# A Bregman Learning Framework for Sparse Neural Networks

**Leon Bungert**                                            LEON.BUNGERT@HCM.UNI-BONN.DE
*Hausdorff Center for Mathematics*
*University of Bonn*
*Endenicher Allee 62, Villa Maria, 53115 Bonn, Germany*

**Tim Roith**                                                     TIM.ROITH@FAU.DE
*Department of Mathematics*
*Friedrich–Alexander University Erlangen–Nürnberg*
*Cauerstraße 11, 91058 Erlangen, Germany*

**Daniel Tenbrinck**                                       DANIEL.TENBRINCK@FAU.DE
*Department of Mathematics*
*Friedrich–Alexander University Erlangen–Nürnberg*
*Cauerstraße 11, 91058 Erlangen, Germany*

**Martin Burger**                                            MARTIN.BURGER@FAU.DE
*Department of Mathematics*
*Friedrich–Alexander University Erlangen–Nürnberg*
*Cauerstraße 11, 91058 Erlangen, Germany*

## Abstract

We propose a learning framework based on stochastic Bregman iterations, also known as mirror descent, to train sparse neural networks with an inverse scale space approach. We derive a baseline algorithm called *LinBreg*, an accelerated version using momentum, and *AdaBreg*, which is a Bregmanized generalization of the *Adam* algorithm. In contrast to established methods for sparse training the proposed family of algorithms constitutes a regrowth strategy for neural networks that is solely optimization-based without additional heuristics. Our Bregman learning framework starts the training with very few initial parameters, successively adding only significant ones to obtain a sparse and expressive network. The proposed approach is extremely easy and efficient, yet supported by the rich mathematical theory of inverse scale space methods. We derive a statistically profound sparse parameter initialization strategy and provide a rigorous stochastic convergence analysis of the loss decay and additional convergence proofs in the convex regime. Using only 3.4% of the parameters of ResNet-18 we achieve 90.2% test accuracy on CIFAR-10, compared to 93.6% using the dense network. Our algorithm also unveils an autoencoder architecture for a denoising task. The proposed framework also has a huge potential for integrating sparse backpropagation and resource-friendly training. Code is available at `https://github.com/TimRoith/BregmanLearning`.

**Keywords:** Bregman Iterations, Mirror Descent, Sparse Neural Networks, Sparsity, Inverse Scale Space, Optimization

## 1. Introduction

Large and deep neural networks have shown astonishing results in challenging applications, ranging from real-time image classification in autonomous driving, over assisted diagnoses in healthcare, to surpassing human intelligence in highly complex games (Amato et al., 2013; Rawat and Wang, 2017; Silver et al., 2016). The main drawback of many of these architectures is that they require huge amounts of memory and can only be employed using specialised hardware, like GPGPUs and TPUs. This makes them inaccessible to normal users with only limited computational resources on their mobile devices or computers (Hoefler et al., 2021). Moreover, the carbon footprint of training large networks has become an issue of major concern recently (Dhar, 2020), hence calling for resource-efficient methods.

The success of large and deep neural networks is not surprising as it has been predicted by universal approximation theorems (Cybenko, 1989; Lu et al., 2017), promising a smaller error with increasing number of neurons and layers. Besides the increase in computational complexity, each neuron added to the network architecture also adds to the amount of free parameters and local optima of the loss.

Consequently, a significant branch of modern research aims for training "sparse neural networks", which has lead to different strategies, based on neglecting small parameters or such with little influence on the network output, see Hoefler et al. (2021) for an extensive review. Apart from computational and resource efficiency, sparse training also sheds light on neural architecture design and might answer the question why certain architectures work better than others.

A popular approach for generating sparse neural networks are pruning techniques (LeCun et al., 1990; Han et al., 2015), which have been developed to sparsify a dense neural network during or after training by dropping dispensable neurons and connections. Another approach, which is based on the classical Lasso method from compressed sensing (Tibshirani, 1996), incorporates $\ell_1$ regularization into the training problem, acting as convex relaxation of sparsity-enforcing $\ell_0$ regularization. These endeavours are further supported by the recently stated "lottery ticket hypothesis" (Frankle and Carbin, 2018), which postulates that dense, feed-forward networks contain sub-networks with less neurons that, if trained in isolation, can achieve the same test accuracy as the original network.

An even more intriguing idea is "grow-and-prune" (Dai et al., 2019), which starts with a sparse network and augments it during training, while keeping it as sparse as possible. To this end new neurons are added, e.g., by splitting overloaded neurons into new specimen or using gradient-based indicators, while insignificant parameters are set to zero by thresholding.

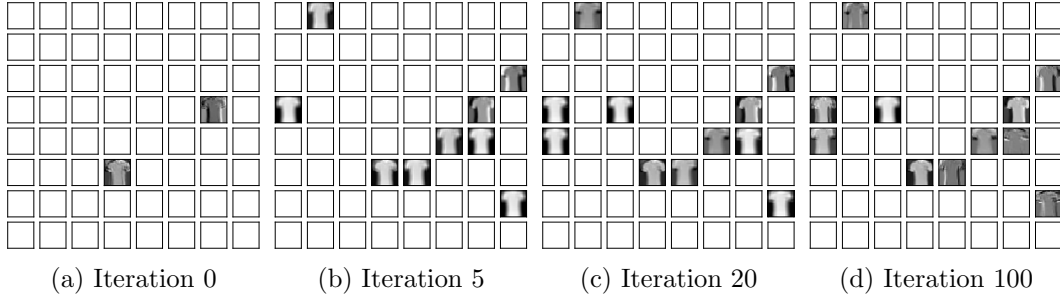(a) Iteration 0      (b) Iteration 5      (c) Iteration 20      (d) Iteration 100

Figure 1: Inverse scale space character of LinBreg visualized through feature maps of a convolutional neural network. Descriptive kernels are gradually added in the training process.

Many of the established methods in the literature are bound to specific architectures, e.g., fully-connected feedforward layers (Castellano et al., 1997; Liu et al., 2021). In this paper we propose a more conceptual and optimization-based approach. The idea is to mathematically follow the intuition of starting with very few parameters and adding only necessary ones in an inverse scale space manner, see Figure 1 for an illustration of our algorithm on a convolutional neural network. For this sake we propose a Bregman learning framework utilizing linearized Bregman iterations—originally introduced for compressed sensing by Yin et al. (2008)—for training sparse neural networks.

Our **main contributions** are the following:

- We derive an extremely simple and efficient algorithm for training sparse neural networks, called *LinBreg*.

- We also propose a momentum-based acceleration and *AdaBreg*, which utilizes the Adam algorithm (Kingma and Ba, 2014).

- We perform a rigorous stochastic convergence analysis of LinBreg for strongly convex losses, in infinite dimensions, and without any smoothness assumptions on the sparsity regularizer.

- We propose a sparse initialization strategy for the network parameters.

- We show that our algorithms are effective for training sparse neural networks and show their potential for architecture design by unveiling a denoising autoencoder.

The structure of this paper is as follows: In Section 1.1 we explain our baseline algorithm *LinBreg* in a nutshell and in Section 1.2 we discuss related work. Sections 1.3 and 1.4 clarify notation and collect preliminaries on neural networks and convex analysis, the latter being important for the derivation and analysis of our algorithms. In Section 2 we explain how Bregman iterations can be incorporated into the training of sparse neural networks, derive and discuss variants of the proposed

---

**Algorithm 1:** *LinBreg*, an inverse scale space algorithm for training sparse neural networks by successively adding weights whilst minimizing the loss. The functional $J$ is sparsity promoting, e.g., the $\ell_1$-norm.

---
**default:** $\delta = 1$

$\theta \leftarrow$ Section 4.1, $\quad v \leftarrow \partial J(\theta) + \frac{1}{\delta}\theta$       // `initialize`

**for** epoch $e = 1$ **to** $E$ **do**

    **for** minibatch $B \subset \mathcal{T}$ **do**

        $g \leftarrow \nabla L(\theta; B)$         // `Backpropagation`

        $v \leftarrow v - \tau g$         // `Gradient step`

        $\theta \leftarrow \mathrm{prox}_{\delta J}(\delta v)$         // `Regularization`

---

Bregman learning algorithm, including accelerations using momentum and Adam. We perform a mathematical analysis for stochastic linearized Bregman iterations in Section 3 and discuss conditions for convergence of the loss function and the parameters. In Section 4 we first discuss our statistical sparse initialization strategy and then evaluate our algorithms on benchmark data sets (MNIST, Fashion-MNIST, CIFAR-10) using feedforward, convolutional, and residual neural networks.

## 1.1 The Bregman Training Algorithm in a Nutshell

Algorithm 1 states our baseline algorithm *LinBreg* for training sparse neural networks with an inverse scale space approach. Mathematical tools and derivations of LinBreg and its variants *LinBreg with momentum* (Algorithm 2) and *AdaBreg* (Algorithm 3), a generalization of *Adam* (Kingma and Ba, 2014), are presented in Section 2; a convergence analysis is provided in Section 3.

    LinBreg can easily be applied to any neural network architecture $f_\theta$, parametrized with parameters $\theta \in \Theta$, using a set of training data $\mathcal{T}$, and an empirical loss function $L(\theta; B)$, where $B \subset \mathcal{T}$ is a batch of training data. LinBreg's most important ingredient is a sparsity enforcing functional $J : \Theta \rightarrow (-\infty, \infty]$, which acts on groups of network parameters as, for instance, convolutional kernels, weight matrices, biases, etc. Following Scardapane et al. (2017) and denoting the collection of all parameter groups for which sparsity is desired by $\mathcal{G}$, two possible regularizers which induce sparsity or group sparsity, respectively, can be defined as

$$J(\theta) = \lambda \sum_{\mathbf{g} \in \mathcal{G}} \|\mathbf{g}\|_1, \qquad \text{the } \ell_1\text{-norm,} \tag{1.1}$$

$$J(\theta) = \lambda \sum_{\mathbf{g} \in \mathcal{G}} \sqrt{n_{\mathbf{g}}} \, \|\mathbf{g}\|_2, \qquad \text{the group } \ell_{1,2}\text{-norm.} \tag{1.2}$$

Here $\lambda > 0$ is a parameter controlling the regularization strength, $n_{\mathbf{g}}$ denotes the number of elements in $\mathbf{g}$, and the factor $\sqrt{n_{\mathbf{g}}}$ ensures a uniform weighting of all groups (Scardapane et al., 2017).

LinBreg uses two variables $v$ and $\theta$, coupled through the condition that $v \in \partial J_\delta(\theta)$ is a subgradient of the *elastic net regularization* $J_\delta(\theta) := J(\theta) + \frac{1}{2\delta}\|\theta\|^2$ introduced by Zou and Hastie (2005) (see Sections 1.3 and 1.4 for definitions). The algorithm successively updates $v$ with gradients of the loss and recovers sparse parameters $\theta$ by applying a proximal operator. For instance, if $J(\theta) = \lambda\|\theta\|_1$ equals the $\ell_1$-norm, the proximal operator in Algorithm 1 coincides with the soft shrinkage operator:

$$\mathrm{prox}_{\delta J}(\delta v) = \delta\,\mathrm{shrink}(v; \lambda) := \delta\,\mathrm{sign}(v)\max(|v| - \lambda, 0). \qquad (1.3)$$

In this case only those parameters $\theta$ will be non-zero whose subgradients $v$ have magnitude larger than the regularization parameter $\lambda$. Furthermore, $\delta > 0$ only steers the magnitude of the resulting weights and not their support. Furthermore, if $J(\theta) = 0$ then $\mathrm{prox}_{\delta J}(\delta v) = \delta v$ and therefore Algorithm 1 coincides with stochastic gradient descent (SGD) with learning rate $\delta\tau$. These two observations explain our default choice of $\delta = 1$.

In general, the proximal operators of the regularizers above can be efficiently evaluated since they admit similar closed form solutions based on soft thresholding. Hence, the computational complexity of LinBreg is dominated by the backpropagation and coincides with the complexity of vanilla stochastic gradient descent. However, note that our framework has great potential for complexity reduction via sparse backpropagation methods, cf. Dettmers and Zettlemoyer (2019).

The special feature which tells LinBreg apart from standard sparsity regularization (Louizos et al., 2017; Scardapane et al., 2017; Srinivas et al., 2017) or pruning (LeCun et al., 1990; Han et al., 2015) is its inverse scale space character. LinBreg is derived based on Bregman iterations, originally developed for scale space approaches in imaging (Osher et al., 2005; Burger et al., 2006; Yin et al., 2008; Cai et al., 2009b,a; Zhang et al., 2011). Instead of removing weights from a dense trained network, it starts from a very sparse initial set of parameters (see Section 4.1) and successively adds non-zero parameters whilst minimizing the loss.

## 1.2 Related Work

**Dense-to-Sparse Training**  A well-established approach for training sparse neural network consists in solving the regularized empirical risk minimization

$$\min_{\theta \in \Theta} L(\theta; B) + J(\theta), \qquad (1.4)$$

where $J$ is a (sparsity-promoting) non-smooth regularization functional. If $J$ equals the $\ell_1$-norm this is referred to as *Lasso* (Tibshirani, 1996) and was extended to *Group Lasso* for neural networks by Scardapane et al. (2017) by using group norms.

We refer to de Dios and Bruna (2020) for a mean-field analysis of this approach. The regularized risk minimization (1.4) is a special case of Dense-to-Sparse training. Even if the network parameters are initialized sparsely, any optimization method for (1.4) will instantaneously generate dense weights, which are subsequently sparsified. A different strategy for Dense-to-Sparse training is *pruning* (LeCun et al., 1990; Han et al., 2015), see also Zhu and Gupta (2017), which first trains a network and then removes parameters to create sparse weights. This procedure can also be applied alternatingly, which is referred to as *iterative pruning* (Castellano et al., 1997). The weight removal can be achieved based on different criteria, e.g., their magnitude or their influence on the network output.

**Sparse-to-Sparse Training** In contrast, Sparse-to-Sparse training aims to grow a neural network starting from a sparse initialization until it is sufficiently accurate. This is also the paradigm of our LinBreg algorithm, generating an inverse sparsity scale space. Other approaches from literature are grow-and-prune strategies (Mocanu et al., 2018; Dettmers and Zettlemoyer, 2019; Dai et al., 2019; Liu et al., 2021; Evci et al., 2020) which, starting from sparse networks, successively add and remove neurons or connections while training the networks.

**Proximal Gradient Descent** A related approach to LinBreg is *proximal gradient descent* (ProxGD) for optimizing the regularized empirical risk minimization (1.4), which is an inherently non-smooth optimization problem due to the presence of the $\ell_1$-norm-type functional $J$. Therefore, proximal gradient descent alternates between a gradient step of the loss with a proximal step of the regularization:

$$g \leftarrow \nabla L(\theta; B) \tag{1.5a}$$

$$\theta \leftarrow \theta - \tau g \tag{1.5b}$$

$$\theta \leftarrow \text{prox}_{\tau J}(\theta). \tag{1.5c}$$

Applications for training neural networks and convergence analysis of this algorithm and its variants can be found, e.g., in Nitanda (2014); Rosasco et al. (2020); Reddi et al. (2016); Yang et al. (2019); Yun et al. (2020). It differs from Algorithm 1 by the lack of a subgradient variable and by using the learning rate $\tau$ within the proximal operator. These seemingly minor algorithmic differences cause major differences for the trained parameters. Indeed, the effect of $J$ kicks in only after several iterations when the proximal operator has been applied sufficiently often to set some parameters to zero, as can be observed in Figure 2, Section 4.2. Furthermore, proximal gradient descent does not decrease the loss monotonously which we are able to prove for LinBreg.

