Learning To Learn Using Gradient Descent

Chapter *in* Lecture Notes in Computer Science · September 2001

DOI: 10.1007/3-540-44668-0_13 · Source: DBLP

3 authors, including:

Sepp Hochreiter
Johannes Kepler University Linz
124 PUBLICATIONS **4,275** CITATIONS
SEE PROFILE

Arthur Steven Younger
9 PUBLICATIONS **119** CITATIONS
SEE PROFILE

Learning To Learn Using Gradient Descent

Sepp Hochreiter[1], A. Steven Younger[1], and Peter R. Conwell[2]

[1] Department of Computer Science
University of Colorado, Boulder, CO 80309–0430
[2] Physics Department
Westminster College, Salt Lake City, Utah

Abstract. This paper introduces the application of gradient descent methods to meta-learning. The concept of "meta-learning", i.e. of a system that improves or discovers a learning algorithm, has been of interest in machine learning for decades because of its appealing applications. Previous meta-learning approaches have been based on evolutionary methods and, therefore, have been restricted to small models with few free parameters. We make meta-learning in large systems feasible by using recurrent neural networks with their attendant learning routines as meta-learning systems. Our system derived complex well performing learning algorithms from scratch. In this paper we also show that our approach performs non-stationary time series prediction.

1 Introduction

Phrases like "I have experience in ...", "This is similar to ...", or "This is a typical case of ..." imply that the person making such statements learns the task at hand faster or more accurately than an unexperienced human. This learning enhancement results from solution regularities in a problem domain. In a conventional machine learning approach the learning algorithm mostly does not take into account previous learning experiences despite the fact that methods similar to human reasoning are expected to yield better performance. The use of previous learning experiences in inductive reasoning is known as "knowledge transfer" [4, 1, 14] or "inductive bias shifts" [15, 6, 13].

In the research field of "knowledge transfer" we focus on one of the most appealing topics: "meta-learning" or "learning to learn" [4, 14, 11, 12]. A meta-learner searches out and finds appropriate learning algorithms tailored to specific learning tasks. To find such learning methods, a supervisory algorithm that reviews and modifies the training algorithm must be added. In contrast to the subordinate learning scheme, the supervisory routine has a broader scope. It must ignore the details unique to specific problems, and look for symmetries over a long time scale, i.e. it must perform "knowledge transfer". For example consider a human as the supervisor and a kernel density estimator as the subordinate method. The human has previous experiences with overfitting and tries to avoid it by adding a bandwidth adaptation and improving the estimator. We want to automatically obtain such learning method improvements by replacing

the human part with an appropriate system. This automatic system must include an objective function to judge the performance of the learning algorithm and rules for the adjustment of the algorithm. Meta-learning is known in the

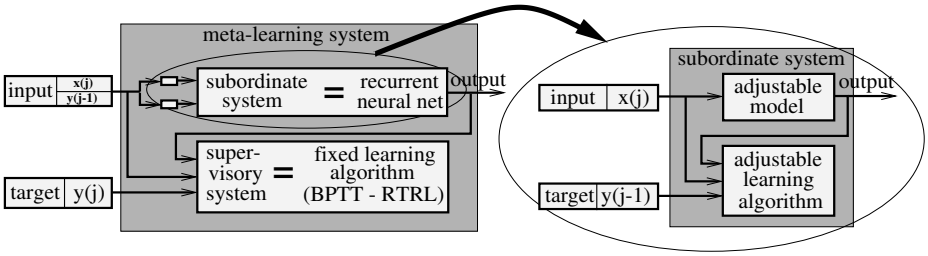

Fig. 1. The meta-learning system consists of the supervisory and the subordinate system (sequence element j is processed). The subordinate system is a recurrent network. Its attendant learning algorithm represents the fixed supervisory system. Target function arguments x are mapped to results y, e.g. $y(j) = f(x(j))$. The previous function result $y(j-1)$ is supplied to the subordinate system so that it can determinate the previous error of the subordinate model. Subordinate and supervisory outputs are identified.

