Exact Spectral Norm Regularization for Neural Networks

Anton Johansson [1] **Niklas Engsner** [1] **Claes Strannegård** [1] **Petter Mostad** [1]

Abstract

We pursue a line of research that seeks to regularize the spectral norm of the Jacobian of the input-output mapping for deep neural networks. While previous work rely on upper bounding techniques, we provide a scheme that targets the exact spectral norm. We showcase that our algorithm achieves an improved generalization performance compared to previous spectral regularization techniques while simultaneously maintaining a strong safeguard against natural and adversarial noise. Moreover, we further explore some previous reasoning concerning the strong adversarial protection that Jacobian regularization provides and show that it can be misleading.

1. Introduction

Ensuring that deep neural networks generalize can often be a question of applying the right regularization scheme. While long-established regularization schemes such as weight decay (Krogh & Hertz, 1991) can reduce the function complexity and prevent the network from overfitting, it can at times do so in a crude manner, reducing the complexity more than what is needed and inhibiting the overall performance of the network. Another important consideration for real-world generalizability that many regularization schemes fail to account for is robustness. Robustness will aid in ensuring that the model behaves as expected even when the input is perturbed, e.g., by natural or adversarial noise specifically crafted to fool a given model. With certain adversarial attack methods bridging the gap between theoretical concern and practical considerations by fooling commercial road signs detector with adversarial attacks (Morgulis et al., 2019; Chen et al., 2018), robustness is becoming a progressively more important aspect of model deployment.

Previous work has demonstrated that regularizing the l_p-norms of the Jacobian of the network mapping can meet these two goals concurrently and different techniques have thus been developed to target these quantities (Sokolic et al., 2017). Although obtaining the Jacobian is theoretically straightforward, it is computationally expensive and thus most schemes only seek to approximate a given norm. For example, the Frobenius norm has been approximated through sampling schemes and layer-wise approximations (Hoffman et al., 2019; Gu & Rigazio, 2015) while the spectral norm has been targeted by upper-bounding the spectral norm of each weight matrix in the network (Yoshida & Miyato, 2017; Sokolic et al., 2017).

In this work we extend on the schemes that target the spectral norm. While penalizing an upper-bound of the spectral norm does improve generalization and robustness, it is also crude in the sense that it does not directly target the quantity of interest and might thus inhibit the performance more than necessary. We instead provide an efficient algorithm that targets the *exact* spectral norm of the Jacobian. Using this algorithm we demonstrate that targeting the exact spectral norm can yield an improved generalization performance while preserving a healthy defence against natural and adversarial perturbations.

2. Background

We follow (Yoshida & Miyato, 2017) and represent an L-layer neural network $f : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}$ recursively as $x^l = f^l(G^l(x^{l-1}) + b^l)$, $l = 1, 2, ..., L$ where G^l is either a linear operator (e.g., convolution) or a piecewise linear operator (e.g., max-pool), f^l the corresponding activation function, $b_l \in \mathbb{R}^{n_l}$ is the associated bias for layer l and we set the input $x = x^0$. Denoting the collection of all parameters of the network as θ, and making the dependence of the network on the parameters explicit as f_θ, the full network function will be given as $f_\theta(x) = x^L$.

Momentarily restricting ourselves to the classification setting, the task that we are interested in is then the supervised learning problem of finding parameters θ such that f_θ can associate feature-values $x \in \mathbb{R}^{n_{in}}$ with one-hot encoded labels $y \in \mathbb{R}^{n_{out}}$ obtained from an unknown distribution P. This is achieved by collecting a training set $\mathcal{D}_t := \{(x_i, y_i)\}_{i=1}^N$ where $(x_i, y_i) \sim P$ and employing an appropriate loss function $l : \mathbb{R}^{n_{out}} \times \mathbb{R}^{n_{out}} \to \mathbb{R}$ which encourages f_θ to model a probability distribution for the possible labels for a given feature-value. Minimizing the full loss $l_{bare}(\theta, \mathcal{D}_t) := 1/|\mathcal{D}_t| \sum_{(x_i,y_i)\in\mathcal{D}_t} l(f_\theta(x_i), y_i)$ will thus align the distribution of $f_\theta(x_i)$ with that of the ground-

[1]Chalmers University of Technology. Correspondence to erikantonjohansson@gmail.com

truth label y_i. The minimization is done through some variant of stochastic gradient descent (SGD) where we split $\mathcal{D}_t$ into smaller disjoint random batches $\bigcup_i \mathcal{B}_i = \mathcal{D}_t$ and subsequently minimize $l_{bare}(\theta, \mathcal{D}_t)$ by reducing the partial loss $l_{\text{bare}}(\theta, \mathcal{B}_i)$ for every batch $\mathcal{B}_i$, whereupon the training set is split into new batches and the process repeated. We additionally utilize a validation set $\mathcal{D}_v := \{(x_i, y_i)\}_{i=1}^M$ with $(x_i, y_i) \sim P$ and $\mathcal{D}_t \cap \mathcal{D}_v = \emptyset$ to measure the performance of the model.

2.1. Regularization

Although the sole minimization of $l_{\text{bare}}(\theta, \mathcal{D}_t)$ can yield networks that perform adequately, the networks are often lacking in different regards such as generalization and robustness. While there exists a wide variety of methods that attempt to mitigate these deficiencies, for example by controlling the magnitude of the weights as in weight decay, by utilizing knowledge distillation techniques (Arani et al., 2021; Papernot et al., 2016) or by augmenting the training data with adversarially perturbed examples (Madry et al., 2018), here we focus on the regularization techniques obtained by penalizing with some function $h : \mathbb{R} \to \mathbb{R}$ the norm of the Jacobian. This means that we seek to minimize

$$l_{\text{jac}}(\theta, \mathcal{D}_t, \lambda) := l_{\text{bare}}(\theta, \mathcal{D}_t) + \frac{\lambda}{|\mathcal{D}_t|} \sum_{(x_i, y_i) \in \mathcal{D}_t} h\left(\left|\left| \frac{df_\theta(x_i)}{dx} \right|\right| \right), \tag{1}$$

where λ is a hyper-parameter that controls the trade-off between the two terms and with typical choices for h being either $h(x) = x$ or $h(x) = x^2$.

