On the Expressive Power of Deep Neural Networks

Maithra Raghu [1 2] **Ben Poole** [3] **Jon Kleinberg** [1] **Surya Ganguli** [3] **Jascha Sohl Dickstein** [2]

Abstract

We propose a new approach to the problem of *neural network expressivity*, which seeks to characterize how structural properties of a neural network family affect the functions it is able to compute. Our approach is based on an interrelated set of measures of expressivity, unified by the novel notion of *trajectory length*, which measures how the output of a network changes as the input sweeps along a one-dimensional path. Our findings show that: (1) The complexity of the computed function grows exponentially with depth (2) All weights are not equal: trained networks are more sensitive to their lower (initial) layer weights (3) Trajectory regularization is a simpler alternative to batch normalization, with the same performance.

1. Introduction

Deep neural networks have proved astoundingly effective at a wide range of empirical tasks, from image classification (Krizhevsky et al., 2012) to playing Go (Silver et al., 2016), and even modeling human learning (Piech et al., 2015).

Despite these successes, understanding of how and why neural network architectures achieve their empirical successes is still lacking. This includes even the fundamental question of *neural network expressivity*, how the architectural properties of a neural network (depth, width, layer type) affect the resulting functions it can compute, and its ensuing performance.

This is a foundational question, and there is a rich history of prior work addressing expressivity in neural networks. However, it has been challenging to derive conclusions that provide both theoretical generality with respect to choices of architecture as well as meaningful insights into practical performance.

Indeed, the very first results on this question take a highly theoretical approach, from using functional analysis to show universal approximation results (Hornik et al., 1989; Cybenko, 1989), to analysing expressivity via comparisons to boolean circuits (Maass et al., 1994) and studying network VC dimension (Bartlett et al., 1998). While these results provided theoretically general conclusions, the shallow networks they studied are very different from the deep models that have proven so successful in recent years.

In response, several recent papers have focused on understanding the benefits of depth for neural networks (Pascanu et al., 2013; Montufar et al., 2014; Eldan and Shamir, 2015; Telgarsky, 2015; Martens et al., 2013; Bianchini and Scarselli, 2014). These results are compelling and take modern architectural changes into account, but they only show that a specific choice of weights for a deeper network results in inapproximability by a shallow (typically one or two hidden layers) network.

In particular, the goal of this new line of work has been to establish lower bounds — showing separations between shallow and deep networks — and as such they are based on hand-coded constructions of specific network weights. Even if the weight values used in these constructions are robust to small perturbations (as in (Pascanu et al., 2013; Montufar et al., 2014)), the functions that arise from these constructions tend toward extremal properties by design, and there is no evidence that a network trained on data ever resembles such a function.

This has meant that a set of fundamental questions about neural network expressivity has remained largely unanswered. First, we lack a good understanding of the "typical" case rather than the worst case in these bounds for deep networks, and consequently have no way to evaluate whether the hand-coded extremal constructions provide a reflection of the complexity encountered in more standard settings. Second, we lack an understanding of upper bounds to match the lower bounds produced by this prior work; do the constructions used to date place us near the limit of the expressive power of neural networks, or are there still large gaps? Finally, if we had an understanding of these two issues, we might begin to draw connections between network expressivity and observed performance.

[1]Cornell University [2]Google Brain [3]Stanford University. Correspondence to: Maithra Raghu <maithrar@gmail.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

Our contributions: Measures of Expressivity and their Applications In this paper, we address this set of challenges by defining and analyzing an interrelated set of *measures of expressivity* for neural networks; our framework applies to a wide range of standard architectures, independent of specific weight choices. We begin our analysis at the start of training, after random initialization, and later derive insights connecting network expressivity and performance.

Our first measure of expressivity is based on the notion of an *activation pattern*: in a network where the units compute functions based on discrete thresholds, we can ask which units are above or below their thresholds (i.e. which units are "active" and which are not). For the range of standard architectures that we consider, the network is essentially computing a linear function once we fix the activation pattern; thus, counting the number of possible activation patterns provides a concrete way of measuring the complexity beyond linearity that the network provides. We give an upper bound on the number of possible activation patterns, over any setting of the weights. This bound is tight as it matches the hand-constructed lower bounds of earlier work (Pascanu et al., 2013; Montufar et al., 2014).