**Bregman Iterations** Bregman iterations and in particular linearized Bregman iterations have been introduced and thoroughly analyzed for sparse regularization approaches in imaging and compressed sensing (see, e.g., Osher et al. (2005); Yin et al. (2008); Bachmayr and Burger (2009); Cai et al. (2009b,a); Yin (2010); Burger

et al. (2007, 2013)). More recent applications of Bregman type methods in the context of image restoration are Benfenati and Ruggiero (2013); Jia et al. (2016); Li et al. (2018). Linearized Bregman iterations for non-convex problems, which appear in machine learning and imaging applications like blind deblurring, have first been analyzed by Bachmayr and Burger (2009); Benning and Burger (2018); Benning et al. (2021). Benning and Burger (2018) also showed that linearized Bregman iterations for convex problems can be formulated as forward pass of a neural network. Benning et al. (2021) applied them for training neural networks with low-rank weight matrices, using nuclear norm regularization. Huang et al. (2016) suggested a split Bregman approach for training sparse neural networks and Fu et al. (2022) provided a deterministic convergence result along the lines of Benning et al. (2021). A recent analysis of Bregman stochastic gradient descent, which is the same as linearized Bregman iterations, however using strong regularity assumptions on the involved functions, is done by Dragomir et al. (2021).

**Mirror Descent**  As it turns out, linearized Bregman iterations are largely known under yet another name: *mirror descent*. This method was first proposed by Nemirovskij and Yudin (1983) and related to Bregman distances by Beck and Teboulle (2003). Dragomir et al. (2021) present a literature overview of stochastic mirror descent. Some months after the release of the preprint version of the present article, D'Orazio et al. (2021) presented a convergence analysis of stochastic mirror descent a.k.a. Bregman iterations, using a weaker bounded variance condition for the stochastic gradients, albeit working in a smooth setting. In contrast, our analysis does not require any smoothness of $J$.

### 1.3 Preliminaries on Neural Networks

We denote neural networks, which map from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$ and have parameters in some parameter space $\Theta$, by

$$f_\theta : \mathcal{X} \to \mathcal{Y}, \quad \theta \in \Theta. \tag{1.6}$$

In principle $\mathcal{X}$, $\mathcal{Y}$, and $\Theta$ can be infinite-dimensional and we only assume that $\Theta$ is a Hilbert space, equipped with an inner product $\langle \tilde{\theta}, \theta \rangle$ and associated norm $\|\theta\| = \sqrt{\langle \theta, \theta \rangle}$. Given a set of training pairs $\mathcal{T} \subset \mathcal{X} \times \mathcal{Y}$ and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ we denote the empirical loss associated to the training data by

$$\mathcal{L}(\theta) := \frac{1}{|\mathcal{T}|} \sum_{(x,y) \in \mathcal{T}} \ell(f_\theta(x), y). \tag{1.7}$$

The empirical risk minimization approach to finding optimal parameters $\theta \in \Theta$ of the neural network $f_\theta$ then consists in solving

$$\min_{\theta \in \Theta} \mathcal{L}(\theta). \tag{1.8}$$

If one assumes that the training set $\mathcal{T}$ is sampled from some probability measure $\rho$ on the product space $\mathcal{X} \times \mathcal{Y}$, the empirical risk minimization is an approximation of the infeasible population risk minimization

$$\min_{\theta \in \Theta} \int_{\mathcal{X} \times \mathcal{Y}} \ell(f_\theta(x), y) \, \mathrm{d}\rho(x, y). \tag{1.9}$$

One typically samples batches $B \subset \mathcal{T}$ from the training set and replaces $\mathcal{L}(\theta)$ by the empirical risk of the batch

$$L(\theta; B) := \frac{1}{|B|} \sum_{(x,y) \in B} \ell(f_\theta(x), y), \tag{1.10}$$

which is utilized in *stochastic* gradient descent methods.

For a feed-forward architecture with $L \in \mathbb{N}$ layers of sizes $n_l$ we split the variable $\theta$ into weights and biases $W^l \in \mathbb{R}^{n_l, n_{l-1}}$, $b^l \in \mathbb{R}^{n_l}$ for $l \in \{1, \ldots, L\}$. In this case we have

$$f_\theta(x) = \Phi^L \circ \cdots \circ \Phi^1(x), \tag{1.11}$$

where the $l$-th layer for $l \in \{1, \ldots, L\}$ is given by

$$\Phi^l(z) := \sigma^l(W^l z + b^l). \tag{1.12}$$

Here $\sigma^l$ denote activation functions, as for instance ReLU, TanH, Sigmoid, etc., (Goodfellow et al., 2016). In this case, sparsity promoting regularizers are the $\ell_1$-norm or the group $\ell_{1,2}$-norm

$$J(\theta) = \lambda \sum_{l=1}^{L} \|W^l\|_{1,1}, \tag{1.13}$$

$$J(\theta) = \lambda \sum_{l=1}^{L} \sqrt{n_{l-1}} \, \|W^l\|_{1,2}, \tag{1.14}$$

which induce sparsity of the weight matrices and of the non-zero rows of weight matrices, respectively. Here the scaling $\sqrt{n_{l-1}}$ weighs the influence of the $l$-th layer based on the number of incoming neurons.

### 1.4 Preliminaries on Convex Analysis

In this section we introduce some essential concepts from convex analysis which we need to derive LinBreg and its variants and in order to make our argumentation more self-contained. For an overview of these topics we refer to Benning and Burger (2018); Rockafellar (1997); Bauschke and Combettes (2011). A functional $J : \Theta \to (-\infty, \infty]$ on a Hilbert space $\Theta$ is called convex if

$$J(\lambda \bar{\theta} + (1 - \lambda)\theta) \leq \lambda J(\bar{\theta}) + (1 - \lambda)J(\theta), \quad \forall \lambda \in [0, 1], \bar{\theta}, \theta \in \Theta. \tag{1.15}$$

We define the effective domain of $J$ as $\mathrm{dom}(J) := \{\theta \in \Theta \,:\, J(\theta) \neq \infty\}$ and call $J$ proper if $\mathrm{dom}(J) \neq \emptyset$. Furthermore, $J$ is called lower semicontinuous if $J(u) \leq \liminf_{n\to\infty} J(u_n)$ holds for all sequences $(u_n)_{n\in\mathbb{N}} \subset \Theta$ converging to $u$. First, we define the subdifferential of a convex and proper functional $J : \Theta \to (-\infty, \infty]$ at a point $\theta \in \Theta$ as

$$\partial J(\theta) := \left\{ p \in \Theta \,:\, J(\theta) + \langle p, \overline{\theta} - \theta \rangle \leq J(\overline{\theta}), \; \forall \overline{\theta} \in \Theta \right\}. \qquad (1.16)$$

The subdifferential is a non-smooth generalization of the derivative and coincides with the classical gradient (or Fréchet derivative) if $J$ is differentiable. We denote $\mathrm{dom}(\partial J) := \{\theta \in \Theta \,:\, \partial J(\theta) \neq \emptyset\}$ and observe that $\mathrm{dom}(\partial J) \subset \mathrm{dom}(J)$.

Next, we define the Bregman distance of two points $\theta \in \mathrm{dom}(\partial J), \overline{\theta} \in \Theta$ with respect to a convex and proper functional $J : \Theta \to (-\infty, \infty]$ as

$$D_J^p(\overline{\theta}, \theta) := J(\overline{\theta}) - J(\theta) - \langle p, \overline{\theta} - \theta \rangle, \quad p \in \partial J(\theta). \qquad (1.17)$$

The Bregman distance can be interpreted as the distance between the linearization of $J$ at $\theta$ and its graph and hence somewhat measures the degree of linearity of the functional. Note furthermore that the Bregman distance (1.17) is neither definite, symmetric nor fulfills the triangle inequality, hence it is not a metric. However, it fulfills the two distance axioms

$$D_J^p(\overline{\theta}, \theta) \geq 0, \quad D_J^p(\theta, \theta) = 0, \quad \forall \overline{\theta} \in \Theta, \theta \in \mathrm{dom}(\partial J). \qquad (1.18)$$

By summing up two Bregman distances, one can also define the symmetric Bregman distance with respect to $p \in \partial J(\theta)$ and $\overline{p} \in \partial J(\overline{\theta})$ as

$$D_J^{\mathrm{sym}}(\overline{\theta}, \theta) := D_J^p(\overline{\theta}, \theta) + D_J^{\overline{p}}(\theta, \overline{\theta}). \qquad (1.19)$$

Here, we suppress the dependency on $p$ and $\overline{p}$ to simplify the notation.

Last, we define the proximal operator of a convex, proper and lower semicontinuous functional $J : \Theta \to (-\infty, \infty]$ as

$$\mathrm{prox}_J(\overline{\theta}) := \arg\min_{\theta \in \Theta} \frac{1}{2}\|\theta - \overline{\theta}\|^2 + J(\theta). \qquad (1.20)$$

Proximal operators are a key concept in non-smooth optimization since they can be used to replace gradient descent steps of non-smooth functionals, as done for instance in proximal gradient descent (1.5). Obviously, given some $\overline{\theta} \in \Theta$ the proximal operator outputs a new element $\theta \in \Theta$ which has a smaller value of $J$ whilst being close to the previous element $\overline{\theta}$.

## 2. Bregmanized training of Neural Networks

In this section we first give a short overview of inverse scale space flows which are the time-continuous analogue of our algorithms. Subsequently, we derive *LinBreg*

(Algorithm 1) by passing from Bregman iterations to linearized Bregman iterations, which we then reformulate in a very easy and compact form. We then derive *LinBreg with momentum* (Algorithm 2) by discretizing a second-order in time inverse scale space flow and propose *AdaBreg* (Algorithm 3) as a generalization of the popular Adam algorithm (Kingma and Ba, 2014).

### 2.1 Inverse Scale Space Flows (with Momentum)

In the following we discuss the inverse scale space flow, which arises as gradient flow of a loss functional $\mathcal{L}$ with respect to the Bregman distance (1.17). In particular, it couples the minimization of $\mathcal{L}$ with a simultaneous regularization through $J$. To give meaning to this, one considers the following implicit Euler scheme

$$\theta^{(k+1)} = \arg\min_{\theta \in \Theta} D_J^{p^{(k)}}(\theta, \theta^{(k)}) + \tau^{(k)}\mathcal{L}(\theta), \tag{2.1a}$$

$$p^{(k+1)} = p^{(k)} - \tau^{(k)}\nabla\mathcal{L}(\theta^{(k+1)}) \in \partial J(\theta^{(k+1)}) \tag{2.1b}$$

which is know as *Bregman iteration*. Here, $\theta^{(k)}$ is the previous iterate with subgradient $p^{(k)} \in \partial J(\theta^{(k)})$, and $\tau^{(k)} > 0$ is a sequence of time steps. Note that the subgradient update in the second line of (2.1) coincides with the optimality conditions of the first line.

The time-continuous limit of (2.1) as $\tau^{(k)} \to 0$ is the inverse scale space flow

$$\begin{cases} \dot{p}_t = -\nabla\mathcal{L}(\theta_t), \\ p_t \in \partial J(\theta_t), \end{cases} \tag{2.2}$$

see Burger et al. (2006, 2007) for a rigorous derivation in the context of image denoising.

If $J(\theta) = \frac{1}{2}\|\theta\|^2$ then $\partial J(\theta) = \theta$ and (2.2) coincides with the standard gradient flow

$$\dot{\theta}_t = -\nabla\mathcal{L}(\theta_t). \tag{2.3}$$

Hence, the inverse scale space is a proper generalization of the gradient flow and allows for regularizing the path along which the loss is minimized using $J$ (see Benning et al. (2021)). For strictly convex loss functions this might seem pointless since they have a unique minimum anyways, however, for merely convex or even non-convex losses the inverse scale space allows to 'select' (local) minima with desirable properties.

In this paper, we also propose an inertial version of (2.2) which depends on an inertial parameter $\gamma \geq 0$ and takes the form

$$\begin{cases} \gamma\ddot{p}_t + \dot{p}_t = -\nabla\mathcal{L}(\theta_t), \\ p_t \in \partial J(\theta_t). \end{cases} \tag{2.4}$$

One can introduce the momentum variable $m_t := \dot{p}_t$ which solves the differential equation

$$\gamma \dot{m}_t + m_t = -\nabla \mathcal{L}(\theta_t).$$

If one assumes $m_0 = 0$, this equation has the explicit solution

$$m_t = -\int_0^t \exp\left(\frac{s-t}{\gamma}\right) \nabla \mathcal{L}(\theta_s) \mathrm{d}s$$

and hence the second-order in time equation (2.4) is equivalent to the gradient memory inverse scale space flow

$$\begin{cases} \dot{p}_t = -\int_0^t \exp\left(\frac{s-t}{\gamma}\right) \nabla \mathcal{L}(\theta_s) \mathrm{d}s, \\ p_t \in \partial J(\theta_t). \end{cases} \tag{2.5}$$

For a nice overview over the derivation of gradient flows with momentum we refer to Orvieto et al. (2020).

## 2.2 From Bregman to Linearized Bregman Iterations

The starting point for the derivation of Algorithm 1 is the Bregman iteration (2.1), which is the time discretization of the inverse scale space flow (2.2).

Since the iterations (2.1) require the minimization of the loss in every iteration they are not feasible for large-scale neural networks. Therefore, we consider linearized Bregman iterations (Cai et al., 2009b), which linearize the loss function by

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta^{(k)}) + \left\langle g^{(k)}, \theta - \theta^{(k)} \right\rangle, \quad g^{(k)} := \nabla \mathcal{L}(\theta^{(k)}),$$

and replace the energy $J$ with the strongly convex elastic-net regularization

$$J_\delta(\theta) := J(\theta) + \frac{1}{2\delta}\|\theta\|^2, \quad \delta \in (0, \infty). \tag{2.6}$$

Omitting all terms which do not depend on $\theta$, the first line of (2.1) then becomes

$$\begin{aligned} \theta^{(k+1)} &= \underset{\theta \in \Theta}{\arg\min} \langle \tau^{(k)} g^{(k)}, \theta \rangle + J_\delta(\theta) - \langle v^{(k)}, \theta \rangle \\ &= \underset{\theta \in \Theta}{\arg\min} \langle \tau^{(k)} g^{(k)}, \theta \rangle + J(\theta) + \frac{1}{2\delta}\|\theta\|^2 - \langle v^{(k)}, \theta \rangle \\ &= \underset{\theta \in \Theta}{\arg\min} \frac{1}{2\delta}\left\|\theta - \delta\left(v^{(k)} - \tau^{(k)} g^{(k)}\right)\right\|^2 + J(\theta) \\ &= \mathrm{prox}_{\delta J}\left(\delta\left(v^{(k)} - \tau^{(k)} g^{(k)}\right)\right). \end{aligned} \tag{2.7}$$

The vector $v^{(k)} \in \partial J_\delta(\theta^{(k)})$ is a subgradient of the functional $J_\delta$ in the previous iterate. Using the update

$$v^{(k+1)} := v^{(k)} - \tau^{(k)} g^{(k)}$$

and combining this with (2.7) we obtain the compact update scheme

$$g^{(k)} = \nabla\mathcal{L}(\theta^{(k)}), \tag{2.8a}$$

$$v^{(k+1)} = v^{(k)} - \tau^{(k)} g^{(k)}, \tag{2.8b}$$

$$\theta^{(k+1)} = \mathrm{prox}_{\delta J}\left(\delta v^{(k+1)}\right). \tag{2.8c}$$

This iteration is an equivalent reformulation of linearized Bregman iterations (Yin et al., 2008; Cai et al., 2009b,a; Osher et al., 2010; Yin, 2010; Benning and Burger, 2018), which are usually expressed in a more complicated way. Furthermore, it coincides with the mirror descent algorithm (Beck and Teboulle, 2003) applied to the functional $J_\delta$.

