Bergische Universität Wuppertal

Fakultät für Mathematik und Naturwissenschaften

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM)

Preprint BUW-IMACM 20/39

Malena Reiners, Kathrin Klamroth and Michael Stiglmayr

Efficient and Sparse Neural Networks by Pruning Weights in a Multiobjective Learning Approach

September 23, 2020

http://www.imacm.uni-wuppertal.de

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Efficient and Sparse Neural Networks by Pruning Weights in a Multiobjective Learning Approach

Malena Reiners[a], Kathrin Klamroth[a], Michael Stiglmayr[a]

[a]*School of Mathematics and Natural Sciences, University of Wuppertal, Germany*
{reiners, klamroth, stiglmayr}@math.uni-wuppertal.de

Abstract

Overparameterization and overfitting are common concerns when designing and training deep neural networks, that are often counteracted by pruning and regularization strategies. However, these strategies remain secondary to most learning approaches and suffer from time and computational intensive procedures. We suggest a multiobjective perspective on the training of neural networks by treating its *prediction accuracy* and the *network complexity* as two individual objective functions in a biobjective optimization problem. As a showcase example, we use the cross entropy as a measure of the prediction accuracy while adopting an l_1-penalty function to assess the total cost (or complexity) of the network parameters. The latter is combined with an intra-training pruning approach that reinforces complexity reduction and requires only marginal extra computational cost. From the perspective of multiobjective optimization, this is a truly large-scale optimization problem. We compare two different optimization paradigms: On the one hand, we adopt a scalarization-based approach that transforms the biobjective problem into a series of weighted-sum scalarizations. On the other hand we implement stochastic multi-gradient descent algorithms that generate a single Pareto optimal solution without requiring or using preference information. In the first case, favorable knee solutions are identified by repeated training runs with adaptively selected scalarization parameters. Preliminary numerical results on exemplary convolutional neural networks confirm that large reductions in the complexity of neural networks with neglibile loss of accuracy are possible.

Keywords: multiobjective learning, unstructured pruning, stochastic multi-gradient descent, l_1-regularization, automated machine learning

1. Introduction

1.1. Motivation

Deep neural networks (DNNs) use an immense number of parameters and therefore require powerful computer hardware during operation, and even more so for the initial training of the network. As a consequence, they become impractical when only limited hardware resources and/or time are available. Furthermore, with a large number of weights and nodes there is an increasing risk of overfitting. In addition to the neural network's parameters, many other components and hyperparameters are to be selected and fine-tuned when designing and training neural networks, e.g., the type of layers, the batch size, the amount of dropout or the regularization and the learning rate. This pre-training processes also account for a large part of extra costs of training neural networks, which increases with its number. Neural network training thus has to compromise between at least two conflicting goals: On one hand, the *prediction accuracy* should be as high as possible (which usually asks for many network parameters and sophisticated hyperparameter settings), while on the other hand, the *network complexity* should be as low as possible and the training should require a minimum number of parameters. *Network pruning* methods are an effective approach to limit overparameterization of DNNs and to reduce the complexity of the network. Pruning removes edges (weights), nodes

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

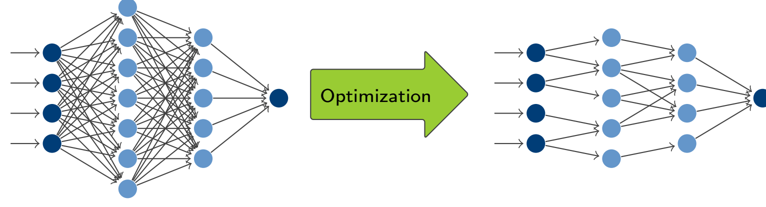


Figure 1: Towards efficient and sparse network architectures.

(neurons) or even feature maps (filters) from the network according to their importance. Figure 1 illustrates the desired transformation from highly complex and dense networks to *efficient and sparse* network designs. Removing parts of the network reduces the required storage capacity and speeds up the inferece time of the network (response time). Compared to initializing the search already with small networks, training and pruning of complex network structures often yields a better performance. This is reported in several recent studies, see, e.g., Li et al. [34] and He et al. [23]. For the success of network pruning the hyperparameter setup, including how much regularization is applied, plays a crucial role. To shorten the time-consuming search for promising hyperparameter values in general, automated recognition algorithms are suggested, e.g., by Domhan et al. [11]. As pruning can be seen as a learning of the network architecture used for the actual learning process, it also relates to the concept of *metalearning*. The following section reviews the related literature.

In this paper, we take a *biobjective* perspective on network training and consider the network complexity as an equitable objective function rather than as a regularization term. We argue that this is the basis for a thorough analysis of the *trade-off* between prediction accuracy on one hand and network complexity on the other hand, and it supports the selection of preferable network architectures. We use the *cross entropy* as a classical *loss function* to model the prediction accuracy. In order to minimize network complexity, we assess the relevance of individual weights in dense layers and apply pruning *during* the actual training procedure. Since the relevance of individual weights has been shown to be correlated to their absolute values, see, e.g., Mummadi et al. [40], we use the l_1-norm of the weight vector as an efficiently computable second training objective. Our goal is to guide the search towards weight values that are closer to zero. At fixed intervals during the training, weights that fall below a threshold value are tentatively pruned in order to strengthen the regularizing effect of the l_1-objective function.

In order to analyze the pros and cons of this multiobjective perspective on neural network training, we adopt and combine a variety of different strategies commonly used in multiobjective optimization and/or machine learning. Crucial components are (i) the implementation of the second, complexity-related objective function, and (ii) the approximation of the most promising Pareto optimal solutions of the biobjective problem. While (i) involves a sophisticated combination of complexity assessment (realized through an l_1-objective) and pruning (realized by an intra-training strategy), we adopt both a weighted sum approach and a variant of a stochastic multi-gradient descent algorithm for (ii). In the weighted sum approach we particularly analyze the influence of the constant weighting parameter. Thereby we are interested in those weightings of the two objectives for which a knee solution is found.

The actual training is implemented based on variants of a single-objective *stochastic gradient descent* (SGD) algorithm (see Robbins and Monro [43] for an early reference and Bottou [5] for its application in machine learning) and of its multiobjective counterpart, a *stochastic multi-gradient descent* (SMGD) algorithm suggested by Liu and Vicente [35]. We compare vanilla implementations of SGD and SMGD with their respective variants including adaptive moment estimation (Adam) by Kingma and Ba [28] and root mean square propagation (RMSProp) by Tieleman and Hinton [47], respectively. A short review of these optimizers is given in Section 3. All optimizers are compared and analyzed in relation to the chosen learning rate and other hyperparameters used. To demonstrate the efficiency of our intra-training pruning strategy, we limit ourselves to image classification

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

problems and focus on the structural design of dense layers in *convolutional neural networks* (CNNs). The main advantage of this novel approach can be seen in the fact that the network quality and the network architecture are optimized consistently and simultaneously. This is highly efficient and saves significant amounts of time and computing costs. Our pruning strategy does not require any fine tuning before or after the training while reaching a reduction of almost 99% in nonzero weights, for example, in a dense layer of the CNN model for training on the image classification dataset CIFAR10 by Krizhevsky [29]. The resulting networks are sparse and can be evaluated efficiently. They are thus suitable for different types of limited hardware devices including, e.g., embedded systems. To the best of our knowledge, these are the first results using SMGD for a Keras [8] neural network architecture which considers regularization as a second objective function. It is based on custom implementations[1] adapted from the Keras SGD, Adam and RMSProp methods and the SMGD algorithm suggested by Liu and Vicente [35]. Furthermore, the combination of pruning with a biobjective training approach compromising between loss and regularization has not been investigated before.

1.2. Related Research

Many methods have been developed to deal with the problem of overparameterization in DNNs and in particular CNNs. Examples include weight quantization [3, 25], knowledge distillation [24, 46] and network pruning [2, 19]. Network pruning has received by far the most attention and can be traced back to the 1990s when optimal brain damage [33] and optimal brain surgeon [21] were investigated. Both use second derivative information to assess the increase in the loss function of the network when a single weight is set to zero. In this way, the Hessian of the loss function is used to guide weight pruning, see also [36] for a discussion.

Further published methods are often based on complex *pruning criteria* which decide which weight, neuron or filter is set to zero. Examples are the Hessian of the loss function used in LeCun et al. [31], the first and second-order Taylor expansions needed in Molchanov et al. [39] or the entropy calculated in Ng et al. [41]. Their evaluation requires a significant part of the additional calculation costs, most of them at least in the order of a gradient computation as analyzed by Yeom et al. [52]. Consequently, the effort to find the most relevant connections in the network before or after the actual training is enormous, not to mention the often time-consuming search for suitable hyperparameter settings. This is in contrast to the paradigm of automatic network architecture design, which is denoted as automated machine learning (*AutoML*). Simpler pruning criteria for individual weights have been established which include approaches based on the l_1-norm of weights, see, e.g., [19, 50].

Pruning methods can be categorized w.r.t. two different pruning strategies. In *structured pruning* neurons or entire filters (i.e., complete substructures of the network) are removed, see, e.g., [22, 30, 40]. In contrast to this, *unstructured pruning* aims at removing individual weights that are labeled as less relevant, see, e.g., [2, 50, 53, 54]. The two approaches usually lead to quite different types of weights: While in the first case, pruning neurons or feature maps implies that complete tensors are dropped, resulting in dense weight vectors of reduced dimension, pruning single edges in the latter case leads to sparse weight vectors that retain the original dimension.

Most of the approaches use pruning in a-priori learning strategies that aim at finding a best possible starting architecture. This is called the *winning ticket* in [14]. It can also be integrated during or after the training, some approaches even prune and retrain the weights many times [20]. In most cases, weights or neurons which have been set to zero once during the training will be excluded from further consideration, which is referred to as *hard pruning* [20]. A *soft pruning* approach is suggested, e.g., in [17] where pruned weights might exceed a given threshold value later during the training and are hence reconsidered during the remaining training steps. It can be observed that many pruning approaches increase the overall training time as a result of costly a-priori training and/or a-posteriori

[1] The python code to follow up and reproduce our results can be found at: https://github.com/malena1906/Pruning-Weights-with-Biobjective-Optimization-Keras

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

fine tuning, see, e.g., [14, 20]. Moreover, recent papers question the efficiency of pruning methods and disprove the efficiency of the winning ticket architecture by varying the learning rate in structured pruning methods, see, e.g., [36]. Other authors interrelate network pruning with adversarial attacks [16, 51]. It was empirically shown in [18] that sparse models resulting from unstructured pruning lead to a higher robustness against adversarial attacks. Common to all approaches is that the relative importance of the pruning criterion has to be set manually depending on the given training data, the network architecture and the remaining hyperparameters.
Considering the network complexity as an equitable second optimization goal in a biobjective optimization model (rather than as a regularization term) is not new, see, e.g., [26, 27, 42, 48] and [10], in which evolutionary algorithms are used. However, these methods usually use time-consuming a priori procedures and are not yet well established in the machine learning community. In order to improve the performance of multiobjective training approaches we suggest to use a simple and efficient l_1-objective as a second optimization criterion that guides the pruning process. A similar approach was suggested in [37], where pruning is an integral part of the network training, however, without considering regularization as a method to guide the pruning process. While we aim at reducing both the cost of training and of inference, [37] focuses primarily on reducing the cost of the training. Integrating pruning in the training process is also suggested for structured pruning by [49]. To the best of our knowledge, [35] is the only reference that extends SGD to multiobjective optimization. However, their SMGD algorithm has not yet been used for a biobjective neural network training with conceptually different objective functions, nor for training in a Keras and Tensorflow environment.