Key to our analysis is the notion of a *transition*, in which changing an input x to a nearby input $x + \delta$ changes the activation pattern. We study the behavior of transitions as we pass the input along a one-dimensional parametrized trajectory $x(t)$. Our central finding is that the *trajectory length* grows exponentially in the depth of the network.

Trajectory length serves as a unifying notion in our measures of expressivity, and it leads to insights into the behavior of trained networks. Specifically, we find that the exponential growth in trajectory length as a function of depth implies that small adjustments in parameters lower in the network induce larger changes than comparable adjustments higher in the network. We demonstrate this phenomenon through experiments on MNIST and CIFAR-10, where the network displays much less robustness to noise in the lower layers, and better performance when they are trained well. We also explore the effects of regularization methods on trajectory length as the network trains and propose a less computationally intensive method of regularization, *trajectory regularization*, that offers the same performance as batch normalization.

The contributions of this paper are thus:

(1) *Measures of expressivity*: We propose easily computable measures of neural network expressivity that capture the expressive power inherent in different neural network architectures, independent of specific weight settings.

(2) *Exponential trajectories:* We find an exponential depth dependence displayed by these measures, through a unifying analysis in which we study how the network transforms its input by measuring *trajectory length*

(3) *All weights are not equal (the lower layers matter more)*: We show how these results on trajectory length suggest that optimizing weights in lower layers of the network is particularly important.

(4) *Trajectory Regularization* Based on understanding the effect of batch norm on trajectory length, we propose a new method of regularization, trajectory regularization, that offers the same advantages as batch norm, and is computationally more efficient.

In prior work (Poole et al., 2016), we studied the propagation of *Riemannian curvature* through random networks by developing a mean field theory approach. Here, we take an approach grounded in computational geometry, presenting measures with a combinatorial flavor and explore the consequences during and after training.

2. Measures of Expressivity

Given a neural network of a certain architecture A (some depth, width, layer types), we have an associated function, $F_A(x; W)$, where x is an input and W represents all the parameters of the network. Our goal is to understand how the behavior of $F_A(x; W)$ changes as A changes, for values of W that we might encounter during training, and across inputs x.

The first major difficulty comes from the high dimensionality of the input. Precisely quantifying the properties of $F_A(x; W)$ over the entire input space is intractable. As a tractable alternative, we study simple one dimensional *trajectories* through input space. More formally:

Definition: Given two points, $x_0, x_1 \in \mathbb{R}^m$, we say $x(t)$ is a trajectory (between x_0 and x_1) if $x(t)$ is a curve parametrized by a scalar $t \in [0, 1]$, with $x(0) = x_0$ and $x(1) = x_1$.

Simple examples of a trajectory would be a line ($x(t) = tx_1 + (1 - t)x_0$) or a circular arc ($x(t) = \cos(\pi t/2)x_0 + \sin(\pi t/2)x_1$), but in general $x(t)$ may be more complicated, and potentially not expressible in closed form.

Armed with this notion of trajectories, we can begin to define measures of expressivity of a network $F_A(x; W)$ over trajectories $x(t)$.

2.1. Neuron Transitions and Activation Patterns

In (Montufar et al., 2014) the notion of a "linear region" is introduced. Given a neural network with piecewise lin-

ear activations (such as ReLU or hard tanh), the function it computes is also piecewise linear, a consequence of the fact that composing piecewise linear functions results in a piecewise linear function. So one way to measure the "expressive power" of different architectures A is to count the number of linear pieces (regions), which determines how nonlinear the function is.

In fact, a change in linear region is caused by a *neuron transition* in the output layer. More precisely:

Definition For fixed W, we say a neuron with piecewise linear region *transitions* between inputs $x, x+\delta$ if its activation function switches linear region between x and $x+\delta$.