For most norms the regularized loss (1) does not yield itself to any effective optimization schemes, requiring time-consuming operations to obtain the Jacobian for each x_i in every batch $\mathcal{B}_i$. An exception to this is the Frobenius norm where one can obtain estimates either through a double-backpropagation scheme (Drucker & LeCun, 1992) or by using a more efficient sampling scheme (Hoffman et al., 2019) where one samples n_{proj} vectors v^j from the $n_{out} - 1$ dimensional unit sphere $S^{n_{out}-1}$ to approximate the squared Frobenius norm as

$$\left|\left| \frac{df_\theta(x)}{dx} \right|\right|_F^2 = n_{out} \mathbb{E}_{v \sim S^{n_{out}-1}} \left[\left|\left| v \frac{df_\theta(x)}{dx} \right|\right|^2 \right] \approx \frac{n_{out}}{n_{proj}} \sum_{j=1}^{n_{proj}} \left[\frac{d(v^j \cdot x^L)}{dx} \right]^2,$$

and thus minimizes the expression

$$l_{\text{frob}}(\theta, \mathcal{D}_t, \lambda) := l_{\text{bare}}(\theta, \mathcal{D}_t) + \frac{\lambda n_{out}}{|\mathcal{D}_t| n_{proj}} \sum_{(x_i, y_i) \in \mathcal{D}_t} \sum_{j=1}^{n_{proj}} \left[\frac{d(v^j \cdot x_i^L)}{dx} \right]^2. \tag{2}$$

We on the other hand are interested in penalizing the spectral norm of the Jacobian at a point x, defined as

$$\left|\left| \frac{df_\theta(x)}{dx} \right|\right|_2 = \max_{\substack{v \in \mathbb{R}^{n_{in}} \\ ||v||=1}} \left|\left| \frac{df_\theta(x)}{dx} v \right|\right|_2 = \sigma_{max}, \tag{3}$$

where σ_{max} denotes the largest singular value of $df_\theta(x)/dx$. A constraint on (3) implies that we restrict the maximum rate at which f_θ can change as the input x is perturbed, thus promoting robustness of our model. While the spectral norm does not immediately give itself to any viable method, (Yoshida & Miyato, 2017) managed to develop an efficient scheme by restricting themselves to the setting where all activation functions are piecewise linear. Networks with piecewise linear activation functions are themselves piecewise linear functions and the input space can thus be decomposed into a partition $\mathcal{R}$ where for each $R \in \mathcal{R}$ there exists $W_R \in \mathbb{R}^{n_{in} \times n_{out}}$, $b_R \in \mathbb{R}^{n_{out}}$ such that $f_\theta(x) = W_R x + b_R$, $\forall x \in R$ (Hanin & Rolnick, 2019). For these piecewise linear networks, the Jacobian df_θ/dx is constant in each region $R \in \mathcal{R}$ and given by W_R. Calculating the spectral norm of the Jacobian at some input x is thus reduced to calculating the spectral norm of W_R associated with $R \ni x$.

Although the regularization scheme is valid for all piecewise linear activation functions, it is easiest to present for networks with only ReLU (Nair & Hinton, 2010) activation functions and we thus momentarily restrict ourselves to this setting. By restricting ourselves to these networks and by using the fact that all linear and piecewise linear operators G^l can locally be represented as a matrix W^l, one can obtain the identity

$$W_R = W^L Z_R^{L-1} W^{L-1} \cdots W^2 Z_R^1 W^1 \tag{4}$$

where Z_R^i is a diagonal boolean matrix indicating which neurons in layer i that have an output > 0 when passing $x \in R$ through the network. Using this identity, an upper bound for $||W_R||_2$ can be obtained as $||W_R||_2 \leq \prod_l ||W^l||_2$ and subsequently (Yoshida & Miyato, 2017) regularize the spectral norm by bounding the spectral norm of each weight matrix. They thus minimize the expression

$$l_{\text{specUB}}(\theta, \mathcal{D}_t, \lambda) := l_{\text{bare}}(\theta, \mathcal{D}_t) + \lambda \sum_{l=1}^{L} ||W^l||_2^2, \tag{5}$$

and additionally suggest to further effectivize the scheme by using power iteration on the matrices W^l as $v \sim S^{n_l - 1}$, $u \leftarrow W^l v$, $v \leftarrow (W^l)^T u$ to approximate the spectral norm as $||W^l||_2 \approx ||u||_2 / ||v||_2$. While this scheme will penalize the spectral norm of the Jacobian, it only does so through an upper bound, thus potentially inhibiting the performance of the network more than necessary.

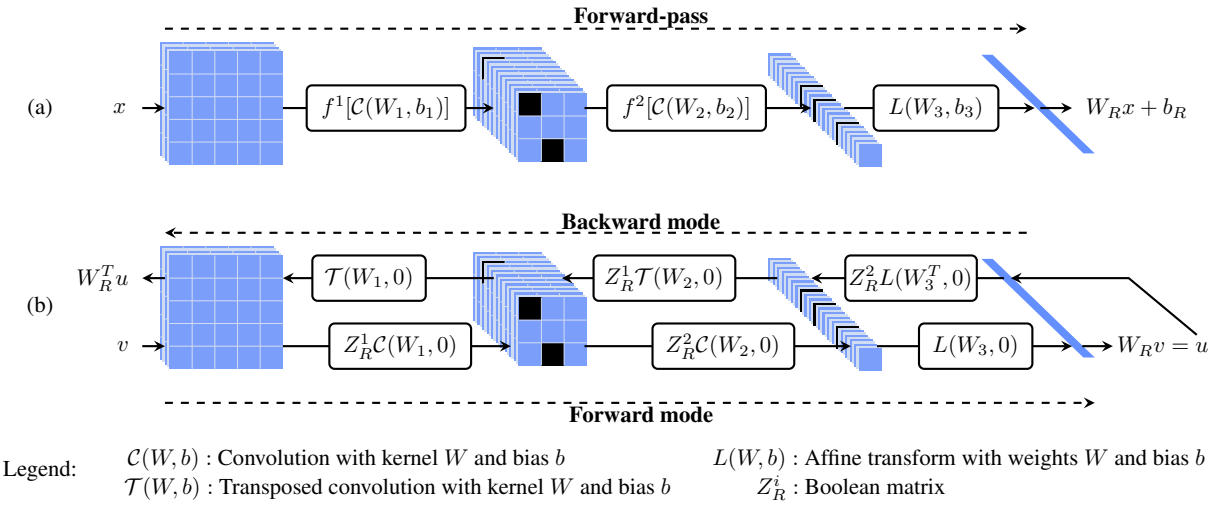

Figure 1. The difference between a regular forward-pass and the forward and backward modes for a two hidden layer network. (a) A regular forward-pass of x through the network. Each box showcases the operation that maps the input between the layers. The black squares indicate the neurons mapped to zero by the ReLU activation functions f^i. (b) The forward and backward modes used to estimate $||W_R||_2$. An input v is sent through the network to yield u whereupon u is sent backwards through the network. Note how each operation is now bias-free with the same weights as during the forward-pass. The activation functions are replaced by multiplication with the Boolean matrices designed to keep the activation pattern fixed, see equation (6) - (7). The backward mode is achieved through transposed convolutions and linear transformations.