Note that Yin (2010) showed that for quadratic loss functions the elastic-net regularization parameter $\delta > 0$ has no influence on the asymptotics of linearized Bregman iterations, if chosen larger than a certain threshold. This effect is referred to as *exact regularization* (Friedlander and Tseng, 2008). The iteration scheme simply computes a gradient descent in the subgradient variable $v$ and recovers the weights $\theta$ by evaluating the proximal operator of $v$. This makes it significantly cheaper than the original Bregman iterations (2.1) which require the minimization of the loss in every iteration.

Note that the last line in (2.8) is equivalent to $v^{(k+1)}$ satisfying the optimality condition

$$v^{(k+1)} \in \partial J_\delta(\theta^{(k+1)}). \tag{2.9}$$

In particular, by letting $\tau^{(k)} \to 0$ the iteration (2.8) can be viewed as explicit Euler discretization of the inverse scale space flow (2.2) of the elastic-net regularized functional $J_\delta$:

$$\begin{cases} \dot{v}_t = -\nabla\mathcal{L}(\theta_t), \\ v_t \in \partial J_\delta(\theta_t). \end{cases} \tag{2.10}$$

By embedding (2.8) into a stochastic batch gradient descent framework we obtain LinBreg from Algorithm 1.

### 2.3 Linearized Bregman Iterations with Momentum

More generally we consider an inertial version of (2.10), which as in Section 2.1 is given by

$$\begin{cases} \gamma\ddot{v}_t + \dot{v}_t = -\nabla\mathcal{L}(\theta_t), \\ v_t \in \partial J_\delta(\theta_t). \end{cases} \tag{2.11}$$

We discretize this equation in time by approximating the time derivatives as

$$\ddot{v}_t \approx \frac{v^{(k+1)} - 2v^{(k)} + v^{(k-1)}}{(\tau^{(k)})^2},$$

$$\dot{v}_t \approx \frac{v^{(k+1)} - v^{(k)}}{\tau^{(k)}},$$

such that after some reformulation we obtain the iteration

$$v^{(k+1)} = \frac{\tau^{(k)} + 2\gamma}{\tau^{(k)} + \gamma} v^{(k)} - \frac{\gamma}{\tau^{(k)} + \gamma} v^{(k-1)} - \frac{(\tau^{(k)})^2}{\tau^{(k)} + \gamma} \nabla \mathcal{L}(\theta^{(k)}), \qquad (2.12a)$$

$$\theta^{(k+1)} = \mathrm{prox}_{\delta J}(\delta v^{(k+1)}). \qquad (2.12b)$$

To see the relation to the gradient memory equation (2.5), derived in Section 2.1, we rewrite (2.12), using the new variables

$$m^{(k+1)} := v^{(k)} - v^{(k+1)}, \quad \beta^{(k)} := \frac{\gamma}{\tau^{(k)} + \gamma} \in [0, 1). \qquad (2.13)$$

Plugging this into (2.12) yields the iteration

$$m^{(k+1)} = \beta^{(k)} m^{(k)} + (1 - \beta^{(k)}) \tau^{(k)} \nabla \mathcal{L}(\theta^{(k)}), \qquad (2.14a)$$

$$v^{(k+1)} = v^{(k)} - m^{(k+1)}, \qquad (2.14b)$$

$$\theta^{(k+1)} = \mathrm{prox}_{\delta J}(\delta v^{(k+1)}). \qquad (2.14c)$$

Similar to before, embedding this into a stochastic batch gradient descent framework we obtain LinBreg with momentum from Algorithm 2. Note that, contrary to stochastic gradient descent with momentum (Orvieto et al., 2020), the momentum acts on the subgradients $v$ and not on the parameters $\theta$.

Analogously, we propose AdaBreg in Algorithm 3, which is a generalization of the Adam algorithm (Kingma and Ba, 2014). Here, we also apply the bias correction steps on the subgradient $v$ and reconstruct the parameters $\theta$ using the proximal operator of the regularizer $J$.

## 3. Analysis of Stochastic Linearized Bregman Iterations

In this section we provide a convergence analysis of stochastic linearized Bregman iterations. They are valid in a general sense and do not rely on $\mathcal{L}$ being an empirical loss or $\theta$ being weights of a neural network. Still we keep the notation fixed for clarity. All proofs can be found in the appendix.

We let $(\Omega, F, \mathbb{P})$ be a probability space, $\Theta$ be a Hilbert space, $\mathcal{L} : \Theta \to \mathbb{R}$ a Fréchet differentiable loss function, and $g : \Theta \times \Omega \to \Theta$ an unbiased estimator of $\nabla \mathcal{L}$, meaning $\mathbb{E}[g(\theta; \omega)] = \nabla \mathcal{L}(\theta)$ for all $\theta \in \Theta$. This and all other expected values are taken with respect to the random variable $\omega \in \Omega$ which, in our case, models

---

**Algorithm 2:** *LinBreg with Momentum,* an acceleration of LinBreg using momentum-based gradient memory.

---

**default:** $\delta = 1, \quad \beta = 0.9$

$\theta \leftarrow$ Section 4.1, $\quad v \leftarrow \partial J(\theta) + \frac{1}{\delta}\theta, \quad m \leftarrow 0$
 `// initialize`
**for** *epoch $e = 1$* **to** $E$ **do**
$\quad$ **for** *minibatch $B \subset \mathcal{T}$* **do**
$\quad\quad g \leftarrow \nabla L(\theta; B)$ $\qquad\qquad\qquad$ `// Backpropagation`
$\quad\quad m \leftarrow \beta\, m + (1 - \beta)\tau\, g$ $\qquad$ `// Momentum update`
$\quad\quad v \leftarrow v - m$ $\qquad\qquad\qquad\quad$ `// Momentum step`
$\quad\quad \theta \leftarrow \mathrm{prox}_{\delta J}(\delta v)$ $\qquad\qquad\;$ `// Regularization`

---

---

**Algorithm 3:** *AdaBreg,* a Bregman version of the Adam algorithm which uses moment-based bias correction.

---

**default:** $\delta = 1, \beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \epsilon = 10^{-8}$

$\theta \leftarrow$ Section 4.1, $\quad v \leftarrow \partial J(\theta) + \frac{1}{\delta}\theta, \quad m_1 \leftarrow 0, \quad m_2 \leftarrow 0$
 `// initialize`
**for** *epoch $e = 1$* **to** $E$ **do**
$\quad$ **for** *minibatch $B \subset \mathcal{T}$* **do**
$\quad\quad k \leftarrow k + 1$
$\quad\quad g \leftarrow \nabla L(\theta; B)$ $\qquad\qquad\qquad$ `// Backpropagation`
$\quad\quad m_1 \leftarrow \beta_1\, m_1 + (1 - \beta_1)\, g$ $\qquad$ `// First moment estimate`
$\quad\quad \hat{m}_1 \leftarrow m_1/(1 - \beta_1^k)$ $\qquad\quad$ `// Bias correction`
$\quad\quad m_2 \leftarrow \beta_2\, m_2 + (1 - \beta_2)\, g^2$ $\qquad$ `//Second raw moment estimate`
$\quad\quad \hat{m}_2 \leftarrow m_2/(1 - \beta_2^k)$ $\qquad\quad$ `// Bias correction`
$\quad\quad v \leftarrow v - \tau\, \hat{m}_1/(\sqrt{\hat{m}_2} + \epsilon)$ $\qquad$ `// Moment step`
$\quad\quad \theta \leftarrow \mathrm{prox}_{\delta J}(\delta v)$ $\qquad\qquad\;$ `// Regularization`

---

the randomly drawn batch of training in data in (1.10). We study the stochastic

linearized Bregman iterations

$$\text{draw } \omega^{(k)} \text{ from } \Omega \text{ using the law of } \mathbb{P}, \tag{3.1a}$$

$$g^{(k)} := g(\theta^{(k)}; \omega^{(k)}), \tag{3.1b}$$

$$v^{(k+1)} := v^{(k)} - \tau^{(k)} g^{(k)}, \tag{3.1c}$$

$$\theta^{(k+1)} := \text{prox}_{\delta J}(\delta v^{(k+1)}). \tag{3.1d}$$

For our analysis we need some assumptions on the loss function $\mathcal{L}$ which are very common in the analysis of nonlinear optimization methods. Besides boundedness from below, we demand differentiability and Lipschitz-continuous gradients, which are standard assumptions in nonlinear optimization since they allow to prove *sufficient decrease* of the loss. We refer to Benning et al. (2021) for an example of a neural network the associated loss of which satisfies the following assumption.

**Assumption 1 (Loss function)** *We assume the following conditions on the loss function:*

- *The loss function $\mathcal{L}$ is bounded from below and without loss of generality we assume $\mathcal{L} \geq 0$.*

- *The function $\mathcal{L}$ is continuously differentiable.*

- *The gradient of the loss function $\theta \mapsto \nabla\mathcal{L}(\theta)$ is L-Lipschitz for $L \in (0, \infty)$:*

$$\|\nabla\mathcal{L}(\tilde{\theta}) - \nabla\mathcal{L}(\theta)\| \leq L\|\tilde{\theta} - \theta\|, \quad \forall \theta, \tilde{\theta} \in \Theta. \tag{3.2}$$

**Remark 1** *Note that the Lipschitz continuity of the gradient in particular implies the classical estimate (Beck, 2017; Bauschke and Combettes, 2011)*

$$\mathcal{L}(\tilde{\theta}) \leq \mathcal{L}(\theta) + \langle \nabla\mathcal{L}(\theta), \tilde{\theta} - \theta \rangle + \frac{L}{2}\|\tilde{\theta} - \theta\|^2, \quad \forall \theta, \tilde{\theta} \in \Theta. \tag{3.3}$$

Furthermore, we need the following assumption, being of stochastic nature, which requires the gradient estimator to have uniformly bounded variance.

**Assumption 2 (Bounded variance)** *There exists a constant $\sigma > 0$ such that for any $\theta \in \Theta$ it holds*

$$\mathbb{E}\left[\|g(\theta; \omega) - \nabla\mathcal{L}(\theta)\|^2\right] \leq \sigma^2. \tag{3.4}$$

This is a standard assumption in the analysis of stochastic optimization methods and many authors actually demand the more restrictive condition of uniformly bounded stochastic gradients $\mathbb{E}\left[\|g(\theta; \omega)\|^2\right] \leq C$ for all $\theta \in \Theta$. Remarkably, both assumptions have been shown to be unnecessary for proving convergence of stochastic gradient descent of convex (Nguyen et al., 2018) and non-convex functions (Lei et al., 2019). Generalizing this to linearized Bregman iterations is however completely non-trivial which is why we stick to Assumption 2.

We also state our assumptions on the regularizer $J$, which are extremely mild.

**Assumption 3 (Regularizer)** *We assume that $J : \Theta \to (-\infty, \infty]$ is a convex, proper, and lower semicontinuous functional on the Hilbert space $\Theta$.*

All other assumptions will be stated when they are needed.

## 3.1 Decay of the Loss Function

We first analyze how the iteration (3.1) decreases the loss $\mathcal{L}$. Such decrease properties of deterministic linearized Bregman iterations in a different formulation were already studied by Benning et al. (2021); Benning and Burger (2018). Note that for the loss decay we do not require any sort of convexity of $\mathcal{L}$ whatsoever, but merely Lipschitz continuity of the gradient, i.e., (3.3).

**Theorem 2 (Loss decay)** *Assume that Assumptions 1–3 hold true, let $\delta > 0$, and let the step sizes satisfy $\tau^{(k)} \leq \frac{2}{\delta L}$. Then there exist constants $c, C > 0$ such that for every $k \in \mathbb{N}$ the iterates of (3.1) satisfy*

$$\mathbb{E}\left[\mathcal{L}(\theta^{(k+1)})\right] + \frac{1}{\tau^{(k)}}\mathbb{E}\left[D_J^{\text{sym}}(\theta^{(k+1)}, \theta^{(k)})\right] + \frac{C}{2\delta\tau^{(k)}}\mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right]$$
$$\leq \mathbb{E}\left[\mathcal{L}(\theta^{(k)})\right] + \tau^{(k)}\delta\frac{\sigma^2}{2c}, \tag{3.5}$$

**Corollary 3 (Summability)** *Under the conditions of Theorem 2 and with the additional assumption that the step sizes are non-increasing and square-summable, meaning*

$$\tau^{(k+1)} \leq \tau^{(k)}, \quad \forall k \in \mathbb{N}, \qquad \sum_{k=0}^{\infty} (\tau^{(k)})^2 < \infty,$$

*it holds*

$$\sum_{k=0}^{\infty} \mathbb{E}\left[D_J^{\text{sym}}(\theta^{(k+1)}, \theta^{(k)})\right] < \infty, \qquad \sum_{k=0}^{\infty} \mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right] < \infty.$$

**Remark 4 (Deterministic case)** *Note that in the deterministic setting with $\sigma = 0$ the statement of Theorem 2 coincides with Benning et al. (2021); Benning and Burger (2018). In particular, the loss decays monotonously and one gets stronger summability than in Corollary 3 since one does not have to multiply by $\tau^{(k)}$.*

## 3.2 Convergence of the Iterates

In this section we establish two convergence results for the iterates of the stochastic linearized Bregman iterations (3.1). According to common practice and for self-containedness we restrict ourselves to strongly convex losses. Obviously, our results remain true for non-convex losses if one assumes convexity around local

minima and applies our arguments locally. We note that one could also extend the deterministic convergence proof of Benning et al. (2021)—which is based on the Kurdyka–Łojasiewicz (KL) inequality and works for non-convex losses—to the stochastic setting. However, this is beyond the scope of this paper and conveys less intuition than our proofs.

