View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Research Commons@Waikato

arXiv:1804.05965v2 [stat.ML] 1 Jul 2018

MaxGain: Regularisation of Neural Networks by Constraining Activation Magnitudes

Henry Gouk[1], Bernhard Pfahringer[1], Eibe Frank[1], and Michael Cree[2]

[1] Department of Computer Science, University of Waikato
hgrg1@students.waikato.ac.nz,
{bernhard.pfahringer,eibe.frank}@waikato.ac.nz
[2] School of Engineering, University of Waikato
michael.cree@waikato.ac.nz

Abstract. Effective regularisation of neural networks is essential to combat overfitting due to the large number of parameters involved. We present an empirical analogue to the Lipschitz constant of a feed-forward neural network, which we refer to as the maximum gain. We hypothesise that constraining the gain of a network will have a regularising effect, similar to how constraining the Lipschitz constant of a network has been shown to improve generalisation. A simple algorithm is provided that involves rescaling the weight matrix of each layer after each parameter update. We conduct a series of studies on common benchmark datasets, and also a novel dataset that we introduce to enable easier significance testing for experiments using convolutional networks. Performance on these datasets compares favourably with other common regularisation techniques.

Keywords: Deep learning, regularisation, classification, dataset

1 Introduction

Regularisation is a crucial component in machine learning systems. This is particularly true for neural networks, whose huge number of parameters can lead to extreme overfitting, such as memorising the training set—even in the case where the labels have been randomised [19]. In this work, we investigate a regularisation technique inspired by recent work regarding the Lipschitz continuity of neural networks [7]. Most work in machine learning that deals with the concept of Lipschitz continuity assumes, often implicitly [7,13], that the input domain of the function of interest is $\mathbb{R}^d$—sometimes with the additional assumption that each component in this vector space is bounded in, for example, the range $[-1, 1]$. However, when working with unstructured data—a task at which neural networks excel—a common assumption is that the data lie in a low dimensional manifold embedded in a high dimensional space. This is known as the manifold hypothesis [2]. In this paper, we explore the idea of constraining the Lipschitz continuity of neural network models when they are viewed in this light: as mappings from the subset of $\mathbb{R}^d$ that contains the low dimensional manifold, to some

meaningful vector space, such as the distribution over possible classes. The precise structure of the manifold is unknown to us, which makes constraining a function that operates on this manifold difficult. To circumvent this problem, we introduce the concept of *gain*—an empirical analogue to the operator norm technique used by Gouk et al. [7] to compute the Lipschitz constant of a neural network layer.

We present a regularisation scheme that improves the generalisation performance of neural networks by constraining the maximum gain of each layer. This is accomplished using a simple modification to conventional neural network optimisers that applies a stochastic projection function in addition to a stochastic estimate of the gradient. We demonstrate the effectiveness of our regularisation algorithm on several classification datasets. A novel dataset that facilitates significance testing for convolutional network-based classifiers is introduced as part of these experiments. Additionally, we show how our technique performs when used in conjunction with other regularisation methods such as dropout [17] and batch normalisation [9]. We also provide empirical evidence that constraining the gain on the training set results in observing lower gain on the test set compared to when the gain on the training set is not constrained. Details of how the performance of models trained with out regularisation technique as its hyperparameter is varied are also provided.

2 Related Work

Several recent publications have addressed the idea of Lipschitz continuity of neural networks. Most of this work has been on generative adversarial networks (GANs) [6]. Wasserstein GANs [1] are the first GAN variant that require some way of enforcing Lipschitz continuity in order to converge. They accomplish this by clipping each weight whenever its absolute value exceeds some predefined threshold. While this will maintain Lipschitz continuity, the Lipschitz constant will not be known. An alternative to weight clipping is to penalise the norm of the gradient of the critic network [8], which has been shown to improve the stability of training Wasserstein GANs. This technique for constraining Lipschitz continuity is similar to ours, in the sense that it uses an approximate measure of the Lipschitz constant on the training data. It is, however, quite different in the sense that it is not being used for regularisation and that it is applied as a soft constraint using a penalty term. Miyato et al. [13] have also proposed normalising the weights in each layer of the discriminator network of a GAN using the spectral norm of the respective weight matrix, but they provide no evidence showing that their heuristic for applying this to convolutional layers actually constrains the spectral norm. Some recent work has shown how to precisely compute and constrain the Lipschitz constant of a network with respect to the ℓ_1 and ℓ_∞ norms [7] and demonstrated that constraining the Lipschitz constant with respect to these norms has a regularising effect comparable to dropout and batch normalisation.

