MPLP: Learning a Message Passing Learning Protocol

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a novel method for learning the weights of an artificial neural network: a Message Passing Learning Protocol (MPLP). In MPLP, we abstract every operation occurring in ANNs as independent agents. Each agent is responsible for ingesting incoming multidimensional messages from other agents, updating its internal state, and generating multidimensional messages to be passed on to neighbouring agents. We demonstrate the viability of MPLP as opposed to traditional gradient-based approaches on simple feed-forward neural networks, and present a framework capable of generalizing to non-traditional neural network architectures. MPLP is meta learned using end-to-end gradient-based meta-optimisation.

1 Introduction

For the most part, learning algorithms have been hand-crafted. The deep learning community has largely converged to use almost exclusively gradient-based approaches for learning a model's parameters. Such gradient-based approaches generally impose limitations in terms of the loss landscape, choice of network architecture and training dynamics. Nevertheless, backpropagation is still the best tool in our toolkit for optimising models with an extensive set of parameters.

In this work, we leverage gradient-based learning to find a new learning protocol for tuning an arbitrary computational graph to adapt to a task from a given family of tasks. We show how this learning protocol and its associated meta-learner can be used to train traditional neural networks. This is accomplished by rethinking neural networks as self-organising systems: a graph composed of nodes representing operations such as synapses (individual weights and biases), activations and losses, that have to communicate in order to solve a given task.

We therefore propose to learn a Message Passing Learning Protocol (MPLP): given a directed graph composed of (sparsely) connected nodes, we let these nodes communicate with each other by passing k-dimensional vectors along directed edges. The meta-training phase consists of learning a MPLP that, given an initial configuration/initialization, and a training set, is able to adapt to a given task. The kind of graphs we explore in this work are all end-to-end differentiable - we therefore meta-optimize MPLP through gradient-based approaches.

We show how MPLP can be applied to feed-forward neural networks as a meta-learned replacement to gradient-based approaches. In fact, we can consider gradient-based learning algorithms as a specific instance of MPLP. We can specialize MPLP as follows: every time an operation occurs, we consider it a node. Examples of such nodes are single weights w_{ij} (or biases b_i) multiplied by an input x_i, or a value modified by an activation function. The forward pass remains unchanged. In the backward pass, instead of receiving the simple gradient of a loss scalar, each node receives a multidimensional message vector, updates its weight through a parameterized function f, and backpropagates a modified message through a parameterized function g. These functions are trainable neural networks. Given that MLPs have been shown to work as universal function approximators, we

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

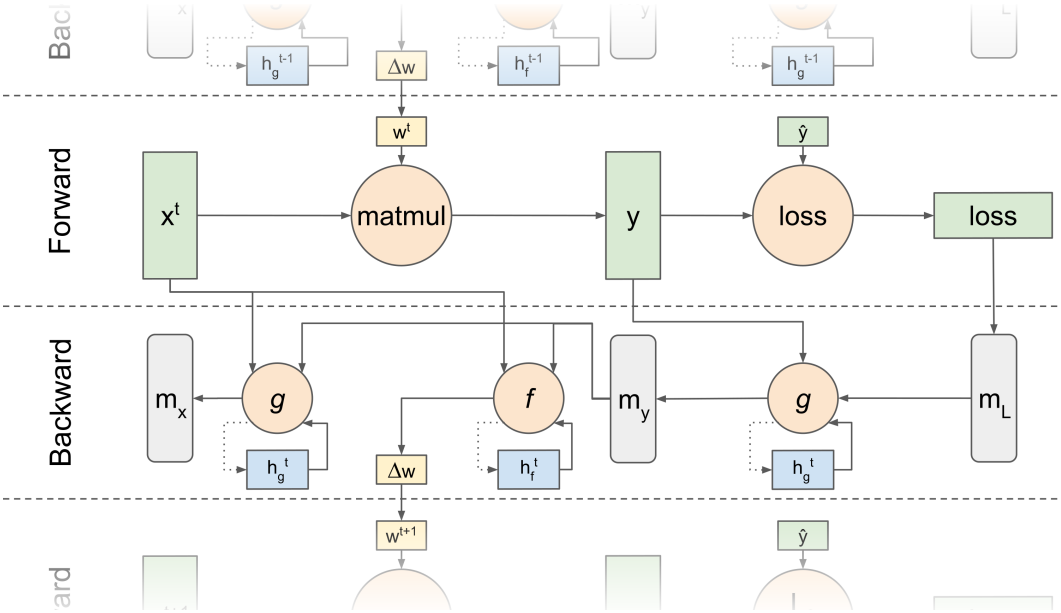

Figure 1: Diagram of simple nodes being trained with MPLP, representing a 1-dimensional matrix multiplication and a loss. f and g are the weight update and message generating networks, respectively. They each maintain their own recurrent hidden state.

