Lifting Layers: Analysis and Applications

Peter Ochs[1], Tim Meinhardt[2], Laura Leal-Taixe[2], and Michael Moeller[3]

[1] Saarland University, ochs@math.uni-sb.de
[2] Technical University of Munich, {tim.meinhardt,leal.taixe}@tum.de
[3] University of Siegen, michael.moeller@uni-siegen.de

Abstract. The great advances of learning-based approaches in image processing and computer vision are largely based on deeply nested networks that compose linear transfer functions with suitable non-linearities. Interestingly, the most frequently used non-linearities in imaging applications (variants of the rectified linear unit) are uncommon in low dimensional approximation problems. In this paper we propose a novel non-linear transfer function, called *lifting*, which is motivated from a related technique in convex optimization. A *lifting layer* increases the dimensionality of the input, naturally yields a linear spline when combined with a fully connected layer, and therefore closes the gap between low and high dimensional approximation problems. Moreover, applying the lifting operation to the loss layer of the network allows us to handle non-convex and flat (zero-gradient) cost functions. We analyze the proposed lifting theoretically, exemplify interesting properties in synthetic experiments and demonstrate its effectiveness in deep learning approaches to image classification and denoising.

Keywords: Machine Learning, Deep Learning, Interpolation, Approximation Theory, Convex Relaxation, Lifting

1 Introduction

Deep Learning has seen a tremendous success within the last 10 years improving the state-of-the-art in almost all computer vision and image processing tasks significantly. While one of the main explanations for this success is the replacement of handcrafted methods and features with data-driven approaches, the architectures of successful networks remain handcrafted and difficult to interpret.

The use of some common building blocks, such as convolutions, in imaging tasks is intuitive as they establish translational invariance. The composition of linear transfer functions with non-linearities is a natural way to achieve a simple but expressive representation, but the choice of non-linearity is less intuitive: Starting from biologically motivated step functions or their smooth approximations by sigmoids, researchers have turned to rectified linear units (ReLUs),

$$\sigma(x) = \max(x, 0) \tag{1}$$

to avoid the optimization-based problem of a vanishing gradient. The derivative of a ReLU is $\sigma'(x) = 1$ for all $x > 0$. Nonetheless, the derivative remains zero

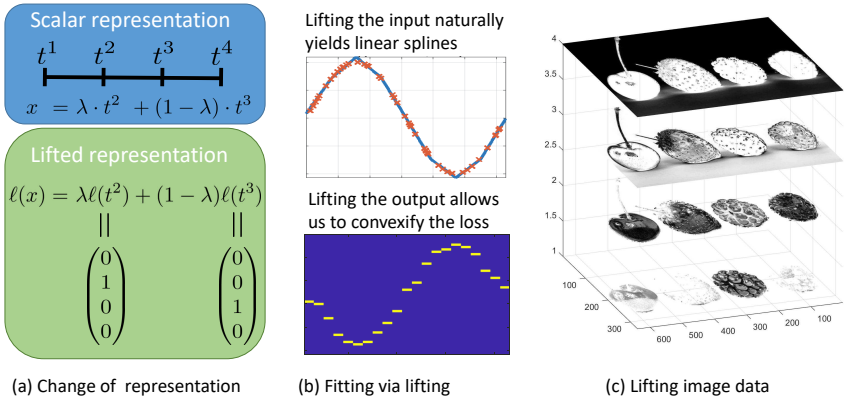


Fig. 1. The proposed lifting identifies predefined labels $t^i \in \mathbb{R}$ with the unit vectors e_i in $\mathbb{R}^L$, $L \geq 2$. As illustrated in (a), a number x that is represented as a convex combination of t^i and t^{i+1} has a natural representation in a higher dimensional *lifted* space, see (3). When a lifting layer is combined with a fully connected layer it corresponds to a linear spline, and when both the input as well as the desired output are *lifted* it allows non-convex cost functions to be represented as a convex minimization problem (b). Finally, as illustrated in (c), coordinate-wise lifting yields an interesting representation of images, which allows textures of different intensities to be filtered differently.

for $x < 0$, which does not seem to make it a natural choice for an activation function, and often leads to "dead" ReLUs. This problem has been partially addressed with ReLU variants, such as leaky ReLUs [1], parameterized ReLUs [2], or maxout units [3]. These remain amongst the most popular choice of non-linearities as they allow for fast network training in practice.

In this paper we propose a novel type of non-linear layer, which we call *lifting layer* ℓ. In contrast to ReLUs (1), it does not discard large parts of the input data, but rather *lifts* it to different channels that allow the input x to be processed independently on different intervals. As we discuss in more detail in Section 3.4, the simplest form of the proposed lifting non-linearity is the mapping

$$\sigma(x) = \begin{pmatrix} \max(x, 0) \\ \min(x, 0) \end{pmatrix}, \tag{2}$$

which essentially consists of two complementary ReLUs and therefore neither discards half of the incoming inputs nor has intervals of zero gradients.

