Machine Learning (2021) 110:393–416
https://doi.org/10.1007/s10994-020-05929-w

Regularisation of neural networks by enforcing Lipschitz continuity

Henry Gouk[1] · Eibe Frank[2] · Bernhard Pfahringer[2] · Michael J. Cree[2]

Received: 20 December 2019 / Revised: 12 October 2020 / Accepted: 25 October 2020 /
Published online: 6 December 2020

Abstract
We investigate the effect of explicitly enforcing the Lipschitz continuity of neural networks with respect to their inputs. To this end, we provide a simple technique for computing an upper bound to the Lipschitz constant—for multiple *p*-norms—of a feed forward neural network composed of commonly used layer types. Our technique is then used to formulate training a neural network with a bounded Lipschitz constant as a constrained optimisation problem that can be solved using projected stochastic gradient methods. Our evaluation study shows that the performance of the resulting models exceeds that of models trained with other common regularisers. We also provide evidence that the hyperparameters are intuitive to tune, demonstrate how the choice of norm for computing the Lipschitz constant impacts the resulting model, and show that the performance gains provided by our method are particularly noticeable when only a small amount of training data is available.

Keywords Neural networks · Regularisation · Lipschitz continuity

1 Introduction

Supervised learning is primarily concerned with the problem of approximating a function given examples of what output should be produced for a particular input. For the approximation to be of any practical use, it must generalise to unseen data points. Thus,

Editor: Paolo Frasconi.

✉ Henry Gouk
henry.gouk@ed.ac.uk

Eibe Frank
eibe.frank@waikato.ac.nz

Bernhard Pfahringer
bernhard.pfahringer@waikato.ac.nz

Michael J. Cree
michael.cree@waikato.ac.nz

[1] University of Edinburgh, Edinburgh, Scotland

[2] University of Waikato, Hamilton, New Zealand

we need to select an appropriate space of functions in which the machine should search for a good approximation, and select an algorithm to search through this space. This is typically done by first picking a large family of models, such as support vector machines or decision trees, and applying a suitable search algorithm. Crucially, when performing the search, regularisation techniques specific to the chosen model family must be employed to combat overfitting. For example, one could limit the depth of decision trees considered by a learning algorithm, or impose probabilistic priors on tunable model parameters.

Regularisation of neural network models is a particularly difficult challenge. The methods that are currently most effective (Srivastava et al. 2014; Ioffe and Szegedy 2015) are heuristically motivated. In contrast, well-understood regularisation approaches adapted from linear models, such as applying an ℓ^2 penalty term to the model parameters, are known to be less effective than the heuristic approaches (Srivastava et al. 2014). This provides a clear motivation for developing well-founded and effective regularisation methods for neural networks. Following the intuition that functions are considered simpler when they vary at a slower rate, and thus generalise better, we develop a method that allows us to control the Lipschitz constant of a network—a measure of the maximum variation a function can exhibit. Our experiments show that this is a useful inductive bias to impose on neural network models.

One of the prevailing themes in the theoretical work surrounding neural networks is that the magnitude of the weights directly impacts the generalisation gap (Bartlett 1998; Bartlett et al. 2017; Neyshabur 2017; Golowich et al. 2020), with larger weights being associated with poorer relative performance on new data. In several of the most recent works (Bartlett et al. 2017; Neyshabur 2017; Golowich et al. 2020), some of the dominant terms in these bounds are equal to the upper bound of the Lipschitz constant of neural networks as we derive it in this paper. While previous works have only considered the Lipschitz continuity of networks with respect to the ℓ^2 norm, we put a particular emphasis on working with ℓ^1 and ℓ^∞ norms and construct a practical algorithm for constraining the Lipschitz constant of a network during training. The algorithm takes a hyperparameter for each layer that specifies its maximum allowable Lipschitz constant, and these parameters together determine an upper bound on the allowable Lipschitz constant of the entire network. We reuse the same parameter value across multiple layers in our experiments to accelerate the hyperparameter optimisation process.

Several interesting properties of this regularisation technique are demonstrated experimentally. We show that although our algorithm is not competitive when used in isolation, it is highly effective when combined with other commonly used regularisers. Moreover, gains over conventional regularisation approaches are relatively more pronounced when only a small amount of training data is available. We verify that the hyperparameters behave in an intuitive manner: when set to small values, the model capacity is reduced, and as the values of the hyperparameters are increased, the model capacity also increases. Crucially, there is a range of hyperparameter settings where the performance is greater than that of a model trained without our regulariser.

The paper begins with an outline of previous work related to regularisation and the Lipschitz continuity of neural networks in Sect. 2. This is followed by a detailed derivation of the upper bound on the Lipschitz constant of a wide class of feed forward neural networks in Sect. 3, where we give consideration to multiple choices of vector norms. Section 4 shows how this upper bound can be used to regularise the neural network in an efficient manner. Experiments showing the utility of this regularisation approach are given in Sect. 5, and conclusions are drawn in Sect. 6.

2 Related work

One of the most widely applied regularisation techniques currently used for deep networks is dropout (Srivastava et al. 2014). By randomly setting the activations of each hidden unit to zero with some probability, p, during training, this method noticeably reduces overfitting for a wide variety of models. Various extensions have been proposed, such as randomly setting weights to zero instead of activations (Wan et al. 2013). Another modification, concrete dropout (Gal et al. 2017), allows one to directly learn the dropout rate, p, thus making the search for a good set of hyperparameters easier. Kingma et al. (2015) have also shown that the noise level in Gaussian dropout can be learned during optimisation. Srivastava et al. (2014) found that constraining the ℓ^2 norm of the weight vector for each unit in isolation—a technique that they refer to as maxnorm—can improve the performance of networks trained with dropout.

The recent work on optimisation for deep learning has also contributed to our understanding of the generalisation performance of neural networks. Most work in this area aims to be descriptive, rather than prescriptive, in the sense that the focus is on providing explanations for existing heuristic methods as opposed to developing new approaches to improving performance. For example, Hardt et al. (2016) quantify the relationship between generalisation error and early stopping. Several papers have shown that the generalisation gap of a neural network is dependent on the magnitude of the weights (Bartlett et al. 2017; Neyshabur 2017; Bartlett 1998; Golowich et al. 2020). Early results, such as Bartlett (1998), present bounds that assume sigmoidal activation functions, but nevertheless relate generalisation to the sum of the absolute values of the weights in the network. More recent work has shown that the product of spectral norms, scaled by various other weight matrix norms, can be used to construct bounds on the generalisation gap. Bartlett et al. (2017) scale the spectral norm product by a term related to the element-wise ℓ^1 norm, whereas Neyshabur et al. (2018) use the Frobenius norm. The key quantity used in these bounds is the Lipschitz constant of the parameters of a class of neural networks, which are in turn used in covering number arguments to bound the generalisation performance of models in the hypothesis space.

Neyshabur et al. (2018) speculate that Lipschitz continuity with respect to the ℓ^2 norm alone is insufficient to guarantee generalisation. However, the upper bound presented in Sect. 3 appears in multiple generalisation bounds (Neyshabur 2017; Bartlett et al. 2017; Golowich et al. 2020), and we show empirically in this paper that it is an effective aide for controlling the generalisation performance of a deep network. Moreover, the work of Xu and Mannor (2012) demonstrate the concrete link between the Lipschitz constant of a model with respect to its inputs and the resulting generalisation performance. This is accomplished using robustness theory, rather than the tools more typically used in learning theoretic bounds, such as Rademacher complexity and VC dimensions (Shalev-Shwartz et al. 2014). Interestingly, Golowich et al. (2020) present a bound on the Rademacher complexity of deep networks that depends only on the product of ℓ^∞ operator norms for each weight matrix, which corresponds exactly to the upper bound for the ℓ^∞ Lipschitz constant we consider in this paper. This provides yet more evidence that constraining the ℓ^∞ Lipschitz constant of a network is a principled method for improving generalisation performance.

