Learning Step Size Controllers for Robust Neural Network Training

Christian Daniel
TU Darmstadt
Hochschulstrasse 10
64285 Darmstadt, Germany

Jonathan Taylor
Microsoft Research
21 Station Road
Cambridge, UK

Sebastian Nowozin
Microsoft Research
21 Station Road
Cambridge, UK

Abstract

This paper investigates algorithms to automatically adapt the learning rate of neural networks (NNs). Starting with stochastic gradient descent, a large variety of learning methods has been proposed for the NN setting. However, these methods are usually sensitive to the initial learning rate which has to be chosen by the experimenter. We investigate several features and show how an adaptive controller can adjust the learning rate without prior knowledge of the learning problem at hand.

Introduction

Due to the recent successes of Neural Networks for tasks such as image classification (Krizhevsky, Sutskever, and Hinton 2012) and speech recognition (Hinton et al. 2012), the underlying gradient descent methods used for training have gained a renewed interest by the research community. Adding to the well known stochastic gradient descent and RMSprop methods (Tieleman and Hinton 2012), several new gradient based methods such as Adagrad (Duchi, Hazan, and Singer 2011) or Adadelta (Zeiler 2012) have been proposed. However, most of the proposed methods rely heavily on a good choice of an initial learning rate. Compounding this issue is the fact that the range of good learning rates for one problem is often small compared to the range of good learning rates across different problems, i.e., even an experienced experimenter often has to manually search for good problem-specific learning rates.

A tempting alternative to manually searching for a good learning rate would be to learn a control policy that automatically adjusts the learning rate without further intervention using, for example, reinforcement learning techniques (Sutton and Barto 1998). Unfortunately, the success of learning such a controller from data is likely to depend heavily on the features made available to the learning algorithm. A wide array of reinforcement learning literature has shown the importance of good features in tasks ranging from Tetris (Thiery and Scherrer 2009) to haptile object identification (Kroemer, Lampert, and Peters 2011). Thus, the first step towards applying RL methods to control learning rates is to find good features. Subsequently, the main contributions of this paper are

- Identifying informative features for the automatic control of the learning rate.
- Proposing a learning setup for a controller that automatically adapts the step size of NN training algorithms.
- Showing that the resulting controller generalizes across different tasks and architectures.

Together, these contributions enable robust and efficient training of NNs without the need of manual step size tuning.

Method

The goal of this paper is to develop an adaptive controller for the learning rate used in training algorithms such as Stochastic Gradient Descent (SGD) or RMSprop (Tieleman and Hinton 2012). We start with a general statement of the problem we are aiming to solve.

Problem Statement

We are interested in finding the minimizer

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}} F(\boldsymbol{X}; \boldsymbol{\omega}), \tag{1}$$

where in our case $\boldsymbol{\omega}$ represents the weight vector of the NN and $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ is the set of N training examples (e.g., images and labels). The function $F(\cdot)$ sums over the function values induced by the individual inputs such that

$$F(\boldsymbol{X}; \boldsymbol{\omega}) = \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{x}_i; \boldsymbol{\omega}). \tag{2}$$

In order to find $\boldsymbol{\omega}^*$, we have access to an optimization operator $T(\cdot)$ which yields a weight update vector

$$\Delta\boldsymbol{\omega} = T(\nabla F, \boldsymbol{\rho}, \boldsymbol{\xi}). \tag{3}$$

The optimization operator itself can be any arbitrary training algorithm, such as SGD or RMSprop. The variable $\boldsymbol{\xi}$ defines the open parameters of the training algorithm, such as the learning rate η. The vector $\boldsymbol{\rho}$ is often used to accumulate statistics of the training process and serves as a memory. While most training algorithms are guaranteed to find a locally optimal value for $\boldsymbol{\omega}$ given a sufficiently small value for η and lower bounds for selecting η exist, these bounds are often not practical. The goal of this paper, thus, is to learn

Method	$T(\nabla F, \boldsymbol{\rho}, \boldsymbol{\xi})$	ρ_1	ρ_2
SGD	$-\eta \nabla F$	—	—
Momentum	$-\eta \rho_1$	$\xi_1 \rho_1 + \nabla F$	—
Adagrad[1]	$-\eta \nabla F / \sqrt{\rho_1}$	$\sum (\nabla F)^2$	—
RMSprop[2]	$-\eta \nabla F / \sqrt{\rho_1}$	$\hat{\mathbb{E}}[(\nabla F)^2]$	—
Adam[3]	$-\eta \rho_2 / \sqrt{\rho_1}$	$\hat{\mathbb{E}}[(\nabla F)^2]$	$\hat{\mathbb{E}}[\nabla F]$
Adadelta[4]	$-\eta \nabla F \sqrt{\rho_2/\rho_1}$	$\hat{\mathbb{E}}(\nabla F)^2]$	$\hat{\mathbb{E}}[(\Delta \boldsymbol{\omega})^2]$

