ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

MSC THESIS

Learning Optimization Algorithms with Neural Networks

Author:
Nikolaos Stefanos
KOSTAGIOLAS

Supervisor:
Prof. Dr. Evangelos
MARKAKIS

A thesis submitted in fulfillment of the requirements for the degree of MSc in Data Science

in the

Athens University of Economics and Business
Department of Computer Science

January 8, 2019

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Abstract

Evangelos Markakis
Department of Computer Science

MSc in Data Science

Learning Optimization Algorithms with Neural Networks

by Nikolaos Stefanos KOSTAGIOLAS

The field of Machine Learning owes much for its success to the ability of its algorithms to automatically discover patterns in data. However, these algorithms are still written by hand, despite them being similar in their core element, i.e. the usage of past gradients as a means of locally updating their search for optima. This commonality naturally leads to the debate of whether one could learn those algorithms, instead of designing them manually, i.e., can we learn the optimal parameters of an optimization algorithm that we would like to use for a machine learning problem? Should these "learned" algorithms perform better than their manually-designed counterparts, it could assist in lessening the time spent for hyperparameter tuning among ML practitioners and researchers alike, while also clearing the way for more generalized approaches in cases where an optimizer is needed. However, although recent approaches in the field of optimizer learning or, as it is commonly dubbed, in the field of meta-learning have resulted in learned optimizers that outperform their standard variants in tasks they have been trained on, they have showcased limited examples of their generalization capabilities, i.e. their ability to perform well in tasks outside of their training regime. The goal of the thesis is to explore the generalization capabilities of this form of meta-learning that usually comes in the form of a recurrent neural network optimizer, thus attempting to extend previous studies on learning optimization algorithms.

The thesis also involves an implementation of the relevant experiments in PyTorch, a recently introduced deep learning framework that is widely popular throughout the research community and the code will be open-sourced in order to facilitate further contributions towards answering this research question.

Οικονομικό Πανεπιστήμιο Αθηνών

Διπλωματική Εργασία

Εκμάθηση Αλγορίθμων Βελτιστοποίησης με Νευρωνικά Δίκτυα

Συγγραφέας:
Νικόλαος Στέφανος Κωσταγιόλας

Επιβλέπων:
Επικ. Καθηγητής
Ευάγγελος Μαρκάκης

Περίληψη: Το πεδίο της Μηχανικής Μάθησης χρωστάει μεγάλο μέρος της επιτυχίας του στην ικανότητα των αλγορίθμων του στο να ανιχνεύουν αυτόματα μοτίβα σε δεδομένα. Παρ΄όλα αυτά οι αλγόριθμοι αυτοί ακόμα γράφονται χειροκίνητα, μολονότι εμφανίζουν ομοιότητες ως προς το κεντρικό στοιχείο τους, το οποίο είναι η χρήση προηγούμενων κλίσεων συναρτήσεων ως μέσο για την τοπική επικαιροποίηση της αναζήτησής τους για ακρότατα. Το κοινό στοιχείο αυτό οδηγεί φυσικά στην έγερση του ερωτήματος σχετικού με το αν θα μπορούσαμε να μάθουμε τους αλγορίθμους αυτους αντί να τους σχεδιάζουμε χειροκίνητα, δηλαδή αν είναι εφικτό να μάθουμε τις βέλτιστες παραμέτρους ενός αλγόριθμου βελτιστοποίησης ώστε να τον χρησιμοποιήσουμε σε ένα πρόβλημα μηχανικής μάθησης. Σε περίπτωση που αυτοί οι ¨εκμαθημένοι' αλγόριθμοι αποδίδουν καλύτερα από τους χειροκίνητα σχεδιασμένους, κάτι τέτοιο θα σήμαινε πως θα μπορούσαν να βοηθήσουν στη μείωση του χρόνου που απαιτείται για την εύρεση ιδανικών υπερπαραμέτρων από όσους εφαρμόζουν τεχνικές Μηχανικής Μάθησης αλλά και τους ερευνητές και παράλληλα θα ξεκαθάριζε το τοπίο ώστε να γίνεται χρήση πιο γενικευμένων μεθόδων σε περιπτώσεις όπου είναι αναγκαία η εύρεση ενός βελτιστοποιητή. Παρ' όλα

αυτά, αν και οι πρόσφατες προσεγγίσεις στο πεδίο της εκμάθησης βελτιστοποιητων ή, όπως αλλιώς αναφέρεται στην επιστημονική βιβλιογραφία, στο πεδίο της μετα-μάθησης έχουν οδηγήσει σε "εκμαθημένους" βελτιστοποιητές που ξεπερνάνε σε απόδοση τους καθιερωμένους αντιπάλους τους σε καθήκοντα στα οποία έχουν εκπαιδευτεί, έχουν επιδείξει περιορισμένα παραδείγματα όσον αφορά τις δυνατότητες γενίκευσής τους, δηλαδή την ικανότητά τους να αποδώσουν καλά και σε καθήκοντα που βρίσκονται εκτός των πλαισίων εκπαίδευσής τους. Ο στόχος αυτής της διπλωματικής εργασίας είναι να διερευνήσει τις ικανότητες γενίκευσης αυτής της μορφής μετα-μάθησης η οποία συνδέεται με βελτιστοποιητές υπό τη μορφή βελτιστοποιητών-ανατροφοδοτούμενων νευρωνικών δικτύων, επιχειρώντας έτσι να επεκτείνει προηγούμενες μελέτες στην εκμάθηση αλγορίθμων βελτιστοποίησης. Η εργασία αυτή περιλαμβάνει επίσης μια υλοποίηση των ανάλογων πειραμάτων σε PyTorch, ένα framework βαθιάς μάθησης το οποίο είναι ευρέως δημοφιλές ανά την ερευνητική κοινότητα και ο κώδικας θα είναι ανοιχτός, προκειμένου να διευκολυνθούν έτσι μελλοντικές συνεισφορές προς την απάντηση αυτού του ερευνητικού ερωτήματος.

Acknowledgements

First and foremost, I am grateful for my advisor Evangelos Markakis for his support and guidance during this brief but rather productive period that took for this thesis to be completed. I also thank him for providing me with the freedom to work on topics that constitute my scientific interests and which I regard as of great importance: generalization and learning to learn. Prof. Dr. Markakis trusted me with this thesis and, even though learning wasn't his area of expertise, led me towards a result for which I am proud of. I am sure that when I will look back on it in the future I will always remember it as the starting point of my scientific career towards tackling many interesting problems that exist out there waiting to be solved.

Secondly, I would like to thank Prof. Dr. Titsias who assisted me in matters related to machine learning. Our communication was excellent throughout the course of this thesis and I hope it will remain as such in the future. I would like to also thank all the committee members that agreed to be there during the defence of this thesis and, especially, the director of the MSc in Data Science, Prof. Dr. Vassilis Vassalos who agreed to let me undertake a research-oriented thesis instead of the default industrial one that was originally intended for all the students of the program.

Lastly, I would like to thank my family for all their support during this strenuous one-year period that spanned the duration of the course.

Contents

List of Figures

Chapter 1

Introduction

Machine Learning took its name in 1959 by Arthur Samuel and constitutes the scientific field that studies the algorithms and models that are used by computers in order for them to become increasingly better at performing a specific task. Thus, it addresses the question that Alan Turing posed in his seminal paper "Computing Machinery and Intelligence" (Turing, 1995) related to whether machines can imitate human learning, a process that is argued as being essential for building artificially intelligent systems. Machines can exhibit learning capabilities in a way that was formally defined by Tom M. Mitchell as: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"*.

It is generally accepted that the field of Machine Learning owes its success to the distinctive capability of its systems to automatically discover patterns in data. However, the very algorithms which empower that "learning" are still designed by hand. Therefore, it is natural to wonder whether the process of designing these algorithms could be cast as a learning problem and, given that this proves to be feasible, how do these learned optimizers fare against their handwritten counterparts. The field of study that deals with the aforementioned problems is often dubbed "learning to learn" or simply "meta-learning".

The reasons for opting to learn an optimization algorithm instead of using one that already exists are twofold: first comes the fact that many optimizers, the conception of which assumes convexity, are applied to problems that involve optimizing non-convex functions; this could be overcome by using optimizers that are learned under settings that are similar to the ones that they will be practically used at. Secondly, developing a novel optimization algorithm by hand is a lengthy and strenuous process that can take months or even years, thus delaying the progress of research; learning that algorithm could, instead, save us both time and manual labour.

However, there exist several different approaches to meta-learning which can be divided into three distinct classes:

- approaches towards learning *what* to learn the goal of which is to learn a flexible base-model that can be applied successfully on a variety of related tasks. This is done by distilling the important information that is common among these tasks into the parameters of this model.

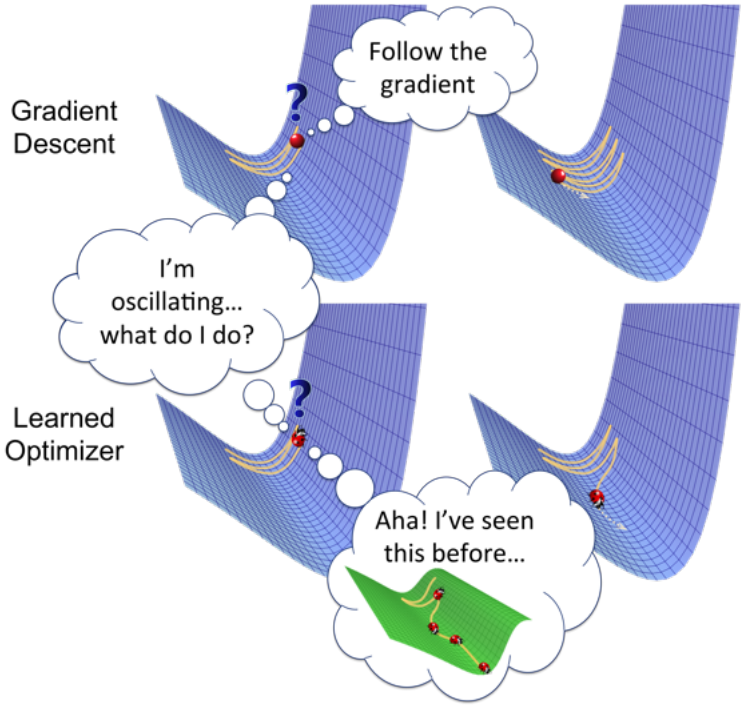

FIGURE 1.1: Intuition behind opting for meta-learned optimizers instead of hand-crafted ones (Source: https://bair.berkeley.edu/blog/)

- learning *which* model to learn, a strategy that is related with neural architecture search, i.e. the process of finding the ideal model architecture to tackle a specific problem.
- learning *how* to learn, which aims towards acquiring knowledge related to specific patterns that may be present in the behaviours of the different learning algorithms used and using it to define novel ones.

When thinking about meta-learning it is also important to contemplate the desired generalization capabilities of the final model. Generalization refers to the ability of a model to be able to maintain similar levels of performance on unseen test examples stemming from the same class from which the examples it was trained on were drawn from. In standard machine learning, a task is defined by a dataset that contains a set of examples and their corresponding labels (i.e. the classes to which they belong) which is used for model training. However, in the field of meta-learning the dataset is characterized by a meta-training set consisting of several different objective functions while the meta-test set is composed of objective functions of the same family. Therefore, in our case, generalization refers to the ability of a model (in our case an optimizer) to perform well on a variety of tasks.

The extent of generalization displayed by a learned optimizer can vary with relation to the way its train and test sets of tasks are shaped. For instance, if we neglect the need for generalization in exchange for an optimizer

that performs best in a single-case scenario we could evaluate its performance on the same objective functions that were used for its training. However, this may indicate that, instead of learning how to reach an optimum, an optimizer trained that way may instead memorize the optimum and, thus, converge to it at every single run. Memorizing optima can be a strenuous process that leads to learned optimizers that often take longer to converge than standard optimizers, therefore pressing the need for learned optimizers that are capable of generalizing to different objective functions.