For our first convergence result—asserting norm convergence of a subsequence—we need the condition on the loss function Assumption 1, the bounded variance condition from Assumption 2 and strong convexity of the loss $\mathcal{L}$:

**Assumption 4 (Strong convexity)** *The loss function $\theta \mapsto \mathcal{L}(\theta)$ is $\mu$-strongly convex for $\mu \in (0, \infty)$, meaning*

$$\mathcal{L}(\tilde{\theta}) \geq \mathcal{L}(\theta) + \langle \nabla \mathcal{L}(\theta), \tilde{\theta} - \theta \rangle + \frac{\mu}{2} \|\tilde{\theta} - \theta\|^2, \quad \forall \theta, \tilde{\theta} \in \Theta. \tag{3.6}$$

Note that by virtue of (3.3) it holds $\mu \leq L$ if the loss satisfies Assumptions 1 and 4.

Our second convergence convergence result—asserting convergence in the Bregman distance of $J_\delta$ which is a stronger topology than norm convergence—requires a stricter convexity condition, tailored to the Bregman geometry.

**Assumption 5 (Strong Bregman convexity)** *The loss function $\theta \mapsto \mathcal{L}(\theta)$ satisfies*

$$\mathcal{L}(\tilde{\theta}) \geq \mathcal{L}(\theta) + \langle \nabla \mathcal{L}(\theta), \tilde{\theta} - \theta \rangle + \nu D_{J_\delta}^v(\tilde{\theta}, \theta), \quad \forall \theta, \tilde{\theta} \in \Theta, \ v \in \partial J_\delta(\theta), \tag{3.7}$$

*where $J_\delta$ for $\delta > 0$ is defined in (2.6). In particular, $\mathcal{L}$ satisfies Assumption 4 with $\mu = \nu/\delta$.*

**Remark 5 (The Bregman convexity assumption)** *Assumption 5 seems to be quite restrictive, however, in finite dimensions it is locally equivalent to Assumption 4, as we argue in the following. Note that it suffices if the assumptions above are satisfied in a vicinity of the (local) minimum to which the algorithm converges. For proving convergence we will use Assumption 5 with $\tilde{\theta} = \theta^*$, a (local) minimum, and $\theta$ close to $\theta^*$. Using Lemma 13 in the appendix the assumption can be rewritten as*

$$\mathcal{L}(\theta^*) \geq \mathcal{L}(\theta) + \langle \nabla \mathcal{L}(\theta), \theta^* - \theta \rangle + \frac{\nu}{2\delta} \|\theta^* - \theta\|^2 + \nu D_J^p(\theta^*, \theta), \quad p \in \partial J(\theta)$$

*and we will argue that for $\theta$ close to $\theta^*$ the extra term vanishes, i.e. $D_J^p(\theta^*, \theta) = 0$. For this we have to show that $p$ is not only a subgradient at $\theta$ but also at $\theta^*$: If $p$ is a subgradient of $J$ at both points, i.e., $p \in \partial J(\theta) \cap \partial J(\theta^*)$ we obtain that their Bregman distance is zero. This can be seen using the definition of the Bregman distance (1.17):*

$$\left. \begin{array}{r} D_J^p(\theta^*, \theta) \geq 0 \\ D_J^p(\theta, \theta^*) \geq 0 \end{array} \right\} \implies 0 \leq D_J^p(\theta^*, \theta) + D_J^p(\theta, \theta^*) = 0.$$

In finite dimensions and for $J(\theta) = \|\theta\|_1 = \sum_{i=1}^{N} |\theta_i|$ equal to the $\ell_1$-norm we can use that $\langle p, \theta \rangle = J(\theta) = \sum_{i=1}^{N} \text{sign}(\theta_i)\theta_i = \sum_{i=1}^{N} |\theta_i| = J(\theta)$ and simplify the Bregman distance to

$$D_J^p(\theta^*, \theta) = J(\theta^*) - \langle p, \theta^* \rangle = \sum_{i=1}^{N} |\theta_i^*| - \text{sign}(\theta_i)\theta_i^* = \sum_{i=1}^{N} \theta_i^* \left( \text{sign}(\theta_i^*) - \text{sign}(\theta_i) \right).$$

Obviously, the terms in this sum where $\theta_i^* = 0$ vanish anyways. Hence, the expression is zero whenever the non-zero entries of $\theta$ have the same sign as those of $\theta^*$ which is the case if $\|\theta - \theta^*\|_\infty < \min \{|\theta_i^*| : i \in \{1, \ldots, N\}, \theta_i^* \neq 0\}$. Since all norms are equivalent in finite dimensions, one obtains

$$D_J^p(\theta^*, \theta) = 0 \quad \text{for } \|\theta - \theta^*\| \text{ sufficiently small.}$$

Hence, Assumption 5 is locally implied by Assumption 4 if $J(\theta) = \|\theta\|_1$.

We would like to remark that—even under the weaker Assumption 4—one needs some coupling of the loss $\mathcal{L}$ and the regularization functional $J$ in order to obtain convergence to a critical point. Benning et al. (2021) demand that a surrogate function involving both $\mathcal{L}$ and $J$ admits the KL inequality and that the subgradients of $J$ are bounded close to the minimum $\theta^*$ of the loss. Indeed, in our theory using Assumption 4 it suffices to demand that $J(\theta^*) < \infty$. This assumption is weaker than assuming bounded subgradients but is nevertheless necessary as the following example taken from Benning et al. (2021) shows.

**Example 1 (Non-convergence to a critical point)** *Let $\mathcal{L}(\theta) = (\theta+1)^2$ for $\theta \in \mathbb{R}$ and $J(\theta) = \chi_{[0,\infty)}(\theta)$ be the characteristic function of the positive axis. Then for any initialization the linearized Bregman iterations (2.8) converge to $\theta = 0$ which is no critical point of $\mathcal{L}$. On the other hand, the only critical point $\theta^* = -1$ clearly meets $J(\theta^*) = \infty$.*

**Theorem 6 (Convergence in norm)** *Assume that Assumptions 1–4 hold true and let $\delta > 0$. Furthermore, assume that the step sizes $\tau^{(k)}$ are such that for all $k \in \mathbb{N}$:*

$$\tau^{(k)} \leq \frac{\mu}{2\delta L^2}, \qquad \tau^{(k+1)} \leq \tau^{(k)}, \qquad \sum_{k=0}^{\infty} (\tau^{(k)})^2 < \infty, \qquad \sum_{k=0}^{\infty} \tau^{(k)} = \infty.$$

*The function $\mathcal{L}$ has a unique minimizer $\theta^*$ and if $J(\theta^*) < \infty$ the stochastic linearized Bregman iterations (3.1) satisfy the following:*

- *Letting $d_k := \mathbb{E}\left[ D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)}) \right]$ it holds*

$$d_{k+1} - d_k + \frac{\mu}{4}\tau^{(k)}\mathbb{E}\left[ \|\theta^* - \theta^{(k+1)}\|^2 \right] \leq \frac{\sigma}{2} \left( (\tau^{(k)})^2 + \mathbb{E}\left[ \|\theta^{(k)} - \theta^{(k+1)}\|^2 \right] \right).$$
$$(3.8)$$

- *The iterates possess a subsequence converging in the $L^2$-sense of random variables:*

$$\lim_{j \to \infty} \mathbb{E}\left[\|\theta^* - \theta^{(k_j)}\|^2\right] = 0. \tag{3.9}$$

*Here, $J_\delta$ is defined as in (2.6).*

**Remark 7 (Choice of step sizes)** *A possible step size which satisfies the conditions of Theorem 6 is given by $\tau^{(k)} = \frac{c}{(k+1)^p}$ where $0 < c < \frac{\mu}{\delta L^2}$ and $p \in (\frac{1}{2}, 1]$.*

**Remark 8 (Deterministic case)** *In the deterministic case $\sigma = 0$ inequality (3.8) even shows that the Bregman distances decrease along iterations. Furthermore, in this case it is not necessary that the step sizes are square-summable and non-increasing since the term on the right hand side does not have to be summed.*

In a finite dimensional setting and using $\ell_1$-regularization one can even show convergence of the whole sequence of Bregman distances.

**Remark 9** *With the help of Lemma 13 in the appendix, the quantity $D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})$ which appears in the decay estimate (3.8) can be simplified as follows:*

$$
\begin{aligned}
D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)}) &= \frac{1}{2\delta}\|\theta^* - \theta^{(k)}\|^2 + D_J^{p^{(k)}}(\theta^*, \theta) \\
&= \frac{1}{2\delta}\|\theta^* - \theta^{(k)}\|^2 + J(\theta^*) - J(\theta^{(k)}) - \langle p^{(k)}, \theta^* - \theta^{(k)}\rangle,
\end{aligned}
$$

*where $p^{(k)} := v^{(k)} - \frac{1}{\delta}\theta^{(k)} \in \partial J(\theta^{(k)})$. In the case that $J$ is absolutely 1-homogeneous, e.g., if $J(\theta) = \|\theta\|_1$ equals the $\ell_1$-norm, this simplifies to*

$$D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)}) = \frac{1}{2\delta}\|\theta^* - \theta^{(k)}\|^2 + J(\theta^*) - \langle p^{(k)}, \theta^*\rangle,$$

*where we used that for absolutely 1-homogeneous functionals it holds $\langle p, \theta\rangle = J(\theta)$ for all $\theta \in \Theta$ and $p \in \partial J(\theta)$. Hence, it measures both the convergence of $\theta^{(k)}$ to $\theta^*$ in the norm and the convergence of the subgradients $p^{(k)} \in \partial J(\theta^{(k)})$ to a subgradient of $J$ at $\theta^*$.*

**Corollary 10 (Convergence in finite dimensions)** *If the parameter space $\Theta$ is finite dimensional and $J$ equals the $\ell_1$-norm, under the conditions of Theorem 6 it even holds $\lim_{k \to \infty} d_k = 0$ which in particular implies $\lim_{k \to \infty} \mathbb{E}\left[\|\theta^* - \theta^{(k)}\|^2\right] = 0$.*

Our second convergence theorem asserts convergence in the Bregman distance and gives quantitative estimates under Assumption 5, which is a stricter assumption than Assumption 4 and relates the loss function $\mathcal{L}$ with the regularizer; cf. Dragomir et al. (2021) for a related approach working with $C^2$ functions. The theorem states that the Bregman distance to the minimizer of the loss can be made arbitrarily small using constant step sizes. For step sizes which go to zero and are not summable one obtains a quantitative convergence result.

**Theorem 11 (Convergence in the Bregman distance)** *Assume that Assumptions 1–3 and 5 hold true and let $\delta > 0$. The function $\mathcal{L}$ has a unique minimizer $\theta^*$ and if $J(\theta^*) < \infty$ the stochastic linearized Bregman iterations (3.1) satisfy the following:*

- *Letting $d_k := \mathbb{E}\left[ D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)}) \right]$ it holds*

$$d_{k+1} \leq \left[1 - \tau^{(k)}\nu \left(1 - \tau^{(k)}\frac{2\delta^2 L^2}{\nu}\right)\right] d_k + \delta(\tau^{(k)})^2\sigma^2. \qquad (3.10)$$

- *For any $\varepsilon > 0$ there exists $\tau > 0$ such that if $\tau^{(k)} = \tau$ for all $k \in \mathbb{N}$ then*

$$\limsup_{k\to\infty} d_k \leq \varepsilon. \qquad (3.11)$$

- *If $\tau^{(k)}$ is such that*

$$\lim_{k\to\infty} \tau^{(k)} = 0 \quad and \quad \sum_{k=0}^{\infty} \tau^{(k)} = \infty \qquad (3.12)$$

*then it holds*

$$\lim_{k\to\infty} d_k = 0. \qquad (3.13)$$

*Here, $J_\delta$ is defined as in (2.6).*

**Corollary 12 (Convergence rate for diminishing step sizes)** *The error recursion (3.10) coincides with the one for stochastic gradient descent. In particular, for step sizes of the form $\tau^{(k)} = \frac{c}{k}$ with a suitably small constant $c > 0$ one can prove with induction (Nemirovski et al., 2009) that $d_k = \frac{C}{k}$ for some $C > 0$.*

## 4. Numerical Experiments

In this section we perform an extensive evaluation of our algorithms focusing on different characteristics. First, we derive a sparse initialization strategy in Section 4.1, using similar statistical arguments as in the seminal works by Glorot and Bengio (2010); He et al. (2015). In Sections 4.2 and 4.3 we study the influence of hyperparameters and compare our family of algorithms with standard stochastic gradient descent (SGD) and the sparsity promoting proximal gradient descent method. In Sections 4.4 and 4.5 we demonstrate that our algorithms generate sparse and expressive networks for solving the classification task on Fashion-MNIST and CIFAR-10, for which we utilize state-of-the-art CNN and ResNet architectures. Finally, in Section 4.6 we show that, using row sparsity, our Bregman learning algorithm allows to discover a denoising autoencoder architecture, which shows the potential of the method for architecture design. Our code is available on `GitHub` at `https://github.com/TimRoith/BregmanLearning` and relies on `PyTorch` (Paszke et al., 2019).

## 4.1 Initialization

Parameter initialization for neural networks has a crucial influence on the training process, see Glorot and Bengio (2010). In order to tackle the problem of vanishing and exploding gradients, standard methods consider the variance of the weights at initialization (Glorot and Bengio, 2010; He et al., 2016), assuming that for each $l$ the entries $W_{i,j}^l$ are i.i.d. with respect to a probability distribution. The intuition here is to preserve the variances over the forward and the backward pass similar to the variance of the respective input of the network, see Glorot and Bengio (2010, Sec. 4.2). If the distribution satisfies $\mathbb{E}\left[W^l\right] = 0$, this yields a condition of the form

$$\mathrm{Var}\left[W^l\right] = \alpha(n_l, n_{l-1}) \tag{4.1}$$

where the function $\alpha$ depends on the activation function. For anti-symmetric activation functions with $\sigma'(0) = 1$, as for instance a sigmoidal function, it was shown by Glorot and Bengio (2010) that

$$\alpha(n_l, n_{l-1}) = \frac{2}{n_l \cdot n_{l-1}}$$

while for ReLU He et al. (2016) suggest to use

$$\alpha(n_l, n_{l-1}) = \frac{2}{n_l} \quad \text{or} \quad \alpha(n_l, n_{l-1}) = \frac{2}{n_{l-1}}.$$