Table 1: The most commonly used training methods. The operator $\hat{\mathbb{E}}$ computes the weighted expectation based on discount factors encoded in $\boldsymbol{\xi}$. The step size η is also encoded in $\boldsymbol{\xi}$. Our controller is compatible with any of the described methods. [1](Duchi, Hazan, and Singer 2011), [2](Tieleman and Hinton 2012), [3](Kingma and Ba 2014), [4](Zeiler 2012)

a policy $\pi(\boldsymbol{\xi}|\boldsymbol{\phi})$ which can adapt the open parameters of the optimization operator $T(\cdot)$ based on features $\boldsymbol{\phi}$. The optimal policy

$$\pi^*(\boldsymbol{\xi}|\boldsymbol{\phi}) = \arg\max_{\pi} \iint p(\boldsymbol{\phi}) \pi(\boldsymbol{\xi}|\boldsymbol{\phi}) r(\boldsymbol{\xi}, \boldsymbol{\phi}) \,\mathrm{d}\boldsymbol{\phi}\, \mathrm{d}\boldsymbol{\xi}, \quad (4)$$

maximizes the average reward $r(\boldsymbol{\xi}, \boldsymbol{\phi})$, where $p(\boldsymbol{\phi})$ is the distribution over features.

The Features

The ability to learn effective policies hinges on the quality of the features available. We are interested in finding features that are informative about the current state of the network while generalizing across different tasks and architectures. At the same time, we are constrained by the limits on computational complexity commonly placed upon algorithms used for training NNs. Due to the high dimensionality of the weight space, NNs are most often trained using first order methods such as SGD. Thus, the computational complexity of generating informative features should not exceed the complexity of the training algorithm. Furthermore, the high dimensionality of the weight space also constrains the memory requirements of the training algorithms. The size of large scale NNs used today is often constrained by the amount of physical memory available. Hence, the proposed features will also need to be conservative in their memory requirements.

Similarly to the function value itself, the overall gradient ∇F is composed of the individual gradient values, i.e., $\nabla F = 1/N \sum_{i=1}^{N} \nabla f_i$, where

$$\nabla f_i = \frac{\partial f(\boldsymbol{x}_i; \boldsymbol{\omega})}{\partial \boldsymbol{\omega}}. \quad (5)$$

While we expect that following the negative average gradient $-\nabla F$ will improve the overall function value $F(\cdot)$, it will usually not improve all individual function values $f_i(\cdot)$ by the same amount. Indeed, we can easily imagine cases where individual components will actually deteriorate by following the average gradient. Thus, evaluating the agreement of the individual gradients is likely to yield informative features. Furthermore, we can use the individual gradients to approximate the change in the actual function values. To this effect, we approximate the functions using first order Taylor expansions

$$\tilde{f}(\boldsymbol{x}_i; \boldsymbol{\omega} + \Delta\boldsymbol{\omega}) = f(\boldsymbol{x}_i; \boldsymbol{\omega}) + \nabla f_i^T \Delta\boldsymbol{\omega}. \quad (6)$$

As stated by Equation (3), the update $\Delta\boldsymbol{\omega}$ only depends on the average gradient and not the individual gradient. Thus, Equation (6) yields the approximate function values $f_i(\cdot)$ after following the average gradient ∇F. For the simplest optimization operator – SGD without momentum – this update evaluates to $\Delta\boldsymbol{\omega} = \eta \nabla F$. However, most training algorithms do not only rely on the current gradient, but also on accumulated statistics of previous gradient information. The predictive function values take include the resulting effects and, thus, will be more informative than taking only the individual gradients into account. Based on the predictive function values we are now ready to introduce the features which will allow us to learn an adaptive step size controller.

Predictive change in function value. The first feature we will consider is based on the predicted change in function values $\Delta \tilde{f}_i = \tilde{f}_i - f_i$, where we abbreviated $\tilde{f}_i = \tilde{f}(\boldsymbol{x}_i; \boldsymbol{\omega} + \Delta\boldsymbol{\omega})$. We will be interested in the variance of the improvement of function values, i.e.,

$$\phi_1 = \log\Big(\mathrm{Var}(\Delta \tilde{f}_i)\Big). \quad (7)$$

This variance will not only tell us how much the gradients disagree but, more importantly, how much the individual function values are expected to change, based on all effects of the chosen training algorithm.