So a ReLU transition would be given by a neuron switching from off to on (or vice versa) and for hard tanh by switching between saturation at -1 to its linear middle region to saturation at 1. For any generic trajectory $x(t)$, we can thus define $\mathcal{T}(F_A(x(t); W))$ to be the number of transitions undergone by output neurons (i.e. the number of linear regions) as we sweep the input $x(t)$. Instead of just concentrating on the output neurons however, we can look at this pattern over the *entire* network. We call this an *activation patten*:

Definition We can define $\mathcal{AP}(F_A(x; W))$ to be the *activation pattern* – a string of form $\{0,1\}^{\text{num neurons}}$ (for ReLUs) and $\{-1,0,1\}^{\text{num neurons}}$ (for hard tanh) of the network encoding the linear region of the activation function of *every* neuron, for an input x and weights W.

Overloading notation slightly, we can also define (similarly to transitions) $\mathcal{A}(F_A(x(t); W))$ as the number of distinct activation patterns as we sweep x along $x(t)$. As each distinct activation pattern corresponds to a different linear function of the input, this combinatorial measure captures how much more expressive A is over a simple linear mapping.

Returning to Montufar et al, they provide a construction i.e. a *specific* set of weights W_0, that results in an exponential increase of linear regions with the depth of the architectures. They also appeal to Zaslavsky's theorem (Stanley, 2011) from the theory of hyperplane arrangements to show that a shallow network, i.e. *one* hidden layer, with the same number of parameters as a deep network, has a much smaller number of linear regions than the number achieved by their choice of weights W_0 for the deep network.

More formally, letting A_1 be a fully connected network with one hidden layer, and A_l a fully connected network with the same number of parameters, but l hidden layers, they show

$$\forall W \mathcal{T}(F_{A_1}([0,1]; W)) < \mathcal{T}(F_{A_1}([0,1]; W_0) \qquad (*)$$

We derive a much more general result by considering the 'global' activation patterns over the *entire* input space, and prove that for any fully connected network, with *any* number of hidden layers, we can upper bound the number of linear regions it can achieve, over *all* possible weight settings W. This upper bound is asymptotically *tight*, matched by the construction given in (Montufar et al., 2014). Our result can be written formally as:

Theorem 1. (Tight) Upper Bound for Number of Activation Patterns *Let $A_{(n,k)}$ denote a fully connected network with n hidden layers of width k, and inputs in $\mathbb{R}^m$. Then the number of activation patterns $\mathcal{A}(F_{A_{n,k}}(\mathbb{R}^m; W)$ is upper bounded by $O(k^{mn})$ for ReLU activations, and $O((2k)^{mn})$ for hard tanh.*

From this we can derive a *chain* of inequalities. Firstly, from the theorem above we find an upper bound of $\mathcal{A}(F_{A_{n,k}}(\mathbb{R}^m; W))$ over all W, i.e.

$$\forall W \ \mathcal{A}(F_{A_{(n,k)}})(\mathbb{R}^m; W) \leq U(n,k,m).$$

Next, suppose we have N neurons in total. Then we want to compare (for wlog ReLUs), quantities like $U(n', N/n', m)$ for different n'.

But $U(n', N/n', m) = O((N/n')^{mn'})$, and so, noting that the maxima of $\left(\frac{a}{x}\right)^{mx}$ (for $a > e$) is $x = a/e$, we get, (for $n, k > e$), in comparison to (*),

$$U(1, N, m) < U(2, \frac{N}{2}, m) < \cdots$$

$$\cdots < U(n-1, \frac{N}{n-1}, m) < U(n, k, m)$$

We prove this via an inductive proof on regions in a hyperplane arrangement. The proof can be found in the Appendix. As noted in the introduction, this result differs from earlier lower-bound constructions in that it is an upper bound that applies to *all* possible sets of weights. Via our analysis, we also prove

Theorem 2. Regions in Input Space *Given the corresponding function of a neural network $F_A(\mathbb{R}^m; W)$ with ReLU or hard tanh activations, the input space is partitioned into convex polytopes, with $F_A(\mathbb{R}^m; W)$ corresponding to a different linear function on each region.*

This result is of independent interest for optimization – a linear function over a convex polytope results in a well behaved loss function and an easy optimization problem. Understanding the density of these regions during the training process would likely shed light on properties of the loss surface, and improved optimization methods. A picture of a network's regions is shown in Figure 1.

