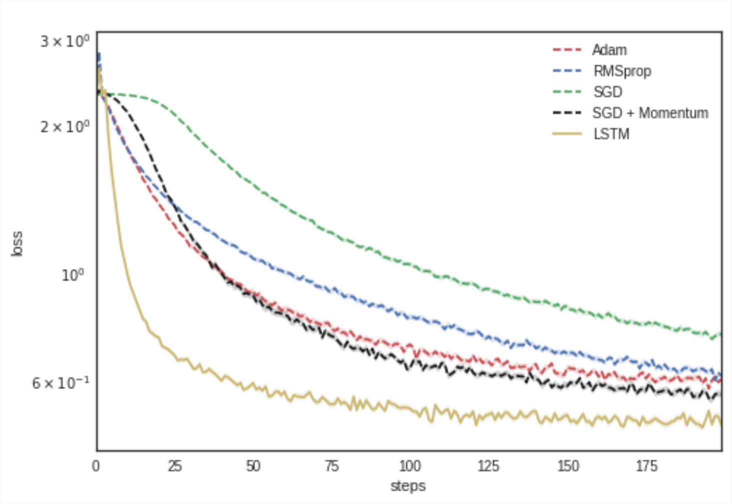
## Experiment #5 - generalization to different MNIST-like datasets



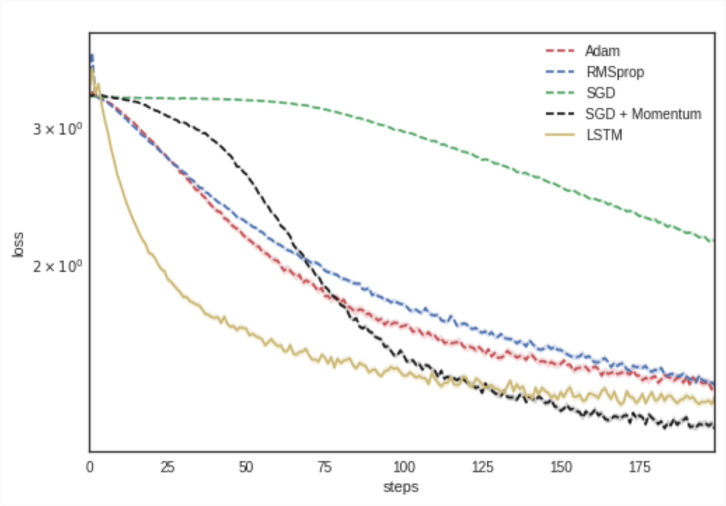
## Experiment #5 - generalization to different MNIST-like datasets



## Experiment #5 - generalization to different MNIST-like datasets



## Experiment #5 - generalization to different MNIST-like datasets



## Experiment #6 - generalization to entirely different datasets

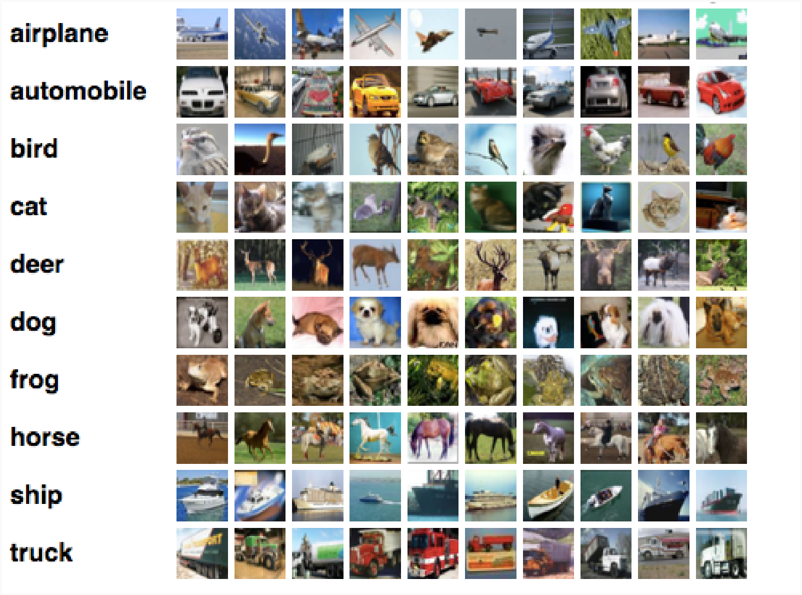
**Motivation**: learned optimizer perform reasonably well in settings dissimilar to its training regime $\rightarrow$ capable of adapting beyond tasks of the same family it was trained on.

**Task**: compare the performance of the optimizer trained on the MNIST dataset on dissimilar datasets like CIFAR-10 and SVHN against its standard hand-written counterparts trained on these very datasets.