Disagreement of function values. The variance of predicted changes in function values $\mathrm{Var}(\Delta \tilde{f}_i)$ can be related to the variance of the current function values $\mathrm{Var}(f_i)$, i.e.,

$$\phi_2 = \log\Big(\mathrm{Var}\left(f(\boldsymbol{x}_i; \boldsymbol{\omega})\right)\Big) \quad (8)$$

In general we expect that in the early stages of learning the variance of function values will be high and, thus, the change in function values will be equally large. As the training of the NN progresses, the individual function values will be funnelled to become more similar.

Mini Batch Setting

The features described above work well in the case of batch training. However, large scale systems are most often trained using mini batches. Mini batch training works by breaking up the training set $\boldsymbol{X}$ into subsets $\tilde{\boldsymbol{X}}_b \subset \boldsymbol{X}$, such that $\cup_{b=1}^{B} \tilde{\boldsymbol{X}}_b = \boldsymbol{X}$. While mini batch training increases the efficiency, it also introduces additional noise in the training process. To counter this additional noise, we propose two measures.

Discounted Average. For every feature, we keep a running average of the observed feature value, i.e.,

$$\hat{\phi}_i \leftarrow \gamma\hat{\phi}_i + (1-\gamma)\phi_i, \tag{9}$$

with a discount factor $\gamma < 1$. Relying on averaged feature values is beneficial in two ways. First, the averaging smoothes over outliers in the feature values, which might otherwise lead to a destabilization of the system due to an overly large reaction of the controller. Furthermore, the averaged feature serves as a form of memory, similar to the momentum term in SGD with momentum. Thus, features observed in the current mini batch will influence the learning process for multiple iterations.

Uncertainty Estimate. While averaging feature values over optimization steps is beneficial due to a reduction of noise, it is often desirable to have an explicit estimate of the noise in the system. Intuitively, we would expect larger step sizes to lead to an ever more unstable, and thus noisy, system. We estimate this noise level for every feature by calculating the variance of the centred feature values. To this effect, we deduct our estimate of the mean feature value $\hat{\phi}_i$ from the observed feature value ϕ_i, i.e.,

$$\hat{\phi}_{K+i} \leftarrow \gamma\hat{\phi}_{K+i} + (1-\gamma)(\phi_i - \hat{\phi}_i)^2, \tag{10}$$

where K is the number of 'base' features.

Computational Complexity & Memory Requirements. As stated above, computational and memory requirements are often hard constraints for large scale learning systems. The proposed features are, thus, designed to have a minimal footprint and their memory requirements are constant in the size of the network. In the naive implementation, the memory requirements for computing the predictive function values $\tilde{f}_i$ are linear in the number of inputs N. However, efficient implementations of the back-prop algorithm usually compute all gradients ∇f_i for one layer in the network first, before computing the gradients for the next layer. Thus, we can simply compute the partial predictive change in function value on a per-layer basis and do not need to store all individual gradients for all layers. Computing the feature requires only a matrix vector multiplication, which introduces minimal overhead.

Learning a Controller

Given access to a set of informative features, we now describe the proposed setting to learn a policy which maximizes the average reward as given in Equation (4). The most general solution to the problem stated above is to solve the delayed reward problem using infinite horizon RL techniques. Unfortunately, this approach poses two problems in the proposed setting. First, the system we are aiming to control will be very noisy, mostly due to the mini batch setting. Second, the problem we are aiming to solve is not Markovian in nature. This means that we cannot hope to have access to features which fully describe the state of the system. At the same time training of NNs exhibits many long-term effects, such that changes in the step size may have large effects later on. Thus, we propose to learn the parameters $\boldsymbol{\theta}$ of a controller $\boldsymbol{\xi} = g(\boldsymbol{\phi};\boldsymbol{\theta})$, such that the described effects are abstracted and the policy $\pi(\boldsymbol{\theta})$ will only have to decide on a parametrization of the controller in the beginning of a training run.