Yoshida and Miyato (2017) propose a new regularisation scheme that adds a term to the loss function which penalises the sum of spectral norms of the weight matrices. This is related to but different from what we do in this paper. Firstly, we investigate norms other than ℓ^2. Secondly, Yoshida and Miyato (2017) use a penalty term, whereas we employ a hard constraint on the induced weight matrix norm, and they penalise the sum of the norms. The Lipschitz constant is determined by the product of operator norms. Finally, they use a heuristic to regularise convolutional layers. Specifically, they compute the largest singular value of a flattened weight tensor, as opposed to deriving the true matrix corresponding to the linear operation performed by convolutional layers, as we do in Sect. 3.2. Explicitly constructing this matrix and computing its largest singular value—even approximately—would be prohibitively expensive. We provide efficient methods for computing the ℓ^1 and ℓ^∞ norms of convolutional layers exactly, and show how one can approximate the spectral norm efficiently by avoiding the need to explicitly construct the matrix representing the linear operation performed by convolutional layers. Balan et al. (2017) provide a means for computing an upper bound to the Lipschitz constant of a restricted class of neural networks known as scattering networks. Although their approach computes tighter bounds than those presented in this paper for the networks they consider, most neural networks that are used in practice do not fit into the scattering network framework.

Enforcing Lipschitz continuity of a network is not only interesting for regularisation. Miyato et al. (2018) show that constraining the weights of the discriminator in a generative adversarial network to have a specific spectral norm can improve the quality of generated samples. They use the same technique as Yoshida and Miyato (2017) to compute these norms, and thus may benefit from the improvements presented in this paper. Szegedy et al. (2014) demonstrate that a naïve approach to constraining the Lipschitz constant can improve the adversarial robustness of neural networks.

Several pieces of related work have been carried out concurrently to this study. Sedghi et al. (2018) propose a method for characterising all the singular values of a convolutional layer through the use of Fourier analysis. Tsuzuku e al. (2018) propose a similar method for computing the spectral norm of a convolutional layer, with the intention of regularising it in order to improve the adversarial robustness of the resulting model. Zou et al. (2019) propose a general framework for computing bounds on Lipschitz constants by solving a linear program.

3 Computing the Lipschitz constant

A function, $f : X \rightarrow Y$, is said to be Lipschitz continuous if it satisfies

$$D_Y(f(\mathbf{x}_1), f(\mathbf{x}_2)) \leq k D_X(\mathbf{x}_1, \mathbf{x}_2) \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in X, \tag{1}$$

for some real-valued $k \geq 0$, and metrics D_X and D_Y. The value of k is known as the Lipschitz constant, and the function can be referred to as being k-Lipschitz. Generally, we are interested in the smallest possible Lipschitz constant, but it is not always possible to find it. In this section, we show how to compute an upper bound to the Lipschitz constant of a feed-forward neural network with respect to the input features. Such networks can be expressed as a series of function compositions:

$$f(\mathbf{x}) = (\phi_l \circ \phi_{l-1} \circ \ldots \circ \phi_1)(\mathbf{x}), \tag{2}$$

where each ϕ_i is an activation function, linear operation, or pooling operation. A particularly useful property of Lipschitz functions is how they behave when composed: the composition of a k_1-Lipschitz function, f_1, with a k_2-Lipschitz function, f_2, is a $k_1 k_2$-Lipschitz function. Denoting the Lipschitz constant of some function, f, as $L(f)$, repeated application of this composition property yields the following upper bound on the Lipschitz constant for the entire feed-forward network:

$$L(f) \leq \prod_{i=1}^{l} L(\phi_i). \tag{3}$$

Thus, we can compute the Lipschitz constants of each layer in isolation and combine them in a modular way to establish an upper bound on the constant of the entire network. It is important to note that $k_1 k_2$ will not necessarily be the smallest Lipschitz constant of $(f_2 \circ f_1)$, even if k_1 and k_2 are individually the best Lipschitz constants of f_1 and f_2, respectively. It is possible in theory that a tighter upper bound can be obtained by considering the entire network as a whole rather than each layer in isolation. In the remainder of this section, we derive closed form expressions for the Lipschitz constants of common layer types when D_X and D_Y correspond to ℓ^1, ℓ^2, or ℓ^∞ norms respectively. As we will see in Sect. 4, Lipschitz constants with respect to these norms can be constrained efficiently.

3.1 Fully connected layers

A fully connected layer, $\phi^{fc}(\mathbf{x})$, implements an affine transformation parameterised by a weight matrix, W, and a bias vector, $\mathbf{b}$:

$$\phi^{fc}(\mathbf{x}) = W\mathbf{x} + \mathbf{b}. \tag{4}$$

Others have already established that, under the ℓ^2 norm, the Lipschitz constant of a fully connected layer is given by the spectral norm of the weight matrix (Miyato et al. 2018; Neyshabur 2017). We provide a slightly more general formulation that will prove to be more useful when considering other p-norms. We begin by plugging the definition of a fully connected layer into the definition of Lipschitz continuity:

$$\|(W\mathbf{x}_1 + \mathbf{b}) - (W\mathbf{x}_2 + \mathbf{b})\|_p \leq k\|\mathbf{x}_1 - \mathbf{x}_2\|_p. \tag{5}$$

By setting $\mathbf{a} = \mathbf{x}_1 - \mathbf{x}_2$ and simplifying the expression slightly, we arrive at

$$\|W\mathbf{a}\|_p \leq k\|\mathbf{a}\|_p, \tag{6}$$

which, assuming $\mathbf{x}_1 \neq \mathbf{x}_2$, can be rearranged to

$$\frac{\|W\mathbf{a}\|_p}{\|\mathbf{a}\|_p} \leq k, \quad \mathbf{a} \neq 0. \tag{7}$$

The smallest Lipschitz constant is therefore equal to the supremum of the left-hand side of the inequality,

$$L(\phi^{fc}) = \sup_{\mathbf{a} \neq 0} \frac{\|W\mathbf{a}\|_p}{\|\mathbf{a}\|_p}, \tag{8}$$

which is the definition of the operator norm of W.

For the p-norms we consider in this paper, there exist efficient algorithms for computing operator norms on relatively large matrices. Specifically, for $p = 1$, the operator norm is the maximum absolute column sum norm; for $p = \infty$, the operator norm is the maximum absolute row sum norm. The time required to compute both of these norms is linearly related to the number of elements in the weight matrix. When $p = 2$, the operator norm is given by the largest singular value of the weight matrix—the spectral norm—which can be approximated relatively quickly using a small number of iterations of the power method.

3.2 Convolutional layers

Convolutional layers, $\phi^{conv}(X)$, also perform an affine transformation, but it is usually more convenient to express the computation in terms of discrete convolutions and point-wise additions. For a convolutional layer, the ith output feature map is given by

$$\phi_i^{conv}(X) = \sum_{j=1}^{M_{l-1}} F_{i,j} * X_j + B_i, \tag{9}$$

where each $F_{i,j}$ is a filter, each X_j is an input feature map, B_i is an appropriately shaped bias tensor exhibiting the same value in every element, and the previous layer produced M_{l-1} feature maps.

The convolutions in Eq. (9) are linear operations, so one can exploit the isomorphism between linear operations and square matrices of the appropriate size to reuse the matrix norms derived in Sect. 3.1. To represent a single convolution operation as a matrix–vector multiplication, the input feature map is serialised into a vector, and the filter coefficients are used to construct a doubly block circulant matrix. Due to the structure of doubly block circulant matrices, each filter coefficient appears in each column and row of this matrix exactly once. Consequently, the ℓ^1 and ℓ^∞ operator norms are the same and given by $\|F_{i,j}\|_1$, the sum of the absolute values of the filter coefficients used to construct the matrix.