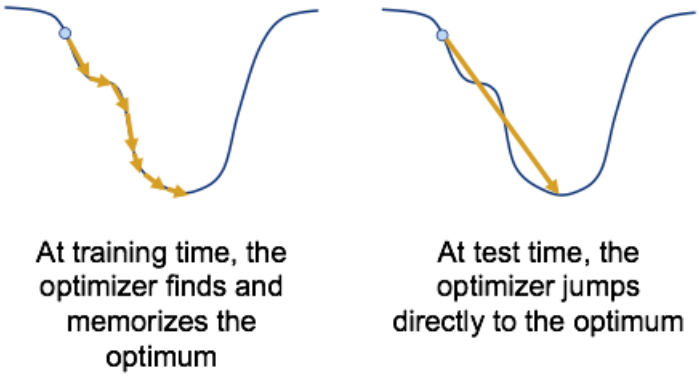

FIGURE 1.2: Example of optimum memorization (Source: https://bair.berkeley.edu/blog/)

The difference between the objective functions that constitute the train and test sets can vary. If we opt to check a learned algorithm's performance on tasks that are related then the memorization problem still remains, this time in the form of the optimizer memorizing parts of a model's parameters that refer to the commonalities between these tasks. Consequently, due to baring resemblances to the learning-*what*-to-learn approach discussed earlier, it could be argued that a learned optimizer that cannot generalize further lacks the ability to learn *how* to learn.

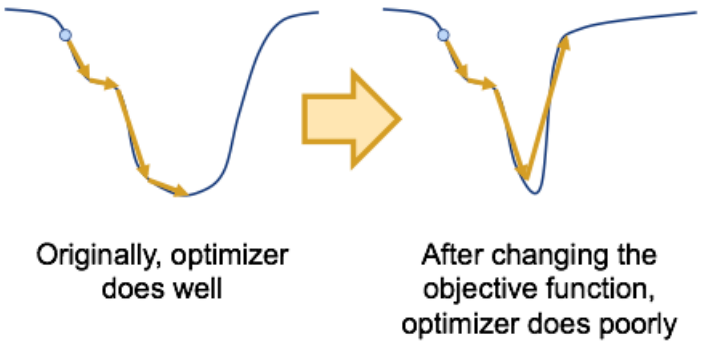

FIGURE 1.3: Example of a meta-learned optimizer that memorizes parts of model parameters (Source: https://bair.berkeley.edu/blog/)

In order to achieve learning *how* to learn our conception regarding generalization should be further solidified, by incorporating the need of learned

optimizers that perform well even when tested on tasks they are unfamiliar with. Parameter memorization in these settings is discouraged by the fact that the optimal neural network weights are entirely different between unrelated tasks. For example, it is likely that even the weight values of the shallow layers (i.e. layer that usually encapsulate generic information) between models trained on MNIST (LeCun and Cortes, 2010) and models trained on CIFAR (Krizhevsky, Nair, and Hinton, 2014) differ greatly. However, there is no point at identifying generalization with a learned optimizer's capabilities of performing well on dissimilar models, thus exhibiting tolerance on objective functions that are exotic to its training regime.

The purpose of this study is, thus, to extend previous approaches on meta-learning optimizers for neural networks (Andrychowicz et al., 2016) by evaluating their generalization capabilities on a variety of tasks that can be either *similar* or *dissimilar* to the ones that they're trained on. Should these optimizers perform well on dissimilar tasks it could be sufficiently proved that the learning *how* to learn paradigm discussed earlier is already followed in contemporary research efforts. On the other hand, should these learned optimizers perform well only on tasks that are similar to their training settings, it would mean that the corresponding studies resorted on applying a form of the learning *what* to learn paradigm. This would indicate that a meta-learning strategy that captures and uses information concerned with the exact process of learning eludes modern research, thus stressing the need of further studies towards this direction.

The structure of the rest of this thesis consists of four more chapters. In Chapter 2 a detailed overview of past and recent studies in the field of meta-learning is provided along with a detailed list of meta-learning strategies and fields of application. The meta-learning problem statement that is considered in this study and the approach we followed are located in Chapter 3 and the experiments that complement this study are presented in Chapter 4. Finally, the conclusions of our work are summed up in Chapter 5 which also contains speculative future work that could follow in the field of meta-learning.

Chapter 2

Related Work

2.1 Introduction

Versatility is, arguably, one of the cornerstones of human intelligence, as it allows us to be capable of performing well in a variety of different things via lifelong acquisition of the relevant skills and proficiencies. While this holds true for human intelligence it lies far from the truth when it comes to current AI systems that, despite achieving superhuman performance at mastering a single skill (Krizhevsky, Sutskever, and Hinton, 2012, Silver et al., 2016, Wu et al., 2016) they ultimately fail when trying to combine these different skills in a single system. Therefore, there is an apparent lack of adaptability in modern AI systems which, unlike humans, are not yet capable of performing well to new and unseen situations where their previously-acquired experience is of little use. But how could we overcome a problem like this and develop an AI that can acquire the aforementioned skills of adaptability and versatility?

When opting to build an AI system that is able to master a single skill from scratch the norm of providing it with large amounts of data and time in order to gain enough experience is followed. While this procedure is possible and desirable for tasks of limited scope, it is not affordable in a multi-task regime where we want our agents to acquire several different skills fast while harnessing previous experience instead of learning each task separately. Acquiring a skill from scratch is something that is rarely present in human intelligence, in which the procedure of learning a new task is always based on related tasks that we mastered earlier in our lives through experience and reuse of strategies that have proved to be successful in the past (Lake et al., 2016). As more and more skills are learned, humans become less and less dependent on the amount of examples needed to be provided during the learning phase of a new skill which, ultimately, involves less trial-and-error. This form of learning can be described as *learning how to learn* or simply *meta-learning* and is an essential step towards building intelligent agents that can achieve lifelong learning by replicating the aforementioned functions present in human learning.

The key challenge of meta-learning is learning to acquire a more efficient learning procedure by leveraging prior experience in a data-driven way (Vanschoren, 2018). The data used for this type of learning is often called *meta-data* and contains information about previous learning tasks and learned

models. This can include data about how the learning algorithms were configured (e.g. choice of hyperparameters, model architectures, etc.) and how the trained models were evaluated (e.g. performance metrics, parameter values, etc.). This data can be later used in order to provide valuable knowledge from which optimal models for new tasks can be learned. Therefore, we are trying to develop a strategy that promotes fast and data-efficient learning of a new task by leveraging prior experience concerning the learning procedure of other tasks. While it is important to note that data efficiency is highly dependent on the similarity between the new task and the previous ones, therefore implying no "free lunch" (Wolpert and Macready, 1997, Giraud-Carrier and Provost, 2005) presence, there are always ways to use prior experience in real-world tasks.

2.2 Meta-Learning in Neural Networks: A brief historical overview

Work in the field of meta-Learning began in the late 1980s and early 1990s with the now legacy works of Schmidhuber (Schmidhuber, 1987, Schmidhuber, 1992, Schmidhuber, 1993) that enhanced recurrent neural networks with the ability to alter their own weights. This was achieved by using these weights as additional input data, thus allowing the networks to reactively modify them upon observing their own errors. Despite this strategy facilitating the optimization of both the network and the training algorithm by simply applying gradient descent, its high computational requirements rendered it obsolete.

Later on (Bengio, Bengio, and Cloutier, 2002), the Bengio brothers tried replacing backpropagation with parametric rules which seemed more biologically-plausible for updating the network weights. These rules were learned using gradient descent or evolution across a selection of tasks.

Finally, the last study on meta-learning that signaled the end of its infancy period came with a subsequent work (Hochreiter, Younger, and Conwell, 2001) that introduced the use of LSTM (Long Short-Term Memory) networks (Hochreiter and Schmidhuber, 1997a) as multi-layer perceptron optimizers.

Despite its slow start, meta-learning has recently become a hot topic within the machine learning research community with works focusing on a variety of problems. These works include using meta-learning strategies for hyperparameter (Maclaurin, Duvenaud, and Adams, 2015) and neural network optimization (Li and Malik, 2017, Wichrowska et al., 2017, Chen et al., 2017), deep learning architecture search (Zoph and Le, 2016, Baker et al., 2016, Negrinho and Gordon, 2017), few-shot learning (Vinyals et al., 2016, Ravi and Larochelle, 2016, Finn, Abbeel, and Levine, 2017, Hariharan and Girshick, 2017) and speeding-up reinforcement learning (Duan et al., 2016, Wang et al., 2016, Finn, Abbeel, and Levine, 2017). Several of these approaches will be discussed in later sections of this work.

2.3 Meta-Learning Strategies

As it was previously noted, meta-learning involves a training phase involving a large number of tasks followed by a test phase on an entirely new task. This is an approach that is distinct from the standard approaches in machine learning that involve splitting the available data into two parts and then using them for training and testing the model respectively. Thus, in meta-learning there are two different optimization pipelines - optimizing the learner, which is responsible for learning new tasks and optimizing the meta-learner which is responsible for training the learner. The different methods for meta-learning can be categorized into one of four classes: recurrent models, metric learning, optimizer learning and model-initialization learning.

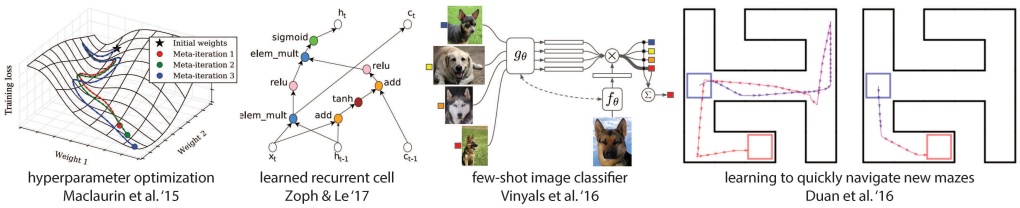

FIGURE 2.1: Various meta-learning applications (Source: https://bair.berkeley.edu/blog/)

2.3.1 Recurrent Models

This approach involves training a recurrent model, usually an LSTM (Hochreiter and Schmidhuber, 1997b), which is trained on a single task sequentially and is then exposed to new inputs produced from the distribution of that specific task. An example of this procedure in image classification would be to provide the recurrent model with the sequence of (image, label) pairs of a specific dataset, followed by novel examples that it has to classify.

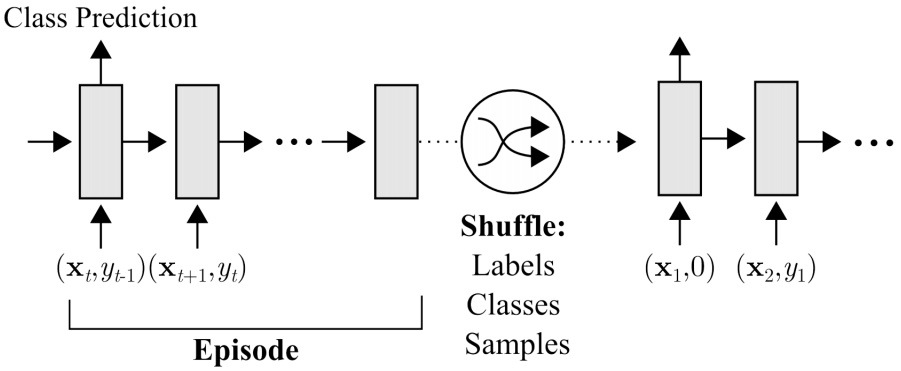

FIGURE 2.2: Example of the recurrent model approach where x_t are the input vectors and y_t are their corresponding labels (Source: Santoro et al., 2016)

In this scenario the meta-learner is trained using gradient descent, while the learner simply unrolls the LSTM. This strategy despite being the most

generic and widely used one (Santoro et al., 2016, Mishra et al., 2017, Duan et al., 2016, Wang et al., 2016) has shown to be the most inefficient one due to the absence of a learning strategy for the learner network which instead has to be learned from scratch.

