10 Oct 2018

Meta-Learning Related Tasks With Recurrent Networks: Optimization And Generalization

Thy Nguyen

A. Steven Younger

Emmett Redd

Tayo Obafemi-Ajayi
Missouri University of Science and Technology, towd2@mst.edu

Recommended Citation

T. Nguyen et al., "Meta-Learning Related Tasks With Recurrent Networks: Optimization And Generalization," *Proceedings of the International Joint Conference on Neural Networks*, article no. 8489583, Institute of Electrical and Electronics Engineers, Oct 2018.
The definitive version is available at https://doi.org/10.1109/IJCNN.2018.8489583

Meta-Learning Related Tasks with Recurrent Networks: Optimization and Generalization

Thy Nguyen
Computer Science Dept.
Missouri State University
Springfield, MO
Nguyen318@live.missouristate.edu

A. Steven Younger
Department of Physics, Astronomy, and Materials Science
Missouri State University
Springfield, MO
SteveYounger@missouristate.edu

Emmett Redd
Department of Physics, Astronomy, and Materials Science
Missouri State University
Springfield, MO
EmmettRedd@missouristate.edu

Tayo Obafemi-Ajayi
Engineering Program
Missouri State University
Springfield, MO
TayoObafemiajayi@missouristate.edu

***Abstract*—There have been recent interest in meta-learning systems: i.e. networks that are trained to learn across multiple tasks. This paper focuses on optimization and generalization of a meta-learning system based on recurrent networks. The optimization investigates the influence of diverse structures and parameters on its performance. We demonstrate the generalization (robustness) of our meta-learning system to learn across multiple tasks including tasks unseen during the meta-training phase. We introduce a meta-cost function (Mean Squared Fair Error) that enhances the performance of the system by not penalizing it during transitions to learning a new task. Evaluation results are presented for Boolean and quadratic functions datasets. The best performance is obtained using a Long Short-Term Memory (LSTM) topology without a forget gate and with a clipped memory cell. The results demonstrate i) the impact of different LSTM architectures, parameters, and error functions on the meta-learning process; ii) that the mean squared fair error function does improve performance for best learning; and iii) the robustness of our meta-learning framework as it generalizes well when tested on tasks unseen during meta-training. Comparison between No-Forget-Gate LSTM and Gated Recurrent Unit also suggest that absence of a memory cell tends to degrade performance.**

***Index Terms*—meta-learning, recurrent networks, performance optimization, long short-term memory.**

I. Introduction

Recently, there has been a resurgence of interest in meta-learning neural networks [1]–[6]. However, recent attempts revolve mainly around weight-generating networks and few-shot learning. In weight-generating networks, a network (meta-network) is trained to modify the weight of another network. The meta-network can be regarded as a learned optimizer. Unlike traditional supervised learning, in few-shot learning the amount of data available for each class is relatively small, thus the challenge is how to generalize from such limited data.

In this paper, we focus on a different perspective of meta-learning. This approach tackles how to learn meta-information underlying multiple related tasks the neural network (NN) system is trying to solve, rather than training an optimizer in the meta-network approach or learning to generalize from a limited amount of training data. Our meta-learning framework is based on the Fixed-Weight Learning (FWL) Theorem [7]. Let N_c denote a (changing) weight NN with it's associated learning algorithm such as Backpropagation. The theorem demonstrates that there exists a FWL recurrent network which can learn any mapping that N_c can learn, but without changing synaptic weights, hence the term fixed-weight (changing signal) learning. This is because the recurrent activation signals (called potencies) contain information about the particular mapping or task that is being learned. The learning algorithm is embedded in (a subset of) the network's fixed synaptic weights. The FWL-NN computes new potencies that control the network's learned behavior based on feedback obtained on the error signals of prior iterations.

The meta-learning framework presented and analyzed in this work is referred to as Fixed Weight Meta-Learning (FWML). It is not about the learning of a particular task, but rather learning to learn better [8], [9]. During meta-learning, the "fixed synaptic weights are varied so the network can derive novel and efficient learning methods. It leverages the gradient and smoothness of the parameter space to find and optimize effective learning, unlike some other meta-learning techniques [2], [10]–[13]. In Fixed-Weight Learning Neural Networks (FWL-NNs) the learning algorithm is encoded in a manner that differentiable and continuous (i.e. synaptic weights). This means that means that small changes in the parameters results in small changes in performance. In all optimization algorithms of which we are aware, this smoothness of the error surface results in expediting the finding of a solution. The differentiability of the encoding makes it possible to compute a multi-dimensional gradient in which to go for improved performance.

The goal of FWML is to generate an effective embedded learning algorithm that has learned how to learn a set of related tasks. This is very useful in design of stable intelligent au-

978-1-5090-6014-6/18/$31.00 ©2018 IEEE

tonomous systems that are capable of lifelong learning. FWL-NNs carry out all computations by the neural network itself just like in biological brains. Animal behavior and learning are concurrent and intertwined at a deep level. Extremely robust and flexible learning capabilities evolved under this scheme, suggesting that FWL-NNs should also have some of the robustness and plasticity of biological learning.

