

Learning Learning Algorithms

Samuel Oluwadara Esho

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Luís Filipe Barbosa de Almeida Alexandre

junho de 2020

Dedication

I like to dedicate this research work to the Almighty God, who gave me the health, courage and strength to bounce back from the most difficult and depressing moment of my life, making me a stronger person able to press on, not only with this thesis work, but with life in general.

Acknowledgements

Developing this thesis was only possible thanks to the emotional and academic support from a few special individuals who were always ready to lend a helping hand regardless of my faults, imperfections or sometimes lack of motivation.

I wish to express my sincere appreciation to my supervisor professor Luis Alexandre who always reminded me of the importance of completing the thesis paper in person or via emails, even at times when I had no more intention or motivation whatsoever of going through with the thesis work, as well as his welcoming disposition and maturity or perhaps professionalism any time I came back or resumed the thesis work, and was also always apt to render academic recommendations and guidance whenever I needed it.

I also like to show my gratitude to my friends Quoc Trong Nyugen, and Jose Danilson Ferreira, and compatriot Musa Samaila, for their timely assistance, advice and support.

And finally, this list won't be complete without paying my special regards to my parents Joshua O. Esho and Elizabeth A. Esho and siblings Esther O. Esho, Deborah O. Esho, Ruth V. Esho and Mary P. Esho for their incessant emotional support and prayers in my journey to completing this research work.

Resumo

Os modelos de aprendizagem automática dependem dos dados para aprender qualquer tarefa e, dependendo da diversidade de cada um dos elementos da tarefa e dos objetivos do projeto, a quantidade de dados pode ser elevada, o que, por sua vez, pode aumentar exponencialmente o tempo de aprendizagem e o custo computacional. Embora a maioria do treino dos modelos de aprendizagem automática hoje seja feito usando GPUs (unidade de processamento gráfico), ainda é necessária uma quantidade enorme de tempo de treino para obter o desempenho desejado.

Este trabalho tem como objetivo analisar os algoritmos de aprendizagem de aprendizagem ou popularmente conhecidos como metalearning, que são métodos que não apenas tentam melhorar a velocidade de aprendizagem, mas também o desempenho do modelo e, além disso, requerem menos dados e envolvem várias tarefas. O conceito envolve o treino de um modelo que aprende constantemente a aprender tarefas novas em ritmo acelerado, a partir de tarefas aprendidas anteriormente.

Para a revisão do trabalho relacionado, será dada atenção aos métodos baseados em otimização e, mais precisamente, ao MAML (Model Agnostic MetaLearning), porque em primeiro lugar é um dos métodos de metalearning mais populares e em segundo lugar, esta tese foca a criação de um método baseado em MAML, chamado MAML-DBL, que usa uma técnica de taxa de aprendizagem adaptável com limites dinâmicos que permite obter convergência rápida no início do processo de treino e boa generalização no fim.

A proposta variante de MAML tem como objetivo tentar evitar o desaparecimento das taxas de aprendizagem durante o treino e a desaceleração no fim onde entradas densas são predominantes, embora possa ser necessário um ajuste adicional dos hiperparâmetros para alguns modelos ou onde entradas esparsas podem ser predominantes, para melhorar o desempenho.

O MAML-DBL e o MAML foram testados nos conjuntos de dados mais comumente usados para modelos de metalearning, e com base nos resultados das experiências, o método proposto mostrou um desempenho bastante competitivo em alguns dos modelos e até superou o baseline em alguns dos testes realizados.

Os resultados obtidos com o MAML e MAML-DBL (num dos conjuntos de dados) mostram que os métodos de metalearning são soluções altamente recomendáveis sempre que um bom desempenho, menos dados e um modelo versátil ou com várias tarefas são necessários ou desejados.

Palavras-chave

Metalearning;MAML;MAML-DBL

Abstract

Machine learning models rely on data to learn any given task and depending on the universal diversity of each of the elements of the task and the design objectives, multiple data may be required for better performance, which in turn could exponentially increase learning time and computational cost. Although most of the training of machine learning models today are done using GPUs (Graphics Processing Unit) to speed up the training process, most however, depending on the dataset, still require a huge amount of training time to attain good performance.

This study aims to look into learning learning algorithms or popularly known as metalearning which is a method that not only tries to improve the learning speed but also the model performance and in addition it requires fewer data and entails multiple tasks. The concept involves training a model that constantly learns to learn novel tasks at a fast rate from previously learned tasks.

For the review of the related work, attention will be given to optimization-based methods and most precisely MAML (Model Agnostic MetaLearning), because first of all, it is one of the most popular state-of-the-art metalearning method, and second of all, this thesis focuses on creating a MAML based method called MAML-DBL that uses an adaptive learning rate technique with dynamic bounds that enables it to attain quick convergence at the beginning of the training process and good generalization towards the end.

The proposed MAML variant aims to try to prevent vanishing learning rates during training and slowing down at the end where dense features are prevalent, although further hyperparameter tuning might be necessary for some models or where sparse features may be prevalent, for improved performance.

MAML-DBL and MAML, were tested on the most commonly used datasets for metalearning models, and based on the results of the experiments, the proposed method showed a rather competitive performance on some of the models and even outperformed the baseline in some of the carried out tests.

The results obtained with both MAML-DBL (in one of the dataset) and MAML, show that metalearning methods are highly recommendable solutions whenever good performance, less data and a multi-task or versatile model are required or desired.

Keywords

Metalearning, MAML, MAML-DBL

Contents

1	Introduction	1
1.1	Objectives	1
1.2	Motivation: Stating the problem and solution	1
1.2.1	Justifying my Choice of Approach	2
1.3	Research Aims and Objectives	2
1.4	Research Methodology	3
1.5	Contribution to Knowledge	4
1.6	Thesis Structure	4
2	Overview of Learning to Learn	5
2.1	Objectives	5
2.2	Classical Optimization	5
2.3	Machine Learning	6
2.4	Metalearning	7
2.5	Optimization versus Learning versus Metalearning	8
2.6	The Machine Learning Task	8
2.6.1	Vanilla classification task vs. Metalearning classification task	9
2.7	Datasets	9
2.7.1	Omniglot	9
2.7.2	MiniImagenet	10
2.8	Summary	10
3	Related Work	11
3.1	Objectives	11
3.2	Types	11
3.3	Metalearning approaches	11
3.4	Optimization-Based	11
3.4.1	Model Agnostic Metalearning (MAML)	11
3.4.2	LSTM Optimizer: Learning to Learn by Gradient Descent by Gradient Descent	13
3.4.3	Meta-SGD	14
3.4.4	Others	15
3.4.5	Best Approach in This Category	16
3.5	Metric-Based Approaches	16
3.5.1	Siamese Neural Networks	17
3.5.2	Prototypical Neural Networks	17
3.5.3	Relation Nets	18
3.5.4	Others	19
3.5.5	Best Approach in This Category	19
3.6	Memory Augmented Methods	20
3.6.1	MANN (Memory Augmented Neural Network)	20
3.6.2	Others	20
3.7	Other Applications and Areas of Metalearning	21
3.7.1	Bayesian Inference	21
3.7.2	Reinforcement Learning	21

3.7.3	Demonstration and Imitation Learning in Robotics	22
3.7.4	Behavioral Analysis	23
3.8	Summary	24
4	MAML and MAML-DBL	25
4.1	Objective	25
4.2	MAML’s Algorithm for Classification	25
4.3	MAML as a Learned Optimizer	27
4.3.1	Batch Normalization	28
4.3.2	Other Useful Techniques	29
4.4	MAML’s Structure for Classification	30
4.4.1	Implementation of the MAML Algorithm in Tensorflow	30
4.5	Proposed Approach	31
4.6	Summary	32
5	Experimental Evaluation	33
5.1	Objective	33
5.2	Software requirements	33
5.3	MAML’s Experiment Details	33
5.4	Experiments	34
5.4.1	Experiment 1: 5-way 1-shot Omniglot dataset	34
5.4.2	Experiment 2: 20-way 1-shot Omniglot dataset	35
5.4.3	Experiment 3: 5-way 1-shot MiniImagenet dataset	37
5.4.4	Experiment 4: 5-way 5-shot MiniImagenet dataset	39
5.4.5	Assessment of the Proposed Method	40
5.5	Summary	41
6	Conclusion	42
6.1	Thesis Overview	42
6.2	Future Work	43
6.3	Final Thoughts	43
	Bibliografia	44
A	Appendix	48
A.1	MAML data preprocessing	48
A.2	Running the MAML code	48
A.2.1	Errors and Fixes	49

List of Figures

1	A single 2-class classification task with four examples per class or 4-shot, from [Rava]	8
2	A single 5-class classification task with 1 example per class or 1-shot, from [Ravb]	8
3	A task batch B_t of batch-size S_b equal to more than two, containing more than two 5-class classification tasks with 1 example per class or 1-shot, from [Ravb]	9
4	Example of the images in the Omniglot dataset from [LSGT11]	10
5	Samples from 6 classes in the miniImagenet dataset from [Dev].	10
6	MAML pseudocode for few-shot supervised learning as presented in [FAL17]	26
7	Pseudocode of Adam's algorithm and update rule as illustrated in [KB14]	28
8	Code defining the CNN routine extracted from <code>utils.py</code> in [Fin]	30
9	Code defining the hidden layers of the CNN extracted from <code>maml.py</code> in [Fin]	30
10	Lines 34-41 of <code>maml.py</code> in [Fin], denoting the convolutional and non-convolution pathways.	31
11	Pseudocode of Adabound's algorithm and update rule as illustrated in [LXLS19]	32
12	First error, saying that module <code>tensorflow.contrib.layers</code> could not be found since it has been replaced by <code>tf.keras.layers</code> in the new tensorflow version.	49
13	Second error.	50
14	I installed python 3.6.0 in order to install tensorflow 1.14	50
15	I installed tensorflow 1.14 in order to see if it will solve the syntax incompatibilities	50
16	Error saying that procedure <code>google.protobuf.pyext._message</code> could not be found.	50
17	Checked the protobuf version installed.	50
18	Upgraded it to protobuf version 3.6.0.	51
19	Running the code for omniglot 5-way 1-shot experiment.	51
20	Begining training.	51

List of Tables

1	MAML performance on Omniglot as illustrated in [FAL17]	13
2	MAML performance on MiniImagenet as illustrated in [FAL17]	13
3	Meta-SGD performance on Omniglot as illustrated in [LZCL17]	15
4	Meta-SGD performance on MiniImagenet as illustrated in [LZCL17]	15
5	FOMAML and Reptile performance on Omniglot as illustrated in [NAS18]	15
6	FOMAML and Reptile performance on MiniImagenet as illustrated in [NAS18]	16
7	LSTM Meta-Learner performance on MiniImagenet as illustrated in [RL17]	16
8	Best performance on MiniImagenet and Omniglot in this category.	16
9	Siamese Network performance on Omniglot as illustrated in [Koc15]	17
10	Prototypical neural network performance on Omniglot as illustrated in [SSZ17]	18
11	Prototypical neural network performance on MiniImagenet as illustrated in [SSZ17]	18
12	Relation network performance on Omniglot as illustrated in [SYZ ⁺ 17]	18
13	Relation network performance on MiniImagenet as illustrated in [SYZ ⁺ 17]	19
14	Matching network performance on Omniglot as illustrated in [VBL ⁺ 16]	19
15	Matching network performance on MiniImagenet as illustrated in [VBL ⁺ 16]	19
16	Best performance on both MiniImagenet and Omniglot datasets in this category.	19
17	MANN performance on Omniglot as illustrated in [SBB ⁺ 16]	20
18	Meta-Networks performance on Omniglot as illustrated in [MY17]	20
19	Meta-Networks performance on MiniImagenet as illustrated in [MY17]	21
20	Training results in terms of average accuracy and approximated training time.	34
21	Training results with reference to average accuracy and approximated training time.	36
22	Training results with reference to average accuracy and approximated training time.	38
23	Training results in terms of average accuracy and approximated training time.	39

Chapter 1

Introduction

1.1 Objectives

Modern advancements in machine learning and its current application in several fields and tools we use in our daily endeavours, such as the internet, automobiles, and mobile apps, has caused a growing interest in AI and machine learning based development amongst software companies and engineers as well as amateur programmers and developers worldwide. However, as machine learning grows and is being applied to several new fields and devices, so do new challenges, such as computational cost to train a model on massive amount of data, data scarcity for new fields, inflexible or non-versatile learning methods, among others. Several techniques that attempt to solve such challenges are constantly being designed, and among the proposed techniques, one promising direction that has shown a certain degree of success and received growing attention in recent times is learning to learn or metalearning. In this thesis, we will begin by looking at machine learning, then talk briefly in general terms about metalearning, and as the title specifies, we will be focusing strictly on the learning algorithms or the learning of some neural network-based metalearning techniques. Be advised that only minimum attention will be given to the learning model and architecture as it is not the focus and therefore out of the scope of this work.

1.2 Motivation: Stating the problem and solution

To achieve high accuracy most machine learning models usually require lots of data and time, which amounts to high computational cost. Today, most software engineers and IT companies rely on at least a GPU but can sometimes use a ridiculously large number of GPUs (2,048)[JSH⁺18] for high accuracy as well as to cut down time and cost while still retaining the same amount of data.

However, in situations where only limited resources are available (e.g. a single GPU) and high accuracy as well as speed is desired or required, training a model on a huge dataset from scratch could be painfully lengthy. For this problem, I was able to find two interesting possible solutions, which are, transfer learning and metalearning. Transfer learning relies on pretraining, that is, to use transfer learning for a new task (eg. recognizing trucks), known as the target task T_t on a domain D_t there must be a model already trained to perform a related task (eg. recognizing cars), known as the source task T_s on a related domain D_s in which the loss output layer of the existing model of T_s is replaced with a new loss output layer related to T_t to make the transferred model achieve quick convergence for T_t with fewer datapoints, known as few-shot learning. However in cases where there is no model pretrained on a related task T_s , the model will have to be trained from the scratch on the entire dataset however large it is, which brings us back to the original problem.[Wik20g]

Metalearning on the other hand, claims to provide a more independent, flexible, robust and versatile solution where a model can learn not just one task but multiple tasks (task flexible) T_i with fewer data from the scratch without the need to rely on a model previously pretrained on a related task,

thus leading to less data, less training time, and less cost. In addition, metalearning could also be very useful in situations where only limited data is available.

There are three main metalearning types, namely optimization-based, metric-based and memory augmented types and several approaches. MAML, an optimization-based approach, was chosen after an extended research was made to select a metalearning approach with which to experiment to prove or disprove the above mentioned claims, as well as see if it is a viable solution to the limited hardware resources (since the authors only used one GPU for their experiments) problem. This thesis also proposes a method that tries to improve the performance of MAML, called MAML-DBL, trained also with a single GPU.

