Learning to Learn while Learning

Daniel Kappler[1], **Stefan Schaal**[1,3] **and Franziska Meier**[1,2]
[1]Autonomous Motion Department, MPI-IS, Tübingen,Germany.
[2]RSE-Lab, University of Washington, Seattle,USA.
[3]CLMC-Lab, USC, Los Angeles, USA.
`daniel.kappler@tuebingen.mpg.de`

Abstract

Learning-to-Learn is a branch of *meta-learning* that has recently received much attention. The promise of learning to optimize is twofold: First, automatic identification of optimization policies that minimize loss functions optimally in terms of convergence rate and final loss; Second, being able to transfer such learned optimizers to new learning tasks. Here we propose a computationally efficient online learning-to-learn approach, that continuously learns to optimize. Our *meta-learning* algorithm is general-purpose in the sense that it can be used in conjunction with any gradient based learner. We demonstrate the effectiveness of our learning-to-learn approach on a variety of learning problems involving CNNs, RNNs or GANs. When employing our learning-to-learn approach the learning tasks converge faster – requiring less samples. Furthermore, we show that we can transfer our meta-learner to related but previously unseen optimization tasks to speed up convergence.

1 Introduction

First-order gradient methods are the optimization algorithm of choice for most large scale learning problems. This is true for general machine learning problems involving high-capacity models such as deep neural networks, but also for many learning control problems in robotics. Thus, in this work we focus on learning-to-learn in the context of gradient descent:

$$w^{t+1} = w^t - \eta\ \nabla_w \mathcal{L}(f(w)) = w^t - \eta(\theta)\ \nabla_w \mathcal{L}(f(w)) \tag{1}$$

where $\mathcal{L}$ typically corresponds to some loss function, f the model – e.g. a neural network – with parameters w to be optimized, and η the learning rate. Here, we assume the learning rate to be a mapping that can be learned by optimizing parameters θ.

Recent work on learning how to optimize [1, 7, 3, 6, 4] follow a similar path, but employ a two-phase approach: First the optimization policy (in our case $\eta(\theta)$) is optimized to perform well on a priori chosen tasks. Then, once this *meta-learner* has been trained, it is utilized in similar optimization tasks. In this work, we propose an online meta-learning algorithm, that learns to predict learning rates given currently observed gradients, *while* optimizing task-specific problems. We show how we can train such a *memory of learning rates* online and in a computationally efficient manner. Our proposed approach has several advantages: Each observed data point is utilized for both the task learning as well as the meta-learner, since our learning-to-learn approach is performed online. Not only does this mean that we utilize observed data more effectively, but also that the effect is immediate. Furthermore, performing the meta-learning online alleviates the need for having to collect data on which to perform the learning-to-learn optimization process. Thus, our meta-learner can improve when necessary, while recent work is constrained to perform well on the task-distributions it was trained on. Finally, the resulting memory of learning rates can be transferred to similar optimization problems, to guide the learning of the new task.

Our experiments, on challenging models structures and objectives such as RNNs and GANs, show that when combining an optimizer with our meta-learner, we generally increase convergence speed, indicating that we utilize observed data points more effectively to reduce errors in the learning task. Furthermore, we show that when transferring our meta-learner's internal state to new learning tasks, learning progress is faster.

2 Learning a Memory for Learning Rates

In this work we investigate how we can learn a *memory of learning rates* online, in a computationally efficient manner. In order to develop such a meta learning approach several challenges need to be met. One of the core challenges is the choice of training signal to learn such a memory on. We take inspiration from [9] and start out by showing how to derive a simple learning algorithm to train a learning rate memory $\hat{\eta}$ for one-dimensional optimization problems. Then, we discuss a representation for $\hat{\eta}$ that supports computationally efficient, incremental learning.