More generally, the proposed non-linearity depends on *labels* $t^1 < \ldots < t^L \in \mathbb{R}$ (typically linearly spaced) and is defined as a function $\ell \colon [t^1, t^L] \to \mathbb{R}^L$ that maps a scalar input $x \in \mathbb{R}$ to a vector $\ell(x) \in \mathbb{R}^L$ via

$$\ell(x) = \Big(0, \ldots, 0, \underbrace{\frac{t^{l+1} - x}{t^{l+1} - t^l}}_{l\text{-th coordinate}}, \frac{x - t^l}{t^{l+1} - t^l}, 0, \ldots, 0\Big)^T \quad \text{for } x \in [t^l, t^{l+1}]. \tag{3}$$

The motivation of the proposed lifting non-linearity is illustrated in Figure 1. In particular, we highlight the following *contributions*:

1. The concept of representing a low dimensional variable in a higher dimensional space is a well-known optimization technique called *functional lifting*, see [4]. Non-convex problems are reformulated as the minimization of a convex energy in the higher dimensional 'lifted' space. While the **introduction of lifting layers** does not directly correspond to the optimization technique, some of the advantageous properties carry over as we detail in Section 3.
2. ReLUs are commonly used in deep learning for imaging applications, however their low dimensional relatives of interpolation or regression problems are typically tackled differently, e.g. by fitting (piecewise) polynomials. We show that a lifting layer followed by a fully connected layer **yields a linear spline**, which **closes the gap between low and high dimensional interpolation problems**. In particular, the aforementioned architecture can **approximate any continuous function** $f\colon \mathbb{R} \to \mathbb{R}$ to arbitrary precision and can still be trained **by solving a *convex* optimization problem** whenever the loss function is convex, a favorable property that is, for example, not shared even by the simplest ReLU-based architecture.
3. By additionally lifting the desired output of the network, one can **represent non-convex cost functions in a convex fashion**. Besides handling the non-convexity, such an approach allows for the minimization of cost functions with large areas of zero gradients such as truncated linear costs.
4. We demonstrate that the proposed lifting **improves the test accuracy in comparison to similar ReLU-based architectures in several experiments** on image classification and produces state-of-the-art image denoising results, making it an attractive universal tool in the design of neural networks.

2 Related Work

Lifting in Convex Optimization. One motivation for the proposed non-linearity comes from a technique called *functional lifting* which allows particular types of non-convex optimization problems to be reformulated as convex problems in a higher dimensional space, see [4] for details. The recent advances in functional lifting [5] have shown that (3) is a particularly well-suited discretization of the continuous model from [4]. Although, the techniques differ significantly, we hope for the general idea of an easier optimization in higher dimensions to carry over. Indeed, for simple instances of neural network architecture, we prove several favorable properties for our lifting layer that are related to properties of functional lifting. Details are provided in Sections 3 and 4.

Non-linearities in neural networks. While many non-linear transfer functions have been studied in the literature (see [6, Section 6.3] for an overview), the ReLU in (1) remains the most popular choice. Unfortunately, it has the drawback that its gradient is zero for all $x < 0$, thus preventing gradient based optimization

techniques to advance if the activation is zero (dead ReLU problem). Several variants of the ReLU avoid this problem by either utilizing smoother activations such as softplus [7] or exponential linear units [8], or by considering

$$\sigma(x;\alpha) = \max(x,0) + \alpha \min(x,0), \tag{4}$$

e.g. the absolute value rectification $\alpha = -1$ [9], leaky ReLUs with a small $\alpha > 0$ [1], randomized leaky ReLUs with randomly choosen α [10], parametric ReLUs in which α is a learnable parameter [2]. Self-normalizing neural networks [11] use scaled exponential LUs (SELUs) which have further normalizing properties and therefore replace the use of batch normalization techniques [12]. While the activation (4) seems closely related to the simplest case (2) of our lifting, the latter allows to process $\max(x,0)$ and $\min(x,0)$ separately, avoiding the problem of predefining α in (4) and leading to more freedom in the resulting function.

Another related non-linear transfer function are maxout units [3], which (in the 1-D case we are currently considering) are defined as

$$\sigma(x) = \max_j (\theta_j x + b_j). \tag{5}$$

They can represent any piecewise linear *convex* function. However, as we show in Proposition 1, a combination of the proposed lifting layer with a fully connected layer drops the restriction to *convex* activation functions, and allows us to learn *any* piecewise linear function.

Universal approximation theorem. As an extension of the universal approximation theorem in [13], it has been shown in [14] that the set of feedforward networks with one hidden layer, i.e., all functions $\mathcal{N}$ of the form

$$\mathcal{N}(x) = \sum_{j=1}^{N} \theta_j^1 \sigma(\langle \theta_j^2, x \rangle + b_j) \tag{6}$$

for some integer N, and weights $\theta_j^1 \in \mathbb{R}$, $\theta_j^2 \in \mathbb{R}^n$, $b_j \in \mathbb{R}$ are dense in the set of continuous functions $f\colon [0,1]^n \to \mathbb{R}$ if and only if σ is not a polynomial. While this result demonstrates the expressive power of all common activation functions, the approximation of some given function f with a network $\mathcal{N}$ of the form (6) requires optimization for the parameters θ^1 and (θ^2, b) which inevitably leads to a non-convex problem. We prove the same expressive power of a lifting based architecture (see Corollary 1), while, remarkably, our corresponding learning problem is a convex optimization problem. Moreover, beyond the qualitative density result for (6), we may quantify the approximation quality depending on a simple measure for the "complexity" of the continuous function to be approximated (see Corollary 1 and the supplementary material).

3 Lifting Layers

In this section, we introduce the proposed lifting layers (Section 3.1) and study their favorable properties in a simple 1-D setting (Section 3.2). The restriction

to 1-D functions is mainly for illustrative purposes and simplicity. All results can be transferred to higher dimensions via a vector valued lifting (Section 3.3). The analysis provided in this section does not directly apply to deep networks, however it provides an intuition for this setting. Section 3.4 discusses some practical aspects and reveals a connection to ReLUs. All proofs are provided in the supplementary material.

3.1 Definition

The following definition formalizes the lifting layer from the introduction.