The idea of constraining the Lipschitz constant of a network is conceptually related to quantifying the flatness of minima. While there is no single formalisation for what constitutes a flat minimum, the unifying intuition is that a minimum is flat when a small perturbation of the model parameters does not have a large impact on the performance of the model. Dinh et al. [4] have shown that Lipschitz continuity is not a reliable tool for quantifying the flatness of minima. However, there is a subtle but very important difference between how they employ Lipschitz continuity, and how it is used by Gouk et al. [7] and in this work. Neural networks are functions parameterised by two distinct sets of variables: the model parameters, and the features. Dinh et al. [4] consider Lipschitz continuity with respect to the model parameters, whereas we consider Lipschitz continuity with respect the features being supplied to the network. The crux of the argument given by Dinh et al. is that the Lipschitz constant of a network with respect to its weights is not invariant to reparameterisation.

Dropout [17] is one of the most widely used methods for regularising neural networks. It is popular because it is efficient and easy to implement, requiring only that each activation is set to zero with some probability, p, during training. An extension proposed by Srivastava et al. [17], known as maxnorm, is to constrain the maginitude of the weight vector associated with each unit in some layer. One can also use multiplicative Gaussian noise, rather than Bernoulli noise. Kingma et al. [11] provide a technique that enables automatic tuning of the amount of noise that should be applied in the case of Gaussian dropout. A similar technique exists for automatically tuning p for Bernoulli dropout—this extension is known as concrete dropout [5].

Batch normalisation [9], which was originally motivated by the desire to improve the convergence rate of neural network optimisers, is often used as a regularisation scheme. It is similar to our technique in the sense that it rescales the activations of a layer, but it does so in a different way: by standardising them and subsequenty multiplying them by a learned scale factor. Unlike other regularisation techniques, there is no hyperparameter for batch normalisation that can be tuned to control the capacity of the network. A similar technique, which does not rely on measuring activation statistics over minibatches, is weight normalisation [15]. This approach decouples the length and direction of the weight vector associated with each unit in the network, and enables one to train networks on very small batch sizes, which is a situation where batch normalisation cannot be applied reliably.

3 Lipschitz Continuous Neural Networks

Gouk et al. (2018) [7] recently demonstrated that constraining the Lipschitz constant of a neural network improves generalisation in the context of classification. We briefly review their technique to aid overall understanding and provide several useful definitions. Recall the definition of Lipschitz continuity:

$$D_B(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2)) \leq k D_A(\boldsymbol{x}_1, \boldsymbol{x}_2) \quad \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in A, \tag{1}$$

for some real-valued $k \geq 0$, and metrics D_A and D_B. We refer to f as being k-Lipschitz. We are most interested in the smallest possible value of k, which is sometimes referred to as the best Lipschitz constant. A particularly useful property of Lipschitz continuity is that the composition of a k_1-Lipschitz function with a k_2-Lipschitz function is a k_1k_2-Lipschitz function. Given that a feed-forward neural network can be expressed as a series of function compositions,

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

one can compute the Lipschitz constant of the entire network by computing the constant of each layer in isolation and taking the product of these constants:

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

where $L(\phi_i)$ indicates the Lipschitz constant of some function, ϕ_i.

Many functions in this product, such as commonly used activation functions and pooling operations, have a Lipschitz constant of one for all vector p-norms on $\mathbb{R}^d$. Other commonly used functions, such as fully connected and convolutional layers, can be expressed as affine transformations,

$$f(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}, \tag{4}$$

where W is a weight matrix and $\boldsymbol{b}$ is a bias vector. For fully connected layers, there is no special structure to W. In the case of convolutional layers, W is a block matrix where each block is in turn a doubly block circulant matrix. Batch normalisation layers can also be expressed as affine transformations, where the linear operation is a diagonal matrix. Each element on the diagonal is one of the scaling parameters divided by the standard deviation of the corresponding activation. The Lipschitz constant of an affine function is given by the operator norm of the weight matrix,

$$\|W\|_p = \sup_{\boldsymbol{x} \neq 0} \frac{\|W\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}, \tag{5}$$

for some vector p-norm. For the ℓ_1 and ℓ_∞ vector norms, the matrix operator norms are given by the maximum absolute column sum and maximum absolute row sum norms, respectively. In the case of the ℓ_2 norm, the operator norm of a matrix is given by the spectral norm—the largest singular value. This can be approximated for fully connected layers relatively efficiently using the power iteration method. Once the operator norms have been computed, projected gradient methods can be used to constrain the Lipschitz constant of each layer to be less than a user specified value.