2.1 Training Signal for the Learning Rate Memory

Let us assume that our main objective is a learning task that requires us to minimize the objective $\mathcal{L}_{\text{task}}(w)$ with respect to parameter w. While optimizing parameter w we also aim to learn $\hat{\eta}(z, \theta)$ that can predict a learning rate given gradient information of the main objective $\mathcal{L}_{\text{task}}(w)$.

$$w^{t+1} = w^t - \eta\, z = w^t - \hat{\eta}(z, \theta)\, z \tag{2}$$

where z is the gradient of the loss wrt w, $z = \nabla_w \mathcal{L}(f(w))$, and θ are the parameters of the *learning rate memory* $\hat{\eta}$. Ideally, we would like to optimize the parameters θ with respect to the loss $\mathcal{L}_{\text{lr}}$

$$\mathcal{L}_{\text{lr}}(\eta, \hat{\eta}(z;\theta)) = \frac{1}{2}(\eta - \hat{\eta}(z;\theta))^2 \tag{3}$$

where η is the true optimal learning rate, and $\hat{\eta}(z;\theta)$ the predicted learning rate for input $z = \nabla_w \mathcal{L}_{\text{task}}(w)$. To optimize parameters θ via gradient descent, we would need access to the true learning rate η which is unknown to us. However, inspired by [9] we can determine whether we over or underestimated η by comparing the gradient of the current time step with the gradient of the previous time step. If the gradient has flipped between two consecutive optimization steps, the learning rate was too large, meaning $\eta - \hat{\eta}(z;\theta) > 0$. On the other hand, if the signs of the gradients are the same, then we can most likely increase the learning rate.

With this knowledge at time step t, the memory parameter updates can be approximated as

$$\theta^{t+1} = \theta^t - \xi\, \left(-(\eta^t - \hat{\eta}(z^t;\theta^t))\right) \frac{\partial \hat{\eta}(z^t;\theta^t)}{\partial \theta^t} \approx \theta^t + \xi\, \text{sign}\left(z^{t+1} z^t\right) \frac{\partial \hat{\eta}(z^t;\theta^t)}{\partial \theta^t} \tag{4}$$

where ξ is a step size parameter for the gradient descent on θ. The exact form of this gradient update depends on what parametric form $\hat{\eta}(z;\theta)$ takes.

2.2 Learning Rate Memory Representation

As mentioned previously, we aim at developing an algorithm that can continuously update the learning rate memory $\hat{\eta}$. Designing the function approximator $\hat{\eta}$, such that forgetting of previously learned parameters is minimized, is one of the challenges of this approach. Here we choose to use locally weighted regression [10], which is known for computational efficiency and which has the capability to increase model complexity when necessary. With locally weighted regression we decompose the memory $\hat{\eta}$ into M local models $\hat{\eta}_m$, each parameterized by their own parameters θ_m. Furthermore, in locally weighted regression, the loss function $\mathcal{L}_{lr}$ also decomposes into M separately weighted losses, each dependent on only their respective parameters θ_m:

$$\mathcal{L}_{lr}(\theta) = \sum_{m=1}^{M} \mathcal{L}_m(\theta) = \sum_{m=1}^{M} \psi_m(z)(\eta - \hat{\eta}_m(z, \theta_m))^2 \tag{5}$$

where $\psi_m(z)$ is the weighting function that defines the active neighborhood for each local model. A standard selection of this weight function is the squared exponential kernel. Notice, the kernel lengthscale can be learned but within the scope of this work we assume that a reasonable lenghtscale is easy to provide. Using this memory representation leads to following update rule per local model

$$\theta_m^{t+1} = \theta_m^t + \xi\, \text{sign}\left(z^{t+1} z^t\right) \psi_m(z^t) \frac{\partial \hat{\eta}_m(z^t;\theta_m^t)}{\partial \theta_m^t}. \tag{6}$$

Note, how only local models that are sufficiently activated require updating. Finally, at prediction time the predicted learning rate is a weighted average over all local models' predictions. We now have an algorithm that can train a model $\hat{\eta}$ to predict a learning rate for one-dimensional optimization problems. We refer the reader to [8] to find out how this approach can be generalized to models with multiple high-dimensional parameter groups, while remaining computationally efficient. We only require extra memory resources in the order of $O(M)$. Nevertheless, each parameter has its own unique $\hat{\eta}$ due to ψ. Therefore, the computational complexity increases to $O(DM)$, where D is the gradient dimensionality. An illustration of what kind of learning rate landscapes can be trained with this algorithm is also given in the experimental section of [8].

