Meta-Learning with Backpropagation

A. Steven Younger
University of Colorado
Computer Science
Boulder, CO 80309 USA
syounger@boulder.net

Sepp Hochreiter
University of Colorado
Computer Science
Boulder, CO 80309 USA
hochreit@cs.colorado.edu

Peter R. Conwell
Westminster College
Physics Department
Salt Lake City, UT 84105 USA
conwellp@xmission.com

Abstract

This paper introduces gradient descent methods applied to meta-learning (learning how to learn) in Neural Networks. Meta-learning has been of interest in the machine learning field for decades because of its appealing applications to intelligent agents, non-stationary time series, autonomous robots, and improved learning algorithms. Many previous neural network-based approaches toward meta-learning have been based on evolutionary methods. We show how to use gradient descent for meta-learning in recurrent neural networks. Based on previous work on Fixed-Weight Learning Neural Networks, we hypothesize that any recurrent network topology and its corresponding learning algorithm(s) is a potential meta-learning system. We tested several recurrent neural network topologies and their corresponding forms of Backpropagation for their ability to meta-learn. One of our systems, based on the Long Short-Term Memory neural network developed a learning algorithm that could learn any two-dimensional quadratic function (from a set of such functions) after only 30 training examples.

1 Introduction

This paper reports on our work utilizing gradient descent methods (i.e. Backpropagation) to *search out and find* learning algorithms tailored to specific learning tasks (meta-learning).

After a brief review previous meta-learning systems, we will discuss Fixed-Weight Learning Neural Networks, which motivates our method. We will also review the Long-Short Term Memory Network. Section 3 describes our meta-learning evaluation experimental set-up. In Section 4, we summarize our results. Finally, we will discuss some of the questions raised by our work.

2 Previous Work

In meta-learning, there are two learning processes proceeding simultaneously. There is a supervisory system, which is attempting to learn a good learning algorithm for a set of problems with similar characteristics. There is also a subordinate learning algorithm, which is attempting to learn a specific problem. Periodically, the supervisor alters the subordinate algorithm slightly to improve its learning performance. Mostly, these two algorithms must perform the same task: they must leverage the regularities of their respective problems in order to efficiently solve them. However, there are differences in the time scale and scope of their problems. The supervisory process has a broader scope. It must ignore the details unique to specific problems, and look for symmetries over a long time scale, while the opposite is true for a subordinate learning scheme.

2.1 Review of Meta-Learning

Several researchers have used meta-learning techniques to derive or improve learning algorithms [1,2,3]. For example, Runarsson and Jonsson in [2] used a genetic algorithm to evolve neural networks that implemented sophisticated learning rules. Some conclusions of the study were that the evolved networks are fast learners; and the derived learning rule is biased, i.e. it is 'tuned' to solve a given problem class fast.

The self-modifying neural networks of Schmidhuber et al. [3], which run their own learning algorithms, are similar to our meta-learning method. Unlike our networks, their networks required special units to read and modify their synaptic weights during learning.

2.2 Fixed-Weight Learning Neural Networks

For la]rge networks, genetic-based meta-learning can become intractable due to the number of computations required. We used Fixed-Weight Learning Neural Networks (FWNNs) [4-7] to motivate how to use of gradient descent to speed up meta-learning.

FWNNs are recurrent networks that have a learning algorithm encoded or wired into their synaptic weights. Recurrent signal loops store information about the particular mapping being learned by the network. Thus, they can learn without changing any of their synaptic weights.

Figure 1 illustrates the conceptual steps involved in converting a single synapse neural network and its attendant learning algorithm into an equivalent FWNN. (This example ignores certain timing issues that change the details of the conversion, but not the overall concept.).

We will use the term *embedded learning algorithm* to refer to a learning algorithm encoded in synaptic weights.