1.2.1 Justifying my Choice of Approach

My choice of optimization-based meta learning methods instead of other methods such as metric-based or memory augmented is related to:

- **Success:** First of all, they are one of the most successful metalearning types in machine learning not only in terms of accuracy or convergence speed, but also when applied to other fields such as robotics [ABB⁺17], combination with probabilistic methods [GFL⁺18], imitation [DAS⁺17], among others.
- **Algorithm:** The simple and straightforward algorithms of most optimization-based methods show an elegant way of defining metalearning with commonly used or already known neural network techniques, unlike other methods that involve the use of sometimes a series of networks or not so clear and rather unpopular structures, concepts and architectures. e.g. relation [SYZ⁺17], attention block [MRCA18], fast weights, slow weights [MY17], etc.
- **Robustness and flexibility:** The robustness and maleability of most optimization-based methods, allows them to be easily applied to different settings of machine learning such as classification, regression, reinforcement learning and be easily modified, experimented and implemented in real world systems and devices.

My choice of MAML among the optimization-based techniques boils down to its gradient based model agnostic nature, that is, apart from being one of the most effective techniques in the category, it is actually the standard or model for most of the other optimization techniques, for instance, Meta-SGD, which, from the results in Table 8, is the most successful optimization-based technique, at the time of this thesis, and it is just a slight modification of MAML. This means that understanding or analyzing MAML is also a way to understand almost all the other methods (excluding the RNN based ones) in the category.

MAML-DBL was proposed to try to improve MAML by trying a more recent learning rate approach with dynamic bounds that starts off as an adaptive method but gradually transforms to a momentum-based SGD method with the intent of trying to reduce any chance or risk of vanishing learning rates and bringing about generalization as training progresses as opposed to the fixed adaptive method used in the vanilla MAML.

1.3 Research Aims and Objectives

The aims and goals for this thesis could be divided into:

- First, introduce the subject, analyze metalearning types and methods via extensive literature review of learning algorithms, in order to investigate or get to know the existing state of the art methods.
- Focus on optimization-based methods and carry out a theoretical analysis of their learning model, learning algorithm setup or how they work.
- Propose a metalearning method based on the optimization approach called MAML-DBL.
- Finally, evaluate MAML as well as MAML-DBL using a single GPU on the two most commonly used metalearning datasets in terms of accuracy and speed, and draw conclusions as to their effectiveness or veracity of their claims in solving or addressing the problem.

1.4 Research Methodology

The research methodology could be split into two parts;

The theoretical analysis, some sort of survey or literature review of state of the art learning to learn methods, which as earlier said will be based on reviewing existing available literature such as articles, research papers and publications of state of the art approaches of metalearning. And the experimental part, which was done using the Omniglot and MiniImagenet datasets.

For the theoretical analysis, in order to ensure the credibility as well as the effectiveness of each approach in the fast-evolving machine learning and technological world, each of the available literature, especially the research papers considered, had to meet certain criteria as detailed below:

- Reliable source such as the cornell university archive (arxiv.org), paperswithcode.com, among others to ensure some level of credibility.
- At least 20 research papers were studied to ensure an extensive knowledge and understanding of the subject.
- The papers were chosen based on their relevance to learning algorithms, metalearning and learning to learn.
- The date range on average was set to 2016 to 2020, to ensure the approach still has some effectiveness today and has not been long outperformed by other techniques.
- The choice of approach for the experimental analysis required the technique used as baseline is still currently effective, as MAML not only is still efficient today but is being applied and used in many other ways and machine learning settings.

For the experimental analysis,

- The experiments had to be similar to the ones mentioned in the research paper to confirm their claims.
- The programming language and technologies had to be the same as well to ensure compatibility.
- MAML as well as MAML-DBL were evaluated on accuracy and training duration on a GPU to ensure relevance to the problem being addressed.
- To measure the training time, the default linux execution timing tool, 'time', was used.

1.5 Contribution to Knowledge

1. This thesis attempts to introduce or propose methods that could be helpful in cases where data for a particular task or set of tasks is limited and where no prior knowledge or pretrained model is available.
2. This thesis could also serve as a way of providing basic background knowledge or as a guide for a novice or anyone interested in working with metalearning techniques for whatever reason in a future work.
3. This thesis proposes MAML-DBL (MAML with Dynamic Bound Learning rate).

1.6 Thesis Structure

This thesis is made up of six chapters, this first chapter which is the introduction, contained the general objectives and focus of this thesis, the problem statement, the aims and objectives, the motivation and justification of chosen method and the contribution to knowledge. The following chapters are as described below:

- **Chapter 2;** Introduces and defines the concepts used in addressing the problem. Defines and introduces optimization, machine learning, metalearning and learning learning algorithms, as well as the metalearning task and the commonly used metalearning datasets.
- **Chapter 3;** Presents the state of the art in metalearning. It analyzes several contemporary state-of-the-art metalearning methods and their performances as well as some of their applications.
- **Chapter 4;** Evaluates the MAML and MAML-DBL, by defining some useful terminologies, describing the MAML pseudocode, supplying some guidelines to access and download the source code, data preprocessing, looking into some aspects of the code, then running it.
- **Chapter 5;** Describes the types of experiments carried out, the results, and the facts observed during the experiments.
- **Chapter 6;** Conclusion and final thoughts.

Chapter 2

Overview of Learning to Learn

2.1 Objectives

This chapter aims to define some concepts that are important to understanding learning to learn also known as metalearning.

2.2 Classical Optimization

Before delving into machine learning, it will be wise to first talk a bit about optimization, as machine learning especially neural networks involve optimization methods [GBC16].

Optimization from a general perspective, is the search for the best or optimum solution to a given problem. From a more technical perspective, it is the use of the best or most appropriate algorithm to find the maximum (maximize) or minimum (minimize) possible outcome or value for an optimization model. [BP14]

An optimization problem is any real world problem that requires a very specific and exclusive solution. Which means that the solution to an optimization problem is tailored to solve only that specified problem to the minutest detail, such that it is useless for any other problem, and has to be updated and re-optimized if the specified problem or any of its detail changes. [BP14]

In general, optimization, for the sake of clarity, is usually broken down into two main stages, the model and the algorithm.

An **optimization model**, consist of at least an objective function, decision variable(s), and, depending on the nature of the problem, some constraints.

- The objective function $f(x)$ of an optimization problem is an algebraic description of your goal with respect to your decision variable(s) x . And is usually denoted either as a minimization or maximization as:

$$\min_{x \in R^n} f(x)$$

- The decision variables x are the inputs to your objective function or the decisions you wish to make.
- The constraint $c_i(x)$ are the restrictions or the area of focus of the model. Usually preceded by "subject to" and denoted as either an equality or inequality as:

$$\begin{aligned} & \text{minimize} && f(x) \\ & x \in R^n \\ & \text{subject to} && c_i(x) = 0, \\ & && c_i(x) \geq 0 \end{aligned} \tag{2.1}$$

An **optimization algorithm** is a set of steps designed to help the model converge to the optimum solution. The set of steps include finding an update direction, defining a step size or step length

indicating how far to go in the update direction, and a convergence check, which based on the chosen algorithm may sometimes require you to evaluate the hessian.

Optimization algorithms are usually either gradient based, which could also require hessian evaluation, or gradient free.[Wik20b]

2.3 Machine Learning

Machine learning, from a general perspective, could be defined as the search for the best parameters or solution to a given problem, with the help of a machine or rather a computer, that can or has been formulated as some data distribution over a specified task. From a more technical perspective, it is the use or formulation of the most suitable or appropriate machine learning algorithm for a machine learning model in order to achieve quick convergence and high accuracy. [GBC16]

Machine learning has several learning settings or paradigms such as supervised learning, unsupervised learning, reinforcement learning, etc. In this work however, focus will be placed on the supervised learning setting and under supervised learning we will be concentrating on classification problems.

Supervised learning involves mapping an input such as an image, or an observation from a gathered data to a labelled output [Wik20f], which could either be a classification or a regression problem.

Classification problem relates to categorical or discrete data where an input is assigned or mapped to a specific category or class from a finite set of labels[Wik20e].

Regression problem deals with continuous data where inputs are mapped or used to estimate or predict a specified output or dependent variable. Usually used for prediction and forecasting[Wik20d]. In general, machine learning, for the sake of clarity, can also be broken down into two main parts, the model and the algorithm.

The **machine learning model**, consist of at least a cost or objective function, parameter(s), and, depending on the nature of the problem, some constraints.

- The loss function measures the error of each example or observation (datapoint) of the training set. The cost function or average total/generalized loss function measures the error or average of the sum of the loss of all or multiple examples in training set, and could also be defined as the algebraic representation of your goal with respect to the parameters, which in this case is the minimization of the error of the equation fitting or splitting all the datapoints in the training set. And it is usually the expected value or average of the sum of all the losses across the training set of a given learning problem or task, since the definite data distribution is usually not known.

When dealing with neural network based models, neural networks have functions known as activations or nonlinearities and deciding on the ones to choose or use could be a deciding factor or have a strong influence on how you choose or formulate the cost or objective function.

- The parameters or weights are the inputs to your cost function that are constantly optimized by the learning algorithm.
- The constraint are the restrictions or the area of focus of the model.

All these four components constitute the machine learning model.

There are several types of machine learning models but we will be focusing on neural network based models. Neural networks could be viewed from a learning and mathematical perspective as a huge composition of functions or mappings structured to represent some qualitative concept such

as the working of the brain, the seeing of the eye, etc, and built into a cost function in terms of the parameters, which is optimized, learned or trained by a learning algorithm. After building or choosing your learning model, you are not there yet, as the learning model alone is like a car with no engine, so we still need a learning algorithm.

The **learning algorithm** is a set of steps or an iterative program or formulation that helps minimize your learning model. It is similar to an optimization algorithm but refined to befit learning models.

There are several types of learning algorithms which are usually grouped into gradient based and gradient free learning algorithms, and as said earlier our focus in this thesis paper will be gradient based learning algorithms which are often considered the most effective for most cost functions (especially smooth or continuous functions).

Although there are several gradient based learning algorithms that follow different steps, however, there are three fundamental steps most of them must adhere to in their journey to training or optimizing the learning mode and achieving convergence.

The basic steps of a learning algorithms include;

- Search direction, decides on the direction to go based on the gradient,
- Learning rate, tells how far to go in the chosen direction, and
- convergence check, checks if minimum value or cost have been attained.

2.4 Metalearning

Metalearning is a subfield or alternative paradigm of machine learning where a model, sometimes called the metalearning system, learns to improve its future learning performance from previous learning episodes and related tasks.[Wik20a][HAMS20]

Technically, it is like adding another learning model M to an existing learning model B and remodelling both of them in such a way as to make B learn a task at a time from a batch of tasks or a distribution of tasks and make M learn the generalized knowledge of all the tasks learned by B , in order to use this generalized knowledge to optimize or help B learn the next task or batch of tasks.

The **metalearning model** usually consist of an outer learning model M and an inner learning model B . M from a machine learning perspective kind of acts as a learned learning algorithm (eg. initializer,optimizer, etc) to B .

Both B and M have all the features of a learning model as discussed in the previous section (2.3), however, they are structured to accomplish different goals; M aims to learn to be a better optimizer to B from the models produced by B , and B aims to learn how to produce new models as fast as possible by taking advantage of the generalized knowledge supplied by M . In metalearning, the combination of both B and M is regarded as a single machine learning model sometimes called the metalearning system [HS97], since they both work together to achieve mutual or common goals, which are, few-shot learning and versatility.

Usually both models have their individual learning algorithms. However, M must use an objective, a set of parameters and a learning algorithm capable of elevating it from just a simple learning model to a learned learning algorithm or optimizer to B .

2.5 Optimization versus Learning versus Metalearning

- Optimization focuses on a definite or specific data distribution of a given problem or task. The solutions are specific and exclusive to the specified problem.
- Learning focuses on an indefinite or unlimited data distribution (could be made up of several solutions to optimization problems) of a given problem or task usually reduced to a train set and test set.
- Metalearning focuses on a specified distribution of learning problems or tasks.

2.6 The Machine Learning Task

A machine learning task varies based on the machine learning paradigm or setting been used. Therefore, a machine learning task could either be a reinforcement learning task, unsupervised learning task, or from the supervised learning setting; a regression task or a classification task, which is the focus of this thesis.

A **classification task** T_c is any task that involves the categorization, classification or splitting of K number of datapoints from a specified N number of classes or labels. As shown in Figure 1 and Figure 2. A conventional machine learning classification model, as discussed in the previous section, is limited to only one classification task.

In few-shot learning K number of datapoints or examples is known as K-shot and N number of classes is known as N-way. Where $K \in \mathbb{Z}$ and where N is the number of all elements of a finite set and $N \in \mathbb{Z}$.



Figure 1: A single 2-class classification task with four examples per class or 4-shot, from [Rava]



Figure 2: A single 5-class classification task with 1 example per class or 1-shot, from [Ravb]

2.6.1 Vanilla classification task vs. Metalearning classification task

We have just seen what a normal machine learning classification task looks like, and we also know that conventional machine learning models including deep learning classification model (although their parameters can be reused for quick convergence of a related task via transfer learning. They are still limited to a single task at a time.) are only limited to a single N -class classification task.

Meta-classification models however, are capable of learning multiple tasks, and therefore not limited to a single task, which brings in two other metalearning jargons or variables:

- **Task batch:** A task batch for a meta-classification model B_t is a set or a group of N -class classification tasks with a specified batch-size S_b . Where N and K is the same for all the tasks in B_t .
- **Batch-size:** The batch-size S_b is the size or amount of tasks in the B_t and it is always a whole number $S_b \in \mathbb{Z}$.