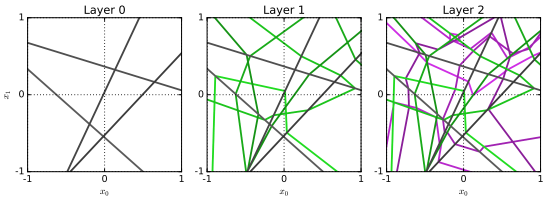

Figure 1. Deep networks with piecewise linear activations subdivide input space into convex polytopes. We take a three hidden layer ReLU network, with input $x \in \mathbb{R}^2$, and four units in each layer. The left pane shows activations for the first layer only. As the input is in $\mathbb{R}^2$, neurons in the first hidden layer have an associated line in $\mathbb{R}^2$, depicting their activation boundary. The left pane thus has four such lines. For the second hidden layer each neuron again has a line in input space corresponding to on/off, but this line is *different* for each region described by the first layer activation pattern. So in the centre pane, which shows activation boundary lines corresponding to second hidden layer neurons in green (and first hidden layer in black), we can see the green lines 'bend' at the boundaries. (The reason for this bending becomes apparent through the proof of Theorem 2.) Finally, the right pane adds the on/off boundaries for neurons in the third hidden layer, in purple. These lines can bend at both black and green boundaries, as the image shows. This final set of convex polytopes corresponds to all activation patterns for this network (with its current set of weights) over the unit square, with each polytope representing a different linear function.

2.1.1. Empirically Counting Transitions

We empirically tested the growth of the number of activations and transitions as we varied x along $x(t)$ to understand their behavior. We found that for bounded non linearities, especially tanh and hard-tanh, not only do we observe exponential growth with depth (as hinted at by the upper bound) but that the *scale* of parameter initialization also affects the observations (Figure 2). We also experimented with sweeping the *weights* W of a layer through a trajectory $W(t)$, and counting the different labellings output by the network. This 'dichotomies' measure is discussed further in the Appendix, and also exhibits the same growth properties, Figure 14.

2.2. Trajectory Length

In fact, there turns out to be a reason for the exponential growth with depth, and the sensitivity to initialization scale. Returning to our definition of trajectory, we can define an immediately related quantity, *trajectory length*

Definition: Given a trajectory, $x(t)$, we define its length, $l(x(t))$, to be the standard *arc length*:

$$l(x(t)) = \int_t \left|\left| \frac{dx(t)}{dt} \right|\right| dt$$

Intuitively, the arc length breaks $x(t)$ up into infinitesimal intervals and sums together the Euclidean length of these intervals.

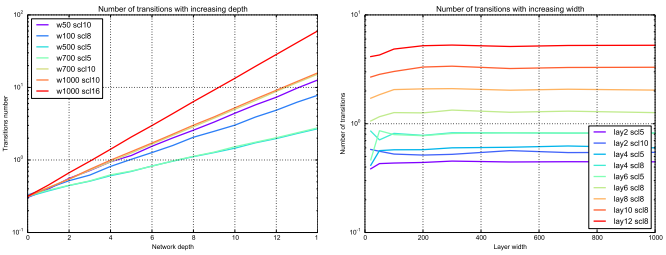

Figure 2. The number of transitions seen for fully connected networks of different widths, depths and initialization scales, with a circular trajectory between MNIST datapoints. The number of transitions grows exponentially with the depth of the architecture, as seen in (left). The same rate of growth is not seen with increasing architecture width, plotted in (right). There is a surprising dependence on the scale of initialization, explained in 2.2.