2.3.2 Metric Learning

The goal of this strategy is to learn a metric space which can facilitate the learning procedure in general and has been mostly applied on few-shot classification tasks. The intuition behind this approach is that comparing an example image with the others that are available can enhance classifiers with sample-efficiency. However, as mere comparisons between images in pixel space seem to be unreliable, a joint metric space needs to be learned via a Siamese network (Koch, 2015). Here the meta-learner is trained using gradient descent while the learner takes the form of a comparison scheme in the aforementioned metric space. While this strategy has been successful on few-shot classification tasks (Vinyals et al., 2016), it has yet to demonstrate any results in other domains.

2.3.3 Optimizer Learning

Another approach involves learning an optimizer (Hochreiter, Younger, and Conwell, 2001) and is a method in which a meta-learner network (typically a recurrent network) learns to update a learner network in a way that facilitates the latter's learning. This meta-learning strategy has been applied on neural network optimization with general success (Li and Malik, 2016, Andrychowicz et al., 2016, Ravi and Larochelle, 2016, Li and Malik, 2017, Chen et al., 2017, Wichrowska et al., 2017). The merit of this approach is that, instead of having to choose among the different available optimization algorithms for neural networks that also requires extensive hyperparameter finetuning, the optimization process is instead learned from scratch. This form of meta-learning will form the central element of this thesis.

2.3.4 Model-initialization Learning

A final approach to meta-learning, inspired from the success of unsupervised pre-training in computer vision, is learning proper model initializations (Finn, Abbeel, and Levine, 2017). This strategy, dubbed Model-Agnostic Meta-Learning, instead of having to learn an optimization algorithm from scratch, it involves the learning of an initial representation space that can offer fast adaptability to new tasks.

2.4 Learning to Optimize

The approach to meta-learning that involved the learning of optimization algorithms began with studies by Li & Malik (Li and Malik, 2016) and Andrychowicz et al. (Andrychowicz et al., 2016) which independently proposed relevant frameworks that used reinforcement learning and supervised learning respectively. These studies focused on the existence of certain common elements among the different continuous optimization algorithms. These commonalities are:

- 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

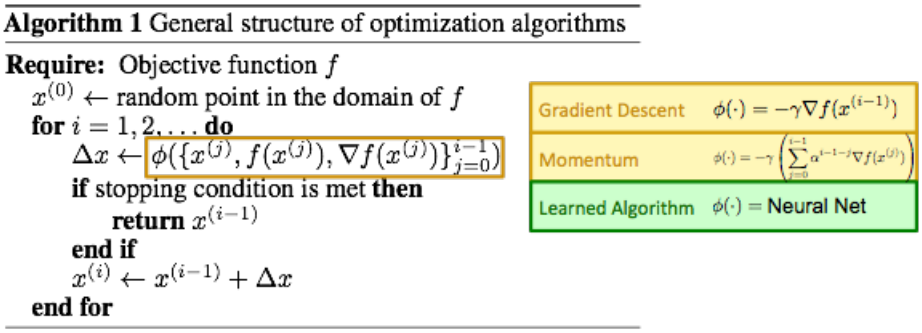

FIGURE 2.3: General structure of optimization algorithms

Despite certain common elements that do exist, a thing that varies from algorithm to algorithm is the form of the update formula. Therefore, learning this very formula is the ultimate objective of learning an optimization algorithm. The aforementioned approaches chose to model this update formula as a neural net, the weights of which would correspond to the learned optimizer. Neural networks were selected due to their representation capacity as they are proven to act as universal function approximators thus being able to model any update formula and due to their ability to allow for efficient search through a simplistic training process in the form of backpropagation. In order for the optimizers to be evaluated a meta-loss was introduced in the form of the sum of the objective function values during the whole training phase. However the learned optimizers that were generated from these works showed little examples of generalization capability.

While those earlier works solely focused on evaluating their learned optimizers on standard computer vision datasets such as MNIST a later work

(Chen et al., 2017) that used an approach similar to the previous two, i.e. meta-learning optimizers using recurrent neural networks, applied them on a variety of black-box optimization tasks and compared their performance with mainstream Bayesian optimization techniques. Despite outperforming the latter, training for very long horizons still proved to be a demanding task. Moreover, the inability of training the model with variable input dimension proved the choice of this strategy to be prohibitive in high dimensions where training optimizers for every dimension separately would not be feasible.

The first trial towards answering the aforementioned problems came through the work of Wichrowska et al. (Wichrowska et al., 2017) who used a hierarchical RNN architecture that proved to be more efficient in terms of memory and computation overhead and also more capable of generalizing to a variety of tasks. However, this study gave rise to new limitations relevant to the robustness of the learned optimizers as, in contrast to their hand-crafted counterparts, they seemed to fail to make any progress in later stages of the training phase. Moreover, the former severely underperformed the latter in terms of wall clock time thus rendering their small gains in terms of performance irrelevant.

In a subsequent work (Ravi and Larochelle, 2016) LSTM-based models for meta-learning were applied on several few-shot learning tasks by using an approach which allowed learning both a successful update rule for the devised optimizer and a robust initialization for the parameters of the learner. The approach of this study managed to outperform the relevant baselines and proved to be a competitive alternative to the then state-of-the-art in few-shot learning. However, it ultimately failed to satisfy the need for meta-learned optimizers that could generalize better by performing well in a variety of tasks, e.g. tasks with varying dataset sizes and varying classes, as its focus was limited on the few-shot and few-classes setting.

The last known study that focused on the "learning optimization algorithms" approach to meta-learning was compiled by the same team that initiated research in the field (Li and Malik, 2017). This work focused on trying to learn flexible optimization algorithms that could show some task-independence in a way that they could be adaptive to a variety of different settings by using a reinforcement learning strategy called guided policy search. The meta-learned optimizer that was proposed in this study was capable of both outperforming previous meta-learned optimizers trained using supervised learning (Andrychowicz et al., 2016) and generalizing to tasks that were unrelated to those it was trained on. However, its results were of limited scope as they were tested solely on shallow neural networks.

Chapter 3

Approaches for Meta-learning

3.1 Problem Definition

The norm in the field of machine learning is to 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 $\theta \in \Theta$. If the objective function describes the loss or regret during a specific point in time t we refer to the optimization problem as a *minimization* problem, i.e. a problem where the goal is to infer a method $\theta^* = argmin_{\theta \in \Theta} f(\theta)$ that minimizes the aforementioned objective function (in this case referred to as the loss function). The most usual way of finding such a method (usually referred to as a minimizer) is through the usage of gradient descent or any of its variants, a procedure which leads to a sequence of updates of the following form:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \tag{3.1}$$

Usually generic rules such as the above are not able to capture second-order information and are, thus, restricted to using gradients as their sole means of figuring out the optimal parameters for the minimizer. However, several efforts have been made towards designing rules that are better-suited to specific problem distributions. For instance, problems concerning the optimization of deep neural networks where the optimization space is high-dimensional and the objective functions are usually non-convex, several optimization algorithms such as momentum (Nesterov, 1983, Tseng, 1998), Rprop (Riedmiller and Braun, 1993), Adagrad (C. Duchi, Hazan, and Singer, 2011), RMSprop (N. Dauphin et al., 2015) and ADAM (Kingma and Ba, 2014) have surfaced.

The goal of this study is to provide alternatives to those aforementioned commonly used hand-designed update rules which, albeit performing well by exploiting the structure of the problems, they fail to generalize to tasks that lie outside of that scope. In order to do so, we propose to learn these update rules which we will, from now on, define as the **optimizer** model m_ϕ. This optimizer can be specified by a set of parameters ϕ. This rule can be used to update the parameters of our predefined loss function (referred to from now on as the **optimizee**) in the following form:

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi) \tag{3.2}$$

This intuition behind the equation above is that the optimizer m_ϕ is constantly provided with information regarding the performance of the optimizee f and, by constantly updating its parameters ϕ, it varies its update rule proposals in order to infer the optimal update rule for updating the optimizee parameters θ, thus maximizing its performance.

3.2 Learning Optimizers using Recurrent Neural Networks

As we previously stated, we opt to replace hand-written optimizers by incorporating a learned update rule to the generic form of an optimization algorithm we described in Figure 2.3. In our case, this learned update will take the form of a recurrent neural network (RNN) or, more specifically, it will be a LSTM. Having a RNN variant take the role of the optimizer can be justified by the fact that gradient descent is, at its very essence, a sequence of updates on separate states that are present in-between. Therefore, instead of using one of the normally used optimizers, we train a RNN to model a new one.

The reason behind our choice of using a RNN instead of a simple neural network is because we want our optimizer to reason about previous events that occurred during the optimization process and use that information towards shaping better update rules in the future. Recurrent neural networks, due to their ability to form loops with themselves, allow information to persist, a feature that is not found in plain neural networks. As we can see in Figure 3.1 a recurrent neural network is essentially a collection of copies of the same network which passes information from earlier points in time to later ones. Thus, the rolled RNN in the left part of Figure 3.1 during the time step t can be thought of as being equivalent to its depiction in the right part of the same Figure that pictures its loop unrolled through time steps 0 to t. Therefore the first requirement of having an optimizer architecture that supports sequential updates is, thus, met.

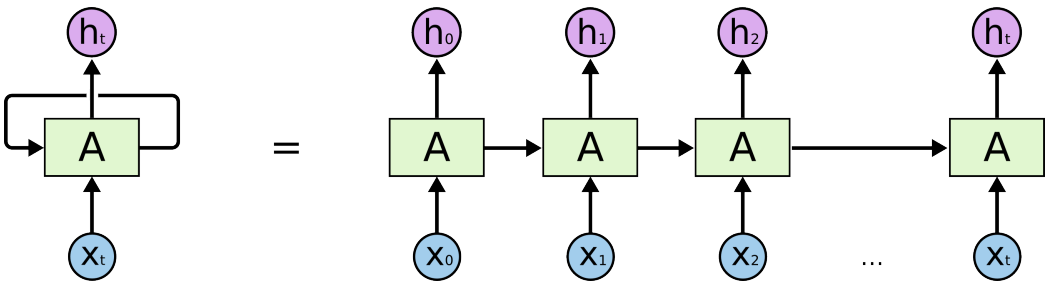

FIGURE 3.1: An example of recurrent neural network with some input x_t and output h_t in its rolled (left part of figure) and unrolled state (right part of figure) (Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

However, in order for our optimizer to work we need it to be able to maintain information about its previous states, i.e. information about the way its

previous suggested updates affected the performance of the optimizee. Due to this, instead of choosing a simple RNN as our optimizer, we opted for one of its variants the Long Short-Term Memory network, or simply LSTM. The LSTM's most prominent characteristic which is called the cell state (i.e., a memory-like feature that allows the network to store and access information) allows an optimizer of the same architecture to satisfy the second requirement of having access to previous states during its learning phase. The satisfaction of this requirement ensures that an optimal update rule is being learned based on previous experience. The idea that knowing a history of gradients would be beneficial to the proposed gradient updates was inspired by the way momentum (Nesterov, 1983) works.