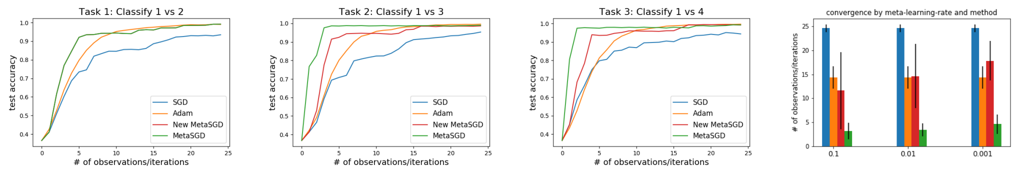

Figure 1: Results on the 3 binary MNIST learning tasks (left 3 plots top) Test accuracy as a function of observed data points with gradient descent based optimizers. (right): average number of iterations it took for the test accuracy to reach 98% accuracy on the final task for different meta-learning-rate choices ($[0.1, 0.01, 0.001]$). Learning rate of SGD and Adam is $\eta = 0.001$. Meta-learning with reloading (green) is the most sample-efficient - leading to a good predictive model faster.

3 Experiments

In [8] we have evaluated our new meta-learning algorithm on a very diverse set of experiments, including sequential binary MNIST classification tasks and online inverse dynamics learning tasks. These results show that combining gradient based optimizers such as stochastic gradient descent with our meta-learner improves convergence speed, and thus improves sample efficiency. Furthermore we showed that transferring the memory of learning rates to new optimization tasks further improves learning speed. In the following we report new experiments further analyzing the convergence improvements on challenging machine learning tasks such as optimizing CNNs, RNNs and GANs, to illustrate the generality of our meta-learning approach. All experiments are performed with at least 5 different random seeds and with a single set of hyper-parameters. For more extensive hyper-parameter analysis we refer the reader to [8].

3.1 Optimizing Convolutational Neural Networks

Optimizing feed forward neural networks is among the most common learning problems in current research, both for regression and classification tasks. Here, we present new results on a binary mnist classification task which has previously been discussed in more detail in [8]. We choose a sequence of 3 binary classification learning tasks. Task 1 learns to classify digits 1 and 2, task 2 digits 1 and 3 and task 4 digit 1 and 4. Note, that we simply choose the first 4 digits for this experiment, and did not optimize that selection. We use a standard convolutional neural network with rectified linear units and a softmax cross entropy loss. For all meta-learning variants the memories (one per hidden layer) allocate a total of $M = 20$ local models. In initial work [8] we extensively evaluated our meta-learning algorithm on training convergence and showed that our meta-learner increases convergence speed over the base optimizer and compares favorably to [1]. Here we showcase how the classification rate on held out test data behaves. Each binary classification task is trained online - one data point at the time. With each new data sample the neural network is updated, the meta-learner is updated and the test classification rate is evaluated. For the second (1 vs 3) and third (1 vs 4) learning task we can choose to initialize a new meta-learner or reuse the previously trained one and continue to update it. We show the convergence of the test classification results in Figure 1. We compare to stochastic gradient descent and Adam with a base learning rate $\eta = 0.0001$. For all 3 tasks the stochastic gradient descent with meta-learning results in high test accuracy with significantly less observed data points. Furthermore, when reloading the meta-learner for task 2 and 3, sample efficiency is improved even further (see 3 left most plots). On the right hand side of Figure 1 we summarize how many neural network updates (e.g. observed data samples) it took to reach 98% test accuracy, averaged across 5 randomly seeded runs. The meta-learning with re-loading by far outperforms non-meta-learning variants.

3.2 Optimizing Recurrent Neural Networks

Recurrent Neural Networks (RNNs) represent the class of sequence based neural networks which has been becoming very popular in natural language processing/translation, video prediction, and more recently also robotics applications. Despite specific cell designs such as LSTM and GRU, optimizing RNNs is still considered to be a challenging optimization problem.