4 Regularisation by Constraining Gain

A common assumption in machine learning is that many types of unstructured data, such as images and audio, lie near a low dimensional manifold embedded in

a high dimensional vector space. This is known as the manifold hypothesis. If we assume that the manifold hypothesis holds, then a network will only be supplied with elements of some set $\mathcal{X} \subset \mathbb{R}^d$. As a consequence, the training procedure need only ensure that the network is Lipschitz continuous on $\mathcal{X}$ in order to construct a network with a slowly varying decision boundary. In practice, the exact structure of $\mathcal{X}$ is unknown, but we do have a finite sample of instances, $X \subset \mathcal{X}$, which we can use to empirically estimate various characteristics of $\mathcal{X}$. The work presented in this paper differs from that of Gouk et al. (2018) [7] in that an approximation of the Lipschitz constant on $\mathcal{X}$ is constrained, rather than the true Lipschitz constant on $\mathbb{R}^d$. The technique used to enforce the constraint during is detailed in this section.

4.1 Gain

Lipschitz continuity is not something that can be established empirically. However, one can find a lower bound for k by sampling pairs of points from the training set and determining the smallest value of k that satisfies Equation 1. This solution, while conceptually simple, has a number of finer details that can greatly impact the result. For example, how should pairs be sampled? If they are chosen randomly, then a very large number of pairs will be required to provide a good estimate of k. On the other hand, if a hard-negative mining approach were employed, fewer pairs would be required, but the amount of computation per pair would be greatly increased.

By restricting our analysis to feed-forward neural networks, we derive a simpler and more computationally efficient approach. Recall that the Lipschitz constant of a feed-forward network is given by the product of the Lipschitz constants associated with each activation function—which are usually less than or equal to one and cannot be changed during training—and the operator norms associated with the linear transformations in the learned layers. We define gain using the fraction from Equation 5,

$$Gain_p(W, \boldsymbol{x}) = \frac{\|W\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}, \tag{6}$$

for some input instance $\boldsymbol{x}$, and we use the maximum gain observed over some set of input vectors from our manifold of interest as an approximation of the operator norm. This empirical estimate of the operator norm of a matrix has several advantages over computing the true operator norm. Firstly, it fulfills our desire to approximately compute the Lipschitz constant of an affine function on $\mathcal{X}$. It is also well behaved, in the sense that $X = \mathcal{X} \implies \sup_{\boldsymbol{x}} Gain(W, \boldsymbol{x}) = \|W\|_p$. Some more practical advantages include not having to explicitly construct W, but merely requiring a means of computing $W\boldsymbol{x}$—a property that is extremely useful when computing the operator norm of a convolutional layer. Also, because one need not compute a matrix norm directly, it is possible to compute the gain with respect to a p-norm for which it would be NP-hard to compute the induced matrix operator norm.

4.2 MaxGain Regularisation

The crux of our regularisation technique is to limit the gain of each layer in a feed-forward neural network. Each layer is constrained, in isolation, to have a gain less than or equal to a user specified hyperparameter, γ. Put formally, we wish to solve the following optimisation problem:

$$W_{1..l} = \underset{W_{1..l}}{\arg\min} \sum_{\boldsymbol{x}_i^1 \in X} L(\boldsymbol{x}_i^1, \boldsymbol{y}_i) \tag{7}$$

$$s.t. \max_{\boldsymbol{x}_i^j} Gain_p(W_j, \boldsymbol{x}_i^j) \leq \gamma \qquad \forall j \in \{1 \dots l\}, \tag{8}$$

where $\boldsymbol{x}_i^j$ indicates the input to the jth layer for instance i, $\boldsymbol{y}_i$ is a label vector associated with instance i, W_j is the weight matrix for layer j, and $L(\cdot)$ is some task-specific loss function. Note that if $\|\boldsymbol{x}_i^j\|_p$ is zero, we set the gain for that particular measurement to zero rather than leaving it undefined.

The conventional approach to solving Equation 7 without the constraint in Equation 8 is to use some variant of the stochastic gradient method. For simple constraints, such as requiring W_j to lie in some known convex set, a projection function can be used to enforce the constraint after each parameter update. In our case, applying the projection function after each parameter update would involve propagating the entire training set through the network to measure the maximum gain for each layer. Even for modest sized datasets this is completely infeasible, and it defeats the purpose of using a stochastic optimiser. Instead, we propose the use of a stochastic projection function, where the max in Equation 8 is taken over the same minibatch used to compute an estimate of the loss function gradient. We reuse the "stale" activations computed before the weight update in order to avoid the extra computation required for propagating all of the instances through the network again. The following projection function is used:

$$\pi(W, \hat{\gamma}, \gamma) = \frac{1}{\max(1, \frac{\hat{\gamma}}{\gamma})} W, \tag{9}$$

where $\hat{\gamma}$ is our estimate of the maximum gain for layer j. If the MaxGain constraint is not violated, then W will be left untouched. If the constraint is violated, W will be rescaled to fix the violation. In the case where the maximum gain is computed exactly, this function will rescale the weight matrix such that the maximum gain is less than or equal to γ. Because we are only approximately computing the maximum gain, this constraint will not be perfectly satisfied on the training set.