FWNNs move the adaptation associated with learning a particular mapping to the dynamics of the networks. The adaptation is manifest in the changing signals in the recurrent loops. On the other hand, the weights in a FWNN network represent the learning algorithm. Since the networks output error is continuous with respect to changes in the synaptic weights, gradient decent applied to these weights *is* meta learning.

The new idea we bring with this paper is that *any* recurrent network can be considered a potential fixed weight learning network. In other words, a recurrent network with random weights is simply a very inefficient learning machine. By applying standard gradient decent to these synaptic weights we improve the embedded learning algorithm associated with these weights. Furthermore we can perform meta-learning without any modifications to the training algorithm(s) normally used for that network. A fully recurrent network trained with the Williams and Zipser algorithm [8] can, in principle, be used for meta-learning. However, the training set must include exemplars from many different types of functional mappings.

We have found certain recurrent architectures to be better than others at meta learning. One architecture in particular is the Long Short-Term Memory (LSTM).

2.3 The Long Short-Term Memory Network

The LSTM Network [9] is a type of recurrent network that was designed to overcome the problems that appear when trying to learn to store information long time intervals. In addition to standard neurons, the LSTM has special *memory cells*, shown in Figure 3. The memory cells consist of three main components: a self-recurrent linear neuron, input and output gate units controlled by gatekeeper neurons, and a non-linear output squash unit. A LSTM can have either one or two hidden layers. Neurons within each layer are fully interconnected. A special LSTM Truncated Backpropagation is used to train the network. We included the LSTM in our study because Shitoot [10] noticed strong similarities between the LSTM and the FWNNs reported in [7].

3 The Key to Meta-Learning: Preparing the Meta-Training Data Set

The selection of the training data is what determines the difference between regular (non meta-) learning and meta-learning. Regular learning uses several examples of inputs and the associated target outputs from a single functional mapping. For meta-learning, we need many training pairs from many different functional mappings from a given set of such mappings. We will illustrate by giving a specific example: the set or class of all Boolean mappings with two arguments and one result. This set of sixteen functions includes the standard AND, OR, and XOR. Our training corpus consisted of 100 instances the Boolean maps, selected in random order. For each instance of a Boolean map, there was a *sequence* of 256 randomly generated training vectors. During each training *cycle*, we presented one of these vectors to the network.

A training *epoch* consisted of a complete pass through all 25,600 vectors. Each training vector also contains the target output for the inputs of the current cycle. However, this value was only used to maintain a running tally of the mean squared error. The tally was used by the supervisory program for the meta-learning.

The FWNN's embedded learning algorithm needed a supplementary input so it can learn the presented mapping. We could have used the error of the network's output associated with the *previous* cycle's input vector. Another possible supplementary input was the target (i.e. the function's result) associated with the previous cycle's input vector. We choose the latter approach. Thus, each training vector had three input values: the two Boolean arguments, and the target for the previous training cycle.

4 Experimental Results

In [11] we detailed results of our experiments with gradient-based meta-learning. We evaluated several different recurrent network topologies (with their corresponding versions of Backpropagation) for their meta-learning ability. After meta-training, we evaluated the resulting learning networks on separately generated test data.

We tested the potential meta-learning topologies and algorithms on three sets of functional mappings. The first is the set of Boolean mappings described above. The second was a set of semi-linear function mappings given by the expression $y = \frac{1}{2} \cdot (1 + \tanh(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3))$, where $x_i, w_i \in [-1,1]$. The x's are the network inputs, the y is the target function value. The w's parameterize or specify the particular mapping. This is the set of *all* mappings that a single neuron with one bias and two inputs can learn exactly (with weights in the range [-1,+1]).

The third set of mappings was the set of two-parameter quadratic functions given by: $y = ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$, where $a, \ldots, f \in [-1,1]$ parameterize the particular mapping. The x's were as above, and the y was scaled to the interval [0.2,0.8] before being used as the target value.

