NONPARAMETRICALLY LEARNING ACTIVATION FUNCTIONS IN DEEP NEURAL NETS

Carson Eisenach
Princeton University
eisenach@princeton.edu

Han Liu
Princeton University
hanliu@princeton.edu

Zhaoran Wang
Princeton University
zhaoran@princeton.edu

ABSTRACT

We provide a principled framework for nonparametrically learning activation functions in deep neural networks. Currently, state-of-the-art deep networks treat choice of activation function as a hyper-parameter before training. By allowing activation functions to be estimated as part of the training procedure, we expand the class of functions that each node in the network can learn. We also provide a theoretical justification for our choice of nonparametric activation functions and demonstrate that networks with our nonparametric activation functions generalize well. To demonstrate the power of our novel techniques, we test them on image recognition datasets and achieve up to a 15% relative increase in test performance compared to the baseline.

1 INTRODUCTION

Deep learning techniques have proven particularly useful in the classification setting, recently surpassing even human performance on some image-classification tasks. We seek to advance the state of the art by learning not only weights between neurons in the network but also part of the network structure itself – the activation functions. Current deep learning literature largely focuses on improving architectures and adding regularization to the training process.

By contrast, the realm of learning the network structure itself is relatively unexplored. Current best practice often treats network size, shape and choice of activation function as a hyper-parameter to be chosen empirically. Instead, we propose to learn the activation functions through nonparametric estimation. We introduce a class of nonparametric models for activation functions. Importantly, we ensure that our technique can be incorporated into the back-propagation framework. This is crucial because it means our method can be easily added to current practice. To this end, we propose learning functions via basis expansion. In particular, we find that using a Fourier basis works well in practice and offers improved performance over the baseline on several benchmark datasets. We see relative improvements in test error rates of up to 15%. Nonparametric activation functions in dropout nets are especially successful.

We also introduce a two-stage training process. First, a network without nonparametric activation functions is trained. Then, the learned network is used as an initialization for an identical network with nonparametric activation functions. This method of initializing the network yields considerable improvements in performance for convolution neural networks on image classification tasks.

Lastly, we consider the algorithmic stability approach to accessing generalization bounds. We use this to demonstrate that feed-forward networks with our method of nonparametrically estimating activation functions generalize well.

To summarize, our contributions are the following:

- A theoretically justified framework for learning activation functions in a neural network,

- Provable bounds on the generalization error of networks where the activation functions are learned, and
- An optional two-stage training process that can greatly improve results for certain network architectures, specifically those with convolution layers.

2 Related Works

Recent work from Agostinelli et al. (2015) describes piecewise linear activation functions. Theoretically speaking, our approach improves upon this by fully exploring the space of possible activation functions – not merely considering those which are piecewise linear.

Learning activation functions, in conjunction with the weights in the layer transformations, allows for the fitting of an arbitrary function of the output of the previous layer. A similar idea is Network in Network, due to Lin et al. (2013). There the authors propose to replace standard convolution layers with what they call mlpconv layers. Traditional convolution layers, if using a sigmoid activation, are essentially learning a generalized linear model of the previous layer's output. Their innovation is to use a more general nonlinear function estimator as the filter – a multilayer perceptron. Thus, their scheme fits in the back-propagation framework and so is easy to implement. Our approach is different because though we are learning arbitrary functions of the previous layer, our activation function approach can be used in any layer of the network and in networks without convolution layers, which theirs cannot.

Maclaurin et al. (2015) propose to learn the hyper-parameters of a neural net through reverse gradient descent. This is similar to the work we present here in that they also provide a method for learning parameters of the network.

3 Nonparametric Activation Functions

Here we describe our main contribution: the nonparametric activation function model. We also provide justification for a weight tying scheme to use nonparametric activations in conjunction with convolution layers, a regularization which proves powerful empirically (see Section 5).

3.1 Fourier Basis Expansion for Activation Functions

For the nonparametric estimation we use a Fourier series basis expansion. One concern, however, with Fourier basis expansion is that function approximation over a fixed interval often exhibits poor tail behavior. To combat that, we fit the approximation over an interval $[-L, L]$ and then use it over truncation of the interval, $[-L+T, L+T]$. The designation NPF stands for *Nonparametric Fourier Basis Expansion for Activation Functions*.

Definition 3.1. Given some sample size n, an interval $[-L, L]$ and tail truncation T where $0 \leq T \leq L$, the activation function $\mathrm{NPF}(L, T)$ is parametrized by $(a_0, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ given by

$$f(x) = \begin{cases} a_0 + \sum_{i=1}^{k} a_i \cos((-L+T)i\pi x/L) + b_i \sin((-L+T)i\pi x/L) & x < -L+T \\ a_0 + \sum_{i=1}^{k} a_i \cos(i\pi x/L) + b_i \sin(i\pi x/L) & -L+T \leq x \leq L-T \\ a_0 + \sum_{i=1}^{k} a_i \cos((L-T)i\pi x/L) + b_i \sin((L-T)i\pi x/L) & x > L-T \end{cases}$$

k grows with sample size n and is given by $k = \lceil n^{1/7} \rceil$.

We choose to present basis expansion of arbitrary functions on the interval $[-L, L]$ because it is thought to be advantageous in practice to train neural networks with activation functions that are anti-symmetric through the origin (LeCun et al., 1998). Though the nonparametric activation functions learned may not be symmetric about the origin, this approach will hopefully result in them being as close to symmetric as possible.

Each node in the fully connected layers estimates a potentially different activation function. However, in the convolution layers *one activation function is learned per filter*. The power of a convolution net is the weight tying scheme in the convolution layer, and we preserve that by only estimating one activation function per filter. In Section 5 the effectiveness of this approach can be seen.

$\mathrm{NPFC}(L, T)$ is the notation for the nonparametrically estimated activation functions in convolution layers. Figure 1 shows some examples of activation functions learned on the CIFAR-10 dataset.

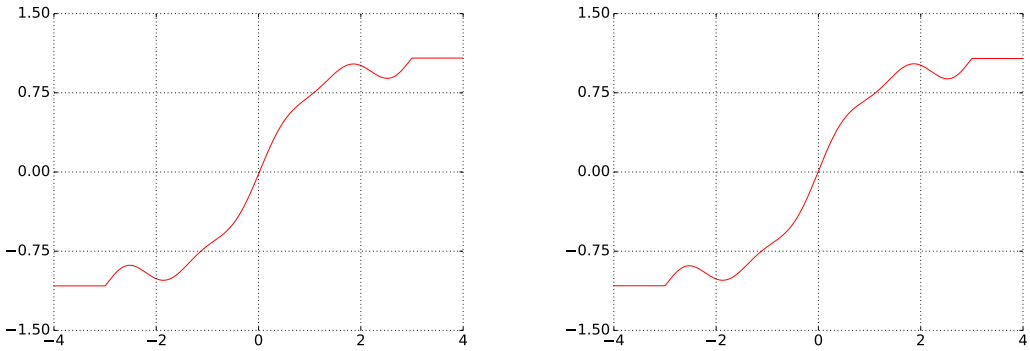


Figure 1: Examples of activation functions learned for convolution layers on CIFAR-10. Specifically these are of the type $\mathrm{NPFC}(4, 1)$. $k = 5$

3.2 Training with Learned Activation Units

A practical difficulty is that training nonparametric deep nets often proved unstable, specifically for networks with convolution layers. To remedy this our two-stage procedure is given as a generic description in Algorithm 1.

Algorithm 1 Generic Two Stage Training for Deep Convolutional Neural Networks

Input: A network architecture, hyper parameters T,L

1: Instantiate a network with ReLU activations in convolution layers and $\mathrm{NPF}(T, L)$ in all others.
2: Run training algorithm of choice.
3: Instantiate a new network with $\mathrm{NPFC}(T, L)$ activations in convolution layers and $\mathrm{NPF}(T, L)$ in all others. Instantiate all weights from the trained network. Instantiate $\mathrm{NPFC}(T, L)$ as desired.
4: Run training algorithm of choice.
5: **return** Network weights resulting from the second training.

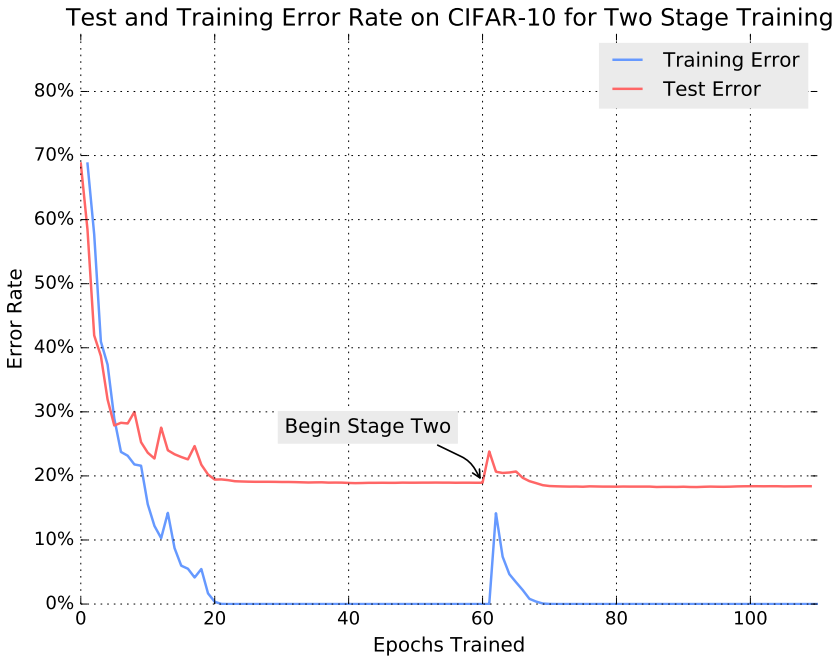
One point that is not immediately clear is how to initialize $\mathrm{NPF}(L, T)$ activation functions. A good initialization, of any parameter in the network, is crucial to the success of fitting the model. We choose to initialize them to the Fourier series approximation of the $\tanh$ function. This works quite well as will be seen in Section 5.

