Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Abstract

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during training, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.

1 Deep Neural Networks

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical success in vision (Ranzato et al., 2007; Larochelle et al., 2007; Vincent et al., 2008) and natural language processing (NLP) (Collobert & Weston, 2008; Mnih & Hinton, 2009). Theoretical results reviewed and discussed by Bengio (2009), suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need *deep architectures*.

Most of the recent experimental results with deep architecture are obtained with models that can be turned into deep supervised neural networks, but with initialization or training schemes different from the classical feedforward neural networks (Rumelhart et al., 1986). Why are these new algorithms working so much better than the standard random initialization and gradient-based optimization of a supervised training criterion? Part of the answer may be found in recent analyses of the effect of unsupervised pre-training (Erhan et al., 2009), showing that it acts as a regularizer that initializes the parameters in a "better" basin of attraction of the optimization procedure, corresponding to an apparent local minimum associated with better generalization. But earlier work (Bengio et al., 2007) had shown that even a purely supervised but greedy layer-wise procedure would give better results. So here instead of focusing on what unsupervised pre-training or semi-supervised criteria bring to deep architectures, we focus on analyzing what may be going wrong with good old (but deep) multi-layer neural networks.

Our analysis is driven by investigative experiments to monitor activations (watching for saturation of hidden units) and gradients, across layers and across training iterations. We also evaluate the effects on these of choices of activation function (with the idea that it might affect saturation) and initialization procedure (since unsupervised pre-training is a particular form of initialization and it has a drastic impact).

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

2 Experimental Setting and Datasets

Code to produce the new datasets introduced in this section is available from: `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepGradientsAISTATS2010`.

2.1 Online Learning on an Infinite Dataset: Shapeset-3×2

Recent work with deep architectures (see Figure 7 in Bengio (2009)) shows that even with very large training sets or online learning, initialization from unsupervised pre-training yields substantial improvement, which does not vanish as the number of training examples increases. The online setting is also interesting because it focuses on the optimization issues rather than on the small-sample regularization effects, so we decided to include in our experiments a synthetic images dataset inspired from Larochelle et al. (2007) and Larochelle et al. (2009), from which as many examples as needed could be sampled, for testing the online learning scenario.

We call this dataset the **Shapeset-**3×2 dataset, with example images in Figure 1 (top). **Shapeset-**3×2 contains images of 1 or 2 two-dimensional objects, each taken from 3 shape categories (triangle, parallelogram, ellipse), and placed with random shape parameters (relative lengths and/or angles), scaling, rotation, translation and grey-scale.

We noticed that for only one shape present in the image the task of recognizing it was too easy. We therefore decided to sample also images with two objects, with the constraint that the second object does not overlap with the first by more than fifty percent of its area, to avoid hiding it entirely. The task is to predict the objects present (e.g. triangle + ellipse, parallelogram + parallelogram, triangle alone, etc.) without having to distinguish between the foreground shape and the background shape when they overlap. This therefore defines nine configuration classes.

The task is fairly difficult because we need to discover invariances over rotation, translation, scaling, object color, occlusion and relative position of the shapes. In parallel we need to extract the factors of variability that predict which object shapes are present.

The size of the images are arbitrary but we fixed it to 32×32 in order to work with deep dense networks efficiently.

2.2 Finite Datasets

The MNIST digits (LeCun et al., 1998a), dataset has 50,000 training images, 10,000 validation images (for hyper-parameter selection), and 10,000 test images, each showing a 28×28 grey-scale pixel image of one of the 10 digits.

CIFAR-10 (Krizhevsky & Hinton, 2009) is a labelled subset of the tiny-images dataset that contains 50,000 training examples (from which we extracted 10,000 as validation data) and 10,000 test examples. There are 10 classes corresponding to the main object in each image: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. The classes are balanced. Each image is in color, but is just 32×32 pixels in size, so the input is a vector of $32 \times 32 \times 3 = 3072$ real values.