The only fully successful topology in meta-learning was the LSTM neural network and its associated LSTM Backpropagation. The LSTM meta-learning could successfully derive a learning network for all mapping sets attempted. We used two versions for the LSTM, the standard three-layer version, and a modified four-layer version. The latter was required to derive a learning network for the quadratic problem set. Table 1 summarizes the LSTM results. The first column shows the structure of the hidden layers. The first LSTM had one hidden layer with six memory cells and six standard neurons. The second meta-learning network had 12 memory cells and 6 regular neurons in its first hidden layer. It also had a second hidden layer with 40 standard neurons. The second column is the set of mappings that were to be meta-learned. The third column is the number of examples for each mapping's presentation sequence. The fourth column is the number of epochs that the meta-learning program required to derive the learning algorithm. The fifth column is the Mean Squared Error on test data after meta-training has occurred. The final column is the average number of steps that the derived learning algorithm required to converge.

Figure 3 shows a plot of absolute error versus time, after meta-learning was successful. The plot is for the Boolean set of functional mappings. The peaks at 512, 768, and 1024 indicate large error when a new mapping begins. Note that the error rapidly reduced after each change, indicating that the learning network performed successfully.

Note that the resulting learning networks were rapid learners. The Boolean learning network, for instance, took only about ten steps to learn a new mapping – including XOR and NOT XOR.

The most important aspect of our work was that effective learning networks were automatically derived by the LSTM meta-training, not the specific learning networks that were generated.

5 Discussion

Why could the LSTM meta-learn while other architectures could not? We believe that there were two necessary features. We showed in [11] that the recurrent loop-back synaptic weights must be 1.0 and the neuron must have a linear squashing function into store information long-term. (Actually, the constraint is slightly less restrictive than this.) We also showed this experimentally in [7]. The second necessary feature was the input gatekeeper units, which control the input to the loop cell. By learning when to allow and (perhaps more importantly) when to disallow new information into the memory cell, the

LSTM can store information for the longer periods of time needed to do meta-learning.

The Π units could be replaced by an equivalent (standard neuron) network at the expense of more complexity.

How did the resultant learning networks work? Were they similar to known methods? It is very difficult to take apart a neural network (especially a recurrent network) and extract the rules that are encoded in its synaptic weights. However, examination of the output of the memory cells revealed that the Boolean problem learner encoded the sixteen possible functions by a four-neuron binary encoding scheme. Obviously, this way of enumerating the mappings would only work for small sets of mappings, each with a small number of possible results (in this case 0 or 1). The meta-learning correctly extracted these properties from the meta-training data set. This is similar to the way a human being may try to solve the problem.

Meta-learning on the set of Semi-linear functions resulted in a learning network that stored three continuous values in the memory cells. This reflects the continuous, three-parameter nature of the set of mappings.

The Quadratic problem learner also generated continuous values in its memory cells. Another signal it generated was approximately inversely proportional to the cycle step number within a sequence. We believe that the network used this signal to increase the influence of the errors near the beginning of the sequence, speeding up learning.

References

[1] David J. Chalmers, "The Evolution of Learning: Experiments in Genetic Connectionism" in *Proceedings of the 1990 Connectionist Models Summer School.* Editors D.S. Touretsky, J.L. Elman, T.J. Sejnowski & G.E. Hinton, Morgan Kauffmann; San Mateo, CA

[2] Thomas Philip Runarsson and Magnus Thor Jonsson. "Evolution and Design of Distributed Learning Rules" *2000 IEEE Symposium of Combinations of Evolutionary Computing and Neural Networks*. San Antonio, Texas (2000) p. 59

[3] J. Schmidhuber. "A neural network that embeds its own meta-levels." *In Proc. Of the International Conference on Neural Networks '93, San Fransisco,* IEEE 1993

[4] N. E. Cotter and P. R. Conwell. "Fixed-Weight Networks Can Learn." In *International Joint Conference on Neural Networks held in San Diego 1990,* IEEE, New York, 1990, pp. II-553- 559.