Also left to choice is how to perform both stages of training. Of particular interest is the training method used for the second stage. We explored the following two approaches:

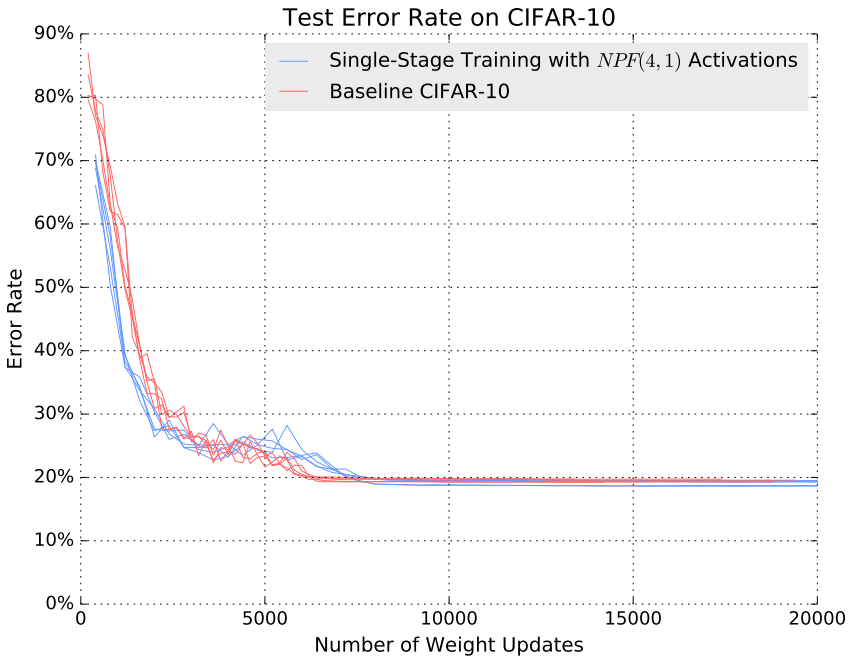
- In stage 2, train only the nonparametric activation functions, holding the remainder of the network constant.
- In stage 2, train all weights in the network together.

We discovered that both methods are successful, but allowing for the activations and weights in the network to be trained together gave slightly better results empirically, so it is those that are reported in Section 5.

Figure 2a shows the training path for a convolution network on the CIFAR-10 dataset. The network has three convolution layers followed by two fully connected layers. It is initially trained with rectified linear units as activations in the convolution layers and with $\mathrm{NPF}(4, 1)$ activations in the fully connected layers. The stage two network, instead, has $\mathrm{NPFC}(4, 1)$ activations in the convolution layer.



(a) The training and testing error paths during training. Notice the jump in error rate after stage two begins, but that ultimately it settles into better generalization performance. This is for a convolutional neural net with three convolution and two fully connected layers trained on CIFAR-10.



(b) Test errors for a convolution neural net trained on CIFAR-10. Each line represents one network trained with random initialization. Five random initializations and training paths for each network type are shown.

4 ALGORITHMIC STABILITY APPROACH TO GENERALIZATION BOUNDS

For our model to be theoretically sound, its generalization error should vanish with a sufficiently large sample size. This is true for standard neural networks, and such generalization bounds are well known (for example via VC bounds (Bartlett & Anthony, 1999)). However, because we expand the function class, we can not naively apply the traditional results. Instead of the VC bound route, we take the algorithmic stability approach.

In recent work from Hardt et al. (2015) the authors explore a class of bounds for stochastic gradient methods. They build on the definition of stability given in Bousquet & Elisseeff (2002). Denote by $f(w, z)$ some loss function of the output of a model, where w indicates the parameters of the learned model and z is an example from $X \times Y$, the space of data and labels. A is an algorithm and D is the training data. Denote $w = A(D)$ as the parameters of the model learned by the algorithm A. We take the following definition from Hardt et al. (2015).

Definition 4.1. An algorithm is ϵ-uniformly stable if for all data sets $D, D' \in (X \times Y)^n$ such that D, D' differ in at most one example, we have

$$\sup_z \mathbb{E}_A[f(A(D), z) - f(A(D'), z)] \leq \epsilon.$$

The expectation is taken with respect to any randomness present in the algorithm itself. By $\epsilon_{stab}(A, n)$ be the infimum over all ϵ for which the statement holds.

Both Bousquet & Elisseeff (2002); Hardt et al. (2015) present Theorem 4.2 regarding the generalization ability of ϵ-uniformly stable algorithms is given

Theorem 4.2. Take some algorithm ϵ-uniformly stable algorithm A. Then

$$|\mathbb{E}_{D,A}[R_D[A(D)] - R[A(D)]]| \leq \epsilon.$$

The notation $R_D[w]$ signifies the empirical risk when the loss function depends on w in addition to z. Likewise for the notation $R_D[A(D)]$. Lastly, we have Theorem 4.5 also from Hardt et al. (2015) that will allow us to derive a generalization bound for our model trained with stochastic gradient descent.

Definition 4.3. A function f is L-Lipschitz if for all points $u, v \in dom(f)$,

$$|f(u) - f(v)| \leq L||u - v||.$$

Definition 4.4. A function $f : \Omega \to \mathbb{R}$ is β-smooth if for all $u, v \in \Omega$,

$$||\nabla f(u) - \nabla f(v)|| \leq \beta||u - v||.$$

Theorem 4.5. Assume that $f(\cdot; z) \in [0, 1]$ is an $L-$Lipschitz and β-smooth loss function for every z. Suppose that we run SGM for T steps with monotonically non increasing step sizes $\alpha_t \leq c/t$. Then SGM has uniform stability with

$$\epsilon_{stab} \leq \frac{1 + 1/\beta c}{n - 1}(2cL^2)^{\frac{1}{\beta c+1}} T^{\frac{\beta c}{\beta c+1}}.$$

In particular, up to some constant factors, the following relationship approximately holds

$$\epsilon_{stab} \leq \frac{T^{1-1/(\beta c+1)}}{n}.$$

(Hardt et al., 2015)

In order to apply the theorem, we demonstrate our network architecture and choice of loss function meets this requirement. Once ϵ-stability is established, generalization follows by Theorem 4.2.

4.1 FEED-FORWARD NETWORKS WITH NPF(T, L) ACTIVATIONS

For the full proofs of results in this section, see Appendix A. Assume a network architecture of Q fully connected hidden layers (affine transformation layers) fed into a softmax layer which classifies into one of c classes. The weight vector for the network w should be viewed as the all parameters of the affine transformations and activation functions.

The loss function is the negative log-likelihood of the network output, which is a softmax over c classes. We treat the softmax layer as two layers for the purposes of demonstrating the generalization

bound. One layer is an affine transformation of the output of the lower layers in the network to c nodes. We will denote the result of this computation at each node i in the affine transformation layer within the softmax layer as $g_{Q+1}(w,x)_i$. The second layer maps the outputs at these c nodes to c nodes – this will be the network output. It computes at the the i^{th} node

$$o_i(w,x) = \frac{\exp g_{Q+1}(w,x)_i}{\sum_{i=1}^{c} \exp g_{Q+1}(w,x)_i}.$$

Thus, on some training example (x,y), the negative log-likelihood of the network output is

$$f(w,(x,y)) = -\log(o_i(w,x)) = -g_{Q+1}(w,x)_y + \log\left(\sum_{i=1}^{c} \exp g_{Q+1}(w,x)_c\right).$$

If we can demonstrate that this f has the desired properties, then the generalization bound given in Section 4 will apply.

One point to note is that in order to control the gradients of the various transformations in the layers in the network, we need to control the norm of the weight matrices W and biases b in the affine layers and the coefficients of the activations $\mathrm{NPF}(L,T)$. This is a common technique and is compatible with the framework in Hardt et al. (2015), as it only makes updates *less* expansive. Note that though f does not have range $[0,1]$, it is bounded due to the max-norm constraints and can be rescaled. Some smoothing is assumed in order achieve the bound, but in practice the smoothing is not a concern as it can be assumed to take place in an ϵ-neighborhood of the transition from one piece of the activation function to another. The smoothed nonparametric activation functions are denoted $\mathrm{NPF}_\epsilon(L,T)$. Recall that $k = \lceil n^{1/7} \rceil$.

Theorem 4.6. Consider a fully connected network with a softmax output layer and $\mathrm{NPF}_\epsilon(L,T)$ activation functions. Assume we use SGD with a max-norm constraint on the weight matrix and biases in the affine layers and on the coefficients of the activation functions to train the network. Then there exists some K such that the loss function as given in Section 4.1 is K-Lipschitz. Specifically, for any such network where the number of hidden layers $Q \geq 1$ and there are p nodes per layer,

$$K = \max\left\{\mathcal{O}\left(\frac{\sqrt{Q}(p^{Q+1}k^{3Q}+p^{(2Q+1)/2}k^{3Q-1}L)}{\epsilon^Q}\right), \mathcal{O}\left(\frac{\sqrt{Q}(p^{Q+1}k^{3Q}+p^{(2Q+1)/2}k^{3Q-1}L)}{\epsilon^Q L^Q}\right)\right\}.$$

Proof. The full proof is in Appendix A. The proof idea is to bound the first partial derivatives of the loss function f with respect to each parameter in the network. Then we can proceed recursively down through the network to obtain a bound. From there the gradient can be bounded, implying the loss function f is Lipschitz. □

Theorem 4.7. Consider a fully connected network with a softmax output layer and $\mathrm{NPF}_\epsilon(L,T)$ activation functions. Assume we use SGD with a max-norm constraint on the weight matrix and biases in the affine layers and on the coefficients of the activation functions to train the network. Then there exists some K such that the loss function as given in Section 4.1 is β-smooth. Specifically, for any such network where the number of hidden layers $Q \geq 1$ and there are p nodes per layer,

$$\beta = \max\left\{\mathcal{O}\left(Q\frac{p^{3Q+5/2}k^{9Q-2}}{L^{2Q}\epsilon^{3Q-1}}\right), \mathcal{O}\left(Q\frac{p^{2^Q+Q+2}k^{5\times 2^Q+3Q}L^{2^{Q-1}+1}}{\epsilon^{2^Q+Q}}\right)\right\}.$$

Proof. The full proof is in Appendix A. The proof idea is to bound the second partial derivatives of f with respect to each of the parameters in the network. Then we can bound $|\lambda_{\max}(\nabla^2 f)|$ and thus the largest singular value. This gives an upper bound of β on $||\nabla^2 f||_2$, implying f is β-smooth. □

Theorem 4.8. Consider a fully connected network with a softmax output layer and $\mathrm{NPF}_\epsilon(L,T)$ activation functions. Assume we use SGD with a max-norm constraint on the weight matrix and biases in the affine layers and on the coefficients of the activation functions to train the network. Then, for all practical values of L,T,ϵ,Q and p, the resulting Lipschitz constant K and smoothness constant β are sufficiently large that

$$\epsilon_{stab} \leq \frac{T}{n-1}.$$

Thus if $T = \mathcal{O}(\sqrt{n})$, $\epsilon_{stab} \to 0$ as sample size $n \to \infty$.