1.3. Contribution and Structure of the Paper

The main contributions of this paper are (i) a consistent multiobjective perspective on two conflicting training goals in machine learning, (ii) a thorough review, implementation and testing of stochastic gradient descent algorithms in the context of multiobjective optimization, (iii) an intra-training pruning strategy to reinforce the reduction of the complexity of neural networks, and (iv) a novel stochastic dichotomic search approach to approximate favorable knee solutions on the Pareto front that is tailored for problems with fundamentally different objectives and when scalarized subproblems can not be guaranteed to yield globally optimal solutions.
The remainder of this paper is organized as follows: In Sections 2 and 3 we present the general modelling assumptions and the theoretical background. In Section 2 we take a multiobjective view on the trade-off between prediction accuracy and network complexity, and in Section 3 the underlying methods and optimization techniques from machine learning are reviewed. Novel intra-training pruning strategies that are based on the biobjective re-interpretation of the regularized loss function are presented in Section 4. Moreover, approaches for finding knee solutions of the Pareto front of the biobjective model are discussed in Section 5. In Section 6 our numerical experiments are presented. All experiments are based on two widely used network architectures and datasets from image classification, namely MNIST [32] and CIFAR10 [29]. We document the experiments for three different pruning methods and compare the success of pruning in dependence of different optimizers. The advantages and disadvantages of the stochastic multi-gradient optimization methods are illustrated and different approaches for approximating the Pareto front are compared. Finally, we summarize our results in Section 7 and mention future research directions.

2. A Multiobjective View on Training Neural Networks

Classical objective (loss) functions in machine learning, for example, the cross entropy loss, the mean squared error or the log likelihood function, solely measure the performance of the neural network on the training data and are used to represent the prediction accuracy of the network. Prediction accuracy is defined as the ratio of the number of correct predictions to the total number of predictions. We exemplarily consider the cross entropy loss function for classification problems,

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

which is to be minimized. For an arbitrary but fixed class (i.e., a fixed output category) the cross entropy is given by

$$E_{\mathrm{CE}}(w, y^d) = -\frac{1}{M} \sum_{i \in S^d} y^d(i) \cdot \log(y(i)). \tag{1}$$

Here, S^d denotes the index set of $M := |S^d|$ training samples $(x^d(i), y^d(i))$, $i \in S^d$. Moreover, $y^d(i)$ denotes the given correct binary classification of the i-th sample $x^d(i)$, i.e., $y^d(i) = 1$ if the i-th sample is in the considered class and $y^d(i) = 0$ otherwise. Corresponding to this, $y(i) \in (0, 1)$ is the probability determined by the network for the i-th sample being in the considered class. Assuming a predetermined network structure, these probabilities depend on the decision variables $w \in \mathbb{R}^N$ (weights of the neural network) and that y^d is defined on a probability space (with probability measure independent of w) for which we assume that i.i.d. samples can be observed and generated. We refer to [15] for a more detailed and comprehensive introduction to neural networks.
Typically, DNNs with large degrees of freedom achieve excellent prediction accuracies w.r.t. the training data (denoted by (x^d, y^d)). However, the generalization, i.e., the performance on *test data* (denoted by (x^t, y^t)), is often unsatisfactory due to overfitting. Overfitted models are also prone to adversarial attacks and thus their applicability is limited, see, e.g., [18]. Indeed, we usually do not only aim at an excellent performance on the training data, but rather envisage a versatile network that generalizes well to various datasets whose characteristics are not necessarily known beforehand.
An effective approach to avoid overfitting is to extend the loss function by a regularization term that, for example, penalizes large weights and/or minimizes the number of nonzero weights, thus trying to reduce the network complexity. The regularized objective function can then be written as

$$J^{\mathrm{R}_\lambda}(w, y^d) = E(w, y^d) + \lambda \cdot \Omega(w) \tag{2}$$

with a loss function E and a regularization term Ω that measures the complexity of the network. The regularization hyperparameter $\lambda > 0$ in (2) reflects the relative importance of regularization and is of central importance for the quality of the trained network. Examples for regularization terms are
the weight decay (l_2-regularization)

$$\Omega_{l_2}(w) = \frac{1}{2} \sum_{n=1}^{N} w_n^2, \tag{3}$$

the sum of absolute values of weights (l_1-regularization)

$$\Omega_{l_1}(w) = \sum_{n=1}^{N} |w_n|, \tag{4}$$

and the number of nonzero weights (l_0-regularization)

$$\Omega_{l_0}(w) = \sum_{n=1}^{N} \delta(w_n), \tag{5}$$

with n being an index summing over all weights of the neural network, and $\delta(w_n) = 1$ if and only if $w_n \neq 0$. Observe that the l_0-regularization Ω_{l_0} is only piece-wise differentiable on intervals on which it has a constant value, i.e., its gradient is zero. Consequently, Ω_{l_0} can not directly be included as a regularization term in a gradient-based training approach.
In the following, we re-interpret the loss function and the regularization function as two *independent* training goals. The regularization hyperparameter λ then reflects the *trade-off* between prediction

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

accuracy and network complexity. In other words, we want to *simultaneously* optimize the prediction accuracy *and* the network complexity in a biobjective model

$$\min_{w \in \mathbb{R}^N} J(w, y^d) = (J_1(w, y^d), J_2(w)) = \big(E(w, y^d), \Omega(w)\big). \tag{6}$$

Using mathematical methods of multiobjective optimization now offers a new perspective on neural network training and thus paves the way for an adaptive decision support tool for preferable network architectures.
Fig. 2 illustrates one possible outcome vector in the objective space of problem (6). Assuming that the training data (x^d, y^d) is given and fixed, a weight vector $\bar{w} \in \mathbb{R}^N$ is called *Pareto optimal* if and only if

$$\nexists\, w \in \mathbb{R}^N : \; J(w, y^d) \leqslant J(\bar{w}, y^d),$$

i.e., there is no $w \in \mathbb{R}^N$ such that $E(w, y^d) \leq E(\bar{w}, y^d)$ and $\Omega(w) \leq \Omega(\bar{w})$, where at least one of these two inequalities is strict. The corresponding image $J(\bar{w}, y^d)$ is then called *nondominated*. Thus, Pareto optimal weight vectors are such weight vectors that can not be improved w.r.t. both objective functions, E and Ω, simultaneously. Note that the outcome vector $J(\bar{w}, y^d)$ illustrated in Fig. 2 is nondominated if and only if the cone $J(\bar{w}, y^d) - \mathbb{R}^2_{\geqslant}$ does not contain any other feasible outcome vector. We refer to [12] for a comprehensive introduction to multiobjective optimization.

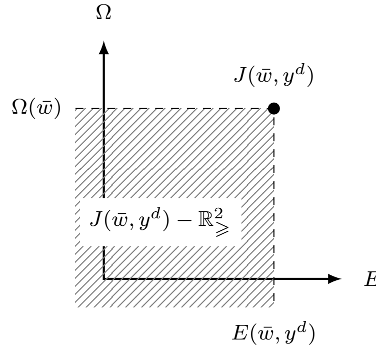


Figure 2: Illustration of the outcome $J(\bar{w}, y^d) \in \mathbb{R}^2$ of a particular weight $\bar{w}$ in the objective space of problem (6).

An approximation of the *Pareto front*, i.e., the set of *all* nondominated outcome vectors in the objective space, provides trade-off information and supports the network developer in selecting a most preferred network architecture. Algorithmically this can be realized, for example, by using evolutionary multiobjective optimization (EMO) algorithms, see e.g., [26], with the disadvantage of possibly very high training costs depending on the size of the network and the training data. A simple and efficient alternative is the successive solution of parametric scalarizations, e.g., the weighted sum scalarization, which is given by an unconstrained single-objective optimization problem

$$\min_{w \in \mathbb{R}^N} J_\lambda(w, y^d) = (1 - \lambda) \cdot E(w, y^d) + \lambda \cdot \Omega(w) \tag{7}$$

with a weighting parameter $\lambda \in [0, 1]$. It is well known that an optimal solution of a weighted sum scalarization (7) with $\lambda \in (0, 1)$ is always Pareto optimal. Under convexity assumptions the weighted sum scalarization also guarantees to generate all Pareto optimal solutions by variation of the weighting parameter $\lambda \in [0, 1]$.
Note that problems (2) and (7) are closely related. Indeed, minimizing the regularized objective function (2) with a fixed regularization hyperparameter $\bar{\lambda} > 0$ is equivalent to minimizing the weighted sum scalarization (7) with weighting parameter $\tilde{\lambda} = \frac{\bar{\lambda}}{1+\bar{\lambda}}$. Conversely, every weighted sum scalarization (7) with a fixed weighting parameter $\tilde{\lambda} \in [0, 1)$ can be equivalently formulated using

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

a regularized objective function (2) with regularization hyperparameter $\bar{\lambda} = \frac{\tilde{\lambda}}{1-\tilde{\lambda}}$, while this is not possible when $\tilde{\lambda} = 1$. However, this difference is not important in the context of neural network training since minimizing $\Omega(w)$ without considering a loss function $E(w, y^d)$ usually leads to collapsed networks (with all weights equal to zero) which is not meaningful. As a consequence, regularization can be interpreted as a particular weighted sum scalarization of the biobjective problem (6), and hence an optimal solution of problem (2) is always Pareto optimal for the biobjective problem (6). We argue that the biobjective perspective supports the analysis of the trade-off between the "original" and the "regularizing" objective and hence supports an informed or even automatic choice of the weighting parameter λ. Moreover, it opens the door for using other scalarizations that can potentially find solutions on non-convex parts of the Pareto front.
One may observe a Pareto front forming a sharp *knee*, a point on the Pareto front, where the trade-off significantly changes, see, e.g., [9]. Such knee points are interesting since they often provide good compromise solutions, i.e., in our case they achieve a relatively high prediction accuracy with a relatively small number of nonzero weights.
In the following, we focus on a particular choice for E and for Ω, namely the cross entropy $E = E_{\text{CE}}$, see (1), and the l_1-regularization $\Omega = \Omega_{l_1}$, see (4). To reduce the number of nonzero weights we combine l_1-regularization with an intra-training pruning strategy which is described in detail in Section 4.
However, the concept is generally applicable and can also be implemented with other choices for the loss function E and the regularization function Ω.