Definition 1 (Lifting). *We define the lifting of a variable $x \in [\underline{t}, \overline{t}]$, $\underline{t}, \overline{t} \in \mathbb{R}$, with respect to the Euclidean basis $\mathcal{E} := \{e^1, \dots, e^L\}$ of $\mathbb{R}^L$ and a knot sequence $\underline{t} = t^1 < t^2 < \dots < t^L = \overline{t}$, for some $L \in \mathbb{N}$, as a mapping $\ell \colon [\underline{t}, \overline{t}] \to \mathbb{R}^L$ given by*

$$\ell(x) = (1 - \lambda_l(x))e^l + \lambda_l(x)e^{l+1} \quad \text{with } l \text{ such that } x \in [t^l, t^{l+1}]\,, \tag{7}$$

where $\lambda_l(x) := \frac{x - t^l}{t^{l+1} - t^l} \in \mathbb{R}$. The (left-)inverse mapping $\ell^\dagger \colon \mathbb{R}^L \to \mathbb{R}$ of ℓ, which satisfies $\ell^\dagger(\ell(x)) = x$, is defined by

$$\ell^\dagger(z) = \sum_{l=1}^{L} z_l t^l\,. \tag{8}$$

Note that while liftings could be defined with respect to an arbitrary basis $\mathcal{E}$ of $\mathbb{R}^L$ (with a slight modification of the inverse mapping), we decided to limit ourselves to the Euclidean basis for the sake of simplicity. Furthermore, we limit ourselves to inputs x that lie in the predefined interval $[\underline{t}, \overline{t}]$. Although, the idea extends to the entire real line by linear extrapolation, i.e., by allowing $\lambda_1(x) > 1$, $\lambda_2(x) < 0$, respectively, $\lambda_L(x) > 1$, $\lambda_{L-1}(x) < 0$, it requires more technical details. For the sake of a clean presentation, we omit these details.

3.2 Analysis in 1D

Although, here we are concerned with 1-D functions, these properties and examples provide some intuition for the implementation of the lifting layer into a deep architecture. Moreover, analogue results can be stated for the lifting of higher dimensional spaces.

Proposition 1 (Prediction of a Linear Spline). *The composition of a fully connected layer $z \mapsto \langle \theta, z \rangle$ with $\theta \in \mathbb{R}^L$, and a lifting layer, i.e.,*

$$\mathcal{N}_\theta(x) := \langle \theta, \ell(x) \rangle\,, \tag{9}$$

yields a linear spline (continuous piecewise linear function). Conversely, any linear spline can be expressed in the form of (9).

Although the architecture in (9) does not fall into the class of functions covered by the universal approximation theorem, well-known results of linear spline interpolation still guarantee the same results.

Corollary 1 (Prediction of Continuous Functions). *Any continuous function $f\colon [\underline{t}, \overline{t}] \to \mathbb{R}$ can be represented arbitrarily accurate with a network architecture $\mathcal{N}_\theta(x) := \langle \theta, \ell(x) \rangle$ for sufficiently large L, and $\theta \in \mathbb{R}^L$.*

Furthermore, as linear splines can of course fit any (spatially distinct) data points exactly, our simple network architecture has the same property for a particular choice of labels t^i. On the other hand, this result suggests that using a small number of labels acts as regularization of the type of linear interpolation.

Corollary 2 (Overfitting). *Let (x_i, y_i) be training data, $i = 1, \dots, N$ with $x_i \neq x_j$ for $i \neq j$. If $L = N$ and $t^i = x_i$, there exists θ such that $\mathcal{N}_\theta(x) := \langle \theta, \ell(x) \rangle$ is exact at all data points $x = x_i$, i.e. $\mathcal{N}_\theta(x_i) = y_i$ for all $i = 1, \dots, N$.*

Note that Proposition 1 highlights two crucial differences of the proposed non-linearity to the maxout function in (5): (i) maxout functions can only represent convex piecewise linear functions, while liftings can represent arbitrary piecewise linear functions; (ii) The maxout function is non-linear w.r.t. its parameters (θ_j, b_j), while the simple architecture in (9) (with lifting) is linear w.r.t. its parameters (θ, b). The advantage of a lifting layer compared to a ReLU, which is less expressive and also non-linear w.r.t. its parameters, is even more significant.

Remarkably, the optimal approximation of a continuous function by a linear spline (for any choice of t^i), yields a convex minimization problem.

Proposition 2 (Convexity of a simple Regression Problem). *Let training data $(x_i, y_i) \in [\underline{t}, \overline{t}] \times \mathbb{R}$, $i = 1, \dots, N$, be given. Then, the solution of the problem*

$$\min_{\theta} \sum_{i=1}^{N} \mathcal{L}(\langle \theta, \ell(x_i) \rangle \, ; y_i) \tag{10}$$

yields the best linear spline fit of the training data with respect to the loss function $\mathcal{L}$. In particular, if $\mathcal{L}$ is convex, then (10) is a convex optimization problem.

As the following example shows, this is not true for ReLUs and maxout functions.

Example 1. The convex loss $\mathcal{L}(z; 1) = (z-1)^2$ composed with a ReLU applied to a linear transfer function, i.e., $\theta \mapsto \max(\theta x_i, 0)$ with $\theta \in \mathbb{R}$, leads to a non-convex objective function, e.g. for $x_i = 1$, $\theta \mapsto (\max(\theta, 0) - 1)^2$ is non-convex.

Therefore, in the light of Proposition 2, the proposed lifting closes the gap between low dimensional approximation and regression problems (where linear splines are extremely common), and high dimensional approximation/learning problems, where ReLUs have been used instead of linear spline type of functions. In this one-dimensional setting, the proposed approach in fact represents a kernel method with a particular feature map ℓ from (1) that gives rise to linear splines. It is interesting to see that approximations by linear splines recently arose as an optimal architecture choice for second-order total variation minimization in [15].