Summing over several different convolutions associated with different input feature maps and the same output feature map, as done in Eq. (9), can be accomplished by horizontally concatenating matrices. For example, suppose $V_{i,j}$ is a matrix that performs a convolution of $F_{i,j}$ with the jth feature map serialised into a vector. Equation (9) can now be rewritten in matrix form as

$$\phi_i^{conv}(\mathbf{x}) = [V_{1,1}\ V_{1,2}\ \dots\ V_{1,M_{l-1}}]\mathbf{x} + \mathbf{b}_i, \tag{10}$$

where the inputs and biases, previously represented by X and B_i, have been serialised into vectors $\mathbf{x}$ and $\mathbf{b}_i$, respectively. The complete linear transformation, W, performed by a convolutional layer to generate M_l output feature maps can be constructed by adding additional rows to the block matrix:

$$W = \begin{bmatrix} V_{1,1} & \dots & V_{1,M_{l-1}} \\ \vdots & \ddots & \\ V_{M_l,1} & & V_{M_l,M_{l-1}} \end{bmatrix}. \tag{11}$$

To compute the ℓ^1 and ℓ^∞ operator norms of W, recall that the operator norm of $V_{i,j}$ for $p \in \{1, \infty\}$ is $\|F_{i,j}\|_1$. A second matrix, W', can be constructed from W, where each block, $V_{i,j}$, is replaced with the corresponding operator norm, $\|F_{i,j}\|_1$. Each of these operator norms can be thought of as a partial row or column sum for the original matrix, W. Now, based on the discussion in Sect. 3.1, the ℓ^1 operator norm is given by

$$\|W\|_1 = \max_j \sum_{i=1}^{M_l} \|F_{i,j}\|_1, \tag{12}$$

and the ℓ^∞ operator norm is given by

$$\|W\|_\infty = \max_i \sum_{j=1}^{M_{l-1}} \|F_{i,j}\|_1. \tag{13}$$

We now consider the spectral norm for convolutional layers. Yoshida and Miyato (2017) and Miyato et al. (2018) both investigate the effect of penalising or constraining the spectral norm of convolutional layers by reinterpreting the weight tensor of a convolutional layer as a matrix,

$$U = \begin{bmatrix} \mathbf{u}_{1,1} & \cdots & \mathbf{u}_{1,M_{l-1}} \\ \vdots & \ddots & \\ \mathbf{u}_{M_l,1} & & \mathbf{u}_{M_l,M_{l-1}} \end{bmatrix}, \tag{14}$$

where each $\mathbf{u}_{i,j}$ contains the elements of the corresponding $F_{i,j}$ serialised into a row vector. They then proceed to compute the spectral norm of U, rather than computing the spectral norm of W, given in Eq. (11). As Cisse et al. (2017) and Tsuzuku e al. (2018) show, this only computes a loose upper bound of the true spectral norm.

Explicitly constructing W and applying a conventional singular value decomposition to compute the spectral norm of a convolutional layer is infeasible, but we show how the power method can be adapted to use standard convolutional network primitives to compute it efficiently. Consider the usual process for computing the largest singular value of a square matrix using the power method, provided in Algorithm 1. The expression of most interest to us is inside the for loop, namely

$$\mathbf{x}_i = W^T W \mathbf{x}_{i-1}, \tag{15}$$

which, due to the associativity of matrix multiplication, can be broken down into two steps:

$$\mathbf{x}'_i = W \mathbf{x}_{i-1} \tag{16}$$

and

$$\mathbf{x}_i = W^T \mathbf{x}'_i. \tag{17}$$

When W is the matrix in Eq. (11), the expressions given in Eqs. (16) and (17) correspond to a forward propagation and a backwards propagation through a convolutional layer, respectively. Thus, if we replace these matrix multiplication with convolution and transposed convolution operations respectively, as implemented in many deep learning frameworks, the spectral norm can be computed efficiently. Note that only a single vector must undergo the forward and backward propagation operations, rather than an entire batch of instances. This means, for most cases, only a small increase in runtime will be incurred by using this method. It also automatically takes into account the padding and stride hyperparameters used by the convolutional layer. In contrast to the reshaping method used by Yoshida and Miyato (2017) and Miyato et al. (2018), the approach we use is capable of computing the spectral norm of a convolutional layer exactly if it is run until convergence.

Algorithm 1 Power method for producing the largest singular value, σ_{max}, of a non-square matrix, W.

```
Randomly initialise $\mathbf{x}_0$
for $i = 1$ to $n$ do
    $\mathbf{x}_i \leftarrow W^T W \mathbf{x}_{i-1}$
end for
$\sigma_{max} \leftarrow \frac{\|W\mathbf{x}_n\|_2}{\|\mathbf{x}_n\|_2}$
```

3.3 Pooling layers and activation functions

Computing Lipschitz constants for pooling layers and activations is trivial for commonly used pooling operations and activation functions. Most common activation functions and pooling operations are, at worst, 1-Lipschitz with respect to all p-norms. For example, the maximum absolute sub-gradient of the ReLU activation function is 1, which means that ReLU operations have a Lipschitz constant of one. A similar argument yields that the Lipschitz constant of max pooling layers is one. The Lipschitz constant of the softmax is one (Gao and Pavel 2017).

3.4 Residual connections

Recently developed feed-forward architectures often include residual connections between non-adjacent layers (He et al. 2016). These are most commonly used to construct structures known as residual blocks:

$$\phi^{res}(\mathbf{x}) = \mathbf{x} + (\phi_{j+n} \circ \ldots \circ \phi_{j+1})(\mathbf{x}), \tag{18}$$

where the function composition may contain a number of different linear transformations and activation functions. In most cases, the composition is formed by two convolutional layers, each preceded by a batch normalisation layer[1] and a ReLU function. While networks that use residual blocks still qualify as feed-forward networks, they no longer conform to the linear chain of function compositions we formalised in Eq. (2). Fortunately, networks with residual connections are usually built by composing a linear chain of residual blocks of the form given in Eq. (18). Hence, the Lipschitz constant of a residual network will be the product of Lipschitz constants for each residual block. Each block is a sum of two functions (see Eq. 18). Thus, for a k_1-Lipschitz function, f_1, and a k_2-Lipschitz function, f_2, we are interested in the Lipschitz constant of their sum:

$$\|(f_1(\mathbf{x}_1) + f_2(\mathbf{x}_1)) - (f_1(\mathbf{x}_2) + f_2(\mathbf{x}_2))\|_p \tag{19}$$

which can be rearranged to

$$\|(f_1(\mathbf{x}_1) - f_1(\mathbf{x}_2)) + (f_2(\mathbf{x}_1) - f_2(\mathbf{x}_2))\|_p. \tag{20}$$

The subadditivity property of norms and the Lipschitz constants of f_1 and f_2 can then be used to bound Eq. (20) from above:

[1] We discuss batch normalisation and the corresponding Lipschitz constant in Sect. 4.2 below.

$$\begin{aligned} &\|(f_1(\mathbf{x}_1) - f_1(\mathbf{x}_2)) + (f_2(\mathbf{x}_1) - f_2(\mathbf{x}_2))\|_p \\ &\quad \leq \|f_1(\mathbf{x}_1) - f_1(\mathbf{x}_2)\|_p + \|f_2(\mathbf{x}_1) - f_2(\mathbf{x}_2)\|_p \end{aligned} \tag{21}$$

$$\leq k_1 \|\mathbf{x}_1 - \mathbf{x}_2\|_p + k_2 \|\mathbf{x}_1 - \mathbf{x}_2\|_p \tag{22}$$

$$= (k_1 + k_2)\|\mathbf{x}_1 - \mathbf{x}_2\|_p. \tag{23}$$

Thus, we can see that the Lipschitz constant of the addition of two functions is bounded from above by the sum of their Lipschitz constants. Setting f_1 to be the identity function and f_2 to be a linear chain of function compositions, we arrive at the definition of a residual block as given in Eq. (18). Noting that the Lipschitz constant of the identity function is one, we can see that the Lipschitz constant of a residual block is bounded by

$$L(\phi^{res}) \leq 1 + \prod_{i=j+1}^{j+n} L(\phi_i), \tag{24}$$

where the property given in Eq. (3) has been applied to the function compositions.

4 Constraining the Lipschitz constant

The assumption motivating our work is that adjusting the Lipschitz constant of a feed-forward neural network controls how well the model will generalise to new data. Using the composition property of Lipschitz functions, we have shown that the Lipschitz constant of a network is the product of the Lipschitz constants associated with its layers. Ideally, one would simply add a term to the training objective consisting of the product of weight matrix norms. In practice, we found it difficult to train any deep networks with such an approach, and we suspect this is due to very poor conditioning of the resulting optimisation problem, as the product of norms can become very large. Instead, controlling the Lipschitz constant of a network can be accomplished by constraining the Lipschitz constant of each layer in isolation. This can be achieved by performing constrained optimisation when training the network. In practice, we pick a single hyperparameter, λ, and use it to control the upper bound of the Lipschitz constant for each layer. This means the network as a whole will have a Lipschitz constant less than or equal to λ^d, where d is the depth of the network.

Instead of using a projected gradient descent method, one might be tempted to add a sum of norms penalty term, which would not have the same issues encountered when attempting to train with a product of norms term. Although the penalty and constraint-based formulations of common regularisation methods are equivalent when training linear models (Oneto et al. 2016), the line of reasoning used to prove this does not extend to deep learning for two reasons that we can see. First, it relies on the convexity of the objective function. Second, it assumes the resulting penalty-based regularisation algorithm finds a critical point of the training objective. In practice, this is rarely the case in deep learning—practitioners typically determine convergence by looking at the validation accuracy rather than the training loss or gradient magnitudes.