For our Bregman learning framework we have to adapt this argumentation, taking into account sparsity. For classical inverse scale space approaches and Bregman iterations of convex losses, as for instance used for image reconstruction (Osher et al., 2005) and compressed sensing (Yin et al., 2008; Burger et al., 2013; Osher et al., 2010; Cai et al., 2009b), one would initialize all parameters equal to zero. However, for neural networks this yields an unbreakable symmetry of the network parameters, which makes training impossible, see, e.g., Goodfellow et al. (2016, Ch. 6). Instead, we employ an established approach for sparse neural networks (see, e.g., Liu et al. (2021); Dettmers and Zettlemoyer (2019); Martens (2010)) which masks the initial parameters, i.e.,

$$W^l := \tilde{W}^l \odot M^l.$$

Here, the mask $M^l \in \mathbb{R}^{n_l, n_{l-1}}$ is chosen randomly such that each entry is distributed according to the Bernoulli distribution with a parameter $r \in [0, 1]$, i.e.,

$$M_{i,j}^l \sim \mathcal{B}(r).$$

The parameter $r$ coincides with the expected percentage of non-zero weights

$$\mathrm{N}(W^l) := \frac{\|W^l\|_0}{n_l \cdot n_{l-1}} = 1 - \mathrm{S}(W^l), \tag{4.2}$$

where $S(W^l)$ denotes the sparsity. In the following we derive a strategy to initialize $\tilde{W}^l$. Choosing $\tilde{W}^l$ and $M^l$ independent and using $\mathbb{E}\left[\tilde{W}^l\right] = 0$ standard variance calculus implies

$$
\begin{aligned}
\mathrm{Var}\left[W^l\right] &= \mathrm{Var}\left[\tilde{W}^l \odot M^l\right] \\
&= \mathbb{E}\left[M^l\right]^2 \mathrm{Var}\left[\tilde{W}^l\right] + \underbrace{\mathbb{E}\left[\tilde{W}^l\right]^2 \mathrm{Var}\left[M^l\right]}_{=0} + \mathrm{Var}\left[M^l\right] \mathrm{Var}\left[\tilde{W}^l\right] \\
&= \left(\mathbb{E}\left[M^l\right]^2 + \mathrm{Var}\left[M^l\right]\right) \mathrm{Var}\left[\tilde{W}^l\right] \\
&= \mathbb{E}\left[(M^l)^2\right] \mathrm{Var}\left[\tilde{W}^l\right] \\
&= r\,\mathrm{Var}\left[\tilde{W}^l\right]
\end{aligned}
$$

and thus deriving from (4.1) we obtain the condition

$$
\mathrm{Var}\left[\tilde{W}^l\right] = \frac{1}{r}\,\alpha(n_l, n_{l-1}). \tag{4.3}
$$

Instead of having linear feedforward layers with corresponding weight matrices, the neural network architecture at hand might consist of other groups of parameters which one would like to keep sparse, e.g., using the group sparsity regularization (1.2). For example, in a convolutional neural network one might be interested in having only a few number of non-zero convolution kernels in order to obtain compact feature representations of the input. Similarly, for standard feedforward architectures sparsity of rows of the weight matrices yields compact networks with a small number of active neurons. In such cases one can apply the same arguments as above and initialize single elements $\mathbf{g} \in \mathcal{G}$ of a parameter group $\mathcal{G}$ as non-zero with probability $r \in [0,1]$ and with respect to a variance condition similar to (4.3):

$$
\mathbf{g} = \tilde{\mathbf{g}} \cdot m, \quad m \sim \mathcal{B}(r), \tag{4.4}
$$

$$
\mathrm{Var}\left[\tilde{\mathbf{g}}\right] = \frac{1}{r}\alpha(\mathbf{g}) \tag{4.5}
$$

Note that these arguments are only valid in a linear regime around the initialization, see Glorot and Bengio (2010) for details. Hence, if one initializes with sparse parameters but optimizes with very weak regularization, e.g., by using vanilla SGD or choosing $\lambda$ in (1.1) or (1.2) very small, the assumption is violated since the sparse initial parameters are rapidly filled during the first training steps.

The biases are initialized non-sparse and the precise strategy depends on the activation function. We would like to emphasize that initializing biases with zero is not a good idea in the context of sparse neural networks. In this case, the neuron activations would be equal for all "dead neurons" whose incoming weights are zero,

which would then yield an unbreakable symmetry. For ReLU we initialize biases with positive random values to ensure a flow of information and to break symmetry also for dead neurons, which is similar to the strategy proposed by Goodfellow et al. (2016, Ch. 6). For other activation functions $\sigma$ which meet $\sigma(x) = 0$ if and only if $x = 0$, e.g., TanH, Sigmoid, or Leaky ReLU, biases can be initialized with random numbers of arbitrary sign.
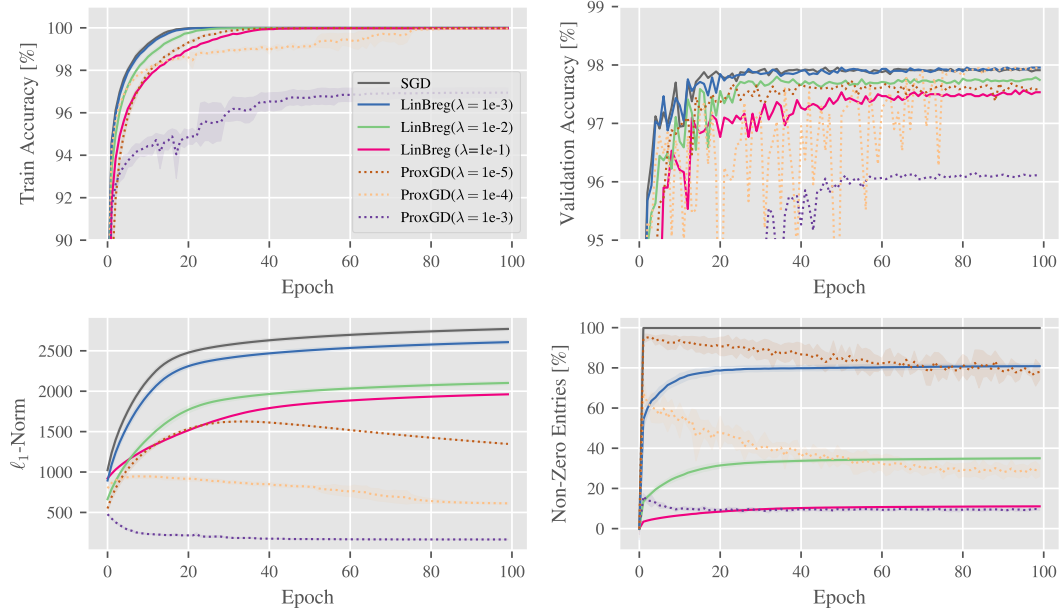
## 4.2 Comparison of Algorithms



Figure 2: Comparison of vanilla SGD (black solid line), LinBreg (colored solid lines), and ProxGD (colored dotted lines) for different regularization parameters on MNIST. The curves show the averaged accuracies on train and validation sets, $\ell_1$-norms, and non-zero entries over three runs. The shaded area visualizes the standard deviation.

We start by comparing the proposed LinBreg Algorithm 1 with vanilla stochastic gradient descent (SGD) without sparsity regularization and with the Lasso-based approach from Scardapane et al. (2017), for which we compute solutions to the sparsity-regularized risk minimization problem (1.4) using the proximal gradient descent algorithm (ProxGD) from (1.5).

We consider the classification task on the MNIST dataset (LeCun and Cortes, 2010) for studying the impact of the hyperparameters of these methods. The set consists of $60,000$ images of handwritten digits which we split into $55,000$ images used for the training and $5,000$ images used for a validation process during training.

We train a fully connected net with ReLU activations and two hidden layers (200 and 80 neurons), and use the $\ell_1$-regularization from (1.13),

$$J(\theta) = \lambda \sum_{l=1}^{L} \|W^l\|_{1,1}$$

In Figure 2 we compare the training results of vanilla SGD, the proposed LinBreg, and the ProxGD algorithm. Following the strategy introduced in Section 4.1 we initialize the weights with 1% non-zero entries, i.e., $r = 0.01$. The learning rate is chosen as $\tau = 0.1$ and is multiplied by a factor of 0.5 whenever the validation accuracy stagnates. For a fair comparison the training is executed for three different fixed random seeds, and the plot visualizes mean and standard deviation of the three runs, respectively. We show the training and validation accuracies, the $\ell_1$-norm, and the overall percentage of non-zero weights. Note that for the validation accuracies we do not show standard deviations for the sake of clarity.

While SGD without sparsity regularization instantaneously destroys sparsity, LinBreg exhibits the expected inverse scale space behaviour, where the number of non-zero weights gradually grows during training, and the train accuracy increases monotonously. This is suggested by Theorem 2, even though our experimental setup, in particular the non-smooth ReLU activation functions, is not covered by the theoretical framework which require at least $L$-smoothness of the loss functions.

In contrast, ProxGD shows no monotonicity of training accuracy or sparsity and the validation accuracies oscillate heavily. Instead, it adds a lot of non-zero weights in the beginning and then gradually reduces them. Obviously, the regularized empirical risk minimization (1.4) implies a trade-off between training accuracy and sparsity which depends on $\lambda$. For LinBreg this trade-off is neither predicted by theory nor observed numerically. Here, the regularization parameter only induces a trade-off between *validation* accuracy and sparsity, which is to be expected.

LinBreg (blue curves) can generate networks whose validation accuracy equals the one of a full network and use only 80% of the weights. For the largest regularization parameter (magenta curves) LinBreg uses only 10% of the weights and still does not drop more than half a percentage point in validation accuracy.

### 4.3 Accelerated Bregman Algorithms

We use the same setup as in the previous section to compare LinBreg with its momentum-based acceleration and AdaBreg (see Algorithms 1–3). Using the regularization parameter $\lambda = 10^{-1}$ from the previous section (see the magenta curves), Figure 3 shows the validation accuracy of the different networks trained with LinBreg, LinBreg with momentum, and AdaBreg. For comparison we visualize again the results of vanilla SGD as gray curve. It is obvious that all three proposed algorithms generate very accurate networks using approximately 10% of the weights. As expected, the accelerated versions increase both the validation accuracy and the number of non-zero parameters faster than the baseline algorithm LinBreg. While after
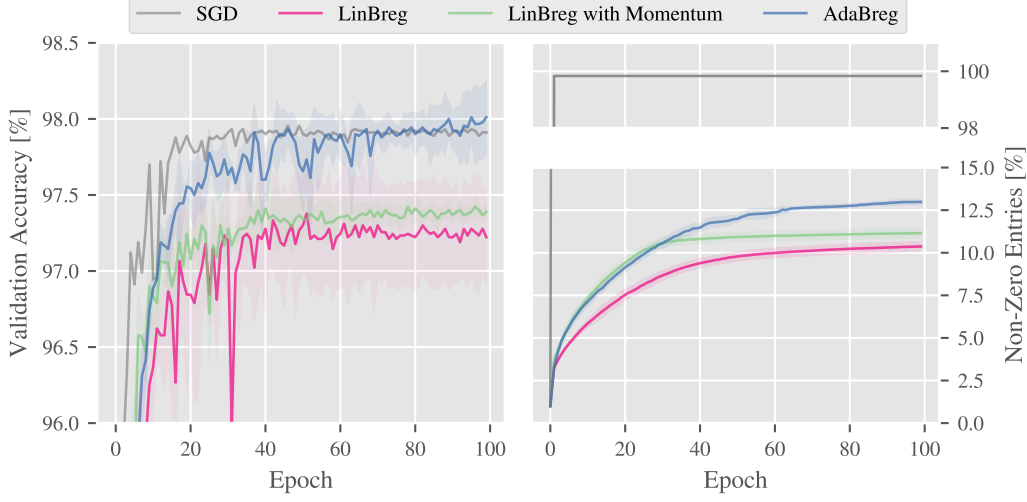
Figure 3: Comparison of LinBreg (with momentum), AdaBreg, and vanilla SGD. The networks generated by AdaBreg are sparse and generalize better.

100 epochs LinBreg and its momentum version have slightly lower validation accuracies than the non-sparse networks generated by SGD, AdaBreg outperforms the other algorithms including SGD in terms of validation accuracy while maintaining a high degree of sparsity.

### 4.4 Sparsity for Convolutional Neural Networks (CNNs)

In this example we apply our algorithms to a convolutional neural network of the form

$$5 \times 5 \text{ conv, } 64 \stackrel{\text{Maxpool/2}}{\longrightarrow} 5 \times 5 \text{ conv, } 64 \stackrel{\text{Maxpool/2}}{\longrightarrow} \text{fc } 1024 \longrightarrow \text{fc } 128 \longrightarrow \text{fc } 10$$

with ReLU activations to solve the classification task on Fashion-MNIST. We run experiments both for sparsity regularization utilizing the $\ell_1$-norm (1.1) and for a combination of the $\ell_1$-norm on the linear layers and the group $\ell_{1,2}$-norm (1.2) on the convolutional kernels. This way, we aim to obtain compressed network architectures with only few active convolutional filters.

The inverse scale space character of our algorithms is visualized in Figure 1 from the beginning of the paper which shows the 64 feature maps of an input image, generated by the first convolutional layer of the network after 0, 5, 20, and 100 epochs of LinBreg with group sparsity. One can observe that gradually more kernels are added until iteration 20, from where on the number of kernels stays fixed and the kernels themselves are optimized.

Tables 1 and 2 shows the test and training accuracies as well as sparsity levels. For plain sparsity regularization we only show the total sparsity level of all network

parameters whereas for the group sparsity regularization we show the sparsity of the linear layers and the relative number of non-zero convolutional kernels.

A convolutional layer for a input $z \in \mathbb{R}^{c_{l-1}, n_{l-1}, m_{l-1}}$ is given as

$$\Phi_j^l(z) = b_j + \sum_{i=1}^{c_{l-1}} K_{i,j}^l * z_{i,\bullet},$$

where $K_{i,j}^l \in \mathbb{R}^{k,k}$ denote kernel matrices with corresponding biases $b_j \in \mathbb{R}^{n_l, m_l}$ for in-channels $i \in \{1, \ldots, c_{l-1}\}$ and out-channels $j \in \{1, \ldots, c_l\}$. Therefore, we denote by

$$N_{\text{conv}} := \frac{\sum_{l \in I_{\text{conv}}} \#\{K_{i,j}^l : K_{i,j}^l \neq \mathbf{0}\}}{\sum_{l \in I_{\text{conv}}} c_l \cdot c_{l-1}}$$

the percentage of non-zero kernels of the whole net where $I_{\text{conv}}$ denotes the index set of the convolutional layers. Analogously, using a similar term as in (4.2) we denote by

$$N_{\text{linear}} := \frac{\sum_{l \in I_{\text{linear}}} \|W^l\|_0}{\sum_{l \in I_{\text{linear}}} n_l \cdot n_{l-1}}$$

the percentage of weights used in the linear layers. Finally, we define $N_{\text{total}} := N_{\text{conv}} + N_{\text{linear}}$.

We compare our algorithms LinBreg (with momentum) and AdaBreg against vanilla training without sparsity, iterative pruning (Han et al., 2015), and the Group Lasso approach from Scardapane et al. (2017), and train all networks to a comparable sparsity level, given in brackets. The pruning scheme is taken from Han et al. (2015), where in each step a certain amount of weights is pruned, followed by a retraining step. For our experiment the amount of weights pruned in each iteration was chosen, so that a specified target sparsity is met. For the Group Lasso approach, which is based on the regularized risk minimization (1.4), we use two different optimizers. First, we apply SGD applied to the (1.4) and apply thresholding afterwards to obtain sparse weights, which is the standard approach in the community (cf. Scardapane et al. (2017)). Second, we apply proximal gradient descent (1.5) to (1.4) which yields sparse solutions without need for thresholding. Our Bregman algorithms were initialized with 1% non-zero parameters, following the strategy from Section 4.1, all other algorithms were initialized non-sparse using standard techniques (Glorot and Bengio, 2010; He et al., 2015). For all algorithms we tuned the hyperparameters (e.g., pruning rate, regularization parameter for Group Lasso and Bregman) in order to achieve comparable sparsity levels.