During training, batch normalisation applies a transformation to the activations of a minibatch using statistics computed using only the instances contained in that minibatch. Thus, the gain measured for a particular instance is dependent on the other instances in the batch in which it is observed by the network. Specifically, the activations, $\boldsymbol{x}$, produced by some layer, are standardised:

$$\phi^{bn}(\boldsymbol{x}) = \mathrm{diag}(\frac{\boldsymbol{\alpha}}{\sqrt{\mathrm{Var}[\boldsymbol{x}]}})(\boldsymbol{x} - \mathrm{E}[\boldsymbol{x}]) + \boldsymbol{\beta}, \tag{10}$$

where $\mathrm{diag}(\cdot)$ denotes a diagonal matrix, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are learned parameters, and the $\mathrm{Var}[\cdot]$ and $\mathrm{E}[\cdot]$ operations are computed over only the instances in the current minibatch. If the estimated mean and variance values are particularly unstable, then the gain values will also be very unstable and the training procedure will converge very slowly—or possibly not at all. We have found that the high dimensionality of neural network hidden layer activation vectors, and their sparse nature when using the ReLU activation function, coupled with a relatively small batch size, leads to unstable measurements when using MaxGain in conjunction with batch normalisation. We remedy this by recomputing the batch normalisation output in the projection function using the running averages of the standard deviation estimates that are kept for performing test-time predictions. By standardising the minibatch activations using these more stable estimates of the activation statistics, we observed considerably more reliable convergence. Note that the stochastic estimates of the mean and standard deviation of activations are still used for computing the gradient—it is only the projection function that uses the running averages of these values.

Pseudocode for our constrained optimisation algorithm based on stochastic projection is provided in Algorithm 1. The inputs to each layer for each minibatch, $X_{1:l}^{(t)}$, and the results of transforming these by the linear term of the affine transformations, $Z_{1:l}^{(t)}$, are cached during the gradient computation to be reused in the projection function. We use a single hyperparameter, γ, to control the allowed gain of each layer. There is no fundamental reason that a different γ cannot be selected for each layer other than the added difficulty in optimising more hyperparameters. The $update(\cdot, \cdot)$ function can be any stochastic optimisation algorithm commonly used with neural networks. We consider both Adam [10] and SGD with Nesterov momentum.

4.3 Compatibility with Dropout

There are two parts to applying dropout regularisation to a network. Firstly, during training, one must stochastically corrupt the activations of some hidden layers, usually by multiplying them with vectors of Bernoulli random variables. Secondly, during test time, the activations are scaled such that the expected magnitude of each activation is the same as what it would have been during training. In the case of standard Bernoulli dropout, this just means multiplying each activation by the probability that it was not corrupted during training. This scaling is known to change the Lipschitz constant of a network over $\mathbb{R}^d$ [7], and the same argument applies to the Lipschitz constant on $\mathcal{X}$. Because many commonly used activation functions are homogeneous, namely ReLU and its many variants, scaling the output activations is equivalent to scaling the output of the affine transformation. This, in turn, has an identical effect to scaling both the weight matrix and bias vector. Due to the homogeneity of norms, this

Algorithm 1 This algorithm makes use of the stochastic gradient method (or some variation thereof) and a stochastic projection function to approximately solve the constrained optimisation problem outlined in Equations 7 and 8. In this procedure, $zip(\cdot,\cdot)$ constructs an array of tuples consisting of elements from the sequences passed as arguments.

$t \leftarrow 0$
while $W_{1:l}^{(t)}$ not converged **do**
  $t \leftarrow t+1$
  $(g_{1:l}^{(t)}, X_{1:l}^{(t)}, Z_{1:l}^{(t)}) \leftarrow \nabla_{W_{1:l}} f(W_{1:l}^{(t-1)})$
  $\widehat{W}_{1:l}^{(t)} \leftarrow update(W_{1:l}^{(t-1)}, g_{1:l}^{(t)})$
  for $i = 1$ **to** l **do**
    $\hat{\gamma} \leftarrow 0$
    for $(\boldsymbol{x}_j, W_i^{(t)}\boldsymbol{x}_j)$ **in** $zip(X_i^{(t)}, Z_i^{(t)})$ **do**
      $\hat{\gamma} \leftarrow max(\hat{\gamma}, \frac{\|W_i^{(t)}\boldsymbol{x}_j\|_p}{\|x_j\|_p})$
    end for
    $W_i^{(t)} \leftarrow \pi(\widehat{W}_i^{(t)}, \hat{\gamma}, \gamma)$
  end for
end while

scaling also directly affects the gain. Therefore, one might expect that one needs to increase γ when using our technique in conjunction with dropout.