propose this property would allow our learning protocol to implement traditional gradient descent arbitrarily well, but would no be limited to or encouraged to do exactly so. As a proof of concept, we demonstrate how a gradient-free MPLP can be used to train feed-forward neural networks for few-shot sinusoidal fitting and for an MNIST classification task. In the sinusoidal fitting task, we also show how we can enhance Model-Agnostic Meta-Learning (MAML) (Finn et al. [2017]) approaches by jointly learning both priors and learning rules.

2 Related Work

(Bengio et al. [1991], Bengio et al. [1997]) introduced the idea of discovering local learning rules instead of gradient-based optimization. We consider our work a generalization of their approach. Schmidhuber [1993] and follow-up work (Hochreiter et al. [2001], Younger et al. [2001]) demonstrated how LSTMs (Long Short Term Memory networks) can be used for meta-learning. We find LSTMs useful in our meta learning approach and their use is well documented in similar approaches, such as (Andrychowicz et al. [2016], Ravi and Larochelle [2017]), who present a method for optimally applying an incoming gradient signal to parameters by using an LSTM. Our work is to the largest degree inspired by the original MAML paper (Finn et al. [2017], and follow-up works Li et al. [2017], Antoniou et al. [2018]. Recent work in meta-learning has independently yielded approaches that share some key properties with our work. Metz et al. [2018] trains meta-learning unsupervised learning rules through backpropagation. Gregor [2020] uses k-dimensional messages as a communication protocol and it is conceptually very similar to our work. Bertens and Lee [2019] also uses k-dimensional message passing and a variant of an LSTM/GRU as an underlying building block. To the best of our understanding, none of these fundamentally set out to replace gradient-based approaches for training traditional NNs.

From a Graph Neural Networks (GNN) perspective, our framework could be considered an example of a Message Passing Neural Network (Gilmer et al. [2017]). However, GNNs are generally not applied to meta-learning, instead are used to ingest graph-structured data.

3 Model

In this work, we focus on specialising our models to mimic traditional feed-forward neural networks forward and backward passes in supervised learning scenarios. A typical feed-forward neural network goes through a forward pass, where given some input x and parameters θ we compute $y = F(x, \theta)$, and a backward pass, where a given loss $\mathcal{L}$ and some stored intermediate data, typically computed after a forward pass, are used to compute a gradient-based update on θ. Our computational model can be used in a forward/backward routine for architectures equivalent to traditional NN, where the forward pass is effectively unchanged. However, instead of relying on gradients for the backward pass, we meta-learn a MPLP. Every node (synapse, activation, loss value) computes a message to send back, given its stored forward input, the message being passed from the successive layer, and any internal states. We refer to this function as the *message passing rule*, or g. Nodes representing parameterized operations, such as an affine transform, further undergo a *weight update rule* defined by f during the backwards pass. For more details, refer to Appendix A. The appendix also goes into details about parameter sharing configurations, normalization schemes and the type of learners we experiment with.

3.1 Training regime

Cross-validation loss. We compute a cross-validation loss after performing k-steps of adaptation, similarly to the meta-training procedure used in MAML.

Let us call F and G the forward pass and backward pass functions of a neural network. We define G using our MPLP. The underlying neural network is parameterized by θ and MPLP is parameterized by ϕ. Let $\mathcal{L}$ be an arbitrary loss function. We adapt the neural network on a training set $(x^{(t)}, \hat{y}^{(t)})$ and keep a held-out cross-validation set $(x^{(e)}, \hat{y}^{(e)})$.

We update θ using the training set:

$$y^{(ti)} = F(x^{(ti)}, \theta_i) \tag{1}$$

$$\theta_{i+1} = G(\mathcal{L}(y^{(ti)}, \hat{y}^{(ti)}), \theta_i, \phi) \tag{2}$$

This is repeated k times. Afterwards, we compute a cross-validation loss with the held-out set:

$$\mathcal{L}_{cv} = \mathcal{L}(F(x^{(e)}, \theta_k), \hat{y}^{(e)}) \tag{3}$$

While we do not pass hidden states to the equations above, you can assume every function ingests and returns hidden states.