Proof. By Theorems 4.5, 4.6, and 4.7 the result is immediate. □

5 EXPERIMENTAL RESULTS

In this section we provide empirical results for our novel nonparametric deep learning model. The goal of these experiments is to demonstrate its efficacy when used with several different architectures and on several different datasets, not to beat the current best result on either dataset. Crucially, our experiments provide an apples to apples comparison between networks with identical architectures and employing the same regularization techniques *where the only difference is choice of activation function*. The generalization results from the Section 4 are directly relevant to the experimental results here. In all experiments listed in this section, the reported testing error is an estimation of the generalization error because the final training error was zero in all cases.

5.1 IMPLEMENTATION, REGULARIZATION AND ARCHITECTURES

To implement the various deep architectures, we use the Python CPU and GPU computing library Theano (Bergstra et al., 2010). We use Theano as an interface to the powerful CUDA and CuDNN libraries (John Nickolls, 2008; Chetlur et al., 2014) enabling fast training for our neural networks. We ran our simulations on Princeton's SMILE server and TIGER computing cluster. The SMILE server was equipped with a single Tesla K40c GPU, while the TIGER cluster has 200 K20 GPUs.

We use early stopping and dropout as regularization techniques (Srivastava et al., 2014; Hardt et al., 2015). The error rate reported for each experiment is the lowest testing error seen during training.

A common deep network architecture consists 3 stacked convolutional layers with 2 fully connected layers on top (Srivastava et al., 2014; Agostinelli et al., 2015). The convolutional layers have 96, 128, and 256 filters each and each layer has a 5×5 filter size that is applied with stride 1 in both directions. The max pooling layers pool 3×3 fields and are applied with a stride of 2 in both directions. We use this architecture for the more challenging CIFAR-10 dataset.

5.2 MNIST

This dataset consists of 60,000 training and 10,000 testing images. Images are 28×28 rasterized grayscale images – that is they have one channel. All data points are normalized before training by taking the entire combined dataset, finding the maximum and minimum entry, then centering and rescaling. The dataset is from Lecun & Cortes (1999).

The experiments investigate the effects of our techniques on standard multilayer neural networks as well as convolutional neural networks. Specifically, for the standard neural networks we use networks with three fully connected hidden layers with 2048 units in each layer fed into a 10-way softmax.

The convolutional neural networks in these experiments have two convolutional layers with 32 and 64 filters respectively. Filters are 5×5 and these layers are followed by a max pooling layer with 2×2 pool size. The feed-forward networks consist of 3 fully connected layers with 2048 units each. A mini-batch size of 250 was used for all experiments.

Table 1 contains the results from the MNIST baseline experiments.

Table 1: MNIST baseline comparisons

Name	Layers (Units)	Activation	Best Error	Mean Error
Baseline Neural Net	3 (2048)	tanh	1.66%	1.79%
Baseline Neural Net	3 (2048)	ReLU	1.65%	1.76%
Baseline NN with Dropout	3 (2048)	ReLU	1.35%	1.42%
Baseline CNN	N/A	ReLU	0.88%	0.96%
Baseline CNN with Dropout	N/A	ReLU	0.78%	0.84%

For the experiments with $\mathrm{NPF}(L, T)$ activations, we use the same fully connected and convolutional architectures as in the MNIST baseline experiments. Table 2 contains the results of the experiments.

Learning activation functions via Fourier basis expansion offers sizable improvements on the MNIST dataset as can be seen in Table 2. No experiments were done using our two-stage tech-

Table 2: MNIST experiments with Fourier basis expansion. Conv. Act. is the activation function in the convolutional layers (if present). F.C. Act. is the activation function in the fully connected layers. Epochs is the mean number of epochs in training. Results of note appear in bold.

Architecture	Dropout	Conv. Act.	F.C. Act.	Best Error	Mean Error	Epochs
NN	No	N/A	$\mathrm{NPF}(4,1)$	1.60%	1.64%	35
NN	Yes	N/A	$\mathrm{NPF}(4,1)$	**1.10%**	1.20%	119
CNN	No	ReLU	$\mathrm{NPF}(4,1)$	0.82%	0.87%	39
CNN	Yes	ReLU	$\mathrm{NPF}(4,1)$	0.66%	0.74%	72
CNN	No	$\mathrm{NPF}(4,1)$	$\mathrm{NPF}(4,1)$	0.78%	0.86%	40
CNN	Yes	$\mathrm{NPF}(4,1)$	$\mathrm{NPF}(4,1)$	0.77%	0.95%	80
CNN	No	$\mathrm{NPFC}(4,1)$	$\mathrm{NPF}(4,1)$	**0.77%**	0.84%	37
CNN	Yes	$\mathrm{NPFC}(4,1)$	$\mathrm{NPF}(4,1)$	**0.64%**	0.69%	81

nique, as it proved unnecessary on this comparatively easy to learn dataset. Notice that the weight tying in the NPFC activation makes a considerable difference for the convolution nets, especially when dropout is used in conjunction with learning the activation functions. The biggest improvement is in learning activations for both convolution and fully connected layers and using dropout – a relative 15% improvement over the baseline. Using ReLU activations in the convolution layers with nonparametric activations in the fully connected layers also offered sizable performance improvements over the baseline.

In feedforward networks with three fully connected layers there also is an improvement in the test error rate. Here again the improvement is more pronounced with dropout, from 1.35% to 1.10% versus 1.66% to 1.60%.

5.3 CIFAR-10

The CIFAR-10 dataset is due to Krizhevsky (2009). This dataset consists of 50,000 training and 10,000 test images. The images are three channel color images with dimension 32×32. Thus each image can be viewed either as a $3 \times 32 \times 32$ tensor or a vector of length 3072. These images belong to one of ten classes. We apply ZCA whitening and global contrast normalization to the dataset as in Srivastava et al. (2014). The architecture described at the beginning of Section 5 is used for all experiments. For dropout nets we follow Srivastava et al. (2014) and use a dropout of 0.9 on the input, 0.75 in the convolution layers and 0.5 in the fully connected layers. The baseline results can be found in Table 3. Mini-batch sizes between 125 and 250 are used, depending upon memory constraints.

Table 3: CIFAR-10 baseline results. Epochs is average number of epochs trained. There were five runs of each experiment.

Name	Activation	Best Error	Mean Error	Epochs
Baseline CNN	ReLU	19.23%	19.46%	64
Baseline CNN + Dropout	ReLU	14.52%	15.20%	147

Table 4: CIFAR10 experiments with Fourier basis. Conv. Act. is the activation function in the convolutional layers. F.C. Act. is the activation function in the fully connected layers. In both cases it is the activation in the final stage of training (if two stages are used). For two stage training, error rates and epochs for both first and second stages are given. Results of note appear in bold.

Training	Dropout	Conv. Act.	F.C. Act.	Best Error	Mean Error	Epochs
One Stage	No	$\mathrm{NPFC}(4,1)$	$\mathrm{NPF}(4,1)$	20.09%	20.35%	36
One Stage	No	ReLU	$\mathrm{NPF}(4,1)$	**18.58%**	19.00%	68
One Stage	Yes	ReLU	$\mathrm{NPF}(4,1)$	**14.04** %	14.77%	124
Two Stage	No	$\mathrm{NPFC}(4,1)$	$\mathrm{NPF}(4,1)$	**18.26%**	18.79%	60/61
Two Stage	Yes	$\mathrm{NPFC}(4,1)$	$\mathrm{NPF}(4,1)$	**13.22%**	13.56%	140/64

In Table 4, one-stage training corresponds to standard back-propagation as in the baseline experiments. The two-stage procedure is the one we introduce to train convolution nets with learned activation functions described in Section 3.2. As can be seen in Table 4, one stage training does not work when learning nonparametric activations in convolution layers – test error is worse than the baseline result. However, by using the two-stage process we see a relative 5% boost in performance without dropout and a relative 9% boost with dropout! In absolute terms we achieve up to a 1.3% improvement in generalization performance using learned activations.

Also in Table 4, we can see that we achieved improved performance both with and without dropout when using ReLU activations for convolution layers and $\mathrm{NPF}(4, 1)$ for fully connected layers. Furthermore, Figure 2b shows that in terms of number of weight updates required, adding nonparametric activations did not slow training.

6 Discussion and Conclusions

As can be seen in Section 5, nonparametric activation functions offer a meaningful improvement in the generalization performance of deep neural nets in practice. We achieve relative improvements in performance of up to 15% and absolute improvements of up to 1.3% on two benchmark datasets. Equally importantly, networks with the activation functions $\mathrm{NPF}(L, T)$ and $\mathrm{NPFC}(L, T)$ can be trained as fast as their baseline comparison. To realize these improvements on more challenging datasets such as CIFAR-10 we introduced a two-stage training procedure. Our nonparametric activation functions are principled in that they have the necessary properties to guarantee vanishing generalization error in the sense of Bousquet & Elisseeff (2002). Specifically, it was shown in Theorem 4.8, from Section 4.1, that for any feed-forward network architecture using $\mathrm{NPF}(L, T)$ activation functions, we achieve vanishing generalization error as sample size increases.

One interesting direction for future work is to investigate why Fourier basis expansion is successful where other methods, such as polynomial basis expansion (which we also explored), were not. Both can be theoretically justified, yet only one works well in practice. Further study of how to eliminate the two-stage process is needed.

Ultimately, we improve upon other approaches of expanding the function class which can be learned by deep neural networks. An important previous work on learning more general function approximations at each node is that of Lin et al. (2013). Their result offers a new type of convolution filter that can learn a broader function class, and in practice their technique also works well. It is limited however in that its use is restricted to convolution layers in convolution neural networks. Ours is applicable to *any* network architecture and, in Section 5, we demonstrate its success on multiple datasets.