Meta-classification models learn with or feed on task batches as a single metalearning instance involves a task batch B_t with a specified S_b containing a series of N -class classification tasks with K number of examples for each class in each task T_c . As shown in Figure 3

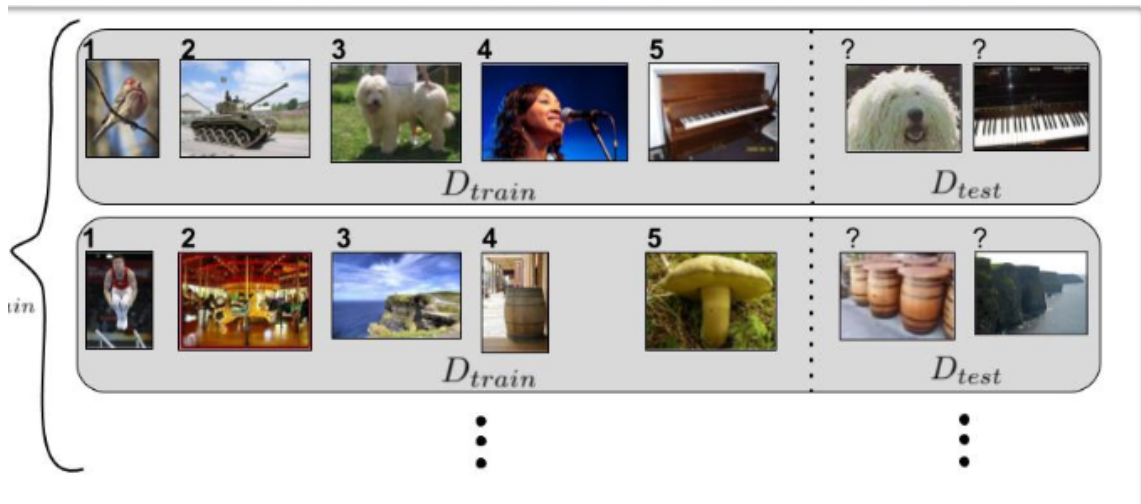


Figure 3: A task batch B_t of batch-size S_b equal to more than two, containing more than two 5-class classification tasks with 1 example per class or 1-shot, from [Ravb]

2.7 Datasets

A dataset refers to the gathered or available set of examples. Which is usually made up of several observations or solutions to the given problem. It is usually split into training set and test set or support set and query set in metalearning. As said earlier, we would not be focusing on the learning model but the algorithms. It would be necessary however, to introduce the most commonly used datasets in metalearning research at the time of this thesis work, which include;

2.7.1 Omniglot

The Omniglot dataset[br], introduced in Lake et al[LSGT11], to aid few-shot evaluation and learned learning algorithm research. It is a dataset of writing systems, made up of over 1600

distinct characters from 50 different alphabets, each of which was handwritten by 20 different people. Omniglot images are grayscale. A sample of some of the characters in the omniglot dataset is shown in Figure 4.

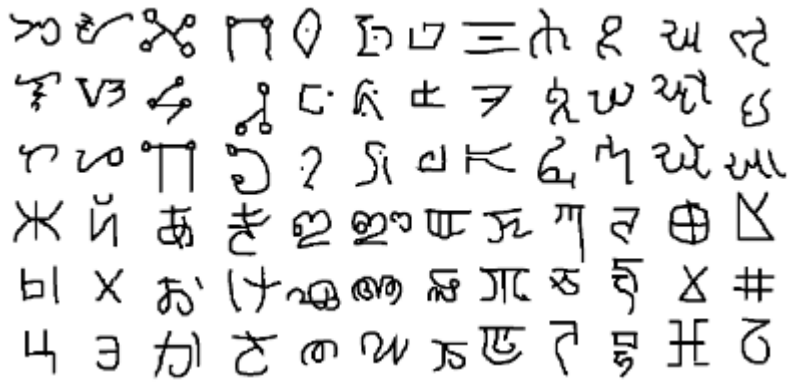


Figure 4: Example of the images in the Omniglot dataset from [LSGT11]

2.7.2 MiniImagenet

MiniImagenet dataset[y1], introduced in Vinyals et al[VBL+16], to aid few-shot evaluation and learned learning algorithm research. It is a mini-sized version of the ImageNet dataset, and consist of 60,000 colour images of size 84 x 84 shared into 100 classes of 600 examples. Authors split it into 80 classes for training and 20 classes for testing their approach. A sample of a few classes in the miniImagenet dataset is shown in Figure 5



Figure 5: Samples from 6 classes in the miniImagenet dataset from [Dev].

2.8 Summary

This chapter talked about optimization and how it relates to machine learning and metalearning. It also defined the metalearning task and introduced the most commonly used metalearning datasets.

Chapter 3

Related Work

3.1 Objectives

In this chapter, we will be looking at current state of the art metalearning approaches and their learned learning algorithm techniques and methods. We will talk about part of the metalearning model which includes; the loss functions and cost functions in terms of parameters and focus more on the learning algorithms, used to optimize the loss function for a single example or the cost function for multiple examples, and the learned learning algorithm or optimizer.

3.2 Types

Although, in the thesis paper our focus will be on current state of the art optimization based metalearning types. We will briefly talk about other common so called metalearning types, some of which are considered metalearning techniques due to the fact that they accomplish few-shot learning and not because they have a metalearning structure, which include:

- Optimization-Based
- Metric-Based
- Memory Augmented methods

3.3 Metalearning Approaches

3.4 Optimization-Based

Called optimization based simply because they use optimization or learning methods and ideas both in the construction and training of the metalearning model.

3.4.1 Model Agnostic Metalearning (MAML)

- The basic idea of MAML by Finn et al [FAL17] is to find a better initial parameters so that, the model can quickly learn new tasks with fewer gradient steps.
- MAML is model agnostic, meaning that we can apply MAML to any model that is trainable with gradient descent.

This metalearning model uses the cross-entropy cost function for both the inner and outer learning models in terms of the parameters for discrete classification between the predicted and true class. Cross entropy cost function measures the performance of a classifier or classification model whose output is a probability value between 0 and 1 such as the outputs of a sigmoid or a softmax activation function. And as the predicted probability diverges from the true label or target output the cross entropy loss increases and vice-versa.[Re17]

$$\mathcal{L}_{T_i}(f_\phi) = \sum_{x^{(j)}, y^{(j)} \sim T_i} y^{(j)} \log f_\phi(x^{(j)}) + (1 - y^{(j)}) \log(1 - f_\phi(x^{(j)}))$$

Where $y^{(j)}$ and $f_\phi(x^{(j)})$ are the true label and the predicted probability of an input/output pair $x^{(j)}$, $y^{(j)}$ or example sampled from a learning problem or task T_i from a given distribution over tasks.

The cross entropy cost function of the inner learning model in terms of parameters f_θ , of each example or datapoint (since it is GD the average error of all examples in training set are first measured before the GD learning algorithm is applied) of a single task at a time in sampled batch of tasks from the distribution over tasks, is optimized or trained by the gradient descent (GD) learning algorithm, while the cross entropy cost function of the outer model (or metalearning level or learned learning algorithm to be) in terms of parameters $f_{\theta'_i}$ of a single model at a time from the sampled batch of models produced by the inner learning model (or base level) is optimized or trained by the stochastic gradient descent (SGD) learning algorithm. Gradient descent (GD) is a learning algorithm that waits until the inner learning model samples through all datapoints or examples in the training set of a given task from the batch of sampled tasks from a specified distribution over related tasks, so as to minimize the cost or empirical loss or expected value of all the losses of all datapoints in the training set of the said task.

$$\min_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'_i}) = \sum_{T_i \sim p(T)} \mathcal{L}_T(f_\theta - \alpha \nabla_{\theta} L_{T_i}(f_\theta))$$

where:

$\nabla_{\theta} \mathcal{L}_{T_i}(f_\theta)$ - is the gradient of $\mathcal{L}_{T_i}(f_\theta)$, and

α - is the learning rate for inner learning model.

SGD on the other hand is a learning algorithm that optimizes the outer learning model or the cross entropy cost function of the metalearning model in terms of the parameters of the model, to be precise, after each example or mini-batch of examples. The above equation or the supposed cost function of the outer learning model which measures the cost of each model produced by the inner learning model is indeed the objective function of the metalearning model as a whole with respect to the parameters (at this level they refer to each of the produced model parameters $f_{\theta'_i}$), which when minimized using SGD goes through a meta gradient update which involves a gradient through a gradient or rather the hessian vector product computation, as shown in the equation below.

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'_i})$$

After the outer or metalearning model is trained on a batch of models produced by the inner learning model it is able to behave as a learned learning algorithm or optimizer capable of initializing the inner learning model for its next batch of tasks from a specified distribution over tasks. This helps the inner learning model converge quickly and with less gradient steps or iteration. Which is useful in cases where data is scarce or where quick convergence is desired or required. This approach was trained for few-shot image recognition on Omniglot dataset with a learning rate of 0.4 and on MiniImagenet with a learning rate of 0.1. The results are as shown in Tables 1 and 2:

The main challenge encountered in this approach is a significant computational expense coming

Table 1: MAML performance on Omniglot as illustrated in [FAL17]

Omniglot [LSGT11]	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
MAML, no conv.	$89.7 \pm 1.1\%$	$97.5 \pm 0.6\%$	-	-
MAML	$98.7 \pm 0.4\%$	$99.9 \pm 0.1\%$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$

Table 2: MAML performance on MiniImagenet as illustrated in [FAL17]

MiniImagenet [VBL ⁺ 16]	5-way 1-shot Accuracy	5-way 5-shot Accuracy
MAML, first order approx.	$48.07 \pm 1.75\%$	$63.15 \pm 0.91\%$
MAML	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$

from the use of second derivatives or hessian product vector when backpropagating the meta-gradient through the gradient operator in the meta-objective. They however the original technique with a version using only the first order gradients and noticed that the performance was almost the same as the original MAML technique, which they believe is due to the fact that ReLU modelled neural networks are locally almost linear as most of the second order update had almost no effect on the metalearning model as they either close to zero or zero. They suggested an where the second order update step will be omitted, a suggestion that was tested by Nichol et al [NAS18] in their Fomaml (first order maml) and Reptile techniques. [FAL17]

3.4.2 LSTM Optimizer: Learning to Learn by Gradient Descent by Gradient Descent

Learning to learn by gradient descent by gradient descent by Andrychowicz et al [ADC⁺16] proposes a solution to traditional learning algorithms failing in the face of data scarcity by learning an RNN (LSTM) structured learning model to optimize or learn a base model. This technique uses gradient descent to optimize the RNN which in turn optimizes the parameters of the base network by gradient descent as well, hence the title "learning to learn by gradient descent by gradient descent".

As said earlier, it is made up of an outer learning model called the optimizer which is RNN(LSTM) based and a inner learning model called the optimizee which is MLP based. Suppose the objective of the optimizee is $f(\theta_t)$, after every input to the optimizee the optimizer, whose job is to optimize the optimizee, tries to do its job by taking the gradient of $f(\theta_t)$ in terms of the parameters (which are arranged through time t) together with the previous hidden state value (of a previous timestep or optimizee data input) and of course its own weights ϕ as input and with this information it acts as a learning algorithm (which is kind of designed or tailored to match the structure of an RNN/LSTM), optimizes the current parameter and feeds it back to the optimizee. So, instead of the optimizee relying on the normal gradient descent learning algorithm, the optimizer becomes a learned or trained learning algorithm to the optimizee.

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

Where:

θ_{t+1} is the updated parameter,

θ_t is the previous parameter,

g_t is the output from the metalearning compartment function that receives two arguments; the gradient of loss with respect to old parameter and its set of parameters.

$\nabla f(\theta_t)$ is the gradient of old parameter.

ϕ - The parameters of the optimizer.

The optimizer is updated using backpropagation through time (BPTT), which is done at test time or rather with the test or meta-train set.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

Where:

c_t - Refers to the updated or optimized cell state.

f_t - Refers to the forget gate

c_{t-1} - Previous cell state.

i_t - Step size.

\tilde{c}_t - Gradient with respect to loss.

Learning to learn by gradient descent by gradient descent was evaluated on MNIST and CIFAR-10 datasets using ADAM for the optimizer and a learning rate chosen by random search. This method outperformed baselines or standard optimizers such as SGD, RMSprop, ADAM, and Nesterov's accelerated gradient (NAG).

3.4.3 Meta-SGD

While vanilla gradient based algorithms methods work well with a huge amount of labeled data, they are unlikely reliable for few-shot learning. Meta-SGD by Li et al [LZCL17] which uses a very similar approach as MAML, whose aim is to find a gradient based learned learning algorithm or optimizer that achieves few-shot learning. Gradient based few-shot learning which is unattainable with vanilla or classical gradient based learning algorithms or methods because they require a huge data of different, if not all, examples of a single task to avoid overfitting to certain types of examples or a specific training set but rather generalize to all of them if possible. Learned learning algorithms or optimizers also known as metalearners not only help avoid overfitting to a particular training set of a task but also generalize to multiple tasks from a specified distribution over tasks (accomplish versatility).

This is the cost function or empirical loss used for both the inner and outer learning model of Meta-SGD;

$$\mathcal{L}_T(\theta) = \frac{1}{|T|} \sum_{(x,y) \in T} l(f\theta(x), y)$$

The learning algorithm used for the inner model is a GD algorithm, in which the learning rate is elementwise multiplied with the gradient.

$$\theta' = \theta - \alpha \circ \nabla \mathcal{L}_T(\theta)$$

Which influences the update or search direction of descent and which is similar to the GD learning algorithm used on the inner or base learning model of MAML.

The objective function of the metalearning model as a whole or cost function of the outer model,

$$\min_{\theta, \alpha} E_{T \sim p(T)}[\mathcal{L}_{test(T)}(\theta')] = E_{T \sim p(T)}[\mathcal{L}_{test(T)}(\theta - \alpha \circ \nabla \mathcal{L}_{train(T)}(\theta))]$$

in terms of the parameters and the learning rate of the GD learning algorithm used to update the

inner learning model (which is slightly different from MAML that only focuses on the parameters). Like MAML, the objective is optimized using SGD learning algorithm. Data preprocessing involves splitting dataset into training set and test set. The training set is used to train the inner model using GD while the test set is used to train the outer model using SGD that also tries to convert it into a learned (trained) learning algorithm or optimizer (learns to initialize the inner learning model for a new task).

Meta-SGD was evaluated on few-shot classification using Omniglot and MiniImagenet datasets and was compared alongside some metric-based techniques or baselines. The results are as follows.

Table 3: Meta-SGD performance on Omniglot as illustrated in [LZCL17]

Omniglot	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
Siamese Nets	97.3%	98.4%	88.2%	97.0%
Matching Nets	98.1%	98.9%	93.8%	98.5%
Meta-SGD	$99.53 \pm 0.26\%$	$99.93 \pm 0.09\%$	$95.93 \pm 0.38\%$	$98.97 \pm 0.19\%$

Table 4: Meta-SGD performance on MiniImagenet as illustrated in [LZCL17]

MiniImagenet	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
Matching Nets	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$	$17.31 \pm 0.22\%$	$22.69 \pm 0.20\%$
Meta-SGD	$50.47 \pm 1.87\%$	$64.03 \pm 0.94\%$	$17.56 \pm 0.64\%$	$28.92 \pm 0.35\%$

3.4.4 Others

Other methods under this category include;

- First-Order MAML (FOMAML) and Reptile (which are first-order variants of MAML) by Nichol et al[NAS18], which was evaluated on few-shot classification on Omniglot and Mini-Imagenet datasets.