3.3 Vector-Valued Lifting Layers

A vector-valued construction of the lifting similar to [16] allows us to naturally extend all our previous results for functions $f\colon [\underline{t}, \overline{t}] \to \mathbb{R}$ to functions $f\colon \Omega \subset \mathbb{R}^d \to \mathbb{R}$. Definition 1 is generalized to d dimensions by triangulating the compact domain Ω, and identifying each vertex of the resulting mesh with a unit vector in a space $\mathbb{R}^N$, where N is the total number of vertices. The lifted vector contains the barycentric coordinates of a point $x \in \mathbb{R}^d$ with respect its surrounding vertices. The resulting lifting remains a continuous piecewise linear function when combined with a fully connected layer (cf. Proposition 1), and yields a convex problem when looking for the best piecewise linear fit on a given triangular mesh (cf. Proposition 2).

Unfortunately, discretizing a domain $\Omega \subset \mathbb{R}^d$ with L labels per dimension leads to $N = L^d$ vertices, which makes a vector-valued lifting prohibitively expensive for large d. Therefore, in high dimensional applications, we turn to narrower and deeper network architectures, in which the scalar-valued lifting is applied to each component separately. The latter sacrifices the convexity of the overall problem for the sake of a high expressiveness with comparably few parameters. Intuitively, the increasing expressiveness is explained by an exponentially growing number of kinks for the composition of layers that represent linear splines. A similar reasoning can be found in [17].

3.4 Scaled Lifting

We are free to scale the lifted representation defined in (7), when the inversion formula in (8) compensates for this scaling. For practical purposes, we found it to be advantageous to also introduce a **scaled lifting** by replacing (7) in Definition 1 by

$$\ell_s(x) = (1 - \lambda_l(x))t^l e^l + \lambda_l(x) t^{l+1} e^{l+1} \quad \text{with } l \text{ such that } x \in [t^l, t^{l+1}], \tag{11}$$

where $\lambda_l(x) := \frac{x - t^l}{t^{l+1} - t^l} \in \mathbb{R}$. The inversion formula reduces to the sum over all components of the vector in this case. We believe that such a scaled lifting is often advantageous: (i) The magnitude/meaning of the components of the lifted vector is preserved and does not have to be learned; (ii) For an uneven number of equally distributed labels in $[-\overline{t}, \overline{t}]$, one of the labels t^l will be zero, which allows us to omit it and represent a scaled lifting into $\mathbb{R}^L$ with $L-1$ many entries. For $L = 3$ for example, we find that $t^1 = -\overline{t}$, $t^2 = 0$, and $t^3 = \overline{t}$ such that

$$\ell_s(x) = \begin{cases} \left(1 - \dfrac{x + \overline{t}}{0 + \overline{t}}\right)(-\overline{t})e^1 = xe^1 \text{ if } x \leq 0, \\ \dfrac{x - 0}{\overline{t} - 0} \quad \overline{t}\ e^3 = xe^3 \text{ if } x > 0. \end{cases} \tag{12}$$

As the second component remains zero, we can introduce an equivalent more memory efficient variant of the scaled lifting which we already stated in (2).

4 Lifting the Output

So far, we considered liftings as a non-linear layer in a neural network. However, motivated by lifting-based optimization techniques, which seek a tight convex approximation to problems involving non-convex loss functions, this section presents a convexification of non-convex loss functions by lifting in the context of neural networks. This goal is achieved by approximating the loss by a linear spline and predicting the output of the network in a lifted representation. The advantages of this approach are demonstrated at the end of this section in Example 2 for a robust regression problem with a vast number of outliers.

Consider a loss function $\mathcal{L}_y \colon \mathbb{R} \to \mathbb{R}$ defined for a given output y (the total loss for samples (x_i, y_i), $i = 1, \dots, N$, may be given by $\sum_{i=1}^N \mathcal{L}_{y_i}(x_i)$). We achieve the tight convex approximation by a lifting function $\ell_y \colon [\underline{t}_y, \overline{t}_y] \to \mathbb{R}^{L_y}$ for the range of the loss function $\operatorname{im}(\mathcal{L}_y) \subset \mathbb{R}$ with respect to the standard basis $\mathcal{E}_y = \{e_y^1, \dots, e_y^{L_y}\}$ and a knots $\underline{t}_y = t_y^1 < \ldots < t_y^{L_y} < \overline{t}_y$ following Definition 1.

The goal of the convex approximation is to predict the lifted representation of the loss, i.e. a vector $z \in \mathbb{R}^{L_y}$. However, in order to assign the correct loss to the lifted variable, it needs to lie in $\operatorname{im}(\ell_y)$. In this case, the following lemma proves a one-to-one representation of the loss between $[\underline{t}_y, \overline{t}_y]$ and $\operatorname{im}(\ell_y)$.

Lemma 1 (Characterization of the range of ℓ). *The range of the lifting $\ell \colon [\underline{t}, \overline{t}] \to \mathbb{R}^L$ is given by*

$$\operatorname{im}(\ell) = \left\{ z \in [0,1]^L \,:\, \exists! l \colon z_l + z_{l+1} = 1 \text{ and } \forall k \notin \{l, l+1\} \colon z_k = 0 \right\} \tag{13}$$

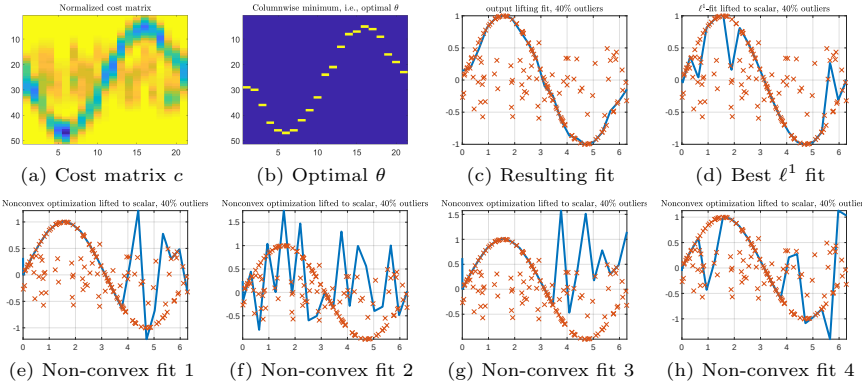
and the mapping ℓ is a bijection between $[\underline{t}, \overline{t}]$ and $\operatorname{im}(\ell)$ with inverse $\ell^\dagger$.