Hint losses. In addition to the cross-validation loss described above, we have observed it is beneficial to add a *hint loss*: at each step of the inner loop, *after* Equations 1 and 2, we compute a loss on the same training data just observed:

$$\mathcal{L}_{hi} = \mathcal{L}(F(x^{(ti)}, \theta_{i+1}), \hat{y}^{(ti)}) \tag{4}$$

This meta-loss is scaled such that the sum of all hint losses across all the inner loop steps has the same magnitude as the final cross-validation loss. We find better convergence by weighting the total hint loss to be larger than the global meta-loss. We suspect such an intermediate loss produces a smoother local loss landscape, but its inherent greediness may also hamper our model in finding an optimal convergence path.

We meta-learn ϕ through Adam and normalize the gradients for each optimizer parameter.

4 Experiments

4.1 Sinusoidal fitting

Following the example in MAML, we show convergence on tasks drawn from a family of sinusoidal functions. We sample a set of sinusoidal tasks for each training step, where a sinusoidal task is defined as a regression on samples from the underlying function $y(x) = A\sin(x + p)$, with $A \in [0.1, 0.5]$ and $p \in [0, \pi]$. We sample inner batches with $x \in [-5, 5]$. For an inner training loop consisting of k steps, we sample k inner batches. We show convergence of our approach both when trained from a randomly initialized prior, and when combined with MAML to learn prior weights of the optimizee network. We use 8-dimensional messages.

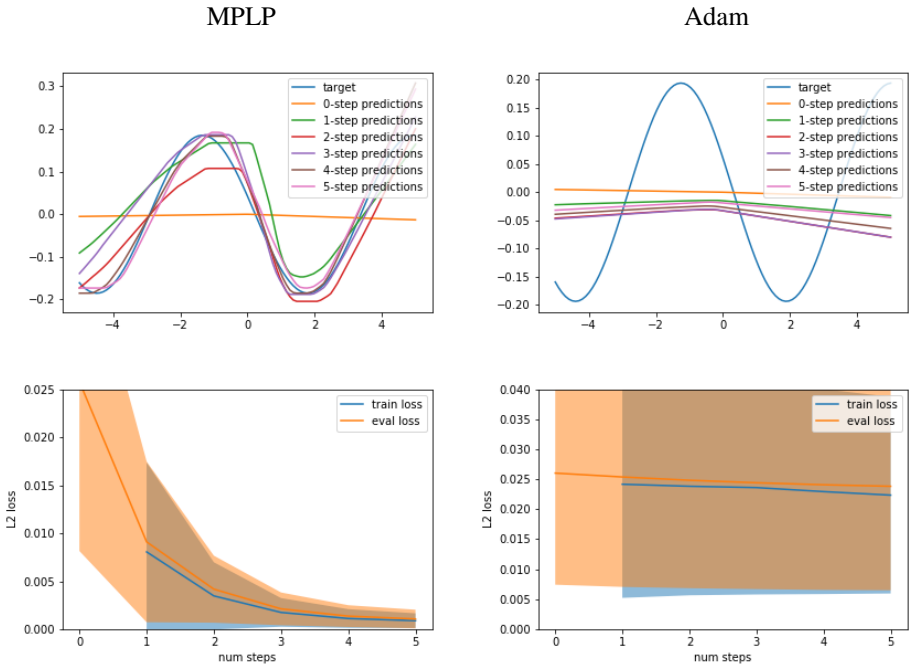

Figure 2: Comparison of MPLP (left column) learning with Adam (learning rate of 0.01) (right column). The top row shows a typical example run for the evaluation of 5-step training with inner batch size of 10. The bottom row shows means and standard deviations of the losses over 5 steps, across 100 runs; the eval loss is the L2 loss averaged across the entire domain, the train loss is the L2 loss average across all training points observed so far.

4.1.1 Randomly initialized priors

In this experiment, we train MPLP to generalize to adapt for arbitrary sinusoidals and arbitrary networks initializations. Thus, for each outer batch, we sample a random sinusoidal task and a randomly initialized network prior. The optimizee network is a traditional deep neural network with two hidden layers of 20 units each, and ReLu activations on the hidden layers. We use stateful learners for MPLP and do not share any parameters across layers.