To this effect, we aim to find a distribution $\pi(\boldsymbol{\theta})$, such that the resulting controllers are optimal, i.e,

$$\pi^*(\boldsymbol{\theta}) = \arg\max_{\pi} \int p(\boldsymbol{\phi})\pi(\boldsymbol{\theta})r(g(\boldsymbol{\phi};\boldsymbol{\theta}),\boldsymbol{\phi})\, \mathrm{d}\boldsymbol{\phi}\, \mathrm{d}\boldsymbol{\theta}. \tag{11}$$

In particular, due to the continuous nature of the parameter space, we choose a RL method from the robot learning literature. Relative Entropy Policy Search (REPS) (Peters, Mülling, and Altun 2010) has been shown to work well on high dimensional control tasks of real robots (Daniel et al. 2013) and works well in combination with parametrized controllers. The main insight used in REPS is that consecutive policy updates should remain 'close' to each other. Constraining these updates is realized through a bound on the Kullback-Leibler (KL) divergence of subsequent policies, i.e.,

$$\mathrm{D}_{\mathrm{KL}}\left(\pi(\boldsymbol{\theta})||q(\boldsymbol{\theta})\right) \le \epsilon, \tag{12}$$

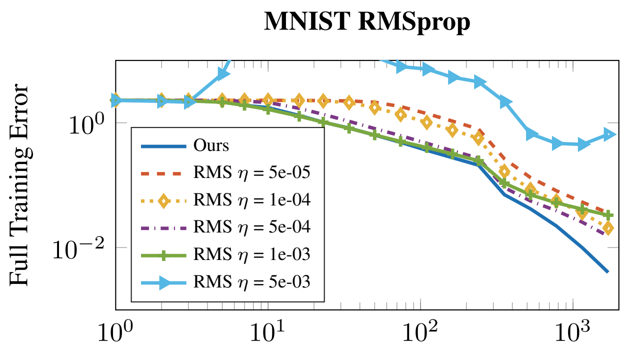
where $q(\boldsymbol{\theta})$ is the previous policy and ϵ is the user-defined bound on the KL. The update rule for the policy $\pi(\boldsymbol{\theta})$ can be obtained by first forming the Lagrangian and then its dual formulation of the optimization problem defined by REPS. The update is given as

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta})\exp(r(\boldsymbol{\theta})/\eta), \tag{13}$$

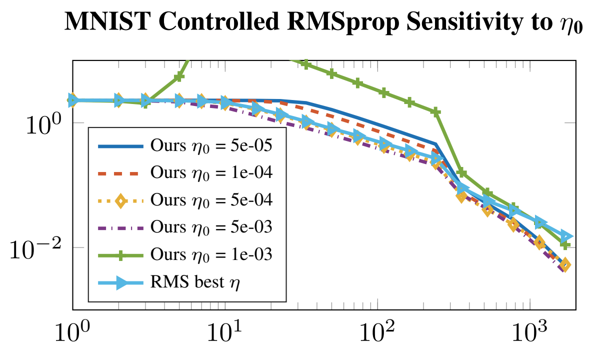
where η is found through the optimization of the dual representation. Given an initial distribution, REPS will iteratively restrict the search space and converge to a locally optimal solution for $\pi(\boldsymbol{\theta})$.

Related Work

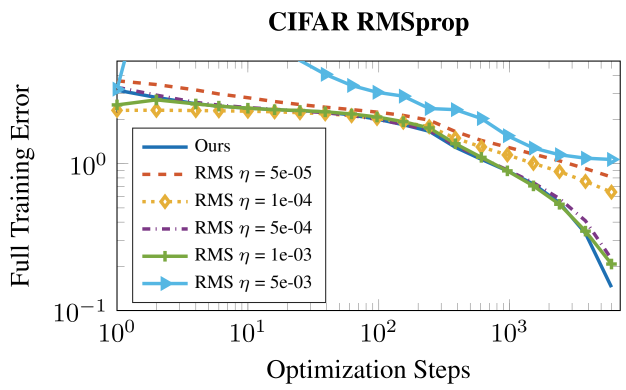
The NN community has proposed multiple training methods based on statistics similar in spirit to the features we propose. In Table 1, we give an overview of the most commonly used methods. One of the first method to make use of additional statistics was SGD with momentum (Jacobs 1988), which keeps a discounted history of past gradients. RMSprop (Tieleman and Hinton 2012) is inspired by RPROP (Riedmiller and Braun 1993), which adapts a learning rate per weight based on the observed sign changes in the gradients. RMSprop uses a discounted history of the squared gradients as a form of preconditioner. It has been argued that this form of preconditioning approximates the diagonal of the Hessian if the changes $\Delta\boldsymbol{\omega}$ are close to being Gaussian distributed (Dauphin et al. 2015). Adagrad (Duchi, Hazan, and Singer 2011) replaces the discounted history over squared gradients with the sum over squared gradients. Using the sum instead of the gradient can be problematic since it may decrease the effective step sizes too fast if the gradients remain large over multiple iterations. Adam (Kingma and Ba 2014) follows RMSprop in discounting the preconditioner and additionally averages over