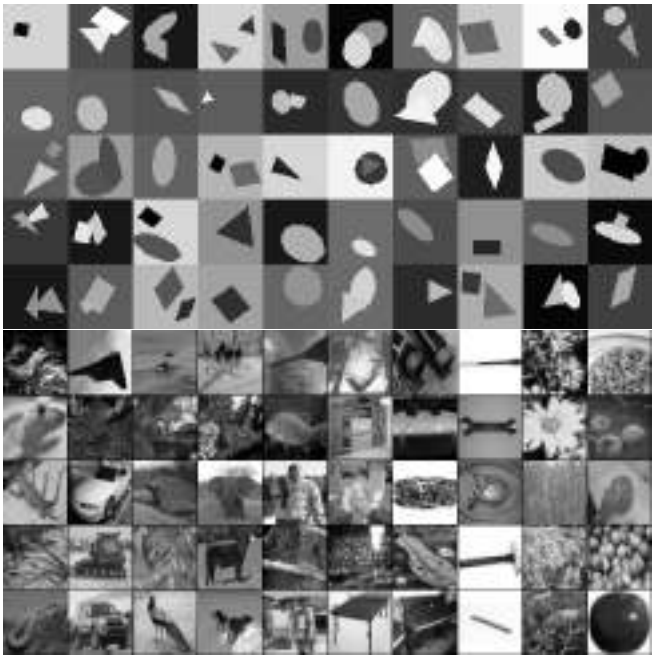

Figure 1: *Top: Shapeset-3×2 images at 64×64 resolution. The examples we used are at 32×32 resolution. The learner tries to predict which objects (parallelogram, triangle, or ellipse) are present, and 1 or 2 objects can be present, yielding 9 possible classifications. Bottom: Small-ImageNet images at full resolution.*

Small-ImageNet which is a set of tiny 37×37 gray level images dataset computed from the higher-resolution and larger set at `http://www.image-net.org`, with labels from the WordNet noun hierarchy. We have used 90,000 examples for training, 10,000 for the validation set, and 10,000 for testing. There are 10 balanced classes: reptiles, vehicles, birds, mammals, fish, furniture, instruments, tools, flowers and fruits Figure 1 (bottom) shows randomly chosen examples.

2.3 Experimental Setting

We optimized feedforward neural networks with one to five hidden layers, with one thousand hidden units per layer, and with a softmax logistic regression for the output layer. The cost function is the negative log-likelihood $-\log P(y|x)$, where (x, y) is the (input image, target class) pair. The neural networks were optimized with stochastic back-propagation on mini-batches of size ten, i.e., the average g of $\frac{\partial -\log P(y|x)}{\partial \theta}$ was computed over 10 consecutive

training pairs (x, y) and used to update parameters θ in that direction, with $\theta \leftarrow \theta - \epsilon g$. The learning rate ϵ is a hyper-parameter that is optimized based on validation set error after a large number of updates (5 million).

We varied the type of non-linear activation function in the hidden layers: the sigmoid $1/(1 + e^{-x})$, the hyperbolic tangent $\tanh(x)$, and a newly proposed activation function (Bergstra et al., 2009) called the softsign, $x/(1 + |x|)$. The softsign is similar to the hyperbolic tangent (its range is -1 to 1) but its tails are quadratic polynomials rather than exponentials, i.e., it approaches its asymptotes much slower.

In the comparisons, we search for the best hyper-parameters (learning rate and depth) separately for each model. Note that the best depth was always five for Shapeset-3 $\times$ 2, except for the sigmoid, for which it was four.

We initialized the biases to be 0 and the weights W_{ij} at each layer with the following commonly used heuristic:

$$W_{ij} \sim U\Big[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\Big], \qquad (1)$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and n is the size of the previous layer (the number of columns of W).

3 Effect of Activation Functions and Saturation During Training

Two things we want to avoid and that can be revealed from the evolution of activations is excessive saturation of activation functions on one hand (then gradients will not propagate well), and overly linear units (they will not compute something interesting).

3.1 Experiments with the Sigmoid

The sigmoid non-linearity has been already shown to slow down learning because of its none-zero mean that induces important singular values in the Hessian (LeCun et al., 1998b). In this section we will see another symptomatic behavior due to this activation function in deep feedforward networks.

