INS(P)ECT: Meta-Learning Algorithms Put Under The Microscope

Eric Chen* and Quentin Wellens†
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
(Dated: May 16, 2019)

This report describes the results of a comparison performed between three meta-learning algorithms - MAML, Reptile, and Insect. The algorithms analyzed here are part of a subclass of meta-learners that seek to learn a set of initial network parameters by training on a subset of tasks. In theory, these initial parameters can be quickly optimized when presented with a new task drawn from a similar distribution (few-shot learning). We implement our own versions of both MAML and Reptile algorithms, and introduce a new baseline meta-learning algorithm of our own - Insect. We compare the performance of each as we vary the number of samples provided from each task during training and testing (K), and the number of SGD steps taken during training. We analyze how well each algorithm performs on a subset of periodic function regression tasks with and without the addition of random noise, and we discuss our thoughts as to how and why each algorithm achieves the specific behavior exhibited.

I. INTRODUCTION

Many recent advances in the field of machine learning leverage vast amounts of data and large computational infrastructures to solve very specific problem scenarios. Recently, an increasing number of papers have investigated how machine learning can artificially encode more human-like cognitive abilities [6] [15] [12]. While many characteristics define human intelligence, one prominent characteristic is our ability to learn quickly from far fewer examples than current machine learning methods. For example, the average person will be able to successfully prepare a meal they have never prepared before the first time they try to make it. This could potentially require some supervision to an extent (be it a recipe, an expert cook, or the latest VR cooking app), but overall we are able to leverage reinforced baselines acquired from past experiences executing similar tasks in order to quickly adapt to a new task. Humans have thus developed a flexible cognitive ability - one that finds the correct balance between leveraging previous experience and existing knowledge while specializing to the required constraints that define a new task.

In an attempt to install a similar notion of cognitive flexibility and generalizability to current machine learning methods, the field of meta-learning offers a set of approaches that attempt to achieve few-shot learning - that is, the ability to learn a new task from limited samples, based off of experience previously gathered training on similar tasks. A subclass of meta-learning algorithms that aims to achieve rapid few-shot optimization on new tasks includes two very popular model-agnostic methods - MAML [7], and Reptile [11]. These algorithms train on example subtasks in order to return a weight initialization that achieves rapid few-shot optimization on similar new task settings.

Both MAML and Reptile have been shown to demonstrate the target behavior they were designed for in a laboratory setting on common simulated tasks (such as image classification, reinforcement learning in simulation [4], and simple regression tasks). Like any algorithm useful in a real world setting however, their sensitivity to individual parameter changes as well as their robustness to the realities of real world tasks such as functional discontinuities and random noise are important features to analyze [1]. Motivated by these challenges potentially encountered in real world contexts, we analyze how the performance of all three algorithms respond to changes in individual parameters, and varying levels of noise at training time. In this paper, we develop our own insights into the nature of this subclass of meta-learning algorithms. We examine the performance of both algorithms - MAML and Reptile - on a subset of periodic function regression tasks. Additionally, we introduce a very simplistic third algorithm - Insect - as an additional baseline comparison with which the performance of the first two more complex algorithms might be studied further. Lastly, we apply MAML to Omniglot image classification and introduce noise.

II. META-LEARNING

We start by providing an overview of the various performance goals and existing methodologies

* ericrc@mit.edu
† qwellens@mit.com

that currently constitute the broader field of meta-learning. We then describe the specific problem formulation encountered with MAML and Reptile, and introduce the notation that will be used throughout this report to describe our findings.

II.1. Overview

Meta-Learning - or learning to learn - can be summarized as a process that enables intelligent systems to draw conclusions from previous learning experiences in order to gain flexibility and insight on how to solve new tasks and improve the learning process [9]. As opposed to many machine learning techniques, in which an engineer designs a learning algorithm that will train a model capable of performing well on a very specific task on a large amount of task specific samples, meta-learning hopes to become good at learning novel tasks after having been exposed to varying but sufficiently similar underlying tasks. This is achieved by gathering *meta data*, information relevant to a distribution of tasks (i.e. a distribution over distributions of examples), rather than to a specific task. If there were a hierarchy of cognitive processes within intelligence, meta-learning would therefore just be one level up the ladder from the classic supervised learning popular in many of today's real world problem scenarios.

Based on the types of meta data being are collected and the ways in which these are leveraged, many techniques used for meta-learning can be subdivided into three main categories: learning from model evaluations, learning from task properties, and learning from prior models [9]. A model evaluation approach to meta-learning has access to a set of tasks and configurations of learning algorithms (learners), such as hyperparameters, network architectures, loss functions, update rules, etc. The meta-learner then proposes a learning algorithm configuration that is estimated to be optimal for learning a given new task, based on prior performance evaluations for that or all other tasks. In the case of learning from task properties, the meta-learner learns some similarity metric to characterize and define tasks [10]. It then makes selections in the learner configuration space by analyzing relationships in the task space. Finally, learning from prior models arises when the meta-learner has access to optimal trained models for certain tasks. It can then provide good learner configurations by inferring properties of the optimal models (which themselves are the result of other learners). For example, it can provide a set of initial model weights that constitute a good starting point for similar tasks. Good initialization should then speed up learning for subsequent tasks. The latter approach forms the core of the analysis presented in this report.

A particular approach to meta-learning is that of few-shot learning, partly popularized by Brenden Lake et al [3]. Few-shot learning aims to minimize the number of samples required to learn a useful model. The problem gives the algorithm only a few training examples for a goal learning task, while large amounts of training data exist for other similar tasks. In the specific case of a regression problem scenario, the model's end goal is to predict a function over a full range, given only K sample points to train on. In the particular case of N-way, K-shot classification, the model's end goal is to discriminate among N different previously unseen classes by having access to only K labeled examples of each of those N classes.

Learning in this manner is closer in nature to the way in which humans learn, and allows machine learning algorithms to rapidly adapt to real world tasks similar to those on which they were meta-trained. This imbues machine learning methods with more generalizability and facilitates their application to real world problem domains. In the remainder of this report, we explore how well this generalizability holds for a small subset of very simple and popular model agnostic meta-learning algorithms.

II.2. Notation and Problem Formulation

In this section, we introduce the notation that we will be using throughout our analysis. We also formally describe supervised few-shot learning in a meta-learning context.