Note that it is non-trivial to compare different algorithms for sparse training since they optimize different objectives, and both the sparsity, the train, and the test accuracy of the resulting networks matter. Therefore, we show results whose sparsity levels and accuracies are in similar ranges.

Table 1 shows that all algorithms manage to compute very sparse networks with ca. 2% drop in test accuracy on Fasion-MNIST, compared to vanilla dense training with Adam. Note that we optimized the hyperparameters (regularization and thresholding parameters) of all algorithms for optimal performance on a validation set, subject to having comparable sparsity levels. Our algorithms LinBreg and AdaBreg yield sparser networks with the same accuracies as Pruning and Lasso.

Similar observations are true for Table 2 where we used group sparsity regularization on the convolutional kernels. Here all algorithms apart from pruning yield similar results, whereas pruning exhibits a significantly worse test accuracy despite using a larger number of non-zero parameters. The combination of SGD-optimized Group Lasso with subsequent thresholding yields the best test accuracy using a moderate sparsity level.

As mentioned above the Lasso and Group Lasso results using SGD underwent an additional thresholding step after training in order to generate sparse solutions. Obviously, one could also do this with the results of ProxGD and Bregman which would further improve their sparsity levels. However, in this experiment we refrain from doing so in order not to change the nature of the algorithms.

| Strategy | Optimizer | $N_{total}$ in [%] | Test Acc | Train Acc |
|---|---|---|---|---|
| Vanilla | Adam | 100 | 92.1 | 100.0 |
| Pruning (5%) | SGD | 4.7 | 89.2 | 92.0 |
| Lasso | SGD + thresh. | 3.5 | 90.1 | 94.7 |
| | ProxGD | 4.8 | 89.4 | 91.4 |
| **Bregman** | LinBreg | 1.9 | 89.2 | 91.1 |
| | LinBreg ($\beta = 0.9$) | 2.7 | 89.9 | 93.8 |
| | AdaBreg | 2.3 | 90.5 | 93.6 |

Table 1: Sparsity levels and accuracies on the Fashion-MNIST data set.

| Strategy | Optimizer | $N_{linear}$ in [%] | $N_{conv}$ in [%] | Test Acc | Train Acc |
|---|---|---|---|---|---|
| Vanilla | Adam | 100 | 100 | 92.1 | 100.0 |
| Pruning (7%) | SGD | 7.0 | 6.5 | 86.9 | 89.9 |
| GLasso | SGD + thresh. | 3.6 | 4.3 | 90.3 | 94.8 |
| | ProxGD | 3.0 | 3.7 | 89.8 | 91.6 |
| **Bregman** | LinBreg | 3.8 | 4.2 | 89.5 | 93.1 |
| | LinBreg ($\beta = 0.9$) | 3.5 | 4.7 | 89.9 | 93.5 |
| | AdaBreg | 3.5 | 2.8 | 89.4 | 92.6 |

Table 2: Group sparsity levels and accuracies on the Fashion-MNIST data set.

## 4.5 Residual Neural Networks (ResNets)

In this experiment we trained a ResNet-18 architecture for classification on CIFAR-10, enforcing sparsity through the $\ell_1$-norm (1.1) and comparing different strategies,

as before. Table 3 shows the resulting sparsity levels of the total number of parameters and the percentage of non-zero convolutional kernels as well as the train and test accuracies. Note that even though we used the standard $\ell_1$ regularization (1.1) and no group sparsity, the trained networks exhibit large percentages of zero-kernels.

For comparison we also show the unregularized vanilla results using SGD with momentum and Adam, which both use 100% of the parameters. The LinBreg result with thresholding shows that one can train a very sparse network using only 3.4% of all parameters with 3.4% drop in test accuracy. With AdaBreg we obtain a sparsity level of 14.7%, resulting in a drop of only 1.3%. The combination of Adam-optimized Lasso with subsequent thresholding yields a 3% sparsity with a drop of 3.6% in test accuracy, which is the sparsest result in this comparison.

| Strategy | Optimizer | $N_{total}$ in [%] | $N_{conv}$ in [%] | Test Acc | Train Acc |
|---|---|---|---|---|---|
| Vanilla | SGD with momentum | 100.0 | 100.0 | 92.15 | 99.8% |
| | Adam | 100.0 | 100.0 | 93.6 | 100.0% |
| Lasso | Adam | 99.7 | 100.0 | 91.1 | 100 |
| | Adam + thresh. | 3.0 | 15.7 | 90.0 | 99.8 |
| **Bregman** | LinBreg | 5.5 | 24.8 | 90.9 | 99.5 |
| | LinBreg + thresh. | 3.4 | 16.9 | 90.2 | 99.4 |
| | LinBreg ($\beta = 0.9$) | 4.8 | 21.0 | 90.4 | 100.0 |
| | LinBreg ($\beta = 0.9$) + thresh. | 3.6 | 17.4 | 90.0 | 99.9 |
| | AdaBreg | 14.7 | 56.7 | 92.3 | 100.0 |
| | AdaBreg + thresh. | 9.2 | 42.2 | 90.5 | 99.9 |

Table 3: Sparsity levels and accuracies on the CIFAR-10 data set.

### 4.6 Towards Architecture Design: Unveiling an Autoencoder

In this final experiment we investigate the potential of our Bregman training framework for architecture design, which refers to letting the network learn its own architecture. Hoefler et al. (2021) identified this as one of the main potentials of sparse training.

The inverse scale space character of our approach turns out to be promising for starting with very sparse networks and letting the network choose its own architecture by enforcing, e.g., row sparsity of weight matrices. The simplest yet widely used non-trivial architecture one might hope to train is an autoencoder, as used, e.g., for denoising images. To this end, we utilize the MNIST data set and train a fully connected feedforward network with five hidden layers, all having the same dimension as the input layer, to denoise MNIST images. Similar to the experiments in Section 4.2 we split the dataset in 55,000 images used for training and 5,000 images to evaluate the performance. We enforce row sparsity by using the regular-

izer (1.14), which in this context is equivalent to having few active neurons in the network.

Figure 4 shows the number of active neurons in the network at different stages of the training process using LinBreg. Here darker colors indicate more iterations. We initialized around 1% of all rows non-zero, which corresponds to 8 neurons per layer being initially active. Our algorithm successively adds neurons and converges to an autoencoder-like structure, where the number of neurons decreases until the middle layer and then increases again. Note the network developed this structure "on its own" and that we *did not* artificially generate this result by using different regularization strengths for the different layers. In Figure 5 we additionally show the denoising performance of the trained network on some images from the MNIST test set. The network was trained for 100 iterations, using a regularization value of $\lambda = 0.07$ in (1.14) and employing a standard MSE loss. We evaluate the test performance using the established structural similarity index measure (SSIM) (Wang et al., 2004), which assigns values close to 1 to pairs of images that are perceptually similar. Averaging this value over the whole test set we report a value of SSIM $\approx$ 0.93. For comparison, we also trained a network with 100 iterations of standard SGD which yields no sparsity and a value of SSIM $\approx$ 0.89.
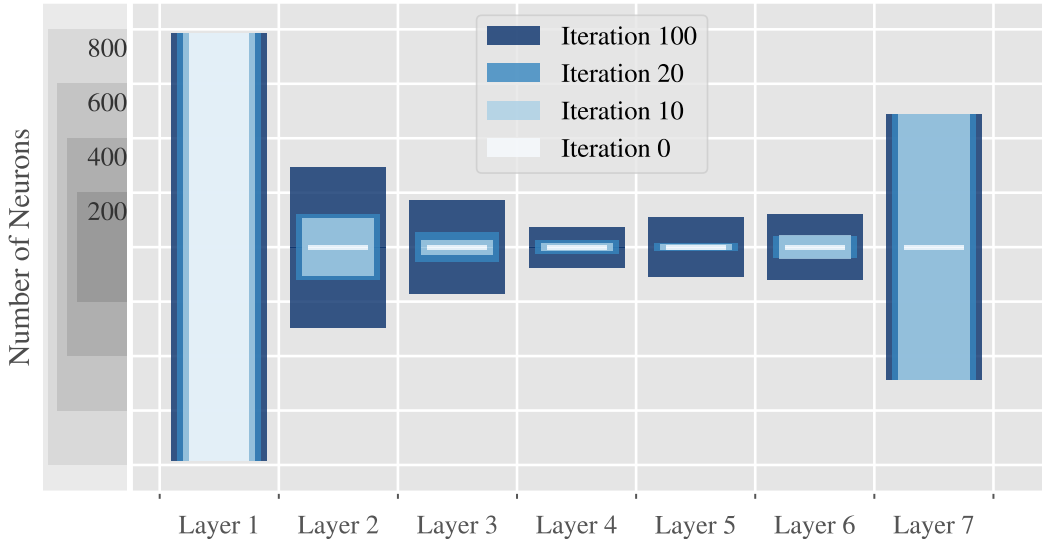


Figure 4: Architecture design for denoising: LinBreg automatically unveils an autoencoder.

## 5. Conclusion

In this paper we proposed an inverse scale space approach for training sparse neural networks based on linearized Bregman iterations. We introduced *LinBreg* as baseline
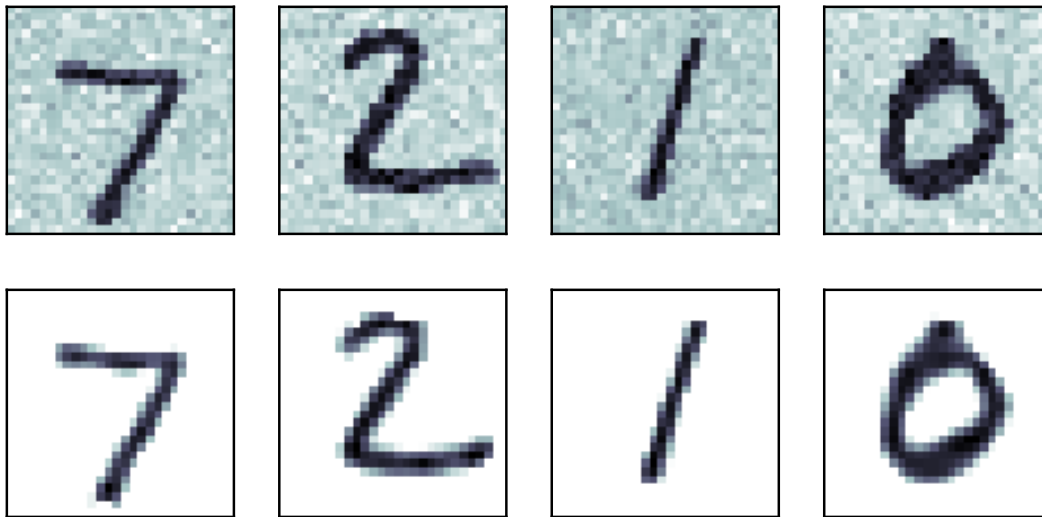
Figure 5: The denoising performance of the trained autoencoder on the test set with an average SSIM value of $\approx 0.93$.

algorithm for our learning framework and also discuss two variants using momentum and Adam. The effect of incorporating Bregman iterations into neural network training was investigated in numerical experiments on benchmark data sets. Our observations showed that the proposed method is able to train very sparse and accurate neural networks in an inverse scale space manner without using additional heuristics. Furthermore, we gave a glimpse of its applicability for discovering suitable network architectures for a given application task, e.g., an autoencoder architecture for image denoising. We mathematically supported our findings by performing a stochastic convergence analysis of the loss decay, and we proved convergence of the parameters in the case of convexity.

The proposed Bregman learning framework has a lot of potential for training sparse neural networks, and there are still a few open research questions (see also Hoefler et al. (2021)) which we would like to emphasize in the following.

First, we would like to use the inverse scale space character of the proposed Bregman learning algorithms in combination with sparse backpropagation for resource-friendly training, hence improving the carbon footprint of training (Anthony et al., 2020). This is a non-trivial endeavour for the following reason: A-priori it is not clear which weights are worth updating since estimating the magnitude of the gradient with respect to these weights already requires evaluating the backpropagation. A possible way to achieve this consists in performing a Bregman step to obtain a sparse support of the weights, performing several masked backpropagation steps to optimize the weights in these positions, and alternate this procedure.

Second, our experiment from Section 4.6, where our algorithm discovered a denoising autoencoder, suggests that our method has great potential for general ar-

chitecture design tasks. Using suitable sparsity regularization, e.g., on residual connections and rows of the weight matrices, one can investigate whether networks learn to form a U-net (Ronneberger et al., 2015) structure for the solution of inverse problems.

On the analysis side, it is worth investigating the convergence of LinBreg in the fully non-convex setting based on the Kurdyka–Łojasiewicz inequality and to extend these results to our accelerated algorithms LinBreg with momentum and AdaBreg. Furthermore, it will be interesting to remove the bounded variance condition from Assumption 2, which is known to be possible for stochastic gradient descent if the batch losses satisfy (3.3), see Lei et al. (2019). Finally, the characterizing the limit point of LinBreg for non-strongly convex losses as minimizer which minimizes the Bregman distance to the initialization will be worthwhile.

## Acknowledgments

## References

F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel. Artificial neural networks in medical diagnosis, 2013.

L. F. W. Anthony, B. Kanding, and R. Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

M. Bachmayr and M. Burger. Iterative total variation schemes for nonlinear inverse problems. *Inverse Problems*, 25(10):105004, 2009.

H. Bauschke and P. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. Springer, New York, 2011. ISBN 978-3-319-48311-5.

A. Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017. doi: 10.1137/1.9781611974997.

A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

A. Benfenati and V. Ruggiero. Inexact Bregman iteration with an application to poisson data reconstruction. *Inverse Problems*, 29(6):065016, 2013.

M. Benning and M. Burger. Modern regularization methods for inverse problems. *Acta Numerica*, 27:1–111, 2018.

M. Benning, M. M. Betcke, M. J. Ehrhardt, and C.-B. Schönlieb. Choose your path wisely: gradient descent in a Bregman distance framework. *SIAM Journal on Imaging Sciences*, 14(2):814–843, 2021.

M. Burger, G. Gilboa, S. Osher, J. Xu, et al. Nonlinear inverse scale space methods. *Communications in Mathematical Sciences*, 4(1):179–212, 2006.

M. Burger, K. Frick, S. Osher, and O. Scherzer. Inverse total variation flow. *Multiscale Modeling & Simulation*, 6(2):366–395, 2007.

M. Burger, M. Möller, M. Benning, and S. Osher. An adaptive inverse scale space method for compressed sensing. *Mathematics of Computation*, 82(281):269–299, 2013.

J.-F. Cai, S. Osher, and Z. Shen. Convergence of the linearized Bregman iteration for $\ell_1$-norm minimization. *Mathematics of Computation*, 78(268):2127–2136, 2009a.

J.-F. Cai, S. Osher, and Z. Shen. Linearized Bregman iterations for compressed sensing. *Mathematics of computation*, 78(267):1515–1536, 2009b.