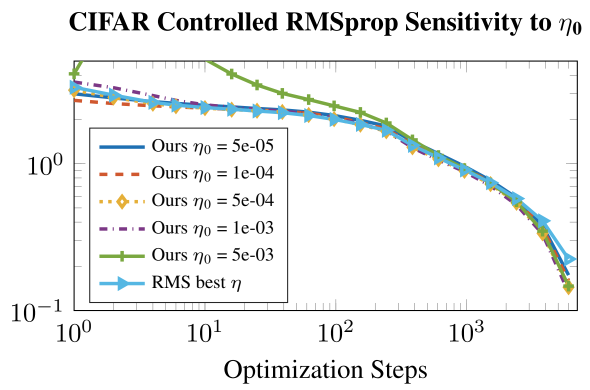
(a) Sensitivity analysis of static step sizes on MNIST.



(b) Sensitivity analysis of the proposed approach on MNIST.



(c) Sensitivity analysis of static step sizes on CIFAR.

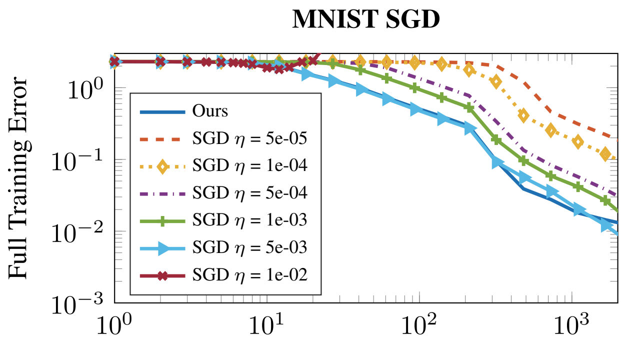


(d) Sensitivity analysis of the proposed approach on CIFAR.

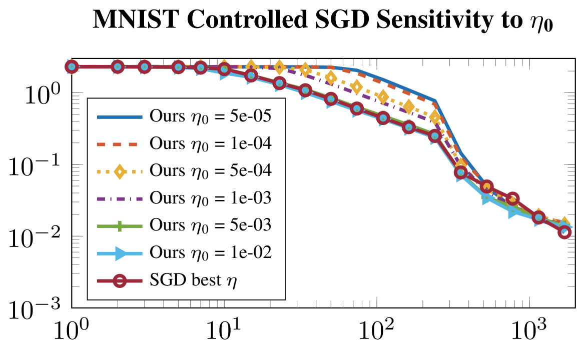
Figure 1: Evaluation of the proposed approach in combination with the RMSprop training algorithm. (a) The RMSprop algorithm is sensitive to the choice of the step size. The proposed approach of controlling step size outperforms the best static step size. (b) The proposed approach is not sensitive to the choice of initial step size. Asympotic performance of the proposed approach is better than the best static step size in all cases. In early iterations a poor choice of η_0 decreases performace while the controller is adapting the step size. (c,d) The results on the CIFAR data set confirm the results of the experiments on the MNIST data set.

the discounted average of the gradients ∇F. The discounting scheme in Adam is more complex than in the other methods as the discount factor itself is decayed over time. Adadelta (Zeiler 2012) also follows RMSprop in the use of the preconditioner but introduces an additional statistic, i.e., the expected squared change of weights $\hat{\mathbb{E}}[\Delta\boldsymbol{\omega}^2]$. This statistic rescales the effective step size proportionally to the history of effective step sizes.

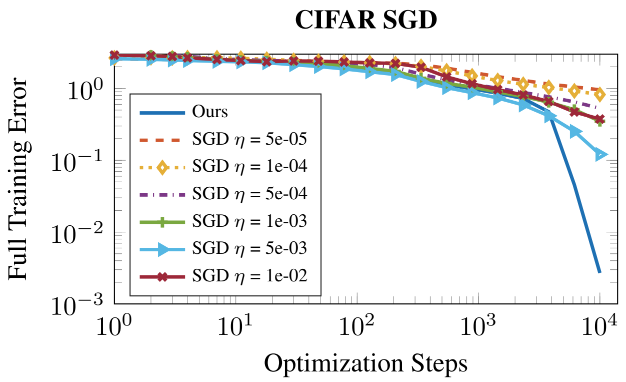
Since setting a static learning rate for a whole training run is often insufficient, popular approaches usually start by finding a good initial learning rate either manually, or, for example, through Bayesian Optimization (Snoek, Larochelle, and Adams 2012). Subsequently, the learning rate is decreased following a predefined scheme (Senior et al. 2013). Possible schemes include the 'waterfall' scheme (Senior et al. 2013), which keeps η constant for multiple steps and then reduces it by large amounts, as well as the exponential scheme (Sutton 1992) and power scheduling (Darken and Moody 1990). However, all of these methods require the practitioner to define additional open parameters. Schaul et al. (Schaul, Zhang, and LeCun 2013) showed how for the SGD method good learning rates can be chosen under the assumption that the optimal weights $\boldsymbol{\omega}_i$ for the instances $\boldsymbol{x}_i$ are normal distributed and access to the second order derivative is available. Using these assumptions they show how learning rates can be set either globally or on a per-weight basis. While our proposed approach is based on global learning rates, we are not restricted to specific methods. TONGA (Le Roux, Bengio, and Fitzgibbon 2012) models the uncertainty in the weight update by building a model of the gradients. This approach is closely related to the idea of the natural gradient (Amari 1998), which has also been applied to neural networks (Bastian, Gunther, and Moon 2011). However, these methods are expensive in terms of computational requirements. Finally, the proposed method is compatible with distributed approaches (Zhang, Duchi, and Wainwright 2012).