Since the range of ℓ_y is not convex, we relax it to a convex set, actually to the smallest convex set that contains $\operatorname{im}(\ell_y)$, the convex hull of $\operatorname{im}(\ell_y)$.

Lemma 2 (Convex Hull of the range of ℓ). *The convex hull* $\operatorname{conv}(\operatorname{im}(\ell))$ *of* $\operatorname{im}(\ell)$ *is the unit simplex in* $\mathbb{R}^L$.

Putting the results together, using Proposition 1, we obtain a tight convex approximation of the (possibly non-convex) loss $\mathcal{L}_y(x)$ by $\langle l_y, z \rangle$ with $z \in \operatorname{im}(\ell_y)$, for some $l_y \in \mathbb{R}^{L_y}$. Instead of evaluating the network $\mathcal{N}_\theta(x)$ by $\mathcal{L}_y(\mathcal{N}_\theta(x))$, we consider a network $\widetilde{\mathcal{N}}_\theta(x)$ that predicts a point in $\operatorname{conv}(\operatorname{im}(\ell_y)) \subset \mathbb{R}^{L_y}$ and evaluate the loss $\langle l_y, \widetilde{\mathcal{N}}_\theta(x) \rangle$. As it is hard to incorporate range-constraints into the network's prediction, we compose the network with a lifting layer ℓ_x, i.e. we consider $\langle l_y, \tilde{\theta} \ell_x(\widetilde{\mathcal{N}}_\theta(x)) \rangle$ with $\tilde{\theta} \in \mathbb{R}^{L_y \times L_x}$, for which simpler constraints may be derived. The following proposition states the convexity of the relaxed problem.

Proposition 3 (Convex Approximation of a simple non-convex regression problem). *Let $(x_i, y_i) \in [\underline{t}, \overline{t}] \times [\underline{t}_y, \overline{t}_y]$ be training data, $i = 1, \dots, N$. Moreover, let ℓ_y be a lifting of the common image $[\underline{t}_y, \overline{t}_y]$ of the loss $\mathcal{L}_{y_i}$, $i = 1, \dots, N$,*



(a) Cost matrix c (b) Optimal θ (c) Resulting fit (d) Best ℓ^1 fit

(e) Non-convex fit 1 (f) Non-convex fit 2 (g) Non-convex fit 3 (h) Non-convex fit 4

Fig. 2. Visualization of Example 2 for a regression problem with 40% outliers. Our lifting of a (non-convex) truncated linear loss to a convex optimization problem robustly fits the function nearly optimally (see (c)), whereas the most robust convex formulation (without lifting) is severely perturbed by the outliers (see (d)). Trying to optimize the non-convex cost function directly yields different results based on the initialization of the weights and is prone to getting stuck in suboptimal local minima, see (e)-(h).

and ℓ_x is the lifting of $[\underline{t}, \overline{t}]$. Let $l_{y_i} \in \mathbb{R}^{L_y}$ be such that $t_y \mapsto \langle l_{y_i}, \ell_y(t_y)\rangle$ is a piecewise linear spline approximation of $t_y \mapsto \mathcal{L}_{y_i}(t_y)$, for $t_y \in [\underline{t}_y, \overline{t}_y]$. Then

$$\min_{\theta} \sum_{i=1}^{N} \langle l_{y_i}, \theta \ell_x(x_i)\rangle \quad s.t.\ \theta_{p,q} \geq 0, \sum_{p=1}^{L_y} \theta_{p,q} = 1, \begin{cases} \forall p = 1, \dots, L_y\,, \\ \forall q = 1, \dots, L_x\,. \end{cases} \tag{14}$$

is a convex approximation of the (non-convex) loss function, and the constraints guarantee that $\theta \ell_x(x_i) \in \operatorname{conv}(\operatorname{im}(\ell_y))$.

The objective in (14) is linear (w.r.t. θ) and can be written as

$$\sum_{i=1}^{N} \langle l_{y_i}, \theta \ell_x(x_i)\rangle = \sum_{i=1}^{N} \sum_{p=1}^{L_y} \sum_{q=1}^{L_x} \theta_{p,q} \ell_x(x_i)_q (l_{y_i})_p =: \sum_{p=1}^{L_y} \sum_{q=1}^{L_x} c_{p,q} \theta_{p,q} \tag{15}$$

where $c := \sum_{i=1}^{N} l_{y_i} \ell_x(x_i)^\top$.

Moreover, the closed-form solution of (14) is given for all $q = 1, \dots, L_x$ by $\theta_{p,q} = 1$, if the index p minimizes $c_{p,q}$, and $\theta_{p,q} = 0$ otherwise.

Example 2 (Robust fitting). For illustrative purposes of the advantages of this section, we consider a regression problem with 40% outliers as visualized in Figure 2(c) and (d). Statistics motivates us to use a robust non-convex loss function. Our lifting allows us to use a robust (non-convex) truncated linear loss in a convex optimization problem (Proposition 3), which can easily ignore the outliers and achieve a nearly optimal fit (see Figure 2(c)), whereas the most robust convex loss (without lifting), the ℓ_1-loss, yields a solution that is severely perturbed by the outliers (see Figure 2(d)). The cost matrix c from (15) that

represents the non-convex loss (of this example) is shown in Figure 2(a) and the computed optimal θ is visualized in Figure 2(b). For comparison purposes we also show the results of a direct (gradient descent + momentum) optimization of the truncated linear costs with four different initial weights chosen from a zero mean Gaussian distribution. As we can see the results greatly differ for different initializations and always got stuck in suboptimal local minima.