REFERENCES

Agostinelli, Hoffman, Sadowski, and Baldi. Learning activation functions to improve deep neural networks. In *ICLR*, 2015.

Francis Bach and Eric Moulines. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems (NIPS)*. 2011.

Peter Bartlett and Martin. Anthony. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 8RU, UK, 1999.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

Olivier Bousquet and Andre Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2, 2002.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning, 2014.

Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240, 2015. URL `http://arxiv.org/abs/1509.01240`.

Michael Garland Kevin Skadron John Nickolls, Ian Buck. Scalable parallel programming with cuda. *ACM Queue*, 6, 2008.

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

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K. (eds.), *Neural Networks: Tricks of the trade*. Springer, 1998.

Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits. 1999. URL `http://yann.lecun.com/exdb/mnist/`.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2013.

D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *JMLR*, volume 37, 2015.

Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

A GENERALIZATION BOUND FOR FULLY CONNECTED NETWORKS WITH FOURIER BASIS ACTIVATION FUNCTIONS

This is an extension of Section 4.1. For the purposes of this section assume that all data x are vectors in $[-1, 1]^m$, which is permissible because we classify input that is bounded. Furthermore, we assume that the network architecture is a series of Q fully connected hidden layers (affine transformation layers) fed into a softmax layer which classifies into one of c classes. The weight vector for the network w should be viewed as the all parameters of the affine transformations and activation functions.

The loss function is the negative log-likelihood of the network output, which is a softmax over c classes. The softmax layer in our framework should be treated as two layers for the purposes of demonstrating the generalization bound. One layer is an affine transformation of the output of the lower layers in the network to c nodes. We will denote the result of this computation at each node i in the affine transformation layer within the softmax layer as $g_{Q+1}(w, x)_i$. The second layer maps the outputs at these c nodes to c nodes – this will be the network output. It computes at the the i^{th} node

$$o_i(w, x) = \frac{\exp g_{Q+1}(w, x)_i}{\sum_{i=1}^{c} \exp g_{Q+1}(w, x)_i}.$$

Thus, on some training example (x, y), the negative log-likelihood of the network output is

$$f(w, (x, y)) = -\log(o_i(w, x)) = -g_{Q+1}(w, x)_y + \log\left(\sum_{i=1}^{c} \exp g_{Q+1}(w, x)_c\right).$$

If we can demonstrate that this f has the desired properties, then the generalization bound given in Section 4 will apply.

For clarity, we refer to network of stacked affine transformation layers as a fully connected network. These layers compute $Wx + b$, where x is either the input layer (so $x \in \mathbb{R}^m$) or it is the output of the previous layer in the network (in which case $x \in \mathbb{R}^p$, p being the number of nodes in that layer).

Regarding notation:

- w is a vector denoting all the parameters in the network, so it consists of all parameters noted below.
- W_i denotes a portion of the parameter vector, corresponding to the affine transformation of the p_{i-1}-dimensional output of the $(i-1)^{th}$ layer to the p_i-dimensional input to the i^{th} layer in the network.
- $W_{i,j}$ denotes the j^th row of the matrix W_i.
- $W_{i,j,k}$ refers to the weight of the edge from node j in the $(i-1)^{th}$ layer to edge c in the i^{th} layer.
- b_i denotes the constant term in the affine transformation from the $(i-1)^{th}$ layer to the i^{th} layer.
- $b_{i,j}$ denotes the j^{th} component of the vector b_i.
- $a_0^{i,j}$, $(a_1^{i,j}, \ldots, a_k^{i,j})$ and $(b_1^{i,j}, \ldots, b_k^{i,j})$ denote the parameters of activation function of the j^{th} node in the i^{th} layer.

The 0^{th} layer is the input layer. Let Q be the number of layers in the network; if $i = Q + 1$, it refers to weights on edges in the affine transformation contained within the softmax layer.

Intuitively, our choice of network topology and use of $\mathrm{NPF}(L, T)$ activation function should have the necessary properties to apply Theorem 4.5. The definition of $\mathrm{NPF}(L, T)$ is repeated below for clarity.

Definition A.1. Given some sample size n, an interval $[-L, L]$ and tail truncation T where $0 \leq T \leq L$, the activation function $\mathrm{NPF}(L, T)$ is parametrized by $(a_0, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ given

by

$$f(x) = \begin{cases} a_0 + \sum_{i=1}^{k} a_i \cos((-L+T)i\pi/L) + b_i \sin((-L+T)i\pi/L) & x < -L+T \\ a_0 + \sum_{i=1}^{k} a_i \cos(i\pi x/L) + b_i \sin(i\pi x/L) & -L+T \leq x \leq L-T \\ a_0 + \sum_{i=1}^{k} a_i \cos((L-T)i\pi/L) + b_i \sin((L-T)i\pi/L) & x > L-T \end{cases}$$

k grows with sample size n and is given by $k = \lceil n^{1/7} \rceil$.

We need to control the norm of the weight matrices W and biases b in the affine transformations. In addition, we also need to control the magnitude of the coefficients in the basis expansion. This does not affect the applicability of the results from Hardt et al. (2015), as it at most reduces the expansivity of the updates.

Furthermore, for the analysis of their properties we assume a slightly modified version of the activation functions – in an epsilon neighborhood about the points where the pieces of the function connect we smooth their joint by taking a convex combination of the two pieces to smoothly transition from one to the other. To this end define the following functions

$$L(x) = a_0 + \sum_{i=1}^{k} a_i \cos((-L+T)i\pi/L) + b_i \sin((-L+T)i\pi/L),$$

$$M(x) = a_0 + \sum_{i=1}^{k} a_i \cos(i\pi x/L) + b_i \sin(i\pi x/L),$$

$$U(x) = a_0 + \sum_{i=1}^{k} a_i \cos((L-T)i\pi/L) + b_i \sin((L-T)i\pi/L).$$

for given $L, T, k, (a_0, \ldots, a_k)$, and $(b_1, \ldots, b_k)$. Then the ϵ-neighborhood smoothed nonparametric activation functions are given by Definition A.2.

Definition A.2. Given some sample size n, an interval $[-L, L]$ and tail truncation T where $0 \leq T \leq L$, the activation function $\mathrm{NPF}_\epsilon(L, T)$ is parametrized by $(a_0, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ given by

$$f(x) = \begin{cases} L(x) & x < -L+T-\epsilon \text{ [Case 1]} \\ \frac{x+L-T+\epsilon}{2\epsilon} M(x) + \frac{-L+T+\epsilon-x}{2\epsilon} L(x) & -L+T-\epsilon \leq x \leq -L+T+\epsilon \text{ [Case 2]} \\ M(x) & -L+T+\epsilon \leq x \leq L-T-\epsilon \text{ [Case 3]} \\ \frac{L-T+\epsilon-x}{2\epsilon} M(x) + \frac{x-L+T+\epsilon}{2\epsilon} U(x) & L-T-\epsilon \leq x \leq L-T+\epsilon \text{ [Case 4]} \\ U(x) & x > L-T+\epsilon \text{ [Case 5]} \end{cases}$$

k grows with sample size n and is given by $k = \lceil n^{1/7} \rceil$.

Since we are concerned with behavior as ϵ is made small – because we are approximating $\mathrm{NPF}(T, L)$ with $\mathrm{NPF}_\epsilon(T, L)$ – assume without loss of generality that $\epsilon \leq 1$.

A.1 Lipschitz Constant

We proceed layer by layer to obtain bounds on the first partial derivatives of the loss function – which will allow us to bound the gradient. Lemma A.3 gives bounds for the first layer. The notation $a(w, x)$ denotes the output of a layer-wide activation transformation. $g(w, x)$ denotes the output of a layer-wide affine transformation.

Lemma A.3. Assume $x \in [-a, a]^m$. Now, consider the affine transformation $g_1(w) = W_1 x + b_1$ from $\mathbb{R}^m \to \mathbb{R}^p$, thus $W_1 \in \mathbb{R}^{m \times p}$ and $b \in \mathbb{R}^p$. Denoting by $g_1(w, x)_i = W_{1,i} x + b_{1,i}$, it follows that g_1 has first partial derivatives bounded by $\max\{1, a\}$.

Proof. Clearly, $\nabla g_1(w, x)_i = [1, \ldots, 1; x; 0, \ldots, 0]$. Thus it has first partial derivatives bounded by $\max\{1, a\}$. □

What if we have instead as input to the affine transformation a function of x rather than x itself? Then we have the Lemma A.4.

Lemma A.4. Let $a_{i-1}(w,x) \in [-a,a]^{p_{i-1}}$, the output of the $(i-1)^{th}$ layer in the network. Now, consider the affine transformation

$$g_i(w,x) = W_i a_{i-1}(w,x) + b_i.$$

from $\mathbb{R}^{p_{i-1}} \to \mathbb{R}^{p_i}$. $b_i \in \mathbb{R}^d$. If we have the following assumptions:

- $a_i(w,x)_q$ has first partial derivatives bounded by B, and
- $||W_{i,j}||_2 \leq N$.

Denoting by $g_i(w,x)_j = W_{i,j}a_{i-1}(w,x) + b_{i,j}$, it follows that $g_i(w,x)_j$ has first partial derivatives bounded by $\max\{1, a, \sqrt{n}NL\}$.