3. Stochastic Gradient Descent Algorithms used in ML

3.1. Stochastic Gradient Descent and Extensions

We first consider the single-objective optimization problem (7) with loss function $J_1(w, y^d) = E(w, y^d)$ and regularization $J_2(w) = \Omega(w)$. The *stochastic gradient descent (SGD)* algorithm, suggested already in 1951 by Robbins and Monro [43], is probably the most popular and widely used optimization algorithm in machine learning. It avoids the high computational cost of computing the true gradient $\nabla_w J_1(w, y^d)$ of the loss function $J_1(w, y^d)$ (which grows with the size of the sample vector y^d) by considering only one – or a *mini-batch* of – randomly chosen sample(s) from y^d to approximate $\nabla_w J_1(w, y^d)$ in each iteration. When only one sample $i \in S^d$ is randomly chosen, then the stochastic gradient of a loss function of the form $J_1(w, y^d) = \frac{1}{M} \sum_{i \in S^d} J_1\big(w, y^d(i)\big)$ is defined as

$$g(w, y^d) = \nabla_w J_1\big(w, y^d(i)\big).$$

Similarly, for a mini-batch, i.e., for a random subset $P \subset S^d$ with batch size $p = |P| \leq M$, it is given by

$$g(w, y^d) = \frac{1}{p} \sum_{i \in P} \nabla_w J_1\big(w, y^d(i)\big). \tag{8}$$

In combination with stochastic gradients the gradient descent algorithm becomes very efficient and useful for large-scale optimization. Note that the regularization objective $J_2(w) = \Omega(w)$ does not depend on the training data, and hence its true gradient $\nabla_w J_2(w)$ can usually be computed efficiently. The use of mini-batches is motivated by the fact that only the expectation of the stochastic gradient is a descent direction for a loss function at a given solution w and the performance of the algorithm is sensitive to the variance of the stochastic gradient. Other variance reduction techniques are discussed, e.g., in [6].
To simplify the notation, we denote the weight vector of the k-th iteration by w^k and the stochastic gradient (8) at w^k by $g^k := g\big(w^k, y^d\big)$. Furthermore, let w^0 denote the (randomly chosen) initial weight vector and let the sequence of learning rates (i.e., step sizes) be given by $(t_k)_{k \in \mathbb{N}}$. A typical stopping criterion in machine learning is the number of *epochs* κ that counts how often each training sample is taken into account. Note that the batch size and the size of the training data hence defines

the number of iterations in each epoch as the random choice in the stochastic gradient computation is implemented without replacement. The complete SGD algorithm is summarized in Algorithm 1 for the minimization of a loss function J_1. An extension to handle weighted sum objectives (7) is immediate. We refer to [44] for a convergence analysis.

Algorithm 1: (Mini-batch) Stochastic Gradient Descent (SGD)

Input: Training objetive J_1, training data S^d, learning rate sequence $(t_k)_{k\in\mathbb{N}}$, random initial weights w^0, stopping criterion, batch size p, $k=0$

Output: Trained model parameters w^*

1 **repeat**
2   $S \leftarrow S^d$;
3   **while** $|S| \geq p$ **do**
4   randomly choose $P \subseteq S$ with $|P| = p$;
5     $g^k \leftarrow \frac{1}{p}\sum_{i\in P} \nabla_w J_1(w^k, y^d(i))$;
6   **for** $n \in \{1, \ldots, N\}$ **do**
7       $w_n^{k+1} \leftarrow w_n^k - t_k \cdot g_n^k$;
8     $k \leftarrow k+1$;
9     $S \leftarrow S \setminus P$;
10 **until** *stopping criterion*;

Most frequently, a constant learning rate $t_k = c \in (0,1]$ (for all k) is used in neural network training. However, it is often advisable to integrate a decreasing (or oscillating) sequence of adjustable learning rates. We refer to a predefined sequence of decreasing learning rates as a *learning rate schedule (LRS)*. The Keras library [8] comes with a time-based LRS which is controlled via a *decay* hyperparameter. Then the learning rate in iteration k is computed using the formula

$$t_k = t_0 \cdot \frac{1}{1 + \mathit{decay} \cdot k}. \tag{9}$$

In addition, the SGD implementation of Keras offers the possibility to use a *momentum* term. In this case it requires an additional momentum hyperparameter M_O. Then the momentum v^k in iteration k is recursively calculated, starting from the initial value $v^0 = 0$, as

$$v^k = v^{k-1} \cdot M_O - t_k \cdot g^k,$$

and the weight update in line 7 of Algorithm 1 changes to

$$w^{k+1} \leftarrow w^k + v^k. \tag{10}$$

SGD with momentum usually converges faster than the standard algorithm. Indeed, the momentum method helps to follow prevalent descent directions and reduces oscillation caused by the variance in the stochastic gradients. Therefore, we can use a higher learning rate when using momentum. A typical choice for the *decay* is 0.01. For the momentum hyperparameter it is recommended to choose $M_O \in [0.5, 0.9]$.
Several variants and extensions of vanilla SGD implementations are available. Popular examples are the *RMSProp* [47] and the *Adam* [28] optimizer. The differences between these variants relate to the calculation of the weight updates which also influence the effect of the learning rates during the training. To better understand their generalization towards solving multiobjective problems, i.e., SMGD algorithms, some more details are described below.
The central idea of *RMSProp* is to use moving averages of the squared components of the stochastic gradients to scale the step lengths, which helps to smoothen the search and again avoids overly large oscillations of the lengths of the actual steps. In this context, the (overall) step size can be interpreted

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

as an adaptive learning rate that changes over time. To implement the RMSProp algorithm in the context of the vanilla SGD algorithm (Algorithm 1), the following line has to be added directly after line 6 of Algorithm 1:

$$\text{mov}^k\big((g_n^k)^2\big) = \beta \cdot \text{mov}^{k-1}\big((g_n^{k-1})^2\big) + (1-\beta) \cdot (g_n^k)^2, \tag{11}$$

where $\beta \in (0,1)$ is a hyperparameter that balances between the current and the previous iteration. Then line 7 of Algorithm 1 changes to

$$w_n^{k+1} \leftarrow w_n^k - \frac{t_k\, g_n^k}{\sqrt{\text{mov}^k((g_n^k)^2)}} \tag{12}$$

Note that using a momentum hyperparameter of M_O in the vanilla SGD actually has a similar effect. The optimization algorithm *Adam* is based on adaptive estimates of first and second order moments of the stochastic gradients. In this way, individual learning rates are calculated for *each* component w_n^k of the weight vector w^k (in iteration k). Adam uses squared components of the stochastic gradients like RMSProp to scale the learning rate and uses momentum with moving averages of the gradient. Consequently, it can be interpreted as a combination of RMSProp and SGD with momentum, and it is is well comparable with SGD when both momentum and learning rate schedules are applied. A specific advantage of Adam is that the magnitudes of the updates of the weights are invariant to a re-scaling of the stochastic gradient. Moreover, the learning rates are limited by the step size hyperparameter which implies an implicit annealing of the learning rate when updating the weights. In the vanilla SGD algorithm (Algorithm 1) this requires the following extensions after line 5:

$$\begin{aligned} m_n^k &= \beta_1 \cdot m_n^{k-1} + (1-\beta_1) \cdot g_n^k \\ v_n^k &= \beta_2 \cdot v_n^{k-1} + (1-\beta_2) \cdot (g_n^k)^2 \\ \hat{m}_n^k &= \frac{m_n^k}{1-(\beta_1)^k} \\ \hat{v}_n^k &= \frac{v_n^k}{1-(\beta_2)^k}, \end{aligned} \tag{13}$$

where $\beta_1, \beta_2 \in (0,1)$ and $\epsilon = 10^{-7}$ (to avoid dividing by zero). Line 7 of Algorithm 1 then changes to

$$w_n^{k+1} \leftarrow w_n^k - t_k \cdot \frac{\hat{m}_n^k}{\sqrt{\hat{v}_n^k} + \epsilon}. \tag{14}$$

We refer to [6] for a more detailed introduction to single objective descent algorithms used for neural network training.

3.2. Stochastic Multi-Gradient Descent and Extensions

Instead of first converting the biobjective problem (6) into one (or a series of) single-objective scalarizations, e.g., using weighted sum scalarizations (7), we now consider a multiobjective gradient descent algorithm for neural network training.

Deterministic multiobjective gradient descent algorithms were first introduced by [13]. Starting from an initial solution, the idea is to iteratively move into directions that are steepest *common* descent directions, and hence descent directions for *all* objective functions. For an unconstrained and continuous problem, the algorithm terminates with a stationary point (a local Pareto optimal solution) when no such direction exists. Multiobjective gradient descent algorithms do not require any preference information and generally converge quickly (under appropriate assumptions) to *one* local Pareto optimal solution. However, guiding the search towards specific regions of the Pareto front without prior knowledge (e.g., towards a knee solution) is generally difficult.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

An extension of this method to problems where, rather than true gradients, only stochastic gradients can be efficiently computed was suggested in [35]. The *stochastic multi-gradient descent* (SMGD) algorithm is based on earlier works by [7] and [38]. Liu and Vicente [35] provide a convergence analysis that is similar to that for the SGD algorithm. They specifically apply the SMGD algorithm for the detection of dependencies in training data sets, but actually not for neural network training itself.
The SMGD algorithm is defined for general multiobjective optimization problems of the form $\min_{w\in\mathbb{R}^N} J(w,y^d) = \big(J_1(w,y^d),\ldots,J_q(w,y^d)\big)$, where $q \geq 2$ denotes the number of objective functions. In our case, we have $q = 2$ and the second objective function $J_2(w,y^d) = J_2(w) = \Omega(w)$ does not depend on the training data. As a consequence, the stochastic gradient $g_2(w,y^d) = \nabla_w J_2(w)$ of J_2 coincides with the true gradient and can be efficiently computed. To keep the notation simple when describing the SMGD algorithm, we will nonetheless refer to both gradients as "stochastic gradients" and adopt the notation $J_j(w,y^d)$ and $g_j(w,y^d)$, $j = 1,2$. Similar to Section 3.1 we abbreviate $g^k = g(w^k,y^d)$ and $g_j^k = g_j(w^k,y^d)$. A common stochastic descent direction in iteration k is then determined as a weighted sum of the individual stochastic gradients

$$g^k = \sum_{j=1}^{q} \lambda_j^k \cdot g_j^k,$$

however, not with fixed weighting parameters as in a weighted sum approach but rather with variable weighting parameters $\lambda^k \in \mathbb{R}^q_{\geqq}$, $\sum_{j=1}^{q} \lambda_j^k = 1$, that are selected such as to obtain a *steepest* common descent direction in each iteration. Note that when $q = 2$ we equivalently set $\lambda_2 = \lambda$ and $\lambda_1 = 1 - \lambda$ with only one weighting parameter $\lambda \in [0,1]$, see (7).
The steepest common descent direction is determined by computing the weighting parameters λ^k in iteration k as an optimal solution of a convex quadratic optimization subproblem, i.e.,

$$\begin{aligned} \lambda^k &= \arg\min \left\| \sum_{j=1}^{q} \lambda_j \, g_j^k \right\|^2 \qquad (15) \\ \text{s.t.} \quad \lambda &\in \Big\{ \lambda \in \mathbb{R}^q \colon \sum_{j=1}^{q} \lambda_j = 1, \lambda_j \geq 0, \ \forall\, j \in \ \{1,\ldots,q\} \Big\}. \end{aligned}$$

For a more detailed derivation and theoretical foundation of stochastic multi-gradient descent we refer to [35].
The vanilla SMGD algorithm formulated in Algorithm 2 can now be extended similar to the vanilla SGD algorithm given in Algorithm 1 in order to improve the training results. We suggest two such variants of SMGD, namely a *multiobjective RMSProp optimizer* (MRMSProp) and a *multiobjective Adam optimizer* (MAdam). Note that in our case the two objective functions $J_1 = E_{\mathrm{CE}}$ and $J_2 = \Omega_{l_1}$ are fundamentally different, which will become apparent in the numerical tests presented in Section 6. To avoid the risk that one of the objectives consistently overrides the other, we hence suggest to compute the moving averages mov_j^k, $j = 1,\ldots,q$ for the MRMSProp algorithm according to (10) *independently* for each of the q objective functions and combine them using the weights λ^k only when updating the weight vector according to (12). Similarly, we suggest to compute the estimates of the first and second moment for the MAdam optimizer according to (13) *independently* for each objective function and combine them using the weights λ^k only for the weight update according to (14). As a consequence, the two goals of minimizing the prediction error and minimizing the network complexity are treated equivalently in our setting.