5 Numerical Experiments

In this section we provide synthetic numerical experiments to illustrate the behavior of lifting layers on simple examples, before moving to real-world imaging applications. We implemented lifting layers in MATLAB as well as in PyTorch with https://github.com/michimoeller/liftingLayers containing all code for reproducing the experiments in this paper. A description of the presented network architectures is provided in the supplementary material.

5.1 Synthetic Examples

The following results were obtained using a stochastic gradient descent (SGD) algorithm with a momentum of 0.9, using minibatches of size 128, and a learning rate of 0.1. Furthermore, we use weight decay with a parameter of 10^{-4}. A squared ℓ^2-loss (without lifting the output) was used.

1-D Fitting To illustrate our results of Proposition 2, we first consider the example of fitting values $y_i = \sin(x_i)$ from input data x_i sampled uniformly in $[0, 2\pi]$. We compare the lifting-based architecture $\mathcal{N}_\theta(x) = \langle \theta, \ell_9(x) \rangle$ (Lift-Net) including an unscaled lifting ℓ_9 to $\mathbb{R}^9$ with the standard design architecture $\mathrm{fc}_1(\sigma(\mathrm{fc}_9(x)))$ (Std-Net), where $\sigma(x) = \max(x, 0)$ applies coordinate-wise and fc_n denotes a fully connected layer with n output neurons. Figure 3 shows the resulting functions after 25, 75, 200, and 2000 epochs of training.

2-D Fitting While the above results were expected based on the favorable theoretical properties, we now consider a more difficult test case of fitting

$$f(x_1, x_2) = \cos(x_2 \, \sin(x_1)) \tag{16}$$

on $[0, 2\pi]^2$. Note that although a 2-D input still allows for a vector-valued lifting, our goal is to illustrate that even a coordinate-wise lifting has favorable properties (beyond being able to approximate any separable function with a single layer which is a simple extension of Corollary 1). Hence, we compare two networks

$$f_{\text{Lift-Net}}(x_1, x_2) = \mathrm{fc}_1(\sigma(\mathrm{fc}_{20}([\ell_{20}(x_1), \ell_{20}(x_2)]))), \qquad \text{(Lift-Net)}$$
$$f_{\text{Std-Net}}(x_1, x_2) = \mathrm{fc}_1(\sigma(\mathrm{fc}_{20}(\mathrm{fc}_{40}([x_1, x_2])))), \qquad \text{(Std-Net)}$$

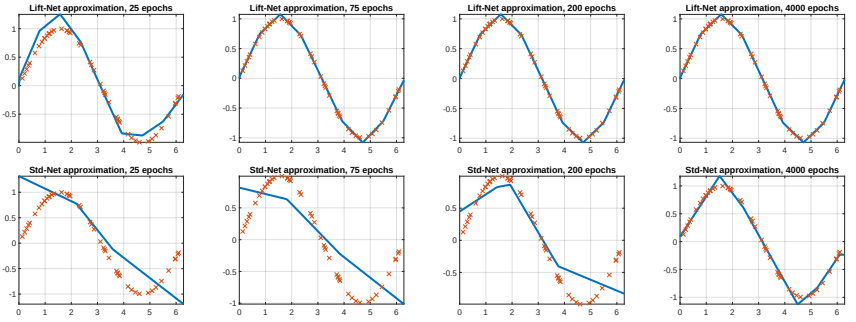


Fig. 3. Illustrating the results of approximating a sine function on $[0, 2\pi]$ with 50 training examples after different number of epochs. While the proposed architecture with lifting yields a convex problem for which SGD converges quickly (upper row), the standard architecture based on ReLUs yields a non-convex problem which leads to slower convergence and a suboptimal local minimum after 4000 epochs (lower row).

where the notation $[u, v]$ in the above formula denotes the concatenation of the two vectors u and v, and the subscript of the lifting ℓ denotes the dimension L we lift to. The corresponding training problems are non-convex. As we see in Figure 4 the general behavior is similar to the 1-D case: Increasing the dimensionality via lifting the input data yields faster convergence and a more precise approximation than increasing the dimensionality with a parameterized filtering. For the sake of completeness, we have included a vector valued lifting with an illustration of the underlying 2-D triangulation in the bottom row of Figure 4.

5.2 Image Classification

As a real-world imaging example we consider the problem of image classification. To illustrate the behavior of our lifting layer, we use the "Deep MNIST for expert model" (*ME-model*) by TensorFlow[4] as a simple convolutional neural network (CNN) architecture which applies a standard ReLU activation. To improve its accuracy, we use an additional batch-normalization (BN) layer and denote the corresponding model by *ME-model+BN*.

Our corresponding lifting model (*Proposed*) is created by replacing all ReLUs with a scaled lifting layer (as introduced in Section 3.4) with $L = 3$. In order to allow for a meaningful combination with the max pooling layers, we scaled with the absolute value $|t^i|$ of the labels. We found the comparably small lifting of $L = 3$ to yield the best results, and provided a more detailed study for varying L in the supplementary material. Since our model has almost twice as many free parameters as the two *ME* models, we include a forth model *Large ME-model+BN* larger than our lifting model with twice as many convolution filters and fully-connected neurons.