Figure 3. Picture showing a trajectory increasing with the depth of a network. We start off with a circular trajectory (left most pane), and feed it through a fully connected tanh network with width 100. Pane second from left shows the image of the circular trajectory (projected down to two dimensions) after being transformed by the first hidden layer. Subsequent panes show projections of the latent image of the circular trajectory after being transformed by more hidden layers. The final pane shows the the trajectory after being transformed by all the hidden layers.

If we let $A_{(n,k)}$ denote, as before, fully connected networks with n hidden layers each of width k, and initializing with weights $\sim \mathcal{N}(0, \sigma_w^2/k)$ (accounting for input scaling as typical), and biases $\sim \mathcal{N}(0, \sigma_b^2)$, we find that:

Theorem 3. Bound on Growth of Trajectory Length *Let $F_A(x', W)$ be a ReLU or hard tanh random neural network and $x(t)$ a one dimensional trajectory with $x(t+\delta)$ having a non trival perpendicular component to $x(t)$ for all t, δ (i.e, not a line). Then defining $z^{(d)}(x(t)) = z^{(d)}(t)$ to be the image of the trajectory in layer d of the network, we have*

(a)

$$\mathbb{E}\left[l(z^{(d)}(t))\right] \geq O\left(\frac{\sigma_w\sqrt{k}}{\sqrt{k+1}}\right)^d l(x(t))$$

for ReLUs

(b)

$$\mathbb{E}\left[l(z^{(d)}(t))\right] \geq O\left(\frac{\sigma_w\sqrt{k}}{\sqrt{\sigma_w^2+\sigma_b^2+k\sqrt{\sigma_w^2+\sigma_b^2}}}\right)^d l(x(t))$$

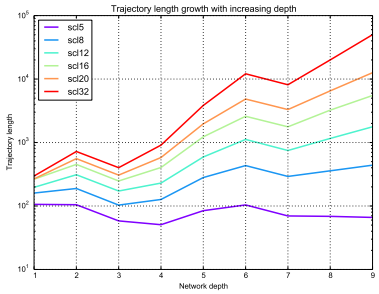

Figure 4. We look at trajectory growth with different initialization scales as a trajectory is propagated through a convolutional architecture for CIFAR-10, with ReLU activations. The analysis of Theorem 3 was for fully connected networks, but we see that trajectory growth holds (albeit with slightly higher scales) for convolutional architectures also. Note that the decrease in trajectory length, seen in layers 3 and 7 is expected, as those layers are pooling layers.

for hard tanh

That is, $l(x(t)$ grows *exponentially* with the depth of the network, but the width only appears as a base (of the exponent). This bound is in fact *tight* in the limits of large σ_w and k.

A schematic image depicting this can be seen in Figure 3 and the proof can be found in the Appendix. A rough outline is as follows: we look at the expected growth of the difference between a point $z^{(d)}(t)$ on the curve and a small perturbation $z^{(d)}(t+dt)$, from layer d to layer $d+1$. Denoting this quantity $\left|\left|\delta z^{(d)}(t)\right|\right|$, we derive a recurrence relating $\left|\left|\delta z^{(d+1)}(t)\right|\right|$ and $\left|\left|\delta z^{(d)}(t)\right|\right|$ which can be composed to give the desired growth rate.

The analysis is complicated by the statistical dependence on the image of the input $z^{(d+1)}(t)$. So we instead form a recursion by looking at the component of the difference perpendicular to the image of the input in that layer, i.e. $\left|\left|\delta z_{\perp}^{(d+1)}(t)\right|\right|$, which results in the condition on $x(t)$ in the statement.

In Figures 4, 12, we see the growth of an input trajectory for ReLU networks on CIFAR-10 and MNIST. The CIFAR-10 network is convolutional but we observe that these layers also result in similar rates of trajectory length increases to the fully connected layers. We also see, as would be expected, that pooling layers act to *reduce* the trajectory length. We discuss upper bounds in the Appendix.