Preprint – Preprint – Preprint – Preprint – Preprint

Algorithm 2: (Mini-batch) Stochastic Multi-Gradient Descent (SMGD)

Input: Training data S^d, learning rate sequence $(t_k)_{k\in\mathbb{N}}$, random initial weights w^0, stopping criterion, batch size p, set $k = 0$

Output: Trained model parameters w^*

1 **repeat**
2 $S \leftarrow S^d$;
3 **while** $|S| \geq p$ **do**
4 randomly choose $P \subset S$ with $|P| = p$;
5 $g_j^k \leftarrow \frac{1}{p}\sum_{i\in P} \nabla_w J_j(w^k, y^d(i))\ \forall j \in \{1,\dots,q\}$;
6 solve the subproblem (15) to obtain $\lambda^k \in \mathbb{R}^q$;
7 $g^k \leftarrow \sum_{j=1}^{q} \lambda_j^k \cdot g_j^k$;
8 **for** $n \in \{1,\dots,N\}$ **do**
9 $w_n^{k+1} \leftarrow w_n^k - t_k \cdot g_n^k$;
10 $k \leftarrow k + 1$;
11 $S \leftarrow S \setminus P$;
12 **until** *stopping criterion*;

4. Unstructured and Soft Intra-Training Pruning

To reinforce the effect of the second objective function $J_2(w) = \Omega_{l_1}(w)$ descent steps are complemented by an associated *intra-training pruning* (ITP) strategy. Thereby, we focus on the weights of the dense layers of the neural network. Such pruning approaches are referred to as *unstructured* pruning. Weight pruning is motivated by the observation that small weights have only a limited impact on the prediction accuracy of the neural network. By setting all weights to zero as soon as their absolute value falls below a predetermined *threshold value* $\tau > 0$, the corresponding edges are removed from the neural network which helps to reduce the network complexity. In our approach pruning is directly incorporated in the training and is guided by the second objective function $\Omega_{l_1}(w)$. We compare two different variants of the ITP strategy, namely batchwise and epochwise pruning, with after training pruning where only one single pruning step is applied after the neural network is fully trained. The most promising pruning strategy is then selected to be used as a baseline when comparing other algorithmic components. Note that we already start with a sparse initialized weight matrix in all pruning approaches, i.e., there is already one pruning step before the training starts.
In *batchwise* pruning, pruning is applied after each batch, while in *epochwise pruning*, a pruning step is performed after each epoch. In our implementation we use Keras callback functions to update the weight matrices of all dense layers. As ITP always ends with a pruning step after the network is totally trained, these methods can be seen as extensions of after-training pruning. It is important to note that in our approach, in contrast to, e.g., [20], pruned weights are set to zero but dot not necessarily stay fixed at zero. Depending on the subsequent iterations weights may exceed the threshold and become relevant again. Hence, our strategy belongs to the class of *soft* pruning methods. Thereby pruning is an integral part of the training and thus there is no need for a recovery phase as suggested by [55], or other fine tuning methods.
We argue that when using ITP then the network *learns* that weights will be pruned if they become too small. The iterative pruning of small weights leads to an increasing number of zero weights. In general, not all zero weights are restored to values above the threshold level in subsequent iterations since this would significantly deteriorate the l_1-regularization term as stated in [37]. Consequently, a smaller number of non-zero weights has to retain the information. It turns out that to a certain degree this is indeed possible without significantly deteriorating the prediction accuracy. This is confirmed by the numerical experiments presented in Section 6.
A careful choice of the pruning threshold is important when fine-tuning the pruning strategy. Indeed, this hyperparameter also trades-off between prediction accuracy and network complexity. Our

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

tests, however, show that fixed moderate threshold values in combination with varying weighting parameters for the Ω_{l_1} objective function effectively control the number of nonzero weights. While previous approaches search for an appropriate penalty value, making it expensive to include pruning from the beginning of the training, we fix this threshold beforehand. In our tests the threshold value varies between $\tau = 0.001$ and $\tau = 0.002$. Moreover, we observed that the learning rate has a strong impact on the pruning ratio and on the possibility that an individual weight exceeds the given threshold after being once set to zero.

5. Finding Knee Solutions

Knee solutions on the Pareto front are particularly interesting since they often realize a favorable trade-off between the considered objective functions. When training neural networks w.r.t. the two conflicting goals $J_1(w, y^d) = E_{\mathrm{CE}}(w, y^d)$ (measuring prediction accuracy) and $J_2(w) = \Omega_{l_1}(w)$ (measuring network complexity) we observed pronounced knee solutions. Indeed, it turns out that a large number of weights can be set to zero without loosing much prediction accuracy. An automatic detection of such knee solutions is highly desirable, particularly when aiming at AutoML approaches. However, SMGD methods like MAdam and MRMSProp as introduced in Section 3 are designed for a fast convergence towards the Pareto front, no matter where. Preference information can hardly be incorporated so that it is extremely unlikely that the final outcome is close to a knee solution (unless the methods are initialized with weights w^0 that are already close to a knee). This can also be observed in our numerical experiments, see Section 6.
In order to nonetheless approximate knee solutions we develop scalarization-based approximation algorithms, namely a stochastic dichotomic search and a bisection search algorithm. Both methods aim at an efficient and automatic identification of weighting parameters (i.e., trade-offs) of the weighted sum scalarization (7) that lead to knee solutions.
The methods are developed to cope with the following challenges: (i) the (scalarized) optimization problems are truly large-scale, (ii) the two objective functions have fundamentally different characteristics (e.g., w.r.t. complexity, scaling, slope, curvature, and number of local minima), and (iii) the scalarized subproblems can generally not be solved to optimality. Indeed, when repeatedly training the same weighted sum scalarization with a single-objective variant of an SGD algorithm, we usually obtain different solutions that may be of largely differing quality. This motivates the introduction of a stochastic component in the usually deterministic selection of weighting parameters in a dichotomic search algorithm [1]: In Algorithm 3 we suggest to make a random choice for the weighting parameter whenever the algorithm gets stuck which is recognized when weighting parameters are chosen too close to each other.
We compare the stochastic dichotomic search method to a simple yet effective bisection search method on the weighting parameters, see, e.g., [4]: Starting from an initial interval of considered weighting parameters, this interval is successively decomposed into sub-intervals of equal size. Relevant intervals for further decomposition are identified by comparing trade-offs.
We remark that both algorithms suffer from the time intensive training, i.e., the calculation of optimized objective values for fixed weighting parameters, as in each case the neural network needs to be re-trained from scratch. To speed up this process only a reduced number of training epochs is used for the training. When an approximation of the Pareto front has been computed with either of these methods, then a knee solution can be identified by comparing trade-offs.

6. Experiments

6.1. Experimental Setup

We validate our approach on the network architecture LeNet-5 [31], a CNN consisting of two convolutional layers followed by three fully connected dense layers. After each convolutional layer the feature maps are subsampled by an average pooling layer. For details see Tab. 1(a). In all layers,

Preprint – Preprint – Preprint – Preprint – Preprint

Algorithm 3: Stochastic Biobjective Dichotomic Search

Data: Training data S^d, hyperparameter settings for single-objective SGD solver, depth of the search (levels), initial weighting parameters $\lambda_1, \lambda_2 \in [0,1)$, $\lambda_1 < \lambda_2$, to approximate extremal solutions focusing on J_1 and J_2, respectively

Result: Trained model parameters w^* approximating a Pareto knee

```
1  list ← {λ1, λ2};
2  cand ← ∅;
3  for l ∈ levels do
4      for λ ∈ list do
5          train with weighted sum objective J_λ (possibly with reduced epochs) and add objective
             vectors to cand;
6          delete all dominated points in cand;
7      sort cand by second objective function (in increasing order);
8      if l < |levels| then
9          for i ∈ {2, ..., |cand|} do
10             diff ← cand(i) − cand(i − 1);
11             λ_new ← −diff_1 / (diff_2 − diff_1);
12             if ∃ λ̂ ∈ list : |λ̂ − λ_new| ≤ 0.001 then
13                 randomly choose λ̃ ∈ [max{0.9 · λ̂, λ1}, min{1.1 · λ̂, λ2}];
14                 λ_new ← λ̃;
15             add λ_new to list ;
16 identify the Pareto knee in cand by comparing trade-offs;
```

except the output `layer3`, ReLU activation functions are used. In the final dense output `layer3`, a softmax activation is used on the whole layer to ensure that the output represents a probability distribution with $y(i) \in (0,1)$ for each output category.

In total, LeNet-5 consists of about $60,000$ trainable parameters (weights) of which approximately $59,000$ weights belong to the dense layers (`layer1`–`layer3`). We use l_2-regularization (3) in the convolutional layers and l_1-regularization (4) for the dense layers to reduce overfitting. While for the convolutional layer the l_2-regularization coefficient is fixed to $\tilde{\lambda} = 0.0001$, we interpret the l_1-regularization for the dense layers as second objective function and vary the corresponding regularization hyperparameter $\lambda \geq 0$ to analyze its effect on pruning strategies. We train the network on the handwritten digits dataset MNIST [32] which does consist of $60,000$ labeled images defining the training data set (x^d, y^d) and additionally $10,000$ labeled test images, stored as (x^t, y^t). No additional techniques (like dropout, data augmentation or batch normalization) are applied. In some of our tests we additionally apply momentum, learning rate schedules and/or weight decay as described in Section 3.1. We train the model for $\kappa_{\max} = 30$ epochs and use, unless otherwise stated, the Adam optimizer with hyperparameters set to the default values: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ and a constant learning rate of $t_k = 0.001$ for all k.

We also examine the performance of ITP using a more complex network and the CIFAR10 dataset for image classification [29]. The dataset consists of $60,000$ color images from 10 classes, with $6,000$ images per class. The dataset is split into $|S^d| = 50,000$ training data and $|S^t| = 10,000$ test data. The adapted VGG-like architecture [45] is defined by six convolutional layers and one final `denselayer`, see Tab. 1(b). Note that, for simplicity, only the layers including parameters are mentioned in the table. Additionally, there is a max pooling layer after each batch normalization layer. All convolutional layers use *exponential linear units* (ELU) as activation functions, while the final `denselayer` uses softmax activation. In total, the VGG-like network consists of about $300,000$ trainable weights of which about $20,000$ belong to the final `denselayer`. We also use l_2-regularization with fixed coefficient $\tilde{\lambda} = 10^{-6}$ on the convolutional layers and interpret the l_1-regularization on the

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Table 1: LeNet-5 and VGG-like network architecture.

(a) LeNet-5 architecture for MNIST.