Table 5: FOMAML and Reptile performance on Omniglot as illustrated in [NAS18]

Omniglot	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
FoMAML + Transduction	$98.3 \pm 0.5\%$	$99.2 \pm 0.2\%$	$89.4 \pm 0.5\%$	$97.9 \pm 0.1\%$
Reptile	$95.39 \pm 0.09\%$	$98.90 \pm 0.10\%$	$88.14 \pm 0.15\%$	$96.65 \pm 0.33\%$
Reptile + Transduction	$97.68 \pm 0.04\%$	$99.48 \pm 0.06\%$	$89.43 \pm 0.14\%$	$97.12 \pm 0.32\%$

Table 6: FOMAML and Reptile performance on MiniImagenet as illustrated in [NAS18]

MiniImagenet	5-way 1-shot Accuracy	5-way 5-shot Accuracy
FoMAML+Transduction	48.07 \pm 1.75%	63.15 \pm 0.91%
Reptile	47.07 \pm 0.26%	62.74 \pm 0.37%
Reptile+Transduction	49.97 \pm 0.32%	65.99 \pm 0.58%

- The Meta-learning system is a legacy approach by Hochreiter et al from their Learning to Learn Using Gradient Descent papers[HYC01]and[YHC01], where they introduced the idea of using LSTMs[HS97] as learned optimizers.
- LSTM Meta-learner by Sachin Ravi and Hugo Larochelle[RL17] which is based on the Meta-learning system introduced by Hochreiter et al[HYC01] and similar to Learning to learn by gradient descent by gradient descent by Andrychowicz et al[ADC⁺16], was evaluated on few-shot classification on MiniImagenet alongside matching network baseline. The results are as shown in table 7.

Table 7: LSTM Meta-Learner performance on MiniImagenet as illustrated in [RL17]

MiniImagenet	5-way 1-shot Accuracy	5-way 5-shot Accuracy
Matching Nets	43.40 \pm 0.78%	51.09 \pm 0.71%
Matching Nets FCE	43.56 \pm 0.84%	55.31 \pm 0.73%
Meta-Learner LSTM	43.44 \pm 0.77%	60.60 \pm 0.71%

3.4.5 Best Approach in This Category

The best approach on aggregate in this category from the results obtained on omniglot and MiniImagenet based on accuracy, as shown below is Meta-SGD, it only slightly lost to Reptile ,64.03% vs. 65.99%, on the 5-class 5-shot evaluation on MiniImagenet as portrayed in table 8.

Table 8: Best performance on MiniImagenet and Omniglot in this category.

	Best 5-way 1-shot Accuracy	Best 5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
MiniImagenet	Meta-SGD \approx 50.47%	Reptile \approx 65.99%	-	-
Omniglot	Meta-SGD \approx 99.5%	MAML/Meta-SGD \approx 99.9%	Meta-SGD \approx 95.9%	MAML/Meta-SGD \approx 98.9%

3.5 Metric-Based Approaches

Approaches using this technique try to attain versatility and quick learning by learning class embeddings from few examples in support set and then comparing or rather applying a metric or distance function to a new example in query set and learned embeddings to tell if it belongs to that class or not.

Approaches here achieve architectural versatility as well as quick learning as they can be applied to various types of inputs such as images, text, sounds, among others, we might only have to

change the embedding function, and are n-shot based with the ability of learning quickly from few datapoints.

3.5.1 Siamese Neural Networks

A siamese neural network by Gregory Koch[Koc15], is made up of two or more identical neural networks, each having the same architecture and weights. The goal of this approach is to compare a pair or more of inputs (eg. images, text, etc) and tell if they are the same or different. Siamese neural networks are not limited to a specific type of neural network as feature extractors could be a pair of RNNs or CNNs. It is considered a metalearning method probably because it consist of at least two neural networks which work together to achieve a common or mutual goal, which is to correctly discriminate pairs of inputs after a few examples as it also learns using a few-shot regime. Siamese nets work by sampling datapoints from a dataset in form $(X1, X2, Y)$ split into input for the first neural network $X1$, input for the second neural network $X2$ and the true label or target output Y which is either 1 (meaning that they belong to the same class) or 0 (not a match). After sampling an input the embedding functions or identical neural networks will extract the embedding or feature vector from the pair or more of inputs and use an energy function such as the Euclidean distance to measure the metrics or difference between the pair, the output is usually converted to probability using a sigmoid function and a loss function, such as a regularized cross entropy loss function, is applied, which is then updated, if necessary, by a standard backpropagation algorithm, where the gradient is additive across the twin embedding functions due to the tied weights.

This technique was evaluated on the Omniglot and MNIST datasets. Its performance on Omniglot is as shown in Table 9:

Table 9: Siamese Network performance on Omniglot as illustrated in [Koc15]

Siamese Nets on Omniglot	no distortions	affine distortions x8
30k training	90.61%	91.90%
90k training	91.54%	93.15%
150k training	91.63%	93.42%

3.5.2 Prototypical Neural Networks

Prototypical Neural Networks by Snell et al[SSZ17], operate in quite a similar way as the siamese neural network, but with a different architecture. Instead of having twin networks, prototypical nets only have a single embedding function or neural network which could also be a CNN, an RNN, etc. They seem to achieve metalearning as well as few-shot learning by creating and using prototypes. This is done by sampling inputs from a dataset arranged in the usual input-true label format $(x1, y1), (x2, y2), \dots (xn, yn)$ where x is the input or example and y the true label or target output, three or four examples from each class in the dataset is randomly sampled and set aside, this is called the support or train set, then another set of three or four examples are sampled (to be used later) and called the query or test set . Training is done in episodes and in the same way for each class (meaning with the same quantity of examples). The embedding function extracts the features from each class in the support set, then generates a class prototype by computing the mean of the learned features of all the examples in a class. After creating a prototype for each of

the classes in the support set by computing the mean of the features of all the examples in each of the classes. The model is tested with examples from the query set.

In order to predict the class of a new example in query set, the embedding function or neural network extract or generates the embeddings or feature vector of a novel query point, compares the distance between novel example from query set and the embeddings of each class prototype using the Euclidean distance, softmax is applied to the output to the distance found to get the probability, So the class prototype versus novel query point comparison with the highest probability will be the class to which the novel query point belongs. This method was evaluated on Omniglot and MiniImagenet alongside a statistical approach baseline called "neural statistician"[ES16]. The results are presented in Table 10 and Table 11

Table 10: Prototypical neural network performance on Omniglot as illustrated in [SSZ17]

Omniglot	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
Neural Statistician	98.1%	99.5%	93.2%	98.1%
Prototypical Nets	98.8%	99.7%	96.0%	98.9%

Table 11: Prototypical neural network performance on MiniImagenet as illustrated in [SSZ17]

MiniImagenet	5-way 1-shot Accuracy	5-way 5-shot Accuracy
Prototypical Nets	49.42 \pm 0.78%	68.20 \pm 0.66%

3.5.3 Relation Nets

Relation network by Sung et al[SYZ⁺17], is another metric-based technique, which behaves in a very similar way as the two previously discussed approaches in this category. It achieves few-shot learning by measuring the relation between two objects or inputs. It comprises of two connected networks sequentially arranged, one to extract features and the other to tell the relation. This works by first, splitting your dataset into support set and query set both arranged in the usual input, true label manner $(x_1, y_1), (x_2, y_2), \dots(x_n, y_n)$ with few examples from each class in dataset. Use the embedding function or the first neural network to extract the features of an example in the support set, next do the same for an example in the query set, concatenate both feature vectors or embeddings from the support set and query set, then use the relation function, which could be a convolutional neural network classifier that generates the relation score ranging from zero to one, to obtain the relation score, evidently a value between 0 and 1. This technique was also evaluated on Omniglot and MiniImagenet datasets as shown below:

Table 12: Relation network performance on Omniglot as illustrated in [SYZ⁺17]

Omniglot	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
Relation Nets	99.6 \pm 0.2%	99.8 \pm 0.1%	97.6 \pm 0.2%	99.1 \pm 0.1%

Table 13: Relation network performance on MiniImagenet as illustrated in [SYZ⁺17]

MiniImagenet	5-way 1-shot Accuracy	5-way 5-shot Accuracy
Relation Nets	50.44 ± 0.82%	65.32 ± 0.70%

3.5.4 Others

Another technique that fall into this category include;

- Matching networks by Vinyals et al[VBL⁺16], is a metric-based technique that was also evaluated on Omniglot and MiniImagenet datasets as shown below:

Table 14: Matching network performance on Omniglot as illustrated in [VBL⁺16]

Omniglot	5-way 1-shot Accuracy	5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
Matching Nets	98.1%	98.9%	93.8%	98.5%
Matching Nets (fine tuned)	97.9%	98.7%	93.5%	98.7%

Table 15: Matching network performance on MiniImagenet as illustrated in [VBL⁺16]

MiniImagenet	5-way 1-shot Accuracy	5-way 5-shot Accuracy
Matching Nets (fine tuned)	46.6%	60.0%

3.5.5 Best Approach in This Category

The best approach in this category, evaluated on few-shot classification on Omniglot and MiniImagenet datasets is the relation network on aggregate. On Omniglot, the relation network outperformed all other relative approaches on every test. On MiniImagenet, the relation network was slightly outperformed by prototypical network on the 5-class 5-shot evaluation as shown below:

Table 16: Best performance on both MiniImagenet and Omniglot datasets in this category.

	Best 5-way 1-shot Accuracy	Best 5-way 5-shot Accuracy	20-way 1-shot Accuracy	20-way 5-shot Accuracy
MiniImagenet	Relation net≈50.44%	Prototypical net≈68.2%	-	-
Omniglot	Relation net≈99.6%	Relation net≈99.8%	Relation net≈97.6%	Relation net≈99.1%

3.6 Memory Augmented Methods

Memory augmented methods use external memory storage to improve the learning process of neural networks. The most common from this category is MANN (Memory Augmented Neural Network). Memory augmented techniques have an external memory-based outer learning model rather than a neural network based one, like the optimization-based methods.

3.6.1 MANN (Memory Augmented Neural Network)

MANN or memory augmented neural network by Santoro et al[SBB⁺16] is a memory augmented technique based on the idea of neural turing machines (NTM), which is an algorithm capable of storing and retrieving information from an external memory. It tries to replace the use of hidden states as memory like in RNNs with the use of a neural network augmented with an external memory from where information can be stored and retrieved. MANN and memory augmented method are usually made up of three main components; The controller, Memory, Read and write heads. The controller is either a feedforward neural network or a recurrent neural network, that reads from and writes to the memory. The memory is where information is stored, it is called memory bank or memory matrix because it is a 2D matrix made up of memory cells with N x M rows and columns. The read and write heads are the pointers indicating the addresses of the memory from which the controller has to read and write to.

Information is accessed from the memory using attention mechanisms that uses special read and write operations in conjunction with a weight vector to detect which particular location in the memory is important to read from or write to while ignoring other locations. Weight vector is updated using a gradient based learning algorithm. MANN was evaluated on the Omniglot dataset, as shown below.

Table 17: MANN performance on Omniglot as illustrated in [SBB⁺16]

Omniglot	5-way 5-shot Accuracy	15-way 5-shot Accuracy
MANN	88.4%	88.7%

3.6.2 Others

Other similar techniques in this category include;

- Meta-networks by Tsendsuren Munkhdalai and Hong Yu[MY17] which was tested on Omniglot and MiniImagenet as shown below.

Table 18: Meta-Networks performance on Omniglot as illustrated in [MY17]

Omniglot	5-way 1-shot Accuracy	10-way 1-shot Accuracy	15-way 1-shot Accuracy	20-way 1-shot Accuracy
MetaNet	98.95%	98.67%	97.11%	97.0%

Table 19: Meta-Networks performance on MiniImagenet as illustrated in [MY17]

MiniImagenet	5-way 1-shot Accuracy
MetaNet	49.21 \pm 0.96%

3.7 Other Applications and Areas of Metalearning

So far we have seen a couple of common techniques in metalearning and restricted our focus to the classification setting, however metalearning techniques spread far beyond the supervised learning setting. In this section we will briefly look at some other applications of metalearning in settings other than supervised learning both in machine learning related areas and beyond. However, they will not be discussed in detail as they are beyond the focus of this current work but could probably be considered in a future work.

3.7.1 Bayesian Inference

3.7.1.1 LLAMA

LLAMA (lightweight laplace approximation for meta-adaptation) introduced by Grant et al[GFL⁺18] in 2018 is an approach that tries to solve the challenges encountered in learning quickly from a distribution of tasks otherwise known as few-shot learning. This method is the product of combining gradient based approaches (here MAML) and the hierarchical bayes approach together, for this to work, MAML is restructured so that it's used for probabilistic inference purposes in a hierarchical Bayes model. The experimental analysis occur in two phases; a small-scale or warmup phase and a large-scale phase. The aim is to test if this modified version of MAML[FAL17] can generate samples from the distribution over adapted parameters and if this approach is any good in large-scale meta-learning problems like MiniImagenet. The first phase of the experiment used a toy non-linear model with MAML acting as an algorithm that learns the mean of a Gaussian prior on model parameters, and uses the mean of this prior as an initialization for fast adaptation. The result shows that this approach allows to directly sample models from the task-specific parameter distribution after 10 datapoints of a new, previously unseen sinusoid curve. The second phase used the miniImagenet dataset containing 64 training classes, 12 validation classes, and 24 test classes and compared LLAMA with fine-tuning, nearest neighbor, Matching networks FCE [VBL⁺16], Meta learner LSTM [RL17], SNAIL[MRCA18], Prototypical networks [SSZ17], mAP-DLM (Triantafillou et al, 2017), and MAML. The model was structured to receive N (one was used) of instances of J (five was used) unseen classes, and rated on how well it classifies M new instances still within the J classes. The results show that this approach is efficient enough in large-scale setups and despite the fact that the difference to MAML is minimal, it recorded a competitive and state of the art performance among other baselines.

3.7.2 Reinforcement Learning

3.7.2.1 RoboSumo 3D Environment

Continuous adaptation via meta-learning in nonstationary and competitive environments[ABB⁺17], first published in 2017 and revised in 2018 by Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, Pieter Abbeel introduces the RoboSumo 3D environment to attempt to solve the problem of continuous learning and adaptation from limited data in non stationary environments.