This paper presents a systematic optimization and generalization of the FWML process by evaluating various topologies and parameters of the Long Short Term Memory (LSTM) architecture that are important for derivation of generalized learning algorithms. The optimization investigates the influence of diverse structures and parameters on its performance. By generalization (robustness), we imply ability of the system to learn across multiple tasks including tasks unseen during the meta-training phase. We investigate the robustness of the FWML system to learn related tasks that were not explicitly seen during the meta-training phase. For both optimization and generalization, we introduce a meta-cost function, Mean Squared Fair Error (MSFE) (Section III-A), in contrast to Mean Squared Error (MSE), to enhance the performance of the system by not penalizing it during transition interval in learning a new task. Results obtained for both Boolean and quadratic functions datasets demonstrates the performance and robustness of our meta-learning framework.

The remainder of this paper is organized as follows. Section II presents an overview of meta-learning systems, to provide a context for this work. We describe the FWML framework in Section III. Experimental results are presented and discussed in Section IV while the conclusion and next steps are summarized in Section V.

II. Related Work

Meta-learning can be defined generally as "learning how to learn" [14]. There are different approaches and formalizations of meta-learning. One approach (weight-generating network) uses two networks in which one network (the meta-network) generates weights to transfer to another network. Some weight-generating networks systems, such as in [1], [15], [16], utilize a recurrent network to meta-train on a variety of networks including feed-forward network, LSTM, and deep convolutional network, on a variety of tasks. Generalization to new tasks and scalability problems due to a large parameter space is addressed in [17]. The overall theme of these approaches is that there exists simultaneous fast and slow changing weights in the system. Fast weights are generated on the fly to enable the network adjust to a different input, while slow weights are gradually updated in the meta-network. In contrast, our method trains a single network to embed a learning algorithm and then uses it in learning tasks related to those seen during training.

According to [4], another approach of meta-learning could be viewed as training a model on a variety of learning tasks, such that it can solve new learning tasks using only a small number of training samples. This is illustrated in one-shot or few-shot learning, where one or very few examples of each class is learned. The goal of few-shot meta-learning is to train a model that can quickly adapt to a new task using only a few data points and training iterations [4]. The model is trained during the meta-learning phase on a set of tasks, such that the trained model can quickly adapt to new tasks using only a small number of examples or trials. In effect, the meta-learning problem treats entire tasks as training examples. Koch et al. [18] trained a Siamese network how to rank similarity between pairs of input. In [2], [19] memory augmented neural network are introduced to aid the systems quickly assimilate to new data. Woodward and Finn [13] incorporates reinforcement learning with few shot learning to learn when to classify or skip the data.

In this work, we adopt a more general definition of meta-learning, which can be viewed as two learning processes proceeding simultaneously [9]. There is a controlling system, whose objective is to learn a good learning algorithm for a set of related tasks. There is also a subordinate learning algorithm, which attempts to learn a specific single task. Periodically, the controller alters the subordinate algorithm slightly to improve its learning performance. The common theme that runs through these multiple views of meta-learning is the concept of gradually adapting the internal state of the network to adjust to new tasks. For weight generating networks, this is done by a learned optimizer with slow changing weights. In the FWML framework, it is accomplished using the concept of fixed weights via the memory cell in the LSTM. The meta-learning problem studied in this work differs from weight generating networks in that we aim to learn directly from error feedback to infer higher level meta-information, rather than simply learning similarity between data points. Hence, a more general meta-learning framework. Ho Younger et al. [20] presented a method for FWL-NNs using analytically derived sub-networks to perform the learning computations. These networks were applied to the dynamic learning of several classes of problems, such as learning any mapping with two Boolean inputs and one Boolean result. They also discussed the possibility of deriving FWL-NN learning algorithms by an optimization process, called meta-learning. Hochreiter and Younger [8] used FWML with the LSTM topology to derive learning algorithms optimized for classes of problems similar to those used in [21]. This paper explores FWML in a framework where the controlling system is a batch-mode Backpropagation Through Time and the subordinate algorithm is an online FWL-NN implemented on a LSTM neural network.

III. Fixed-Weight Meta-Learning Network

Given a set of related tasks F to be learned, the FWML generates an optimized learning algorithm that is meta-trained using a set of tasks $f \in F$. The goal is to derive a generalized meta-learned algorithm that can learn all tasks in F including tasks unseen during the meta-training phase. Given a meta-training dataset D for a specific F, let N_f denote the number of tasks $f \in F$ presented during meta-training. It should be noted that N_f counts all tasks in D even repeated ones. Randomly repeating tasks is important so that the training

space is more broadly sampled. Beyond this, we also randomly present multiple exemplars of each task denoted by N_E.