We want to study possible saturation, by looking at the evolution of activations during training, and the figures in this section show results on the **Shapeset-**3 $\times$ 2 data, but similar behavior is observed with the other datasets. Figure 2 shows the evolution of the activation values (after the non-linearity) at each hidden layer during training of a deep architecture with sigmoid activation functions. Layer 1 refers to the output of first hidden layer, and there are four hidden layers. The graph shows the means and standard deviations of these activations. These statistics along with histograms are computed at different times during learning, by looking at activation values for a fixed set of 300 test examples.

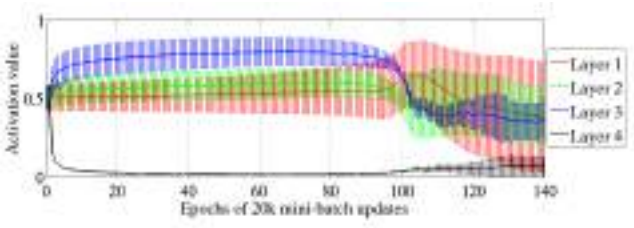

Figure 2: *Mean and standard deviation (vertical bars) of the activation values (output of the sigmoid) during supervised learning, for the different hidden layers of a deep architecture. The top hidden layer quickly saturates at 0 (slowing down all learning), but then slowly desaturates around epoch 100.*

We see that very quickly at the beginning, all the sigmoid activation values of the last hidden layer are pushed to their lower saturation value of 0. Inversely, the others layers have a mean activation value that is above 0.5, and decreasing as we go from the output layer to the input layer. We have found that this kind of saturation can last very long in deeper networks with sigmoid activations, e.g., the depth-five model never escaped this regime during training. The big surprise is that for intermediate number of hidden layers (here four), the saturation regime may be escaped. At the same time that the top hidden layer moves out of saturation, the first hidden layer begins to saturate and therefore to stabilize.

We hypothesize that this behavior is due to the combination of random initialization and the fact that an hidden unit output of 0 corresponds to a saturated sigmoid. Note that deep networks with sigmoids but initialized from unsupervised pre-training (e.g. from RBMs) do not suffer from this saturation behavior. Our proposed explanation rests on the hypothesis that the transformation that the lower layers of the randomly initialized network computes initially is not useful to the classification task, unlike the transformation obtained from unsupervised pre-training. The logistic layer output $\mathrm{softmax}(b+Wh)$ might initially rely more on its biases b (which are learned very quickly) than on the top hidden activations h derived from the input image (because h would vary in ways that are not predictive of y, maybe correlated mostly with other and possibly more dominant variations of x). Thus the error gradient would tend to push Wh towards 0, which can be achieved by pushing h towards 0. In the case of symmetric activation functions like the hyperbolic tangent and the softsign, sitting around 0 is good because it allows gradients to flow backwards. However, pushing the sigmoid outputs to 0 would bring them into a saturation regime which would prevent gradients to flow backward and prevent the lower layers from learning useful features. Eventually but slowly, the lower layers move toward more useful features and the top hidden layer then moves out of the saturation regime. Note however that, even after this, the network moves into a solution that is of poorer quality (also in terms of generalization)

then those found with symmetric activation functions, as can be seen in figure 11.

3.2 Experiments with the Hyperbolic tangent

As discussed above, the hyperbolic tangent networks do not suffer from the kind of saturation behavior of the top hidden layer observed with sigmoid networks, because of its symmetry around 0. However, with our standard weight initialization $U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$, we observe a sequentially occurring saturation phenomenon starting with layer 1 and propagating up in the network, as illustrated in Figure 3. Why this is happening remains to be understood.