Layer (type)	Output Shape	# Param
conv2d_29 (Conv2D)	(None, 26, 26, 6)	60
average_pooling2d_29	(None, 13, 13, 6)	0
conv2d_30 (Conv2D)	(None, 11, 11, 16)	880
average_pooling2d_30	(None, 5, 5, 16)	0
flatten_15 (Flatten)	(None, 400)	0
layer1 (Dense)	(None, 120)	48120
layer2 (Dense)	(None, 84)	10164
layer3 (Dense)	(None, 10)	850
Total params: 60,074		
Trainable params: 60,074		
Non-trainable params: 0		

(b) VGG-like architecture for CIFAR10.

Layer (type)	Output Shape	# Param
conv2d_ 1 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization_1	(None, 32, 32, 32)	128
conv2d_ 2 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_2	(None, 32, 32, 32)	128
conv2d_ 3 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_3	(None, 16, 16, 64)	256
conv2d_ 4 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_4	(None, 16, 16, 64)	256
conv2d_ 5 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_5	(None, 8, 8, 128)	512
conv2d_ 6 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_6	(None, 8, 8, 128)	512
denselayer (Dense)	(None, 10)	20490
Total params: 309,290		
Trainable params: 308,394		
Non-trainable params: 896		

`denselayer` as second objective function and vary the corresponding regularization hyperparameter $\lambda \geq 0$.
Since the CIFAR10 data set is considerably more complex than the MNIST data set we additionally apply classical techniques to get competitive results, including batch normalization after each convolutional layer and max pooling followed by dropout after the convolutional layers 2,4 and 6. Moreover, we use data augmentation in the training process and, unless stated otherwise, train with the RMSProp optimizer with hyperparameters set to the default values $\beta = 0.9$ and $\epsilon = 10^{-7}$ on $\kappa_{\max} = 125$ epochs. The learning rate is varied in different experiments. In particular, learning rate schedules (LRS) are used to reach a higher performance. We develop an LRS that is specifically taylored for our methods and that smoothly reduces the learning rate over the epochs $\kappa \in \mathbb{N}$, $\kappa \in \{1, \ldots, \kappa_{\max}\}$, with a major drop after about 75% of the total number of epochs $\kappa_{\max}$:

$$t(\kappa) = -(t_{\text{start}} - t_{\text{end}}) \frac{\exp\left(\frac{\kappa - 0.75 \cdot \kappa_{\max}}{0.075 \cdot \kappa_{\max}}\right)}{\exp\left(\frac{\kappa - 0.75 \cdot \kappa_{\max}}{0.075 \cdot \kappa_{\max}}\right) + 1} + t_{\text{start}}$$

For CIFAR10 training with $\kappa_{\max} = 125$ epochs, and setting $t_{\text{start}} = 0.001$ and $t_{\text{end}} = 0.0001$, the evolution of the learning rate is shown in Fig. 3. Unless stated otherwise this schedule is used when talking about an LRS.

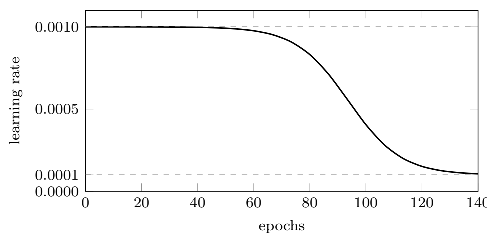


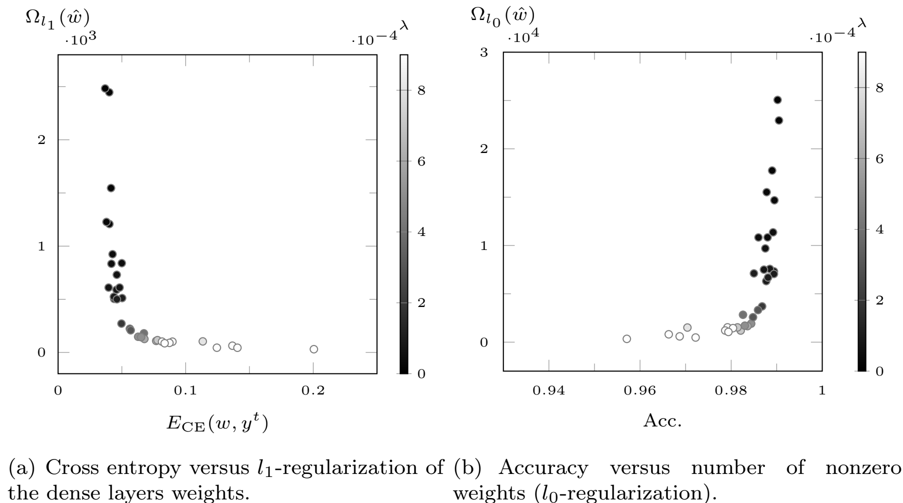
Figure 3: Learning rate schedule reducing the learning rate over the 125 training epochs.

6.2. Experimental Results and Trade-Off Analysis

We first report experimental results for our ITP strategy. To identify the most promising pruning approach, all training runs rely on the standard (single objective) SGD algorithms described in Section 3.1. Subsequently, we present test results on the SMGD algorithms discussed in Section 3.2.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

We observe that the effect of ITP varies epending on the value of the regularization hyperparameter λ in (2). Indeed, the larger the impact of the l_1-regularization is, the more weights are pruned. From a multiobjective perspective we evaluate different hyperparameter settings including the threshold value τ, the influence of the learning rate t and of the regularization hyperparameter λ. Note that, although we use cross entropy (1) and l_1-regularization (4) as objectives (see Fig. 4(a) for an example), we often use the l_0-regularization (5) and the prediction accuracy (see Fig. 4(b)) to indicate the success of ITP. However, both illustrations in Fig. 4 show the same behavior with a pronounced knee in the Pareto front.



(a) Cross entropy versus l_1-regularization of the dense layers weights.

(b) Accuracy versus number of nonzero weights (l_0-regularization).

Figure 4: LeNet-5 on MNIST dataset: Pareto fronts illustrating the trade-off between loss/accuracy and regularization for ITP.

Furthermore, we compare the SMGD algorithms that determine only one approximation of a Pareto optimal solution with scalarization based methods to approximate the Pareto front as introduced in Section 5. Based on these approximations we identify knee solution. With λ^* we refer to a marginal weighting parameter in (7) corresponding to the knee solution. Note that, as the pruning strategy takes only dense layers into account, $\hat{w}$ refers to the weights in the dense layers of a network and, e.g., $\hat{w}^1$ denotes all weights in `denselayer1`.

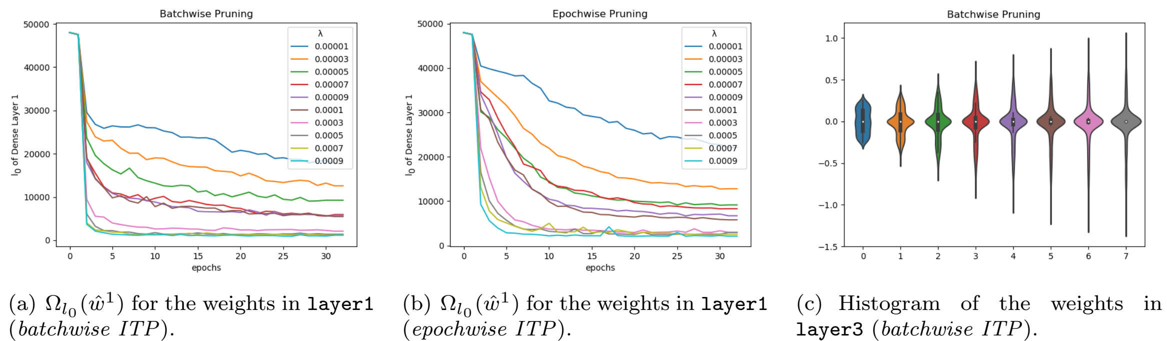
*6.2.1. Test Results for ITP on LeNet-*5

On the LeNet-5 network architecture we compare *batchwise pruning*, i.e., pruning weights below the threshold after each training batch, with *epochwise pruning*, where pruning is only applied at the end of each epoch κ. The results are then compared with *after training pruning*, where pruning is applied only at the end of the training (after all epochs). In all cases we initially prune the weight matrix before starting the training, and the pruning threshold is uniformly set to $\tau = 0.001$.

Our results confirm that the stronger the ITP strategy, i.e., the more frequently pruning is applied, the higher is the potential for reducing the number of nonzero weights while maintaining a high prediction accuracy. This is clearly visible in Fig. 5(a) and Fig. 5(b). It shows for representative training runs and for different values of the regularization hyperparameter λ, how the number of nonzero weights (Ω_{l_0}) decreases over the epochs for batchwise and epochwise pruning, respectively. It can be recognized that batchwise pruning leads to a faster and stronger reduction of the number of nonzero weights. However, this difference decreases with an increasing value of the regularization hyperparameter λ. Furthermore, it can be observed that the higher the value of λ, the earlier the number of nonzero weights converges, see Fig. 5(a).

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

In Fig. 5(c) the distribution of weight values over a training process is shown via violin plots. The eight histograms represent the distribution of weights in `layer3`: how they have been initialized, before the training, after every fifth epoch and after the training (with a regularization hyperparameter of $\lambda = 0.00005$). Note that the l_1-regularization causes the sum of the absolute values of the weights to decrease, and the pruning strategy leads to a large number of weights that are exactly zero. Fig. 5(c) clearly shows that a large majority of weights is relatively close to zero. Actually there are no weights with values in the interval $[-\tau, \tau]$ (here, $\tau = 0.001$), which can't be seen in Fig. 5(c) due to the scaling.



(a) $\Omega_{l_0}(\hat{w}^1)$ for the weights in `layer1` (*batchwise ITP*).

(b) $\Omega_{l_0}(\hat{w}^1)$ for the weights in `layer1` (*epochwise ITP*).

(c) Histogram of the weights in `layer3` (*batchwise ITP*).

Figure 5: LeNet-5 on MNIST dataset: Development of the regularization terms and the distribution of weights over the epochs of the training, in c) over every 5th epoch.

Both Fig. 4(a) and Fig. 4(b) show the trade-off between the respective loss and regularization terms. The results indicate that the number of nonzero weights (Ω_{l_0}), respectively their sum of absolute values (Ω_{l_1}), can be considerably reduced without significantly deteriorating prediction accuracy and cross entropy. Indeed, the prediction accuracy can be maintained at a very high level of more than 98% even if only about 1,250 out of the approximately 48,000 considered `layer1`-weights are nonzero (about 2.6%). Note that similar outcome vectors are obtained for a range of λ-values. For ITP training, for example, excellent results are obtained for regularization hyperparameter values $\lambda \in [0.0001, 0.0007]$, see, for example, Tab. 2(c).

The impact of the regularization hyperparameter λ on the effectiveness of the pruning strategy and on the prediction accuracy is summarized in Tab. 2(a) – (c). The rows show exemplary results of independently performed training runs for different values of the regularization hyperparameter λ. We observe that the prediction accuracy deteriorates only slightly up to a value of $\lambda = 0.0003$ while the number of nonzero weights decreases significantly. For this *critical* value of the regularization hyperparameter, $\lambda^* = 0.0003$, the total number of nonzero weights in all dense layers was reduced in batchwise pruning by about 96% and in epochwise and after training pruning only by about 93%. In all tables the row highlighted in gray corresponds to a critical λ^* beyond which the prediction accuracy deteriorates significantly. An efficient identification of this critical value and of the associated knee solution on the Pareto front is thus highly desirable.