We borrow part of our notation from MIT's 6.036 notes [13], while making the changes necessary to aptly describe the meta-learning problem scenario. We denote models by $h^i(\mathbf{x}^i, \boldsymbol{\theta}^i)$, where h^i is some functional form, $\mathbf{x}^i$ is a vector of input samples, and $\boldsymbol{\theta}^i$ is a set of particular model parameters. The functional form is a specific $h^i \in \mathcal{H}$, where $\mathcal{H}$ is a family of possible models that the learning algorithm can choose from, characterized by a possible set of model parameters Θ, such that $\boldsymbol{\theta}^i \in \Theta$.

All input vectors $\mathbf{x}^i$ are sampled in an independently identically distributed (i.i.d) manner

from a particular distribution whose properties are characterized by a task τ_i. A task τ_i is itself part of a distribution over tasks $\mathcal{T}$, such that $\tau_i \in \mathcal{T}$. Every $\mathbf{x}^i$ sampled from the distribution τ_i has an accompanying ground truth (label) $\mathbf{y}^i$, which $h^i(\mathbf{x}^i, \boldsymbol{\theta}^i)$ aims to approximate.

In terms of terminology for the meta-learning process itself, a meta-learning algorithm tries to find an optimal learning algorithm (which we will also refer to as the learner) that will efficiently produce an optimal initial model $h^i(\mathbf{x}^i, \boldsymbol{\theta}^i)$ for a given task τ_i (i.e. by finding optimal h^i and $\boldsymbol{\theta}^i$ for task τ_i). We assume the meta-learning process consists of meta-training and meta-testing, as is the norm in most recent meta-learning papers [11] [8] [10]. During meta-training, the algorithm has access to a wide range of training tasks $\mathcal{T}_{train}$ from which it samples specific tasks τ_i. In the case of supervised few-shot learning, the algorithm is provided with a mini-training set consisting of K samples $\mathbf{x}^i_0, ..., \mathbf{x}^i_{k-1}$ as well as their corresponding labels $\mathbf{y}^i_0, ..., \mathbf{y}^i_{k-1}$ for each $\tau_i \in \mathcal{T}_{train}$. Additionally, it generally also has access to a mini-testing set for each such $\tau_i \in \mathcal{T}_{train}$, which consists of a number of samples as just defined (typically K of them here as well [8]). Formally, we can thus conclude that $\tau_i = \{\mathcal{D}^i_{mini\_train}, \mathcal{D}^i_{mini\_test}\}$ $= \{(\mathbf{x}^i_0, ., \mathbf{x}^i_{k-1}, \mathbf{y}^i_0, ., \mathbf{y}^i_{k-1}), (\mathbf{x}^i_k, ., \mathbf{x}^i_{2k-1}, \mathbf{y}^i_k, ., \mathbf{y}^i_{2k-1})\}$.

The meta-learning algorithm is evaluated for a specific set of hyperparameters and resulting initialization weights by testing the performance of N models produced by the learner for N tasks sampled from $\mathcal{T}_{test}$ (i.e. one model specialized for each task). Specifically, for each of the N test tasks, the learner uses that task's $\mathcal{D}_{mini\_train}$ to specialize θ_{init} into θ^i, the latter which is then evaluated on that task's $\mathcal{D}_{mini\_test}$. While we would have ideally meta-trained many models for any given configuration of hyperparameters (to obtain c weight initializations per hyperconfiguration and thus test every configuration on cN specialized models), limited resources forced us to only consider at most two different initializations (resulting in $2N$ test models) for any hyperconfiguration. We note that the authors of MAML only considered N test models per configuration in their testing procedure.

We will concentrate our attention on comparing approaches that learn from prior models (see section II.1) and that specifically optimize weight initialization. Additionally, we will restrict the functional form to a particular instance. Using the notation introduced above, the learner cannot optimize over $h^i \in \mathcal{H}$, because h^i is fixed such that $h^i = h$ regardless of the task at stake. However, it aims to optimize the initialization parameters $\boldsymbol{\theta}_{init}$ such that a minimal amount of model updates and samples from a τ_i's $\mathcal{D}_{mini\_train}$ will result in an optimal $h(\mathbf{x}^i, \boldsymbol{\theta}^i) = h(\mathbf{x}^i, f(\boldsymbol{\theta}_{init}))$ for that task.

III. META-LEARNING ALGORITHMS

In this section, we discuss three different meta-learning algorithms (MAML, Reptile, and Insect) that are all concerned with finding optimal initialization of model parameters to enable fast learning.

III.1. MAML

Model-Agnostic Meta-Learning [8] is a meta-learning approach that finds a good weight initialization for any type of model trained with some loss function. To minimize the number of parameter changes while maximizing the performance improvement on new tasks, MAML tries to find model parameters that are sensitive to changes in tasks. The approach achieves this in a model-agnostic and learning-task-agnostic fashion. It can therefore be used for regression, classification, and reinforcement learning models.

MAML consists of two main parts: an outer loop and an inner loop. The outer loop is the one that corresponds to the meta-learning process, to find the best parameters $\boldsymbol{\theta}_{init}$. The inner loop is essentially a run of how the initialized model would adapt to a specific task. Namely, in the inner loop, the model's parameters $\boldsymbol{\theta}$ become specialized ('fast') parameters $\boldsymbol{\theta}^i$ through the following stochastic gradient descent update rule:

$$\boldsymbol{\theta}^i = u(\boldsymbol{\theta}_{init}) = \boldsymbol{\theta}_{init} - \alpha \nabla_{\theta_{init}} \mathcal{L}_{\mathcal{D}^i_{mini\_train}}(h(\mathbf{x}^i, \boldsymbol{\theta}_{init}))$$

where u is the SGD update function, α is the inner learning rate and $\mathcal{L}$ is some loss function. This update rule can be applied a number of times in the hope to get more specialized and accurate fast parameters for the task. In the outer loop, the model parameters $\boldsymbol{\theta}_{init}$ are updated such as to optimize performance of the just obtained fast parameters $\boldsymbol{\theta}^i$ on their respective task's mini-test set. Therefore:

$$\boldsymbol{\theta}_{init} \leftarrow \boldsymbol{\theta}_{init} - \beta \nabla_{\theta_{init}} \mathcal{L}_{\mathcal{D}^i_{mini\_test}}(h(\mathbf{x}^i, \boldsymbol{\theta}^i))$$

where β is the outer learning rate and $\mathcal{L}$ is again some loss function. This update rule thus requires higher-order derivatives to be computed. In this

way, the paper claims that this approach "optimizes model parameters such that one or a small number of gradient steps on a new task will produce maximally effective behavior on that task."