Proof. We need to first examine

$$\begin{aligned}\frac{\partial g_i(w,x)_j}{\partial b_{i,j}} &= \frac{\partial b_{i,j}}{\partial b_{i,j}} + \frac{\partial}{\partial b_{i,j}}\left(\sum_{q=1}^{n} W_{i,j,q}a_{i-1}(w,x)_q\right) \\ &= 1 + 0 = 1.\end{aligned}$$

Next we have for any m,

$$\begin{aligned}\frac{\partial g_i(w,x)_j}{\partial W_{i,j,m}} &= \frac{\partial b_{i,j}}{\partial W_{i,j,m}} + \frac{\partial}{\partial W_{i,j,m}}\left(\sum_{q=1}^{n} W_{i,j,q}a_{i-1}(w,x)_q\right) \\ &= 0 + a_{i-1}(w,x)_m + \sum_{q\neq m} W_{i,j,q}\frac{\partial a_{i-1}(w,x)_q}{\partial W_{i,j,m}} \\ &= a_{i-1}(w,x)_m.\end{aligned}$$

Partial derivatives with respect to all other parameters from the i^{th} layer are clearly 0. If $l > i$, partial derivatives with respect to all parameters from the l^th layer are clearly 0. If $l < i$,

$$\begin{aligned}\left|\frac{\partial g_i(w,x)_j}{\partial W_{l,m,q}}\right| &= \left|\frac{\partial b_{i,j}}{\partial W_{l,m,q}} + \frac{\partial}{\partial W_{l,m,q}}\left(\sum_{r=1}^{p_{i-1}} W_{i,j,r}a_{i-1}(w,x)_r\right)\right| \\ &= \left|0 + \sum_{r=1}^{p_{i-1}} W_{i,j,r}\frac{\partial a_{i-1}(w,x)_r}{\partial W_{l,m,q}}\right| \\ &\leq N\sqrt{\sum_{r=1}^{p_{i-1}}\left(\frac{\partial a_{i-1}(w,x)_r}{\partial W_{l,m,q}}\right)^2} \qquad \text{(Cauchy-Schwartz)} \\ &\leq \sqrt{p_{i-1}}NB.\end{aligned}$$

Likewise,

$$|\frac{\partial g_i(w,x)_j}{\partial b_{l,q}}| \leq \sqrt{p_{i-1}}NB$$

and for the partial derivatives with respect to activation function parameters. It follows that the transformations $g_i(w,x)_j$ have first partial derivatives bounded by $\max\{1, a, \sqrt{p_{i-1}}NB\}$. □

Lemma A.5 provides a bound on the range of affine transformations of outputs of $\mathrm{NPF}_\epsilon(T,L)$ activations.

Lemma A.5. Consider the affine transformation in layer i, $g_i(w,x)$. Assume

- All layers in the network have $\mathrm{NPF}_\epsilon(L,T)$ activations and that coefficients in these activations are bounded by M,
- For all i and j, $||W_{i,j}||_2 \leq N$, and
- For all i and j, $|b_{i,j}| \leq O$.

Then, for all j, $|g_i(w,x)_j| \leq \sqrt{p_{i-1}}(2k+1)NM + O = \mathcal{O}(\sqrt{p_{i-1}}k)$.

Proof. Recall that $g_i(w,x)_j = W_{i,j}a_{i-1}(w,x) + b_{i,j}$, thus by Cauchy-Schwartz and that $\mathrm{im}(\sin) = \mathrm{im}(\cos) = [-1,1]$,

$$|g_i(w,x)_j| \leq \sqrt{p_{i-1}}(2k+1)NM + O = \mathcal{O}(\sqrt{p_{i-1}}k).$$

□

Next, we bound the component functions of $\mathrm{NPF}_\epsilon(L,T)$. In addition, to derive asymptotic bounds, we assume N,M, and O are constants as they are part of the training procedure and not the network structure. We are more concerned with how the smoothness and continuity properties change as n,k,L,T,Q and p_i change. Also note that for the activation function $\mathrm{NPF}_\epsilon(L,T)$ to be well defined, it must be that $\epsilon \leq L$. We also assume $\epsilon \leq 1$.

Lemma A.6. Assume $g_i(w,x) \in [-a,a]^{p_i}$ and that $g_i(w,x)_j$ has first partial derivatives bounded by B. Further assume that $|a_d| \leq M$ and define $f(w,x) = a_d \cos\left(\frac{d\pi g_i(w,x)_j}{L}\right)$. Then $f(w,x)$ has first partial derivatives bounded by $\max\{\frac{d\pi MB}{L}, 1\}$.

Proof. Taking the derivative of f with respect to a_d yields

$$\left|\frac{\partial f}{\partial a_d}\right| = \left|\cos\left(\frac{d\pi g_i(w,x)_j}{L}\right)\right| \leq 1.$$

Next if we take the derivative of f with respect to any weight from elsewhere in the network – some z – we get

$$\frac{\partial f}{\partial z} = -\frac{d\pi a_d}{L}\frac{\partial g_i(w,x)_j}{\partial z}\sin\left(\frac{d\pi g_i(w,x)_j}{L}\right),$$

and therefore that

$$\left|\frac{\partial f}{\partial z}\right| \leq \left|\frac{d\pi a_d}{L}\frac{\partial g_i(w,x)_j}{\partial z}\right| \leq \frac{d\pi MB}{L},$$

and so the partial derivatives of f are bounded by $\max\{\frac{d\pi MB}{L}, 1\}$. □

Immediately from the proof to Lemma A.6 we see that an identical result holds if we replaced $\cos$ with $\sin$.

Lemma A.7. Let $g_i(w,x) \in [-a,a]^{p_{i-1}}$ and assume

- $g_i(w,x)_j$ has first partial derivatives bounded by B,
- All layers in the network have $\mathrm{NPF}_\epsilon(L,T)$ activations and that coefficients in these activations are bounded by M,
- For all i and j, $||W_{i,j}||_2 \leq N$, and
- For all i and j, $|b_{i,j}| \leq O$.

Consider the layer-wide activation transformation $a_i(w,x)$ where $a_i(w,x)_j$ is given by a function of the type $\mathrm{NPF}_\epsilon(L,T)$. $a_i(w,x)_j$ then has first partial derivatives with bound

$$B' = \mathcal{O}\left(\max\left\{\frac{\sqrt{p_{i-1}}k^3B + k^2L}{\epsilon L}, \frac{\sqrt{p_{i-1}}k^3B + k^2L}{\epsilon}\right\}\right).$$

Proof. To bound the partial derivatives, first consider Cases 1 and 5, clearly the derivative with respect to some parameter z is 0. For Case 3, by Lemma A.6,

$$\left|\frac{\partial M(g_i(w,x)_j)}{\partial z}\right| \leq \frac{(k^2+k)MB\pi}{L} + (k^2+k) = \frac{(k^2+k)(MB+L)\pi}{L} = \mathcal{O}\left(k^2 + \frac{k^2B}{L}\right).$$

Next consider Case 2, we have

$$\frac{\partial}{\partial z}\left(\frac{g_i(w,x)_j + L - T + \epsilon}{2\epsilon} M(g_i(w,x)_j) + \frac{-L + T + \epsilon - g_i(w,x)_j}{2\epsilon} L(g_i(w,x)_j)\right)$$
$$= \frac{\partial}{\partial z}\left(\frac{g_i(w,x)_j + L - T + \epsilon}{2\epsilon} M(g_i(w,x)_j)\right) - \frac{\partial g_i(w,x)_j}{\partial z}\frac{1}{2\epsilon} L(g_i(w,x)_j)$$
$$= \frac{1}{2\epsilon}\frac{\partial g_i(w,x)_j}{\partial z} M(g_i(w,x)_j) + \left(\frac{g_i(w,x)_j + L - T + \epsilon}{2\epsilon}\right)\frac{\partial M(g_i(w,x)_j)}{\partial z} - \frac{\partial g_i(w,x)_j}{\partial z}\frac{1}{2\epsilon} L(g_i(w,x)_j)$$
$$= \left(\frac{M(g_i(w,x)_j) - L(g_i(w,x)_j)}{2\epsilon}\right)\frac{\partial g_i(w,x)_j}{\partial z} + \left(\frac{g_i(w,x)_j + L - T + \epsilon}{2\epsilon}\right)\frac{\partial M(g_i(w,x)_j)}{\partial z}.$$

Both $M(x)$ and $L(x)$ are trivially bounded above by $(k^2 + k + 1)M$. Therefore

$$\left|\frac{\partial}{\partial z}\left(\frac{g_i(w,x)_j + L - T + \epsilon}{2\epsilon} M(g_i(w,x)_j) + \frac{-L + T + \epsilon - g_i(w,x)_j}{2\epsilon} L(g_i(w,x)_j)\right)\right|$$
$$= \left|\left(\frac{M(g_i(w,x)_j) - L(g_i(w,x)_j)}{2\epsilon}\right)\frac{\partial g_i(w,x)_j}{\partial z} + \left(\frac{g_i(w,x)_j + L - T + \epsilon}{2\epsilon}\right)\frac{\partial M(g_i(w,x)_j)}{\partial z}\right|$$
$$\leq \frac{(k^2 + k + 1)BM}{\epsilon} + \left(\frac{\sqrt{p_{i-1}}(2k+1)NM + O + L + \epsilon}{2\epsilon}\right)\left(\frac{(k^2 + k)(MB + L)\pi}{L}\right)$$
$$= \mathcal{O}\left(\max\left\{\frac{\sqrt{p_{i-1}}k^3 B + k^2 L}{\epsilon L}, \frac{\sqrt{p_{i-1}}k^3 B + k^2 L}{\epsilon}\right\}\right).$$

By symmetry, we have the same bound for Case 4. Thus overall, the bound on the first partial derivatives of $a_i(w,x)_j$ is

$$\mathcal{O}(\max\left\{\frac{\sqrt{p_{i-1}}k^3 B + k^2 L}{\epsilon L}, \frac{\sqrt{p_{i-1}}k^3 B + k^2 L}{\epsilon}\right\}),$$

as for all values of k,B,L,ϵ and p_{i-1} the bound for Cases 2 and 4 dominates the bound for Case 3. □

Finally, we turn to the top layer of the network.

Lemma A.8. With the notation established in Section A, assume there are c classes, the network has Q layers of affine transformations, and each $g_L(w,x)_j$ has first partial derivatives bounded by B. Then the loss function f is $2B\sqrt{(p_l+1)c + \sum_{i=1}^{Q} p_i(p_{i-1}+1)}$−Lipschitz. If all layers have p nodes, then f is $2B\sqrt{(Q-1)p^2 + (c+m+Q)p + c}$-Lipschitz. In asymptotic notation, it is K-Lipschitz with $K = \mathcal{O}(Bp\sqrt{Q})$.