For the hard tanh case (and more generally any bounded non-linearity), we can formally prove the relation of trajectory length and transitions under an assumption: assume that while we sweep $x(t)$ all neurons are saturated unless transitioning saturation endpoints, which happens very

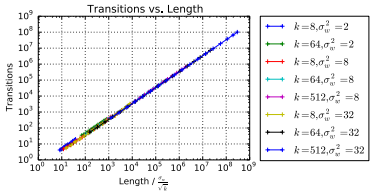

Figure 5. The number of transitions is linear in trajectory length. Here we compare the empirical number of transitions to the length of the trajectory, for different depths of a hard-tanh network. We repeat this comparison for a variety of network architectures, with different network width k and weight variance σ_w^2.

rapidly. (This is the case for e.g. large initialization scales). Then we have:

Theorem 4. Transitions proportional to trajectory length *Let $F_{A_{n,k}}$ be a hard tanh network with n hidden layers each of width k. And let*

$$g(k, \sigma_w, \sigma_b, n) = O\left(\frac{\sqrt{k}}{\sqrt{1+\frac{\sigma_b^2}{\sigma_w^2}}}\right)^n$$

Then $\mathcal{T}(F_{A_{n,k}}(x(t); W) = O(g(k, \sigma_w, \sigma_b, n))$ for W initialized with weight and bias scales σ_w, σ_b.

Note that the expression for $g(k, \sigma_w, \sigma_b, n)$ is exactly the expression given by Theorem 3 when σ_w is very large and dominates σ_b. We can also verify this experimentally in settings where the simpilfying assumption does not hold, as in Figure 5.

3. Insights from Network Expressivity

Here we explore the insights gained from applying our measurements of expressivity, particularly trajectory length, to understand network performance. We examine the connection of expressivity and stability, and inspired by this, propose a new method of regularization, *trajectory regularization* that offers the same advantages as the more computationally intensive batch normalization.

3.1. Expressivity and Network Stability

The analysis of network expressivity offers interesting takeaways related to the parameter and functional stability of a network. From the proof of Theorem 3, we saw that a perturbation to the input would grow exponentially in the depth of the network. It is easy to see that this analysis is not limited to the input layer, but can be applied to *any* layer. In this form, it would say

A perturbation at a layer grows exponentially in the *remaining depth* after that layer.

This means that perturbations to weights in lower layers should be more costly than perturbations in the upper lay-

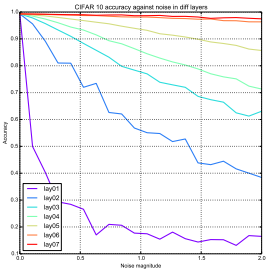

Figure 6. We then pick a single layer of a conv net trained to high accuracy on CIFAR10, and add noise to the layer weights of increasing magnitudes, testing the network accuracy as we do so. We find that the initial (lower) layers of the network are *least* robust to noise – as the figure shows, adding noise of 0.25 magnitude to the first layer results in a 0.7 drop in accuracy, while the same amount of noise added to the fifth layer barely results in a 0.02 drop in accuracy. This pattern is seen for many different initialization scales, even for a (typical) scaling of $\sigma_w^2 = 2$, used in the experiment.

ers, due to exponentially increasing magnitude of noise, and result in a much larger drop of accuracy. Figure 6, in which we train a conv network on CIFAR-10 and add noise of varying magnitudes to exactly one layer, shows exactly this.

We also find that the converse (in some sense) holds: after initializing a network, we trained a single layer at different depths in the network and found monotonically increasing performance as layers lower in the network were trained. This is shown in Figure 7 and Figure 17 in the Appendix.

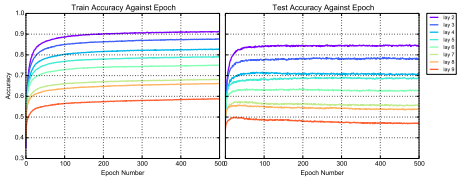

Figure 7. Demonstration of expressive power of remaining depth on MNIST. Here we plot train and test accuracy achieved by training exactly one layer of a fully connected neural net on MNIST. The different lines are generated by varying the hidden layer chosen to train. All other layers are kept frozen after random initialization. We see that training lower hidden layers leads to better performance. The networks had width $k = 100$, weight variance $\sigma_w^2 = 1$, and hard-tanh nonlinearities. Note that we only train from the second hidden layer (weights $W^{(1)}$) onwards, so that the number of parameters trained remains fixed.