reinforcement learning framework [12, 13]. This paper reports on our work on meta-learning in a supervised learning framework where a model is supposed to approximate a function after being trained on examples. Our meta-learning system consists of the supervisory procedure, which is fixed, and of the adjustable subordinate system, which must be run on a certain medium (see left hand side of Figure 1). To exemplify this, for this medium we might have used a Turing machine (i.e. a computer) where the subordinate model and the subordinate training routine is represented by a program (see right hand side of Figure 1). Any changes to the program amount to changes in the subordinate learning algorithm[1]. However, the output of the discrete Turing machine is not differentiable. Thus, only deductive or evolutionary strategies can be used to improve the Turing machine program. Instead of executing the subordinate learning algorithm with a Turing machine, our method executes the algorithm with a recurrent neural network in order to get a differentiable output. This is possible because a (sufficiently large) recurrent neural network can emulate a Turing machine. The differentiable output allows us to apply gradient descent methods to improve the subordinate routine. A recurrent network with random initial weights can be viewed as a learning machine with a very poor subordinate learning algorithm. We hypothesize that gradient based optimization approaches can be used to derive a learning algorithm from a random starting point.

[1] It should be mentioned that in general, the coded model and the coded learning algorithm cannot be separated. Accordingly, with the term "learning algorithm" we mean both.

The capability of recurrent networks to execute the subordinate system was proved and demonstrated in [3, 19]. Several researchers have suggested meta-learning systems based on neural networks and used genetic algorithms to adjust the subordinate learning algorithm [2, 10, 19]. Our goal is to obtain complex subordinate learning algorithms which need a large recurrent network with many parameters. Genetic algorithms are infeasible due to the large number of computations required. This paper introduces gradient descent for meta-learning to handle such large systems, and, thus, to provide an optimization technique in the space of learning algorithms.

Every recurrent neural network architecture with its attendant learning procedure is a possible meta-learning system. One may choose for example backpropagation through time (BPTT [18, 16]) or real-time recurrent learning (RTRL [9, 17]) as attendant learning algorithms. The meta-learning characteristic of these networks is only determined by the special kind of input-target sequences as described in section 2.1. Both BPTT and RTRL applied to standard recurrent nets do not yield good meta-learning performance as will be seen in section 3. The reason for this poor performance is given in section 2.2. In the same section, the use of the Long Short-Term Memory (LSTM [8]) architecture is suggested to achieve better results. Section 2.3 gives an intuition how the "inductive bias shift" ("knowledge transfer") takes place during meta-learning. The experimental section 3 demonstrates how different learning procedures for different problem domains are automatically derived by our meta-learning systems.

2 Theoretical Considerations

2.1 The Data-Setup for Meta-Learning with Recurrent Nets

This section describes the kind of input-target sequences that allow meta-learning in recurrent nets. The training data for the meta-learning system is a set of sequences $\{s_k\}$, where sequence s_k is obtained from a target function f_k. At each time step j during processing the kth sequence, the meta-learning system needs the function result $y_k(j) = f_k(x_k(j))$ as a target. The input to the meta-learning system consists of the current function argument vector $x_k(j)$ and a supplemental input which is the previous function result $y_k(j-1)$. The subordinate learning algorithm needs the previous function result $y_k(j-1)$ so that it can learn the presented mapping, e.g. to compute the subordinate model error for input $x_k(j-1)$. We cannot provide the current target $y_k(j)$ as an input to the recurrent network since we cannot prevent the model from cheating by hard-wiring the current target to its output. Figure 1 illustrates the inputs and targets for the different learning systems.

The meta-learning system is penalized at each time point when it does not generate the correct target value, i.e. when the subordinate procedure was yet not able to learn the current function. This forces the meta-learning system to improve the subordinate algorithm so that it becomes faster and more exact. Figure 2 shows test sequences after successful meta-learning. New sequences start at 513, 770, and 1027 when the subordinate learning method produces

large errors because the new function is not yet learned. After a few examples the subordinate system learned the new function.

The characteristics of the derived subordinate algorithms can be influenced by the sequence length (more examples per function give more precise but slower algorithms), the error function, and the architecture.