Figure 1 illustrates a single training epoch which consists of N_f task episodes, each consisting of N_E exemplars. The first exemplar of a new task episode is a transition point indicating the point in time when the system switches from one task episode to another (e.g. from f_3 to f_5, from f_5 to f_2). This is illustrated by short vertical bars in Figure 1. The transition region (interval) consists of the first k exemplars starting with the transition point. During the transition interval, the network has to adjust its state from the previous task to the present task. Note that without any input to specify which task is currently being learned, the FWL-NN is trained to learn across multiple tasks. Figure 2 illustrates the meta-learning process of FWL-NN. The input to the network consists of the current input exemplar of task $x(t)$, the networks output from previous input exemplar $y(t-1)$, and the error $\epsilon(t-1)$ of the last exemplar. The error is given by $\epsilon(t) = y(t) - T(t)$ where $T(t)$ is the target or correct value associated with $x(t)$.

There are two different learning mechanisms taking place in the network. First is the update of weights of the network by a gradient optimizer on the networks loss function. Second is the update of the internal states of the network (hidden states and memory cells) using the recurrent feedback error. This enables the network to learn which specific task is taking place. The goal of FWL-NN is then to minimize the error during each task episode. Algorithm 1 outlines the meta-training phase of the FWL-NN on a set of tasks. Note that feeding back both the error and output from the last time step is essential for good FWML performance (as verified by preliminary experimental results). The recurrent network unrolls input x, error ϵ, and output y from previous time steps to output the value for the present time step.

Algorithm 1: Fixed-Weight Meta-Learning System Training

```
while not done do
    Sample D(F, N_f, N_E)
    for t = 0 : size(D) do
        if t = 0 then
            Initialize Unroll(0)
        end
        else
            Update Unroll(t) from x(t), y(t-1), ε(t-1)
        end
        y(t) = NN(Unroll(t); Θ)
        ε(t) = y(t) − T(t)
        Compute MSFE
        Accumulate gradient w.r.t Θ:
        ∂Θ ← ∂Θ + ∂(MSFE(y(t), T(t))
    end
    Update Θ using a gradient optimizer
end
```

A. Mean Squared Fair Error Function

In traditional supervised learning using NN, the loss function (Mean Squared Error - MSE) is applied to all of the data points during training. This approach of computing the error over all instances of tasks may not be optimal for a meta-learner. As the training proceeds, the network is exposed to a series of related but different tasks, instead of a single task as in supervised learning. Thus, the error is expected to spike during the transition interval at the beginning of a new task. As the system adjusts to a new task, this error should subsequently decrease. We propose that to improve the learning performance of the FWML system, the system should not be penalized during these transition intervals. This is accomplished by utilizing an error function, Mean Squared Fair Error (MSFE), that ignores (or disregards) the error of the system during the transition interval. In our experiments, k is set to 10 i.e. a transition interval consists of first 10 exemplars of a task episode. During testing, we also quantify the performance of the system using Mean Fair Absolute Error (MFE), which is defined as the Mean Absolute Error (MAE) over all data points excluding the error during the transition intervals, to allow direct comparison with previously published FWML performance results [8].

B. Subordinate Learning Algorithm: LSTM Variants

One of the objectives of this work is to understand which features are important for efficient derivation of effective learning of the FWML framework. Given that there are different variants of the LSTM, a key contribution of this study is to understand which variants are most useful for the FWML and why? LSTM was introduced by [22]. Since then, there has been several modifications of the LSTM [23]. The Forget Gate is one of the most significant additions to the LSTM as introduced in [24]. This study expands on the LSTM architecture used in [9] which does not utilize a forget gate. We utilize the standard equations for the gates, memory cell and outputs of the LSTM [23].

$$z^t = g(W_z x^t + R_z h^{t-1} + b_z) \tag{1}$$

$$i^t = \sigma(W_i x^t + R_z h^{t-1} + b_i) \tag{2}$$

$$f^t = \sigma(W_f x^t + R_f h^{t-1} + b_f) \tag{3}$$

$$c^t = i^t \odot z^t + f^t \odot c^{t-1} \tag{4}$$

$$o^t = h(W_o x^t + R_o h^{t-1} + b_o) \tag{5}$$

$$y^t = o^t \odot \Phi(c^t) \tag{6}$$

where σ is the logistic sigmoid, used for activation of the gates. The g and h functions utilizes hyperbolic tangent for the block input and output activations. $\odot$ denotes the point-wise multiplication between two vectors.

We empirically explored the various LSTM architectures described in [23]. The following three variants demonstrated superior performances as the subordinate learning algorithms for our system: Full LSTM (LSTM), No Forget Gate (NFG),