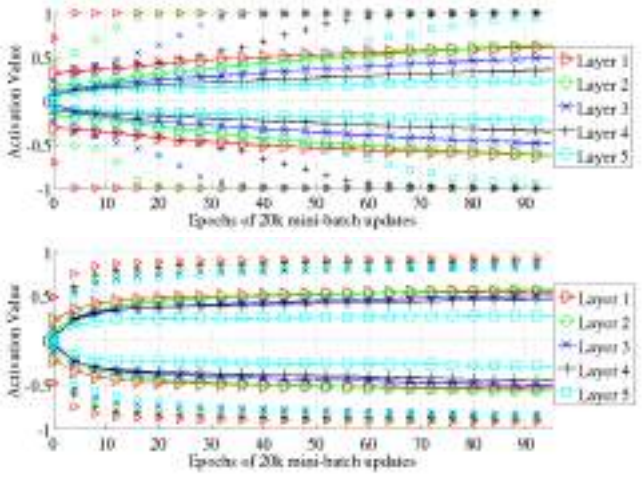

Figure 3: *Top:98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of the activation values for the hyperbolic tangent networks in the course of learning. We see the first hidden layer saturating first, then the second, etc. Bottom: 98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for the softsign during learning. Here the different layers saturate less and do so together.*

3.3 Experiments with the Softsign

The softsign $x/(1+|x|)$ is similar to the hyperbolic tangent but might behave differently in terms of saturation because of its smoother asymptotes (polynomial instead of exponential). We see on Figure 3 that the saturation does not occur one layer after the other like for the hyperbolic tangent. It is faster at the beginning and then slow, and all layers move together towards larger weights.

We can also see at the end of training that the histogram of activation values is very different from that seen with the hyperbolic tangent (Figure 4). Whereas the latter yields modes of the activations distribution mostly at the extremes (asymptotes -1 and 1) or around 0, the softsign network has modes of activations around its knees (between the linear regime around 0 and the flat regime around -1 and 1). These are the areas where there is substantial non-linearity but where the gradients would flow well.

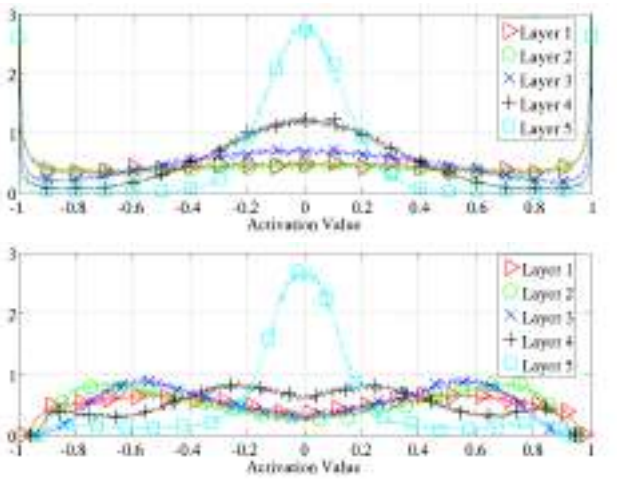

Figure 4: *Activation values normalized histogram at the end of learning, averaged across units of the same layer and across 300 test examples. Top: activation function is hyperbolic tangent, we see important saturation of the lower layers. Bottom: activation function is softsign, we see many activation values around (-0.6,-0.8) and (0.6,0.8) where the units do not saturate but are non-linear.*

4 Studying Gradients and their Propagation

4.1 Effect of the Cost Function

We have found that the logistic regression or conditional log-likelihood cost function ($-\log P(y|x)$ coupled with softmax outputs) worked much better (for classification problems) than the quadratic cost which was traditionally used to train feedforward neural networks (Rumelhart et al., 1986). This is not a new observation (Solla et al., 1988) but we find it important to stress here. We found that the plateaus in the training criterion (as a function of the parameters) are less present with the log-likelihood cost function. We can see this on Figure 5, which plots the training criterion as a function of two weights for a two-layer network (one hidden layer) with hyperbolic tangent units, and a random input and target signal. There are clearly more severe plateaus with the quadratic cost.

4.2 Gradients at initialization

4.2.1 Theoretical Considerations and a New Normalized Initialization

We study the back-propagated gradients, or equivalently the gradient of the cost function on the inputs biases at each layer. Bradley (2009) found that back-propagated gradients were smaller as one moves from the output layer towards the input layer, just after initialization. He studied networks with linear activation at each layer, finding that the variance of the back-propagated gradients decreases as we go backwards in the network. We will also start by studying the linear regime.