Note that when comparing the three dense layers of LeNet-5 pruning is more effective on layers with a larger number of weights. For example, in epochwise pruning the dense `layer3` was reduced by only about 64% while the larger dense `layer1` and `layer2` were reduced by about 94% to 96%, respectively.

6.2.2. Test Results for ITP on VGG

Additional tests for the CIFAR10 dataset with a VGG-like network architecture confirm the observations reported in Section 6.2.1 on the positive effects of ITP. In the case of batchwise pruning the complexity reductions are even more significant than for MNIST, see Tab. 2(d). Indeed, a reduction

Table 2: The impact of the regularization hyperparameter λ and different pruning strategies.

(a) MNIST on LeNet-5 with *after training* pruning and threshold $\tau = 0.001$.

λ	E_{CE} train	E_{CE} test	Ω_{l_1}	Ω_{l_0} $\hat{w}^1$	Ω_{l_0} $\hat{w}^2$	Ω_{l_0} $\hat{w}^3$	Acc. train	Acc. test
0	0.005	0.069	10 670.627	47794	10045	837	0.998	0.987
$1 \cdot 10^{-5}$	0.010	0.037	2275.125	18188	4474	616	0.998	0.990
$3 \cdot 10^{-5}$	0.015	0.035	1102.737	11424	2444	539	0.997	0.991
$5 \cdot 10^{-5}$	0.029	0.049	795.104	8140	1881	476	0.992	0.987
$7 \cdot 10^{-5}$	0.024	0.041	629.910	8171	1592	412	0.994	0.988
$9 \cdot 10^{-5}$	0.030	0.045	507.737	6086	1222	398	0.993	0.988
$1 \cdot 10^{-4}$	0.028	0.049	499.282	6356	1337	364	0.993	0.987
$3 \cdot 10^{-4}$	0.048	0.049	215.541	3668	459	203	0.988	0.988
$5 \cdot 10^{-4}$	0.065	0.064	158.332	3167	937	341	0.984	0.983
$7 \cdot 10^{-4}$	0.076	0.073	119.347	2399	675	266	0.982	0.982
$9 \cdot 10^{-4}$	0.095	0.097	109.441	2245	308	199	0.976	0.975
0.001	0.095	0.089	106.567	2226	625	251	0.977	0.977
0.003	0.166	0.150	46.759	893	179	150	0.957	0.962
0.005	2.301	2.301	0.000	0	0	0	0.112	0.113
0.007	2.301	2.301	0.000	0	0	3	0.112	0.113
0.009	2.301	2.301	0.000	0	0	0	0.112	0.113

(b) MNIST on LeNet-5 with *epochwise* pruning and threshold $\tau = 0.001$.

λ	E_{CE} train	E_{CE} test	Ω_{l_1}	Ω_{l_0} $\hat{w}^1$	Ω_{l_0} $\hat{w}^2$	Ω_{l_0} $\hat{w}^3$	Acc. train	Acc. test
0	0.004	0.048	10 780.185	47817	10050	837	0.999	0.990
$1 \cdot 10^{-5}$	0.011	0.045	2540.030	22114	5596	669	0.998	0.988
$3 \cdot 10^{-5}$	0.017	0.040	1173.861	12793	2801	534	0.996	0.989
$5 \cdot 10^{-5}$	0.044	0.060	809.764	9158	1908	409	0.987	0.982
$7 \cdot 10^{-5}$	0.024	0.037	632.278	8293	1470	383	0.995	0.989
$9 \cdot 10^{-5}$	0.027	0.042	518.291	6706	1130	366	0.994	0.989
$1 \cdot 10^{-4}$	0.028	0.039	452.222	5796	1013	366	0.994	0.989
$3 \cdot 10^{-4}$	0.053	0.059	210.229	2975	443	307	0.987	0.985
$5 \cdot 10^{-4}$	0.068	0.066	151.072	2959	674	273	0.984	0.983
$7 \cdot 10^{-4}$	0.076	0.077	124.072	2562	667	239	0.982	0.979
$9 \cdot 10^{-4}$	0.103	0.096	104.132	2118	401	172	0.973	0.975
0.001	0.088	0.084	99.413	1726	371	251	0.978	0.979
0.003	0.158	0.147	48.033	916	485	150	0.960	0.962
0.005	0.175	0.165	32.276	602	150	97	0.957	0.957
0.007	2.301	2.301	0.000	0	0	5	0.112	0.113
0.009	2.301	2.301	0.000	0	0	5	0.112	0.113

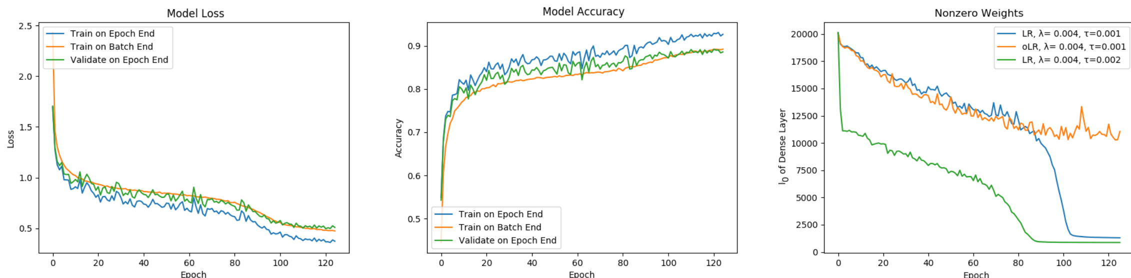
(c) MNIST on LeNet-5 with *batchwise* pruning and threshold $\tau = 0.001$.

λ	E_{CE} train	E_{CE} test	Ω_{l_1}	Ω_{l_0} $\hat{w}^1$	Ω_{l_0} $\hat{w}^2$	Ω_{l_0} $\hat{w}^3$	Acc. train	Acc. test
0	0.006	0.046	10 542.293	46551	9830	830	0.999	0.990
$1 \cdot 10^{-5}$	0.009	0.037	2483.202	17751	4575	614	0.998	0.990
$3 \cdot 10^{-5}$	0.017	0.038	1227.385	12582	2434	503	0.997	0.988
$5 \cdot 10^{-5}$	0.023	0.042	833.953	9260	1689	414	0.995	0.989
$7 \cdot 10^{-5}$	0.031	0.048	610.605	5946	1192	365	0.992	0.987
$9 \cdot 10^{-5}$	0.026	0.044	523.644	5779	977	298	0.995	0.989
$1 \cdot 10^{-4}$	0.028	0.046	500.372	5491	907	288	0.994	0.988
$3 \cdot 10^{-4}$	0.049	0.057	208.399	2107	300	170	0.989	0.985
$5 \cdot 10^{-4}$	0.064	0.065	143.657	1408	179	117	0.985	0.983
$7 \cdot 10^{-4}$	0.077	0.078	115.643	1253	157	116	0.982	0.981
$9 \cdot 10^{-4}$	0.086	0.081	101.795	1176	178	95	0.979	0.980
0.001	0.091	0.084	86.395	877	99	80	0.978	0.979
0.003	0.134	0.124	44.547	359	62	57	0.971	0.972
0.005	2.302	2.301	0.000	0	0	0	0.112	0.113
0.007	2.301	2.301	0.000	0	0	0	0.112	0.113
0.009	2.301	2.301	0.000	0	0	0	0.112	0.113

(d) CIFAR10 on VGG with *batchwise* pruning and threshold $\tau = 0.001$.

λ	E_{CE} train	E_{CE} test	Ω_{l_1}	$\Omega_{l_0}(w^3)$	Acc. train	Acc. test
0	0.270	0.430	12 402.742	20199	0.942	0.894
$1 \cdot 10^{-4}$	0.302	0.448	375.082	3972	0.935	0.887
$3 \cdot 10^{-4}$	0.310	0.451	115.192	1584	0.932	0.889
$5 \cdot 10^{-4}$	0.326	0.460	69.533	1100	0.927	0.886
$7 \cdot 10^{-4}$	0.318	0.456	52.049	885	0.929	0.887
$9 \cdot 10^{-4}$	0.322	0.455	41.586	744	0.929	0.888
0.001	0.344	0.479	37.173	659	0.923	0.882
0.003	0.313	0.455	15.906	443	0.933	0.888
0.005	0.326	0.461	10.737	376	0.929	0.887
0.007	0.340	0.477	8.445	359	0.923	0.882
0.009	0.334	0.470	6.971	312	0.925	0.888
0.01	0.331	0.473	6.478	305	0.925	0.883
0.03	0.360	0.491	2.780	209	0.917	0.880
0.05	0.394	0.530	1.902	171	0.909	0.870
0.07	0.382	0.516	1.453	151	0.910	0.871
0.09	0.400	0.526	1.171	137	0.906	0.870

of almost 99% of nonzero weights is achieved in the dense layer of the CNN model while we lose less than 2% in the prediction accuracy on the test data.



(a) Model loss for $\lambda = 0.03$ using LRS.

(b) Prediction accuracy for $\lambda = 0.03$ using LRS.

(c) $\Omega_{l_0}(\hat{w})$ for $\lambda = 0.004$, different thresholds τ and LRS.

Figure 6: VGG-like network on CIFAR10 trained for $\kappa_{\max} = 125$ epochs. The pruning threshold for a) and b) is $\tau = 0.001$ while in c) two different thresholds are compared with fixed and varying learning rate schedules.

Epochwise and after training pruning are again less effective. A critical value for the regularization hyperparameter is $\lambda^* = 0.03$, for which a reduction of nonzero weights in the `denselayer` by about 98.9% is realized. Since on CIFAR10 it is much more difficult to achieve good prediction accuracies, we additionally initialize the training process (i.e., the RMSProp optimizer) with a decreasing learning rate which decreases from $t_{\text{start}} = 0.001$ to $t_{\text{end}} = 0.0001$ as shown in Fig. 3. This induces a second steep decrease of the loss value and a corresponding increase of the prediction accuracy as can be seen Fig. 6(b) and Fig. 6(a), respectively. It not only leads to a more accurate performance in training and validation data, but also to a higher pruning effect which can be seen in Fig. 6(c). We compare two different pruning thresholds $\tau = 0.001$ and $\tau = 0.002$. The results shown in Fig. 6(c) suggest that the impact of the LRS is also influenced by the threshold value.

Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

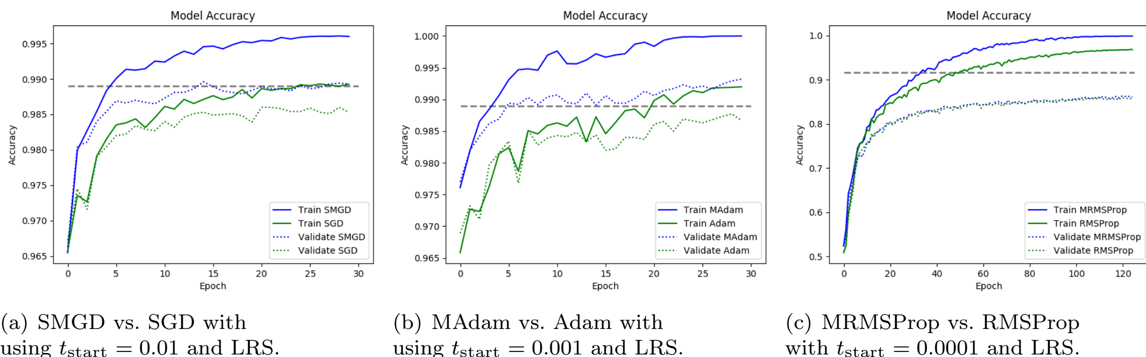
One can see clearly that with the same λ but different thresholds ($\tau = 0.001$ and $\tau = 0.002$) the epoch from which on the majority of weights falls below the threshold differs significantly. Indeed, here the LRS has a strong and favorable effect. Moreover, larger threshold values induce a faster convergence towards sparse networks.

6.3. Multiobjective Training

The following experiments all integrate batchwise ITP to reinforce the minimization of the second regularizing objective function Ω_{l_1}. While different variants of the SMGD algorithm are tested both on MNIST and CIFAR10 datasets, see Section 6.3.1, we restrict ourselves to LeNet-5 training on MNIST when approximating knee solutions in Section 6.3.2 by iteratively solving scalarized subproblems with the SGD algorithm.

6.3.1. Stochastic Multi-Gradient Descent

As our implementation of SMGD is built into the widely used open source SGD methods of Keras (including the Adam and RMSProp extensions) it inherits the same features as their vanilla implementations. Added and new features of our implementation are the consideration of the regularization term as a second objective function and a corresponding built-in update scheme for the weighting parameters according to Algorithm 2, such that in each iteration a direction of multiobjective steepest descent is chosen. We compare different features like the use of a momentum, decay and/or a learning rate schedule.



(a) SMGD vs. SGD with using $t_{\text{start}} = 0.01$ and LRS.

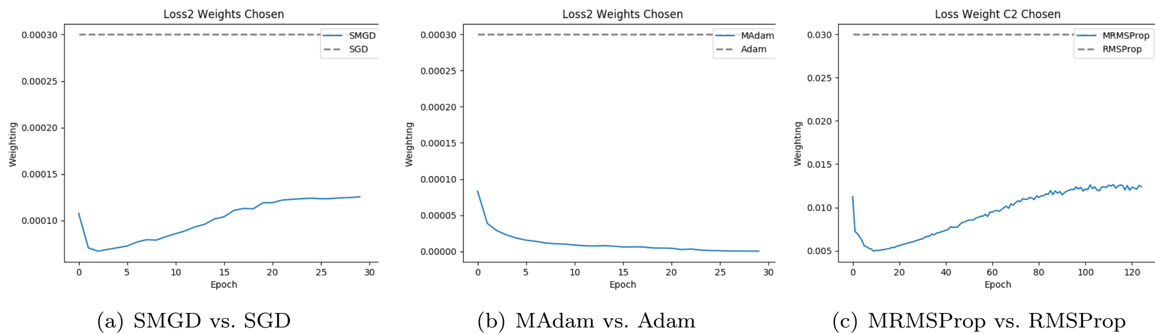
(b) MAdam vs. Adam with using $t_{\text{start}} = 0.001$ and LRS.

(c) MRMSProp vs. RMSProp with $t_{\text{start}} = 0.0001$ and LRS.

Figure 7: Predicition accuracies for MNIST (Fig. 7(a) and Fig. 7(b)) after $\kappa_{\max} = 30$ epochs and for CIFAR10 (Fig. 7(c)) after $\kappa_{\max} = 125$ epochs.

The resulting evolution of training- and validation accuracies are shown in Fig. 7. For comparison, the dashed line corresponds to the baseline training accuracy that we achieved with a critical weighting parameter λ^* as reported in Subsections 6.2.1 and 6.2.2, respectively. We notice that in all three cases both the training- and the validation accuracy of the respective SMGD algorithms are higher even though, except for the weighting of the objective functions which is constant in the SGD algorithm while it is automatically adapted in the SMGD algorithm, all other settings were the same. This comes, however, at the price of a significantly higher network complexity after SMGD training, i.e., a largely inferior value in the second objective function Ω_{l_1}. Indeed, the weighting parameter λ for the second objective function Ω_{l_1} (c.f. (7)), that is automatically adapted during SMGD training while it stays fixed in SGD training, remains significantly below the critical value of λ^* (that approximates a knee solution) during the complete SMGD training. This is shown in Fig. 8, where the dashed line corresponds to the critical weighting parameter λ^*. In general, it is noticeable that the validation accuracy and the training accuracy are further separated when using SMGD, which is also an indication for overfitting.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint



(a) SMGD vs. SGD (b) MAdam vs. Adam (c) MRMSProp vs. RMSProp

Figure 8: Evolution of the weighting parameter λ for the second objective Ω_{l_1} in SMGD as compared to its fixed value in SGD when training for $\kappa_{\max} = 30$ epochs on MNIST (Fig. 8(a) and Fig. 8(b)) and for $\kappa_{\max} = 125$ epochs on CIFAR10 (Fig. 8(c)).

6.3.2. Approximating the Pareto Knee

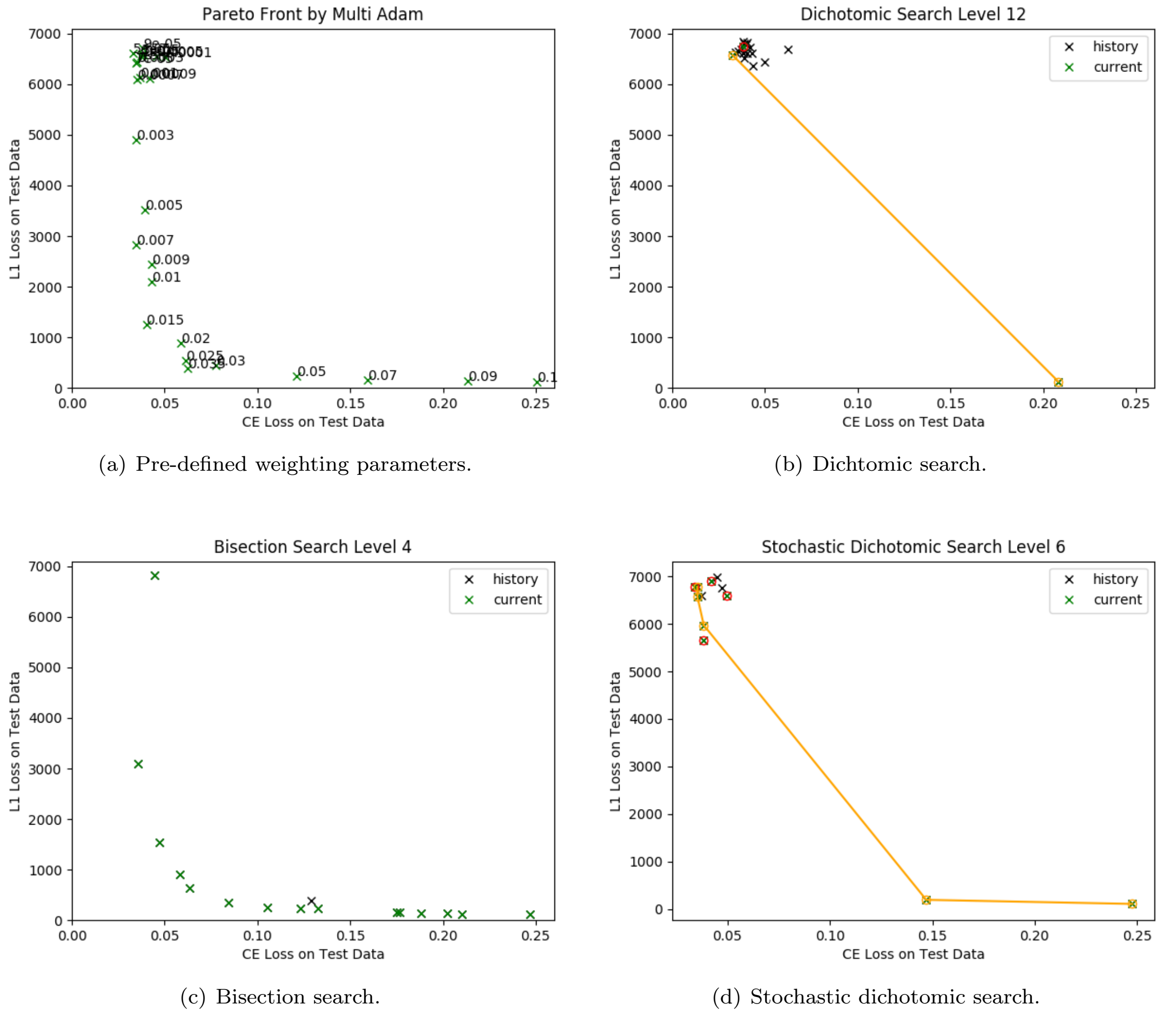
While the SMGD method has proven to converge quickly to an approximate Pareto optimal solution, this solution turns out to be prone to overfitting and usually far from a knee solution. Indeed, the first objective, i.e., the loss function, largely overrides the second regularizing objective function when automatically adapting the weighting parameter in the SMGD method.

In biobjective neural network training critical weighting parameters tend to be in a rather small interval close to zero which can be explained by the very different scaling, slope and curvature of the two objective functions. Slightly larger weighting parameters already lead to collapsed networks (with almost all weights equal to zero) while slightly smaller weighting parameters lead to overfitting (with almost no weights equal to zero). We investigate four different approaches to find critical weighting parameters that are all based on the repeated solution of weighted sum scalarizations (7) using a single-objective SGD method. The corresponding results are presented in Fig. 9. Except in the first case were significantly more training runs performed, the number of SGD calls and hence the total training cost was nearly the same. Moreover, the number of epochs was reduced to save computational time. Fig 9(a) serves as a benchmark by showing the training results w.r.t. a predefined list of weighting parameters from the interval $[0.00001, 0.1]$ that are condensed towards smaller values. A pronounced knee is clearly visible. Fig. 9(b) and Fig. 9(d) show the results of a classical dichotomic search algorithm and of stochastic dichotomic search according to Algorithm 3, respectively. While theoretically the classical dichotomic search algorithm should be able to quickly find a knee solution this is not the case in our experiments. This can be explained by the rather small interval of relevant weighting parameters, the large sensitivity w.r.t. small variations of the weighting parameter and, even more importantly, by the fact that the SGD is not guaranteed to converge to a Pareto optimal point. The stochastic dichotomic search algorithms shows a slightly better performance. However, it is still outperformed by a bisection search on the weighting parameter as shown in Fig. 9(c).

7. Conclusion and Outlook

In this work we take a multiobjective perspective on neural network training and present a straightforward and easy to implement method for unstructured pruning of dense layers in DNNs which is completely integrated in the training process. With this intra-training pruning (ITP) approach, we reduce the number of nonzero weights without a significant effect on the accuracy by adaptively condensing the network to relevant connections. We obtain the most promising results with *batchwise pruning* where weights are pruned after each batch during the training with respect to a predefined threshold value. We use l_1-regularization to guide the pruning process and to simultaneously avoid

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint



(a) Pre-defined weighting parameters.