5 Experiments

The experiments reported in this section aim to demonstrate several aspects of our MaxGain regularisation method. The primary question we wish to answer is whether our technique for constraining the maximum gain of each learned layer in a network is an effective regularisation method. We also demonstrate that constraining the gain on training instances results in observing lower gain on the test, compared to when the gain is not constrained at all. All networks trained with MaxGain regularisation use the same γ parameter for each layer in order to simplify hyperparameter optimisation. While the method we have presented can be used in conjunction with any vector norm, in this work we only investigate how well MaxGain works when using the ℓ_2 vector norm.

Throughout our experiments, we make use of several different datasets. We also introduce a novel dataset larger than some typical benchmark datasets, like CIFAR-10 and MNIST, yet smaller and more manageable than the ImageNet releases used for the Large Scale Visual Recognition challenges. This dataset is designed so that performing significance tests is easy, and a greater degree of confidence can therefore be attributed to conclusions drawn from experiments using this dataset. The pixel intensities of all images have been scaled to lie in the range $[-1, 1]$.

5.1 CIFAR-10

CIFAR-10 [12] is a collection of 60,000 tiny colour images, each labelled with one of 10 classes. In our experiments we follow the standard protocol of using 50,000 images for training and 10,000 images for testing. Additionally, we use a 10,000 image subset of the training set to tune the hyperparameters. We use the VGG-19 network [16] trained using the Adam optimiser [10]. The model is trained for 140 epochs, starting with a learning rate of 10^{-4}, which is decreased to 10^{-5} at epoch 100 and 10^{-6} at epoch 120. We make use of data augmentation in the form of horizontal flips, and padding training images to 40×40 pixels and cropping out a random 32×32 patch.

Results demonstrating how our technique compares with other common regularisation techniques are given in Table 1. Several trends stand out in this table. Firstly, when comparing with each other technique in isolation, our method performs noticeably better than dropout and similarly to batch normalisation. When used in conjunction with batch normalisation the resulting test accuracy improves further. Interestingly, combining the use of dropout with both other regularisation approaches does not seem to have a noticeable cumulative effect.

Table 1: Accuracy of a VGG-19 network trained in CIFAR-10 with different regularisation techniques.

Regulariser	Accuracy
None	88.29%
Dropout	89.71%
Batchnorm	90.80%
Batchnorm + Dropout	90.90%
MaxGain	90.75%
MaxGain + Dropout	90.95%
MaxGain + Batchnorm	91.76%
MaxGain + Batchnorm + Dropout	91.52%

5.2 CIFAR-100

CIFAR-100 [12] is similar to CIFAR-10, on account of containing 60,000 colour images of size 32×32, also split into a predefined set of $50,000$ for training and $10,000$ for testing. It differs in that it contains 100 classes, and exhibits more subtle inter-class variation. We use a Wide Residual Network [18] on this dataset, in order to investigate how well MaxGain works on networks with residual connections. Batch normalisation is applied to all models trained on this dataset. We found convergence to be unreliable when training Wide ResNets without batch normalisation. Stochastic gradient descent with Nesterov momentum is used to train for a total of 200 epochs. We start with a learning rate of 10^{-1} and decrease by a factor of 5 at epochs 60, 120, and 160. We use the same data augmentation as was used for the CIFAR-10 models.

Results for experiments run on CIFAR-100 are given in Table 2. In this case, we can see that our method performs comparably to dropout when both techniques are used in conjunction with batch normalisation. The combination of all three regularisation schemes performs the best.

Table 2: Accuracy of a Wide Residual Network with a depth of 16 and a width factor of four trained on CIFAR-100 with different regularisation techniques.

Regulariser	Accuracy
Batchnorm	75.34%
Batchnorm + Dropout	75.72%
MaxGain + Batchnorm	75.89%
MaxGain + Batchnorm + Dropout	76.44%

5.3 Street View House Numbers (SVHN)

The Street View House Numbers Dataset contains over 600,000 colour images depicting house numbers extracted from Google street view photos. Each image is 32×32 pixels, and the dataset has a predefined train and test split of 604,388 and 26,032 images, respectively. The distributions of the training and test splits are slightly different, in that the majority of the training images are considered less difficult. We train a VGG-style network on this dataset using the Adam [10] optimiser. Likely due to the large size of the dataset, we found that the network only needed to be trained for 17 epochs. We began with a learning rate of 10^{-4} and reduced it by a factor of 10 for the last two epochs.

Table 3 shows how the different models we considered performed on SVHN. An interesting result here is, in isolation, dropout outperforms both MaxGain and batch normalisation in terms of accuracy improvement over the baseline. This is potentially due to the mismatch between the distributions of the training and testing datasets. Despite the lackluster performance of these methods in isolation, they do still provide a benefit when combined with each other and dropout, which is consistent with the results of our other experiments.