3. Method

Here we introduce our method which penalizes the spectral norm of the Jacobian directly. Our scheme relies on power iteration as previous methods but targets $||W_R||_2$ directly. We will follow prior research and momentarily restrict ourselves to piecewise linear networks without skip-connections since this provides a scheme that is easy to present and implement, but keep in mind that the ensuing methodology is valid for networks with skip-connection as well. Additionally, the scheme can be extended efficiently to networks utilizing any non-linear transformation at the cost of a slightly more involved implementation scheme. We detail this extension scheme in Section 3.2.

3.1. Exact spectral norm regularization

To perform power iteration on W_R we need a way to efficiently perform the steps $v \sim S^{n_{in}-1}$, $u \leftarrow W_R v$, $v \leftarrow W_R^T u$ to subsequently approximate the norm as $||W_R||_2 \approx ||u||_2/||v||_2$. Given that the main obstacle for an efficient scheme is the construction of W_R, our scheme circumvents the construction by directly focusing on the matrix-vector products $W_R v$ and $W_R^T u$. Returning to the identity (4), we can see that, given the constituent weight matrices W^l and boolean matrices Z_R^l, one can obtain the desired matrix-vector products as

$$W_R v = W^L Z_R^{L-1} W^{L-1} \cdots W^2 Z_R^1 W^1 v, \tag{6}$$

$$W_R^T u = (W^1)^T Z_R^1 (W^2)^T \cdots (W^{L-1})^T Z_R^{L-1} (W^L)^T u. \tag{7}$$

While the matrices Z_R^l can easily be obtained by recording which neurons that have an output > 0 when passing $x \in R$ through the network, the construction of the matrices W^l is inefficient for most network layers except for the very simplest ones, making the direct application of (6) - (7) impractical.

While the direct application is impractical, we can obtain a practical scheme by interpreting equations (6) - (7) in a particular manner. Equation (6) is nothing other than the forward-pass of v through the network *with all bias vectors set to 0* and *the activation functions replaced with multiplication with boolean matrices* Z_R^l, hereby referred to as the *forward mode* of the network. Similarly, equation (7) is the output obtained by passing u *backwards* through the network, meaning that we start at the final layer and transform u layer by layer with analogous modifications to the bias vectors and activation functions as in the forward mode until we reach the input layer. We will hereby refer to this reverse pass as the *backward mode*[1] of the network. This

[1]Note that the backward mode can be obtained by a standard backward pass to evaluate $d(x^L \cdot u)/dx$. We refer to it here as backward mode to highlight the symmetry with the forward mode which does not have a standard equivalent counterpart.

interpretation circumvents the formation of the matrices W^l and instead relies on forward and backward operators F^l and $(F^L)^T$ that make use of the linear and piecewise linear operators G^l and their corresponding transposed version $(G^l)^T$ that implicitly define W^l and $(W^l)^T$, e.g., through convolution and transposed convolution operators. While for many layers we have that the layer transformations G^l and the resulting forward operators F^l coincide, meaning $F^l = G^l$, there do exist some exceptions to this rule where a little bit of extra care is needed to ensure that the forward and backward modes correctly map to $W_R v$ and $W_R^T u$ respectively, e.g., max-pooling layers where the max-indices of the forward-pass has to be utilized. The reader is referred to the Appendix to see the conversion between the operators G^l, F^l and $(F^l)^T$ for some commonly used layers.

Thus we can target the exact spectral norm of W_R by performing power iteration with $v \sim S^{n_{in}-1}$ and obtain the matrix-vector products $W_R v$ and $(W_R)^T u$ through the forward and backward mode respectively, thereupon estimating the spectral norm as $||W_R||_2 \approx ||u||_2/||v||_2$[2]. For a visualization of the difference between a regular forward-pass, the forward and backward mode of the network and the involved operators, see Figure 1 where all of this is visualized for a simple three layer convolutional network. The network only utilizes ReLU activation functions so that $f^1 = f^2 =$ReLU and G^1, G^2 are given by convolutional layers while G^3 is a linear layer.

Making the association between R and an input $x, x \in R$, explicit as R_x, we can formulate our exact spectral loss as

$$l_{\text{spec}}(\theta, \mathcal{D}_t, \lambda) := l_{\text{bare}}(\theta, \mathcal{D}_t) + \frac{\lambda}{|\mathcal{D}_t|} \sum_{(x_i, y_i) \in \mathcal{D}_t} ||W_{R_{x_i}}||_2. \tag{8}$$

Further, converting the matrix multiplication with the Boolean matrices Z_R^i to component-wise Hadamard products $\odot$ with vectors z^i, we can formulate the entire scheme on a batch level which can be seen in Algorithm 1.

3.2. Extension to non-piecewise linear transforms

While the scheme detailed in Algorithm 1 is capable of regularizing the spectral norm of the Jacobian, it is easiest to implement and most efficient in the piecewise linear setting where all layer-wise transformations are given by piecewise linear functions. Although this is a restriction, many well performing networks rely solely on non-linearities given by piecewise linear activation functions with the addition of batch-normalization layers, see for example VGG (Simonyan & Zisserman, 2015) and ResNet (He et al., 2016) among others. Creating an easily implementable regularization scheme for this well-performing setting thus only requires us to additionally ensure the validity of the scheme when using batch-normalization.

[2]It is possible to perform power iteration multiple times to get a better estimate but we found that performing it once gave sufficiently accurate estimates.

Algorithm 1 Spectral norm regularization

Input: Mini-batch $\mathcal{B}_i$ of feature-value pairs (x, y), weight factor λ, number of power iterations N
Output: Approximate gradient $\nabla_\theta l_{spec}(\theta, \mathcal{B}_i, \lambda)$
$x^0 = x$ {Forward-pass start}
for $l = 1$ **to** L **do**
  $x^l = f^l(G^l(x^{l-1}) + b^l)$
  if $l < L$ **then**
    $z^l = \mathbb{I}\{x^l > 0\}$
  end if
end for
$v \sim \mathcal{N}(0, I)$ {v is of shape $(|\mathcal{B}_i|, n_{in})$}
$v = v/||v||_2$ {Normalize rows}
for $n = 1$ **to** N **do**
  {Forward-mode start}
  for $l = 1$ **to** L **do**
    $v = F^l(v)$
    if $l < L$ **then**
      $v = v \odot z^l$
    end if
  end for
  $u = v$
  $u = u/||u||_2$ {Normalize rows}
  {Backward-mode start}
  for $l = L$ **to** 1 **do**
    $u = (F^l)^T(u)$
    if $l > 1$ **then**
      $u = u \odot z^{l-1}$
    end if
  end for
end for
$\sum_{(x_i, y_i) \in \mathcal{B}_i} ||W_{R_{x_i}}||_2 = \text{sum}(||u||_2/||v||_2)$
$R_{spec}(\theta) = \sum_{(x_i, y_i) \in \mathcal{B}_i} ||W_{R_{x_i}}||_2$
$\nabla_\theta l_{spec}(\theta, \mathcal{B}_i, \lambda) = \nabla_\theta l_{\text{bare}}(\theta, \mathcal{B}_i) + \nabla_\theta \frac{\lambda}{|\mathcal{B}_i|} R_{spec}(\theta)$

Batch-normalization poses two issues which complicates the extension of the regularization scheme.