The easiest way to adapt existing deep learning methods to allow for constrained optimisation is to introduce a projection step and perform a variant of the projected stochastic gradient method. In our particular problem, because each parameter matrix is constrained in isolation, it is straightforward to project any infeasible parameter values back into the set of feasible matrices. Specifically, after each weight update step, we must check that none of the weight matrices (including the filter banks in the convolutional layers) are violating the constraint on the Lipschitz constant. If the weight update has caused a weight matrix to leave the feasible set, we must replace the resulting matrix with the closest matrix that does lie in the feasible set. This can all be accomplished with the projection function

$$\pi(W, \lambda) = \frac{1}{\max\left(1, \frac{\|W\|_p}{\lambda}\right)} W, \tag{25}$$

which will leave the matrix untouched if it does not violate the constraint, and project it back to the closest matrix in the feasible set if it does. We measure closeness by the matrix distance metric induced by taking the operator norm of the difference between two matrices. Note that in order for the stochastic subgradient method to have guaranteed convergence, the measure of closeness should actually be Euclidean distance (Bubeck 2015). In practice, the results of our experiments show that our improper projection method does not have an adverse impact on performance—likely because current techniques for optimising deep networks already disregard conditions required for convergence. This method of projection will work with any valid operator norm because all norms are absolutely homogeneous (Pugh 2002). In particular, it will work with the operator norms with $p \in \{1, 2, \infty\}$, which can be computed using the approaches outlined in Sect. 3.

Pseudocode for this projected gradient method is given in Algorithm 2. We have observed fast convergence when using the Adam update rule (Kingma and Ba 2015), but other variants of the stochastic gradient method also work. For example, in our experiments, we show that stochastic gradient descent with Nesterov's momentum is compatible with our approach.

Algorithm 2 Projected stochastic gradient method to optimise a neural network subject to the Lipschitz Constant Constraint (LCC). $W_{1:l}$ is used to refer to all W_i for $i \in \{1, ..., l\}$.

```
$t \leftarrow 0$
while $W_{1:l}^{(t)}$ not converged do
    $t \leftarrow t + 1$
    $g_{1:l}^{(t)} \leftarrow \nabla_{W_{1:l}} f(W_{1:l}^{(t-1)})$
    $\widehat{W}_{1:l}^{(t)} \leftarrow update(W_{1:l}^{(t-1)}, g_{1:l}^{(t)})$
    for $i = 1$ to $l$ do
        $W_i^{(t)} \leftarrow \pi(\widehat{W}_i^{(t)}, \lambda)$
    end for
end while
```

4.1 Stability of *p*-norm estimation

A natural question to ask is which p-norm should be chosen when using the training procedure given in Algorithm 2. The Euclidean (i.e., spectral) norm is often seen as the default choice, due to its special status when talking about distances in the real world. Like Yoshida and Miyato (2017), we use the power method to estimate the spectral norms of the linear operations in deep networks. The convergence rate of the power method is related to the ratio of the two largest singular values, $\frac{\sigma_2}{\sigma_1}$ (Larson 2016). If the two largest singular values are almost the same, it will converge very slowly. Because each iteration of the power method for computing the spectral norm of a convolutional layer requires both forward propagation and backward propagation, it is only feasible to perform a small number of iterations before one will notice an impact in the training speed. However, regardless of the quality of the approximation, we can be certain that it does not overestimate the true norm: the expression in the final line of Algorithm 1 is maximised when $\mathbf{x}_n$ is the first eigenvector of W. Therefore, if the algorithm has not converged, $\mathbf{x}_n$ will not be a singular vector of W and our approximation of σ_{max} will be an underestimate.

In contrast to the spectral norm, we compute the values of the ℓ^1 and ℓ^∞ norms exactly, in time that is linear in the number of weights in a layer, so it always comprises a relatively small fraction of the overall runtime for training the network. Of course, it may be the case that the ℓ^1 and ℓ^∞ constraints do not provide as suitable an inductive bias as the ℓ^2 constraint. This is something we investigate in our experimental evaluation.

4.2 Compatibility with batch normalisation

Constraining the Lipschitz constant of the network will have an impact on the magnitude of the activations produced by each layer, which is what batch normalisation attempts to explicitly control (Ioffe and Szegedy 2015). Thus, we consider whether batch normalisation is compatible with our Lipschitz constant constraint (LCC) regulariser. Batch normalisation can be expressed as

$$\phi^{bn}(\mathbf{x}) = \operatorname{diag}\left(\frac{\boldsymbol{\gamma}}{\sqrt{\operatorname{Var}[\mathbf{x}]}}\right)(\mathbf{x} - \operatorname{E}[\mathbf{x}]) + \boldsymbol{\beta}, \tag{26}$$

where $\operatorname{diag}(\cdot)$ denotes a diagonal matrix, and $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned parameters. This can be seen as performing an affine transformation with a linear transformation term

$$\operatorname{diag}\left(\frac{\boldsymbol{\gamma}}{\sqrt{\operatorname{Var}[\mathbf{x}]}}\right)\mathbf{x}. \tag{27}$$

Based on the operator norm of this diagonal matrix, the Lipschitz constant of a batch normalisation layer, with respect to the three p-norms we consider, is given by

$$L(\phi^{bn}) = \max_i \left|\frac{\boldsymbol{\gamma}_i}{\sqrt{\operatorname{Var}[\mathbf{x}_i]}}\right|. \tag{28}$$

Thus, when using batch normalisation in conjunction with our technique, the $\boldsymbol{\gamma}$ parameter must also be constrained. This is accomplished by using the expression in Eq. (28) to compute the operator norm in the projection function given in Eq. (25). In practice, when training the network with minibatch gradient descent, we use a moving average estimate of the variance for performing the projection, rather than the variance computed solely on the current minibatch of training examples. This is done because the minibatch estimates of the mean and variance can be quite noisy.

4.3 Interaction with dropout

In the standard formulation of dropout, one corrupts activations during training by performing pointwise multiplication with vectors of Bernoulli random variables. As a consequence, when making a prediction at test time—when units are not dropped out—the activations must be scaled by the probability that they remained uncorrupted during training. This means the activation magnitude at both test time and training time is approximately the same. The majority of modern neural networks make extensive use of rectified linear units, or similar activation functions that are also homogeneous. This implies that scaling the activations at test time is equivalent to scaling the weight matrices in the affine transformation layers. That is, for a homogeneous activation function, $\varphi(\cdot)$, and a dropout rate of r, we have

$$(1-r)\varphi(W\mathbf{x}+\mathbf{b}) = \varphi((1-r)W\mathbf{x}+\mathbf{b}).$$

Moreover, from the homogeneity of norms we also have that

$$\|(1-r)W\|_p = (1-r)\|W\|_p,$$

indicating that when a network is trained with dropout, the Lipschitz constant of each layer is scaled by $1-r$. As a result, one may expect that when using our technique in conjunction with dropout, the λ hyperparameter will need to be increased in order to maintain the desired Lipschitz constant. Note that this does not imply that the optimal value for λ, from the point of view of generalisation performance, can be found by performing hyperparameter optimisation without dropout, and then dividing the best λ found on the validation set by one minus the desired dropout rate: the change in optimisation dynamics and regularisation properties of dropout make it difficult to predict analytically how these two methods interact when considering generalisation performance.

5 Experiments

The experiments in this section aim to answer several questions about the behaviour of the Lipschitz constant constraint (LCC) regularisation scheme presented in this paper. The question of most interest is how well this regularisation technique compares to related regularisation methods, in terms of accuracy measured on held-out data. In addition to this, experiments are performed that demonstrate how sensitive the method is to the choice of values of the λ hyperparameters, how it interacts with existing regularisation methods, and how the additional inductive bias imposed on the learning system impacts the sample efficiency.

Several different network architectures are employed in the experiments. Specifically, fully connected multi-layer perceptrons, VGG-style convolutional networks, and networks with residual connections are used. This is to ensure that the regularisation method works for a broad range of feed-forward architectures. SGD with Nesterov momentum is used for training networks with residual connections, and the Adam optimiser (Kingma and Ba 2015) is used otherwise. Batch normalisation is used in all networks to accelerate training. All regularisation hyperparameters for the convolutional networks were optimised on a per-layer type basis using the hyperopt package[2] of Bergstra et al. (2015). Separate dropout, spectral decay, and λ hyperparameters were optimised for fully connected and convolutional layers. All network weights were initialised using the method of Glorot and Bengio (2010), and the estimated accuracy reported in all tables is the mean of five networks that

[2] https://github.com/hyperopt/hyperopt.