In order to stress that our problem lies on training the optimizer, we are going to directly parameterize it. Therefore, the final *optimizee parameters* can be written as $\theta^*(f, \phi)$ where ϕ are the optimizer parameters and f is the function we are trying to optimize (or simply the optimizee). The expected loss with relation to the optimizer can be written as:

$$L(\phi) = E_{f \longleftarrow D}\Big[f(\theta^*(f, \phi))\Big] \tag{3.3}$$

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 ϕ, while its state at time t can be denoted as h_t. Since the function in Equation (3.3) only relates the expected with the final parameter values of the optimizer it is important to alternatively model our objective to be dependent on the entire optimization trajectory for a given time frame T. Therefore we can model it as such:

$$\begin{aligned} L(\phi) &= E_f\Big[\sum_{t=1}^{T} w_t f(\theta_t)\Big] \\ &\text{where,} \\ \theta_{t+1} &= \theta_t + g_t \\ \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} &= m(\nabla_t, h_t, \phi) \end{aligned} \tag{3.4}$$

In the definition above, $w_t \in R_{\geq 0}$ are weights that correspond to each time-step t, so that formulations in Equations (3.3) and (3.4) can be the same for $w_t = \mathbf{1}[t = T]$. However, the gradient of the objective function will be non-zero only when $w_t \neq 0$ which means that, since $w_t = \mathbf{1}[t = T]$, the only time-step for which this holds true will be the final optimization step, thus rendering Backpropagation Through Time (BPTT) inefficient. Therefore, it is important to ensure that the gradients of the timesteps $t \in [0, T-1]$ are also non-zero, in order to also gain information about all the previous gradients along the optimization trajectory. We can fix this issue by relaxing our objective function so that $w_t > 0$ along this trajectory and, for simplicity purposes, it is assumed in our experiments that $w_t = 1$ for every time-step t. Lastly, we

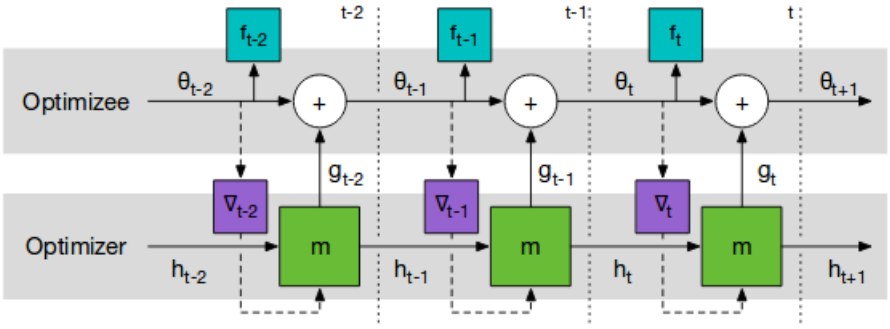

FIGURE 3.2: Computational graph used for computing the optimizer gradients (Source: Andrychowicz et al., 2016)

refer to the gradient of the optimizee f with respect to its parameters θ at time-step t, $\nabla_\theta f(\theta_t)$ as simply ∇_t, again, for simplicity purposes.

An approach towards minimizing our objective $L(\phi)$ is to apply gradient descent on ϕ by computing the gradient estimate $\partial L(\phi)/\partial\phi$ using a randomly sampled function f and applying backpropagation. The computational graph on which we are going to apply the backpropagation algorithm can be seen in Figure 3.2 where two distinct types of lines can be observed. The solid lines refer to the parts of the computation graph where gradient flow is allowed, whereas the dashed lines correspond to the parts where it is dropped. Despite our formulation indicating that the optimizee gradients ∇_t are dependent on the optimizer parameters ϕ this has been proven to over-complicate the process of computing the optimizer's gradients. Therefore we assume that the contribution of these gradients can be negligible and, thus, can be ignored, i.e. $\partial\nabla_t/\partial\phi = 0$. This simplification allows us to avoid second-derivative computations related to the objective function, which is a quite expensive task, and does not seem to hamper the optimizer training as indicated by our results later on.

3.2.1 Parameter Sharing & Preprocessing

Due to our optimizer network needing to provide updates for deep neural networks consisting of tens of thousands of parameters, thus imposing hidden state and parameter requirements of an enormous scale, we decided to use the same strategy of parameter sharing employed in (Andrychowicz et al., 2016) and (Ravi and Larochelle, 2016) which is called *coordinate network architecture*. This approach allows our LSTM optimizer to operate *coordinate-wise* on the objective function parameters in a fashion similar to the way RMSprop and ADAM updates work. This is achieved by employing a small network for every input coordinate (i.e. the gradient w.r.t. a single distinct optimizee parameter) the hidden and cell state values of which are isolated from those of the rest networks which, however, share the same parameters. Thus, not only is each coordinate updated by the same rule that depends only on its respective history but it also allows the optimizer to be invariant

to the order in which its parameters are updated. An instance of the LSTM optimizer described here can be viewed in Figure 3.3.

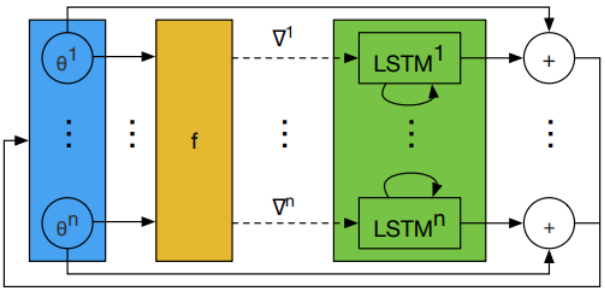

FIGURE 3.3: Example of coordinatewise LSTM optimizer. Notice that all LSTM parameters are shared while hidden states are separated. (Source: Andrychowicz et al., 2016)

Despite hand-written optimizers being robust to extremely varying input and output magnitudes, deep neural networks, which naturally disregard small input values in favor of bigger ones, are susceptible to gradient explosion, i.e. the problem where high gradient values accumulate resulting in extreme parameter updates during the training phase which render it unstable thus preventing it from learning. Due to this problem being more prevalent in our architecture of different input coordinates we opted to use a preprocessing schedule for normalizing the optimizer's inputs (both gradients and loss history). We, thus, applied the same preprocessing formula that was used in (Andrychowicz et al., 2016) and (Ravi and Larochelle, 2016)

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

where $p > 0$ is a parameter indicating the threshold under which gradients are disregarded. This preprocessing also allows us to separate the information regarding their magnitude from their sign, the latter being only meaningful for gradients.

In the next chapter, several comparisons between the LSTM optimizer described here and its standard counterparts will follow with respect to their performance on several datasets. Comments on their generalization capabilities will also be provided when applicable.

Chapter 4

Experiments & Results

4.1 Experimental Setup

Here some details regarding our experimental setup will be provided. The model architectures, the learning methods used as well as the hyperparameters selection are all addressed in the first two paragraphs, while details about the comparisons that follow in Sections 4.2-4.5 are provided in the third paragraph.

All trained optimizers in our experiments constitute of two LSTM layers each layer of which consists of 20 hidden units. These optimizers where trained based on the minimization criterion of Equation (3.4) using the variant of backpropagation detailed in Section 3. For minimizing the objective we used the ADAM optimizer with a learning rate chosen by performing random search on a specific range of values $\in [1.0, 10^{-5}]$.

Whenever possible during the training phase, we used early stopping in order to prevent the optimizer from overfitting. Upon the end of each epoch, which consists of 100 training iterations, we evaluate the performance of the optimizer, the parameters of which are frozen during this process. The best optimizer is selected based on its final test loss and its average performance is reported based on an entirely novel sampled test set.

Our meta-learned optimizers are compared against a selection of standard hand-written ones that are commonly used in Deep Learning tasks, namely SGD, RMSprop, ADAM and SGD with Nesterov Momentum. The learning rates of these optimizers were fine-tuned for each problem and the results that are reported correspond to the best-scoring of them in terms of their task-specific final error. When there were more hyperparameters involved other than the learning rate, the default values from the optim package in Torch7 were used. Finally, imitating the Deep Learning trends, we chose to initialize the optimizee parameter values by sampling from an IID Gaussian distribution.

The code for our experiments is made available through the following link on Github: https://github.com/kostagiolasn/MetaLearning-MScThesis under the MIT License.

4.2 Quadratic Loss Functions

A method of evaluating the performance of a model at fitting a dataset that is standard in the machine learning community comes in the form of loss functions. High deviance between the predictions of our model and the actual values within the dataset corresponds to high loss function values. However a model can learn to fit the data better, thus providing more accurate predictions, if assisted by an optimizer whose purpose is to facilitate the minimization of the loss function. As it is desirable for a loss function to intuitively represent the error rate between the predictions of a model and the corresponding real values, it is usually expressed as a quadratic form in the deviations of the variables of interest and their desired values. This approach is tractable because it results in linear first-order conditions.

Thus, at first, we considered the task of optimizing synthetic quadratic functions defined in the 10-dimensional space. In this particular setting the minimization objective was simply a squared error loss in the form of:

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

where W and y correspond to 10x10 matrices and 10-dimensional vectors respectively, sampled from an IID Gaussian distribution. We trained our optimizers on randomly created functions from this distribution and tested them on freshly sampled functions of the same family. Each objective was optimized for 100 iterations, while the meta-learned optimizers were trained for 20 iterations, i.e. we are updating the trained optimizer's parameters every 20 steps by unrolling the LSTM. The total epochs also amount to 100. No form of preprocessing or postprocessing was necessary for this specific task.

Results for this task are shown in 4.1 in the form of learning curves one for each of the optimizers used. These curves were produced by averaging the performances of each optimization algorithm over a large number of test functions. Our meta-learned optimizer's performance corresponds to the solid learning curve, while the dashed learning curves display the performance of its standard baseline counterparts. By inspecting the graph it is clear that, in this specific scenario, our meta-learned optimizer considerably outperforms the baselines.

4.3 MNIST and Related Datasets

Following the previous experiments we decided to examine whether meta-learned optimizers can be applied to neural network optimization tasks. We, therefore, opted to learn to optimize a small neural network consisting of a single hidden layer of 20 units on the MNIST (LeCun and Cortes, 2010) dataset, which we later tested on different network architectures, training procedures and datasets.

The MNIST handrwitten digits dataset consists of 60.000 training examples and 10.000 test examples of 28x28 greyscale images, each one displaying

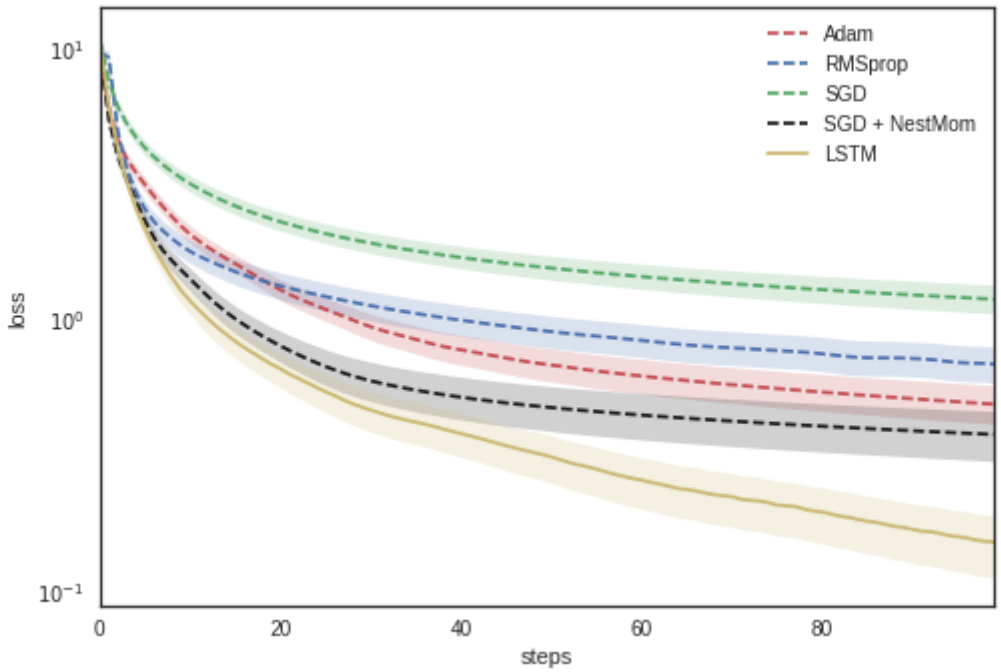

FIGURE 4.1: Comparison between standard optimizers and learned optimizer (LSTM) - Black-box 10-variable quadratic function

a handwritten number digit. An example of the dataset instances and their corresponding labels can be seen in Figure 4.2.