RoboSumo is a competitive multi-agent environment, that allows different agents to play sequences of games against each other and keep adapting to incremental changes in each other’s policies. Non-stationary environments like RoboSumo requires a high level of few-shot learning as the agent is forced to rapidly learn from the little experience it can get before moving to a new environment. To improve continuous adaptation in such environments, the authors use a probabilistic model for MAML in a multi-task reinforcement learning setting, having dynamically changing tasks, policies and trajectories, which are random variables dependent on each other as the policy and trajectories at previous step are used to build a new policy for the current step and the model is optimized using truncated backpropagation through time (TBPTT). This method was applied three types of model architectures or policy networks; a 2-layer MLP, an embedding (1 fully-connected layer replicated across the time dimension) followed by a 1-layer LSTM and a RL2 with a similar architecture as the previous one. The agents are simple multi-leg robots in the form of ants, bugs and spiders capable of nonstationary locomotion. Experiments were carried out to investigate, first, how different adaptation methods would behave in a limited interaction environment of one or very few episodes, second, how many episodes are required for a method to successfully adapt to the changes, third, how do different adaptation methods rank in a competition against each other. The experimental evaluation compared the three previously mentioned adaptation methods with three baselines; naive (no adaptation method attached), an implicitly adapted RL2 and a tracking adaptation that continuously does PPO updates at execution time. Results to the first investigation show that during the first 2 episodes the baselines outperformed the meta-learned adaptations in behavior, however they equalled baselines from the third episode and outperformed them by the sixth and seven episodes. For the second experiment meta-learned adaptation methods required about 100 episodes to adapt and improve their win-rates against constantly improving opponents while baselines declined in performance during the rounds of iterated games. Lastly, in a competition against each other meta-learned policy networks outperformed baselines, however LSTM-based agents were predominant and had a better performance than the others.

3.7.3 Demonstration and Imitation Learning in Robotics

3.7.3.1 NTP (Neural Task Programming)

Learning to Generalize Across Hierarchical Tasks[XNZ⁺17], published in 2018 by Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei and Silvio Savarese introduces the Neural Task Programming (NTP) approach to solve the problem of one-shot meta generalization in robots, by learning modular and reusable neural programs for different hierarchical tasks. NTP has a sophisticated architecture with an LSTM at its core surrounded by other levels such as an observation encoder, task specific interpreter, task specific encoder, API decoder and task specific selection.

NTP algorithm learns to instantiate neural programs from demonstrations of tasks (from a video) so as to generalize to unseen tasks or programs. The algorithm recursively breaks down the general objectives (such as object sorting) into simpler objectives (such as pick and place instead) and delegate a neural program to perform each one of them. Both the decomposition (break down) mechanism and neural programs are trained side by side.

The experimental build up consist of three robot manipulation tasks, which are object sorting, block stacking, and table clean-up that are conducted in a 3D simulated environment. The tasks test for three features which are; generalization to changes in length, topology and semantics, compatibility with image-based input without access to ground truth state and performance in similar

real-world tasks having combinations of the variations used. Training datasets were generated by an expert policy agent and NTP (LSTM) architecture was compared with Flat, Flat (GRU), NTP (no scope) and NTP (GRU) baseline architectures. Results show that NTP was successful and outperformed baselines in the above mentioned tests.

3.7.3.2 One-Shot Imitation Learning

One-shot imitation learning[DAS⁺17] was published in 2017 by Yan Duan, Marcin Andrychowicz, Bradley C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel and Wojciech Zaremba to solve the problem of learning and generalizing from few or a single demonstration of any given task in imitation learning. The solution they came up with was a meta learning framework they called one-shot imitation learning which uses soft attention models initially proposed for machine translations which also gained success in image captioning. Prior to the release of this paper, the most successful techniques in imitation learning had been behavioral cloning which uses a supervised learning approach to map observations to actions and inverse reinforcement learning where a reward function is estimated to acknowledge the demonstrations as near optimal behavior. These techniques (and other previous related publications) however, were either only appropriate for learning a specific task or skill, for other domains or for multiple shot learning which wasn't quite the aim of the authors who not only wanted a meta learning capable imitation learning system but one that is also one-shot learning capable. Their framework *one-shot imitation* was created to achieve this goal. This approach adopts behavioral cloning and DAGGER imitation learning algorithms which only require demonstrations rather than reward functions to be specified and uses a three-module architecture. These modules namely; demonstration network, context network and manipulation network, are actually separate neural nets functioning together to reach the above mentioned mutual goal (one-shot imitation across multiple tasks).

3.7.3.3 DAML

One-shot imitation from observing humans via domain-adaptive meta-learning[YFX⁺18], published in 2018 by Yu et al which introduced DAML or domain-adaptive meta-learning approach, which applies metalearning (MAML based) to try to solve the challenge of imitating humans from just one observation, by enabling robots with the ability to learn from raw video pixels of a human irrespective of substantial domain shift in the perspective, environment, and embodiment between the robot and the observed human. Authors believe that this approach could also be used to imitate animals or a simulated robot, for simulation to real world transfer. They also believe that it is broadly applicable to problems that involve inferring information from out-of-domain data, such as one-shot object recognition from product images.

3.7.4 Behavioral Analysis

3.7.4.1 Machine Theory of Mind (Tom-Net)

This approach[RPS⁺18] uses metalearning to build models for the agents it encounters, just by observing their behaviour. TomNet attempts to tackle the challenge of needing several behavioural observations in order to make good predictions about the characteristics and mental states of agents including their desires, beliefs, and intentions. This is done by gathering a strong a prior model of the behaviour of the agents while allowing for self-improvement to making richer and better predictions about the agents.

TomNet was applied to agents interacting in a simple grid- world environment, and was able to learn to model random algorithmic, and deep reinforcement learning agents from varied populations, outperforming classic Tom (theory of the mind) tasks such as the "Sally-Anne" test. The authors believe that proposed method or framework, which is capable of autonomously learning how to model other agents in its world is a crucial step forward for developing multi-agent AI systems for building intermediating technology for machine - human interaction, and advancing the progress on interpretable AI.

3.8 Summary

This chapter examined state-of-the-art metalearning methods, the best approaches based on the results of their experiments, as well as some of their applications to other areas of machine learning and beyond.

Chapter 4

MAML and MAML-DBL

4.1 Objective

In this chapter we will be examining MAML-DBL alongside MAML. Before introducing the proposed approach we'll take a look at MAML's algorithm and structure or architecture.

4.2 MAML's Algorithm for Classification

MAML's algorithm is split into two very elegant main stages, namely, the single task or task specific stage and the across-tasks stage, and it goes forth and back between these steps at every iteration. The going forth and back of these two steps forms the basis or whole foundation of its originality, robustness, simplicity and efficiency.

Like most metalearning approaches, MAML begins with a task batch which the authors call meta-batch, made up of either training tasks or validation tasks.

For each dataset, the tasks are split into training tasks which are used at train time and test tasks or validation tasks used at test time. Each task, either training or validation, comprises its individual support set and query set. The support set (or train set at train time or validation set at validation time) is used for the task specific stage (the inner loop) while the query set (or meta-train set at train time or meta-validation set at test time) is used for the across tasks stage (the outer loop). Validation is done after a couple of training or metatrain, as used by the authors, iterations. For both Omniglot and MiniImagenet, for example, validation is done after every 500 iterations.

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β : step size hyperparameters

- 1: randomly initialize θ
- 2: **while** not done **do**
- 3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
- 4: **for all** \mathcal{T}_i **do**
- 5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from \mathcal{T}_i
- 6: Evaluate $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ using \mathcal{D} and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
- 7: Compute adapted parameters with gradient descent:
 $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
- 8: Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from \mathcal{T}_i for the meta-update
- 9: **end for**
- 10: Update $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each \mathcal{D}'_i and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
- 11: **end while**

Figure 6: MAML pseudocode for few-shot supervised learning as presented in [FAL17]

The first stage, that is, the inner loop, or from a metalearning perspective the inner learning or base model, focuses, as earlier said, on the acquisition of single task or task-specific knowledge, as shown in lines 4 to 9 of MAML pseudocode in Figure 6 which says:

- After randomly initializing the parameters θ and setting the hyperparameters of either a convolutional or non-convolutional network.
- Sample a task batch or meta batch $\mathcal{T}_i \sim p(\mathcal{T})$ consisting a specified number of N-way classification tasks (task batch or meta batch-size) from a given a distribution over tasks $p(\mathcal{T})$. Although the authors offer an option for non-image classification tasks using non-convolutional networks, we'll be focusing on image classification tasks.
- For each task \mathcal{T}_i in $\mathcal{T}_i \sim p(\mathcal{T})$, sample equal amount of K datapoints \mathcal{D} from the support set S_D for each of the classes of the task \mathcal{T}_i .
- Evaluate or compute the empirical loss or cost $\mathcal{L}_{\mathcal{T}_i}$ using \mathcal{D} , evaluate the gradient $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$.
- Update the parameters θ both for the fully connected and convolutional layer with gradient descent (GD), using the predefined learning rate α for the inner loop or inner learning model.
- Sample a novel or unseen equal amount of K datapoints \mathcal{D}'_i from the query set Q_D of the same task \mathcal{T}_i for the meta-update or outer learning model and set aside.
- Sample next task \mathcal{T}_i from $\mathcal{T}_i \sim p(\mathcal{T})$ repeat previous steps, keep doing this until all the tasks in $\mathcal{T}_i \sim p(\mathcal{T})$ have been evaluated.

This will mark the end of the task-specific stage as well as the inner loop.

The second stage, that is, the outer loop, or from a metalearning perspective the outer learning or meta-level model, which is technically more like an unrolling or unfolding back into the inner learning model to update the initial parameter on the across-tasks knowledge, focuses on the acquisition of across-tasks knowledge, as shown in lines 10 and 11 of Figure 6, which says;

- For the same batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$ as in the first stage, evaluate each \mathcal{T}_i on their previously sampled novel \mathcal{D}'_i from the query set Q_D with respect to their respective updated parameters θ'_i , compute the empirical loss $\mathcal{L}_{\mathcal{T}_i}$ for each of the tasks, summed them up to compute the average loss, which also means combining all the individual losses together to have a single loss; an across tasks loss, as well as an across tasks parameters (weights and biases) of the sampled task batch $\mathcal{T}_i \sim p(\mathcal{T})$.
- Compute the gradient $\nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ and update the initial parameters θ or rather minimize loss in terms of the initial parameters θ using stochastic gradient descent (SGD), using the predefined learning rate β for the outer loop or learning model, which simply means to backpropagate by unrolling through the whole process, that is, the whole of the second stage and first stage, to obtain an across task knowledge, which would obviously involve a gradient through a gradient or the hessian, since we already computed the gradients for the same $\mathcal{T}_i \sim p(\mathcal{T})$ using \mathcal{D} in the first stage.
- Once the initial parameters θ are updated, use them to initialize the weights θ of the inner learning model or task-specific stage for the next batch of tasks sampled from $p(\mathcal{T})$ and repeat the previous steps for n number of iterations. The authors used 60000 iterations.

4.3 MAML as a Learned Optimizer

As mentioned in overview chapter, the basic requirements for a learning algorithm or optimizer is a search direction, which is determined by the gradient, a learning rate and a convergence check. The MAML framework as a whole uses a learned meta-gradient for its search direction to update or optimize the initial parameters (learn the initializations) of the inner learning model, an unlearned or fixed global learning rate (as a learned one, will naturally increase the number of parameters of the network), and iteratively checks if convergence has been attained.

Since MAML uses a fixed learning rate, that is, the learning rate method of the hand-designed optimizer, the choice of the hand-designed optimizer, especially the way it conditions learning rates, plays an important role in both the training speed, convergence, and accuracy of the model. An optimizer with bad learning rate handling could lead to vanishing learning rate and stagnancy or non-convergence, which was one of the main issues of the Adagrad optimizer. On the other hand, non-adaptive learning rates methods, although they usually generalize better, could slow down training and convergence, especially when sparse features are involved such as with vanilla gradient descent methods. But there wouldn't be any need to learn the learning rates and add more learnable parameters to the network as well as more computations and memory overhead if there were optimizers that had a good way of handling the learning rates.

Vanilla MAML, as a learned optimizer, is an Adam-based adaptive method, which is applied in the across tasks stage (in the outer loop).

The authors opted for an Adam-based method, because it is an advanced and more robust variant of the vanilla SGD with added features and desirable properties, as well as to instantly take advantage of the desirable properties it offers and at the same time transmit the benefits to its learned version.

Adaptive Moment Estimation or Adam [KB14] published in 2015, as shown in Figure 7, is a learning algorithm method or optimizer that combines or adds two concepts not present in vanilla SGD, which are momentum and adaptive learning rate. Momentum allows for smoother trajectory as opposed to non-momentum based vanilla SGD methods where movement is wavy and oscillatory,

causing slow training and convergence. The adaptive learning rate technique was developed to address slow convergence of non-adaptive learning rate methods due to sparsity issues, as some parameters could be stuck at the same place for a long time due to no update and since the learning rate is the same for all parameters, the few times they get updated, they only move the same length as the dense features (whose parameters are frequently being updated), thus causing an overall unevenness in movement and hence a slower training and convergence. Adaptive methods adapt the learning rate such that the more frequently a parameter gets updated the lower or the more decayed its learning gets and the less frequently a parameter gets updated the less decayed the global learning rate gets, this is done using exponentially running average. However, in situations where there are almost no sparse features and the parameters get updated very frequently, it might be better to use non-adaptive learning rate methods, as using adaptive learning rate methods (unless carefully tuned by the user) could lead to low learning rates and slow down or even stagnate convergence.

Although Adam, as the name implies, uses initialization bias correction to ensure that the expectation or estimation of the gradients by both the momentum vector and the parameter update history vector (to diminish the learning rates) are always as close as possible to the true first and second moment respectively, in order to try to avoid rapidly diminishing learning rates, nonetheless, there are cases where training with Adam either gets relatively slow with time, stagnates [LXLS19] or even fails to converge to the optimal minimum [RKK19].

This thesis proposes a modified version of the vanilla MAML that uses a more recent and flexible variant of the Adam optimizer but with dynamic bound, that enables it start out as an adaptive learning rate method and later on, a non-adaptive learning rate method, that is, it takes the best out of both adaptive and non-adaptive approaches, making it immune to the vanishing learning rate problem without adding excessive computations or memory overhead.

Require: α : Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
 $m_0 \leftarrow 0$ (Initialize 1st moment vector)
 $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 $t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
 $t \leftarrow t + 1$
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
 $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
 $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
 $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 7: Pseudocode of Adam’s algorithm and update rule as illustrated in [KB14]

4.3.1 Batch Normalization

Another method used by the authors of MAML to improve performance is batch normalization which is a technique to ensure that the inputs into every layer of the neural network comes

from the same distribution, which in turns reduces noise and the variance of inputs from layer to layer and mini-batch to mini-batch, which is meant to speed up training and convergence.

Batch Normalization is the normalization or rescaling of any input or hidden layer of a neural network. This is achieved by appropriately adjusting or balancing the output from each previous activation layer, to ensure that it is neither too high nor too low.

Getting rid of all imbalance in each of the hidden layers of the neural network helps avoid instability and improves training speed as well as convergence [FD17].