Table 1 Performance of VGG19 and WRN-16-10 networks trained with spectral decay, dropout, LCC, and combinations thereof on CIFAR-10

Method	VGG19	WRN-16-10
None	90.43 ± 0.17	95.13 ± 0.17
Dropout	90.46 ± 0.10	95.46 ± 0.20
Spectral decay	90.14 ± 0.16	95.21 ± 0.08
LCC-ℓ^1	92.48 ± 0.13	95.34 ± 0.21
LCC-ℓ^2	92.57 ± 0.28	95.32 ± 0.15
LCC-ℓ^∞	91.64 ± 0.20	**95.82 ± 0.19**
Dropout + spectral decay	90.29 ± 0.17	95.22 ± 0.08
Dropout + LCC-ℓ^1	91.72 ± 0.17	95.47 ± 0.15
Dropout + LCC-ℓ^2	91.23 ± 0.24	95.57 ± 0.15
Dropout + LCC-ℓ^∞	**92.71 ± 0.29**	95.68 ± 0.11

Bold values indicate the method with the highest mean accuracy for each network

LCC-ℓ^p denotes the Lipschitz constant constraint method for a given p-norm

were each initialised using different seeds, unless stated otherwise. The standard deviation is also reported to give an idea of how robust different regularisers are to different initialisations. The code for running these experiments is available online.[3]

5.1 CIFAR-10

The CIFAR-10 dataset (Krizhevsky and Hinton 2009) contains 60,000 tiny images, each belonging to one of 10 classes. The experiments in this section follow the common protocol of using 10,000 of the images in the 50,000 image training set for tuning the model hyperparameters. Two network architectures are considered for this dataset: a VGG19-style network (Simonyan and Zisserman 2014), resized to be compatible with the 32×32 pixel images in CIFAR-10, and a wide residual network (WRN) (Zagoruyko and Komodakis 2016). All experiments on this dataset utilise data augmentation in the form of random crops and horizontal flips, and the image intensities were rescaled to fall into the $[-1, 1]$ range. Each VGG network is trained for 140 epochs using the Adam optimiser (Kingma and Ba 2015). The initial learning rate is set to 10^{-4} and decreased by a factor of 10 after epoch 100 and epoch 120. The WRNs are trained for a total of 200 epochs using the stochastic gradient method with Nesterov's momentum. The learning rate was initialised to 0.1, and decreased by a factor of 5 at epochs 60, 120, and 160.

The performance of LCC is compared to dropout and the spectral decay method of Yoshida and Miyato (2017). Dropout is a widely used regularisation method, often acting as key components of state-of-the-art models (Simonyan and Zisserman 2014; Kaiming He et al. 2016; Zagoruyko and Komodakis 2016), and the spectral decay method has a similar goal to the ℓ^2 instantiation of our method: encouraging the spectral norm of the weight matrices to be small. For this particular experiment, each regulariser is considered in isolation, but we also consider combinations of LCC and spectral decay with dropout. Results are given in Table 1. Interestingly, the performance of the VGG network

[3] https://github.com/henrygouk/keras-lipschitz-networks.

Table 2 Average time (in seconds) required per epoch to train a VGG-19 network on CIFAR-10

Method	Time (s)
None	40
Dropout	40
Spectral decay	48
LCC-ℓ^1	42
LCC-ℓ^2	52
LCC-ℓ^∞	42

Networks were trained using an NVIDIA V100 GPU

Table 3 Performance of networks trained with spectral decay, dropout, LCC, and combinations thereof on CIFAR-100

Method	VGG19	WRN-16-10
None	65.46 ± 0.43	77.94 ± 0.33
Dropout	66.75 ± 0.40	77.98 ± 0.24
Spectral decay	65.32 ± 0.24	77.93 ± 0.20
LCC-ℓ^1	69.59 ± 0.29	78.16 ± 0.04
LCC-ℓ^2	68.25 ± 0.38	79.00 ± 0.33
LCC-ℓ^∞	69.16 ± 0.22	79.39 ± 0.28
Dropout + spectral decay	66.97 ± 0.24	77.70 ± 0.33
Dropout + LCC-ℓ^1	70.17 ± 0.21	79.08 ± 0.11
Dropout + LCC-ℓ^2	**71.76 ± 0.26**	**79.45 ± 0.26**
Dropout + LCC-ℓ^∞	69.25 ± 0.43	78.17 ± 1.86

Bold values indicate the method with the highest mean accuracy for each network

LCC-ℓ^p denotes our Lipschitz constant constraint method for some given p-norm

varies considerably more than that of the Wide Residual Network. VGG networks see the most benefit from LCC-ℓ^2, but dropout and spectral decay do not provide any noticeable improvement in performance. Combining dropout with the other methods is not an effective strategy on this dataset. In the case of WRNs, LCC performs similarly to dropout and marginally better than spectral decay, but there is little separation between methods on this dataset.

We also report the average per-epoch runtime of training a VGG-19 under each regulariser in Table 2. Measurements were made using an NVIDIA V100 GPU. We can see that the ℓ^1 and ℓ^∞ variants of LCC result in negligible increases in runtime compared to using no regularisation or dropout, and the two approach that use the power method result in approximately 20–30% increase in runtime.

5.2 CIFAR-100

CIFAR-100, like CIFAR-10, is a dataset of 60,000 tiny images, but contains 100 classes rather than 10. The same data augmentation methods used for CIFAR-10 are also used for training models on CIFAR-100—random crops and horizontal flips. Once again, WRNs and VGG19-style networks are trained on this dataset. The learning rate schedules used in

Table 4 Test accuracies of the small convolutional networks trained with spectral decay, dropout, LCC, and combinations thereof on the MNIST and Fashion-MNSIT datasets

Method	MNIST	Fashion-MNIST
None	99.29 ± 0.03	92.54 ± 0.10
Dropout	98.93 ± 0.17	91.68 ± 0.17
Spectral decay	99.28 ± 0.07	92.59 ± 0.03
LCC-ℓ^1	99.41 ± 0.05	93.06 ± 0.15
LCC-ℓ^2	99.41 ± 0.05	92.62 ± 0.18
LCC-ℓ^∞	99.32 ± 0.09	92.87 ± 0.12
Dropout + spectral decay	98.85 ± 0.15	91.86 ± 0.18
Dropout + LCC-ℓ^1	99.35 ± 0.08	**93.23 ± 0.23**
Dropout + LCC-ℓ^2	**99.42 ± 0.10**	91.71 ± 0.38
Dropout + LCC-ℓ^∞	99.36 ± 0.04	92.75 ± 0.25

Bold values indicate the method with the highest mean accuracy for each network

the CIFAR-10 experiments also worked well on this dataset, which is not surprising given their similarities. However, the regularisation hyperparameters were optimised specifically for CIFAR-100. The results for the VGG and WRN models are given in Table 3.

It can be seen that the Lipschitz-based regularisation scheme is an effective technique for improving generalisation of networks both with and without residual connections. The results on CIFAR-100 follow a similar trend to those observed on CIFAR-10: LCC performs the best, dropout provides a small increase in performance over no regularisation, and combining dropout other approaches can sometimes provide a small boost in accuracy. Spectral decay performs noticeably worse than LCC-ℓ^2, often having comparable performance to no regularisation.

5.3 MNIST and Fashion-MNIST

The Fashion-MNIST dataset (Xiao et al. 2017) is designed as a more challenging drop-in replacement for the original MNIST dataset of hand-written digits (LeCun et al. 1998). Both contain 70, 000 greyscale images labelled with one of 10 possible classes. The last 10,000 instances are used as the test set. The final 10,000 instances in the training set are used for measuring performance when optimising the regularisation hyperparameters. In these experiments, small convolutional networks are trained on both of these datasets with different combinations of regularisers. The networks contain only two convolutional layers, each consisting of 5×5 kernels, and both layers are followed by 2×2 max pooling layers. The first layer has 64 feature maps, and the second has 128. These layers feed into a fully connected layer with 128 units, which is followed by the output layer with 10 units. ReLU activations are used for all hidden layers, and each model is trained for 60 epochs using Adam (Kingma and Ba 2015). The learning rate was started at 10^{-4} and decreased by a factor of 10 at the fiftieth epoch.

The test accuracies for each of the models trained on these datasets are given in Table 4. For both datasets, dropout and spectral decay decrease performance, whereas LCC-ℓ^1 results in a consistent performance increase.