G. Castellano, A. M. Fanelli, and M. Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531, 1997.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

X. Dai, H. Yin, and N. K. Jha. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

J. de Dios and J. Bruna. On sparsity in overparametrised shallow ReLU networks. *arXiv preprint arXiv:2006.10225*, 2020.

T. Dettmers and L. Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

P. Dhar. The carbon impact of artificial intelligence. *Nat Mach Intell*, 2:423–5, 2020.

R. D'Orazio, N. Loizou, I. Laradji, and I. Mitliagkas. Stochastic mirror descent: Convergence analysis and adaptive variants via the mirror stochastic Polyak stepsize. *arXiv preprint arXiv:2110.15412*, 2021.

R. A. Dragomir, M. Even, and H. Hendrikx. Fast stochastic Bregman gradient methods: Sharp analysis and variance reduction. In *International Conference on Machine Learning*, pages 2815–2825. PMLR, 2021.

U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2943–2952. PMLR, 2020.

J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

M. P. Friedlander and P. Tseng. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18(4):1326–1350, 2008.

Y. Fu, C. Liu, D. Li, Z. Zhong, X. Sun, J. Zeng, and Y. Yao. Exploring structural sparsity of deep networks via inverse scale spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021.

C. Huang, X. Sun, J. Xiong, and Y. Yao. Split LBI: An iterative regularization path with structural sparsity. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3377–3385, 2016.

T. Jia, Y. Shi, Y. Zhu, and L. Wang. An image restoration model combining mixed $L^1/L^2$ fidelity terms. *Journal of Visual Communication and Image Representation*, 38:461–473, 2016.

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

Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

Y. Lei, T. Hu, G. Li, and K. Tang. Stochastic gradient descent for nonconvex learning without bounded gradient assumptions. *IEEE transactions on neural networks and learning systems*, 31(10):4394–4400, 2019.

D. Li, X. Tian, Q. Jin, and K. Hirasawa. Adaptive fractional-order total variation image restoration with split Bregman iteration. *ISA transactions*, 82:210–222, 2018.

S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, and M. Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33(7):2589–2604, 2021.

C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through $l_0$ regularization. *arXiv preprint arXiv:1712.01312*, 2017.

Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.

J. Martens. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

A. S. Nemirovskij and D. B. Yudin. Problem complexity and method efficiency in optimization. 1983.

L. Nguyen, P. H. Nguyen, M. van Dijk, P. Richtarik, K. Scheinberg, and M. Takac. SGD and hogwild! Convergence without the bounded gradients assumption. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3750–3758. PMLR, 10–15 Jul 2018.

A. Nitanda. Stochastic proximal gradient descent with acceleration techniques. *Advances in Neural Information Processing Systems*, 27:1574–1582, 2014.

A. Orvieto, J. Kohler, and A. Lucchi. The role of memory in stochastic optimization. In *Uncertainty in Artificial Intelligence*, pages 356–366. PMLR, 2020.

S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin. An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation*, 4(2):460–489, 2005.

S. Osher, Y. Mao, B. Dong, and W. Yin. Fast linearized Bregman iteration for compressive sensing and sparse denoising. *Communications in Mathematical Sciences*, 8(1):93–111, 2010.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

S. J. Reddi, S. Sra, B. Poczos, and A. J. Smola. Proximal stochastic methods for nonsmooth nonconvex finite-sum optimization. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 1153–1161, 2016.

R. Rockafellar. *Convex analysis*. Princeton University Press, Princeton, N.J, 1997. ISBN 9781400873173.

O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

L. Rosasco, S. Villa, and B. C. Vũ. Convergence of stochastic proximal gradient algorithm. *Applied Mathematics & Optimization*, 82(3):891–917, 2020.

S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587): 484–489, 2016.

S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 138–145, 2017.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

G. Turinici. The convergence of the stochastic gradient descent (SGD): a self-contained proof. *arXiv preprint arXiv:2103.14350*, 2021.

Z. Wang, A. C. Bovik, H. R. Sheikh, S. Member, E. P. Simoncelli, and S. Member. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004.

Y. Yang, Y. Yuan, A. Chatzimichailidis, R. J. van Sloun, L. Lei, and S. Chatzinotas. ProxSGD: Training structured neural networks under regularization and constraints. In *International Conference on Learning Representations*, 2019.

W. Yin. Analysis and generalizations of the linearized Bregman method. *SIAM Journal on Imaging Sciences*, 3(4):856–877, 2010.

W. Yin, S. Osher, D. Goldfarb, and J. Darbon. Bregman iterative algorithms for $\ell_1$-minimization with applications to compressed sensing. *SIAM Journal on Imaging sciences*, 1(1):143–168, 2008.

J. Yun, A. C. Lozano, and E. Yang. A general family of stochastic proximal gradient methods for deep learning. *arXiv preprint arXiv:2007.07484*, 2020.

X. Zhang, M. Burger, and S. Osher. A unified primal-dual algorithm framework based on Bregman iteration. *Journal of Scientific Computing*, 46(1):20–46, 2011.

M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2): 301–320, 2005.

## Appendix

In all proofs we will use the abbreviation $g^{(k)} := g(\theta^{(k)}; \omega^{(k)})$ as in (3.1).

## A. Proofs from Section 3.1

**Proof** [Proof of Theorem 2] Using (3.3) one obtains

$$
\begin{aligned}
&\mathcal{L}(\theta^{(k+1)}) - \mathcal{L}(\theta^{(k)}) \\
&\leq \langle \nabla \mathcal{L}(\theta^{(k)}), \theta^{(k+1)} - \theta^{(k)} \rangle + \frac{L}{2} \|\theta^{(k+1)} - \theta^{(k)}\|^2 \\
&= \langle g^{(k)}, \theta^{(k+1)} - \theta^{(k)} \rangle + \langle \nabla \mathcal{L}(\theta^{(k)}) - g^{(k)}, \theta^{(k+1)} - \theta^{(k)} \rangle + \frac{L}{2} \|\theta^{(k+1)} - \theta^{(k)}\|^2 \\
&= -\frac{1}{\tau^{(k)}} \langle v^{(k+1)} - v^{(k)}, \theta^{(k+1)} - \theta^{(k)} \rangle \\
&\quad + \langle \nabla \mathcal{L}(\theta^{(k)}) - g^{(k)}, \theta^{(k+1)} - \theta^{(k)} \rangle + \frac{L}{2} \|\theta^{(k+1)} - \theta^{(k)}\|^2 \\
&= -\frac{1}{\tau^{(k)}} D_J^{\mathrm{sym}}(\theta^{(k+1)}, \theta^{(k)}) - \frac{1}{\delta \tau^{(k)}} \|\theta^{(k+1)} - \theta^{(k)}\|^2 \\
&\quad + \langle \nabla \mathcal{L}(\theta^{(k)}) - g^{(k)}, \theta^{(k+1)} - \theta^{(k)} \rangle + \frac{L}{2} \|\theta^{(k+1)} - \theta^{(k)}\|^2.
\end{aligned}
$$

Reordering and using the Cauchy–Schwarz inequality yields

$$
\begin{aligned}
\mathcal{L}(\theta^{(k+1)}) - \mathcal{L}(\theta^{(k)}) + \frac{1}{\tau^{(k)}} D_J^{\mathrm{sym}}(\theta^{(k+1)}, \theta^{(k)}) + \frac{2 - L\delta\tau^{(k)}}{2\delta\tau^{(k)}} \|\theta^{(k+1)} - \theta^{(k)}\|^2 &\leq \\
\|\nabla \mathcal{L}(\theta^{(k)}) - g^{(k)}\| \|\theta^{(k+1)} - \theta^{(k)}\|.
\end{aligned}
$$

Taking expectations and using Young's inequality gives for any $c > 0$

$$
\begin{aligned}
\mathbb{E}\left[\mathcal{L}(\theta^{(k+1)})\right] - \mathbb{E}\left[\mathcal{L}(\theta^{(k)})\right] + \frac{1}{\tau^{(k)}} \mathbb{E}\left[D_J^{\mathrm{sym}}(\theta^{(k+1)}, \theta^{(k)})\right] \\
+ \frac{2 - L\delta\tau^{(k)}}{2\delta\tau^{(k)}} \mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right] \leq \tau^{(k)} \delta \frac{\sigma^2}{2c} + \frac{c}{2\delta\tau^{(k)}} \mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right].
\end{aligned}
$$

If $c$ is sufficiently small and $\tau^{(k)} < \frac{2}{L\delta}$, we can absorb the last term into the left hand side and obtain

$$
\begin{aligned}
\mathbb{E}\left[\mathcal{L}(\theta^{(k+1)})\right] - \mathbb{E}\left[\mathcal{L}(\theta^{(k)})\right] + \frac{1}{\tau^{(k)}} \mathbb{E}\left[D_J^{\mathrm{sym}}(\theta^{(k+1)}, \theta^{(k)})\right] + \frac{C}{2\delta\tau^{(k)}} \mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right] \\
\leq \tau^{(k)} \delta \frac{\sigma^2}{2c},
\end{aligned}
$$

where $C > 0$ is a suitable constant. This shows (3.5). ■

**Proof** [Proof of Corollary 3] Using the assumptions on $\tau^{(k)}$, we can multiply (3.5) with $\tau^{(k)}$ and sum up the resulting inequality to obtain

$$\tau^{(K)}\mathbb{E}\left[\mathcal{L}(\theta^{(K)})\right] - \tau^{(0)}\mathbb{E}\left[\mathcal{L}(\theta^{(0)})\right] +$$
$$\sum_{k=0}^{K-1}\left(\mathbb{E}\left[D_J^{\mathrm{sym}}(\theta^{(k+1)}, \theta^{(k)})\right] + \frac{C}{2\delta}\mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right]\right) \leq \delta\frac{\sigma^2}{2c}\sum_{k=0}^{K-1}(\tau^{(k)})^2$$

Since $\mathcal{L} \geq 0$ we can drop the first term. Sending $K \to \infty$ and using that the step sizes are square-summable concludes the proof. ∎

## B. Proof from Section 3.2

The following lemma allows to split the Bregman distance with respect to the elastic net functional $J_\delta$ into two Bregman distances with respect to the functional $J$ and the Euclidean norm.

**Lemma 13** *For all $\tilde{\theta}, \theta \in \Theta$ and $v \in \partial J_\delta(\theta)$ it holds that*

$$D_{J_\delta}^v(\tilde{\theta}, \theta) = D_J^p(\tilde{\theta}, \theta) + \frac{1}{2\delta}\|\tilde{\theta} - \theta\|^2,$$

*where $p := v - \frac{1}{\delta}\theta \in \partial J(\theta)$.*

**Proof** Since $\partial J_\delta(\theta) = \partial J(\theta) + \partial \frac{1}{2\delta}\|\theta\|^2$, we can write $v = p + \frac{1}{\delta}\theta$ with $p \in \partial J(\theta)$. This readily yields

$$
\begin{aligned}
D_{J_\delta}^v(\tilde{\theta}, \theta) &= J_\delta(\tilde{\theta}) - J_\delta(\theta) - \langle v, \tilde{\theta} - \theta \rangle \\
&= J(\tilde{\theta}) + \frac{1}{2\delta}\|\tilde{\theta}\|^2 - J(\theta) - \frac{1}{2\delta}\|\theta\|^2 - \langle p, \tilde{\theta} - \theta \rangle - \frac{1}{\delta}\langle \theta, \tilde{\theta} - \theta \rangle \\
&= D_J^p(\tilde{\theta}, \theta) + \frac{1}{2\delta}\|\tilde{\theta}\|^2 - \frac{1}{2\delta}\|\theta\|^2 - \frac{1}{\delta}\langle \theta, \tilde{\theta} \rangle + \frac{1}{\delta}\|\theta\|^2 \\
&= D_J^p(\tilde{\theta}, \theta) + \frac{1}{2\delta}\|\tilde{\theta} - \theta\|^2.
\end{aligned}
$$

∎

The next lemma expresses the difference of two subsequent Bregman distances along the iteration in two different ways. The first one is useful for proving convergence of (3.1) under the weaker convexity Assumption 4, whereas the second one is used for proving the stronger convergence statement Theorem 11 under Assumption 5.

**Lemma 14** *Denoting* $d_k := \mathbb{E}\left[D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})\right]$ *the iteration* (3.1) *fulfills:*

$$d_{k+1} - d_k = -\mathbb{E}\left[D_{J_\delta}^{v^{(k)}}(\theta^{(k+1)}, \theta^{(k)})\right] + \tau^{(k)}\mathbb{E}\left[\langle g^{(k)}, \theta^* - \theta^{(k+1)}\rangle\right], \qquad (\text{B.1})$$

$$d_{k+1} - d_k = \mathbb{E}\left[D_{J_\delta}^{v^{(k+1)}}(\theta^{(k)}, \theta^{(k+1)})\right] + \tau^{(k)}\mathbb{E}\left[\langle \nabla\mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k)}\rangle\right]. \qquad (\text{B.2})$$

**Proof** We compute using the update in (3.1)

$$D_{J_\delta}^{v^{(k+1)}}(\theta^*, \theta^{(k+1)}) - D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})$$
$$= J_\delta(\theta^{(k)}) - J_\delta(\theta^{(k+1)}) - \langle v^{(k+1)}, \theta^* - \theta^{(k+1)}\rangle + \langle v^{(k)}, \theta^* - \theta^{(k)}\rangle$$
$$= -\left(J_\delta(\theta^{(k+1)}) - J_\delta(\theta^{(k)}) - \langle v^{(k)}, \theta^{(k+1)} - \theta^{(k)}\rangle\right) - \langle v^{(k)}, \theta^{(k+1)} - \theta^{(k)}\rangle$$
$$\qquad - \langle v^{(k+1)}, \theta^* - \theta^{(k+1)}\rangle + \langle v^{(k)}, \theta^* - \theta^{(k)}\rangle$$
$$= -D_{J_\delta}^{v^{(k)}}(\theta^{(k+1)}, \theta^{(k)}) + \langle v^{(k)} - v^{(k+1)}, \theta^* - \theta^{(k+1)}\rangle$$
$$= -D_{J_\delta}^{v^{(k)}}(\theta^{(k+1)}, \theta^{(k)}) + \tau^{(k)}\langle g^{(k)}, \theta^* - \theta^{(k+1)}\rangle.$$

For the second equation we similarly compute

$$D_{J_\delta}^{v^{(k+1)}}(\theta^*, \theta^{(k+1)}) - D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})$$
$$= J_\delta(\theta^{(k)}) - J_\delta(\theta^{(k+1)}) - \langle v^{(k+1)}, \theta^* - \theta^{(k+1)}\rangle + \langle v^{(k)}, \theta^* - \theta^{(k)}\rangle$$
$$= J_\delta(\theta^{(k)}) - J_\delta(\theta^{(k+1)}) - \langle v^{(k+1)}, \theta^* - \theta^{(k+1)}\rangle$$
$$\qquad + \langle v^{(k+1)}, \theta^* - \theta^{(k)}\rangle + \tau^{(k)}\langle g^{(k)}, \theta^* - \theta^{(k)}\rangle$$
$$= J_\delta(\theta^{(k)}) - J_\delta(\theta^{(k+1)}) - \langle v^{(k+1)}, \theta^{(k)} - \theta^{(k+1)}\rangle + \tau^{(k)}\langle g^{(k)}, \theta^* - \theta^{(k)}\rangle$$
$$= D_{J_\delta}^{v^{(k+1)}}(\theta^{(k)}, \theta^{(k+1)}) + \tau^{(k)}\langle g^{(k)}, \theta^* - \theta^{(k)}\rangle.$$