The minimization objective f used in this task was the crossentropy of Multi-layer Perceptron, paremeterized by θ:

$$f(\theta) = -\frac{1}{n}\sum_{c=1}^{M} y_{o,c} log(p_{o,c})$$

where M is the number of classes (here $M = 10$), y is a binary indicator vector where $y_{o,c} = 1$ if class label c is the correct classification for observation o and p is the neural network output probability for observation o belonging to class c. The choice of this function was facilitated by its wide use in classification settings. Intuitively, as the predictions of a model diverge from the real class label of the corresponding example, the cross-entropy loss increases.

We estimated both the values of the objective function f and its gradients $\partial f(\theta)/\partial\theta$ using mini-batches of randomly selected 128 examples. We used the sigmoid activation function for the hidden layer units and we initialized the network's parameters θ by randomly sampling from an IID Gaussian distribution. In order to be consistent with previous experiments we used 100 iterations for optimizing the optimizee, and updated the optimizer every 20 iterations. For time-efficiency purposes we used a total of only 20 epochs, a decision which didn't seem to affect the procedure at all. The preprocessing strategy described in Section 3 was also used for rescaling the outputs of the LSTM by a factor of 0.1.

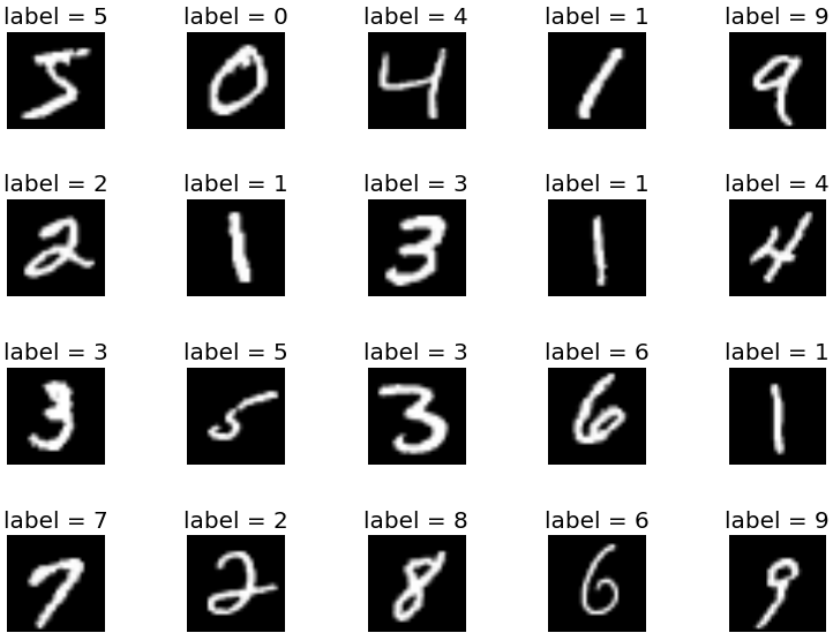

FIGURE 4.2: Examples from the MNIST dataset and their corresponding labels (Source: https://corochann.com/mnist-dataset-introduction-1138.html

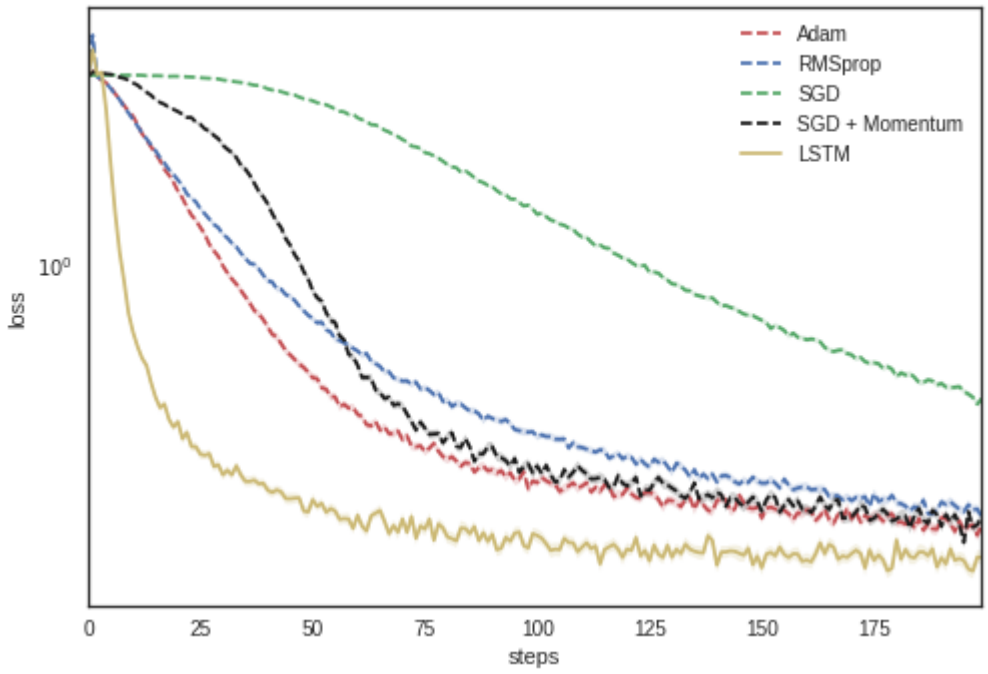

FIGURE 4.3: Comparison between standard optimizers and learned optimizer (LSTM) - MNIST

The learning curves for this specific setting are shown in Figure 4.3. While

the performance exhibited by RMSprop, ADAM and SGD with Nesterov momentum in this task was roughly the same and plain SGD failed to achieve a notable performance, our meta-learned optimizer outperforms these baselines by a significant margin.

4.3.1 Generalization to Different Architectures

Furthermore, we decided to inspect the generalization capabilities of our meta-learned optimizer to different network architectures. Indeed, our meta-learned optimizer managed to outperform the baselines on the MNIST dataset, despite it being trained on a different setting architecture-wise. The relevant performance comparisons are shown in Figures 4.4, 4.5 in which a network consisting of 40 hidden units instead of 20 and a network with two hidden layers were used, respectively. The key feature of these experiments is that our meta-learned optimizer is very robust in terms of altering the neural network architecture, especially when more than one layers are stacked simulating current Deep Learning trends.

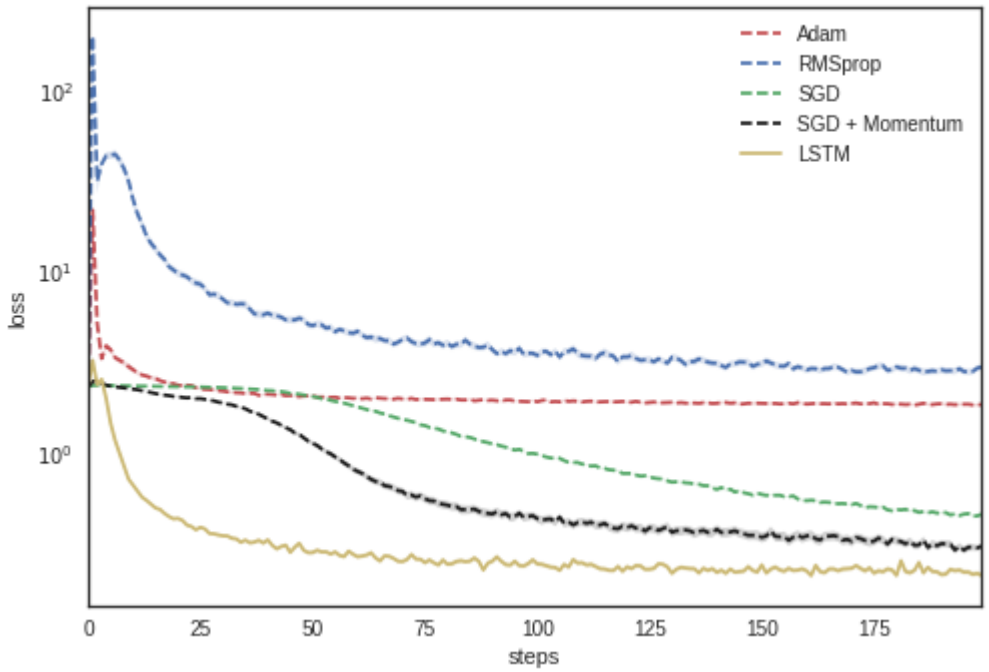

FIGURE 4.4: Comparison between standard optimizers and learned optimizer (LSTM) - MNIST (40 hidden units)

4.3.2 Generalization to Different Learning Dynamics

We also provided an experiment setting in which we altered the activation function of our neural network, thus changing the learning dynamics. In this specific task, our meta-learned optimizer seemed to show a considerable amount of generalization capabilities, despite failing to outperform the

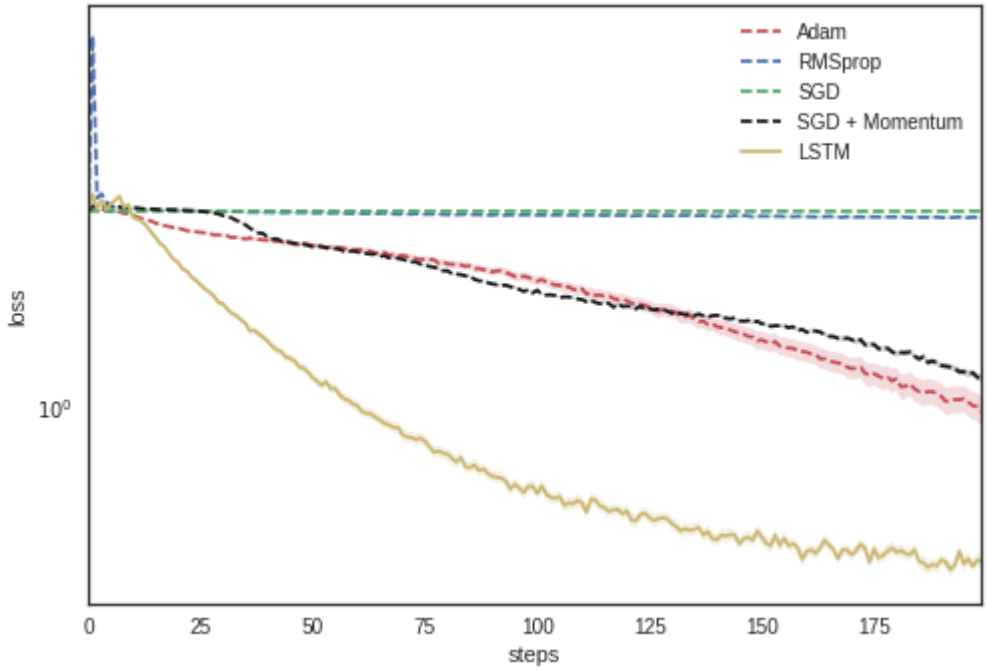

FIGURE 4.5: Comparison between standard optimizers and learned optimizer (LSTM) - MNIST (2 hidden layers)

SGD optimizer and its Nesterov Momentum variant. The relevant results are shown in Figure 4.6.