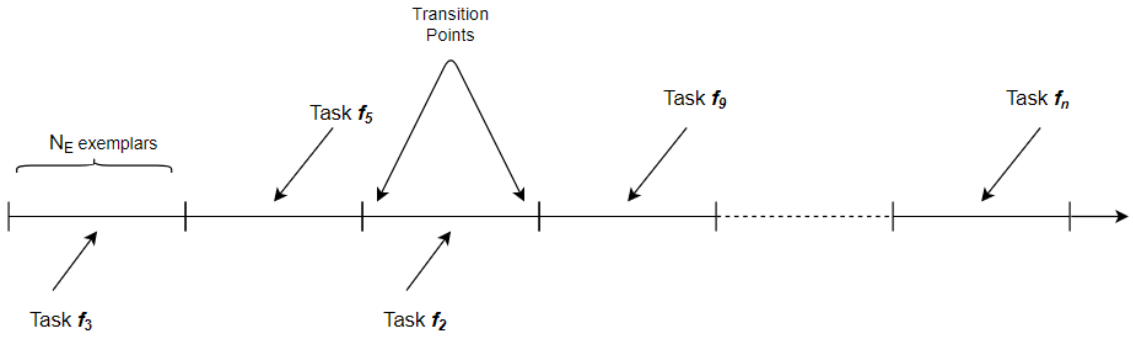


Fig. 1: A meta-learning epoch consists of N_f task episodes, each consisting of N_E exemplars. The first exemplar of a new task episode is a transition point. The transition region consists of the first k exemplars starting with the transition point. During these transition interval, the network has to adjust its state from the previous task to the present task.

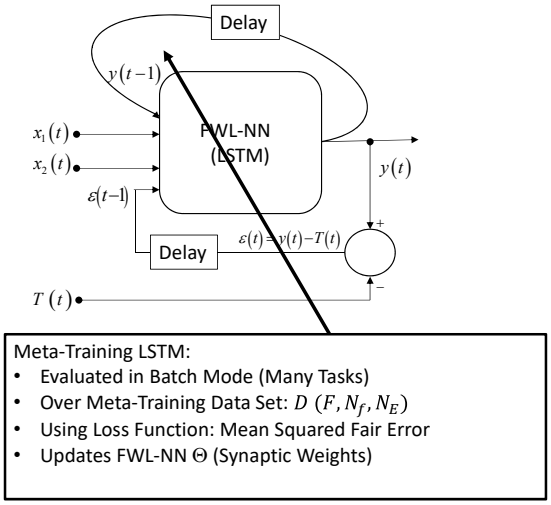


Fig. 2: Fixed-Weight Meta-Learning Framework

Coupled Input and Forget Gate (CIFG). A large bias for the forget gate, observed to improve performance in [25], is implemented for the Full LSTM variant. In the CFG, both the input gate and forget gate are coupled together and thus jointly trained i.e: $f^t = 1 - i^t$. The CFG variants exploits the notion that the input gate and forget Gate are trained independently in the full LSTM architecture. In the NFG variant, there is no forget gate, hence, the name and $f^t = 1$. The NFG variant is analogous to the original LSTM introduced in [22]. The same architecture was used in [9] to solve the Boolean problem. We are curious to see the NFG 's performance in comparison with architecture the forget gate.

The underlying reason for utilizing the LSTM as the recurrent network for the subordinate learning algorithm is its memory cell c^t. This acts as the internal state (potencies) of the network to learn the high level meta-data across tasks. Without the memory cell, we hypothesize performance would suffer. To test this hypothesis, we compare the performance of the LSTMs to the Gated Recurrent Unit (GRU) in [26] which does not have memory cell as described by its equations:

$$z^t = \sigma(W_z x^t + R_z h^{t-1} + b_z) \tag{7}$$

$$r^t = \sigma(W_r x^t + R_r h^{t-1} + b_r) \tag{8}$$

$$h^t = \sigma(W_h x^t + (s^{t-1} \odot r^t) R_h) \tag{9}$$

$$s^t = (1 - z^t) \odot h^t + z \odot s^{t-1} \tag{10}$$

Since memory cell c^t is updated slowly as it learns higher level information of tasks, there is a need for a mechanism to regulate c^t so it adjusts its states efficiently when transitioning between different tasks. There are two different approaches to address this issue. One is learning a forget gate c^t as mentioned in the LSTM equations. A gentler approach is to simply clip the value of c^t to [-1 +1] to prevent a situation when c^t gets too large and takes longer to reset its states to new tasks. The experiments, presented in SectionIV, compare three different mechanisms to adjust c^t: clipping the memory cell, utilizing the forget gate, and clipping along with forget gate.

IV. Experiments Results and Analysis

A. Experimental Setup

The overall structure of the FWML network consists of an RNN variant jointly trained with a two layer fully connected network with sigmoid activations. The network has a first hidden recurrent layer of size 24 followed by a perceptron layer of size 12. The final output node has a sigmoid activation function in the Boolean functions experiments; and tanh activation for the quadratic data. To investigate optimization of the FWML framework, we evaluated multiple RNN architectures: NFG, CIFG, Full LSTM (all LSTM variants with and without clipping memory cell), and GRU. Note that it is not possible to clip the memory cell in GRUs due to its network configuration. The system is implemented is in Tensorflow [27] using rmsprop as the gradient optimizer. The code and data is readily available on our GitHub page (https://github.com/clslabMSU/FwdLearning).