Table 5 Prediction accuracy of VGG-style and WRN-16-4 networks trained with spectral decay, dropout, LCC, and combinations thereof on the SVHN dataset

Method	VGG	WRN-16-4
None	96.90 ± 0.05	97.97 ± 0.04
Dropout	96.98 ± 0.10	98.23 ± 0.05
Spectral decay	96.88 ± 0.04	98.02 ± 0.04
LCC-ℓ^1	97.17 ± 0.09	98.00 ± 0.06
LCC-ℓ^2	96.94 ± 0.04	97.93 ± 0.07
LCC-ℓ^∞	97.35 ± 0.03	98.03 ± 0.05
Dropout + spectral decay	97.10 ± 0.06	98.15 ± 0.04
Dropout + LCC-ℓ^1	97.30 ± 0.07	98.21 ± 0.02
Dropout + LCC-ℓ^2	$\mathbf{97.73 \pm 0.30}$	98.17 ± 0.05
Dropout + LCC-ℓ^∞	97.32 ± 0.06	$\mathbf{98.24 \pm 0.06}$

Bold values indicate the method with the highest mean accuracy for each network

5.4 Street view house numbers

The Street View House Numbers dataset contains over 600, 000 images of digits extracted from Google’s Street View platform. Each image contains three colour channels and has a resolution of 32×32 pixels. As with the previous datasets, the only preprocessing performed is to rescale the input features to the range $[-1, 1]$. However, in contrast to the experiments on CIFAR-10 and CIFAR-100, no data augmentation is performed while training on this dataset. The first network architecture used for this dataset, which follows a VGG-style structure, is comprised of four conv–conv–maxpool blocks with 64, 128, 192, and 256 feature maps, respectively. This is followed by two fully connected layers, each with 512 units, and then the logistic regression layer. Due to the large training set size, it is only necessary to train for 20 epochs. The Adam optimiser (Kingma and Ba 2015) is used with an initial learning rate of 10^{-4}, which is decreased by a factor of 10 at epochs 15 and 18. Small WRN models are also trained on this dataset. Once again, due to the large size of the training set, it is sufficient to only train each network for 20 epochs in total. Therefore, compared to the WRNs trained on CIFAR-10 and CIFAR-100, a compressed learning rate schedule is used. The learning rate is started at 0.1, and is decreased by a factor of 5 at epochs 6, 12, and 16. Measurements of the test set performance for each of the models trained on SVHN are provided in Table 5.

For VGG, using dropout in conjunction with other approaches results in the best performance, but in isolation is not effective. LCC improves accuracy in both the VGG and WRN models, whereas spectral decay does not help for either.

5.5 Scaled ImageNet subset (SINS-10)

The SINS-10 dataset is a collection of 100,000 images taken from ImageNet by Gouk (2018). Each image in this dataset is 96×96 pixels and is labelled with one of 10 classes. What makes this dataset distinct from other commonly used image classification benchmarks is that it is divided into 10 non-overlapping and equal sized predefined folds. Within each fold, 9000 images are used for training and 1000 are used for testing. By gathering

Table 6 Prediction accuracies of VGG-style networks trained with spectral decay, dropout, batchnorm, LCC, and combinations thereof on the SINS-10 dataset

Method	VGG	+/−
None	63.73 ± 1.18	
Dropout	68.65 ± 1.00	+
Spectral decay	63.81 ± 1.25	
LCC-ℓ^1	71.24 ± 1.31	+
LCC-ℓ^2	69.96 ± 2.16	+
LCC-ℓ^∞	70.92 ± 2.04	+
Dropout + spectral decay	68.97 ± 1.31	
Dropout + LCC-ℓ^1	**72.18 ± 1.97**	+
Dropout + LCC-ℓ^2	71.15 ± 1.13	+
Dropout + LCC-ℓ^∞	70.99 ± 1.03	+

Bold value indicates the method with the highest mean accuracy for each network

The +/− column indicates whether adding LCC to the combination of regularisers results in a statistically significant improvement or degradation in performance at the 95% confidence level

Table 7 Prediction accuracies of WRNs trained with spectral decay, dropout, batchnorm, LCC, and combinations thereof on the SINS-10 dataset

Method	WRN-16-4	+/−
None	68.26 ± 1.89	
Dropout	68.14 ± 2.78	
Spectral decay	76.85 ± 1.29	+
LCC-ℓ^1	72.85 ± 1.63	+
LCC-ℓ^2	75.89 ± 2.02	+
LCC-ℓ^∞	74.09 ± 2.19	+
Dropout + spectral decay	**78.57 ± 1.37**	+
Dropout + LCC-ℓ^1	73.27 ± 1.19	+
Dropout + LCC-ℓ^2	77.93 ± 1.19	+
Dropout + LCC-ℓ^∞	76.80 ± 1.05	+

Bold value indicates the method with the highest mean accuracy for each network

The +/− column indicates whether adding LCC to the combination of regularisers results in a statistically significant improvement or degradation in performance at the 95% confidence level

multiple estimates of algorithm performance, one can perform hypothesis tests to determine statistically significant differences between methods.

The experiments conducted on SINS-10 in this paper make use of the same VGG-style and WRN network architectures used for the SVHN experiments. However, because each fold of the SINS-10 dataset has many fewer instances than SVHN, the number of epochs and learning rate schedules are changed. The VGG networks are trained for a total of 60 epochs, beginning with a learning rate of 10^{-4} that is decreased by a factor of 10 at epochs 40 and 50. The WRN models are trained for 100 epochs each, with a starting learning rate of 0.1 that is decreased by a factor of five at epochs

Table 8 Mean test set accuracies obtained using two repetitions of 5-fold cross validation

	None	DO	ℓ^1	ℓ^2	ℓ^∞	DO+ℓ^1	DO+ℓ^2	DO+ℓ^∞
dig44	96.96	95.79	96.83	96.85	**97.11**	96.04	96.04	96.81
letter	95.37	90.28	95.29	95.34	**96.42**	91.44	91.37	93.24
pendigits	99.44	99.14	99.45	99.45	**99.52**	99.21	99.25	99.41
sat	90.82	89.18	90.75	90.73	**91.00**	90.08	89.84	90.06
segment	95.37	93.70	95.91	95.89	**96.52**	93.90	93.85	95.52
spambase	94.11	93.88	94.06	93.86	**94.37**	93.68	93.68	94.06
twonorm	97.16	97.64	97.10	97.05	97.41	**97.71**	97.68	97.69
vehicle	78.02	72.52	77.84	78.07	**80.14**	74.94	74.65	77.60
vowel	86.21	68.18	82.98	83.13	**90.86**	70.61	71.21	77.07
waveform	85.40	86.16	86.00	85.82	86.51	**86.71**	86.54	86.59

The highest mean accuracy achieved on each dataset is bolded

30, 60 and 80. The regularisation hyperparameters are optimised on a per-fold basis. The final 1000 instances of the training set are repurposed as a validation set to determine the quality of a given hyperparameter setting. The results for these experiments are given in Table 6 for the VGG models, and Table 7 for the wide residual network models. Hypothesis tests are carried out using a paired *t*-test to determine whether using the regulariser improves performance. In the case where a method is used in conjunction with dropout, the hypothesis test compares the performance of the combination with that of dropout along.

In contrast to the previous experiments, spectral decay is a very effective regulariser for wide residual network models trained on this dataset, with the combination of dropout and spectral decay being the best results. The hypothesis tests indicate that for both architectures, networks trained with LCC perform statistically significantly better than comparable networks trained without LCC.

5.6 Fully connected networks

Neural networks consisting exclusively of fully connected layers have a long history of being applied to classification problems arising in data mining scenarios. To evaluate how well the LCC regularisers work on tabular data, we have trained fully connected networks on the classification datasets collected by Geurts and Wehenkel (2005). These datasets are primarily from the University California at Irvine dataset repository. The only selection criterion used by Geurts and Wehenkel (2005) is that they contain only numeric features. In these experiments, each network contains two hidden layers consisting of 100 units each, and uses the ReLU activation function. Two repetitions of 5-fold cross-validation are performed for each dataset. Hyperparameters for each regulariser were tuned on a per-fold basis using grid search. The accuracy of a particular hyperparameter combination tried during the grid search was determined using a hold-out set drawn from the training data in each fold. The values considered for dropping a unit when using dropout were $p \in \{0.2, 0.3, 0.4, 0.5\}$. The values considered for λ when using the ℓ^2 and ℓ^∞ approaches were $\{2, 4, ..., 18, 20\}$, and for the ℓ^1 variant we used $\{5, 10, ..., 45, 50\}$. Once again, the combination of LCC with each of the regularisation methods is also evaluated.