III.2. Reptile

Reptile is a meta-learning algorithm that like MAML seeks to produce a good network weight initialization such that the model is able to quickly achieve optimal weights on new tasks [11]. Like MAML, this methodology is general enough to be applied to any problem scenario that is solve-able with standard SGD methods. However, the algorithm differs from MAML in that it does not require computing a double gradient when updating the target weights in the outer loop of the algorithm. According to the authors of the original MAML paper [7], the double gradient computation included in MAML makes the algorithm approximately 33% slower than a similar first order method. The authors of the MAML paper thus also present an approximate first order MAML methodology called FOMAML. Instead of directly computing the second derivative, it is Reptiles first order nature that makes it less costly than MAML in terms of both memory and computational resources. Additionally, Reptile differs from MAML in that it doesn't require mini-testing data for each task. Depending on the problem scenario, this type of structure could be a more fitting way of meta-learning a good weight initialization.

The basic outline of the algorithm is as follows. Reptile starts by sampling a set of tasks τ from a distribution of tasks. For each individual sampled task τ_i, Reptile finds the optimal weights $\boldsymbol{\theta}^i$ associated with that task. The weight initialization $\boldsymbol{\theta}_{init}$ ultimately produced by the algorithm is updated after computing $\boldsymbol{\theta}_i$ for each task by simply moving the $\boldsymbol{\theta}_{init}$ towards the $\boldsymbol{\theta}_i$ using the following update.

$$\boldsymbol{\theta}_{init} = \beta(\boldsymbol{\theta}^i - \boldsymbol{\theta}_{init})$$

This step means Reptile avoids computing a second order derivative, and does not need both training and test data sets for each sub-task. An interesting note to keep in mind is that Reptile requires that the number of inner updates be bigger than 1 in order to provide any learning capability past joint-training. A single fast update corresponds to training the model on an average of all tasks in the task distribution.

III.3. Insect

In order to further explore the extent of simplicity achievable with meta-learning, we also designed and implemented a simple meta-learning method of our own; Insect. If there is one objective this algorithm does not aim to achieve, that objective would be to accomplish state-of-the-art performance resulting from carefully crafted meta-learning insights. Rather, Insect was specifically designed as a baseline of what the simplest weight initialization meta-learning algorithm could be, such as to put the performance of other algorithms in perspective. We hope that the chosen name accurately reflects this mission statement (thanks to Prof. Lozano-Prez for the inspiration!).

Before discussing the specifics of Insect, we explain the high-level reasoning behind the design choice of our algorithm. This choice was inspired by basic concepts in statistics and signal processing and we will therefore borrow the associated terminology from [14]. As explained in section II, tasks are sampled from a distribution of tasks. Thus, the distribution of tasks $\mathcal{T}$ (see II.2) can be seen as a random process T, of which a realization is then referred to as a specific task. We now make a parallel between the random process T consisting of the tasks, and the random process W consisting of the optimal parameter weights for such tasks. By analogy, the optimal weights $\boldsymbol{\theta}^i$ for a task is a realization of this latter random process W. One way to phrase the meta-learning algorithm's objective for few-shot learning, is for it to find the minimum mean square error estimator (MMSE estimator) associated with random process W, given M measurements. We know from [14] that such estimator takes the form $E[W|M]$ (where for simplicity M refers to both the amount of measurements and the measurements themselves). However, the simplest meta-learning algorithm is one that does not actually learn any meta-data. For such a baseline algorithm, M and W will appear independent because it does not actually develop insights into meta-data. The estimator therefore reduces to $E[W|M] = E[W]$, namely the average of the random process of optimal parameter weights.

This leads us to the very simple and intuitive conclusion that the output of our baseline meta-learning algorithm, that is $\boldsymbol{\theta}_{init}$, should simply be the average of all the optimal $\boldsymbol{\theta}^i$ encountered during meta-training. Therefore:

$$\boldsymbol{\theta}_{init} = E[\boldsymbol{\Theta}] = \sum_{i=0}^{N-1} \frac{\boldsymbol{\theta}^i}{N}$$

Algorithm 1 Insect

randomly initialize θ
while not done **do**
  sample N tasks τ_i from $\mathcal{T}_{train}$
  for all τ_i **do**
    find θ^i initialized from θ for τ_i
    (e.g. by using S steps of SGD)
  $\theta = \sum_{i=0}^{N-1} \frac{\theta^i}{N}$
return θ

where N is a batch size, rather than the size of $\mathcal{T}_{train}$, because we apply this repeatedly in batches in order to smooth the learning process and thus reduce the total number of gradient updates needed (the behavior would likely significantly change without batching). The pseudo-code is described in Algorithm 1.

We would like to point out that current meta-learning papers such as [8] have generally used transfer learning as a baseline, in which the output of the model for a specific task tends to be the average over all the tasks, i.e. $h(\mathbf{x}, \boldsymbol{\theta}_{init}) = E[\boldsymbol{\mathcal{T}}(\mathbf{x})]$. This arises partly because, contrary to meta-learning methods, transfer learning is not explicitly aware that there is a distribution of distributions, rather than a single distribution. Additionally, transfer learning is a learning algorithm, whereas our goal is to analyze meta-learning algorithms, i.e. learning algorithms that learn learning algorithms. Therefore, while we believe that such a transfer learning baseline is helpful to establish and confirm the need for meta-learning, it is not in itself a meta-learning baseline.

It is also worth noting that, despite the simplicity of our algorithm, the updates applied to θ in Insect in batches share mathematical similarities with the updates that arise in Reptile (and MAML to some lesser extent). We leave this theoretical analysis as potential future work.