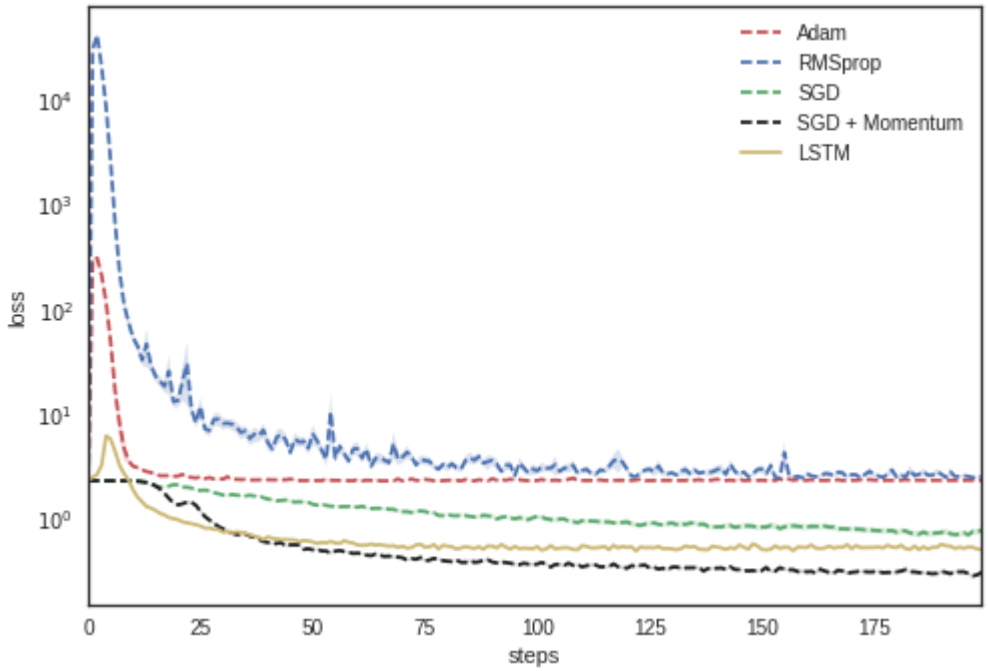

FIGURE 4.6: Comparison between standard optimizers and learned optimizer (LSTM) - MNIST (ReLU activation function)

4.3.3 Generalization to Different MNIST-like Datasets

Following the previous experiments we decided to test the generalization capabilities of our MNIST meta-learned optimizer when applied to tasks related to MNIST. Should our optimizer perform reasonably well in these settings it would mean that, instead of limiting itself by adapting solely on the particularities of the MNIST dataset, it is capable of generalizing its performance to a selection of optimization landscapes that are similar to it. It is important to note that all the baseline optimizers were trained and had their learning rates fine-tuned on these specific tasks in contrast with our learned optimizer that was trained on the original MNIST dataset and is, thus, operating outside of its training regime.

First of all we tested our learned optimizer's performance on the FashionMNIST (Xiao, Rasul, and Vollgraf, 2017) dataset. Although MNIST bares many similarities to MNIST in consisting of 60.000 test examples and 10.000 test examples in the form of 28x28 greyscale images, each one of which belongs to 10 categories, it is anything but redundant. More specifically, FashionMNIST, as one can deduce from its name, consists of examples belonging to different kinds of clothes, thus encapsulating a harder problem than recognizing digits and represents more broadly modern computer vision tasks. Examples from the FashionMNIST data are shown in Figure **??**.

The learning curves for this specific task are depicted in Figure 4.8. By examining this graph it is made clear that our meta-learned optimizer outperforms all of the baseline optimizers and, thus, projects considerable generalization capabilities when applied on novel settings that correspond to similar optimization landscapes to the ones it was trained on.

The next dataset on which we tested the performance of our meta-learned optimizer on MNIST was the EMNIST (Extended MNIST) dataset (Cohen et al., 2017). This set of datasets was created in order to form a selection of more challenging classification tasks for benchmarking modern computer vision systems. It consists of 28x28 greyscale images of handwritten number digits or letters, thus sharing the same structure and parameters with the original MNIST task. In order to differentiate a bit from MNIST we decided to use the EMNIST Letters subset of EMNIST which consists of 145.600 character images belonging to 26 letter classes. Examples of the EMNIST Letters dataset are shown in Figure 4.9. In this experiment we opted for a train-test split of 80-20% respectively.

The learning curves for this specific task are depicted in Figure 4.10. In this experiment we can see that the meta-learned optimizer on MNIST learns more rapidly than the baseline optimizer but its generalization capabilities seem to reach a plateau after epoch 100. Despite this fact, our meta-learned optimizer is outperformed only by the SGD optimizer with Nesterov momentum and even that happens during the latest stages of the experiment, thus indicating that a certain degree of generalization is possible. Furthermore, it should be stressed that having the standard optimizers trained on a dataset almost double the size of MNIST could have played a pivotal role in enabling them to gain much more task-specific information than the meta-learned optimizer and, consequently, perform better. Another factor that

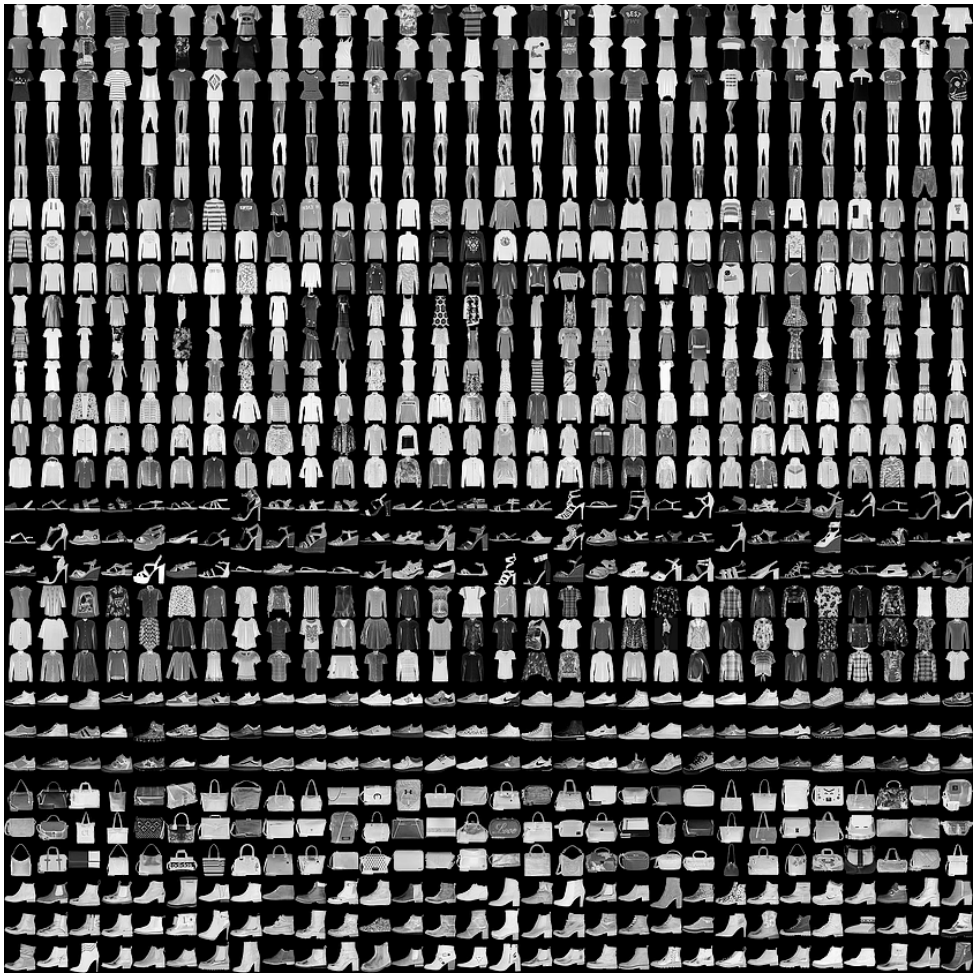

FIGURE 4.7: Examples from the FashionMNIST dataset. Each class takes three rows. (Source: https://github.com/zalandoresearch/fashion-mnist)

possibly prevented the meta-learned optimizer to generalize could be that the number of classes between the MNIST and the EMNIST dataset was very different, with the former consisting of examples belonging to 10 classes and the latter consisting of examples belonging to 26 classes.

4.4 More Complex Datasets

4.4.1 Generalization on Entirely Different DMMNBatasets

Lastly, we decided to test the generalization capabilities of our MNIST meta-learned optimizer in tasks that are entirely different than those examined before. More specifically we examined its performance on the widely-used CIFAR-10 (Krizhevsky, Nair, and Hinton, 2014) and SVHN (Netzer et al., 2011) datasets.

The CIFAR-10 dataset consists of 60.000 32x32 coloured images divided in 10 classes (6.000 image examples per class) the 50.000 of which comprise the training set while the rest 10.000 comprise the test set. It is important to

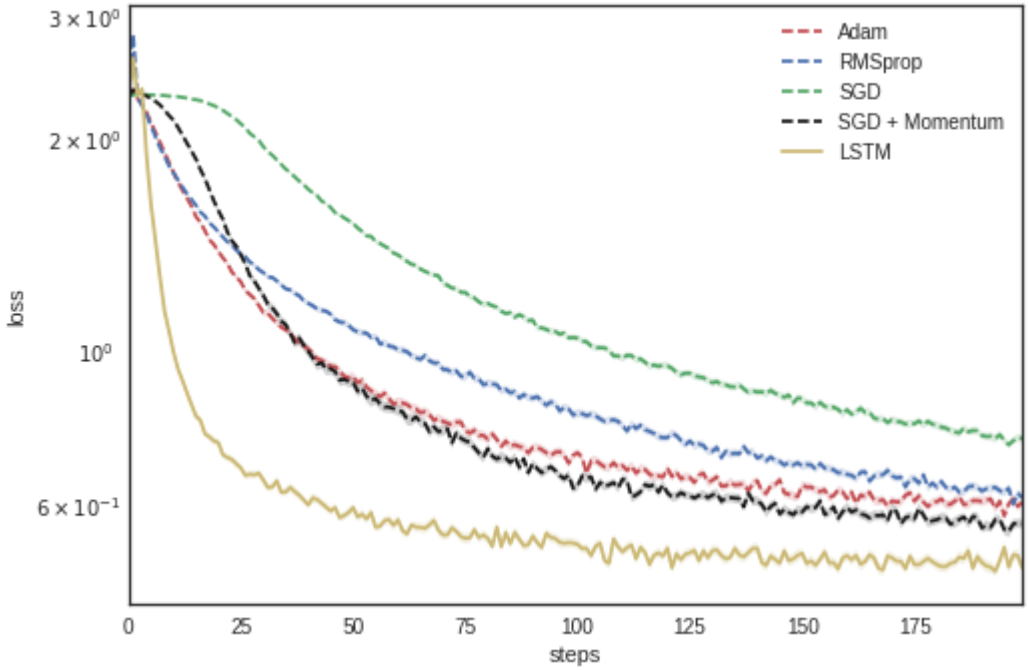

FIGURE 4.8: Comparison between standard optimizers and learned optimizer (LSTM) - FashionMNIST (LSTM optimizer was trained on MNIST)

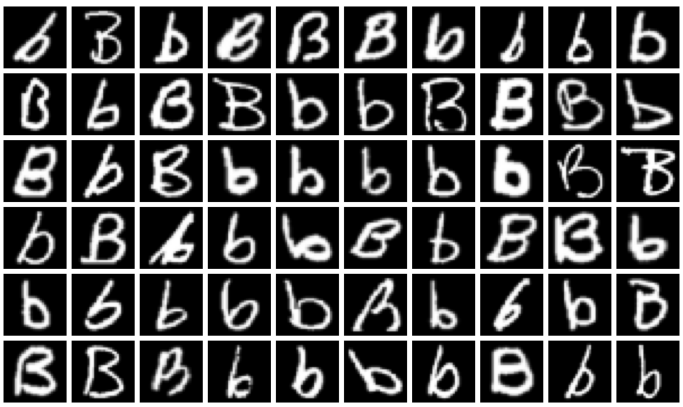

FIGURE 4.9: Examples from the EMNIST Letters dataset. Here examples that depict the letter B are shown (Source:https://statsmaths.github.io/stat395-f17/class20/).

note that these examples depict non-overlapping real-world objects, thus belonging to classes that are mutually exclusive, and have nothing in common in terms of structure and parameters with the examples of the MNIST-like datasets.