1. Division by the variance of the input makes batch-normalization a non-piecewise linear transformation during training.

2. Since the mean and variance are calculated per batch, batch-normalization induces a relation between input x_j and output $f_\theta(x_i)$ where $x_i, x_j \in \mathcal{B}, i \neq j$. This induced relation adds multiple components $df_\theta(x_i)/dx_j$ to the Jacobian which represents how an input x_j affects an output $f_\theta(x_i)$. We believe these components

are not relevant in practice and effort should thus not be spent controlling them.

Since both of these issues are only present during training, we circumvent them by penalizing the spectral norm of the Jacobian obtained by momentarily engaging a pseudo-inference mode where we set the running mean and variance of the batch-normalization layers to be fixed and given by the variance and mean obtained from the batch.

Additionally, Algorithm 1 can be efficiently extended to networks employing non-piecewise linear transformations as well at the cost of a more complicated implementation scheme. While not explicitly stated, Algorithm 1 can be used for non-piecewise linear transformations, replacing the Boolean matrices Z^l with matrices given by df^l/dx^{l-1}. However, the calculations and storage of these matrices is likely to be cumbersome and memory intensive for most naive implementations and networks and we thus recommend that one instead utilizes the internal computational graph present in most deep learning libraries. Calculating equation (7) is equivalent to calculating $(df/dx)^T u$ and can thus be obtained by simply applying back-propagation to $d(x^L \cdot u)/dx$ which is a valid scheme for all networks, not only piecewise linear ones. Similarly we can obtain the matrix-vector product $(df/dx)v$ by utilizing the same computational graph used to obtain $d(x^L \cdot u)/dx$, but reverse the direction of all relevant constituent edges and adding a fictitious node to represent the inner product with v, see Figure 2 for a demonstration of this fact for a simple computational graph. We relegate the proof of this to the Appendix.

We choose not to focus on this possible implementation further though since the piecewise linear setting already encompasses a large amount of models and we believe that most will find the scheme in Algorithm 1 more straightforward to implement than delving deep into the mechanics of computational graphs. Further adding on to this fact is that the internals of the computational graphs of popular deep learning frameworks (such as PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2015)) are written in C++ and having to perform modifications of the graph would thus potentially impede the Python-based workflow which many practitioners operate with. However, if one wishes to utilize spectral regularization for networks that employ non-piecewise linear activation functions, for example sigmoids which can be of relevance for attention mechanisms (Vaswani et al., 2017), then the extension scheme provides a well-principled and efficient approach that one can follow. In that case one would replace the Forward-mode and Backward-mode in Algorithm 1 with the computational graph manipulation techniques to obtain $(df/dx)v$ and $(df/dx)^T u$ respectively.

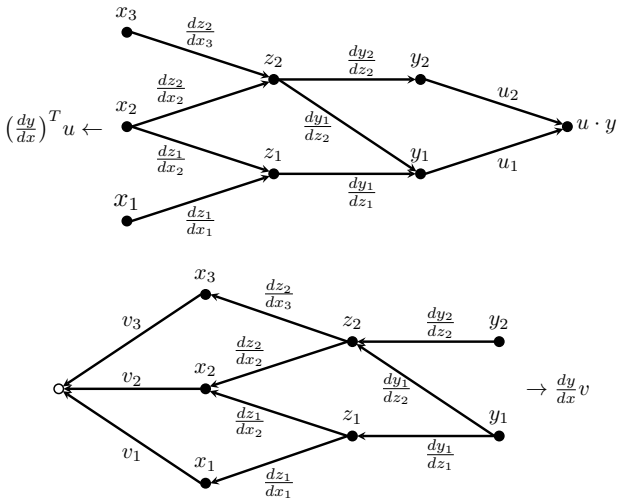

Figure 2. Illustration of the extension scheme. (Top) The computational graph associated with the forward-pass. Each node can perform any non-linear transformation of the associated input. To obtain the i:th component of $(dy/dx)^T u$ we sum the product of the edge elements along every path from the right-most node to x_i. (Bottom) The modified computational graph to obtain $(dy/dx)v$. The direction of all edges are flipped, we remove the right-most node and we add a fictitious node to the computational graph (marked as a empty circle) with connecting edge elements being given by components of v. All other relevant edge elements are preserved from the top graph. The i:th component of $(dy/dx)v$ can then be obtained by starting at the fictitious node and summing the product of the edge elements along every path to y_i

4. Experiments

In this section we evaluate how targeting the exact spectral norm, hereby referred to as the Spectral method, compares to other regularization methods, namely the Frobenius method of (2), the Spectral-Bound method of (5) and weight decay (Krogh & Hertz, 1991) (also referred to as L2-regularization). We compare the generalization performance across different data sets and investigate the robustness of the obtained networks.

4.1. Generalization

The considered data sets where the generalization performance is measured are KMNIST (Clanuwat et al., 2018), FashionMNIST (which at times we will abbreviate as FMNIST) (Xiao et al., 2017) and CIFAR10 (Krizhevsky, 2009). The generalization performance is measured by measuring the accuracy on the corresponding validation set $\mathcal{D}_v$ for each data set. A variant of the LeNet architecture (Lecun et al., 1998) is used for the KMNIST and FMNIST dataset while the VGG16 (Simonyan & Zisserman, 2015) architecture is used for CIFAR10. All three data sets are preprocessed so that they have channelwise mean of 0 and a standard deviation of 1. We perform a grid-search to find the opti-

mal hyperparameters for each network and regularization scheme, see the Appendix for more details regarding the training setup. Each experiment is repeated five times and the model that has the lowest mean loss over all hyperparameters over these five runs is chosen as the representative of a given method. The results of this experiment can be seen in Table 1.

Table 1. Mean test accuracy $\pm$ one standard deviation for the different regularization methods on three data sets computed over 5 runs. Bold indicates best mean accuracy. The method names have been shortened to make the table more compact.

METHOD	CIFAR10	KMNIST	FMNIST
SPEC	90.20 ± 0.61	$\mathbf{96.61 \pm 0.14}$	$\mathbf{91.10 \pm 0.06}$
FROB	$\mathbf{90.21 \pm 0.69}$	96.39 ± 0.08	91.00 ± 0.16
SPEC-B	89.37 ± 0.70	95.72 ± 0.21	90.66 ± 0.26
L2	89.94 ± 0.76	95.58 ± 0.05	90.64 ± 0.30
NONE	88.59 ± 0.67	94.36 ± 0.26	90.35 ± 0.28

From these results we can see that penalizing the exact spectral norm on KMNIST and FMNIST does result in models with higher accuracies than those obtained from models with other regularization schemes, and for CIFAR10 it results in the second best model when considering the mean accuracy. For KMNIST and FMNIST we can additionally see that the Spectral method is significantly better than the Spectral-Bound method, demonstrating that targeting the exact spectral norm yields an improved generalization performance compared to working with an upper bound.