[4] https://www.tensorflow.org/tutorials/layers

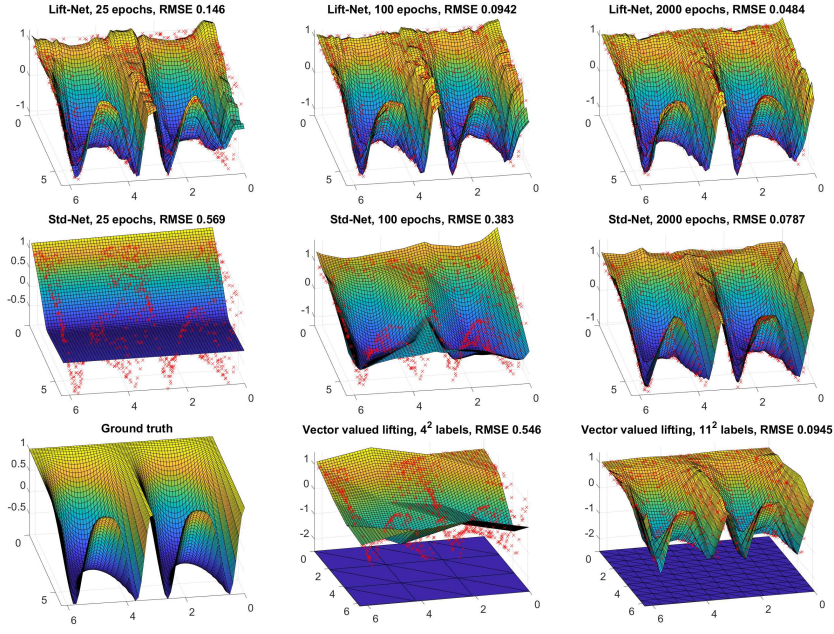
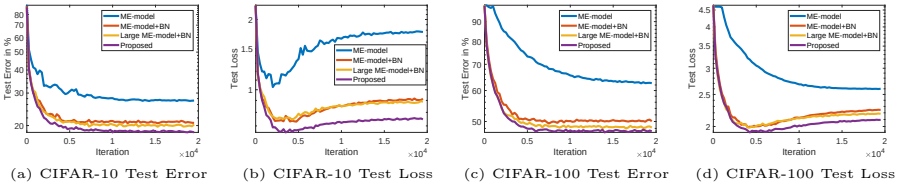


Fig. 4. Illustrating the results of approximating the function in (16) with the standard network in (Std-Net) (middle row) and the architecture in (Lift-Net) based on lifting the input data (upper row). The red markers illustrate the training data, the surface represents the overall network function, and the RMSE measures its difference to the true underlying function (16), which is shown in the bottom row on the left. Similar to the results of Figure 3, our lifting based architecture converges more quickly and yields a better approximation of the true underlying function (lower left) after 2000 epochs. The middle and right approximations in the bottom row illustrate a vector valued lifting (see Section 3.3) into 4^2 (middle) and 11^2 (right) dimensions. The latter can be trained by solving a linear system. We illustrate the triangular mesh used for the lifting below the graph of the function to illustrate that the approximation is indeed piecewise linear (as stated in Proposition 1).

Figure 5 shows the results each of these models obtains on the CIFAR-10 and CIFAR-100 image classification problems. As we can see, the favorable behavior of the synthetic experiments carried over to the exemplary architectures in image classification: Our proposed lifting architecture has the smallest test error and loss in both experiments. Both common strategies, i.e. including batch normalization and increasing the size of the model, improved the results, but both ReLU-based architectures remain inferior to the lifting-based architecture.

5.3 Maxout Activation Units

To also compare the proposed lifting activation layer with the maxout activation, we conduct a simple MNIST image classification experiment with a fully connected one-hidden-layer architecture, applying either a ReLU, maxout or lifting



(a) CIFAR-10 Test Error (b) CIFAR-10 Test Loss (c) CIFAR-100 Test Error (d) CIFAR-100 Test Loss

Fig. 5. Comparing different approaches for image classification on CIFAR-10 and CIFAR-100. The proposed architecture with lifting layers shows a superior performance in comparison to its ReLU-based relatives in both cases.

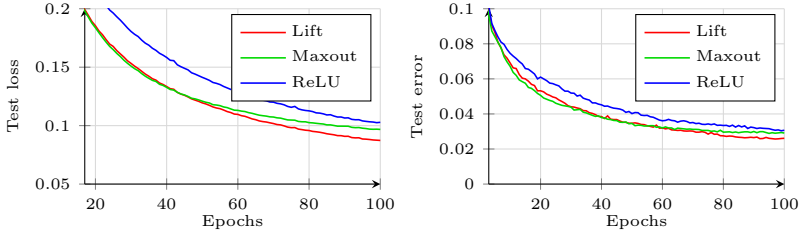


Fig. 6. MNIST image classification comparison of our lifting activation with the standard ReLU and its maxout generalization. The ReLU, maxout and lifting architectures (79510, 79010 and 76485 trainable parameters) achieved a best test error of 3.07%, 2.91% and 2.61%, respectively. The proposed approach behaves favorably in terms of the test loss from epoch 50 on, leading to a lower overall test error after 100 epochs.

as activations. For the maxout layer we apply a feature reduction by a factor of 2 which has the capabilities of representing a regular ReLU and a lifting layer as in (2). Due to the nature of the different activations - maxout applies a max pooling and lifting increases the number of input neurons in the subsequent layer - we adjusted the number of neurons in the hidden layer to make for an approximately equal and fair amount of trainable parameters.

The results in Figure 6 are achieved after optimizing a cross-entropy loss for 100 training epochs by applying SGD with learning rate 0.01. Particularly, each architecture was trained with the identical experimental setup. While both the maxout and our lifting activation yield a similar convergence behavior better than the standard ReLU, our proposed method exceeds in terms of the final lowest test error.