5.4 Scaled ImageNet Subset (SINS-10)

Many datasets used by the deep learning community consist of a single predefined training and test split. For example, in the previous experiments on CIFAR-10 we stated that a set of 50,000 images was used for training, and another set of 10,000 images was used for testing. In order to perform some sort of significance test, and thus have some degree of confidence in our results and the conclusions we draw from them, we must gather multiple measurements of how well models trained using a particular algorithm configuration perform. To this end, we propose the Scaled ImageNet Subset (SINS-10) dataset, a set of 100,000 colour images

Table 3: Accuracy of a VGG-style network on the SVHN dataset when trained with various regularisation techniques.

Regulariser	Accuracy
None	96.99%
Dropout	97.72%
Batchnorm	96.97%
Batchnorm + Dropout	97.86%
MaxGain	97.22%
MaxGain + Dropout	97.89%
MaxGain + Batchnorm	97.31%
MaxGain + Batchnorm + Dropout	97.98%

retrieved from the ImageNet collection [3]. The images are evenly divided into 10 different classes, and each of these classes is associated with multiple synsets from the ImageNet database. All images were first resized such that their smallest dimension was 96 pixels and their aspect ratio was maintained. Then, the central 96×96 pixel subwindow of the image was extracted to be used as the final instance. The labelled images are available online.[3]

An important difference between the proposed dataset and currently available benchmark datasets is how it has been split into training and testing data. The entire dataset is divided into 10 equal sized predefined folds of 10,000 instances. The first 9,000 images in each fold are intended for training a model, and the remaining 1,000 for testing it. One can then apply a machine learning technique to each fold in the dataset, and repeat the process for techniques one wishes to compare against. This will result in 10 performance measurements for each algorithm. A paired t-test can then be used to determine whether there is a significant difference, with some level of confidence, between the performance of the different techniques.

Note that the protocol for SINS-10 is different to the commonly used cross-validation technique. When performing cross-validation, the training sets overlap significantly, and the measurements for the test fold performance are therefore not independent. To mitigate this, one can use a heuristic for correcting the paired t-test [14]. Rather than use this heuristic, we simply avoid fitting models using overlapping training (or test) sets, and can therefore use the standard paired t-test.

We train a Wide Residual Network with a width factor of four on this dataset. No data augmentation was used and each model was trained for 90 epochs using stochastic gradient descent with Nesterov momentum. The learning rate was started at 10^{-1} and decreased by a factor of five at epochs 60 and 80. For each regularisation scheme, we trained a model on each fold of the dataset. Regularisation hyperparameters, such as γ and the dropout rate, were determined on a per-fold basis using a validation set of 1,000 instances drawn from the training set of the fold under consideration.

[3] `https://www.cs.waikato.ac.nz/~ml/sins10/`

Table 4: Performance of the Wide Residual Network on the Scaled ImageNet Subset dataset using various combinations of regularisation techniques. The figures in this table are the mean accuracy ± the standard error, as measured across the 10 different folds.

Regulariser	Accuracy
Batchnorm	70.13% (±0.27)
Batchnorm + Dropout	74.81% (±0.49)
MaxGain + Batchnorm	70.65% (±0.54)
MaxGain + Batchnorm + Dropout	74.80% (±0.51)

Results for the different regularisation schemes trained on this dataset are given in Table 4. We report the mean accuracy across each of the 10 folds, as well as the standard error. Paired t-tests were performed for comparing Batchnorm to MaxGain + Batchnorm, and also for Batchnorm + Dropout versus MaxGain + Batchnorm + Dropout. Neither of the tests resulted in a statistically significant difference ($p = 0.332$ and $p = 0.976$, respectively).

5.5 Gain on the Test Set

Due to the stochastic nature of the projection function, the technique used to constrain the gain on the training set is only approximate. Therefore, it is important that we verify whether the constraint is fulfilled in practice. Moreover, even if the constraint is satisfied on the training set, that does not necessarily mean it will be satisfied on data not seen during training. To investigate this, we supply plots in Figure 1 showing the distribution of gains in each layer in the VGG-19 network trained using MaxGain on the CIFAR-10 dataset. We can see that the distributions between the train and test sets are virtually identical, and are never significantly above 2—the value selected for γ when training this network.

In addition to demonstrating that the stochastic projection function does effectively limit the maximum gain on the test set, we find it interesting to visualise gain measurements taken from each layer in a network trained without the MaxGain regulariser. This visualisation is given in Figure 2. Once again, the distributions of gains measured on the training versus test data are almost identical. Comparing the distributions given in Figure 2 with those provided in Figure 1 show that the MaxGain regulariser has a substantial effect on the activation magnitudes produced by each layer.