IV. METHODOLOGY

In this section, we describe the methodology used to run multiple analyses to compare performance of the three meta-learning algorithms on various metrics. We reproduced and implemented all algorithms ourselves using PyTorch [5]. We implemented them according to the descriptions provided in their respective papers while resolving algorithmic ambiguities left out in the papers by inspecting the author's code for those algorithms. We resolved specific PyTorch implementation difficulties by inspecting other implementations or public code requiring similar mathematical operations available on PyTorch's community forum. Our code is available at https://github.com/qwellens/meta_learning_project.

Because we analyze meta-learning algorithms that optimize for good initial model weights, we fix the neural network architectures used throughout our experiments. Put another way, we are interested in evaluating the estimation error rather than the structural error of our networks. For regression tasks, we use a neural network with 2 hidden layers containing 40 neurons each, followed by ReLu nonlinearities. The output layer is an affine layer with a scalar as output. This is the exact same baseline architecture used in the original MAML paper. [8]

While we will be varying a certain number of hyperparameters, there are some that we keep fixed for all experiments. First, we go over fixed parameters used in meta-training. Specifically, for both MAML and Reptile, the outer learning rate is set to $\beta = 0.001$ and we use the Adam optimizer. [2] For all three evaluated algorithms, the inner learning rate is set to $\alpha = 0.01$. Note that we generally always favor a larger inner learning rate because this corresponds to the fast weights, which should indicate a direction for a specific task but do not need to be perfect. Indeed, these do not contribute as much to the weights for other tasks, as their effect is tampered and overshadowed by the effect of the outer update rule. The outer learning rate, however, is more directly at the core of the meta-learning optimization process, and directly influences all subsequent fast weights. It therefore needs to be tuned more carefully, and requires a lower learning rate so as to approach the optimum with lower risk of overshooting or reaching a suboptimal value. Most hyperparameter values were chosen based off of the original paper implementaions, in order to maximize the change of the approach demonstrating some measure of success, and so as to maintain enough specificity to explore and build on the conclusions of these previous works. For Insect, we used an averager of size $N = 50$. Wherever a loss function is needed for regression, we use a mean squared error loss. When applying MAML to the omniglot dataset, the cross-entropy error between the ground truth labeled class and the models prediction is used as the loss function.

During meta-testing, we test every trained model

(i.e. an initialization model trained using a certain set of meta-training parameters) on 600 sampled tasks from $\mathcal{T}_{test}$. For each such test task τ_i, the model is fine-tuned using $\mathcal{D}^i_{mini\_train}$ of size K with S steps of SGD on MSE loss or cross-entropy loss for regression and classification respectively. The learning rate used for those SGD updates is fixed at $\alpha = 0.01$, reflecting the value used for the inner learning rate during meta-training. For regression, the performance on a single test task is then computed as the mean squared error on a $\mathcal{D}_{mini\_test}$ of size 100, where the $\mathbf{x}^i$ are equally spaced between $[-5, 5]$. For classification, binary accuracy was computed for every task's samples and averaged over 1000 test tasks (each with N categories, K shots).

IV.1. Functional Forms

In order replicate the results of the original papers for both MAML and Reptile, we trained and tested each algorithm on the same sinusoidal regression problem scenario chosen as one of the task distributions in Finn et al. [7] and Nichol et al [11]. In addition to this, two other discontinuous periodic functional forms - the square wave and the sawtooth wave - were chosen to train and evaluate each algorithm. Given that real world scenarios can often contain data with discontinuities and various other irregularities (such as sharp edges and non-smooth features), we were curious to see how each meta-learning algorithm might handle a less well behaved regression problem. For this project, only periodic functional forms were chosen in order to more easily compare and contrast each algorithms behavior on top of the foundation laid by the sinusoid regression task outlined in the original papers.

For each periodic functional form, the phase ϕ and scaling amplitude A were the only two variables used to define a unique task. Varying frequency is not reasonable for K-shot learning due to the aliasing properties of discrete periodic functions. As such, varying frequency no longer produces sets of unique tasks and so frequency is kept a constant 1 throughout all trials. Using the parameters defined, the functional form for a sinusoid task is generated by sampling a random value of A and ϕ and plugging them into the following equation:

$$f(x) = A(\sin[x - \phi])$$

The next periodic function was a standard square wave. The square wave was particularly interesting due to its rapidly changing edges - whether or not a meta-learner might be able to closely approximate these could be an indicator of whether or not this meta-learner is robust to real world applications. The square wave was implemented using the scipy.signal python module, and the equation used to define it was as follows:

$$f(x) = sgn(Asin(2\pi f(x - \phi)))$$

The last periodic function used was the sawtooth wave. The sawtooth wave was also implemented using the scipy.signal module, which uses the following formula to generate the function:

$$f(x) = A((x - \phi) - |(x - \phi)|)$$

It is important to note that a sawtooth wave was used instead of a triangle wave, in order to investigate how well all the algorithms were able to approximate the vertical edges absent in the latter but present in the former. When using the scipy python package, this is equivalent to passing in a width of 1 to produce a rising ramp.

IV.2. K-shot Relationship

In terms of important learning parameters present in all three algorithms, the number of samples K used to train and test on each subtask is of particular interest. In terms of the performance advantages sought by these three meta-learning approaches as described previously, achieving low numbers on the number of samples required to learn a good representation is a central goal. As such, we deemed it important to compare each algorithms robustness to noise while varying the value of this parameter. More detail will be covered in the results section, but in terms of values K was set to 2, 5, and 10 for each of the three functional forms, under all three functional forms. These values were altered at both training and testing time consistently, in order to qualify both meta-learners as K-shot learners for the same value of K and accurately measure their performance as such.

It is important to notice that the MAML algorithm uses the same amount of data as Reptile in this scenario, except it visits each sample twice during the learning process (due to the training/testing split used on each sub task). By halving the number of meta-training iterations from 100,000 to 50,000 for MAML, we make sure that both algorithms utilize the same quantity of data for training and the

only variation occur as a result of the algorithms individual approaches to learning. For these trials, the learning rates for both the inner and outer loop remained the same, and the number of SGD steps used to test and train were both set to 10. A model was trained on each functional form, for each K value listed above.

IV.3. Meta-Training Inner SGD Steps Variation-

The second parameter of interest common to all three algorithms is the number of stochastic gradient descent steps used in the inner loop of meta-training for each task τ_i. This value has an effect on how well the fast weights fit to each specific task during meta-training, and can carry through to the resulting initial weight set returned θ_{init}, thereby changing the algorithm's generalizability depending on how many tasks are used to train.