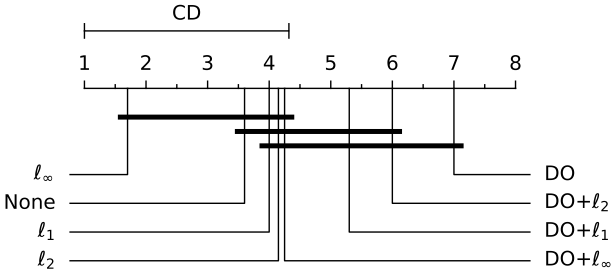
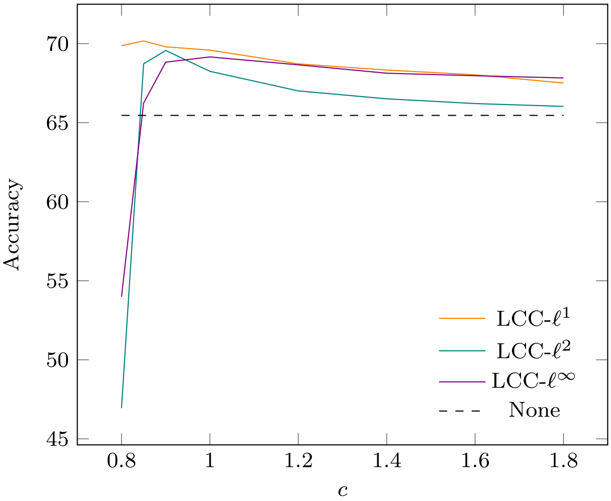


Fig. 1 A critical difference diagram showing the statistically significant (95% confidence) differences between the average rank of each method. The number beside each method is the average rank of that method across all datasets. The thick black bars overlaid on groups of thin black lines indicate a clique of methods that have not been found to be statistically significantly different

Fig. 2 This figure demonstrates the sensitivity of the algorithm to the choice of λ for each of the three p-norms when used to regularise VGG19 networks trained on the CIFAR-100 dataset. Because a different hyperparameter was optimised for each layer type, the horizontal axis represents the value of a single constant that is used to scale the three different λ hyperparameters associated with each curve. Note that when $c = 0.6$, the LCC-ℓ^1 network fails to converge



Several interesting trends can be found in Table 8. One particularly surprising trend is that the presence of dropout is a very good indicator of a degradation in accuracy. Interestingly, the only exceptions to this are the two synthetic datasets, where dropout is associated with an improvement in accuracy. LCC is one of the more reliable approaches to regularisation. In particular, the LCC-ℓ^∞ method achieves the highest mean accuracy on eight of the 10 datasets. On the other two datasets there is no substantial difference in performance between all methods. This provides strong evidence that LCC-ℓ^∞ is a good choice for regularisation of neural network models trained on tabular data.

These results can also be visualised using a critical difference diagram (Demšar 2006), as shown in Fig. 1. When ordering the methods by descending accuracy on each dataset, the average rank of LCC-ℓ^∞ is just over 1.5, whereas the next best method—using no regularisation at all—achieves an average rank of just over 3.5. However, there is insufficient evidence to be able to state that LCC-ℓ^∞ statistically significantly outperforms standard neural networks. Nevertheless, it can also be seen from this diagram that LCC-ℓ^∞ is statistically significantly better than most of the combinations of regularisers that include dropout.

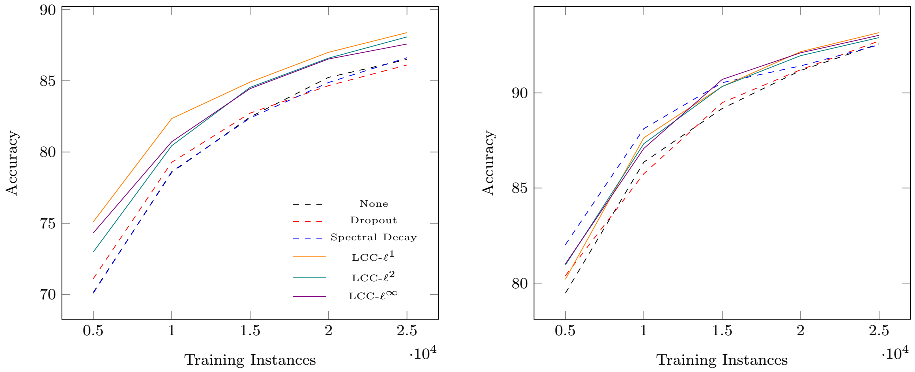


Fig. 3 Learning curves for VGG (left) and WRN (right) trained on CIFAR-10 with each of the regularisation methods

5.7 Sensitivity to λ

The ability to easily tune the hyperparameters of a regularisation method is important. The previous experiments have primarily taken advantage of automated hyperparameter tuning through the use of the hyperopt package (Bergstra et al. 2015), but investigating how sensitive the algorithm is to the choice of λ could lead to useful intuition for both manual hyperparameter tuning and automated methods. The networks that have been trained with LCC regularisation thus far have required up to three different λ hyperparameters—one for each parameterised layer type. Therefore, one cannot simply plot the model accuracy for given values of λ: it is not a scalar quantity. However, one can multiply all three of these hyperparameters by a single scalar value, and vary this scalar quantity to investigate the relationship between hyperparameter magnitude and generalisation performance. Figure 2 visualises this relationship using the CIFAR-100 dataset and several models with the VGG19-style architecture. This plot was generated by defining a hyperparameter vector, $\boldsymbol{\lambda} = [\lambda_{conv}, \lambda_{fc}, \lambda_{bn}]$, where each component is set to the value found during the hyperparameter optimisation procedure performed as part of the experiments carried out in Sect. 5.2. Each data point in the plot is created by training a network with hyperparameters specified by $c\boldsymbol{\lambda}$, where c is a user-provided scalar value, and plotting the resulting test set accuracy for different values of c.

One trend that is particularly salient in Fig. 2 is that choosing values for the hyperparameters that are even slightly too small results in a massive degradation in performance. Conversely, when c is set above the optimal value, each method exhibits a slow decline in performance until the accuracy is comparable to that of a network trained without any regularisation. Although this is the type of behaviour one might expect from a sensible means for controlling model capacity, this second phenomenon can cause difficulty during hyperparameter tuning. It is easy to determine when the hyperparameters have been assigned values that are too small, as the model fails to converge. However, it is not easy to determine how much the hyperparameters should be increased by. It was found that for each dataset, network architecture, and p-norm choice, vastly different hyperparameter settings were chosen by the automated tuning process. This means there is no typical range one should expect the optimal hyperparameters to lie in, and one must use a very uninformative prior when performing hyperparameter optimisation.

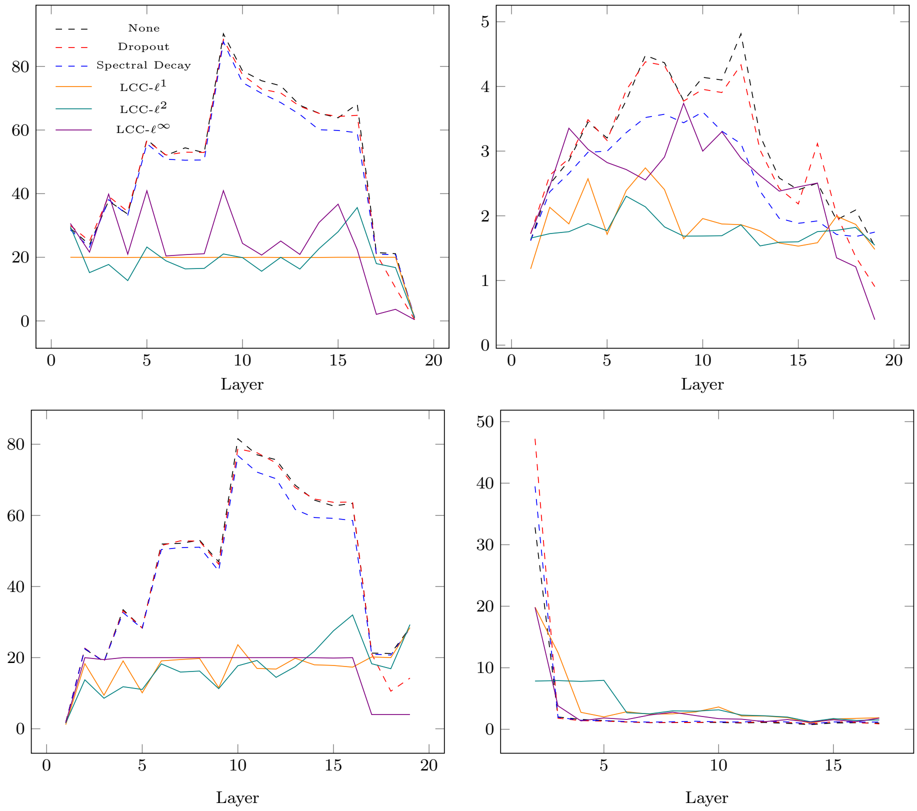


Fig. 4 Plots of the per-layer ℓ^1 (top left), ℓ^2 (top right), and ℓ^∞ (bottom left) Lipschitz constants for convolutional and fully connected operations in models trained with different regularisation methods. The bottom right plot shows the Lipschitz constants for the batch normalisation components