**Dataset description**:

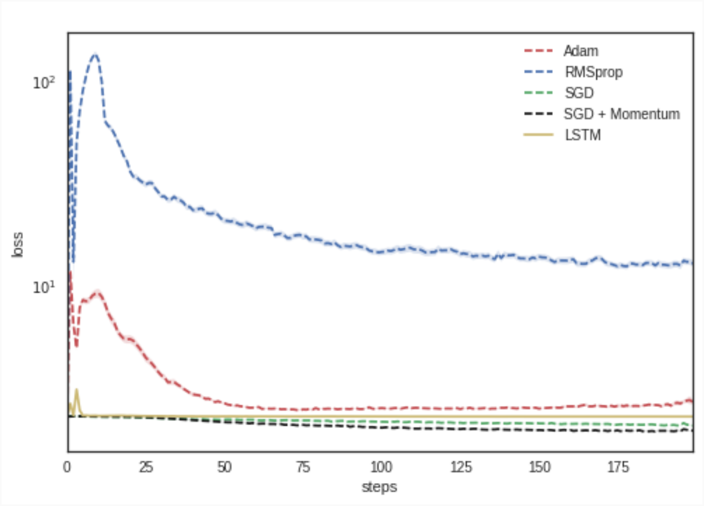
- CIFAR-10: 60.000 32x32 coloured images divided in 10 classes (6.000 image examples per class) the 50.000 of which constitute the training set while the rest 10.000 form the test set.
- SVHN: Another real-world 32x32 coloured image dataset that consists of a training set of 73257 and a test set of 26032 images.

# Experiment #6 - generalization to entirely different datasets

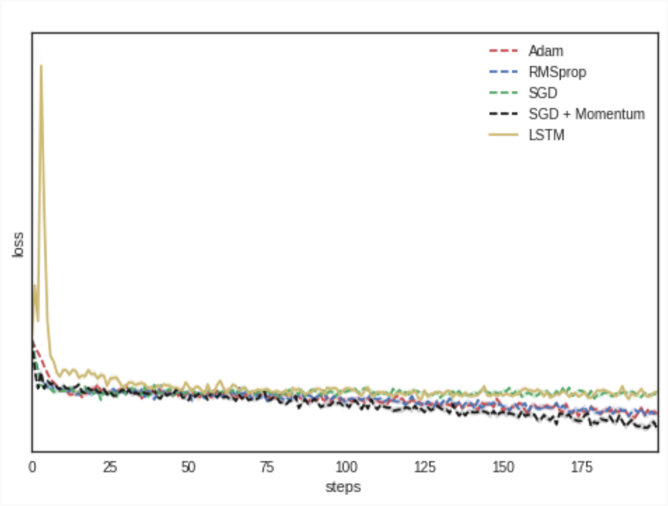


## Experiment #6 - generalization to entirely different datasets

## Experiment #6 - generalization to entirely different datasets



## Experiment #6 - generalization to entirely different datasets



## Conclusion

## Initial goals

The aims of this study were twofold:

- Extend **Andrychowicz et al.**'s earlier work that demonstrated that optimizer design could be cast as a **learning problem** by testing an optimizer's **generalization capabilities** at optimizing functions stemmed from the **same distribution**.
- Systematically examine whether achieving considerable degrees of **cross-task transfer** between **unrelated** settings was feasible, which was also a claim made by that paper, albeit being reinforced with limited evidence.

## What did we notice?

From our experiments it is easily made clear that learned optimizers:

- outperform by a wide margin their standard handwritten counterparts when tested on unseen examples from the same dataset.
- show a remarkable degree of knowledge transfer between optimization tasks stemming from the same family.
- are capable of generalizing to different models as long as heavy changes to the learning dynamics aren't made.
- don't exhibit much generalization capabilities when applied to tasks outside of their training regime likely due to vastly large differences in data structure and number of parameters.
- limited scaling both regarding computational costs and completion time (each epoch can take as much as 2.5 hours to be completed depending on dataset), hindered unbiased experiment selection.

## What did we learn?

- The choice of learned optimizers should be promoted over using hand-written ones in most single-task scenarios, majority of the settings in the deep learning literature.
- Notable degrees of generalization in related tasks may exist due to:
  - a) optimum memorization (potential reason for lengthy completion times) and rapid convergence on familiar loss landscapes and/or
  - b) distillation of important information that is common between tasks
- Absolute multi-task performance based on knowledge transfer is still out-of-hand.
- Initial claims of models that ”learn-*how*-to-learn” can’t be confirmed. Resulting models resemble ”learn-*what*-to-learn” approaches relevant to transfer-learning/multi-task learning strategies.
- Bottom line: **Welcome to the no-free lunch theorem** :)

## What did we learn?

## Thank you & Q/A

Thanks for lending me your ears.

It’s time to dispel the black magic we just discussed about :)

## References i

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas.
**Learning to learn by gradient descent by gradient descent, 2016.**

Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas.
**Learning to learn without gradient descent by gradient descent.**

K. Li and J. Malik.
**Learning to optimize, 2016.**

K. Li and J. Malik.
**Learning to optimize neural nets, 2017.**

S. Ravi and H. Larochelle.
**Optimization as a model for few-shot learning.**

## References ii

O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein.
Learned optimizers that scale and generalize, 2017.