2.2 Selecting a Recurrent Architecture for Meta-Learning

For simplicity we consider one function f giving the sequence $(x_1, y_1), \ldots, (x_J, y_J)$, where $y_j = f(x_j)$. All training examples (x_j, y_j) contain equal information about f. The indices correspond to the time steps of the recurrent net. We want to bias our meta-learning system towards this prior knowledge. The information in the last output O_J (indicated by J) is determined by the entropy $H(O_J \mid X_J)$. Here probability variables are denoted by capital letters, e.g. X for the input, Y for the target, and O for the output. $H(A)$ is the entropy of A and the conditional entropies are denoted by $H(A \mid B)$ The last output is obtained by $o_J = g(y_j; x_J, x_j) + \epsilon$, where g is a bijective function with variable y_j and parameters x_J, x_j. ϵ expresses disturbances during input sequence processing. We assume noisy mappings to avoid infinite entropies. Neglecting ϵ, we get

$$H(O_J \mid X_J, X_j) = H(Y_j \mid X_j) + E_{Y_j, X_j, X_J}\left(\log\left(\left|\frac{\partial g(Y_j; X_j, X_J)}{\partial Y_j}\right|\right)\right) ,$$

where $p(Y_j \mid x_j, x_J) = p(Y_j \mid x_j)$, $\left|\frac{\partial g(Y_j; X_j, X_J)}{\partial X_j}\right|$ is the absolute value of the g's Jacobian determinant, and $E_{A,B,\ldots}$ is the expectation over variables $A, B, \ldots$.

The hidden state at time j is $s_j = u(s_{j-1}, x_j, y_{j-1})$ and the output is $o_j = v(s_j)$. With $i < j < J$ we get

$$\frac{\partial o_J}{\partial y_j} = \frac{\partial o_J}{\partial s_{j+1}} \frac{\partial s_{j+1}}{\partial y_j}, \quad \frac{\partial o_J}{\partial y_i} = \frac{\partial o_J}{\partial s_{j+1}} \frac{\partial s_{j+1}}{\partial s_{i+1}} \frac{\partial s_{i+1}}{\partial y_i}, \quad \frac{\partial s_{j+1}}{\partial s_{i+1}} = \prod_{l=i+1}^{j} \frac{\partial s_{l+1}}{\partial s_l}.$$

Our prior knowledge says that exchanging example i and j should not affect the output information. That is $H(O_J \mid X_J, X_j) = H(O_J \mid X_J, X_i)$, and also ϵ should not change. In this case $Y_j = Y_i$, $X_j = X_i$, $p(Y_j \mid x_j) = p(Y_i \mid x_i)$ for $x_j = x_i$, and $H(Y_j \mid X_j) = H(Y_i \mid X_i)$. At learn begin with arbitrary weight initialization $E_{Y_j, X_j}\left(\frac{\partial S_{j+1}}{\partial Y_j}\right) = E_{Y_i, X_i}\left(\frac{\partial S_{i+1}}{\partial Y_i}\right)$. Thus, we obtain
$E_{Y_j, X_j, X_J}\left(\log\left(\left|\frac{\partial g(Y_j; X_j, X_J)}{\partial Y_j}\right|\right)\right) - E_{Y_i, X_i, X_J}\left(\log\left(\left|\frac{\partial g(Y_i; X_i, X_J)}{\partial Y_i}\right|\right)\right) = 0$, or

$$\sum_{l=i+1}^{j} E_{Y_i, X_i}\left(\log\left(\left|\frac{\partial S_{l+1}}{\partial S_l}\right|\right)\right) = 0 , \quad \text{e.g. with } \left|\frac{\partial s_{l+1}}{\partial s_l}\right| = 1 .$$

u restricted to a mapping from s_l to s_{l+1} should be volume conserving. An architecture which incorporates such a volume conserving substructure should outperform other architectures. An architecture fulfilling this requirement is Long Short-Term Memory (LSTM [8]).

2.3 Bayes View on Meta-Learning

Meta-learning can be viewed as constantly adapting and shifting the hyper-parameters and the prior ("inductive bias shift") because the subordinate learning algorithm is adapted to the problem domain during meta-learning. As the experiments confirm, also the prior of subordinate learning algorithms is data dependent. This was suggested in [7], too. Therefore different previously observed examples might lead to different current learning.

3 Experiments

We choose a squared error function for the supervisory learning routine. All networks possess 3 input and 1 non-recurrent output units. All non-input units are biased and have sigmoid activation functions in $[0, 1]$. Weights are randomly initialized from $[-0.1, 0.1]$. All networks are reset after each sequence presentation.