5.8 Sample efficiency

Imposing on a learning algorithm additional inductive biases that accurately reflect the underlying relationship between the input and output variables should result in a method that can produce well-performing models with fewer training examples than an algorithm without such inductive biases: more informative inductive biases should yield better sample efficiency. To determine if LCC improves the sample efficiency of training neural networks on image data, a series of networks are trained on progressively larger subsets of the CIFAR-10 training set. The full test set is still used for computing estimates of the accuracy of the resulting models. The learning curves for the VGG and WRN models are given in Fig. 3.

In the VGG plot, there is a difference of approximately 10 percentage points between the performance of the networks trained with LCC and those trained with one of the weaker baselines, for the case where only 5000 instances are used during training. As the number of available training instances is increased, the gap between the performance of all methods becomes smaller because each method must rely less on the prior knowledge built into the learning algorithm and more on the evidence provided by the examples in the training set. Interestingly, the wide residual networks trained with the spectral decay method achieve very good performance when only a small amount of training data is available. This agrees with the previous results on the SINS-10 dataset. However, it is interesting to note that as the number of available training examples grows, this advantage is lost and the performance of networks regularised with the spectral decay method tend towards the performance of the unregularised baseline—a trend that is also noticeable in the experiments on other datasets.

5.9 Do other methods constrain the Lipschitz constant?

The results presented so far have indicated that constraining the Lipschitz constant of a network provides effective regularisation, but it is interesting to consider how different the resulting Lipschitz constants are compared to not using LCC regularisation. To further investigate this, we supply plots of the Lipschitz constant of each layer of VGG-19 networks trained on CIFAR-10. These plots are given in Fig. 4. When the network is trained with dropout, we scale each of the operator norms by the probability of retaining an activation for the reasons described in Sect. 4.3. The constants for batch normalisation operations are plotted separately due to the large difference in magnitude. We identify two salient trends. First, the different variants of LCC result in significant reductions in the Lipschitz constant of each layer, and therefore the whole network. Second, spectral decay—another method aimed at reducing the ℓ^2 Lipschitz constant, but via a penalty approach—is less effective than LCC-ℓ^2.

6 Conclusion

This paper has presented a simple and effective regularisation technique for deep feed-forward neural networks called Lipschitz constant constraint (LCC), shown that it is applicable to a variety of feed-forward neural network architectures, and established that it is particularly suited to situations where only a small amount of training data is available. The investigation into the differences between the three p-norms ($p \in \{1, 2, \infty\}$) considered has provided some useful information about which one might be best-suited to the problem at hand. In particular, the ℓ^∞ norm appears particularly suitable for tabular data, and the ℓ^2 norm showed the most consistently competitive performance when used as a regulariser on natural image datasets. However, given that LCC-ℓ^2 with few power method iterations is only approximately constraining the norm, if one wants a guarantee that the Lipschitz constant of the trained network is bounded below some user-specified value, then using the ℓ^1 or ℓ^∞ norm would be more appropriate.

Lastly, recent and concurrent work suggests that the utility of constraining the Lipschitz constant of neural networks is not limited to improving classification accuracy. There is already evidence that constraining the Lipschitz constant of the discriminator networks in GANs is useful (Arjovsky et al. 2017; Miyato et al. 2018). Given the drawbacks in previous approaches to constraining Lipschitz constants we have outlined (cf. Sect. 3.2), one might expect improvements training GANs that are k-Lipschitz with respect to the ℓ^1 or ℓ^∞ norms, and approximately 1-Lipschitz with respect to the ℓ^2 norm, by applying the methods presented in this paper. Exploring how well the technique presented in this paper works with recurrent neural networks would also be of interest. Finally, the experiments carried out in this paper forced all layers of the same type to have the same Lipschitz constant. This is likely an inappropriate assumption in practice, and a more sophisticated hyperparameter tuning mechanism that allows for selecting a different value of λ for each layer could provide a further improvement to performance. However, devising a means for efficiently allocating modelling capacity on a per-layer basis is an open problem.

References

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. In *Proceedings of the 34th international conference on machine learning*.

Balan, R., Singh, M., & Zou, D. (2017). Lipschitz properties for deep convolutional networks. arXiv:1701.05217.

Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, *44*(2), 525–536.

Bartlett, P. L., Foster, D. J., & Telgarsky, M. J. (2017). Spectrally-normalized margin bounds for neural networks. In *Advances in neural information processing systems* (vol. 30).

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, *8*(1), 014008.

Bubeck, S. (2015). Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, *8*(3–4), 231–357.

Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., & Usunier, N. (2017). Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th international conference on machine learning*.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Gal, Y., Hron, J., & Kendall, A. (2017). Concrete dropout. In *Advances in neural information processing systems* (vol. 30).

Gao, B., & Pavel, L. (2017). On the properties of the softmax function with application in game theory and reinforcement learning. arXiv:1704.00805.

Geurts, P., & Wehenkel, L. (2005). Closed-form dual perturb and combine for tree-based models. In *Proceedings of the 22nd international conference on machine learning*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th international conference on artificial intelligence and statistics*.

Golowich, N., Rakhlin, A., & Shamir, O. (2020). Size-independent sample complexity of neural networks. *Information and Inference: A Journal of the IMA*, *9*(2), 473–504.

Gouk, H., Pfahringer, B., Frank, E., & Cree, M. J. (2018). MaxGain: Regularisation of neural networks by constraining activation magnitudes. In *Joint European conference on machine learning and knowledge discovery in databases*.

Hardt, M., Recht, B., & Singer, Y. (2016). Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33rd international conference on machine learning*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd international conference on machine learning*.

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations*.

Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems* (vol. 28).

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*. Master's Thesis, University of Toronto.

Larson, R. (2016). *Elementary linear algebra*. Toronto: Nelson Education.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. In *Proceedings of the 6th international conference on learning representations*.

Neyshabur, B. (2017). *Implicit regularization in deep learning*. PhD thesis, Toyota Technological Institute at Chicago, September 2017.

Neyshabur, B., Bhojanapalli, S., & Srebro, N. (2018). A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. In *Proceedings of the 6th international conference on learning representations*.

Oneto, L., Ridella, S., & Anguita, D. (2016). Tikhonov, Ivanov and Morozov regularization for support vector machine learning. *Machine Learning*, *103*(1), 103–136.

Pugh, C. C. (2002). *Real mathematical analysis*. Berlin: Springer.

Sedghi, H., Gupta, V., & Long, P. M. (2018). The singular values of convolutional layers. In *Proceedings of the 7th international conference on learning representations*.

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge: Cambridge University Press.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd international conference on learning representations*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. In *International conference on learning representations*.

Tsuzuku, Y., Sato, I., & Sugiyama, M. (2018). Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *Advances in neural information processing systems* (vol. 31).

Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., & Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th international conference on machine learning*.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747.

Xu, H., & Mannor, S. (2012). Robustness and generalization. *Machine Learning*, *86*(3), 391–423.

Yoshida, Y., & Miyato, T. (2017). Spectral norm regularization for improving the generalizability of deep learning. arXiv:1705.10941.

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. In *Proceedings of the 27th British machine vision conference*.

Zou, D., Balan, R., & Singh, M. (2019). On Lipschitz bounds of general convolutional neural networks. *IEEE Transactions on Information Theory*, *66*(3), 1738–1759.