The SVHN (Street View House Numbers) dataset is another real-world 32x32 coloured image dataset that consists of examples that are semantically similar to those of MNIST, in that they are small images of cropped digits. However, the amount of data it incorporates is an order of magnitude larger than that of MNIST, since it comprises of a training set of 73257 and a test

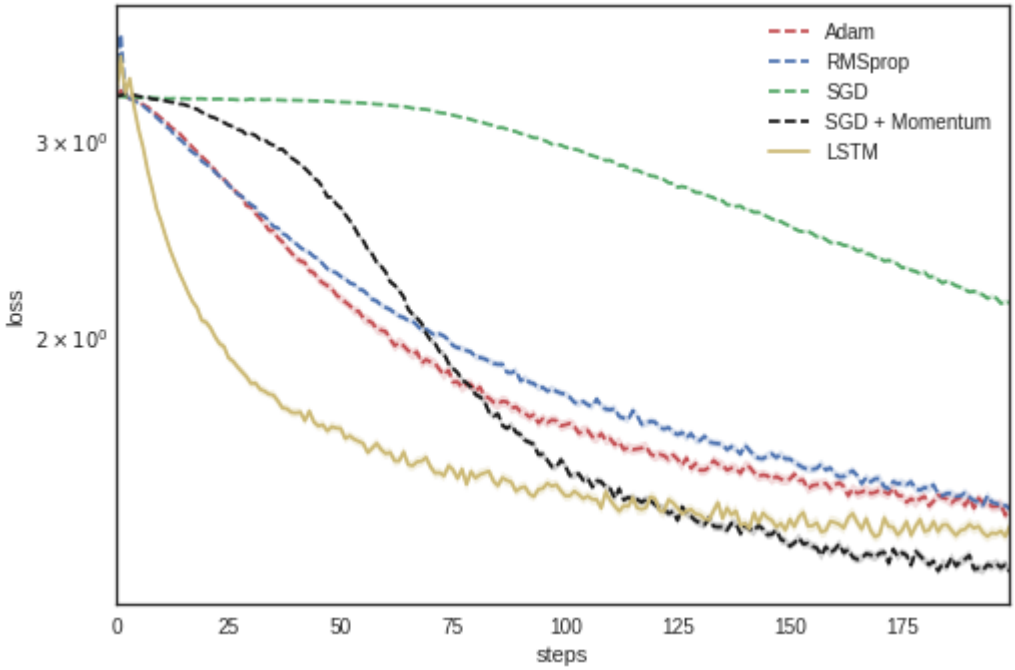

FIGURE 4.10: Comparison between standard optimizers and learned optimizer (LSTM) - EMNIST Letters (LSTM optimizer was trained on MNIST).

set of 26032 images. Furthermore, in contrast to MNIST, the SVHN dataset is related to a real world problem that still remains unsolved. Examples for each of these datasets can be viewed in Figures and respectively.

The results of the CIFAR-10 image labeling task in terms of the learning curves of each optimizer tested are shown in Figure 4.13. The takeaways from this specific experiment is that, although the learned optimizer on MNIST outperforms both Adam and RMSprop it ultimately fails to outperform SGD and its Nesterov momentum variant that were trained on this setting, despite displaying roughly the same performance during the early parts of the training phase. It should be also noted that, apart from a relative fluctuation early on, both the learned optimizer and those that outperform it seize to display any difference in terms of learning performance, a feature that can be attributed to the low representation power of the neural network architecture employed. Nevertheless, it can also be observed that the difference margin in terms of performance between the three top-performing optimizers is very small.

The learning curves of each optimizer for the SVHN image labeling task are displayed in Figure 4.14. Although the performance of the learned optimizer on MNIST is shown to keep up with that of the baseline optimizers trained on this task during the early stages of the training phase, it ultimately manages to just keep up with the performance of the SGD optimizer. The assumptions about it failing to outperform the other standard optimizers which were discussed in the previous paragraph can be also transferred to this setting. A notable difference between this and the previous task is that

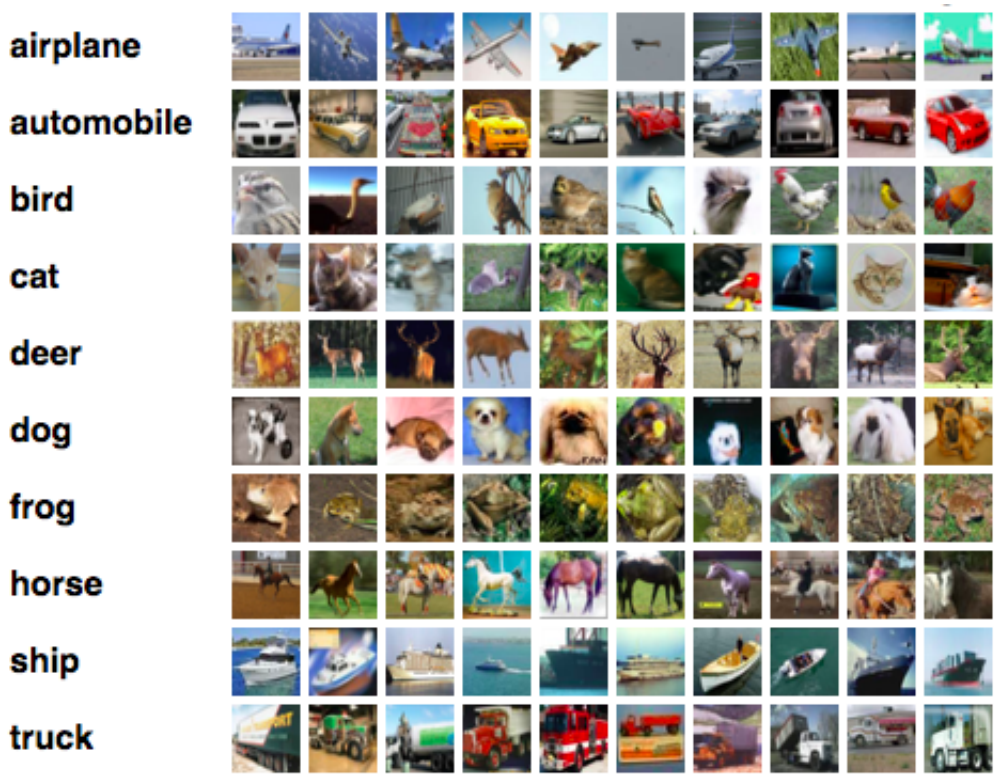

FIGURE 4.11: Examples from the CIFAR-10 dataset along with their corresponding labels (Source: https://www.cs.toronto.edu/ kriz/cifar.html).

the network architecture seems to have enough representation power for the problem as the top-performing optimizer, SGD with Nesterov Momentum, appears to increase its performance as the training phase progresses.

Ultimately, the meta-learned optimizer on MNIST didn't show enough generalization capabilities to outperform the baseline optimizers either when tested on CIFAR-10 or on SVHN. This can be attributed to two facts: firstly, it is natural for optimizers that use task-specific knowledge to achieve greater performance in those very tasks than optimizers trying to generalize to unfamiliar tasks to them (different number of parameters and image structure), thus leading to an unfair comparison between them. Secondly, it could also be the case that the optimization landscapes corresponding to these tasks are simply way too exotic for optimizers that were learned and, thus, adapted on unrelated tasks to handle. All in all, when applied on tasks stemming from the same distribution, the learned optimizer seemed to generalize well and achieved notable cross-task performance that in most cases resulting in it outperforming the baseline hand-written optimizers.

FIGURE 4.12: Examples from the SVHN dataset (Source: http://ufldl.stanford.edu/housenumbers/)

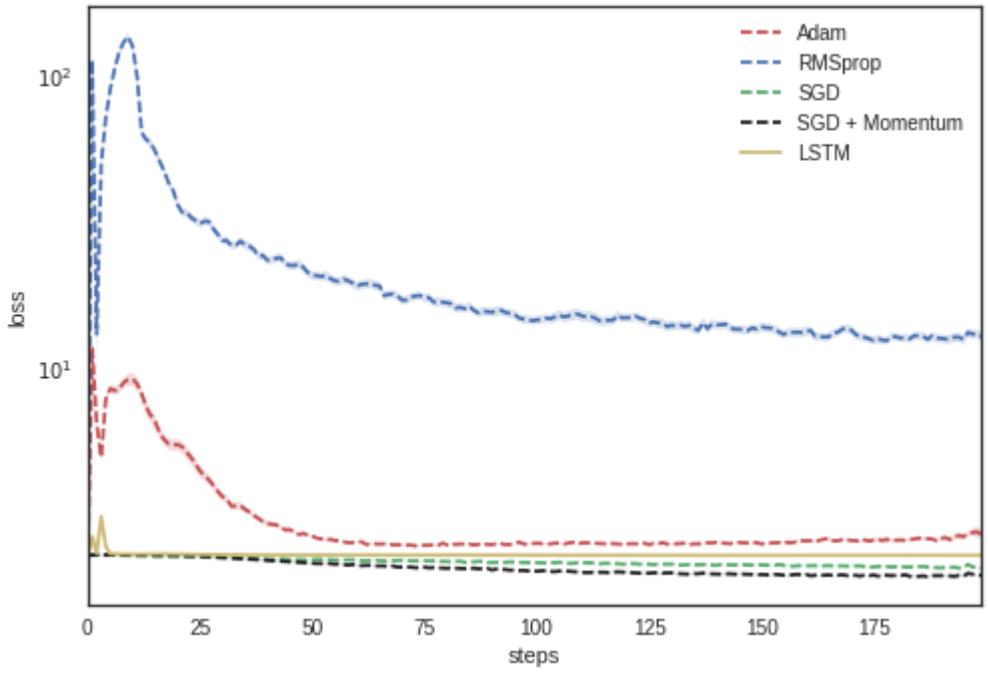

FIGURE 4.13: Comparison between standard optimizers and learned optimizer (LSTM) - CIFAR-10 (LSTM optimizer was trained on MNIST).

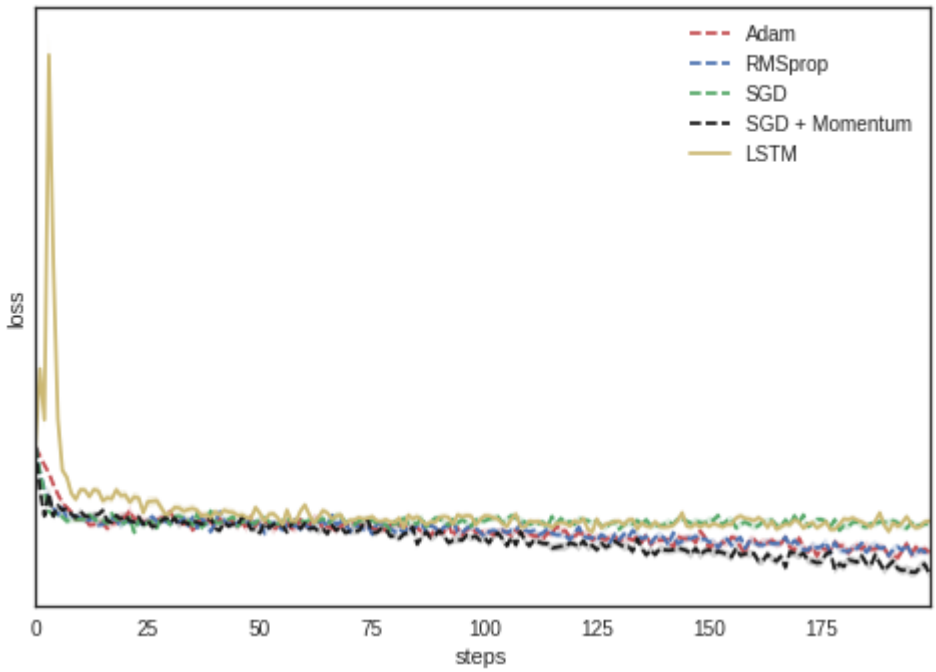

FIGURE 4.14: Comparison between standard optimizers and learned optimizer (LSTM) - SVHN (LSTM optimizer was trained on MNIST).