3.2. Trajectory Length and Regularization: The Effect of Batch Normalization

Expressivity measures, especially trajectory length, can also be used to better understand the effect of regularization. One regularization technique that has been extremely successful for training neural networks is Batch Normalization (Ioffe and Szegedy, 2015).

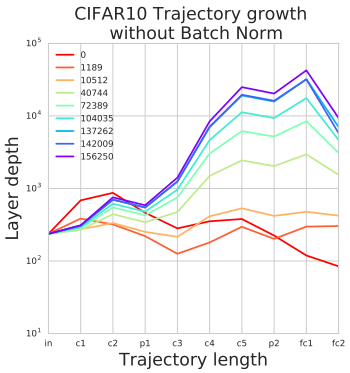

Figure 8. Training increases trajectory length even for typical ($\sigma_w^2 = 2$) initialization values of σ_w. Here we propagate a circular trajectory joining two CIFAR10 datapoints through a conv net without batch norm, and look at how trajectory length changes through training. We see that training causes trajectory length to increase exponentially with depth (exceptions only being the pooling layers and the final fc layer, which halves the number of neurons.) Note that at Step 0, the network is *not* in the exponential growth regime. We observe (discussed in Figure 9) that even networks that aren't initialized in the exponential growth regime can be pushed there through training.

By taking measures of trajectories during training we find that without batch norm, trajectory length tends to increase during training, as shown in Figures 8 and Figure 18 in the Appendix. In these experiments, two networks were initialized with $\sigma_w^2 = 2$ and trained to high test accuracy on CIFAR10 and MNIST. We see that in both cases, trajectory length increases as training progresses.

A surprising observation is $\sigma_w^2 = 2$ is not in the exponential growth increase regime at initialization for the CIFAR10 architecture (Figure 8 at Step 0.). But note that even with a smaller weight initialization, weight norms increase during training, shown in Figure 9, pushing typically initialized networks into the exponential growth regime.

While the initial growth of trajectory length enables greater functional expressivity, large trajectory growth in the learnt representation results in an unstable representation, witnessed in Figure 6. In Figure 10 we train another conv net on CIFAR10, but this time with batch normalization. We see that the batch norm layers reduce trajectory length, helping stability.

3.3. Trajectory Regularization

Motivated by the fact that batch normalization decreases trajectory length and hence helps stability and generalization, we consider directly regularizing on trajectory length:

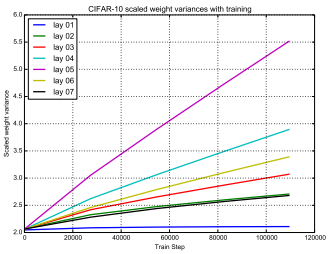

Figure 9. This figure shows how the weight scaling of a CIFAR10 network evolves during training. The network was initialized with $\sigma_w^2 = 2$, which increases across all layers during training.

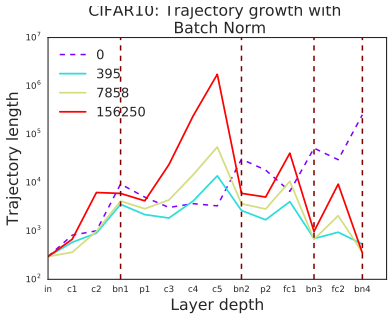

Figure 10. Growth of circular trajectory between two datapoints with batch norm layers for a conv net on CIFAR10. The network was initialized as typical, with $\sigma_w^2 = 2$. Note that the batch norm layers in Step 0 are poorly behaved due to division by a close to 0 variance. But after just a few hundred gradient steps and continuing onwards, we see the batch norm layers (dotted lines) reduce trajectory length, stabilising the representation without sacrificing expressivity.

we replace every batch norm layer used in the conv net in Figure 10 with a *trajectory regularization layer*. This layer adds to the loss $\lambda(\text{current length}/\text{orig length})$, and then scales the outgoing activations by λ, where λ is a parameter to be learnt. In implementation, we typically scale the additional loss above with a constant (0.01) to reduce magnitude in comparison to classification loss. Our results, Figure 11 show that both trajectory regularization and batch norm perform comparably, and considerably better than not using batch norm. One advantage of using Trajectory Regularization is that we don't require different computations to be performed for train and test, enabling more efficient implementation.