3.1 Boolean Functions

Here we consider the set B_{16} of all Boolean functions with two arguments and one result. The linearly separable Boolean functions set $B_{14} = B_{16} \setminus \{XOR, \neg XOR\}$ is used to evaluate meta-learning architectures.

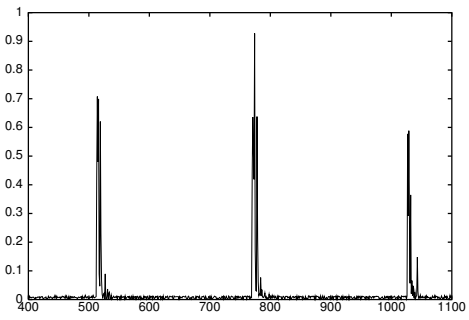

Fig. 2: Error vs. time after meta-learning.

B_{14} Experiments
We compared following methods: **(A)** Elman network [5]. **(B)** Recurrent network with fully recurrent hidden layer trained with Back Propagation Through Time (BPTT [18, 16]) truncated after 2 time steps and with Real Time Recurrent Learning (RTRL [9, 17]). **(C)** Long Short-Term Memory (LSTM [8]) with its corresponding learning procedure. The cell input is squashed to $[-2, 2]$ by a sigmoid and the cell output is a sigmoid in $[-1, 1]$. For input gates the bias is set to -1.0. Table 1 gives the results. Only LSTM was successful.

B_{16} Experiments
The results are shown in Table 2. The mean squared errors per time step (MSEts) are lower than at B_{14} because the large error at the beginning of a new function scales down with more examples. See Figure 2 for absolute error meta-learning. The peaks at 513, 770, and 1027 indicate large error when the function changes.

3.2 Semi-linear Functions

We obtain functions $0.5\,(1.0 + \tanh\,(w_1 x_1 + w_2 x_2 + w_3))$ with input vector $x = (x_1, x_2)$ by choosing each parameter w_l randomly from $[-1, 1]$. Table 2 presents

arch.	hid. units	learning method	up-date	α	train time	train MSEt	test MSEt	suc-cess
Elman	15	Elman	b	0.001-0.1	5000			NO
Rec.	20	BPTT(2)	b & o	0.001-0.01	40000			NO
Rec.	10	RTRL	b & o	0.001-0.1	20000	0.22	0.21	NO
LSTM	6/6(1)	LSTM	o	0.001	1000	0.033	0.038	YES

Table 1. *The B_{14} experiments for Elman nets ("Elman"), recurrent networks with fully recurrent hidden layer ("Rec."), and LSTM. The columns show: (1) architecture ("arch."), (2) number of hidden units – for LSTM "6/6(1)" means 6 hidden units and 6 memory cells of size 1 –, (3) learning method – for Elman nets and LSTM their learning methods are used, and BPTT is truncated after 2 time steps –, (4) batch ("b") or online ("o") update, (5) learning rate α – 0.001-0.1 means different learning rates in this range –, (6) training epochs, (7) training and (8) test mean squared error per time step ("MSEt"), and (9) successful training ("success").*

the results. With more examples per function the pressure on error reduction for the first examples is reduced which leads to slower but more exact learning.

3.3 Quadratic Functions

The problem domain are the quadratic functions $a\ x_1^2\ +\ b\ x_2^2\ +\ c\ x_1\ x_2\ +\ d\ x_1\ +\ e\ x_2\ +\ f$ scaled to the interval $[0.2, 0.8]$. The parameters $a, \ldots, f$ are randomly chosen from $[-1, 1]$. We introduced another hidden layer in the LSTM architecture which receives incoming connections from the first standard LSTM hidden layer, and has outgoing connections to the output and the first hidden layer. The first hidden layer has no output connections. The second hidden layer might serve as a model which is seen by the first hidden layer. The standard LSTM learning algorithm is used after the error is propagated back into the first hidden layer.

LSTM has a 6/12(1) architecture in the first hidden layer (notation as in Table 1) and 40 units in the second hidden layer (5373 weights). To speed up learning, we first trained on 100 examples per function and then increased this number to 1000. This corresponds to a bias towards fast learning algorithms. The results are listed in Table 2. The authors are not aware of any iterative learning algorithm with to the derived subordinate method comparable performance.