Our experiments are based on two different sets of related tasks: Boolean and quadratic. Let $F_{bool}(x_1, x_2)$ denote the Boolean functions space with two Boolean value inputs x_1 and x_2. The Boolean task set consists of 16 different tasks (functions) $f_{bool} : x_1, x_2 \rightarrow y$ in F_{bool}. The quadratic task is defined as follows. Let $F_{quad}(x_1, x_2)$ denote the quadratic functions space. Each function in F_{quad} has the following form $f_{quad} : x_1, x_2 \rightarrow Ax_1 + Bx_2 + Cx_1x_2 + Dx_1^2 + Ex_2^2 + F$, where A, B, C, D, E and F are the coefficients associated to each function and x_1, x_2 are randomly drawn from a uniform distribution. Note that for the quadratic tasks, functions do not repeat in a training/testing epoch. For both tasks, the training data consists of $N_f = 100$ task episodes with $N_E = 200$ exemplars. After each epoch, the task episode order is shuffled to prevent over-learning.

B. Comparative analysis of meta-training using MSFE vs MSE as the meta-cost function

The Boolean function experiments investigated the effect of training using the loss function as the newly defined MSFE in comparison to MSE (Figure 3). (Note that the testing performance for all systems evaluated is quantified using the Mean Fair Error (MFE) and Mean Absolute Error (MAE), as defined in SectionIII-A.) The FWML system demonstrated in Figure 3 is the NFG architecture with clipping and two different unroll values (16, 40). (Unroll parameter can be varied to obtain a good tradeoff between speed and performance.) As can be observed from Figure 3, the best performing system, according to both MFE (Figure 3a) and MAE (Figure 3b) values, is obtained with the MSFE-trained system with unroll of 40. For a lower value of unroll (16), according to testing performance using the MAE criteria, the MSE-trained performed slightly better than the MSFE-trained for fewer exemplars though at higher exemplars, there is no difference in performance. For MAE, a few large errors in a transition interval could dominate its value over the episode. This can be seen in how the MAE values of the different systems (Figure 3b) cluster closely together. The difference which can be seen is that the MAE for MSFE-trained network for the higher unroll values is lower. This also highlights the usefulness of using MSFE for training by not penalizing the system's performance during the transition interval. For all subsequent experiments, the meta-training is conducted using MSFE with unroll value of 40.

C. Performance analysis of varied RNN architectures

Table I demonstrates the performance of different network topologies on the Boolean dataset. As can be observed, the best performing system is the NFG with clipping of the memory cell. The results also show that clipping improves performance in NFG and CIFG but not in the full LSTM. The clipping mechanism is designed to help the network adjust to different tasks. When clipping is coupled with another adapting mechanism such as the forget gate, as in the case of the full LSTM, it results in a degradation of performance rather than improvement. Table I also illustrates the usefulness of the memory cell in the LSTM variants compared to the GRU, the worst performing system. The GRU performs worse in every case, possibly due to its lack of memory cells. This confirms our hypothesis that memory cell is central to meta-learning system, absence of it results in degraded performance.

TABLE I: Performance across diverse RNN architectures using Mean Square Fair Error as the loss function during meta-training. Number of exemplars for test dataset is varied from 32 to 250.

FWL-NN	Clipped	Mean Fair Error on Test Data				
		32	64	128	200	250
NFG	Yes	.00434	.00268	.00034	.00062	.00056
NFG	No	.00522	.00350	.00076	.00099	.00060
CIFG	Yes	.00512	.00282	.00069	.00079	.00069
CIFG	No	.00875	.00432	.00086	.00119	.00093
LSTM	Yes	.00960	.00501	.00176	.00135	.00103
LSTM	No	.00661	.00340	.00067	.00105	.00080
GRU[1]	No	.02201	.00699	.00342	.00294	.00160

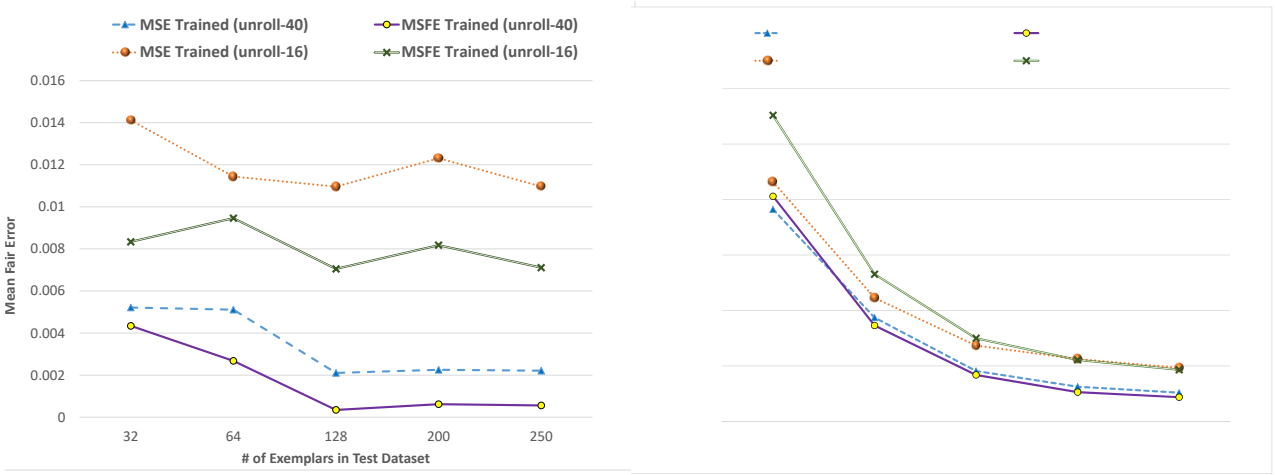
NFG: No Forget Gate LSTM; NFG: No forget Gate; CIFG: Coupled Input and Forget Gate; GRU: Gated Recurrent Unit Network; [1] Clipping can't be implemented in GRU;