Chapter 5

Conclusion

5.1 Discussion

The aims of this study were twofold: the first objective was to extend earlier work (Andrychowicz et al., 2016) which demonstrated that optimizer design could be casted as a learning problem by testing an optimizer's generalization capabilities at optimizing functions stemmed from the same distribution. The second objective was to 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.

From our experiments it is easily made clear that when applied on the same family of functions to which it was trained on, a learned optimizer generally outperforms the baseline hand-written optimizers that are extensively used in literature by a large margin. This was showcased by the generalization performance of our learned optimizer on MNIST when tested on the MNIST-like datasets FashionMNIST and EMNIST that contained image examples whose structure and parameters had very much in common with MNIST. Our learned optimizer consistently outperformed the standard optimizers that were trained on these very tasks despite acting outside of its training regime.

Furthermore, the meta-learned optimizer also showed a remarkable degree of transfer when changes were made in the network architecture-wise in the MNIST task. However, the same can't be said for the tasks that involved testing the optimizer on unfamiliar settings that corresponded to real-world object recognition tasks unrelated to MNIST. The datasets CIFAR-10 and SVHN that contained very differently-structures images and were related to optimization problems with a vastly larger number of parameters prevented the learned optimizer from generalizing, thus confirming the No-free lunch theorem.

All in all, however, it is natural to say the learned optimizer's superiority on related tasks is a fact that reinforces the idea of promoting the choice to train neural learned optimizers instead of opting for hand-written ones in most single-task scenarios, which form the majority of the settings in deep learning literature.

5.2 Future Work

A feature that eludes modern meta-learned optimizer is that of generalizing to unrelated tasks. Being able to transfer knowledge between tasks is a key element of human intelligence and could lead to lifelong learning models replacing single-task ones that demand being trained from scratch for each task they encounter due to catastrophic forgetting.

A possible source of inspiration could be the field of developmental psychology, in which infant learning is thoroughly examined. Recent studies (Smith and Slone, 2017) have showcased that, until 2 years old, humans are bombarded with images that comprise almost entirely of human faces, therefore indicating to a visual experience that corresponds to an abundance of examples belonging to a single class of objects that are not labeled by any supervisor. As humans grow older their visual experience is altered, entailing the learning of visually recognizing objects belonging to a vast number of classes through limited paradigms and scarce, but prevalent, labeling. Humans, however, that did not come in contact with faces earlier in their life, mainly due to visual impairments, seem to develop problematic visual recognition systems (Maurer, Mondloch, and Lewis, 2007), thus indicating the importance of these earlier visual encounters at shaping the visual recognition systems in our brains.

Developing feature extraction processes from largely unlabeled data is a thing that eludes solution in modern machine learning literature and is the main target of study in the field of unsupervised learning. Progress in this field could, thus, possibly facilitate imitating the way the human visual system is developed by allowing meta-optimizers to firstly acquire valuable cross-task transferable knowledge that could later enable further specialization depending on the task at hand.

Furthermore the generalization capabilities of a meta-learned optimizer could also be facilitated by progress in transfer learning. A lately developed method dubbed "Model-agnostic Meta-learning" (Finn, Abbeel, and Levine, 2017) has succeeded at developing a strategy that focuses on learning neural network parameter initializations that are model-agnostic and, thus, can facilitate learning any task by allowing flexibility that does not depend on the properties of each task. However, no transfer is available once the model is learned, therefore pressing the need for future approaches to meta-learning that could possibly pave the way for lifelong learning. Maybe relating the meta-objective to the steps needed for a model to perform well in a variety of tasks (possibly in the form of distance traveled within the loss space) could be a key element of flexible meta-learned models in the future.

Another possible future extension to our study that not only could drive the field of meta-learning forward but can also potentially benefit the whole field of machine learning would be associated with steps in the so-called field of machine learning explainability. Although meta-learning techniques that focus on having a neural network acting as the optimizer (e.g. our case) certainly do have their place in the research world, their learning outcomes are

often *black-box* optimizers the drawbacks of which are twofold: firstly, despite outperforming standard optimizers in single tasks they ultimately fail to provide us with information relevant to how their structure differs from their hand-written counterparts. Consequently, any intuition they could provide us towards hand-crafting better optimizers is absent. Being able to explain what exactly do neural networks learn could make the aforementioned elements of the learned neural network optimizers accessible, therefore their is hope that relevant research in the field of neural network model explainability will eventually shed light on these shortcomings.

Lastly, more systematic studies must be done towards scaling meta-learning beyond toy problems or few-shot learning tasks. Optimization in modern deep learning settings can involve backpropagating through thousands of gradient steps a thing that leads to impractical training of meta-learned optimizers in terms of time and computational costs and could introduce instability. Therefore, shortening the training process should also be the aim of future work.

Bibliography

Andrychowicz, Marcin et al. (2016). *Learning to learn by gradient descent by gradient descent*. arXiv: 1606.04474 [cs.NE].

Baker, Bowen et al. (2016). *Designing Neural Network Architectures using Reinforcement Learning*. arXiv: 1611.02167 [cs.LG].

Bengio, Y, Samy Bengio, and Jocelyn Cloutier (2002). "Learning a Synaptic Learning Rule". In: DOI: 10.1109/IJCNN.1991.155621.

C. Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12, pp. 2121–2159.

Chen, Yutian et al. (2017). "Learning to Learn without Gradient Descent by Gradient Descent". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 748–756. URL: http://proceedings.mlr.press/v70/chen17e.html.

Cohen, Gregory et al. (2017). *EMNIST: an extension of MNIST to handwritten letters*. arXiv: 1702.05373 [cs.CV].

Duan, Yan et al. (2016). *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning*. arXiv: 1611.02779 [cs.AI].

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. arXiv: 1703.03400 [cs.LG].

Giraud-Carrier, Christophe and Foster Provost (2005). "Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper?" In: *Proceedings of the ICML-2005 Workshop on Meta-learning*.

Hariharan, Bharath and Ross Girshick (2017). "Low-Shot Visual Recognition by Shrinking and Hallucinating Features". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/iccv.2017.328. URL: http://dx.doi.org/10.1109/ICCV.2017.328.

Hochreiter, Sepp and Jürgen Schmidhuber (1997a). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hochreiter, Sepp and Jürgen Schmidhuber (1997b). "Long Short-term Memory". In: *Neural computation* 9, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

Hochreiter, Sepp, A. Steven Younger, and Peter R. Conwell (2001). "Learning To Learn Using Gradient Descent". In: *IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001*. Springer, pp. 87–94.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Koch, Gregory R. (2015). "Siamese Neural Networks for One-Shot Image Recognition". In:

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton (2014). "The CIFAR-10 dataset". In: *online: http://www. cs. toronto. edu/kriz/cifar. html*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Lake, Brenden M. et al. (2016). "Building Machines That Learn and Think Like People". In: *CoRR* abs/1604.00289. arXiv: 1604.00289. URL: http://arxiv.org/abs/1604.00289.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/.

Li, Ke and Jitendra Malik (2016). *Learning to Optimize*. arXiv: 1606.01885 [cs.LG].

— (2017). *Learning to Optimize Neural Nets*. arXiv: 1703.00441 [cs.LG].

Maclaurin, Dougal, David Duvenaud, and Ryan Adams (2015). "Gradient-based Hyperparameter Optimization through Reversible Learning". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 2113–2122. URL: http://proceedings.mlr.press/v37/maclaurin15.html.

Maurer, Daphne, Catherine J Mondloch, and Terri L Lewis (2007). "Sleeper effects". In: *Developmental Science* 10.1, pp. 40–47.

Mishra, Nikhil et al. (2017). *A Simple Neural Attentive Meta-Learner*. arXiv: 1707.03141 [cs.AI].

N. Dauphin, Yann et al. (2015). "RMSProp and equilibrated adaptive learning rates for non-convex optimization". In: *arXiv* 35.

Negrinho, Renato and Geoffrey J. Gordon (2017). "DeepArchitect: Automatically Designing and Training Deep Architectures". In: *CoRR* abs/1704.08792.

Nesterov, Yurii (1983). "A method of solving a convex programming problem with convergence rate O(1/sqr(k))". In: *Soviet Mathematics Doklady* 27, pp. 372–376. URL: http://www.core.ucl.ac.be/\~{}nesterov/Research/Papers/DAN83.pdf.

Netzer, Yuval et al. (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In:

Ravi, Sachin and Hugo Larochelle (2016). "Optimization as a Model for Few-shot Learning". In: URL: http://openreview.net/pdf?id=rJY0-Kcll.

Riedmiller, M. and H. Braun (1993). "A direct adaptive method for faster backpropagation learning: the RPROP algorithm". In: *IEEE International Conference on Neural Networks*, 586–591 vol.1. DOI: 10.1109/ICNN.1993.298623.

Santoro, Adam et al. (2016). *One-shot Learning with Memory-Augmented Neural Networks*. arXiv: 1605.06065 [cs.LG].

Schmidhuber, Jurgen (1987). *Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook*. Diploma Thesis.

Schmidhuber, Jürgen (1992). "Learning to Control Fast-weight Memories: An Alternative to Dynamic Recurrent Networks". In: *Neural Comput.* 4.1, pp. 131–139. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.1.131. URL: http://dx.doi.org/10.1162/neco.1992.4.1.131.

Schmidhuber, Jürgen (1993). "A Neural Network That Embeds Its Own Meta-Levels". In: *In Proc. of the International Conference on Neural Networks '93*. IEEE.

Silver, David et al. (2016). "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587, pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961.

Smith, Linda B. and Lauren K. Slone (2017). "A Developmental Approach to Machine Learning?" In: *Frontiers in Psychology* 8, p. 2124. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2017.02124. URL: https://www.frontiersin.org/article/10.3389/fpsyg.2017.02124.

Tseng, Paul (1998). "An Incremental Gradient(-Projection) Method with Momentum Term and Adaptive Stepsize Rule". In: *SIAM J. on Optimization* 8.2, pp. 506–531. ISSN: 1052-6234. DOI: 10.1137/S1052623495294797. URL: http://dx.doi.org/10.1137/S1052623495294797.

Turing, A. M. (1995). "Computers &Amp; Thought". In: ed. by Edward A. Feigenbaum and Julian Feldman. Cambridge, MA, USA: MIT Press. Chap. Computing Machinery and Intelligence, pp. 11–35. ISBN: 0-262-56092-5. URL: http://dl.acm.org/citation.cfm?id=216408.216410.

Vanschoren, Joaquin (2018). *Meta-Learning: A Survey*. arXiv: 1810.03548 [cs.LG].

Vinyals, Oriol et al. (2016). *Matching Networks for One Shot Learning*. arXiv: 1606.04080 [cs.LG].

Wang, Jane X et al. (2016). *Learning to reinforcement learn*. arXiv: 1611.05763 [cs.LG].

Wichrowska, Olga et al. (2017). *Learned Optimizers that Scale and Generalize*. arXiv: 1703.04813 [cs.LG].

Wolpert, David and William Macready (1997). "Macready, W.G.: No Free Lunch Theorems for Optimization. IEEE Transactions on Evolutionary Computation 1(1), 67-82". In: *Evolutionary Computation, IEEE Transactions on* 1, pp. 67 –82. DOI: 10.1109/4235.585893.

Wu, Yonghui et al. (2016). *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv: 1609.08144 [cs.CL].

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv: 1708.07747 [cs.LG].

Zoph, Barret and Quoc V. Le (2016). *Neural Architecture Search with Reinforcement Learning*. arXiv: 1611.01578 [cs.LG].