Training. We perform 5-step learning with an inner batch size of 10. We use an outer batch size of 4. We use a cross-validation loss at the end of the 5-step learning, and a hint loss at every step. We use an L2 loss for both cross-validation and hint losses. The L2 loss is also the final node of the network (that is, the loss in the initial message in the backward stage is L2).

Figure 2 shows a comparison between MPLP and Adam[1]. The MPLP optimizer was trained for ~40,000 steps. The optimized MPLP is able to fit the task, and it vastly outperforms what Adam can achieve in 5 steps.

Training MPLP showed a great variance in its convergence results. Anecdotally, we observed a trend of obtaining better and more reliable results the larger number of parameters MPLP has. However, all parameter sharing configurations can, albeit seldom, converge to successful MPLP similar to what shown in Figure 2. We give more details in Appendix B.

4.1.2 Learning with a prior

In this experiment, we show how it is possible to meta learn the priors (as in MAML) and MPLP jointly. To showcase different properties of MPLP from what was observed in Experiment 4.1.1, we choose to use a stateless learning algorithm, perform 2-shot learning, and use only a cross-validation loss at the end (i.e. no hint loss). We also do not share any parameters across layers.

[1]We use out-of-the-box Adam parameters, except for the learning rate, that we fine-tuned to be 0.01, since it gave the best results.

MPLP & MAML MAML

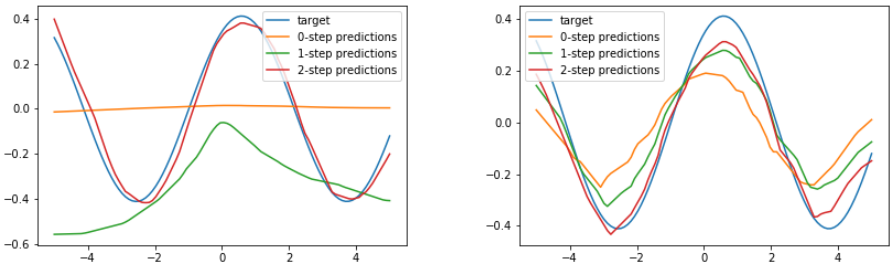

Figure 3: Comparison of models trained with MPLP and MAML (left), and standalone MAML (right).

Table 1: MNIST Accuracy at 20 steps

Model	Architecture	Accuracy @ 20 (%)
MPLP	(144,50,10), Sigmoid	48.95 ± 6.35
MPLP (transfer)	(144,50,10), Step	43.07 ± 7.36
MPLP (transfer)	(784,100,10), Sigmoid	37.60 ± 9.60
SGD	(144,50,10), Sigmoid	14.74 ± 5.80
Adam	(144,50,10), Sigmoid	39.46 ± 9.83

Figure 3 showcases how adding MPLP to MAML can drastically change the behaviour of the learner. While both models successfully solve the task, MAML runs are all always approximating a sinusoidal with priors. Moreover, since they are forced to follow the gradient at every step, it is no surprise they increasingly approximate a sinusoidal. Using MPLP, instead, the learning algorithm is not restricted to performing incremental improvements at every step. In this case, this results in nothing like sinusoidal approximation until the last step. We also verified that the learned MPLP is tightly coupled with the learned priors, as it would not succeed with a randomly initialized network. Meta-training on this experiment will yield different MPLPs and henceforth different visual results. Therefore, what we show in Figure 3 is just one example of many possibilities.

4.2 MNIST classification

The goal of this experiment is to learn a MPLP capable of generalizing to arbitrary network initializations to adapt on MNIST. To do so, we perform a 20-step training with inner batch size of 8 on a scaled-down (12x12) MNIST. We scale-down MNIST mostly for performance reasons, but this allows us to evaluate our trained learners on a full-scale MNIST afterwards. We also standardize the inputs by computing mean and standard deviation on the train dataset. The architecture used is a (144,50,10) network with sigmoid activation for the hidden layer, a softmax for the final layer, and a cross-entropy loss node afterwards. Every task in the outer batch is initialized with a different network prior. We used a stateful learner with a message size of 4 as a compromise between quality and computational efficiency.