Figure 4 demonstrates the result of testing the networks with the MAE function compared to testing with the MFE function. Lower values of both the MAE and MFE result in higher-slope trend lines for various N_E ranging from 32 to 250 within an episode. The highest slope indicates the NFG-Clipped network variation has the best performance. The GRU network has the worst performance. The points from higher N_E are closer to the origin and shown more clearly in the inset.

D. Generalization of FWML

To demonstrate the generalization of the system to learn unseen tasks for the Boolean functions, we compared the performance of FWML on a set of unseen tasks, i.e. tasks not encountered in meta-training, across multiple architectures. Based on performance of different topologies in Table I, NFG, CIFG, GRU with no clipping and LSTM with clipping were selected for this experiment. The selected networks are meta-trained on a subset of Boolean tasks. The testing data consists of new instantiations of tasks seen in meta-training as well as the unseen tasks. Table II shows the results of generalization to unseen tasks. In all cases, the performance on tasks seen during meta-training were better than for unseen tasks. However, NFG still yielded the best performances across all unseen tasks. We observed that the GRU still had poor performance as observed in Table I. A surprise finding is CIFG's poor performance on unseen task of OR(8) and NAND(15).

Figure 5 shows the absolute error of the FWL-NN on a series of seen and unseen Boolean tasks. The error can be observed to rise at the start of task transition, and rapidly falls once the task is learned. Tasks that were unseen during meta-training are shaded in yellow. Note that the error occasionally jumps up after it initially falls. This is more frequent in unseen tasks, causing the error to be larger for these tasks (see Figure 5b Task AND at time index 5128-5500). However, the FWL-NN was still able to learn the unseen tasks in many cases, although the network's error after learning was larger than for the functions that it was meta-trained with.



(a) Performance evaluated using Mean Fair Error.

(b) Performance evaluated using Mean Absolute Error.

Fig. 3: Mean Fair Error (MFE) vs. Mean Absolute Error (MAE) test results for four training variations of the No Forget Gate (NFG) with Clipping network. Two networks each have unroll values of 16 and 40. Two meta-cost functions, Mean Squared Error and Mean Squared Fair Error, are used for each network's training. The best performing system during test using both MFE and MAE metrics occurs with the MSFE-trained network of unroll value of 40.