4.2. Robustness

While generalization on a validation or test set gives an indication of model performance in practice, data encountered in reality is often not as exemplary as a curated benchmark data set and ensuring robustness against both natural and adversarial noise can often be a precondition for model deployment.

As previously mentioned, earlier research has indicated that controlling the norm of the Jacobian is beneficial for robustness of our networks and we thus follow the path of (Hoffman et al., 2019) and investigate how the robustness of the different schemes compare.

4.2.1. ROBUSTNESS AGAINST WHITE NOISE

We measure the robustness against white noise by creating a noisy validation set $\mathcal{D}_{v,\sigma^2}$ for FashionMNIST and KMNIST, consisting of data points $\tilde{x}$ obtained by adding independent Gaussian distributed noise to each individual pixel of validation points $x \in \mathcal{D}_v$ as

$$\tilde{x}_{ij} = x_{ij} + \epsilon, \;\; \epsilon \sim \mathcal{N}(0, \sigma^2) \tag{9}$$

whereupon we clip the value of all pixels into the range [0,1] and perform the aforementioned pre-processing. Further, to enable a fair comparison between the different methods and to not have the result obscured by the initial baseline accuracies, we measure the difference between the baseline accuracy on $\mathcal{D}_v$ and the accuracy on $\mathcal{D}_{v,\sigma^2}$. These results can be seen to the left in Figure 3 where we see that there is not a large difference in robustness between either of the training schemes.

4.2.2. ROBUSTNESS AGAINST ADVERSARIAL NOISE

The last decade has seen an increased growth in the amount of research into adversarial noise, noise that may be imperceptible to the human eye but which has a considerable impact on the prediction of a deep learning model. Here we will work with the adversarial noise technique known as projected gradient descent (PGD) method (Madry et al., 2018) and related variants. PGD obtains the perturbation $\tilde{x}$ through a constrained gradient ascent, moving in a direction which increases the loss $l_{bare}(\theta, \{(x, y)\})$ while simultaneously restricting the ascent to the ball $B_\delta = \{z \in \mathbb{R}^{n_0} : ||x - z||_\infty \leq \delta\}$ which ensures that the perturbation $\tilde{x}$ is visually similar to x. Formally, PGD obtains $\tilde{x}$ as

$$\tilde{x} = \text{Proj}_{B_\delta}\left[x + \eta \text{sign}\left(\frac{dl_{bare}(\theta, \{(x, y)\})}{dx}\right)\right], \tag{10}$$

where Proj denotes the projection operator and η the stepsize for the gradient ascent. The gradient ascent process can be repeated over several iterations to yield perturbations $\tilde{x}$ indistinguishable from x but for which the network predicts an incorrect label. After the ascent procedure we clip the pixel values into the range [0,1] and perform the pre-processing as before. We will additionally consider the adversarial attack methods TPGD (Zhang et al., 2019) that performs PGD on a Kullback-Leibler divergence of the softmax-scores, and the gradient-free attack Square (Andriushchenko et al., 2020). All attacks are implemented through the torchattacks library (Kim, 2020) with default parameters except for the parameters δ, η which we set to be 32/255 and 2/255 respectively.

To control the strength of the adversarial noise we vary the number of iterations for each attack. As before, to enable a fair comparison we compute the difference between the baseline accuracy on FashionMNIST and KMNIST with the adversarially perturbed validation sets. These results can be seen in Figure 3 where we see that the Spectral regularization scheme is able to consistently ensure stronger robustness for the PGD and TPGD attacks on KMNIST compared to other methods, but that the full results indicate

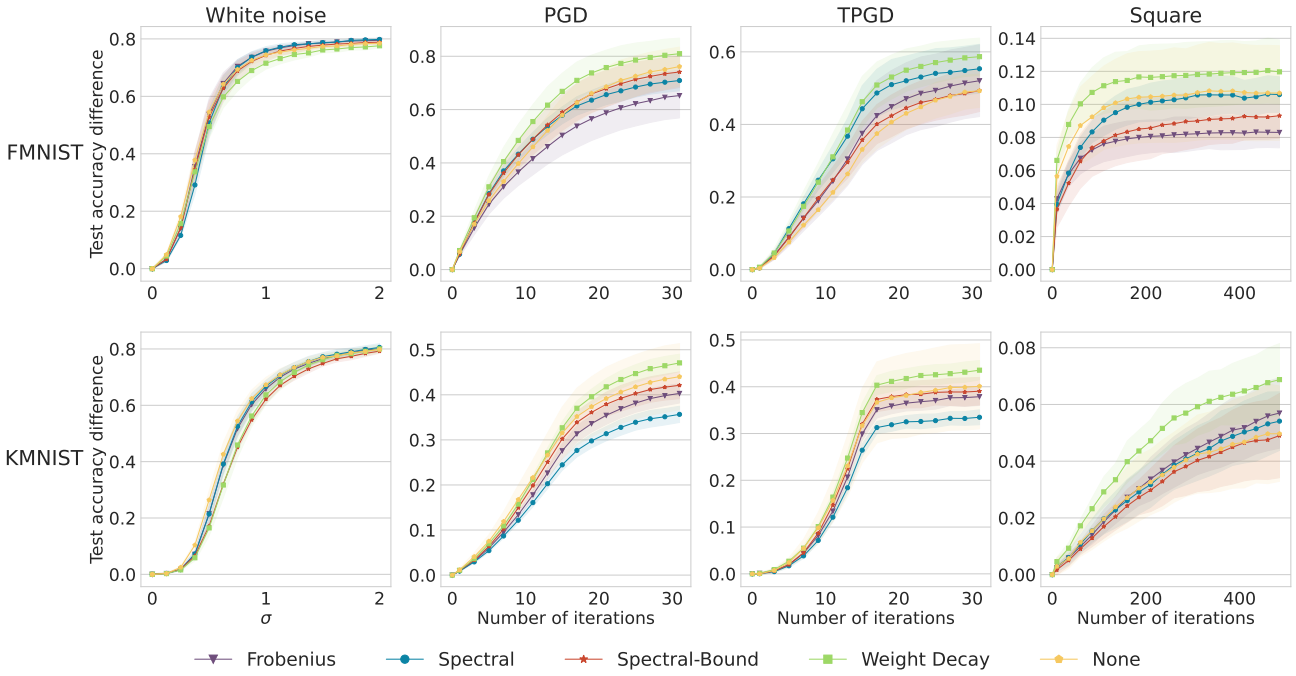

Figure 3. Robustness against perturbations. Each plot displays how the test accuracy drops as the perturbations gets stronger. Each column corresponds to a perturbation method and each row is associated with a given data set. The curves and intervals are obtained as the mean and standard deviation over 5 different networks.

that no regularization algorithm is clearly dominant in all settings. Each regularization method (except weight decay) has some data set and attack where it outperforms the other methods in terms of mean test accuracy difference. Some form of Jacobian regularization is thus beneficial, but the exact penalization method to achieve an optimal robustness against adversarial noise is likely situation dependant.