In order to test how sensitive each algorithm's performance is to changes in the number of SGD steps, we conducted trials using SGD step values of 1, 2, 5, and 10. These values were only varied during meta-training. During meta-testing, 10 SGD steps were consistently taken to obtain task-specific weights θ_i from θ_{init}. These steps were taken in order to ensure that each algorithm's meta-training was affected by the choice of SGD step, but that at meta-testing time each approach had the opportunity to take 10 SGD steps to show maximal performance (conditioned upon the algorithm having learned not to overfit to a particular task's $\mathcal{D}^i_{mini\_train}$). However, we did evaluate performance on $\mathcal{D}^i_{mini\_test}$ in between each of those 10 SGD steps in meta-testing, such as to obtain an accurate picture of accuracy as the weights specialize to the task. Additionally, a value of $K = 10$ was set consistently during both meta-training and meta-testing. All other hyperparameters were set according to IV.

IV.4. Robustness to Noise

In order to reflect on each algorithm's robustness to noise in practice, we varied each functional form during meta-testing by randomly adding noise to the test task's $\mathcal{D}^i_{mini\_train}$, while still evaluating on a noise-free $\mathcal{D}^i_{mini\_test}$.

For regression tasks, we added Gaussian noise to the labels (i.e. to the to be predicted values) $\mathbf{y}^i$ for each $\mathbf{x}^i$ in $\mathcal{D}^i_{mini\_train}$. The Gaussian noise has mean 0 for all our experiments, but we tried multiple standard deviation values. Specifically, we ran experiments with standard deviations set at one of 0%, 5%, and 10% of the amplitude of the task wave being regressed on. We were hesitant to add any further noise, in order to retain the underlying functional form intact. Too much noise could lead to ambiguity regarding what functional form constitutes a proper fit result for the algorithm during testing. These three noise settings were evaluated for hyperparameter configuration outlined above. Noise was not applied during meta-training in order to more accurately replicate a real world scenario in which an agent is trained in a practically noise-free lab environment (or in simulation) before being deployed in a real world setting, where access to noise-free data is very sparse.

IV.5. Images

One appealing aspect of all three algorithms analyzed is their model agnostic nature. Each algorithm is generally applicable to any machine learning method that uses SGD, and as such can be used in theory to learn good initial weights for a wide variety of problem scenarios dealing with very different underlying data sets. In an attempt to expand the nature of the tasks on which we perform our analysis, we tested our MAML and Reptile implementations on the Omniglot image data set. In this new problem scenario, the task is now to correctly classify an image as one of N classes using only K samples drawn from each of the classes.

In order to limit training time to a reasonable margin, we trained MAML and Reptile for 5-way classification with $K = 1$ (one-shot learning). Additionally, as done in the original paper [7] the base model contains a 3x3 convolution with 64 filters, a batch normalization layer, a ReLU layer, a strided convolutional layer and a final softmax layer. Each image was downsampled to dimensions of 28x28 pixels in order to achieve reasonable processing time. The same training/testing splits of 1200 characters for training and the remainder for testing was used to obtain comparable results.

Following the motivation elaborated above, noise was also applied to images in order to see the effect it would have on the meta-learning classification performance. Specifically, salt an pepper noise was applied to the images. Salt and pepper noise occurs when random pixels are flipped to either black and white binary values. This kind of noise is a realistic example of the kind of interference present in real world data, and can accurately reflect how much

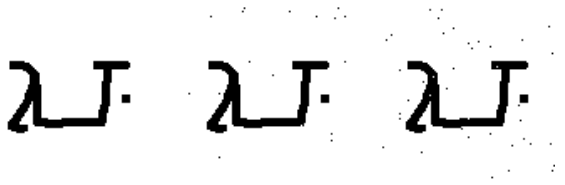

FIG. 1. Salt and Pepper noise added to an omniglot character (noise values from left to right: 0.0, 0.003, 0.007).

noise there could be. Salt and pepper noise levels are determined by varying two parameters - one governing the tradeoff between "pepper" (black pixels) and "salt" (white pixels) randomly added, and another governing the overall amount of both. For our trials, we used a tradeoff factor of 0.5 to equally prioritize pixel omissions and additions. We used overall salt and pepper amounts of 0, 0.003, and 0.007.

Each of noise trial was conducted with a K value of 10, with inner SGD steps also set to 10. The inner and outer loop learning rates remained the same as in previous trials.

V. RESULTS AND ANALYSIS

In this section, we report the results that we obtained for the experimental settings outlined above. For models that were trained on regression tasks, we report mean square errors at meta-test time in percentage of the wave's amplitude. We chose to normalize the results in that way because the amplitudes of the waves vary from task to task, such that one same absolute error value could indicate very strong performance on one task, but very poor performance on a different task. As MSE errors are averaged over 600 test tasks, we deemed such normalization to be necessary to provide an accurate picture of the algorithms' relative performances. All bars surrounding line plots indicate 95% confidence intervals of the value being represented.

V.1. Functional Forms

Before comparing how the various algorithms perform against each other, we analyze the relative difficulty of our three task distributions: sines, sawtooth waves, and square waves. As can be seen in table I, all three algorithms perform the best on sine wave regression. Performance on sawtooth waves and square waves is consistently worse by about 40% and 80% respectively, compared to sine waves. This makes intuitive sense as sawtooth waves and square waves have respectively 1 and 2 sharp discontinuities. Additionally, while square waves spend half the time at one value and half the time at the other, sawtooth waves vary linearly in between those extrema, making it easier to minimize the MSE. Overall, it thus appears that it is indeed harder for meta-learning algorithms to learn general task meta-data when such tasks include discontinuities.

Algorithm	Sine	Sawtooth	Square
MAML	20.1 [18.4, 21.9]	65.7 [62.0, 69.4]	108 [100.9, 114.7]
Reptile	5.7 [4.9, 6.6]	44.4 [41.8, 47.0]	90.5 [85.8, 95.2]
Insect	23.5 [19.8, 27.2]	57.3 [53.4, 61.2]	92.5 [87.5, 97.6]

TABLE I. MSE [95% CI] performance of each meta-learning algorithm on sine, sawtooth and square waves with fixed hyperparameters for K=10. Sines are the easiest to predict, followed by sawtooth waves, followed by square waves.