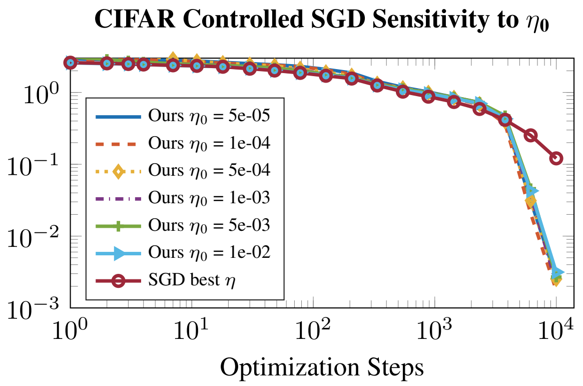
(a) Sensitivity analysis of static step sizes on MNIST.



(b) Sensitivity analysis of the proposed approach on MNIST.



(c) Sensitivity analysis of static step sizes on CIFAR.



(d) Sensitivity analysis of the proposed approach on CIFAR.

Figure 2: Evaluation of the proposed approach in combination with the SGD training algorithm with momentum. (a,b) The results show that while choosing a static step size has a large influence on the training performance, the learned controller has very similar asymptotic performance for all choices of η_0. (c,d) The results on the CIFAR data set confirm the results of the MNIST experiment. The proposed approach in not sensitive to the choice of η_0 and outperforms the static step size.

Experiments

We evaluated the proposed features and the learning approach on two settings, the MNIST hand written digit classification task and the CIFAR-10 image classification task. Furthermore, we evaluated the proposed features across two learning algorithms, SGD and RMSprop. We aim to show that both the features and the controller generalize across problems. Thus, instead of learning the best possible controller for a given setup, we learned the parameters of the controller on small convolutional networks using only a subset of the MNIST data set. We called this smaller data set MNIST-Small, which contained half the MNIST data set.

The Training Network. To learn a controller that generalizes, we randomly sampled networks and data sets during learning. Specifically, we fixed the architecture to a convolutional neural network of the structure c-p-c-p-c-r-c-s, where {c, p, r, s} represented convolutional, pooling, rectified linear and softmax layers, respectively. Within this structure we randomly sampled the number of convolutional filters between [2 5 50] and [10 25 250] per layer. Similarly, we randomly sampled the ratio of the MNIST-Small data set we were training on between 0.1 and 0.5 of the full MNIST data set, where we always sampled from only one half of the full data set, such that during training the algorithms never saw the second half. We trained the controller exclusively on this smaller network and the MNIST-Small data set. Finally, we randomly sampled the number of optimization steps the NN was trained for between 300 steps and 1000 steps.

In this work we considered two widely used training algorithms: SGD with momentum and RMSprop and set the remaining open parameters to the values recommended in the respective literature. For SGD, the momentum was set to 0.9. For RMSprop, the discount factor was set to 0.9 and $\epsilon = 10^{-6}$. The setup was robust to restarts with different random seeds. For example, Fig. 1d shows that even with different initial values for η_0, the performance of the proposed approach was close to identical. Thus, for the presented experiments, we only performed a single run per setting.