Proof. Consider the partial derivative of f with respect to an arbitrary parameter z

$$\frac{\partial f}{\partial z} = \frac{\partial g_L(w,x)_y}{\partial z} + \frac{1}{\sum_{j=1}^{c}\exp(g_L(w,x)_j)}\sum_{j=1}^{c}\frac{\partial g_L(w,x)_j}{\partial z}\exp(g_L(w,x)_j).$$

By the assumption $|\frac{\partial g_L(w,x)_y}{\partial z}| \leq Q$. Therefore the following holds

$$\left|\frac{\partial f}{\partial z}\right| = \left|\frac{\partial g_L(w,x)_y}{\partial z} + \frac{1}{\sum_{j=1}^{c}\exp(g_L(w,x)_j)}\sum_{j=1}^{c}\frac{\partial g_L(w,x)_j}{\partial z}\exp(g_j)\right|$$
$$\leq \left|\frac{\partial g_L(w,x)_y}{\partial z}\right| + \left|\frac{1}{\sum_{j=1}^{c}\exp(g_L(w,x)_j)}\sum_{j=1}^{k}\frac{\partial g_L(w,x)_j}{\partial z}\exp(g_L(w,x)_j)\right|$$
$$\leq B + \left|\max_j \frac{\partial g_L(w,x)_j}{\partial z}\right|$$
$$\leq 2B.$$

If layer i has p_i nodes there are $p_i \times p_{i-1} + p_i = p_i(p_{i-1}+1)$ parameters corresponding to the affine transformation from layer $i-1$ to layer i. Recall layer 0 is the input layer, so $p_0 = m$.

Furthermore, the affine transformation in the softmax has $p_l \times c + c$ parameters. Thus the total number of parameters in the network is

$$(p_l + 1)c + \sum_{i=1}^{Q} p_i(p_{i-1} + 1),$$

which implies that f is $2B\sqrt{(p_l + 1)c + \sum_{i=1}^{Q} p_i(p_{i-1} + 1)}$−Lipschitz. If all layers have the same number of nodes, then the total number of parameters in the network is

$$(Q - 1)p^2 + (c + m + Q)p + c$$

and so in this case f is $2B\sqrt{(Q - 1)p^2 + (c + m + Q)p + c}$-Lipschitz. In asymptotic notation, it is K-Lipschitz with $K = \mathcal{O}(Bp\sqrt{Q})$. □

Combining all the results above we have Theorem A.9.

Theorem A.9. Consider a fully connected network with a softmax output layer and $\mathrm{NPF}_\epsilon(L, T)$ activation functions. Assume we use SGD with a max-norm constraint on the weight matrix and biases in the affine layers and on the coefficients of the activation functions to train the network. Then there exists some K such that the loss function as given in Section A is K-Lipschitz. Specifically, for any such network, where the number of hidden layers $Q \geq 1$ and there are p nodes per layer,

$$K = \max\left\{\mathcal{O}\left(\frac{\sqrt{Q}(p^{Q+1}k^{3Q} + p^{(2Q+1)/2}k^{3Q-1}L)}{\epsilon^Q}\right), \mathcal{O}\left(\frac{\sqrt{Q}(p^{Q+1}k^{3Q} + p^{(2Q+1)/2}k^{3Q-1}L)}{\epsilon^Q L^Q}\right)\right\}.$$

Proof. By Lemmas A.3, A.4, A.7 and A.8 such an K must exist. Note that the output of a $\mathrm{NPF}_\epsilon(T, L)$ activation is in $[-(k^2 + k + 1)N, (k^2 + k + 1)N]$. Then, the computation is as follows, using the aforementioned theorems:

- $g_1 - B = 1$
- $a_1 - B = \max\left\{\mathcal{O}\left(\frac{\sqrt{p}k^3 + k^2 L}{\epsilon}\right), \mathcal{O}\left(\frac{\sqrt{p}k^3 + k^2 L}{\epsilon L}\right)\right\}$
- $g_2 - B = \max\left\{\mathcal{O}\left(\frac{pk^3 + \sqrt{p}k^2 L}{\epsilon}\right), \mathcal{O}\left(\frac{pk^3 + \sqrt{p}k^2 L}{\epsilon L}\right)\right\}$
- $a_2 - B = \max\left\{\mathcal{O}\left(\frac{p^{3/2}k^6 + pk^5 L}{\epsilon^2}\right), \mathcal{O}\left(\frac{p^{3/2}k^6 + pk^5 L}{\epsilon^2 L^2}\right)\right\}$
- $g_3 - B = \max\left\{\mathcal{O}\left(\frac{p^2k^6 + p^{3/2}k^5 L}{\epsilon^2}\right), \mathcal{O}\left(\frac{p^2k^6 + p^{3/2}k^5 L}{\epsilon^2 L^2}\right)\right\}$
- $a_3 - B = \max\left\{\mathcal{O}\left(\frac{p^{5/2}k^9 + p^2k^8 L}{\epsilon^3}\right), \mathcal{O}\left(\frac{p^{5/2}k^9 + p^2k^8 L}{\epsilon^3 L^3}\right)\right\}$

Above the transformations are listed with the bounds on their first partial derivatives. A simple inductive argument yields for $i \geq 2$

- $g_i -$
  $B = \max\left\{\mathcal{O}\left(\frac{p^{i-1}k^{3(i-1)} + p^{(2i-3)/2}k^{3i-4}L}{\epsilon^{i-1}}\right), \mathcal{O}\left(\frac{p^{i-1}k^{3(i-1)} + p^{(2i-3)/2}k^{3i-4}L}{\epsilon^{i-1}L^{i-1}}\right)\right\}$
- $a_i -$
  $B = \max\left\{\mathcal{O}\left(\frac{p^{(2i-1)/2}k^{3i} + p^{i-1}k^{3i-1}}{\epsilon^i}\right), \mathcal{O}\left(\frac{p^{(2i-1)/2}k^{3i} + p^{i-1}k^{3i-1}}{\epsilon^i L^i}\right)\right\}$

Using Lemma A.8 gives

$$K = \max\left\{\mathcal{O}\left(\frac{\sqrt{Q}(p^{Q+1}k^{3Q} + p^{(2Q+1)/2}k^{3Q-1}L)}{\epsilon^Q}\right), \mathcal{O}\left(\frac{\sqrt{Q}(p^{Q+1}k^{3Q} + p^{(2Q+1)/2}k^{3Q-1}L)}{\epsilon^Q L^Q}\right)\right\}.$$

concluding the proof. □

A.2 β-SMOOTHNESS

To derive the smoothness constant, we begin at the top of the network and work down, bounding partial second derivatives of transformations in the network.

Lemma A.10. With the notation established in Section A, assume there are c classes and each g_j has first partial derivatives bounded by B_1 and second partial derivatives bounded by B_2. Then the loss function f is $2((p_l+1)c+\sum_{i=1}^{Q} p_i(p_{i-1}+1))(B_1^2+B_2)$-smooth. If all layers have p nodes, then f is $2((Q-1)p^2+(c+m+Q)p+c)(B_1^2+B_2)$-smooth.

Proof. We begin by recalling from the proof of Lemma A.8 that for some arbitrary parameter z,

$$\frac{\partial f}{\partial z} = \frac{\partial g_L(w,x)_y}{\partial z} + \frac{1}{\sum_{l=1}^{c}\exp(g_j)}\sum_{l=1}^{c}\frac{\partial g_L(w,x)_l}{\partial z}\exp(g_L(w,x)_l).$$

Therefore differentiating again with respect to another parameter x yields

$$\frac{\partial^2 f}{\partial z \partial x} = \frac{\partial^2 g_L(w,x)_y}{\partial z \partial x} - \left[\sum_{l=1}^{c}\frac{\partial g_L(w,x)_l}{\partial z}\exp(g_L(w,x)_l)\right]\left[\sum_{l=1}^{c}\frac{\partial g_L(w,x)_l}{\partial x}\exp(g_L(w,x)_l)\right]\times$$
$$\left[\sum_{l=1}^{c}\exp(g_L(w,x)_l)\right]^{-2} + \left[\sum_{l=1}^{c}\left(\frac{\partial^2 g_L(w,x)_l}{\partial z\partial x} + \frac{\partial g_L(w,x)_l}{\partial z}\frac{\partial g_L(w,x)_l}{\partial x}\right)\exp(g_L(w,x)_l)\right]\times$$
$$\left[\sum_{l=1}^{c}\exp(g_L(w,x)_l)\right]^{-1}.$$

If we take the absolute value of both sides we can see that

$$\left|\frac{\partial^2 f}{\partial z \partial x}\right| \leq 2B_1^2 + 2B_2.$$

And therefore by Theorem C.2 it follows that

$$|\lambda_{\max}(\nabla^2 f)| \leq 2((p_l+1)c+\sum_{i=1}^{Q} p_i(p_{i-1}+1))(B_1^2+B_2).$$

Because $\nabla^2 f$ is symmetric, its singular values are just the absolute values of its eigenvalues. Therefore $||\nabla^2 f||_2 \leq 2n(B_1^2+B_2)$. By Lemma D.2, we have that f is $2(2(p_l+1)c+\sum_{i=1}^{Q} p_i(p_{i-1}+1))(B_1^2+B_2)$-smooth. If all layers have the same number of nodes, then

$$|\lambda_{\max}(\nabla^2 f)| \leq 2((Q-1)p^2+(c+m+Q)p+c)(B_1^2+B_2)$$

and so f is $2((Q-1)p^2+(c+m+Q)p+c)(B_1^2+B_2)$-smooth. □

Next, it is necessary to control the second partial derivatives of the affine transformations as the first derivatives are bounded in Lemma A.4. This leads to Lemma A.11.

Lemma A.11. Let $a_{i-1}(w,x) \in [-a,a]^{p_{i-1}}$, the output of the $(i-1)^{th}$ layer in the network. Now, consider the affine transformation

$$g_i(w,x) = W_i a_{i-1}(w,x) + b_i,$$

from $\mathbb{R}^{p_{i-1}} \to \mathbb{R}^{p_i}$. $b_i \in \mathbb{R}^d$. If we have the following assumptions:

- $a_i(w,x)_q$ has first partial derivatives bounded by B_1,
- $a_i(w,x)_q$ has second partial derivatives bounded by B_2, and
- $||W_{i,j}||_2 \leq N$.

Denoting by $g_i(w,x)_j = W_{i,j}a_{i-1}(w,x) + b_{i,j}$, it follows that $g_i(w,x)_j$ has second partial derivatives bounded by $\max\{B_1, N\sqrt{p_{i-1}}B_2\}$.