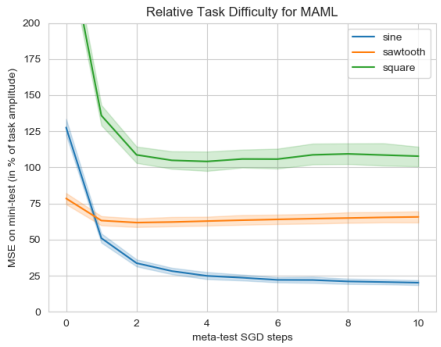

FIG. 3. MSE [95% CI] performance of MAML on sine waves, sawtooth waves, square waves for K=10 at meta-test time.

The MSE errors are pretty high for sawtooth and square waves, and might thus make it seem like meta-learning algorithms do not learn much at all. However, it is clear from figure 7 that such a hasty conclusion should not be made. Indeed, one can notice that all three algorithms learn the general trends for each category of tasks, such as periodicities and discontinuities. Therefore, even though they might not be the fastest/data-efficient or most accurate few-shot learners, they do certainly learn meta-data relevant to the tasks.

As can be seen in figures 2 and 3, it is interesting to note that the weight initializations $\boldsymbol{\theta}_{init}$ for all three algorithms perform best on sawtooth waves,

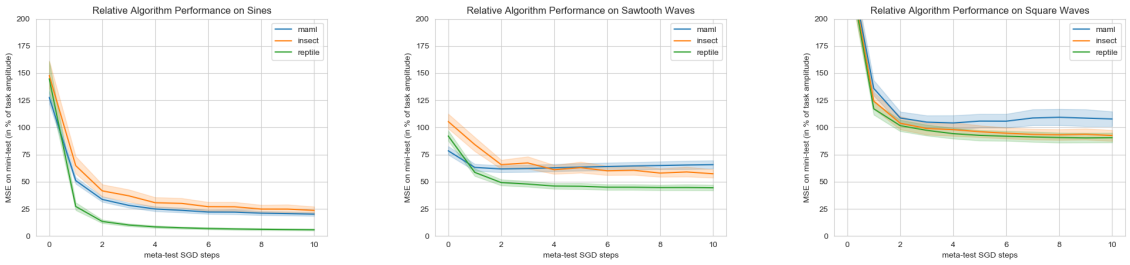

FIG. 2. Performance of the three meta-learning algorithms on all three functional forms (K=10, meta-train inner SGD steps = 10)

.

even though specialized weights (after 10 sgd steps) eventually perform best for sine waves. After only one or two sgd steps, the specialized weights start to perform relatively better on sine waves. Similarly, while the performance improvements are quite large after one gradient step for both sines and squares, weight specialization through gradient updates does not seem to improve performance on sawtooth waves. This seems to indicate that the optimal weights for many different sawtooth tasks are close to each other, while the opposite holds for the two other functional forms. Intuitively, it therefore seems like sawtooth waves are inherently more similar to one another than sines or square waves.

We notice in figure 2 and table I that Reptile outperforms the two other meta-learning algorithms, which achieve similar performance, on all three functional forms. The difference in performance is the strongest for sine waves, which could be explained by the fact that sine waves are the easiest waves to predict, such that differences in algorithmic performances are exacerbated. Additionally, note that the higher performance achieved by Reptile is relatively consistent throughout the weight specialization process. We therefore conclude that Reptile generally does a better job at learning meta-data, resulting in better weight initializations regardless of the task distribution at stake.

V.2. K-Shot Parameter Variation

In order to study the individual affect of the K parameter, each algorithm was trained on each functional form with varying values of K. These trials held the number of SGD steps fixed at 10, and give insight into how robust each method is to fluctuations in the number of samples it receives during training - a key value key to some of the primary motivations behind meta-learning. When we conducted our trials, the following results were obtained for various values of K on each of the three periodic functional form regressions problems described above (Fig. 4).

Although the time required to conduct extensive trials and gather a larger number of K value data points was limited, the current results already highlight a few potentially interesting aspects of each meta-learning approach. In accordance with the difficulty of each task observed in Fig. 3, the range of the mean-squared error values generally present within each individual plot tend to be higher for the square wave regression task, and lower for the sinusoid and sawtooth waves. Across each specific regression task, each algorithms performance improves with larger K. Intuitively this makes sense in that given more samples to train and test on at each subtask, each algorithm makes better informed updates to the overall initialization weights returned at the end of the process. Although exact general trends are hard to determine without further data, it seems as though Insect and Reptile follow similar performance trends on both the sinusoid and sawtooth regression tasks, with Insect always slightly under-performing Reptile. On these the square wave regression task shows the general downward trend demonstrated by each of the three learning algorithms, but given the difficulty of the task it seems as though higher values of K would be required to make any decisive conclusions on the nature of distinct behavioral trends exhibited by each approach.

One potentially interesting aspect of the plots above lies in the manner in which MAML seems to perform better than Insect and equally as well as Reptile at lower K values, but performance improves less drastically than these two as the K value increases. Of course, it is difficult to tell without

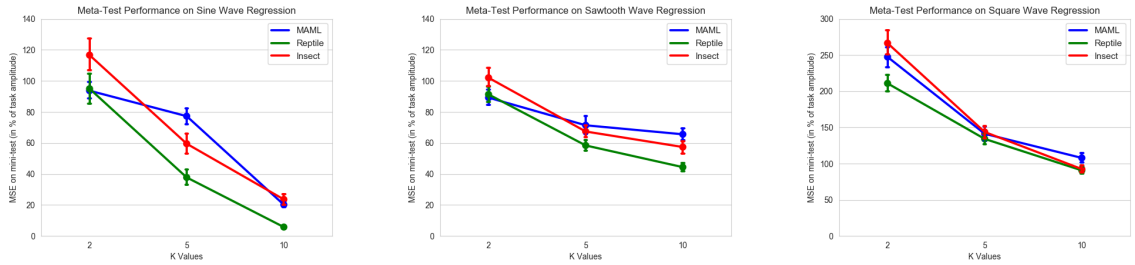

FIG. 4. Plots depicting the effects that variations in the number of samples K used to train and test each algorithm have on the overall MSE performance of each approach on each of the three periodic functional forms.

significantly more trial data. This could be a potential avenue for future work - exploring a larger gamut of K values and conducting repeated trials at each could yield more insight into the exact nature of this behavior, but the clues exhibited here are interesting to contemplate. At the moment, what we can infer from these plots is that each approach exhibits approximately the same changes in behavior and accuracy under variation in K.