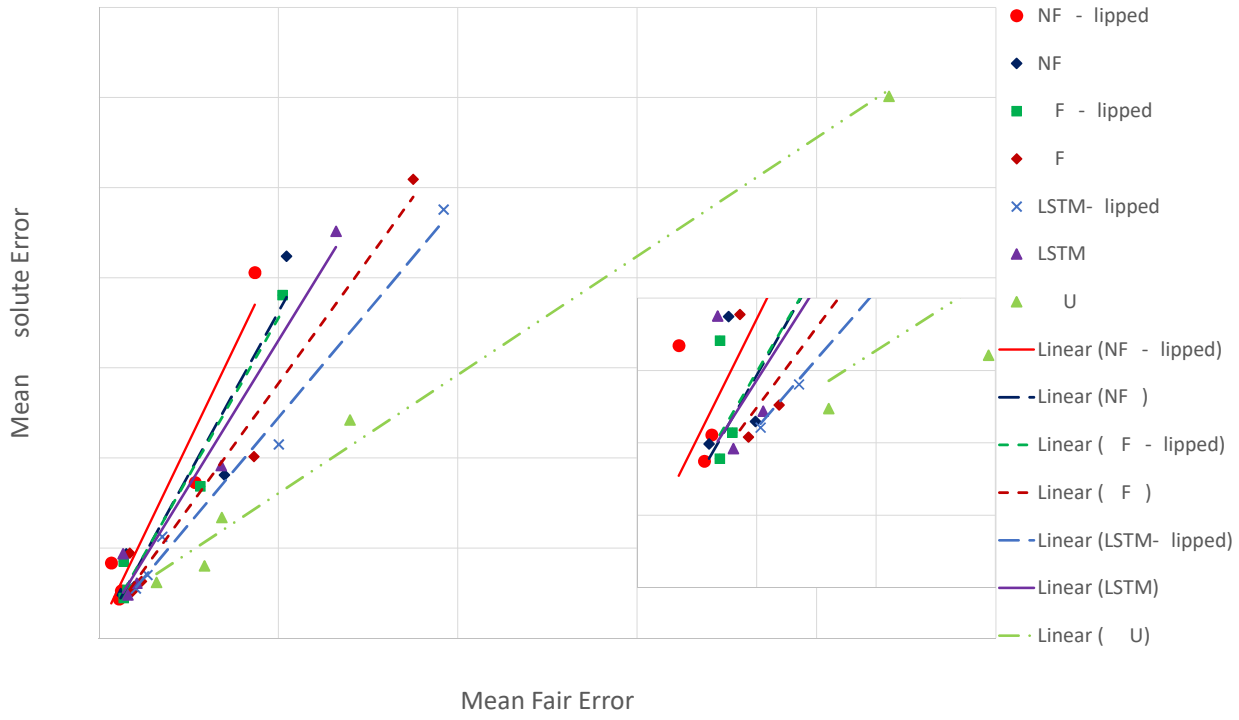
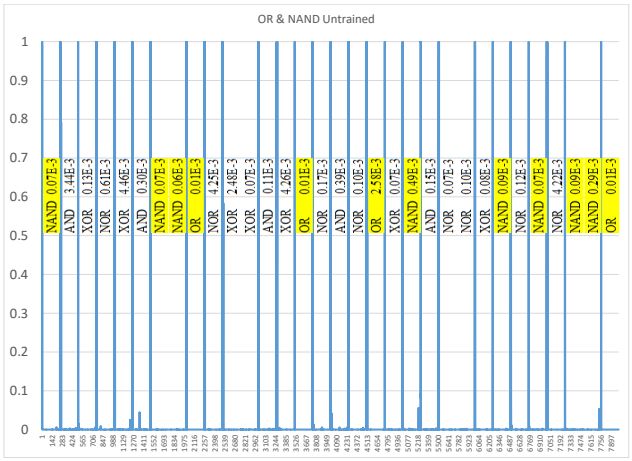
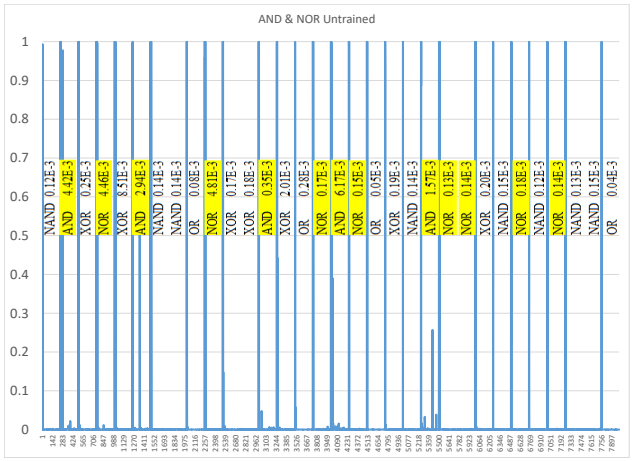


Fig. 4: Comparing Mean Absolute Error (MAE) to Mean Fair Error (MFE) for the networks of Table I. Low MAE and low MFE indicate good performance. The figure shows that both MAE and MFE errors correlate well. In addition, higher N_E give lower error and are concentrated near the origin as shown in the inset.



(a) OR & NAND unseen during meta-training.



(b) AND & NOR unseen during meta-training.

Fig. 5: Generalization of Learning: Two systems each with two tasks unseen during meta-training. Testing error is shown for tested tasks. Error is high during task transition. Unseen-tasks are highlighted in yellow; meta-trained-tasks are not.

TABLE II: FWL-NN generalization performance on tasks unseen during meta-training: AND(2), XOR(7), TRUE(16), OR(8), NOR(9), NAND(15).

FWL-NN Type Unseen Tasks	Trained Tasks MSE	Trained Tasks MFE	Unseen Tasks MSE	Unseen Tasks MFE
NFG (9,15)	.008	.007	.009	.009
NFG (2,8)	.013	.012	.010	.010
NFG (2,9)	.010	.010	.010	.010
NFG (8,15)	.003	.001	.009	.009
NFG (7,16)	.011	.010	.038	.037
GRU (9,15)	.083	.083	.380	.379
GRU (2,8)	.032	.032	.173	.166
GRU (2,9)	.021	.021	.057	.055
GRU (8,15)	.046	.046	.136	.131
GRU (7,16)	.033	.033	.286	.288
CIFG (9,15)	.001	.009	.009	.010
CIFG (2,8)	.009	.009	.012	.012
CIFG (2,9)	.008	.008	.021	.021
CIFG (8,15)	.030	.030	.318	.311
CIFG (7,16)	.014	.014	.116	.107
LSTM (9,15)	.011	.011	.094	.091
LSTM (2,8)	.011	.011	.069	.070
LSTM (2,9)	.009	.009	.014	.015
LSTM (8,15)	.011	.112	.114	.104
LSTM (7,16)	.033	.033	.286	.288

Generalization results tested on the quadratic functions dataset were carried out using NFG LSTM with clipped memory cells and unroll value of 40. Table III summarizes the performance on unseen quadratic functions using both MAE and MFE for two different learning rate values. (The learning rate for all Boolean functions experiments was fixed at 0.01.) We can observe that a lower learning rate yields a better performance. Figure 6 illustrates the absolute error across multiple time points for a set of unseen quadratic tasks. The error follows the same pattern as the Boolean dataset, although the value is higher. The pattern of the error, spiking during transition intervals and then subsequently decreasing, demonstrate that the FWML can fit quadratic functions, after learning has taken place.