Proof. From the proof of Lemma A.4 we have that

$$\frac{\partial g_i(w,x)_j}{\partial b_{i,j}} = 1$$

and for any m

$$\frac{\partial g_i(w,x)_j}{\partial W_{i,j,m}} = a_{i-1}(w,x)_m.$$

For some $m \neq j$ and any q,

$$\frac{\partial g_i(w,x)_j}{\partial W_{i,m,q}} = 0.$$

And, lastly, that the partial derivatives with respect to any parameter z in layer $l < i$

$$\frac{\partial g_i(w,x)_j}{\partial z} = \sum_{q=1}^{p_{i-1}} W_{i,j,q} \frac{\partial a_{i-1}(w,x)_q}{\partial z}.$$

We see then that for the first and third cases the second partial derivatives are zero. For the second case above, the second partial derivatives are either 0 or bounded by B_1. What remains then is to check the fourth case. If we differentiate with respect to $b_{i,j}$,

$$\frac{\partial^2 g_i(w,x)_j}{\partial z \partial b_{i,j}} = 0.$$

If we differentiate with respect to $W_{i,m,q}$, for $m \neq j$,

$$\frac{\partial^2 g_i(w,x)_j}{\partial z \partial W_{i,m,q}} = 0.$$

If we differentiate with respect to $W_{i,j,q}$,

$$\frac{\partial^2 g_i(w,x)_j}{\partial z \partial W_{i,j,q}} = \frac{\partial a_{i-1}(w,x)_q}{\partial z}.$$

Thus the only remaining case is if we differentiate with respect to some parameter z' from a layer $l < i$:

$$\begin{aligned}\frac{\partial^2 g_i(w,x)_j}{\partial z \partial z'} &= \sum_{q=1}^{p_{i-1}} \frac{\partial}{\partial z'} \left(W_{i,j,q} \frac{\partial a_{i-1}(w,x)_q}{\partial z} \right) \\ &= \sum_{q=1}^{p_{i-1}} W_{i,j,q} \frac{\partial^2 a_{i-1}(w,x)_q}{\partial z \partial z'}.\end{aligned}$$

If we take the absolute value of both sides we get that

$$\begin{aligned}\left| \frac{\partial^2 g_i(w,x)_j}{\partial z \partial z'} \right| &= \left| \sum_{q=1}^{p_{i-1}} W_{i,j,q} \frac{\partial^2 a_{i-1}(w,x)_q}{\partial z \partial z'} \right| \\ &\leq N \sqrt{\sum_{q=1}^{p_{i-1}} \left(\frac{\partial^2 a_{i-1}(w,x)_q}{\partial z \partial z'} \right)^2} \qquad \text{(Cauchy-Schwartz)} \\ &\leq N \sqrt{p_{i-1}} B_2\end{aligned}$$

And therefore $g_i(w,x)_j$ has second partial derivatives bounded by $\max\{B_1, N\sqrt{p_{i-1}}B_2\}$. □

We also consider affine transformations on the input – although this is covered by Lemma A.11, we can obtain a better bound directly.

Lemma A.12. Assume $x \in [-a,a]^m$. Now, consider the affine transformation $g_1(w) = W_1 x + b_1$ from $\mathbb{R}^m \to \mathbb{R}^p$, thus $W_1 \in \mathbb{R}^{m \times p}$ and $b \in \mathbb{R}^p$. Denoting by $g_1(w,x)_i = W_{1,i}x + b_{1,i}$, it follows that g_1 has second partial derivatives bounded by 0.

Proof. Recall that from the proof of Lemma A.3 $\nabla g_1(w,x)_i = [1,\ldots,1;x;0,\ldots,0]$. Thus its second derivatives are all 0. □

Finally, a bound is needed for the second partial derivatives of the $\mathrm{NPF}_\epsilon(L,T)$ activations.

Lemma A.13. Assume $g_i(w,x) \in [-a,a]^{p_i}$ and that $g_i(w,x)_j$ has first partial derivatives bounded by B_1 and second partial derivatives bounded by B_2. Further assume that $|a_d| \leq M$ and define $f(w,x) = a_d \cos\left(\frac{d\pi g_i(w,x)_j}{L}\right)$. Then $f(w,x)$ has second partial derivatives bounded by $\max\left\{\frac{d\pi MLB_2 + d^2\pi^2 MB_1^2}{L^2}, \frac{d\pi B_1}{L}\right\}$.

Proof. Recall from the proof of Lemma A.6 that taking the derivative of $f(w,x)$ with respect to a_d yields

$$\frac{\partial f(w,x)}{\partial a_d} = \cos\left(\frac{d\pi g_i(w,x)_j}{L}\right),$$

and that if we take the derivative of $f(w,x)$ with respect to any weight from elsewhere in the network – some z – we get

$$\frac{\partial f(w,x)}{\partial z} = -\frac{d\pi a_d}{L}\frac{\partial g_i(w,x)_j}{\partial z}\sin\left(\frac{d\pi g_i(w,x)_j}{L}\right)$$

Now, if we take the derivative of $\frac{\partial f(w,x)}{\partial a_d}$ with respect to a_d we get 0. If we take the derivative of $\frac{\partial f(w,x)}{\partial a_d}$ with respect to some z we get

$$\frac{\partial^2 f(w,x)}{\partial a_d \partial z} = -\sin\left(\frac{d\pi g_i(w,x)_j}{L}\right)\left(\frac{d\pi}{L}\frac{\partial g_i(w,x)_j}{\partial z}\right)$$

and we can bound this as

$$\left|\frac{\partial^2 f(w,x)}{\partial a_d \partial z}\right| \leq \frac{d\pi B_1}{L}.$$

Next if we take the derivative of $\frac{\partial f(w,x)}{\partial z}$ with respect to a_d we get the same as above. Lastly we take the derivative of $\frac{\partial f(w,x)}{\partial z}$ with respect to some z' which gives

$$\frac{\partial^2 f(w,x)}{\partial z \partial z'} = -\frac{d\pi a_d}{L}\frac{\partial^2 g_i(w,x)_j}{\partial z \partial z'}\sin\left(\frac{d\pi g_i(w,x)_j}{L}\right) - \frac{d^2\pi^2 a_d}{L^2}\frac{\partial g_i(w,x)_j}{\partial z}\frac{\partial g_i(w,x)_j}{\partial z'}\cos\left(\frac{d\pi g_i(w,x)_j}{L}\right),$$

which we can bound as

$$\left|\frac{\partial^2 f(w,x)}{\partial z \partial z'}\right| \leq \frac{d\pi MB_2}{L} + \frac{d^2\pi^2 MB_1^2}{L^2} = \frac{d\pi MLB_2 + d^2\pi^2 MB_1^2}{L^2}.$$

Therefore the second partial derivatives of f are bounded by $\max\left\{\frac{d\pi MLB_2 + d^2\pi^2 MB_1^2}{L^2}, \frac{d\pi B_1}{L}\right\}$. □

Lemma A.14. Let $g_i(w,x) \in [-a,a]^{p_{i-1}}$ and assume

- $g_i(w,x)_j$ has first partial derivatives bounded by B_1,
- $g_i(w,x)_j$ has first partial derivatives bounded by B_2,
- All layers in the network have $\mathrm{NPF}_\epsilon(L,T)$ activations and that coefficients in these activations are bounded by M,
- For all i and j, $||W_{i,j}||_2 \leq N$, and
- For all i and j, $|b_{i,j}| \leq O$.

Consider the layer-wide activation transformation $a_i(w,x)$ where $a_i(w,x)_j$ is given by a function of the type $\mathrm{NPF}_\epsilon(L,T)$. $a_i(w,x)_j$ then has second partial derivatives bounded by

$$B' = \mathcal{O}\left(\frac{B_2^2 k^2}{\epsilon} + \frac{B_1 k^2}{\epsilon} + \frac{\sqrt{p_{i-1}}k^3(B_1+B_2)}{L\epsilon} + \frac{k^4 B_1^2}{L^2\epsilon} + \frac{k^3 B_1^2}{L\epsilon}\right).$$

Proof. Without loss of generality we can assume that if $B_i \neq 0$ that $B_i \geq 1$, as these are upper bounds. To bound the second partial derivatives, first note that Cases 1 and 5 have partial second

derivatives of 0 with respect to all parameters in the network. Next, in Case 3 we see from Lemma A.13 that

$$\begin{aligned}\left|\frac{\partial^2 M(g_i(w,x)_j)}{\partial z \partial z'}\right| &\leq 2\sum_{d=1}^{k}\left(\frac{d\pi MLB_2 + d^2\pi^2 MB_1^2}{L^2} + \frac{d\pi B_1}{L}\right)\\ &= \frac{(k^2+k)\pi MLB_2}{L^2} + \left(\frac{k^3}{3}+\frac{k^2}{2}+\frac{k}{6}\right)\frac{\pi^2 MB_1^2}{L^2} + \frac{(k^2+k)\pi B_1}{L}\\ &= \mathcal{O}\left(\frac{k^2(B_1+B_2)}{L} + \frac{k^3 B_1^2}{L^2}\right)\end{aligned}$$

for any two parameters z and z'. Denote

$$c_1 = L - T + \epsilon \text{ and } c_2 = -L + T + \epsilon.$$

In Case 5 then, we can compute

$$\begin{aligned}\frac{\partial^2 a_i(w,x)_j}{\partial z \partial z'} &= \frac{\partial}{\partial z'}\left(\frac{\partial g_i(w,x)_j}{\partial z'}\frac{M(g_i(w,x)_j)}{2\epsilon} + \frac{\partial M(g_i(w,x)_j)}{\partial z'}\frac{g_i(w,x)_j + c_1}{2\epsilon}\right.\\ &\quad \left. - \frac{\partial g_i(w,x)_j}{\partial z'}\frac{L(g_i(w,x)_j)}{2\epsilon}\right)\\ &= \frac{\partial^2 g_i(w,x)_j}{\partial z \partial z'}\frac{M(g_i(w,x)_j)}{2\epsilon} + \frac{\partial g_i(w,x)_j}{\partial z'}\frac{\partial M(g_i(w,x)_j)}{\partial z}\frac{1}{2\epsilon}\\ &\quad + \frac{\partial g_i(w,x)_j}{\partial z}\frac{\partial M(g_i(w,x)_j)}{\partial z'}\frac{1}{2\epsilon} + \frac{\partial^2 M(g_i(w,x)_j)}{\partial z \partial z'}\frac{g_i(w,x)_j + c_1}{2\epsilon}\\ &\quad - \frac{\partial^2 g_i(w,x)_j}{\partial z \partial z'}\frac{L(g_i(w,x)_j)}{2\epsilon}\end{aligned}$$