V.3. Meta-Training Inner SGD Steps Variation

When changing the number of inner SGD steps during meta-training, we observe significant difference in behavior between Reptile and Insect versus MAML. From figure 5, it appears that for a low number of inner SGD steps, MAML significantly outperforms Reptile and Insect. Indeed, we notice that Reptile and Insect with 10 inner SGD steps perform similarly to MAML with only 1 SGD step. MAML's performance does not vary much with the number of inner SGD steps, a behavior noted in the original paper [8]. Due to limits on computational time available, computing a large enough number of such MAML values for SGD greater than 1 was difficult to achieve for this report. As such, the value of MAML at SGD 1 was extended across 2, 5, and 10 based on an expectation that the behavior described in the original paper holds. Given more time and computational ability, more accurately obtained values of MAML should be extracted by varying the number of SGD steps taken during training, just as was done for both Reptile and Insect.

This significant difference in behavior can be explained by the fact that the three algorithms make different assumptions about the fast weights (the results of the inner SGD steps). Indeed, Reptile and Insect both assume that the fast weights are the optimal weights for the inner loop's task. Naturally, a few more inner SGD steps will push the fast weights closer to their optimal value, which in turn improves how good the meta weights are. However, do note that it seems like Insect already starts overfitting to the task τ_i's $\mathcal{D}^i_{mini\_train}$ after about 5 inner SGD steps, as evident in the upward sloping curve for more than 5 SGD steps in figure 5. MAML, on the other hand, does not make such a strong assumption about the fast weigths. Indeed, it just assumes that the fast weights indicate a general direction in which to orient the weights. This assumption is reflected in the algorithm's use of $\mathcal{D}^i_{mini\_test}$ to compute the loss of the fast weights, while using this loss to update the meta weights. If the fast weights are not optimal, the meta weights thus have the opportunity to account for that through the higher-order meta gradient update. This explains why MAML with 1 SGD step already performs just as well as Reptile or Insect with 5+ steps. Although performance is similar across the board, MAML remains much less sensitive to the SGD parameter value whereas Insect and Reptile show greater performance sensitivity to the SGD parameter. Whether or not fast weights are assumed to be optimal therefore seems to determine how many SGD steps should be taken.

FIG. 5. MSE [95% CI] A higher number of inner SGD updates during meta-training (to obtain fast weights) is important for Reptile and Insect, but not so much for MAML (K=10).

V.4. Robustness to Noise

In terms of each algorithms robustness to noise, the results shown in Fig. 6 seem to indicate that each algorithm exhibits a fair amount of robustness to small perturbations in the regression objective applied at meta-test time.

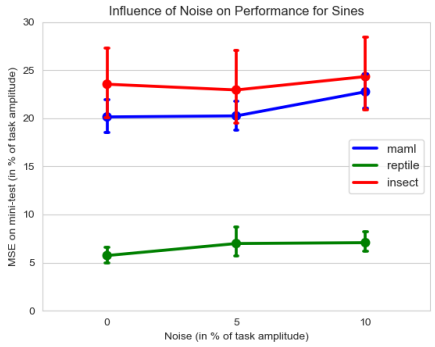

FIG. 6. A plot denoting each algorithms performance under various levels of noise applied at meta-test time.

Although performance drops slightly across the board with more added noise as expected, the generally horizontal nature of the plots seem to indicate a certain level of performance stability even in the presence of random noise. Of course, applying higher levels of noise would most likely give further insight into the nature of each algorithms stability. However, adding more noise doesn't make much sense in that large perturbation values applied to the periodic functions examined in this report have the effect of losing the underlying functional form and potentially aliasing the underlying form with many potential possibilities. A sine wave with amplitude of 3 could easily be confused with a sine wave of amplitude 5 under 3 or 4 units of added random noise. Just as in Fig. 2 on the sinusoid regression task, Reptile outperforms both Insect and MAML.

V.5. Noisy Omniglot Classification Results

The results obtained on the 5-way 1-shot omniglot dataset classification problem are listed in Fig. II. Both Reptile and MAML performed well on the omniglot dataset, and remained fairly robust to salt and pepper noise additions. From the results tabulated however, it seems as though MAML experiences a less sharp decreas in accuracy compared to Reptile. Although it would make sense that the performance and robustness to noise of each algorithm might be problem scenario dependent, it is interesting to see such a difference in accuracy drop rate for Omniglot classification. This could potentially be due to the fact that Reptiles inner loop optimality assumption does not hold as well on noisy image data, and in this case MAML achieves more stability to noise due to its second order derivative. More trials and tests would have to be conducted in order to get a better idea of this, but it was nonetheless interesting to test both approaches on a different problem scenario.

Algorithm	noise 0	noise 0.003	noise 0.007
MAML	98.0 [97.5, 98.6]	96.8 [96.1, 97.5]	95.9 [95.1, 96.7]
Reptile	97.7 [97.1, 98.3]	94.2 [93.5, 94.8]	89.7 [88.2, 91]

TABLE II. Accuracy in % [95% CI] performance of each meta-learning algorithm on omniglot dataset with fixed hyperparameters for K=5, N=1. The noise values correspond to salt and pepper noise levels.