If there is no constraint on the magnitude of the weights, then once the network can almost perfectly classify the training data, the optimiser can easily decrease the log loss by making the weights bigger. This results in an "exploding activation" effect, similar to the exploding/vanishing gradient phenomenon, which is only curbed when the cost of the small number of instances in the training set that are very confidently classified incorrectly begin to outweigh the increase in confidence on the correct classifications. Because MaxGain con-

strains the weight sizes of each layer, those that would have had large weights no longer do, and those that would have had small weights will now need larger weights in order to increase the confidence of the model. This results in the far more uniform changes in activation magnitude in Figure 1 compared to those in Figure 2.

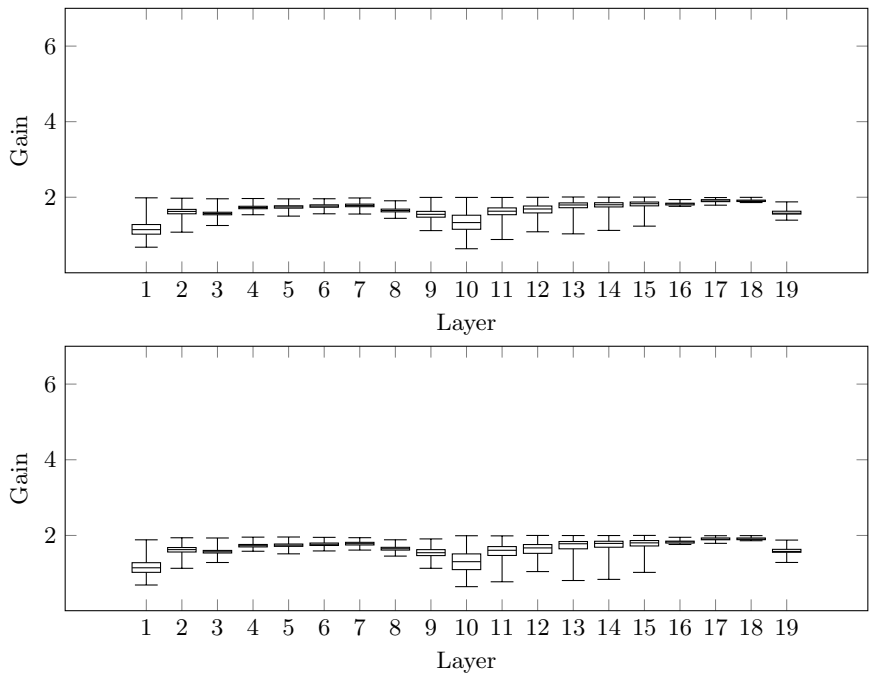

Fig. 1: Boxplots showing the distributions of gains measured on each layer of the MaxGain-regularised VGG-19 network trained on CIFAR-10. The top plot shows the distributions on the training set, and the bottom plot on the test set.

5.6 Sensitivity to γ

The single hyperparameter, γ, that is used to control the capacity of MaxGain-regularised networks should behave similarly to the λ hyperparameter proposed by Gouk et al. [7] which is used to precisely bound the Lipschitz constant. In particular, when γ is set to a small value the model should underfit, and when it is set to a large value one should observe overfitting. We explore this empirically in the context of the VGG-style network trained on SVHN. Figure 3 shows how the performance on the training and test sets of SVHN varies as γ is changed. This plot shows that γ behaves in much the same way as the previously mentioned λ hyperparameter. Specifically, for very low values of γ, the network exhibits low

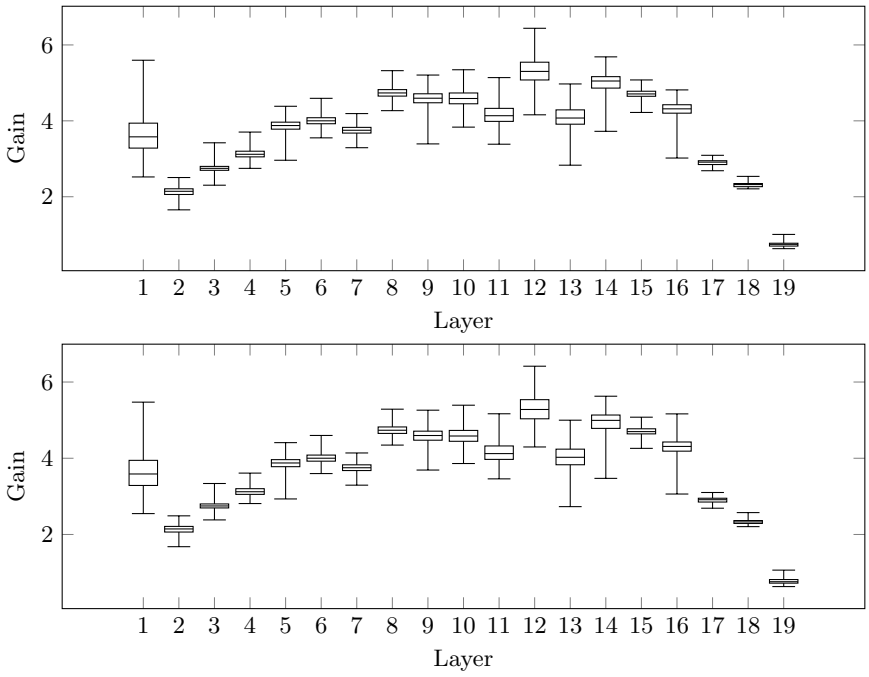

Fig. 2: Boxplots showing the distributions of gains measured on each layer of the unregularised VGG-19 network trained on CIFAR-10. The top plot shows the distributions on the training set, and the bottom plot on the test set.