and then bound it, using B_3 and B_4 as bounds on the first and second partial derivatives of M, respectively,

$$\begin{aligned}\left|\frac{\partial^2 a_i(w,x)_j}{\partial z \partial z'}\right| &\leq \frac{B_2^2}{\epsilon}(k^2+k+1)M + \frac{B_1 B_3}{\epsilon} + B_4\frac{a+c_1}{2\epsilon}\\ &= \mathcal{O}\left(\frac{B_2^2 k^2}{\epsilon} + \frac{B_1 k^2}{\epsilon} + \frac{\sqrt{p_{i-1}}k^3(B_1+B_2)}{L\epsilon} + \frac{k^4 B_1^2}{L^2\epsilon} + \frac{k^3 B_1^2}{L\epsilon} + \frac{k^2(B_1+B_2)}{L}\right.\\ &\quad \left. + \frac{k^3 B_1^2}{L^2}\right)\\ &= \mathcal{O}\left(\frac{B_2^2 k^2}{\epsilon} + \frac{B_1 k^2}{\epsilon} + \frac{\sqrt{p_{i-1}}k^3(B_1+B_2)}{L\epsilon} + \frac{k^4 B_1^2}{L^2\epsilon} + \frac{k^3 B_1^2}{L\epsilon}\right).\end{aligned}$$

Since this bound dominates the one for Case 3, we are done. □

Combining all the results above we have Theorem A.15.

Theorem A.15. Consider a fully connected network with a softmax output layer and $\mathrm{NPF}_\epsilon(L,T)$ activation functions. Assume we use SGD with a max-norm constraint on the weight matrix and biases in the affine layers and on the coefficients of the activation functions to train the network. Then there exists some K such that the loss function as given in Section A is β-smooth. Specifically, for any such network where the number of hidden layers $Q \geq 1$ and and there are p nodes per layer,

$$\beta = \max\left\{\mathcal{O}\left(Q\frac{p^{3Q+5/2}k^{9Q-2}}{L^{2Q}\epsilon^{3Q-1}}\right), \mathcal{O}\left(Q\frac{p^{2^Q+Q+2}k^{5\times 2^Q+3Q}L^{2^{Q-1}+1}}{\epsilon^{2^Q+Q}}\right)\right\}.$$

Proof. By Lemmas A.3, A.4, A.7, A.10, A.11, A.14 and A.12 such a β must exist.

To compute this β, the aforementioned results can be used as well as the computations from the proof of Theorem A.9. The computation is as follows, using the aforementioned results,

- $g_1 - B = 0$
- $a_1 - B = \max\left\{\mathcal{O}\left(\frac{\sqrt{p}k^3}{L\epsilon} + \frac{k^4}{L^2\epsilon}\right), \mathcal{O}\left(\frac{k^2}{\epsilon} + \frac{\sqrt{p}k^3}{L\epsilon} + \frac{k^4}{L^2\epsilon}\right)\right\}$

- $g_2 - B = \max\left\{\mathcal{O}\left(\frac{pk^3}{L\epsilon} + \frac{\sqrt{p}k^4}{L^2\epsilon}\right), \mathcal{O}\left(\frac{\sqrt{p}k^3}{\epsilon} + \frac{k^2L}{\epsilon} + \frac{pk^3}{L\epsilon} + \frac{\sqrt{p}k^4}{L^2\epsilon}\right)\right\}$
- $a_2 - B = \max\left\{\mathcal{O}\left(\frac{p^2k^10}{L^4\epsilon^3}\right), \mathcal{O}\left(\frac{p^2k^10L^2}{\epsilon^2}\right)\right\}$
- $g_3 - B = \max\left\{\mathcal{O}\left(\frac{p^4k^16}{L^6\epsilon^5}\right), \mathcal{O}\left(\frac{p^{5/2}k^10L^2}{\epsilon^2}\right)\right\}$

Above, next to each transformation, is the bound B on its second partial derivatives. The bounds are computed with respect to $L \leq 1$ and $L > 1$, and then the maximum is taken over both cases. By a simple inductive argument for $i \geq 2$,

- $g_i -$
$$B = \max\left\{\mathcal{O}\left(\frac{p^{2i-3/2}k^{6i-8}}{L^{2i-2}\epsilon^{2i-3}}\right), \mathcal{O}\left(\frac{p^{2^{i-1}-1/2}k^{5\times 2^{i-1}}L^{2^{i-2}}}{\epsilon^{2^{i-1}}}\right)\right\}$$
- $a_i -$
$$B = \max\left\{\mathcal{O}\left(\frac{p^{2i-2}k^{6i-2}}{L^{2i}\epsilon^{2i-1}}\right), \mathcal{O}\left(\frac{p^{2^i-1}k^{5\times 2^i}L^{2^{i-1}}}{\epsilon^{2^i}}\right)\right\}$$

Using the bounds derived above, the bounds from Theorem A.9 and Lemma A.10 gives
$$\beta = \max\left\{\mathcal{O}\left(Q\frac{p^{3Q+5/2}k^{9Q-2}}{L^{2Q}\epsilon^{3Q-1}}\right), \mathcal{O}\left(Q\frac{p^{2^Q+Q+2}k^{5\times 2^Q+3Q}L^{2^{Q-1}+1}}{\epsilon^{2^Q+Q}}\right)\right\}.$$

□

A.3 GENERALIZABILITY

With the necessary properties demonstrated, we arrive at Theorem A.16. By p denote the number of nodes per layer and Q the number of hidden layers.

Theorem A.16. Consider a fully connected network with a softmax output layer and $\mathrm{NPF}_\epsilon(L, T)$ activation functions. Assume we use SGD with a max-norm constraint on the weight matrix and biases in the affine layers and on the coefficients of the activation functions to train the network. Then, for all practical values of L,T,ϵ,Q and p, the resulting Lipschitz constant K and smoothness constant β are sufficiently large that
$$\epsilon_{stab} \leq \frac{T}{n-1}.$$
Thus if $T = \mathcal{O}(\sqrt{n})$, $\epsilon_{stab} \to 0$ as sample size $n \to \infty$.

Proof. By Theorems 4.5, A.9, and A.15 the result is immediate. □

B RESULTS FOR FUNCTIONS OVER $\mathbb{R}^n$

Without proof we give the mean value inequality for vector valued function in Theorem B.1.

Theorem B.1. Let f be a differentiable vector valued function, that is $f : A \subset \mathbb{R}^n \to \mathbb{R}^m$. Then for $x, y \in A$, we have that
$$||f(x) - f(y)|| \leq ||Df(z)(x - y)||,$$
where z is some point on the line segment connecting x, y. Here Df represents the Frechet derivative of f, which can be represented in matrix form by the Jacobian.

C RESULTS FOR MATRICES

When we want to bound the largest eigenvalue of the matrix, we can get a loose bound by the sum of the entries of the matrix as in Theorems C.1 and C.2.

Theorem C.1. For some matrix norm $||\cdot||$, the spectral radius of the matrix A is upper bounded by $||A||$.

Proof. Take the eigenvector corresponding to the largest eigenvalue in magnitude of A. Denote this pair by x, λ. Next define the matrix $X = [x|\cdots|x]$. It follows then that $AX = \lambda X$. Therefore

$$|\lambda|||X|| = ||\lambda X|| = ||AX|| \leq ||A||||X||$$

because matrix norms are sub-multiplicative, demonstrating $|\lambda| \leq ||A||$. □

Theorem C.2. Let A be a symmetric $n \times n$ matrix. Let the entries of the matrix be bounded above by some constant k. It follows then that $\max_k |\lambda_k(A)| \leq nk$.

Proof. By Theorem C.1 we can see that

$$\max_k |\lambda_k(A)| \leq \sqrt{\sum_{i,j}(A_{ij})^2} \leq \sqrt{n^2k^2} = nk.$$

□

D RESULTS FOR LIPSCHITZ AND SMOOTH FUNCTIONS

First note that if $\nabla f(x)$ is well defined everywhere and if $||\nabla f(x)|| \leq L$ over the entire domain, it is L-Lipschitz by the mean value theorem and an application of Cauchy-Schwartz. In the convex case, the reverse is also true.

Lemma D.1. Take g to be a function from $\mathbb{R}^n \to \mathbb{R}^d$. If g is L-Lipschitz in each of its components, g is $\sqrt{n}L$-Lipschitz.

Proof. Consider any $x, y \in \mathbb{R}^n$. By the assumption we know that for $i = 1, \ldots, n$ $|g_i(x) - g_i(y)|^2 \leq L^2||x - y||_2^2$. Therefore

$$\sum_{i=1}^{n} |g_i(x) - g_i(y)|^2 \leq nL^2||x - y||_2^2,$$

and, thus, the desired result that

$$||g(x) - g(y)||_2 = \sqrt{\sum_{i=1}^{n} |g_i(x) - g_i(y)|^2} \leq \sqrt{n}L||x - y||_2.$$

□

Directly from the definition it is difficult to prove a gradient to be Lipschitz continuous. However we can utilize Lemma D.2.

Lemma D.2. For some twice differentiable function $f : A \subset \mathbb{R}^n \to \mathbb{R}$, f is β-smooth if its Hessian is bounded by β with respect to the euclidean norm.

Proof. Take the gradient ∇f which is a mapping $A \subset \mathbb{R}^n \to \mathbb{R}^n$. We can apply the mean value inequality for vector valued functions, Theorem B.1, which gives us

$$||\nabla f(x) - \nabla f(y)||_2 \leq ||\nabla^2 f(z)(x - y)||_2,$$

where z is on the line segment connecting x and y. Because of how the induced euclidean norm on a matrix is defined it follows that

$$||\nabla^2 f(z)(x - y)||_2 \leq ||\nabla^2 f(z)||_2||x - y||_2 \leq \beta||x - y||_2.$$

Combining the two equations above gives the desired result,

$$||\nabla f(x) - \nabla f(y)||_2 \leq \beta||x - y||_2.$$

□

The insight to take this approach to demonstrating β-smoothness is from Bach & Moulines (2011).