3.4 Summary of Experiments

The experiments demonstrate that our system automatically generates learning methods from scratch and that the derived online learning algorithms are extremely fast. The test and the training sequence for the meta-learning system contains rapidly changing dynamics, i.e. the changing functions, what can be viewed as a very non-stationary time series. Our system was able to predict well

experi-ment	# func-tions	# exam-ples	train time	train MSEt	test MSEt	train time subordinate	train MSE subordinate
B_{14}	128	64	1000	0.033	0.038	6	0.003
B_{16}	256	256	800	0.0055	0.0058	6	0.002
semil.	128	64	10000	0.0007	0.0008	10	0.07
semil.	128	1000	5000	0.0020	0.0025	50	0.05
quad.	128	1000	25000	0.00061	0.00068	35	0.02

Table 2. *LSTM results for the B_{14}, B_{16}, semilinear ("semil.") and quadratic functions ("quad."). The columns show: (1) experiment name, (2) number of training sequences ("# functions"), (3) length of training sequences (examples per function – "# examples"), (4) training epochs, (5) training MSEt, (6) test MSEt, (7) training time for the derived algorithm ("train time subordinate"), and (8) maximal training mean squared error per example of the subordinate system after training ("train MSE subordinate"). The B_{14} architecture was used except for "quad." (see text for details).*

on never seen changing dynamics in the test sequence. The non-stationary time series prediction is based on rapid learning if the dynamic changes.

4 Conclusion

Previous approaches to meta-learning are infeasible for a large number of system parameters. To handle many free parameters this paper presented the application of gradient descent to meta-learning by using recurrent nets. Our theoretical analysis indicated that LSTM is a good meta-learner what was confirmed in the experiments. With an LSTM net our system derived a learning algorithm able to approximate any quadratic function after only 35 examples.

Our approach requires a single training sequence, therefore, it may be relevant for lifelong learning and autonomous robots. The meta-learner proposed in this paper performed non-stationary time series prediction. We demonstrated how a machine can derive novel, very fast algorithms from scratch.

Acknowledgments
The *Deutsche Forschungsgemeinschaft* supported this work (Ho 1749/1-1).

References

1. R. Caruana. Learning many related tasks at the same time with backpropagation. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 657–664. The MIT Press, 1995.
2. D. Chalmers. The evolution of learning: An experiment in genetic connectionism. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Proc. of the 1990 Con. Models Summer School*, pages 81–90. Morgan Kaufmann, 1990.
3. N. E. Cotter and P. R. Conwell. Fixed-weight networks can learn. In *Int. Joint Conference on Neural Networks*, volume II, pages 553–559. IEEE, NY, 1990.

4. H. Ellis. *Transfer of Learning.* MacMillan, New York, NY, 1965.
5. J. L. Elman. Finding structure in time. Technical Report CRL 8801, Center for Research in Language, University of California, San Diego, 1988.
6. D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.
7. S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Comp.*, 9(1):1–42, 1997.
8. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
9. A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Camb. Uni. Eng. Dep., 1987.
10. T. P. Runarsson and M. T. Jonsson. Evolution and design of distributed learning rules. In *2000 IEEE Symposium of Combinations of Evolutionary Computing and Neural Networks, San Antonio, Texas, USA*, page 59. 2000.
11. J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook. Inst. für Inf., Tech. Univ. München, 1987.
12. J. Schmidhuber, J. Zhao, and M. Wiering. Simple principles of metalearning. Technical Report IDSIA-69-96, IDSIA, 1996.
13. J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.
14. S. Thrun and L. Pratt, editors. *Learning To Learn.* Kluwer Academic Pub., 1997.
15. P. Utgoff. Shift of bias for inductive concept learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning*, volume 2. Morgan Kaufmann, 1986.
16. P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1, 1988.
17. R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. Technical Report ICS 8805, Univ. of Cal., La Jolla, 1988.
18. R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, *Back-propagation: Theory, Architectures and Applications.* Hillsdale, 1992.
19. A. S. Younger, P. R. Conwell, and N. E. Cotter. Fixed-weight on-line learning. *IEEE-Transactions on Neural Networks*, 10(2):272–283, 1999.