The Controller. To learn the controller, we initialized the policy $\pi(\boldsymbol{\theta})$ to a Gaussian with isotropic covariance. In each iteration of learning, we randomly sampled a parameter vector from $\pi(\boldsymbol{\theta})$ as well as a network and a training set. Each

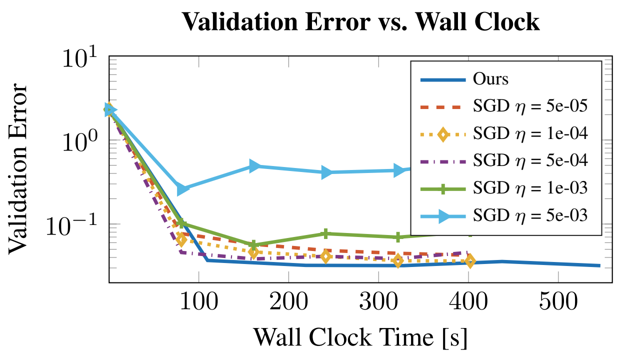


Figure 3: Evaluation of the validation error, adjusted for the computational overhead. Results show the RMSprop training method on the MNIST data set.

iteration of learning was based on 20 samples. The open parameter in REPS was set to $\epsilon = 1$. The parametrized controller was given as $g(\hat{\boldsymbol{\phi}}; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta}^T \hat{\boldsymbol{\phi}})$. The reward function used for learning the controller was given as

$$r = -\frac{1}{S-1} \sum_{s=2}^{S} \Big(\log(E_s) - \log(E_{s-1}) \Big), \qquad (14)$$

where S was the total number of optimization steps for a given run. For all experiments we set the discount factor in Equation (10) to $\gamma = 0.9$. Changing this value did not have a large effect in our experiments within a range of $[0.5, 0.99]$.

Results. We evaluated the proposed methods, controlled SGD-Momentum and controlled RMSprop, on the MNIST and CIFAR data sets. The results show that the proposed controller generalized over different data sets and network architectures. As could be expected, the same controller parameters did, however, not generalize over different training methods. The respective architectures were given as c-p-c-p-c-r-c-s, with [20 50 500] filters per convolutional layer for MNIST and c-p-r-c-r-p-c-r-p-c-r-c-s with [32, 32, 64, 64] for CIFAR. We based our implementation on the MatConvNet toolbox (Vedaldi and Lenc 2014). We evaluated the methods up to the point where the validation error increased. Evaluating the proposed method for even more steps would have lead to bigger advantages for our methods, however, at that point the networks were usually over-fitting to the training data. In our experiments, we observed a computational overhead of 36%, mostly due to the computation of $\tilde{f}(\boldsymbol{x}_i; \boldsymbol{\omega} + \Delta\boldsymbol{\omega})$ in Eq. (6). However, our implementation was not fully optimized and could be improved. This overhead did not include the initial training time of the RL algorithm, which took about six hours. However, as the experiments show, this initial training has to be performed only once and the resulting controller can be re-used for different problems.

Controlled RMSprop. We started by evaluating the controlled RMSprop training method to a range of manually chosen static step sizes. The learned controller balanced the variance in predicted change in function value with the variance of observed function values. The controller also learned to decrease the step size proportionally to the uncertainty estimate in Equation (10). The learned parameters are given as $\boldsymbol{\theta} = [1.62, -0.35, 0.3, -0.33, -0.03]^T * 10^{-2}$, with $\boldsymbol{\phi} = [1, \phi_1, \dots, \phi_4]^T$. The results in Fig. 1a show that the best static step size seemed to be $\eta = 10^{-4}$ and that the static method was sensitive to this choice. Furthermore, the results show that the proposed controller outperformed the best static step size.

After observing that RMSprop is sensitive to the choice of η, we evaluated the sensitivity of the controlled RMSprop to the best choice of a static step size. The results in Fig. 1b show that the proposed method was less sensitive to the choice of initial step size. The results also show that the proposed method had better asymptotic performance than the best static step size in all settings.

We repeated the same evaluation with the same controller on the CIFAR data set. The results show an advantage for the proposed method similar to the results reported above. The results in Fig. 1c show that the static method was sensitive to the choice of a step size, while the results in Fig. 1d show that the proposed method was robust to the choice of initial step size.

To evaluate the robustness of the learned controller to its parameter values, we varied all parameters within 20% of the parameters, i.e., we tried all parameters in a 3x3x3 cube around the learned values. The results in Fig. 4 show that the controller is robust to these small changes in the parameters. Finally, in Fig. 3 we evaluated the validation error, taking into account the computational overhead of our method. The results show that while the total runtime for our method is longer, it yields the best validation error in most time steps.

Controlled SGD-Momentum. The controllers learned for RMSprop largely followed our intuition in that they balanced the predictive change in function value in Equation (7) with the variance of observed function values in Equation (8). Furthermore, the learned controllers decreased η proportionally to the noise levels estimated by the uncertainty estimates in Equation (10). Thus, we could reduce the search space for the SGD experiments by combining the respective features. In particular, the feature vector became $\boldsymbol{\phi} = [1, \hat{\phi}_1 - \hat{\phi}_2, \hat{\phi}_3 + \hat{\phi}_4]^T$ and the learned parameters were $\boldsymbol{\theta} = [-4.55 - 2.87 - 2.52] * 10^{-2}$. The results in Figures 2a to 2d are similar to the results reported for RMSprop. In all experiments the static step size approach seems to be sensitive to the choice of η, while the proposed approach outperforms the best static step size and is less sensitive to the initial choice of η_0.