Taking expectations and using that $g^{(k)}$ and $\theta^* - \theta^{(k)}$ are stochastically independent to replace $g^{(k)}$ with $\nabla\mathcal{L}(\theta^{(k)})$ inside the expectation concludes the proof. Note that this argument does not apply to the first equality since $\theta^{(k+1)}$ is *not stochastically independent* of $\theta^{(k)}$. ∎

Now we prove Theorem 6 by showing that the Bregman distance to the minimizer of the loss and that the iterates converge in norm to the minimizer.
**Proof** [Proof of Theorem 6] Using (3.2), Assumption 4, and Young's inequality we obtain for any $c > 0$

$$\langle \nabla\mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k+1)}\rangle$$
$$= \langle \nabla\mathcal{L}(\theta^{(k+1)}), \theta^* - \theta^{(k+1)}\rangle + \langle \nabla\mathcal{L}(\theta^{(k)}) - \nabla\mathcal{L}(\theta^{(k+1)}), \theta^* - \theta^{(k+1)}\rangle$$
$$\leq \mathcal{L}(\theta^*) - \mathcal{L}(\theta^{(k+1)}) - \frac{\mu}{2}\|\theta^* - \theta^{(k+1)}\|^2 + \frac{L^2}{2c}\|\theta^{(k)} - \theta^{(k+1)}\|^2 + \frac{c}{2}\|\theta^* - \theta^{(k+1)}\|^2.$$

Using that $\mathcal{L}(\theta^*) \leq \mathcal{L}(\theta^{(k+1)})$ we obtain

$$\langle \nabla \mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k+1)} \rangle \leq \frac{L^2}{2c} \|\theta^{(k)} - \theta^{(k+1)}\|^2 + \frac{c - \mu}{2} \|\theta^* - \theta^{(k+1)}\|^2.$$

Plugging this into the expression (B.1) for $d_{k+1} - d_k$ yields

$$d_{k+1} - d_k = -\mathbb{E}\left[D_{J_\delta}^{v^{(k)}}(\theta^{(k+1)}, \theta^{(k)})\right] + \tau^{(k)}\mathbb{E}\left[\langle \nabla \mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k+1)} \rangle\right]$$
$$+ \tau^{(k)}\mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k+1)} \rangle\right]$$
$$\leq -\mathbb{E}\left[D_{J_\delta}^{v^{(k)}}(\theta^{(k+1)}, \theta^{(k)})\right] + \tau^{(k)}\frac{L^2}{2c}\mathbb{E}\left[\|\theta^{(k)} - \theta^{(k+1)}\|^2\right]$$
$$+ \frac{c - \mu}{2}\tau^{(k)}\mathbb{E}\left[\|\theta^* - \theta^{(k+1)}\|^2\right]$$
$$+ \tau^{(k)}\mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k+1)} \rangle\right].$$

Now we utilize that $\theta^* - \theta^{(k)}$ and $g^{(k)} - \nabla \mathcal{L}(\theta^{(k)})$ are stochastically independent and that the latter has zero expectation to infer

$$\mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k+1)} \rangle\right]$$
$$= \mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^{(k)} - \theta^{(k+1)} \rangle\right]$$
$$+ \mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^* - \theta^{(k)} \rangle\right]$$
$$= \mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^{(k)} - \theta^{(k+1)} \rangle\right].$$

Applying Young's inequality and using Assumption 2 yields

$$\tau^{(k)}\mathbb{E}\left[\langle g^{(k)} - \nabla \mathcal{L}(\theta^{(k)}), \theta^{(k)} - \theta^{(k+1)} \rangle\right]$$
$$\leq \frac{\delta(\tau^{(k)})^2}{2}\sigma + \frac{\sigma}{2\delta}\mathbb{E}\left[\|\theta^{(k)} - \theta^{(k+1)}\|^2\right].$$

Plugging this into the expression for $d_{k+1} - d_k$ and reordering we infer that for $\tau^{(k)} \leq \frac{\mu}{2\delta L^2}$ and $c = \mu/2$ it holds

$$d_{k+1} - d_k + \frac{\mu}{4}\tau^{(k)}\mathbb{E}\left[\|\theta^* - \theta^{(k+1)}\|^2\right]$$
$$\leq -\mathbb{E}\left[D_J^{p^{(k)}}(\theta^{(k+1)}, \theta^{(k)})\right] + \frac{\delta(\tau^{(k)})^2}{2}\sigma + \frac{\sigma}{2\delta}\mathbb{E}\left[\|\theta^{(k)} - \theta^{(k+1)}\|^2\right],$$

which yields (3.8). Summing this inequality, using Corollary 3—which is possible since $\tau^{(k)} \leq \frac{\mu}{2\delta L^2} \leq \frac{1}{2\delta L} \leq \frac{2}{\delta L}$—and that the step sizes are square-summable then shows that

$$\sum_{k=0}^{\infty} \tau^{(k)}\mathbb{E}\left[\|\theta^* - \theta^{(k+1)}\|^2\right] < \infty.$$

Since $\sum_{k=0}^{\infty} \tau^{(k)} = \infty$ this means that

$$\min_{k \in \{1, \ldots, K\}} \mathbb{E}\left[\|\theta^* - \theta^{(k)}\|^2\right] \to 0, \quad K \to \infty.$$

Hence, for an appropriate subsequence $\theta^{(k_j)}$ it holds

$$\lim_{j \to \infty} \mathbb{E}\left[\|\theta^* - \theta^{(k_j)}\|^2\right] = 0.$$

∎

**Proof** [Proof of Corollary 10] Since $J$ is equal to the $\ell_1$-norm, $J$ admits the triangle inequality $J(\theta^*) - J(\theta^{(k)}) \leq J(\theta^* - \theta^{(k)})$. Furthermore, since $d := \dim \Theta$ is finite, it admits the norm inequality $J(\theta) \leq \sqrt{d}\|\theta\|$ and has bounded subgradients $\|p\| = \|\operatorname{sign}(\theta)\| \leq d$ for all $\theta \in \Theta$.

Hence, using Lemma 13 we can estimate

$$\begin{aligned}
d_k &= \mathbb{E}\left[D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})\right] = \mathbb{E}\left[D_J^{p^{(k)}}(\theta^*, \theta^{(k)})\right] + \frac{1}{2\delta}\mathbb{E}\left[\|\theta^* - \theta^{(k)}\|^2\right] \\
&= \mathbb{E}\left[J(\theta^*) - J(\theta^{(k)}) - \langle p^{(k)}, \theta^* - \theta^{(k)}\rangle\right] + \frac{1}{2\delta}\mathbb{E}\left[\|\theta^* - \theta^{(k)}\|^2\right] \\
&\leq (\sqrt{d} + d)\mathbb{E}\left[\|\theta^* - \theta^{(k)}\|\right] + \frac{1}{2\delta}\mathbb{E}\left[\|\theta^* - \theta^{(k)}\|^2\right].
\end{aligned}$$

Hence, since $\mathbb{E}\left[\|\theta^* - \theta^{(k_j)}\|^2\right]$ converges to zero according to Theorem 6, the same is true for the sequence $d_{k_j}$. Furthermore, we have proved that $d_{k+1} - d_k \leq c_k$, where $c_k$ is a non-negative and summable sequence. This also implies $d_m \leq d_k + \sum_{j=k}^{\infty} c_k$ for every $m > k$.

Since $c_k$ is summable and $d_{k_j}$ converges to zero there exists $k \in \mathbb{N}$ and $l \in \mathbb{N}$ such that

$$\sum_{j=k}^{\infty} c_j < \frac{\varepsilon}{2}, \quad d_{k_l} < \frac{\varepsilon}{2}, \quad k_l > k.$$

Hence, we obtain for any $m > k_l$

$$d_m \leq d_{k_l} + \sum_{j=k_l}^{\infty} c_k \leq d_{k_l} + \sum_{j=k}^{\infty} c_k < \varepsilon.$$

Since $\varepsilon > 0$ was arbitrary, this implies that $d_m \to 0$ as $m \to \infty$. ∎

Finally we prove Theorem 11 based on Assumption 5, which asserts convergence in the Bregman distance.

**Proof** [Proof of Theorem 11] The proof goes along the lines of Turinici (2021), however Assumption 2 is weaker than the assumption posed there.

**Item 1:** Since proximal operators are 1-Lipschitz it holds

$$\|\theta^{(k+1)} - \theta^{(k)}\| = \|\operatorname{prox}_{\delta J}(\delta v^{(k+1)}) - \operatorname{prox}_{\delta J}(\delta v^{(k)})\| \leq \delta\|v^{(k+1)} - v^{(k)}\|.$$

Using the Cauchy–Schwarz inequality and this estimate yields

$$\begin{aligned}
\frac{1}{\tau^{(k)}}\mathbb{E}\left[D_{J_\delta}^{v^{(k+1)}}(\theta^{(k)}, \theta^{(k+1)})\right] &\leq \frac{1}{\tau^{(k)}}\mathbb{E}\left[D_{J_\delta}^{\mathrm{sym}}(\theta^{(k)}, \theta^{(k+1)})\right] \\
&= \frac{1}{\tau^{(k)}}\mathbb{E}\left[\langle v^{(k+1)} - v^{(k)}, \theta^{(k+1)} - \theta^{(k)}\rangle\right] \\
&= -\mathbb{E}\left[\langle g^{(k)}, \theta^{(k+1)} - \theta^{(k)}\rangle\right] \\
&\leq \sqrt{\mathbb{E}\left[\|g^{(k)}\|^2\right]}\sqrt{\mathbb{E}\left[\|\theta^{(k+1)} - \theta^{(k)}\|^2\right]} \\
&\leq \delta\sqrt{\mathbb{E}\left[\|g^{(k)}\|^2\right]}\sqrt{\mathbb{E}\left[\|v^{(k+1)} - v^{(k)}\|^2\right]} \\
&= \delta\tau^{(k)}\mathbb{E}\left[\|g^{(k)}\|^2\right].
\end{aligned}$$

It can be easily seen that Assumption 2 is equivalent to

$$\mathbb{E}\left[\|g^{(k)}\|^2\right] \leq \sigma^2 + \mathbb{E}\left[\|\nabla\mathcal{L}(\theta^{(k)})\|^2\right],$$

which together with the Lipschitz continuity of $\nabla\mathcal{L}$ from Assumption 1 implies

$$\begin{aligned}
\mathbb{E}\left[D_{J_\delta}^{v^{(k+1)}}(\theta^{(k)}, \theta^{(k+1)})\right] &\leq \delta(\tau^{(k)})^2\left(\sigma^2 + \mathbb{E}\left[\|\nabla\mathcal{L}(\theta^{(k)})\|^2\right]\right) \\
&\leq \delta(\tau^{(k)})^2\left(\sigma^2 + L^2\mathbb{E}\left[\|\theta^* - \theta^{(k)}\|^2\right]\right) \\
&\leq \delta(\tau^{(k)})^2\left(\sigma^2 + 2\delta L^2 d_k\right).
\end{aligned}$$

We plug this estimate into (B.2) and utilize Assumption 5 to obtain

$$\begin{aligned}
d_{k+1} - d_k &\leq \delta(\tau^{(k)})^2\left(\sigma^2 + 2\delta L^2 d_k\right) + \tau^{(k)}\mathbb{E}\left[\mathcal{L}(\theta^*) - \mathcal{L}(\theta^{(k)}) - \nu D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})\right] \\
&\leq \delta(\tau^{(k)})^2\left(\sigma^2 + 2\delta L^2 d_k\right) - \tau^{(k)}\nu\mathbb{E}\left[D_{J_\delta}^{v^{(k)}}(\theta^*, \theta^{(k)})\right] \\
&= \delta(\tau^{(k)})^2\left(\sigma^2 + 2\delta L^2 d_k\right) - \tau^{(k)}\nu d_k,
\end{aligned}$$

which is equivalent to (3.10).

**Item 2:** For $\tau^{(k)}$ sufficiently small (3.10) implies

$$d_{k+1} \leq \left(1 - \tau^{(k)}\tilde{\nu}\right)d_k + \delta(\tau^{(k)})^2\sigma^2$$

for a constant $\tilde{\nu} \in (0, \nu)$. For any $C \in \mathbb{R}$ we can reformulate this to

$$d_{k+1} - C \leq (1 - \tau^{(k)}\tilde{\nu})(d_k - C) + \tau^{(k)}\left(\tau^{(k)}\delta\sigma^2 - \tilde{\nu}C\right). \tag{B.3}$$

If $\tau^k = \tau$ for all $k \in \mathbb{N}$ is constant and we choose $C = \tau \frac{\delta \sigma^2}{\tilde{\nu}}$, we obtain

$$\left(d_{k+1} - \tau \frac{\delta \sigma^2}{\tilde{\nu}}\right)_+ \leq (1 - \tau \tilde{\nu})\left(d_k - \tau \frac{\delta \sigma^2}{\tilde{\nu}}\right)_+,$$

where we passed to the positive part $x_+ := \max(x, 0)$. Iterating this inequality yields

$$\left(d_{k+n} - \tau \frac{\delta \sigma^2}{\tilde{\nu}}\right)_+ \leq (1 - \tau \tilde{\nu})^n \left(d_k - \tau \frac{\delta \sigma^2}{\tilde{\nu}}\right)_+$$

and hence for $\tau < \frac{1}{\tilde{\nu}}$ we get

$$\limsup_{k \to \infty} d_k \leq \tau \frac{\delta \sigma^2}{\tilde{\nu}}.$$

Demanding $\tau < \frac{\varepsilon \tilde{\nu}}{\delta \sigma^2} \wedge \frac{1}{\tilde{\nu}}$ we finally obtain

$$\limsup_{k \to \infty} d_k \leq \varepsilon.$$

**Item 3:** We use the reformulation (B.3) with $C = \varepsilon > 0$. Since $\tau^{(k)}$ converges to zero the second term is non-positive for $k \in \mathbb{N}$ sufficiently large and we obtain

$$(d_{k+1} - \varepsilon)_+ \leq (1 - \tau^{(k)} \tilde{\nu})(d_k - \varepsilon)_+.$$

Iterating this inequality yields

$$(d_{k+n} - \varepsilon)_+ \leq \prod_{l=k}^{k+n-1} (1 - \tau^{(l)} \tilde{\nu})(d_k - \varepsilon)_+.$$

If $k \in \mathbb{N}$ is sufficiently large, then $\tau^{(l)} \tilde{\nu} < 1$ for all $l \geq k$ and the product satisfies

$$\prod_{l=k}^{k+n-1} (1 - \tau^{(l)} \tilde{\nu}) = \exp\left(\sum_{l=k}^{k+n-1} \log(1 - \tau^{(l)} \tilde{\nu})\right) \leq \exp\left(-\sum_{l=k}^{k+n-1} \tau^{(l)} \tilde{\nu}\right) \to 0, \quad n \to \infty,$$

where we used $\log(1-x) \leq -x$ for all $x < 1$ and $\sum_k \tau^{(k)} = \infty$. Since $\varepsilon$ was arbitrary we obtain the assertion. ∎