4.2.3. Distance to decision boundary

One way to attempt to understand the robustness results is to analyze the distance to the closest decision boundary. Previous research demonstrated that controlling the Frobenius norm enlarged the decision cells and thus argued that this made the network more robust to perturbations (Hoffman et al., 2019). We extend their experiments and perform an extensive investigation to measure the robustness where we measure the distance to the decision boundary for all validation points in FashionMNIST and KMNIST. To measure the distance to the decision boundary for a given point we sample points uniformly on concentric spheres of different radii and perform a binary search to find the smallest radii such that a sampled point obtains a predicted class different from the validation point at the center of the sphere. These results are summarized in Figure 4.

From these results we see that penalizing the spectral and Frobenius norm of the Jacobian yields regions which are larger than the other methods on average for FashionMNIST while all regularization methods achieve similarly sized regions for KMNIST. That some methods obtain similarly sized regions yet provide a varying level of safeguard against adversarial noise as seen in Figure 3 implies that the enlargening of the regions cannot fully capture the nuances of robustness against adversarial attacks. Choosing a model based on this intuition that larger regions provide a stronger safeguard can even yield a subpar model as evident from weight decays poor adversarial safeguard on KMNIST despite its large regions.

We hypothesize that one aspect of robustness that this intuition fails to take into account is the structure of the loss landscape. Since a smooth loss landscape with large gradients will facilitate the creation of adversarial examples through gradient ascent we must also consider this aspect to get a holistic view of a regularization methods robustness.

4.3. Time efficiency and relative error

In the previous section we have detailed the generalization and robustness results when using the different regularization schemes. In this section we return to our initial task of investigating the difference between targeting the exact spectral norm of the Jacobian compared to working with

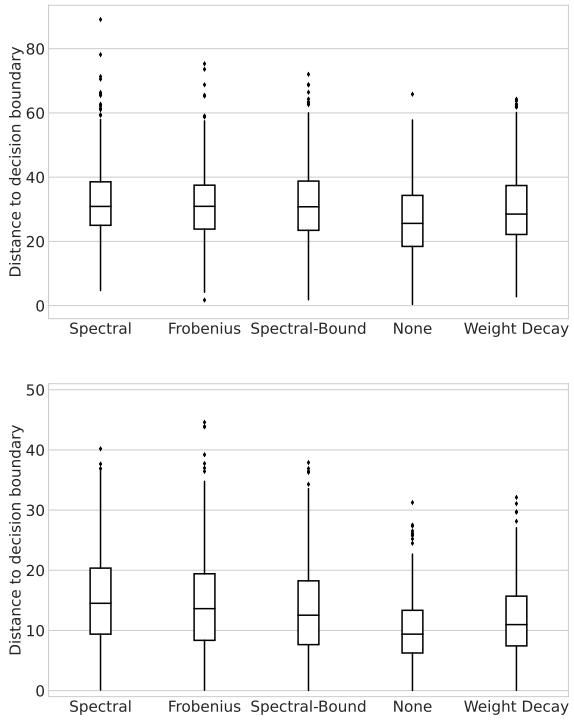

Figure 4. Distance to the decision boundary. (Top) Distance to the nearest boundary for validation points in KMNIST. (Bottom) Distance to the nearest boundary for validation points in Fashion-MNIST. It can be seen that Spectral and Frobenius regularization increases the size of the decision regions on average.

an upper bound. From Table 1 we saw that this yields an improved generalization performance and from Figure 3 we observed that the two methods provide a similar protection against noise, with different strengths against different attacks on the two considered data sets.

While an improved generalization performance is beneficial, it cannot come at a too large of a computational cost. Additionally, with approximate methods it is also important to measure the trade-off between computational speed and accuracy of the approximated quantity. We thus analyze the computational overhead that they add to the training routine and the relative error with the analytical spectral norm.

In Figure 5 (left) we can thus see the average time taken to optimize over a batch for the Spectral method, the Spectral-Bound method, an analytical method that calculates $||W_R||_2$ exactly and a regular forward-pass. In Figure 5 (right) the relative error for the power iteration scheme is visible.

From these plots we can see there is a small extra incurred cost of working with our method compared to regularizing with Spectral-Bound, but that our method has a significantly lower relative error while still being orders of magnitude faster than calculating the analytical spectral norm.

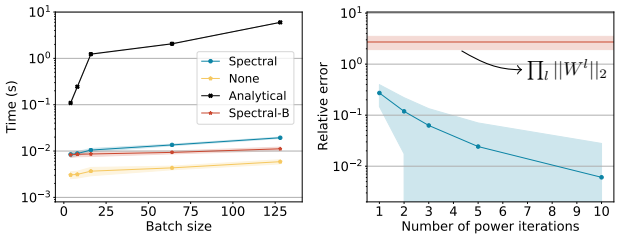

Figure 5. Time and error comparison between the Spectral and Spectral-Bound method for the LeNet network. (Left) Time taken to pass over one batch of data points. The Spectral method is slower than the Spectral-Bound method for larger batch sizes but still around two orders of magnitude faster than calculating the exact spectral norm analytically. (Right) The relative error as the number of power iterations is increased. The relative error decreases quickly and is significantly closer to the exact quantity compared to the upper bound $\prod_l ||W^l||_2$.

5. Conclusion

We have demonstrated a method to improve spectral norm regularization for neural networks. While previous methods relied on inexact upper bounding techniques, our technique targets the exact spectral norm. In the piecewise linear setting our method is easily implemented by performing power iteration through a forward-backward scheme while generally it can be achieved with a slightly more intricate scheme where the underlying computational graphs are modified to perform the power iteration scheme.

This scheme obtained an improved generalization performance while achieving a similar safeguard to natural and adversarial noise as other Jacobian regularization techniques. Further, we investigated the intuition that Jacobian regularization provides a strong defence against adversarial attacks by the enlargening of the decision cells and found that the size of the regions is not necessarily indicative of the robustness of the network.

For future work we are interested in applying our scheme to more complex data sets and tasks, for example Reinforcement learning and Generative Adversarial Networks where controlling the spectral norm has already proven to be beneficial (Gogianu et al., 2021; Miyato et al., 2018). We believe that our scheme can yield additional benefits and will spur further research into accurate and well-principled spectral and Jacobian regularization techniques.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Lev-

enberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Andriushchenko, M., Croce, F., Flammarion, N., and Hein, M. Square attack: A query-efficient black-box adversarial attack via random search. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J. (eds.), *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXIII*, volume 12368 of *Lecture Notes in Computer Science*, pp. 484–501. Springer, 2020. doi: 10.1007/978-3-030-58592-1_29. URL https://doi.org/10.1007/978-3-030-58592-1_29.