4. Discussion

Characterizing the expressiveness of neural networks, and understanding how expressiveness varies with parameters of the architecture, has been a challenging problem due to the difficulty in identifying meaningful notions of expressivity and in linking their analysis to implications for these networks in practice. In this paper we have presented an

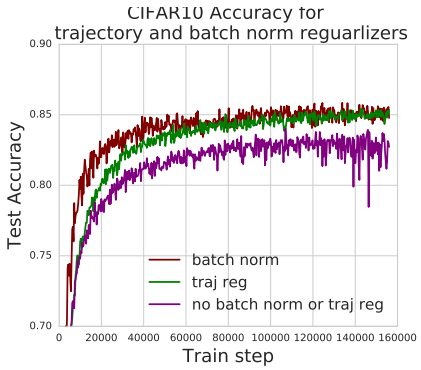

Figure 11. We replace each batch norm layer of the CIFAR10 conv net with a *trajectory regularization* layer, described in Section 3.3. During training trajectory length is easily calculated as a piecewise linear trajectory between adjacent datapoints in the minibatch. We see that trajectory regularization achieves the same performance as batch norm, albeit with slightly more train time. However, as trajectory regularization behaves the same during train and test time, it is simpler and more efficient to implement.

interrelated set of expressivity measures; we have shown tight exponential bounds on the growth of these measures in the depth of the networks, and we have offered a unifying view of the analysis through the notion of *trajectory length*. Our analysis of trajectories provides insights for the performance of trained networks as well, suggesting that networks in practice may be more sensitive to small perturbations in weights at lower layers. We also used this to explore the empirical success of batch norm, and developed a new regularization method – trajectory regularization.

This work raises many interesting directions for future work. At a general level, continuing the theme of 'principled deep understanding', it would be interesting to link measures of expressivity to other properties of neural network performance. There is also a natural connection between adversarial examples, (Goodfellow et al., 2014), and trajectory length: adversarial perturbations are only a small distance away in input space, but result in a large change in classification (the output layer). Understanding how trajectories between the original input and an adversarial perturbation grow might provide insights into this phenomenon. Another direction, partially explored in this paper, is regularizing based on trajectory length. A very simple version of this was presented, but further performance gains might be achieved through more sophisticated use of this method.

Acknowledgements

We thank Samy Bengio, Ian Goodfellow, Laurent Dinh, and Quoc Le for extremely helpful discussion.

References

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. In *Advances in Neural Information Processing Systems*, pages 505–513, 2015.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Wolfgang Maass, Georg Schnitger, and Eduardo D Sontag. *A comparison of the computational power of sigmoid and Boolean threshold circuits*. Springer, 1994.

Peter L Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear vc-dimension bounds for piecewise polynomial networks. *Neural computation*, 10(8):2159–2173, 1998.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *arXiv preprint arXiv:1512.03965*, 2015.

Matus Telgarsky. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

James Martens, Arkadev Chattopadhya, Toni Pitassi, and Richard Zemel. On the representational efficiency of restricted boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2877–2885, 2013.

Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(8):1553–1565, 2014.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

Richard Stanley. Hyperplane arrangements. *Enumerative Combinatorics*, 2011.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.

D. Kershaw. Some extensions of w. gautschi's inequalities for the gamma function. *Mathematics of Computation*, 41(164):607–611, 1983.

Andrea Laforgia and Pierpaolo Natalini. On some inequalities for the gamma function. *Advances in Dynamical Systems and Applications*, 8(2):261–267, 2013.

Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.