5.4 Image Denoising

To also illustrate the effectiveness of lifting layers for networks mapping images to images, we consider the problem of Gaussian image denoising. We designed the *Lift-46* architecture with 16 blocks each of which consists of 46 convolution filters of size 3×3, batch normalization, and a lifting layer with $L = 3$ following the same experimental reasoning for deep architectures as in Section 5.2. As illustrated in Figure 7(a), a final convolutional layer outputs an image we train

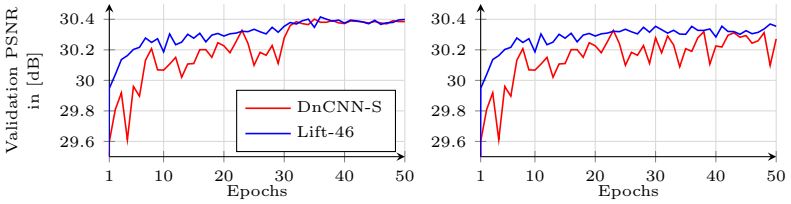


Fig. 7. To demonstrate the robustness of our lifting activation, we illustrate the validation PSNR for denoising Gaussian noise with $\sigma = 25$ for two different training schemes. In (a) both networks plateau - after a learning rate decay at 30 epochs - to the same final PSNR value. However, without this specifically tailored training scheme our method generally shows a favorable and more stable behavior, as seen in (b).

Table 1. Average PSNRs in [dB] for the BSD68 dataset for different standard deviations σ of the Gaussian noise on all of which our lifting layer based architecture is among the leading methods. Please note that (most likely due to variations in the random seeds) our reproduced DnCNN-S results are different - in the second decimal place - from the results reported in [18].

Reconstruction PSNR in [*dB*]									
σ	Noisy	BM3D [19]	WNNM [20]	EPLL [21]	MLP [22]	CSF [23]	TNRD [24]	DnCNN-S [18]	Our
15	24.80	31.07	31.37	31.21	-	31.24	31.42	**31.72**	**31.72**
25	20.48	28.57	28.83	28.68	28.96	28.74	28.92	**29.21**	**29.21**
50	14.91	25.62	25.87	25.67	26.03	-	25.97	26.21	**26.23**

to approximate the residual, i.e., noise-only, image. Due to its state-of-the-art performance in image denoising we adopted the same training pipeline as for the *DnCNN-S* architecture from [18] which resembles our Lift-46 network but implements a regular ReLU and 64 convolution filters. The two architectures contain an approximately equal amount of trainable parameters.

Table 1 compares our architecture with a variety of denoising methods most notably the DnCNN-S [18] and shows that we produce state-of-the-art performance for removing Gaussian noise of different standard deviations σ. In addition, the development of the test PSNR in Figure 7(b) suggests a more stable and favorable behavior of our method compared to DnCNN-S.

6 Conclusions

We introduced lifting layers as a favorable alternative to ReLU-type activation functions in machine learning. Opposed to the classical ReLU, liftings have nonzero derivative almost everywhere, and can represent any continuous piecewise linear function. We demonstrated several advantageous properties of lifting, for example, we can handle non-convex and partly flat loss functions. Our numerical experiments in image classification and reconstruction showed that lifting layers are an attractive building block in various neural network architectures, and we improved the performance of corresponding ReLU-based architectures.

Acknowledgements. This research was partially funded by the Humboldt Foundation through the Sofja Kovalevskaja Award.

References

1. Maas, A., Hannun, A., Ng, A.: Rectifier nonlinearities improve neural network acoustic models. In: ICML. (2013) 2, 4
2. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV. (2015) 2, 4
3. Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: ICML. (2013) 2, 4
4. Pock, T., Cremers, D., Bischof, H., Chambolle, A.: Global solutions of variational models with convex regularization. SIAM Journal on Imaging Sciences **3**(4) (2010) 1122–1145 3
5. Möllenhoff, T., Laude, E., Moeller, M., Lellmann, J., Cremers, D.: Sublabel-accurate relaxation of nonconvex energies. In: CVPR. (2016) 3
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT Press (2016) 3
7. Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., Garcia, R.: Incorporating second-order functional knowledge for better option pricing. In: NIPS. (2001) 4
8. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). In: ICLR. (2016) 4
9. Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y.: What is the best multi-stage architecture for object recognition? In: CVPR. (2009) 4
10. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network (2015) ICML Deep Learning Workshop, online at https://arxiv.org/abs/1505.00853. 4
11. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. NIPS (2017) 4
12. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML (2015) 4
13. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems **2**(4) (1989) 303–314 4
14. Leshno, M., Lin, V., Pinkus, A., Schocken, S.: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks **6**(6) (1993) 861 – 867 4
15. Unser, M.: A representer theorem for deep neural networks (2018) Preprint at https://arxiv.org/abs/1802.09210. 6
16. Laude, E., Möllenhoff, T., Moeller, M., Lellmann, J., Cremers, D.: Sublabel-accurate convex relaxation of vectorial multilabel energies. In: ECCV. (2016) 7
17. Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: NIPS. (2014) 7
18. Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L.: Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. IEEE Transactions on Image Processing **26**(7) (2017) 3142–3155 14
19. Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K.: Image denoising by sparse 3-d transform-domain collaborative filtering. IEEE Transactions on Image Processing **16**(8) (2007) 2080–2095 14
20. Gu, S., Zhang, L., Zuo, W., Feng, X.: Weighted nuclear norm minimization with application to image denoising. In: CVPR. (2014) 14
21. Zoran, D., Weiss, Y.: From learning models of natural image patches to whole image restoration. In: ICCV. (2011) 14
22. Burger, H., Schuler, C., Harmeling, S.: Image denoising: Can plain neural networks compete with BM3D? In: CVPR. (2012) 14

23. Schmidt, U., Roth, S.: Shrinkage fields for effective image restoration. In: CVPR. (2014) 14
24. Chen, Y., Pock, T.: On learning optimized reaction diffusion processes for effective image restoration. In: ICCV. (2015) 14