Batch normalization H' normalizes a minibatch of activations H by subtracting the batch mean μ from H and dividing by the batch standard deviation σ . Which also gives rise to or adds an extra layer to each neuron as well as two trainable parameters to each layer at training time, that is, the learnable weight γ which controls (or which the network uses to adjust) σ , and bias β which controls (or which the network uses to adjust) μ [GBC16], as shown below:

$$H'_i = \frac{H_i - \mu}{\sigma}$$
$$y_i = \gamma H'_i + \beta$$

Batch normalization is used before a ReLU nonlinearity (pre-nonlinearity usage) at every hidden layer of the MAML algorithm for few-shot classification as recommended by the authors of the original batch normalization paper [IS15]. However, it's still a bit unclear as to where to place the batch normalization layer for better performance, as some other researchers [da16] argue that they obtained better results following a post-nonlinearity or activation usage of batch normalization. However, in the end, it's up to the software engineer or data-scientist to decide or examine which is best for his or her model or network.

4.3.2 Other Useful Techniques

4.3.2.1 ReLU nonlinearity

ReLU or the rectifier linear unit function in machine learning, as shown in a simplified form in the equation below, is an activation function that simply gets rid of any negative output from the linear layer by setting them equal to zero [Wik20c]. It can be used in most types of neural networks, especially convolutional neural networks (CNNs), where it has shown great success. ReLU is the main activation used when evaluating MAML on classification tasks.

$$f(x) = x^+ = \max(0, x)$$

4.3.2.2 Softmax Function

Softmax $\sigma(z)_i$ or $\text{Softmax}(x_i)$ is an activation function that is used to convert each element z_i in the vector of logits z (the unnormalized predictions or output vector of a model) into probabilities that add up to one. Its output is a vector representing the probability distribution of all the possible outcomes. It is usually applied to the last layer of a neural network, which is also the case of the MAML algorithm for few-shot classification tasks [Uni18].

The softmax function is as shown below, where e^{z_i} is the application of the exponential function to each element z_i in z and $\sum_{j=1}^K e^{z_j}$ the sum of the application of the exponential function to all

the elements z_j in the vector of logits z :

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

4.4 MAML’s Structure for Classification

4.4.1 Implementation of the MAML Algorithm in Tensorflow

MAML for image classification involves three layers; the embedding function or CNN, the fully connected layer or inner model and the metalearning layer or the outer model.

Below are extracts or snapshots of the implementation of the MAML algorithm in tensorflow, as coded by the authors, illustrating specified definitions used to evaluate MAML on few-shot image classification. For full details and code refer to [Fin].

4.4.1.1 CNN

- The CNN pipeline for this method originally consist of 4 convolution modules each with a 3 x 3 convolutions or sized filters and 64 filters, followed by batch normalization, a ReLU nonlinearity, and a 2 x 2 max-pooling, as shown in lines 25-37 of the utils.py file and lines 210-218 of file maml.py in the source code folder:

```

25  ## Network helpers
26  def conv_block(inp, cweight, bweight, reuse, scope, activation=tf.nn.relu, max_pool_pad='VALID', residual=False):
27      """ Perform, conv, batch norm, nonlinearity, and max pool """
28      stride, no_stride = [1,2,2,1], [1,1,1,1]
29
30      if FLAGS.max_pool:
31          conv_output = tf.nn.conv2d(inp, cweight, no_stride, 'SAME') + bweight
32      else:
33          conv_output = tf.nn.conv2d(inp, cweight, stride, 'SAME') + bweight
34      normed = normalize(conv_output, activation, reuse, scope)
35      if FLAGS.max_pool:
36          normed = tf.nn.max_pool(normed, stride, stride, max_pool_pad)
37      return normed

```

Figure 8: Code defining the CNN routine extracted from utils.py in[Fin]

```

210  def forward_conv(self, inp, weights, reuse=False, scope=''):
211      # reuse is for the normalization parameters.
212      channels = self.channels
213      inp = tf.reshape(inp, [-1, self.img_size, self.img_size, channels])
214
215      hidden1 = conv_block(inp, weights['conv1'], weights['b1'], reuse, scope+'0')
216      hidden2 = conv_block(hidden1, weights['conv2'], weights['b2'], reuse, scope+'1')
217      hidden3 = conv_block(hidden2, weights['conv3'], weights['b3'], reuse, scope+'2')
218      hidden4 = conv_block(hidden3, weights['conv4'], weights['b4'], reuse, scope+'3')

```

Figure 9: Code defining the hidden layers of the CNN extracted from maml.py in[Fin]

- While evaluating MAML on each of the major meta learning datasets, slight modifications were applied, such as, for Omniglot, strided convolutions were used for downsampling instead of max-pooling, and for MiniImagenet instead of 64 filters, 32 filters were used per convolution layer to reduce overfitting.

- The rest of the convolutional network definitions can be found in lines 185-225 of maml.py in [Fin].

4.4.1.2 Fully connected layer

To test the flexibility of the MAML learned learning algorithm in classification tasks, the fully connected layer was implemented for both non-convolutional and convolutional networks. For non-convolutional networks, the authors used 4 hidden layers of sizes 256, 128, 64, 64, each of which includes batch normalization and ReLU nonlinearities, and they are followed by a linear layer and softmax as specified in Figure 10. The channels from line 42-45 in Figure 10 indicate colored input images such as those from miniImagenet or grayscale input images such as those from the omniglot dataset. The cost function used for the fully connected layer is the cross-entropy loss function. The remaining fully connected network definitions can be found in lines 50-177 of maml.py in [Fin].

```

31         elif FLAGS.datasource == 'omniglot' or FLAGS.datasource == 'miniimagenet':
32             self.loss_func = xent
33             self.classification = True
34             if FLAGS.conv:
35                 self.dim_hidden = FLAGS.num_filters
36                 self.forward = self.forward_conv
37                 self.construct_weights = self.construct_conv_weights
38             else:
39                 self.dim_hidden = [256, 128, 64, 64]
40                 self.forward=self.forward_fc
41                 self.construct_weights = self.construct_fc_weights
42             if FLAGS.datasource == 'miniimagenet':
43                 self.channels = 3
44             else:
45                 self.channels = 1
46             self.img_size = int(np.sqrt(self.dim_input/self.channels))

```

Figure 10: Lines 34-41 of maml.py in [Fin], denoting the convolutional and non-convolution pathways.

4.4.1.3 Metalearning layer

The metalearning layer definitions can also be found in lines 50-177 of maml.py as well as in main.py in [Fin]. The loss function used for the metalearning layer is also the cross-entropy loss function.

In general, the authors claim that their method uses fewer overall parameters compared to matching nets[VBL⁺16] and meta-learner LSTM [RL17], since no additional parameters other than the weights of the classifier were used. But still, MAML outperforms both techniques as seen in the previous chapter.

MAML is also applicable for learning tasks in other machine learning settings such as reinforcement learning and regression.

4.5 Proposed Approach

MAML is an adaptive learning rate metalearning approach that relies on a fixed learning rate method. For improved performance, trade offs are usually made.

Some researchers such as Li et al in [LZCL17] suggest not just learning the initializations (like MAML does) but also learning the learning rates for improved performance which obviously comes at the cost of increased amount of trainable parameters, computations and memory overhead. Since MAML, is a method that also involves computing the hessian-vector product which is already computationally costly, further increasing the computations and parameters by learning the learning rates could make MAML difficult to train unless a trade off is made in which the extra second order gradient step must be gotten rid of to avoid excessive computational overhead from the increased number of trainable parameters of the learned learning rate method.

This thesis however proposes a method called MAML-DBL (MAML with Dynamic Bound Learning rate) that uses learning rates with dynamic bounds to try to improve or boost MAML’s convergence speed or/and accuracy (depending on the prevalent type of features) and prevent vanishing learning rates, without the need to increase the amount of parameters, computations, or memory overhead.

MAML-DBL, as a learned optimizer, is an Adabound based adaptive method. An Adabound-based method was opted for because it possesses the ability to transform from an adaptive learning rate method to a momentum based method by gradually annealing the learning rate as training progresses.

Adaptive moment estimation with dynamic bound or Adabound [LXLS19] as shown in Figure 11 is a variant of the Adam optimizer that adds the concept of dynamic bound to its design. Adabound tries to combine the fast initial progress of adaptive methods once training begins and gradually switches to the good final generalization (non-discriminative) properties of SGD towards the end. This is done using dynamic bound a technique inspired by gradient clipping (clips gradients that are larger than a threshold) that gradually clips the learning rates as training progresses such that the learning rates constrained within the dynamic bound (a dynamic lower and upper bound, that gradually draws closer and closer to the learning rate from both sides as training progresses) gets more restricted (less and less adaptive), gradually transforming the behaviour of the learning rate from an adaptive to non-adaptive one, such that it generalizes rather than adapts towards the end of training. Dynamic bound enables Adabound to avoid any chances of vanishing learning rate without compromising or sacrificing the softened oscillations or less intense fluctuations enjoyed by momentum based methods (as opposed to vanilla SGD methods).

Algorithm 2 ADABOUND

Input: $x_1 \in \mathcal{F}$, initial step size α , $\{\beta_{1t}\}_{t=1}^T$, β_2 , lower bound function η_l , upper bound function η_u

- 1: Set $m_0 = 0, v_0 = 0$
 - 2: **for** $t = 1$ **to** T **do**
 - 3: $g_t = \nabla f_t(x_t)$
 - 4: $m_t = \beta_{1t}m_{t-1} + (1 - \beta_{1t})g_t$
 - 5: $v_t = \beta_2v_{t-1} + (1 - \beta_2)g_t^2$ and $V_t = \text{diag}(v_t)$
 - 6: $\hat{\eta}_t = \text{Clip}(\alpha/\sqrt{V_t}, \eta_l(t), \eta_u(t))$ and $\eta_t = \hat{\eta}_t/\sqrt{t}$
 - 7: $x_{t+1} = \Pi_{\mathcal{F}, \text{diag}(\eta_t^{-1})}(x_t - \eta_t \odot m_t)$
 - 8: **end for**
-

Figure 11: Pseudocode of Adabound’s algorithm and update rule as illustrated in [LXLS19]

4.6 Summary

This chapter elaborated on MAML’s algorithm, structure and experimental details. It also introduced MAML-DBL, the proposed method, and the technique it uses.

Chapter 5

Experimental Evaluation

5.1 Objective

The objective of this chapter is to describe the experiments conducted for both MAML-DBL and MAML as well as to show and detail the experimental results, graphs, observations and final assessment.

5.2 Software requirements

The MAML source code requires the following dependencies to run:

- python 2.* or python 3.*
- TensorFlow v1.0 or higher.

The MAML source code can be accessed and downloaded at [Fin]. Usage instructions can also be found on the same page. Other details on data preprocessing, usage and running of the MAML source code can be found in the appendix A.

5.3 MAML’s Experiment Details

To evaluate MAML experiments were carried out on both the omniglot and miniImagenet dataset in the following manner;

For omniglot, the authors used 1200 from the 1623 characters in the dataset for training and the remaining for testing. Data augmentation of the omniglot dataset with rotations by multiples of 90 degrees was also applied as suggested by Santoro et al in [SBB⁺16]. The 5-way or class 1-shot (meaning that each task in batch of task will have 5 classes with a single example each) convolutional and non-convolutional MAML models on omniglot, were each trained with 1 gradient step (per task), a meta batch-size of 32 tasks and a learning rate α of 0.4. The network was evaluated with 3 gradient steps and the same learning rate α .

The 20-way or class 1-shot (20 classes with 1 example each per task in batch) convolutional model was trained with 5 gradient steps, a meta batch-size of 16 tasks, and a learning rate of 0.1. And was evaluated also using 5 gradient steps and the same step-length.

On MiniImagenet, both the 5-way 1-shot and 5-way 5-shot models were trained with 5 gradient steps, a learning rate of 0.01, and a meta batch size of 4 tasks for 1-shot and 2 tasks for 5-shot. The model was evaluated using 10 gradient steps at test time with 15 examples per class used to evaluate the post-update meta gradient in compliance with Ravi and Larochelle in [RL17]. The authors trained all the above mentioned models for 60,000 iterations with a hardware setup that included a single NVIDIA Pascal Titan X GPU.

5.4 Experiments

The experiments are divided into four, two experiments for the Omniglot dataset and the other two for the MiniImagenet dataset.

All experiments were conducted in conformity with the experiments in the MAML paper in terms of architecture and hyperparameter setting, except for the learning rate value for MAML-DBL which was set to 0.01 as recommended in [Kim19]. A single GPU was used to train both MAML and MAML-DBL models.

The details of each experiment on the Omniglot and MiniImagenet datasets will be expressed with reference to the input structure, network structure, results, accuracy comparative graph and observations.

5.4.1 Experiment 1: 5-way 1-shot Omniglot dataset

5.4.1.1 Input Structure

The input structure of MAML-DBL for this experiment was as follows:

- Images are grayscale and are resized to 28x28 pixels with data augmentation consisting of rotations by multiples of 90 degrees as suggested in [SBB⁺16],
- Each task-batch-size or meta batch-size is made up of 32 tasks per meta iteration,
- Each task in batch is a 5-class classification task with a single example for each class,
- One gradient step update per task,
- A learning rate of 0.01.

5.4.1.2 Network Structure

The network structure of MAML-DBL for this experiment was as follows:

- CNN uses strided convolutions for downsampling, 64 filters followed by batch normalization and ReLU activation per convolution layer, no pooling layer involved,
- Fully connected layer consist of 4 hidden layers with batch normalization and ReLU nonlinearity each, followed by a last layer and softmax,
- Uses the cross entropy loss function for both the inner and outer loop.

5.4.1.3 Results

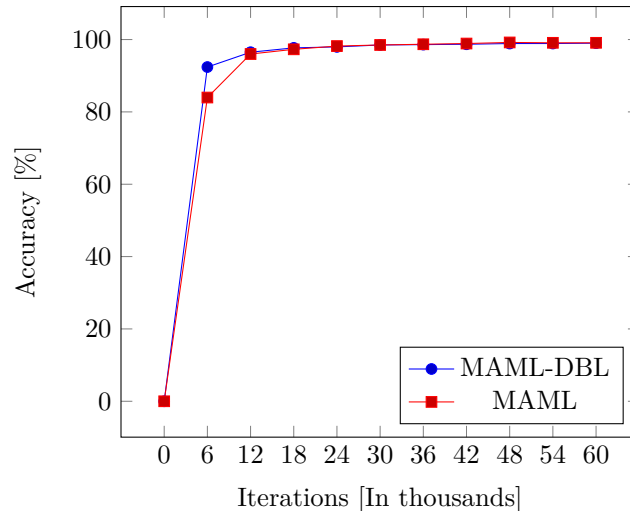
The results of MAML-DBL alongside the MAML baseline for this experiment were as follows:

Table 20: Training results in terms of average accuracy and approximated training time.