Arani, E., Sarfraz, F., and Zonooz, B. Noise as a resource for learning in knowledge distillation. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*, pp. 3128–3137. IEEE, 2021. doi: 10.1109/WACV48630.2021.00317. URL https://doi.org/10.1109/WACV48630.2021.00317.

Chen, S., Cornelius, C., Martin, J., and Chau, D. H. P. Shapeshifter: Robust physical adversarial attack on faster R-CNN object detector. In Berlingerio, M., Bonchi, F., Gärtner, T., Hurley, N., and Ifrim, G. (eds.), *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part I*, volume 11051 of *Lecture Notes in Computer Science*, pp. 52–68. Springer, 2018. doi: 10.1007/978-3-030-10925-7_4. URL https://doi.org/10.1007/978-3-030-10925-7_4.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. Deep learning for classical japanese literature. *CoRR*, abs/1812.01718, 2018. URL http://arxiv.org/abs/1812.01718.

Collins, M. Lecture notes on computational graphs, and backpropagation. URL http://www.cs.columbia.edu/~mcollins/ff2.pdf.

Drucker, H. and LeCun, Y. Improving generalization performance using double backpropagation. *IEEE Trans. Neural Networks*, 3(6):991–997, 1992. doi: 10.1109/72.165600. URL https://doi.org/10.1109/72.165600.

Gogianu, F., Berariu, T., Rosca, M., Clopath, C., Busoniu, L., and Pascanu, R. Spectral normalisation for deep reinforcement learning: An optimisation perspective. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 3734–3744. PMLR, 2021. URL http://proceedings.mlr.press/v139/gogianu21a.html.

Gu, S. and Rigazio, L. Towards deep neural network architectures robust to adversarial examples. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.5068.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 359–368, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/9766527f2b5d3e95d4a733fcfb77bd7e-Abstract.html.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Hoffman, J., Roberts, D. A., and Yaida, S. Robust learning with jacobian regularization. *CoRR*, abs/1908.02729, 2019. URL http://arxiv.org/abs/1908.02729.

Kim, H. Torchattacks: A pytorch repository for adversarial attacks. *arXiv preprint arXiv:2010.01950*, 2020.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Krogh, A. and Hertz, J. A. A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippmann, R. (eds.), *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pp. 950–957. Morgan Kaufmann, 1991.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=B1QRgziT-.

Morgulis, N., Kreines, A., Mendelowitz, S., and Weisglass, Y. Fooling a real car with adversarial traffic signs. *CoRR*, abs/1907.00374, 2019. URL http://arxiv.org/abs/1907.00374.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T. (eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pp. 807–814. Omnipress, 2010. URL https://icml.cc/Conferences/2010/papers/432.pdf.

Papernot, N., McDaniel, P. D., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pp. 582–597. IEEE Computer Society, 2016. doi: 10.1109/SP.2016.41. URL https://doi.org/10.1109/SP.2016.41.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

Sokolic, J., Giryes, R., Sapiro, G., and Rodrigues, M. R. D. Robust large margin deep neural networks. *IEEE Trans. Signal Process.*, 65(16):4265–4280, 2017. doi: 10.1109/TSP.2017.2708039. URL https://doi.org/10.1109/TSP.2017.2708039.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Yoshida, Y. and Miyato, T. Spectral norm regularization for improving the generalizability of deep learning. *CoRR*, abs/1705.10941, 2017. URL http://arxiv.org/abs/1705.10941.

Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. Theoretically principled trade-off between robustness and accuracy. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7472–7482. PMLR, 2019. URL http://proceedings.mlr.press/v97/zhang19p.html.

A. Experimental details

A.1. Network architectures

We will follow (Hoffman et al., 2019) and denote a convolutional-max-pool layer as a tuple $(K, C_{in} \to C_{out}, S, P, M)$ where K is the width of the kernel, C_{in} is the number of in-channels, C_{out} the number of out-channels, S the stride, P the padding of the layer and M the size of the kernel of the max-pool following the convolutional layer. The case $M = 1$ can be seen as a convolutional layer followed by an identity function. Linear layers we will denote as the tuple (N_{in}, N_{out}) where N_{in} is the dimension of the input and N_{out} the size of the output. For KMNIST and FashionMNIST we used the LeNet network which consist of a convolutional-maxpool layer (5, $1 \to 6$, 1, 2, 2), convolutional-maxpool layer (5, $6 \to 16$, 1, 0, 2), linear layer (400, 120), linear layer (120 , 84) and linear layer (84, 10).

We use the VGG16 network as is available from the torchvision package. For this network we use batch-norm layers directly after every convolutional layers. This network consist of the layers (3, $3 \to 64$, 1, 1, 1), (3, $64 \to 64$, 1, 1, 2), (3, $64 \to 128$, 1, 1, 1), (3, $128 \to 128$, 1, 1, 1), (3, $128 \to 256$, 1, 1, 2), (3, $256 \to 256$, 1, 1, 1), (3, $256 \to 256$, 1, 1, 1), (3, $256 \to 512$, 1, 1, 2), (3, $512 \to 512$, 1, 1, 1), (3, $512 \to 512$, 1, 1, 1), (3, $512 \to 512$, 1, 1, 2), (512, 10).

A.2. Training details and code

We train the LeNet networks for 50 epochs with SGD (with momentum=0.8). For every regularization method we perform a hyperparameter search over these three following parameters and values.

- Learning rate: [0.01, 0.001]
- Batch size: [16, 32]
- Weight factor λ: [0.0001, 0.001, 0.01, 0.1]

For the VGG16 network we trained the network for 100 epochs with a batch size of 128, SGD with momentum of 0.8 and performed a hyperparameter search over these parameters and values

- Weight factor λ: [0.00001, 0.0001, 0.001, 0.01, 0.1]

For VGG16 we additionally used a cosine annealing learning rate scheduler with an initial learning rate of 0.1 and the data augmentation techniques of random cropping and horizontal flipping.

For each hyperparameter setting we repeat the training procedure 5 times to be able to obtain mean and standard deviation. We pick the final representative model for each regularization method as the one that achieves the lowest mean validation loss over these 5 training runs.

For the Frobenius regularization we set $n_{proj} = 1$ and for the Spectral-Bound we estimate the spectral norm of the weight matrices through one power iteration.

A.3. Details for figures

Figure 4: The model for each regularization method was chosen randomly among the 5 models from the hyperparameter setting that obtained the best results in Table 1. The distance is only calculated for the points in the validation set that all models predict correctly. In total the distance is predicted for between 8000 - 9000 validation points on FashionMNIST and KMNIST.