In the following we will show the superior convergence properties of our proposed meta-learner compared to the most common optimizers such as Adam and SGD. We learn a forward dynamics model of a 7-DoF robot manipulator on a real world dataset consists of our manipulator tracking an object lifting trajectory. We collect position, velocities, accelerations and torques at each time step (1ms), resulting in datasets of more then 3000 samples each. For this learning problem we utilize a standard RNN with LSTM cells to learn the mapping from the 14-dimensional state measurement $(q^t, \dot{q}^t)$and the 7-dimensional torque command τ^t to predict the immediate next state $(q^{t+1}, \dot{q}^{t+1})$. We

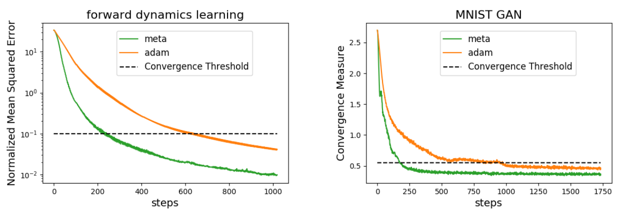

Figure 2: Convergence plots for (left): forward dynamics learning with a RNN on the training dataset (right): generative MNIST model trained with the BEGAN objective. In black we show a convergence threshold which represents acceptable model performance.

optimize on subsequences of 128 steps. Such a learned forward dynamics model can be used for example for model predictive control. Fig. 2 illustrates the normalized mean squared error (NMSE) as a function of computation steps on the left. Our proposed meta-learning based optimizer converges faster which is possible due to the online learning capability of the optimizer.

3.3 Optimizing Generative Adversarial Networks

Goodfellow et al. [5] proposes a novel approach to learn generative models by formulating the problem as a two player game, where a discriminator model attempts to learn to distinguish original data from samples obtained from the generator model and the latter tries to fool the discriminator. This framework has been used extensively in the in vision community and more recently also in robotics to better transfer simulation results to the real world. Nevertheless, GANs are known to be very difficult to optimize. Here we train a very recent GAN optimization objective, Boundary Equilibrium Generative Adversarial Networks [2] to predict MNIST digits. We chose this particular instance since it suggest a convergence measure for their proposed GAN objective. We show the convergence as a function of computation steps in Fig. 2(right). Again the proposed meta-learning based optimizer converges significantly faster relative to the non-meta-optimizer.

4 Discussion and Future Work

In this work we have proposed a new online meta-learning approach that does not only help to transfer information between consecutive learning problems but also requires significantly less computation steps to converge to well performing models. We have evaluated our meta-learning approach on a variety of learning problems, and can consistently showcase the improvement over non-meta-optimizers in terms of convergence properties. Currently our memory of errors is only indexed by the gradient direction itself and we plan to further investigate additional state augmentations. This should enable better task transfer and further stabilize the improved convergence properties of the online meta-learning method. Another very promising avenue to improve the effectiveness of the meta-learner is to use more sophisticated approximations to equation 4.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances In Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] D. Berthelot, T. Schumm, and L. Metz. BEGAN: Boundary Equilibrium Generative Adversarial Networks. *ArXiv e-prints*, March 2017.

[3] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning step size controllers for robust neural network training. In *AAAI*, pages 1519–1525, 2016.

[4] Jie Fu, Zichuan Lin, Miao Liu, Nicholas Leonard, Jiashi Feng, and Tat-Seng Chua. Deep q-networks for accelerating the training of deep neural networks. *arXiv preprint arXiv:1606.01467*, 2016.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, 2014.

[6] Samantha Hansen. Using deep q-learning to control optimization hyperparameters. *arXiv preprint arXiv:1602.04062*, 2016.

[7] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[8] F. Meier, D. Kappler, and S. Schaal. Online Learning of a Memory for Learning Rates. *ArXiv e-prints*, 2017.

[9] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In *IEEE International Conference on Neural Networks*, 1993.

[10] Stefan Schaal and Christopher G Atkeson. Constructive incremental learning from only local information. *Neural computation*, 10(8), 1998.