Omniglot [LSGT11]	Accuracy	Approximated Training time
MAML	96.8%	$\approx 6hrs$
MAML-DBL	97.7%	$\approx 6hrs30$

5.4.1.4 Accuracy Graph

The accuracy graph after every 6000 iterations of MAML-DBL alongside the MAML baseline for this experiment is as follows:



5.4.1.5 Observations

Facts observed during this experiment include:

- For this model, MAML-DBL behaved exactly as expected, and showed faster initial convergence than the baseline as illustrated in the accuracy graph.
- As expected from a learned optimizer and metalearning method as well as an adaptive-based method, the training started at an average speed but became faster as training progresses (as the initializations were being learned) with very high accuracy. Towards the end however, as the switch to a momentum based SGD method was made, training got slower, but MAML-DBL still generalized well and maintained a very high accuracy till training ended.
- Towards the end, as the learning rates were being annealed, the high accuracy did not drop and was sometimes higher than the baseline, meaning that MAML-DBL was able to generalize better towards the end, which could be evidence that sparsity was not much of an issue during training and that the claims of faster initial convergence and better generalizations than adaptive methods at the end are indeed true for this specific model.
- Although MAML also slowed down towards the end, it recorded a better overall training time than MAML-DBL.

5.4.2 Experiment 2: 20-way 1-shot Omniglot dataset

5.4.2.1 Input Structure

The input structure of MAML-DBL for this experiment was as follows:

- Images are grayscale and are resized to 28x28 pixels with data augmentation consisting of rotations by multiples of 90 degrees as suggested in [SBB⁺16],
- Each task-batch-size or meta batch-size is made up of 16 tasks per meta iteration,

- Each task in batch is a 20-class classification task with a single example for each class,
- Trained with five gradient steps per task,
- A learning rate of 0.01.

5.4.2.2 Network Structure

The network structure of MAML-DBL for this experiment was as follows:

- CNN uses strided convolutions for downsampling, 64 filters followed by batch normalization and ReLU activation per convolution layer, no pooling layer involved,
- Fully connected layer consist of 4 hidden layers with batch normalization and ReLU nonlinearity each, followed by a last layer and softmax,
- Uses the cross entropy loss function for both the inner and outer loop.

5.4.2.3 Results

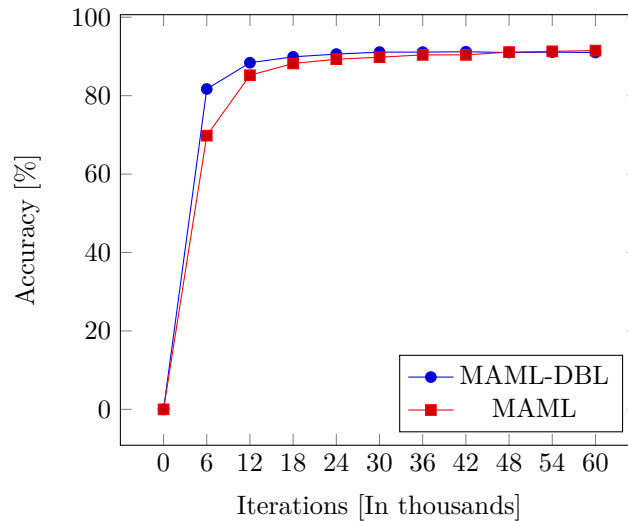
The results of MAML-DBL alongside the MAML baseline for this experiment were as follows:

Table 21: Training results with reference to average accuracy and approximated training time.

Omniglot [LSGT11]	Accuracy	Approximated Training time
MAML	87.7%	$\approx 36hrs$
MAML-DBL	89.7%	$\approx 48hrs$

5.4.2.4 Accuracy Graph

The accuracy graph after every 6000 iterations of MAML-DBL alongside the MAML baseline for this experiment is as follows:



5.4.2.5 Observations

Facts observed during this experiment include:

- For this model, MAML-DBL behaved in quite a similar manner as the previous experiment, and showed faster initial convergence than the baseline with a slightly lower accuracy than the baseline towards the end as illustrated in the accuracy graph.
- As expected from a learned optimizer and metalearning method as well as an adaptive based method training started at a good initial speed, it picked up as training progressed (as the initializations are being learned) while still maintaining high accuracy, but then it gradually got lengthy as training went on till the end, which is probably due to the switch to a momentum based SGD method, but MAML-DBL still generalized well and maintained a high accuracy till the end.
- MAML-DBL started off with a higher accuracy advantage over the baseline, and as the learning rate was gradually annealed, the high accuracy did not drop but was slightly surpassed by the baseline, meaning that MAML-DBL was able to also generalize well towards the end (although not as the previous experiment), which could also indicate that sparsity was not much of an issue during training and that the claim of faster initial convergence is indeed true for this specific model. As an Adabound-based adaptive method, MAML-DBL, I'd say generalized well towards the end but not as good as the baseline at the very end as shown in the accuracy graph.
- Although training with MAML also got lengthy as training went on, but not as lengthy as with MAML-DBL, which is why it features a better overall training speed than the proposed method but falls short as regards the overall average accuracy.

5.4.3 Experiment 3: 5-way 1-shot MiniImagenet dataset

5.4.3.1 Input Structure

The input structure of MAML-DBL for this experiment was as follows:

- Each example is a 84x84 pixel colour image.
- Each task-batch-size or meta batch-size is made up of 4 tasks per meta iteration,
- Each task in batch is a 5-class classification task with a single example per class,
- Five gradient steps per task,
- A learning rate of 0.01.

5.4.3.2 Network Structure

The network structure of MAML-DBL for this experiment was as follows:

- CNN Uses 32 filters followed by batch normalization and ReLU activation per convolution layer. The convolution layer is followed by a 2x2 max-pooling layer,
- Fully connected layer consist of 4 hidden layers with batch normalization and ReLU nonlinearity each, followed by a last layer and softmax,
- Uses the cross entropy loss function for both the inner and outer loop.

5.4.3.3 Results

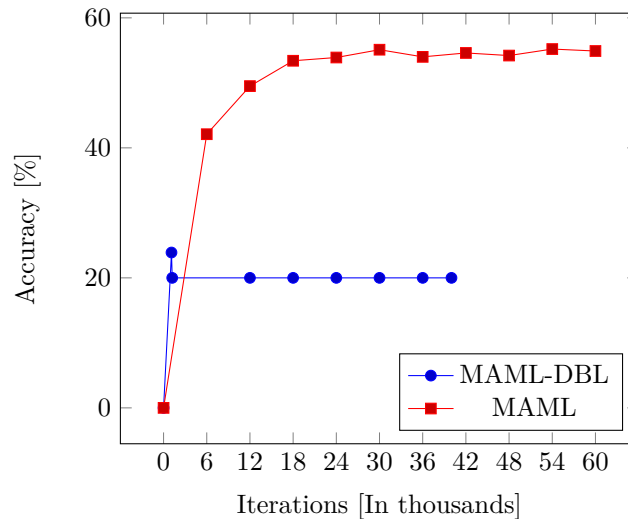
The results of MAML-DBL alongside the MAML baseline for this experiment were as follows:

Table 22: Training results with reference to average accuracy and approximated training time.

MiniImagenet [VBL ⁺ 16]	Accuracy	Approximated Training time
MAML	52.7%	$\approx 87hrs$
MAML-DBL	-	-

5.4.3.4 Accuracy Graph

The accuracy graph after every 6000 iterations of MAML-DBL alongside the MAML baseline for this experiment is as follows:



5.4.3.5 Observations

Facts observed during this experiment include:

- In this model, MAML-DBL behaved in quite a different way from the previous experiments, the only similarity was that it showed a rather promising initial accuracy, but as training progressed it stagnated and after a while, stopped training, as shown in the accuracy graph.
- As expected from a learned optimizer and metalearning method as well as an adaptive-based method, training started at a initial speed, but as training went on or as it gradually transformed to a momentum based SGD method, it failed to generalize as the accuracy gradually declined, then stagnated before the training was brought to an abrupt end.
- Other hyperparameter settings and tuning were experimented to see if MAML-DBL's behaviour would improve, however in all, it showed the same unstable, irregular and unprecedented behaviour always tending, sometimes quickly sometimes slowly, towards stagnancy and non-convergence.
- Judging severe sparsity issues might be the problem, gamma or the convergence rate of the bounds in the Adabound setting was reduced to make the proposed method behave a

bit longer like an adaptive method before transforming, since the baseline, which uses an adaptive-based method, displayed no such problems in training. This brought about some improvements, but in the end, the proposed method was imminently doomed for stagnation and incomplete training.

- I believe further reducing gamma would have brought about further improvements but that would've made the whole point of these experiments useless, as further reducing gamma would gradually disfigure and strip the proposed method of its transformative nature and reduce it to just another Adam based method, making it behave just like the baseline. Rather than aggressively reducing gamma for results, it would probably be better to use a full adaptive method.
- The baseline behaved exactly as usual only at a slower pace. But showed no signs of stagnation like the proposed method.

5.4.4 Experiment 4: 5-way 5-shot MiniImagenet dataset

5.4.4.1 Input Structure

The input structure of MAML-DBL for this experiment was as follows:

- Each example is a 84x84 pixel colour image.
- Each task-batch-size or meta batch-size is made up of 2 tasks per meta iteration,
- Each task in batch is a 5-class classification task with a five examples per class,
- Five gradient steps per task,
- A learning rate of 0.01.

5.4.4.2 Network Structure

The network structure of MAML-DBL for this experiment was as follows:

- CNN Uses 32 filters followed by batch normalization and ReLU activation per convolution layer. The convolution layer is followed by a 2x2 max-pooling layer,
- Fully connected layer consist of 4 hidden layers with batch normalization and ReLU nonlinearity each, followed by a last layer and softmax,
- Uses the cross entropy loss function for both the inner and outer loop.

5.4.4.3 Results

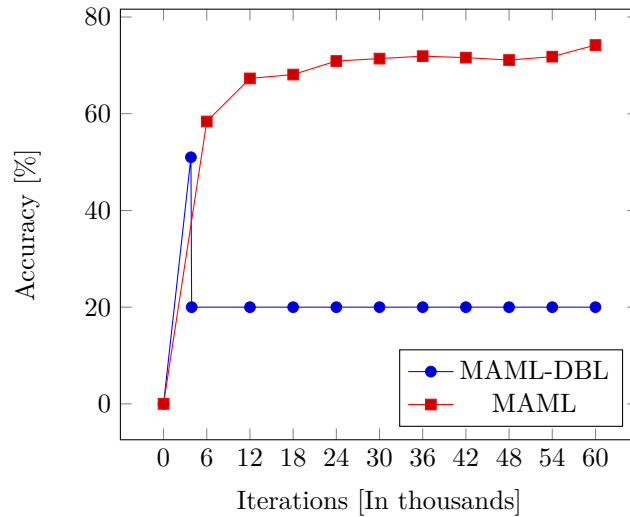
The results of MAML-DBL alongside the MAML baseline for this experiment were as follows:

Table 23: Training results in terms of average accuracy and approximated training time.

MiniImagenet [VBL ⁺ 16]	Accuracy	Approximated Training time
MAML	69.7%	$\approx 115hrs$
MAML-DBL	-	-

5.4.4.4 Accuracy Graph

The accuracy graph after every 6000 iterations of MAML-DBL alongside the MAML baseline for this experiment is as follows:



5.4.4.5 Observations

Facts observed during this experiment include;

- For this model, MAML-DBL behaved in a quite similar way as the previous experiment, the only difference was that it took a little longer to stagnate.
- MAML behaved as usual only at a slower pace. But showed no signs of stagnation like the proposed method.

5.4.5 Assessment of the Proposed Method

Some of the advantages of the proposed technique observed from the experiments include:

- MAML-DBL outperformed baseline in both of the Omniglot experiments.
- Showed high initial accuracy even in the failed experiments.
- Generalized well at the end in both of the Omniglot tests, surpassing the baseline in one of them.

The disadvantages observed from the experiments include;

- Slower overall training speed compared to the baseline.
- Slow training speed before and during stagnation in the MiniImagenet test and gradually reducing training speed towards the end in some of the Omniglot experiments especially the second one.
- Poor performance, stagnation and failure to converge in both of the MiniImagenet experiments.

5.5 Summary

This chapter illustrated the results and observations of the experiments conducted for both MAML-DBL and the baseline on the Omniglot and MiniImagenet datasets.

Chapter 6

Conclusion

6.1 Thesis Overview

Learning learning algorithms are increasingly being proposed in recent research works as machine learning experts and engineers have observed that they perform much better than hand designed learning algorithms, but then, what are learning learning algorithms and what do they look like? This thesis has made an attempt to explain the concept of learned learning algorithms and meta-learning to the best of the author's ability relying on books, literary publications and experiments. It starts with a general definition of optimization and its relevance to learning as well as to learned learning algorithms and metalearning, followed by an itemization of the steps and an introduction to commonly used metalearning datasets. This is followed up by an analysis of state-of-the-art techniques as well as their literature, and applications or extensions to other fields of artificial intelligence, leading to a practical examination, testing and verification of the code of one of the state-of-the-art approach called MAML or model-agnostic metalearning.

MAML, as well as the metalearning method proposed by this thesis, which is also based on MAML called MAML-DBL or MAML with Dynamic Bound Learning rate, were evaluated using the most common metalearning datasets to find out if metalearning is actually a viable solution to achieving high accuracy and training speed with limited resources or data.

The experiments were conducted using the Omniglot and MiniImagenet datasets and a GPU. During the experiments, both techniques showed specific advantages and disadvantages, while MAML showed consistency and better speed in all the experiments, it was outperformed by MAML-DBL, in terms of overall accuracy, in all the experiments conducted on Omniglot. MAML-DBL however failed to perform on the MiniImagenet dataset.

The results of the experiments does prove in most of the cases that metalearning is a possible solution in times of limited computational resources, considering all models were trained from the scratch on entire datasets. Furthermore, certain conclusions could be drawn from the results of the experiments for better accuracy and/or speed when practically using a metalearning method.

First of all, the results on Omniglot show that the 5-way 1-shot model performed better in terms of speed and accuracy than the 20-way 1-shot model, which could indicate that metalearning models whose tasks have fewer number of classes are likely to learn faster and perform better than their counterparts with tasks having larger number of classes, including deep networks and vanilla machine learning methods. So, for accuracy and speed, it may be adviceable to go for lesser number of classes or ways when using a method such as MAML or MAML-DBL, which is one of the benefit of metalearning, that is, its versatility, as an entire dataset can be broken down into small tasks having fewer classes and fed via few-shots into the metalearning network.