Parameter sharing. We share the learners for all the dense layers (all weights share one pair of f and g, all biases share one pair of f and g), and keep normalization parameters layer-specific. We could not manage to achieve comparable results when sharing normalization parameters across layers, and we defer more robust explorations to future work.

Training regime. We use an outer batch size of 1 and meta-learn for 5000 steps. We do so for computational constraints and it is by no means optimal. We compute a hint loss for every step and a cross-validation loss after 20 steps. The losses used are cross-entropy.

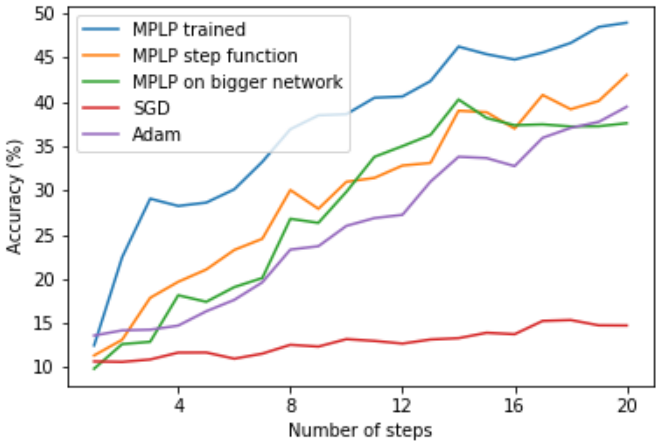

Figure 4: Evalautions of MPLP, SGD and Adam on the MNIST test set. For all models except "MPLP on bigger network", we run this on a 12x12 sized MNIST dataset. Every plot point is computed by averaging 100 test points for every of the 100 runs. Table 1 shows standard deviations on the last step.

Results. Figure 4 and Table 1 show evaluations on the test set, with randomly initialized priors, of our MPLP compared to Adam and SGD[2], averaged over 100 runs. Most evaluations are performed on the scaled-down 12x12 MNIST on a (144,50,10) architecture. The exception is for the "MPLP on bigger network", where we transfer the learnt parameters to a larger network of size (784,100,10). We can see how MPLP outperforms Adam and SGD, on average, for every step. We also experimented with replacing the sigmoid function in the hidden layer with a step function, since the two are closely related. While a step function would only backpropagate zero gradients with traditional means, we are able to train a network with it seamlessly. Finally, we ported the trained MPLP to a larger network of dimensions (784,100,10), with a sigmoid as hidden layer, and evaluated its performance on a 28x28 MNIST. Even if we did not share any standardizers, we observe how the model is still able to train meaningfully.

5 Discussion

In this article, we introduced MPLP and applied it to fairly small feed-forward neural networks with the aim of investigating whether this is a viable generalization to gradient-based approaches. We found that MPLP can be a viable way to overcome some of the problems gradient-based approaches have: adding a cross-validation loss can drive the learners to optimize for generalizing results, and prevent overfitting; catastrophic forgetting avoidance can be encoded in the meta-loss; once learned, the MPLP does not require differentiable architectures or gradient computations; vanishing and exploding gradients may be less of an issue, given that the messages passed do not necessarily need to decrease/increase in magnitude to express certain wishes of changes to be performed downstream.

More broadly, one can learn a MPLP on any end-to-end differentiable configuration of agents. This may lead to exploring novel architectures, where each agent is inherently more complex and powerful than traditional operations, or where computations happen asynchronously and in a distributed fashion. MPLP could also be used for devising implicit reward signals in continual learning or reinforcement learning scenarios.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent, 2016.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml, 2018.

[2]Here as well, we fine-tune the learning rate of both Adam and SGD to 0.01 and 0.1 respectively.

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule, 1997.

Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 969 vol.2–, 1991.

Paul Bertens and Seong-Whan Lee. Network of evolvable neural units: Evolving to learn at a synaptic level, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

Karol Gregor. Finding online neural update rules by learning to remember, 2020.

Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In *IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001*, pages 87–94. Springer, 2001.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning, 2017.

Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning, 2018.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

J. Schmidhuber. A neural network that embeds its own meta-levels. In *IEEE International Conference on Neural Networks*, pages 407–412 vol.1, 1993.

A. S. Younger, S. Hochreiter, and P. R. Conwell. Meta-learning with backpropagation. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 3, pages 2001–2006 vol.3, 2001.