(b) Dichtomic search.

(c) Bisection search.

(d) Stochastic dichotomic search.

Figure 9: Pareto front approximations using different search heuristics for finding critical weighting parameters and knee solutions.

overfitting. By re-interpreting regularization as a second and independent training goal, the trade-off between prediction accuracy and network complexity is analyzed. Optimization algorithms using the stochastic multi-gradient descent algorithm and several variants of scalarization based methods are implemented that address the specific challenges of neural network training.

The main advantage of this new approach can be seen in the fact that the network architecture and the network quality are optimized consistently and simultaneously. Moreover, the role of the regularization hyperparameter is thoroughly analyzed which facilitates informed choices of preferable trade-offs.

Different combinations of the learning rate t, the pruning threshold τ and the regularization hyperparameter λ are evaluated for two well-established datasets and network architectures for image classification. While batchwise pruning achieved excellent results on MNIST without further fine-tuning, we suggest to combine batchwise pruning with a learning rate schedule for CIFAR10. The presented techniques result in higher compression rates while maintaining a competitive prediction accuracy, both for MNIST and for CIFAR10.

The fine-tuning of hyperparameters is a challenging and time-consuming task when designing and training neural networks. While we focus on the regularization hyperparmeter in this work, we plan to extend our studies to include further hyperparameters in an automated machine learning (AutoML) strategy. This also involves a comparison of threshold controlled pruning methods with pruning strategies that aim at a predefined percentage of weights being zero, see, e.g., Zhu and Gupta [55].
Moreover, several other norms and quality measures may be tested as the second regularizing objective function in bi- or multiobjective neural network training and in combination with ITP. Particularly, l_p-regularization terms with $p < 1$ seem to be promising to approximate the l_0-regularizer. However, this leads to nonconvex optimization problems which makes the training problems more complex.

Acknowledgments. This work has been partially supported by EFRE (European fund for regional development) project EFRE-0400216.

References

[1] Aneja, Y., Nair, K., 1979. Bicriteria transportation problem. Management Science 25, 73–78.

[2] Azarian, K., Bhalgat, Y., Lee, J., Blankevoort, T., 2020. Learned threshold pruning. CoRR `arXiv:2003.00075v1`.

[3] Bai, Y., Wang, Y.X., Liberty, E., 2018. ProxQuant: Quantized neural networks via proximal operators. CoRR arxiv:1810.00861. `arXiv:1810.00861v3`.

[4] Bekasiewicz, A., Koziel, S., Leifsson, L., Du, X., 2017. Pareto ranking bisection algorithm for EM-driven multi-objective design of antennas in highly-dimensional parameter spaces. Procedia Computer Science 108, 453–1462. doi:`10.1016/j.procs.2017.05.102`.

[5] Bottou, L., 1999. On-Line Learning and Stochastic Approximations. Cambridge University Press, USA.

[6] Bottou, L., Curtis, F.E., Nocedal, J., 2018. Optimization methods for large-scale machine learning. SIAM Review 60, 223–311. doi:`10.1137/16M1080173`.

[7] Caballero, R., Cerda, E., del Mar Munoz, M., Rey, L., 2004. Stochastic approach versus multiobjective approach for obtaining efficient solutions in stochastic multiobjective programming problems. European Journal of Operational Research Vol. 158, 633–648. doi:`10.1016/S0377-2217(03)00371-0`.

[8] Chollet, F., et al., 2015. Keras. `https://keras.io`.

[9] Das, I., 1999. On characterizing the 'knee' of the Pareto curve based on normal-boundary intersection. Structural Optimization Vol. 18, 107–115. doi:`10.1007/bf01195985`.

[10] de Albuquerque Teixeira, R., Braga, A.P., Takahashi, R.H., Saldanha, R.R., 2000. Improving generalization of MLPs with multi-objective optimization. Neurocomputing doi:`10.1016/S0925-2312(00)00327-1`.

[11] Domhan, T., Springenberg, J., Hutter, F., 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: Proceedings of the 24th International Conference on Artificial Intelligence, AAAI Press. p. 3460–3468.

[12] Ehrgott, M., 2005. Multicriteria Optimization. 2 ed., Springer.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

[13] Fliege, J., Svaiter, B.F., 2000. Steepest descent methods for multicriteria optimization. Mathematical Methods of Operations Research Vol. 51, 479–494.

[14] Frankle, J., Carbin, M., 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. ICLR 2019, arXiv:1803.03635v5.

[15] Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. URL http://www.deeplearningbook.org.

[16] Gui, S., Wang, H., Yu, C., Yang, H., Wang, Z., Liu, J., 2019. Adversarially trained model compression: When robustness meets efficiency. CoRR arXiv:1902.03538.

[17] Guo, Y., Yao, A., Chen, Y., 2016. Dynamic network surgery for efficient dnns. CoRR arXiv:1608.04493.

[18] Guo, Y., Zhang, C., Zhang, C., Chen, Y., 2018. Sparse DNNs with improved adversarial robustness. CoRR arXiv:1810.09619v2.

[19] Han, S., Mao, H., Dally, W.J., 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. CoRR arXiv:1510.00149v5.

[20] Han, S., Pool, J., Tran, J., Dally, W.J., 2015b. Learning both weights and connections for efficient neural networks. Proceedings of the 28th International Conference on Neural Information Processing Systems , 1135–1143.

[21] Hassibi, B., Stork, D., Wolff, G., 1993. Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, IEEE. pp. 293–299. doi:10.1109/icnn.1993.298572.

[22] He, Y., Han, S., 2018. ADC: automated deep compression and acceleration with reinforcement learning. CoRR arXiv:1802.03494.

[23] He, Y., Zhang, X., Sun, J., 2017. Channel pruning for accelerating very deep neural networks. CoRR arXiv:1707.06168.

[24] Hinton, G., Vinyals, O., Dean, J., 2015. Distilling the knowledge in a neural network. CoRR arXiv:1503.02531.

[25] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Quantized neural networks: Training neural networks with low precision weights and activations. CoRR arXiv:1609.07061v1.

[26] Jin, Y. (Ed.), 2006. Multi-Objective Machine Learning. Studies in Computational Intelligence, Springer.

[27] Jin, Y., Sendhoff, B., 2008. Pareto-based multiobjective machine learning: An overview and case studies. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) Vol. 38, 397–415. doi:10.1109/tsmcc.2008.919172.

[28] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. International Conference on Learning Representations, arXiv:1412.6980.

[29] Krizhevsky, A., 2009. Learning multiple layers of features from tiny images. University of Toronto URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[30] Kuzmin, A., Nagel, M., Pitre, S., Pendyam, S., Blankevoort, T., Welling, M., 2019. Taxonomy and evaluation of structured compression of convolutional neural networks. CoRR arXiv:1912.09802v1.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

[31] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE Vol. 86, 2278–2324. doi:10.1109/5.726791.

[32] LeCun, Y., Cortes, C., Burges, C., 2010. MNIST handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist 2.

[33] LeCun, Y., Denker, J.S., Solla, S.A., 1990. Optimal brain damage. Advances in Neural Information Processing Systems Vol. 2 , 598–605.

[34] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2016. Pruning filters for efficient ConvNets. CoRR arXiv:1608.08710v3.

[35] Liu, S., Vicente, L.N., 2019. The stochastic multi-gradient algorithm for multi-objective optimization and its application to supervised machine learning. CoRR arXiv:1907.04472v2.

[36] Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T., 2018. Rethinking the value of network pruning. CoRR arXiv:1810.05270v2.

[37] Lym, S., Choukse, E., Zangeneh, S., Wen, W., Sanghavi, S., Erez, M., 2019. Prunetrain: Fast neural network training by dynamic sparse model reconfiguration. CoRR arXiv:1901.09290v4.

[38] Mercier, Q., Poirion, F., Désidéri, J.A., 2018. A stochastic multiple gradient descent algorithm. European Journal of Operational Research Vol. 271, 808–817. doi:10.1016/j.ejor.2018.05.06.

[39] Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J., 2019. Importance estimation for neural network pruning. CoRR arXiv:1906.10771v1.

[40] Mummadi, C.K., Genewein, T., Zhang, D., Brox, T., Fischer, V., 2019. Group pruning using a bounded lp norm for group gating and regularization, in: German Conference on Pattern Recognition (GCPR), Springer. pp. 139–155.

[41] Ng, G.S., Wahab, A., Shi, D., 2003. Entropy learning and relevance criteria for neural network pruning. International Journal of Neural Systems Vol. 13, 291–305. doi:10.1142/s0129065703001637.

[42] de Pádua Braga, A., Takahashi, R.H.C., Costa, M.A., de Albuquerque Teixeira, R., 2006. Multi-objective algorithms for neural networks learning, in: Multi-Objective Machine Learning, Springer. pp. 151–171. doi:10.1007/3-540-33019-4_7.

[43] Robbins, H., Monro, S., 1951. A stochastic approximation method. The Annals of Mathematical Statistics Vol. 22, 400–407.

[44] Sacks, J., 1958. Asymptotic distribution of stochastic approximation procedures. Annals of Mathematical Statistics 29, 373–405. doi:10.1214/aoms/1177706619.

[45] Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. CoRR arXiv:arxiv:1409.1556.

[46] Tang, J., Shivanna, R., Zhao, Z., Lin, D., Singh, A., Chi, E.H., Jain, S., 2020. Understanding and improving knowledge distillation. CoRR arXiv:2002.03532v1.

[47] Tieleman, T., Hinton, G., 2012. Lecture 6.5-rmsprop: Divide the gradient by a runningaverage of its recent magnitude. COURSERA: Neural networks for machine learning URL: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Preprint – Preprint – Preprint – Preprint – Preprint – Preprint

[48] Tomonari Furukawa, C.J.K.L., Michopoulos, J.G., 2006. Regularization for parameter identification using multi-objective optimization, in: Jin, Y. (Ed.), Multi-Objective Machine Learning, Springer. pp. 125–149. doi:10.1007/3-540-33019-4_6.

[49] Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., Hu, X., 2019. Pruning from scratch. CoRR arXiv:1909.12579v1.

[50] Yang, H., Wen, W., Li, H., 2019. DeepHoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. CoRR arXiv:1908.09979v2.

[51] Ye, S., Xu, K., Liu, S., Cheng, H., Lambrechts, J.H., Zhang, H., Zhou, A., Ma, K., Wang, Y., Lin, X., 2019. Adversarial robustness vs model compression, or both? CoRR arXiv:1903.12561v4.

[52] Yeom, S.K., Seegerer, P., Lapuschkin, S., Wiedemann, S., Müller, K.R., Samek, W., 2019. Pruning by explaining: A novel criterion for deep neural network pruning. CoRR arXiv:1912.08881v1.

[53] Zhang, T., Ma, X., Zhan, Z., Zhou, S., Qin, M., Sun, F., Chen, Y.K., Ding, C., Fardad, M., Wang, Y., 2020. A unified DNN weight compression framework using reweighted optimization methods. CoRR arXiv:2004.05531v1.

[54] Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., Wang, Y., 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. CoRR arXiv:1804.03294v3.

[55] Zhu, M., Gupta, S., 2017. To prune, or not to prune: Exploring the efficacy of pruning for model compression. CoRR arXiv:1710.01878v2.