Figure 5 (left): The time for a batch was measured on a computer with NVIDIA K80 GPU as available through Google Colab[3]. The analytical method works by sequentially calculating $d(x^L \cdot e_i)/dx$ where e_i is a basis-vector for $\mathbb{R}^{n_{out}}$ for $i = 1, 2, ..., n_{out}$. This yields the full Jacobian matrix which we then calculate the singular values of by using inbuilt functions in PyTorch.

Figure 5 (right): The upper bound was evaluated on a network trained with the Spectral-Bound regularization scheme for all data points in the training set. The curve for the spectral method was evaluated on a network trained with the spectral

[3] colab.research.google.com

method for all data points in the training set. For the spectral method there was no significant difference in the shape of the curve when using a different network or by working with data points in the validation set.

B. Conversion between operators

In this section we detail how to convert between the forward F, backward F^T and regular operators G. These can be seen in Table 2 - 4. Other non-linearities such as Dropout can be incorporated identically to ReLU by simply storing the active neurons in a boolean matrix Z.

B.1. Skip-connections

Utilizing networks with skip-connections does not change the forward and backward modes. Simple turn off the bias of all layer transformations and replace the activation functions with the matrices Z_R^i instead. That this is true follows from the definition of a network with skip-connections. For simplicity of presentation, we will assume that the skip-connections only skip one layer. Assume that we have a network with L layers and additionally have skip-connections between layers with indices in the set $\mathcal{S} := \{s_1, s_2, ..., s_m\},\ 1 \leq s_i \leq L$. Then the network f_θ is given recursively as before with

$$x^l = \begin{cases} f^l(G^l(x^{l-1}) + b^l) & \text{if } l \in \mathcal{S}^C, \\ x^{l-1} + f^l(G^l(x^{l-1}) + b^l) & \text{if } l \in \mathcal{S}. \end{cases}$$

Assuming that we are only working with piecewise linear or linear operators G^l, then for $x \in R$ we know that each operator can be represented as a matrix and we can write the derivative of the two cases as

$$\frac{dx^l}{dx^{l-1}} = \begin{cases} Z^l W^l & \text{if } l \in \mathcal{S}^C, \\ I + Z^l W^l & \text{if } l \in \mathcal{S}, \end{cases}$$

where I denotes a unit-matrix. The Jacobian-vector product $W_R v$ can thus be obtained as

$$W_R v = \left(\prod_{l=1}^{L} (I - \mathbb{I}\{l \in \mathcal{S}^C\} + Z^l W^l) \right) v \tag{11}$$

where $\mathbb{I}\{l \in \mathcal{S}^C\}$ is an indicator for the unit-matrix so that we can concisely write the two cases. Thus we see that we can interpret this equation in the same manner as we did for the networks without skip-connections. We simply pass the input v through the network and turn off all the biases and replace the activation functions with Z^l. The same is true for the backward mode.

Forward-pass (G)	Forward-mode (F)	Backward-mode (F^T)
Input: x, W, b y = Linear(x, W, b) **return:** y	**Input:** x, W y = Linear($x, W, 0$) **return:** y	**Input:** x, W y = Linear($x, W^T, 0$) **return:** y

Table 2. Conversion table for the linear operator.

Forward-pass (G)	Forward-mode (F)	Backward-mode (F^T)
Input: x, W, b y = Conv(x, W, b) **return:** y	**Input:** x, W y = Conv($x, W, 0$) **return:** y	**Input:** x, W y = ConvTranspose($x, W, 0$) **return:** y

Table 3. Conversion table for the convolutional operator.

Forward-pass (G)	Forward-mode (F)	Backward-mode (F^T)
Input: x y, indices = maxpool(x) I = indices S = x.shape **return:** y, I, S	**Input:** x, I y = x[I] **return:** y	**Input:** x, I, S y = maxunpool(x, indices=I, shape = S) **return:** y

Table 4. Conversion table for the max-pool operator.

C. Proof for extension scheme

We will denote the directed acyclic graph which when summing the product of every edge element along every path from output to input yields $(df/dx)^T$ as G.

Theorem: *Consider the graph F obtained by flipping the direction of all edges of G and adding a node at the end of F with edge elements given by components of v. Summing the product of every edge element along every path from output to input of F yields $(df/dx)v$.*

Proof: We will follow the notation of (Collins), Theorem 1 and denote the Jacobian between variables $y = f_\theta(x)$ and x as the sum of the product of all intermediate Jacobians, meaning

$$\frac{dy}{dx} = \sum_{p \in \mathcal{P}(x,y)} \prod_{(a,b) \in p} J^{a \to b}(\alpha^b) \tag{12}$$

where $\mathcal{P}(x, y)$ is the set of all directed paths between x and y and (a, b) is two successive edges on a given path.

In our scheme we flip the direction of all relevant edges and add a fictitious node at the end of the path the flipped paths. Since we preserve the edge elements, we can realize that flipping the direction of the edges simply transposes the local Jacobian, meaning that $J^{b \to a}(\alpha^b) = \left(J^{a \to b}(\alpha^b)\right)^T$ with our scheme. Further, our added fictitious node has edge elements given by elements of v, and the Jacobian between that node and the subsequent layer is thus given by v^T. For a path $p = [(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)]$ we define the flipped path with the added fictitious node as p^T as $p^T = [(v_n, v_{n-1}), ..., (v_2, v_1), (v_1, v_f)]$ and the reverse-order path $\neg p$ as $\neg p = [(v_{n-1}, v_n), ..., (v_2, v_3), (v_1, v_2)]$. For our modified graph we thus have the Jacobian for a path as

$$\prod_{(a,b) \in p^T} J^{a \to b}(\alpha^b) = \prod_{(a,b) \in p^T} \left(J^{b \to a}(\alpha^b)\right)^T \tag{13}$$

$$= v^T \left(\prod_{(a,b) \in p} J^{a \to b}(\alpha^b) \right)^T \tag{14}$$

$$= v^T \left(\prod_{(a,b) \in \neg p} J^{a \to b}(\alpha^b)^T \right) \tag{15}$$

Denoting the fictitious node as n_f and summing over all paths we thus get

$$\sum_{p^T \in \mathcal{P}(y, n_f)} \prod_{(a,b) \in p^t} J^{a \to b}(\alpha^b) \tag{16}$$

$$= \sum_{p^T \in \mathcal{P}(y, n_f)} v^T \left(\prod_{(a,b) \in \neg p} J^{a \to b}(\alpha^b)^T \right) \tag{17}$$

$$= v^T \sum_{p^T \in \mathcal{P}(y, n_f)} \left(\prod_{(a,b) \in \neg p} J^{a \to b}(\alpha^b)^T \right) \tag{18}$$

$$= v^T \left(\frac{dy}{dx}\right)^T = \left(\frac{dy}{dx} v\right)^T \tag{19}$$

which proves that working with the modified graph will yield the desired matrix-vector product $\frac{dy}{dx} v$ $\square$.