In addition, the results on MiniImagenet with MAML, show that the 5-way 5-shot attained a

higher accuracy than the 5-way 1-shot model, however it took a shorter time to train the latter than the former. So, if the focus is accuracy rather than speed, increasing the shots by a little bit could do the trick, however if speed is more important, keeping the shots and the classes down could make a whole lot of difference as far as speed is concerned.

6.2 Future Work

I hope in a future work to find better ways of improving the proposed method and make it work efficiently not only on the Omniglot dataset but especially on the MiniImagenet dataset.

As a continuation to this thesis, a direction I believe could be a starting point to improving MAML-DBL is a data and architectural-based approach, rather than an algorithmic approach as presented by this thesis, by researching efficient ways to preprocess and batch-normalize the data and the inputs to the hidden layers as well as find more suitable architectures in order to improve training, especially on the MiniImagenet dataset, although other algorithmic trade-offs could also be researched and examined.

6.3 Final Thoughts

In conclusion, I believe this thesis have presented the research carried out to study, implement and test learned learning algorithms or metalearning methods such as MAML and MAML-DBL, in order to provide a general knowledge on the subject as well as share useful recommendations backed up by the results of the experiments to make metalearning an easily implementable option to anyone desiring a versatile, fast and accurate model, despite certain limitations such as data scarcity or limited computational resources.

Bibliography

- [Re17] Revision c21308ea. Loss functions [online]. 2017. Available from: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html [cited 20 May 2020]. 11
- [ABB⁺17] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *CoRR*, abs/1710.03641, 2017. 2, 21
- [ADC⁺16] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016. Available from: <http://arxiv.org/abs/1606.04474>. 13, 16
- [BP14] Sergiy Butenko and Panos M. Pardalos. *Numerical Methods and Optimization: An Introduction*. Chapman and Hall/CRC; 1 edition, 2014. 5
- [bre] brendenlake. omniglot [online]. Available from: <https://github.com/brendenlake/omniglot>. 9, 48
- [da16] ducha aiki. caffe-net-benchmark [online]. 2016. Available from: <https://github.com/ducha-aiki/caffe-net-benchmark/blob/master/batchnorm.md> [cited June 2020]. 29
- [DAS⁺17] Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *CoRR*, abs/1703.07326, 2017. 2, 23
- [Dev] Michel Devy. Miniimagenet samples from 6 classes [online]. Available from: https://www.researchgate.net/figure/MiniImageNet-samples-from-6-classes_fig3_334758545. xiii, 10
- [ES16] Harrison A Edwards and Amos J. Storkey. Towards a neural statistician. *ArXiv*, abs/1606.02185, 2016. 18
- [FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. Available from: <http://arxiv.org/abs/1703.03400>. xiii, xv, 11, 13, 21, 26
- [FD17] FD. Batch normalization in neural networks [online]. 2017. Available from: <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c> [cited May 2020]. 29
- [Fin] Chelsea Finn. Code for "model-agnostic meta-learning for fast adaptation of deep networks" [online]. Available from: <https://github.com/cbfinn/maml>. xiii, 30, 31, 33
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. 5, 6, 29
- [GFL⁺18] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas L. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *CoRR*, abs/1801.08930, 2018. 2, 21

- [HAMS20] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey, 2020. 7
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 7, 16
- [HYC01] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In *IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001)*, pages 87–94. Springer, 2001. 16
- [IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. Available from: <http://arxiv.org/abs/1502.03167>. 29
- [JSH⁺18] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018. Available from: <http://arxiv.org/abs/1807.11205>. 1
- [KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. Available from: <http://arxiv.org/abs/1412.6980>. xiii, 27, 28
- [Kim19] Junho Kim. Adabound-tensorflow [online]. 2019. Available from: <https://github.com/taki0112/AdaBound-Tensorflow>. 34
- [Koc15] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015. xv, 17
- [LSGT11] Brenden M. Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B. Tenenbaum. One shot learning of simple visual concepts, 2011. xiii, 9, 10, 13, 34, 36
- [LXLS19] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *CoRR*, abs/1902.09843, 2019. Available from: <http://arxiv.org/abs/1902.09843>. xiii, 28, 32
- [LZCL17] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: learning to learn quickly for few-shot learning. 2017. xv, 14, 15, 32
- [MRCA18] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. 25 Feb 2018. 2, 21
- [MY17] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *CoRR*, abs/1703.00837, 2017. xv, 2, 20, 21
- [NAS18] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. Available from: <http://arxiv.org/abs/1803.02999>. xv, 13, 15, 16
- [Rava] Sachin Ravi. Meta-learning: Learning to learn fast [online]. Available from: <https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html>. xiii, 8

- [Ravb] Sachin Ravi. Optimization as a model for few-shot learning [online]. Available from: <https://iclr.cc/archive/>. xiii, 8, 9
- [RKK19] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *CoRR*, abs/1904.09237, 2019. Available from: <http://arxiv.org/abs/1904.09237>. 28
- [RL17] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. xv, 16, 21, 31, 33
- [RPS⁺18] Neil C. Rabinowitz, Frank Perbet, H. Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. Machine theory of mind. *CoRR*, abs/1802.07740, 2018. 23
- [SBB⁺16] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, 2016. xv, 20, 33, 34, 35
- [SSZ17] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017. Available from: <http://arxiv.org/abs/1703.05175>. xv, 17, 18, 21
- [SYZ⁺17] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. *CoRR*, abs/1711.06025, 2017. Available from: <http://arxiv.org/abs/1711.06025>. xv, 2, 18, 19
- [Uni18] Uniqtech. Understand the softmax function in minutes [online]. 2018. Available from: <https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d> [cited May 2020]. 29
- [VBL⁺16] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016. Available from: <http://arxiv.org/abs/1606.04080>. xv, 10, 13, 19, 21, 31, 38, 39, 48
- [Wik20a] Wikipedia contributors. Meta learning — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: [https://en.wikipedia.org/wiki/Meta_learning_\(computer_science\)](https://en.wikipedia.org/wiki/Meta_learning_(computer_science)). 7
- [Wik20b] Wikipedia contributors. Optimization algorithms — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: https://en.wikipedia.org/wiki/Mathematical_optimization#Optimization_algorithms. 6
- [Wik20c] Wikipedia contributors. Rectifier (neural networks) — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)). 29
- [Wik20d] Wikipedia contributors. Regression analysis — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: https://en.wikipedia.org/wiki/Regression_analysis. 6
- [Wik20e] Wikipedia contributors. Statistical classification — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: https://en.wikipedia.org/wiki/Statistical_classification. 6

- [Wik20f] Wikipedia contributors. Supervised learning — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: https://en.wikipedia.org/wiki/Supervised_learning. 6
- [Wik20g] Wikipedia contributors. Transfer learning — Wikipedia, the free encyclopedia, 2020. [Online; accessed May-2020]. Available from: https://en.wikipedia.org/wiki/Transfer_learning. 1
- [XNZ⁺17] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *CoRR*, abs/1710.01813, 2017. 22
- [YFX⁺18] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *CoRR*, abs/1802.01557, 2018. 23
- [YHC01] A. Steven Younger, Sepp Hochreiter, and Peter R. Conwell. Meta-learning with back-propagation. 2001. 16
- [yl] yaoyao liu. mini-imagenet-tools [online]. Available from: <https://github.com/yaoyao-liu/mini-imagenet-tools>. 10, 48

Appendix A

Appendix

A.1 MAML data preprocessing

To use the omniglot and miniImagenet datasets with the Maml code the following must be done;

A.1.0.1 Omniglot

- Go to the Omniglot dataset at [bre], download images_background and images_evaluation in the "omniglot/python/" directory.
- Put the contents of both folders, without the root folder, in the "data/omniglot" directory of the maml code folder,
- Run the following command, from inside the maml code folder in the command line:

```
$ cd data/  
$ cp -r omniglot/* omniglot_resized/  
$ cd omniglot_resized/  
$ python resize_images.py
```

Following these steps, the Omniglot images are downsampled to 28 x 28.

A.1.0.2 MiniImagenet

- Go to the MiniImagenet dataset at [y1]
- Click on the README.md file, go to the bottom of the page to "Download Processed Images", click on "Download jpg files" and download both the train, val, and test images or image folders and the csv files.
- Since they've already been processed, no need to run the authors' instructed script, just extract the contents, that is, the train, val and test folders and put them, together with the csv files, in the "data/miniImagenet" directory in the maml code folder.

A.2 Running the MAML code

If you have all the required dependencies (specified above) installed on your machine, running and evaluating the MAML source code can be done from the terminal. There are several options on which MAML can be evaluated including few-shots classification on the omniglot or on the miniImagenet dataset. The experimental protocol for few-shot classification is similar to the one proposed by Vinyals et al in [VBL⁺16], which is N-way or class classification with either 1-shot or 5-shot training, and which would require the following default inputs to the code to run:

For 5-way, 1-shot omniglot:


```
$ python main.py --datasource=omniglot --metatraining_iterations=60000
--meta_batch_size=32 --update_batch_size=1 --update_lr=0.4
--num_updates=1 --logdir=logs/omniglot5way/
```

For 20-way, 1-shot omniglot:

```
$ python main.py --datasource=omniglot --metatraining_iterations=60000
--meta_batch_size=16 --update_batch_size=1 --num_classes=20
--update_lr=0.1 --num_updates=5 --logdir=logs/omniglot20way/
```

For 5-way 1-shot mini imagenet:

```
$ python main.py --datasource=miniimagenet --metatraining_iterations=60000
--meta_batch_size=4 --update_batch_size=1 --update_lr=0.01
--num_updates=5 --num_classes=5 --logdir=logs/miniimagenet1shot/
--num_filters=32 --max_pool=True
```

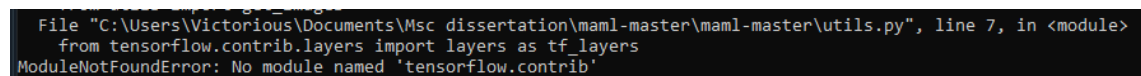
For 5-way 5-shot mini imagenet:

```
$ python main.py --datasource=miniimagenet --metatraining_iterations=60000
--meta_batch_size=4 --update_batch_size=5 --update_lr=0.01
--num_updates=5 --num_classes=5 --logdir=logs/miniimagenet5shot/
--num_filters=32 --max_pool=True
```

A.2.1 Errors and Fixes

Running the MAML code was not without some challenges, errors and compatibility issues, as the MAML source code was written using the old tensorflow syntax (e.g. v.1.*), which was the vogue at the time MAML was proposed, but have either been deprecated or modified in recent versions of tensorflow, that is, the 2.* series.

The way to go about this problem is either to rewrite most of the code using the most recent syntax or downgrade to an older tensorflow version. Below are some of the errors I got while trying to run the code as well as the versions of Python, and tensorflow I was using at the time of the error: After downloading the datasets and doing the necessary preprocessing, the first time I ran the code in the terminal, the error I got was a syntax compatibility issue as shown in Figure 12. At the time, I had python 3.8.2 and tensorflow 2.2 installed which is the latest tensorflow version at the time of the writing of this thesis paper.



```
File "C:\Users\Victorious\Documents\Msc dissertation\maml-master\maml-master\utils.py", line 7, in <module>
    from tensorflow.contrib.layers import layers as tf_layers
ModuleNotFoundError: No module named 'tensorflow.contrib'
```

Figure 12: First error, saying that module tensorflow.contrib.layers could not be found since it has been replaced by tf.keras.layers in the new tensorflow version.

After changing it to "tf.keras import layers" I got another syntax compatibility error somewhere else in the code as illustrated in Figure 13. Which I fixed with "tf.compat.v1.logging.set_verbosity", but then kept getting other errors due to incompatible code. After which I changed my python version to 3.6.0 and tensorflow version to 1.14.0 as shown in Figure 14 and 15.

```

Generating filenames
Traceback (most recent call last):
  File "main.py", line 348, in <module>
    main()
  File "main.py", line 269, in main
    image_tensor, label_tensor = data_generator.make_data_tensor()
  File "C:\Users\Victorious\Documents\Msc dissertation\maml-master\maml-master\data_generator.py", line 104, in make_data_tensor
    filename_queue = tf.train.string_input_producer(tf.convert_to_tensor(all_filenames), shuffle=False)
AttributeError: module 'tensorflow_api.v2.train' has no attribute 'string_input_producer'

```

Figure 13: Second error.

```

C:\>.\venv\Scripts\activate
(venv) C:\>python --version
Python 3.6.0
(venv) C:\>pip install tensorflow==1.14
Collecting tensorflow==1.14

```

Figure 14: I installed python 3.6.0 in order to install tensorflow 1.14

```

Name: tensorflow
Version: 1.14.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\venv\lib\site-packages

```

Figure 15: I installed tensorflow 1.14 in order to see if it will solve the syntax incompatibilities

On downgrading both python and tensorflow versions to the above mentioned versions, I got a whole new type of error, shown in Figure 16, which I fixed with by checking the protobuf version installed on my machine and upgrading it to protobuf version 3.6.0 as portrayed in Figure 17 and Figure 18.

```

File "C:\venv\lib\site-packages\google\protobuf\descriptor.py", line 48, in <module>
    from google.protobuf.pyext import _message
ImportError: DLL load failed: The specified procedure could not be found.

```

Figure 16: Error saying that procedure google.protobuf.pyext._message could not be found.

```

(venv) C:\Users\Victorious\Documents\Msc dissertation\maml-master\maml-master>pip show protobuf
Name: protobuf
Version: 3.12.0
Summary: Protocol Buffers
Home-page: https://developers.google.com/protocol-buffers/
Author: protobuf@googlegroups.com
Author-email: protobuf@googlegroups.com
License: 3-Clause BSD License
Location: c:\venv\lib\site-packages
Requires: setuptools, six
Required-by: tensorflow, tensorboard

```

Figure 17: Checked the protobuf version installed.

```
Installing collected packages: protobuf
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.12.0
    Uninstalling protobuf-3.12.0:
      Successfully uninstalled protobuf-3.12.0
Successfully installed protobuf-3.6.0
```

Figure 18: Upgraded it to protobuf version 3.6.0.

Following this modifications the code was finally able to run, although, not without some deprecation and to be removed in future versions warnings. I first tried the "omniglot 5-way 1-shot" command mentioned in the previous subsection as shown in Figure 19 and Figure 19.

```
(venv) C:\Users\Victorious\Documents\Msc dissertation\maml-master\maml-master>python main.py --datasource=omniglot --metatrain_iterations=60000 --meta_batch_size=32 --update_batch_size=1 --update_lr=0.4 --num_updates=1 --logdir=logs/omniglot5way/
```

Figure 19: Running the code for omniglot 5-way 1-shot experiment.

```
Done initializing, starting training.
Iteration 5000: 0.14375001, 0.95
```

Figure 20: Begining training.