[5] N. E. Cotter and P. R. Conwell. "Learning Algorithms and Fixed Dynamics." In *International Joint Conference on Neural Networks held in Seattle 1991* by IEEE. New York: IEEE 1991, I- 799- 804.

[6] A. Steven Younger, Learning in Fixed-Weight Recurrent Neural Networks. Ph.D. Dissertation, University of Utah 1996

[7] A. Steven Younger, P. R. Conwell, and N. E. Cotter. "Fixed-Weight On-Line Learning." *IEEE Transactions on Neural Networks*. Vol.10 No. 2, March 1999 pp. 272-283

[8] R. J. Williams and D. Zisper, "A learning algorithm for continually running fully recurrent neural networks," Univ of California, San Diego, La Jolla, CA. Tech Report TR-8805.

[9] Sepp Hochreiter and J. Schmidhuber, "Long Short-Term Memory." *Neural Computation 9(8)* pp. 1735-1780, 1997

LSTM source code can be obtained by:
ftp ftp.cs.colorado.edu
cd users/hochreit/software
get hochreiter.lstm.tar.gz

[10] Yashwant Shitoot, *Private Communication,* 1996

[11] Sepp Hochreiter, A. Steven Younger and Peter R. Conwell. "Learning To Learn Using Gradient Descent." to appear in *Proceedings of the International Conference on Artificial Neural Networks,* Springer Verlag 2001

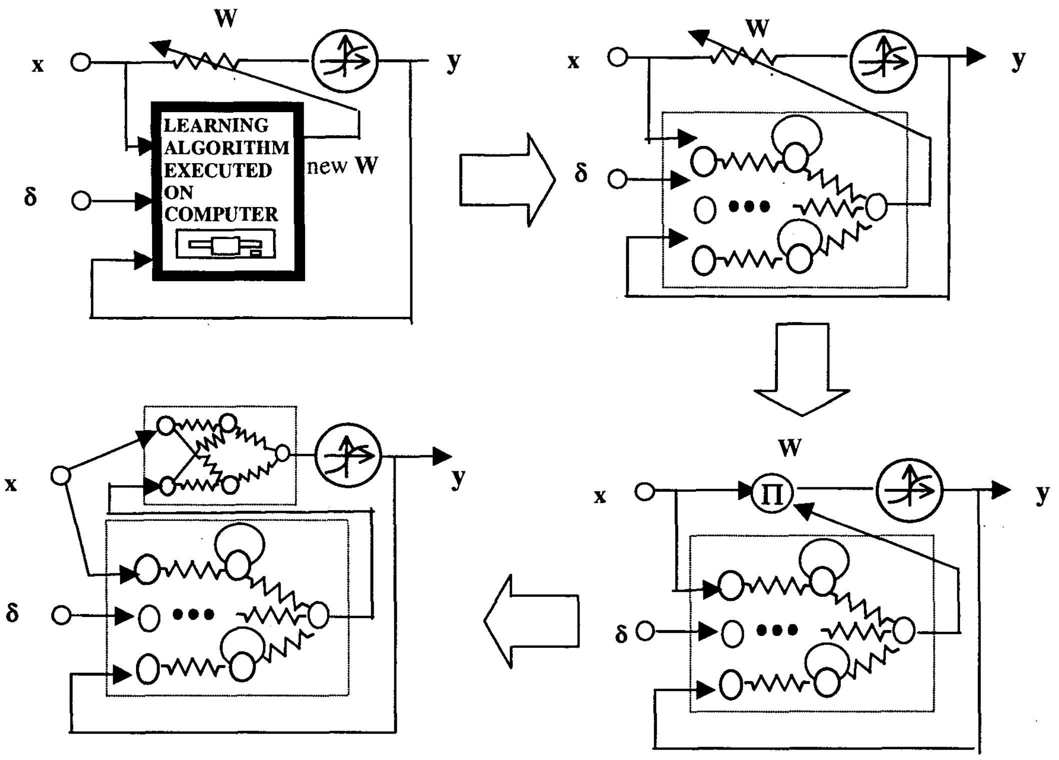

Figure 1: Construction of an equivalent FWNN for a single synapse and its attendant learning algorithm. Clockwise from the upper left: (1) Conventional network with learning algorithm $newW = f(x, y, \delta, oldW)$. (2) Universal approximation allows us to replace the learning algorithm with an equivalent recurrent network. Note that recurrence is necessary to store the oldW information dynamically in signal loops. (3) Replace the synapse with a Π unit, removing the requirement to change the synaptic weight. (4) If required, replace the Π unit with an equivalent non-Π network.