Discussion & Conclusion

We presented a set of features suitable to learn a controller for the step size of NN training algorithms. The results show that the learned controller generalized to larger networks and the full MNIST data set, as well as to a different architecture and the CIFAR data set. The learned controller

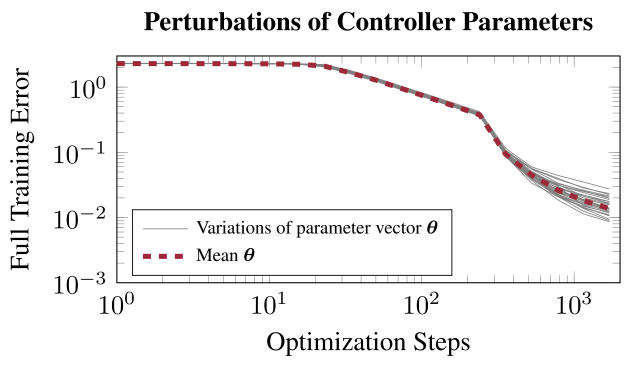


Figure 4: Evaluation of the robustness of the controller parameters on the MNIST data set.

is also robust to initial values of the learning rate. However, the learned controller does not generalize over different training algorithms. Using the presented methodology, it is straight-forward to generalize commonly used methods by treating their specific accumulated statistics as features. Given enough computational resources, future work should investigate learning a controller on this accumulated feature set and possibly combine the best of all worlds.

References

Amari, S. 1998. Natural gradient works efficiently in learning. *Neural computation.*

Bastian, M.; Gunther, J.; and Moon, T. 2011. A simplified natural gradient learning algorithm. *Advances in Artificial Neural Systems.*

Daniel, C.; Neumann, G.; Kroemer, O.; and Peters, J. 2013. Learning sequential motor tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

Darken, C., and Moody, J. 1990. Fast adaptive k-means clustering: some empirical results. *International Joint Conference on Neural Networks (IJCNN).*

Dauphin, Y. N.; de Vries, H.; Chung, J.; and Bengio, Y. 2015. RMSProp and equilibrated adaptive learning rates for non-convex optimization. *arXiv preprint arXiv:1502.04390.*

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research (JMLR)* 12:2121–2159.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; and Others. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

Jacobs, R. 1988. Increased rates of convergence through learning rate adaptation. *Neural networks.*

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems (NIPS).*

Kroemer, O.; Lampert, C.; and Peters, J. 2011. Learning Dynamic Tactile Sensing with Robust Vision-based Training. *IEEE Transactions on Robotics (T-Ro)* (3):545–557.

Le Roux, N.; Bengio, Y.; and Fitzgibbon, A. 2012. Improving First and Second-Order Methods by Modeling Uncertainty. *Optimization for Machine Learning* 403.

Peters, J.; Mülling, K.; and Altun, Y. 2010. Relative Entropy Policy Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI).*

Riedmiller, M., and Braun, H. 1993. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Neural Networks.*

Schaul, T.; Zhang, S.; and LeCun, Y. 2013. No More Pesky Learning Rates. *International Conference on Machine Learning (ICML).*

Senior, A.; Heigold, G.; Ranzato, M.; and Yang, K. 2013. An empirical study of learning rates in deep neural networks for speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).*

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS).*

Sutton, R., and Barto, A. 1998. *Reinforcement learning: An introduction.* MIT press Cambridge.

Sutton, R. 1992. Adapting bias by gradient descent: An incremental version of delta-bar-delta. *Proceedings of the National Conference on Artificial Intelligence (AAAI).*

Thiery, C., and Scherrer, B. 2009. Building controllers for Tetris. *International Computer Games Association Journal.*

Tieleman, T., and Hinton, G. 2012. Lecture 6.5 RMSprop: Divide the Gradient by The Running Average of its recent magnitude. In *COURSERA: Neural Networks for Machine Learning.*

Vedaldi, A., and Lenc, K. 2014. MatConvNet – Convolutional Neural Networks for MATLAB. *CoRR* abs/1412.4.

Zeiler, M. D. 2012. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701.*

Zhang, Y.; Duchi, J. C.; and Wainwright, M. J. 2012. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems (NIPS).*