VI. CONCLUSIONS

In terms of real world applicability and generalizability, it seems as though MAML, Reptile and Insect are all robust enough to withstand tempered amounts of parameter variations and noise as well as misbehaved underlying data. Performance wise, we found that Reptile outperforms MAML even though the former is a first order method while the second computes a second order gradient. Intrigued by the appeal of simplicity, we came up with Insect - even simpler than Reptile, it is capable of achieving performance comparable with both MAML and Reptile. In terms of K variation, each algorithm exhibits fairly stable and consistent behavior - an increase in performance with an increase in K. When analyzing the effect of SGD steps taken during training, MAML demonstrates much more stable behavior due to the weaker assumption it makes on the optimality of the fast changing inner loop weights. Insect on the other hand exhibits overfitting at high values of SGD, and optimal general performance at intermediate levels of SGD weights. All in all, this subclass of model-agnostic few-shot meta-learning algorithms seem to demonstrate a fair amount of stability and consistent behavior in the face of parameter variations. In terms of future work, it could be interesting to explore exactly why such simplifications as the ones made in Reptile and Insect achieve comparably high performance to computationally more expensive algorithms such as MAML. Additionally, the exact nature of each approaches stability in the face of noise and regression data could be further tested by examining non-periodic functional forms, as well as real world time series data. In terms of the work distribution for this project, Quentin implemented Reptile and Insect and produced scripts

for the different functional forms as well as for the influence of inner SGD steps for each approach. Eric implemented MAML and wrote the scripts to analyze the K variations and the robustness to noise for each meta-learning approach. Both members contributed equally in terms of conducting the many trials associated with the project, performing the overall analysis, and with any decision making and writing involved.

ACKNOWLEDGMENTS

We gracefully thank Prof. Kaelbling, Prof. Lozano-Prez and Prof. Isola for their continued support and guidance throughout the course of this semester.

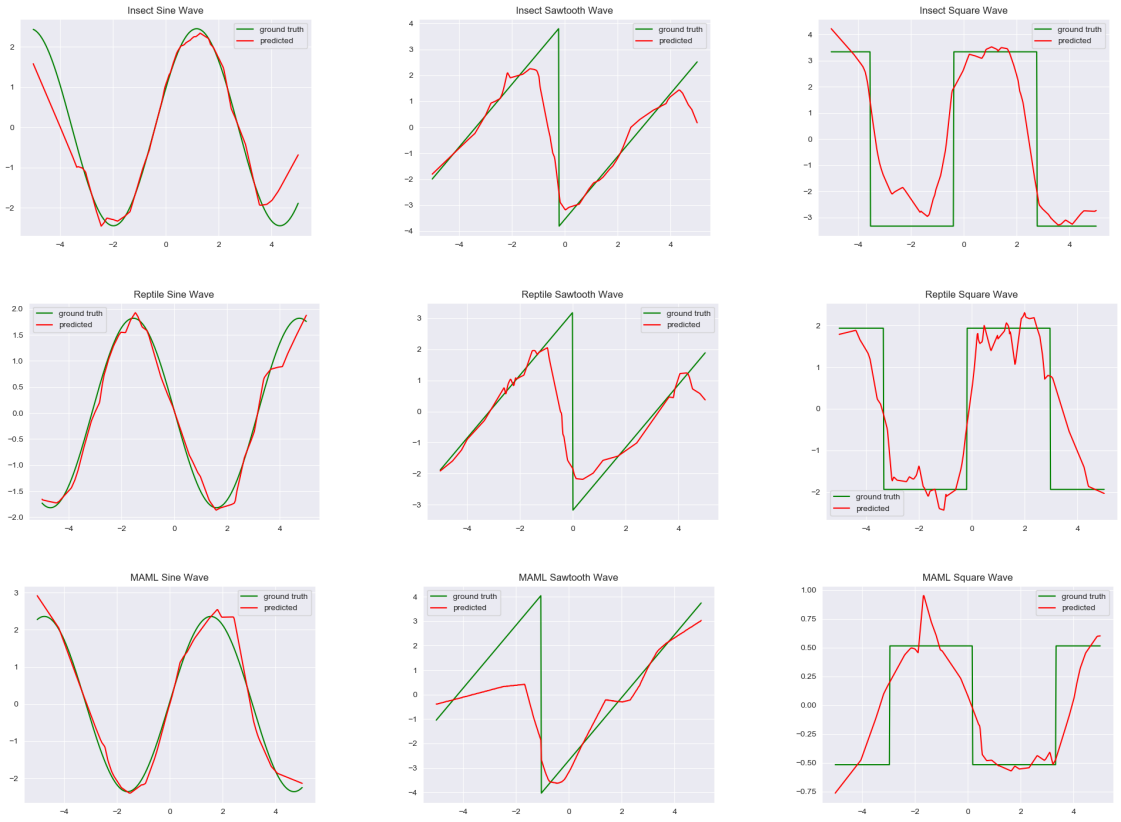

FIG. 7. Typical predictions made by Insect, Reptile, and MAML for K=10, meta-train and meta-test SGD steps=10.

[1] Wagstaff, Kiri(2012): *Machine learning that matters.*

[2] Kingma, Diederik / Ba, Jimmy(2015): *Adam A method for stochastic optimization.*

[3] Lake, Brenden(2015): *Human-level concept learning through probabilistic program induction*, 1: 1332.

[4] Brockman, Greg / Cheung, Vicki / Pettersson, Ludwig / Schneider, Jonas / Schulman, John / Tang, Jie / Zaremba, Wojciech(2016): *Openai gym.*

[5] Paszke, Adam u.a.(2017): *Automatic differentiation in PyTorch.*

[6] Lake, Brenden M / Ullman, Tomer D / Tenenbaum, Joshua B / Gershman, Samuel J(2017): *Building machines that learn and think like people.*

[7] Finn, Chelsea(2017): *Learning to Learn.*

[8] Finn, Chelsea / Abbeel, Pieter / Levine, Sergey(2017): *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.*

[9] Vanschoren, Joaquin(2018): *Meta-Learning: A Survey.*

[10] Alet, Ferran / Lozano Pérez, Tomás / Kaelbling, Leslie P(2018): *Modular meta-learning.*

[11] Nichol, Alex / Achiam, Joshua / Schulman, John(2018): *On first-order meta-learning algorithms.*

[12] Ha, David / Schmidhuber, Jürgen (2018): *Recurrent world models facilitate policy evolution.* : 2450–2462.

[13] Massachusetts Institute of Technology, MIT (2018): *6.036 Lecture Notes.* .

[14] Oppenheim, Alan V. / Verghese, George C. (2015): *Signals, Systems and Inference.* , Pearson Education.

[15] Pathak, Deepak / Agrawal, Pulkit / Efros, Alexei A / Darrell, Trevor (2017): *Curiosity-driven exploration by self-supervised prediction.* : 16–17.