TABLE III: Performance of NFG on unseen quadratic tasks.

Learning rate	128 Exemplars MAE	128 Exemplars MFE	150 Exemplars MAE	150 Exemplars MFE
0.01	0.037	0.027	0.033	0.025
0.001	0.046	0.036	0.044	0.035

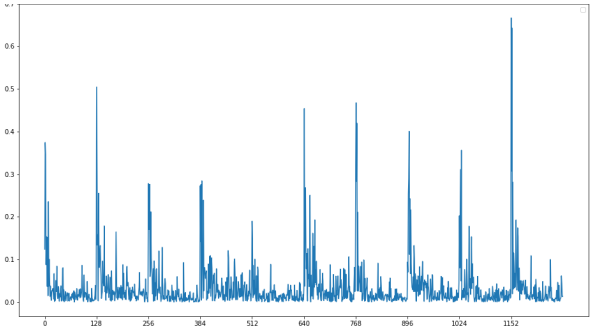


Fig. 6: NFG performance on unseen quadratic tasks with N_E of 128 and learning rate 0.01.

V. Conclusion

This work investigated a systematic optimization and generalization of the FWML system. The FWML learning algorithm was derived using LSTM-based meta-learning methods previously used by us and others [8]. Parameters examined includes unroll values, the presence or absence of LSTM "forget gates, the presence or absence of clipping of the LSTM memory neurons, and two meta-cost functions. The new results presented in this work demonstrate that FWML-derived learning algorithms have the ability to generalize and learn new tasks that were not included in the meta-training process. This ability to generalize is probably due to

the smoothness (continuous and differentiable) of the FWL-NN encoding of the learning algorithm. The results suggest several conclusions: (i) The FWML was able to leverage the regularities of the meta-training data set to generalize to new problems that it had not seen before. (ii) The usefulness of Mean Squared Fair Error as a loss function for the meta-training phase in improving system performance, as it provided the FWL-NN a few time steps to learn a new task without penalty. (iii) It is evident that the Boolean FWL-NN is not just memorizing the possible functions, since such a scheme would not generalize to not-before-seen functions. (iv) Since the meta-training data set contained several examples of functions of that are logical inverses of each other, the FWML was able to generalize operation to the extend that it could logically invert functions that it had been meta-trained on to learn function that it did not know. (v) The fact that the not-seen-before functions had larger error after learning suggests that the FWL-NN learning algorithm was not Backpropagation, since gradient descent performance would not depend on these factors.

We plan to extend this work to real-world problems and continue further performance analysis of the meta-learned FWL-NN to understand how the learning algorithm works and how to improve the scope of its generalization ability.

REFERENCES

[1] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 3981–3989.

[2] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International conference on machine learning*, 2016, pp. 1842–1850.

[3] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[4] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: http://proceedings.mlr.press/v70/finn17a.html

[5] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[6] F. Sung, L. Zhang, T. Xiang, T. Hospedales, and Y. Yang, "Learning to learn: Meta-critic networks for sample efficient learning," *arXiv preprint arXiv:1706.09529*, 2017.

[7] N. Cotter and P. Conwell, "Fixed-weight networks can learn," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*. IEEE, 1990, pp. 553–559.

[8] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *International Conference on Artificial Neural Networks*. Springer, 2001, pp. 87–94.

[9] A. S. Younger, S. Hochreiter, and P. R. Conwell, "Meta-learning with backpropagation," in *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 3. IEEE, 2001.

[10] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[11] S. Bengio, Y. Bengio, and J. Cloutier, "On the search for new learning rules for anns," *Neural Processing Letters*, vol. 2, no. 4, pp. 26–30, 1995.

[12] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl 2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[13] M. Woodward and C. Finn, "Active one-shot learning," *arXiv preprint arXiv:1702.06559*, 2017.

[14] J. Schmidhuber, "On learning how to learn learning strategies," Tech. Rep., 1995.

[15] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu, "Using fast weights to attend to the recent past," in *Advances In Neural Information Processing Systems*, 2016, pp. 4331–4339.

[16] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016.

[17] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," *arXiv preprint arXiv:1703.04813*, 2017.

[18] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML Deep Learning Workshop*, vol. 2, 2015.

[19] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 3630–3638.

[20] A. S. Younger, P. R. Conwell, and N. E. Cotter, "Fixed-weight on-line learning," *IEEE Transactions on Neural Networks*, vol. 10, no. 2, pp. 272–283, 1999.

[21] N. Cotter and P. Conwell, "Learning algorithms and fixed dynamics," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 1. IEEE, 1991, pp. 799–801.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, 2017.

[24] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[25] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 2342–2350.

[26] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.