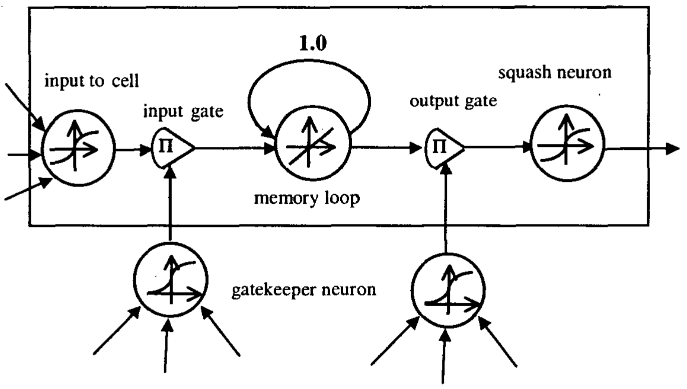

Figure 2: LSTM memory cell. Key features are the input and output gates controlled by gatekeeper neurons, the linear memory loop neuron and the output squash neuron. The gatekeeper neurons learn when to allow data in and out of the memory loop neuron.

Table 1: Performances of Automatically Derived LSTM-Based Learning Networks

Hidden Neurons	**Problem Set**	**Examples per Mapping**	**Epochs**	**MSEt**	**Cycles to Learn**
H_1: 6 Memory + 6 Standard	Boolean	256	800	0.0058	10
	Semi-Linear	64	10000	0.0008	10
	Semi-Linear	1000	5000	0.0025	50
H_1: 12 Memory + 6 Standard H_2:40 Standard	Quadratic	100	25000	0.00068	35

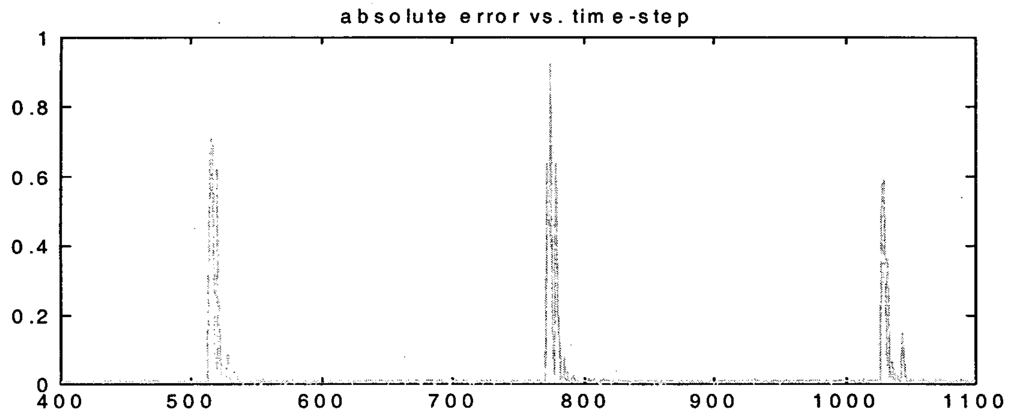

Figure 3: Absolute error versus time, after meta-learning was successful. The plot is for the Boolean set of functional mappings. The peaks at 512, 768, and 1024 indicate a large error when a new mapping begins. The rapid reduction of the error after the peaks shows that the net mapping was learned quickly. Before meta-learning, this entire plot would have consisted of errors the size of the peaks.