accuracy and high loss for both the train and test splits of the dataset. As the value of γ is increased, the training accuracy goes towards 100% and the loss goes towards zero. The test accuracy peaks and then plateaus, however the loss on the training set continues to increase, indicating that the network is more confidently misclassifying instances rather than misclassifying more instances.

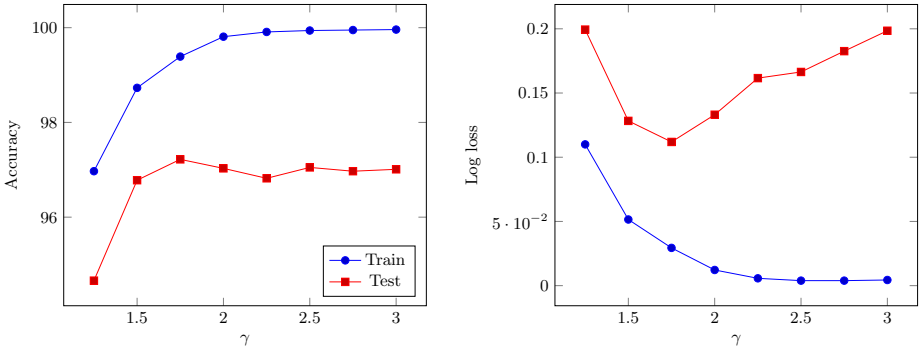

Fig. 3: Accuracy (left) and log loss (right) of the VGG-style model on both the train and test splits of the SVHN dataset as the γ hyperparameter is varied. The legend is shared between both plots.

6 Conclusion

This paper introduced MaxGain, a method for regularising neural networks by constraining how the magnitudes of activation vectors can vary across layers. It was shown how this method can be seen as an approximation to constraining the Lipschitz constant of a network, with the advantage of being usable for any vector norm. The technique is conceptually simple and easy to implement efficiently, thus making it a very practical approach to controlling the capacity of neural networks. We have shown that MaxGain performs competitively with other common regularisation schemes, such as batch normalisation and dropout, when compared in isolation. It was also demonstrated that when these techniques are combined together, further performance gains can be achieved. We envision that people will be most interested in using our method in conjunction with other regularisation schemes, as all of the methods used in our experiments add little runtime overhead to the training process.

Some of the results presented in this paper were obtained using a novel dataset with predefined folds that allows for practical significance testing in experiments involving convolutional networks. We hope that this moderately sized dataset will enable more confident conclusions to be drawn from future experiments.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
2. Cayton, L.: Algorithms for manifold learning. University of California at San Diego Technical Report **12**(1-17), 1 (2005)
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 248–255. IEEE (2009)
4. Dinh, L., Pascanu, R., Bengio, S., Bengio, Y.: Sharp minima can generalize for deep nets. In: International Conference on Machine Learning. pp. 1019–1028 (2017)
5. Gal, Y., Hron, J., Kendall, A.: Concrete dropout. In: Advances in Neural Information Processing Systems. pp. 3584–3593 (2017)
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems. pp. 2672–2680 (2014)
7. Gouk, H., Frank, E., Pfahringer, B., Cree, M.: Regularisation of neural networks by enforcing Lipschitz continuity. arXiv preprint arXiv:1804.04368 (2018)
8. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of Wasserstein gans. In: Advances in Neural Information Processing Systems. pp. 5769–5779 (2017)
9. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. pp. 448–456 (2015)
10. Kingma, D.P., Ba, J.L.: Adam: Amethod for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (2015)
11. Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. In: Advances in Neural Information Processing Systems. pp. 2575–2583 (2015)
12. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, University of Toronto (2009)
13. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. In: 6th International Conference on Learning Representations (2018)
14. Nadeau, C., Bengio, Y.: Inference for the generalization error. In: Advances in Neural Information Processing Systems. pp. 307–313 (2000)
15. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems. pp. 901–909 (2016)
16. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
17. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
18. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: Proceedings of the British Machine Vision Conference (BMVC) (September 2016)
19. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: Proceedings of the 5th International Conference on Learning Representations (2017)