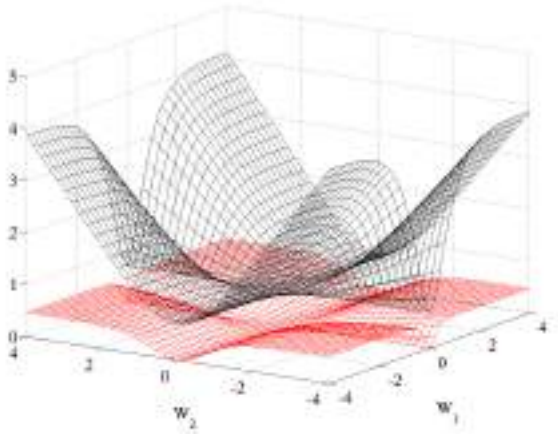

Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

For a dense artificial neural network using symmetric activation function f with unit derivative at 0 (i.e. $f'(0) = 1$), if we write $\mathbf{z}^i$ for the activation vector of layer i, and $\mathbf{s}^i$ the argument vector of the activation function at layer i, we have $\mathbf{s}^i = \mathbf{z}^i W^i + \mathbf{b}^i$ and $\mathbf{z}^{i+1} = f(\mathbf{s}^i)$. From these definitions we obtain the following:

$$\frac{\partial Cost}{\partial s^i_k} = f'(s^i_k) W^{i+1}_{k,\bullet} \frac{\partial Cost}{\partial \mathbf{s}^{i+1}} \tag{2}$$

$$\frac{\partial Cost}{\partial w^i_{l,k}} = z^i_l \frac{\partial Cost}{\partial s^i_k} \tag{3}$$

The variances will be expressed with respect to the input, outpout and weight initialization randomness. Consider the hypothesis that we are in a linear regime at the initialization, that the weights are initialized independently and that the inputs features variances are the same ($= Var[x]$). Then we can say that, with n_i the size of layer i and x the network input,

$$f'(s^i_k) \approx 1, \tag{4}$$

$$Var[z^i] = Var[x] \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}], \tag{5}$$

We write $Var[W^{i'}]$ for the shared scalar variance of all weights at layer i'. Then for a network with d layers,

$$Var\Big[\frac{\partial Cost}{\partial s^i}\Big] = Var\Big[\frac{\partial Cost}{\partial s^d}\Big] \prod_{i'=i}^{d} n_{i'+1} Var[W^{i'}], \tag{6}$$

$$Var\Big[\frac{\partial Cost}{\partial w^i}\Big] = \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}] \prod_{i'=i}^{d-1} n_{i'+1} Var[W^{i'}] \times Var[x] Var\Big[\frac{\partial Cost}{\partial s^d}\Big]. \tag{7}$$

From a forward-propagation point of view, to keep information flowing we would like that

$$\forall (i, i'), Var[z^i] = Var[z^{i'}]. \tag{8}$$

From a back-propagation point of view we would similarly like to have

$$\forall (i, i'), Var\Big[\frac{\partial Cost}{\partial s^i}\Big] = Var\Big[\frac{\partial Cost}{\partial s^{i'}}\Big]. \tag{9}$$

These two conditions transform to:

$$\forall i, \quad n_i Var[W^i] = 1 \tag{10}$$

$$\forall i, \quad n_{i+1} Var[W^i] = 1 \tag{11}$$

As a compromise between these two constraints, we might want to have

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}} \tag{12}$$

Note how both constraints are satisfied when all layers have the same width. If we also have the same initialization for the weights we could get the following interesting properties:

$$\forall i, Var\Big[\frac{\partial Cost}{\partial s^i}\Big] = \Big[n Var[W]\Big]^{d-i} Var[x] \tag{13}$$

$$\forall i, Var\Big[\frac{\partial Cost}{\partial w^i}\Big] = \Big[n Var[W]\Big]^{d} Var[x] Var\Big[\frac{\partial Cost}{\partial s^d}\Big] \tag{14}$$

We can see that the variance of the gradient on the weights is the same for all layers, but the variance of the back-propagated gradient might still vanish or explode as we consider deeper networks. Note how this is reminiscent of issues raised when studying recurrent neural networks (Bengio et al., 1994), which can be seen as very deep networks when unfolded through time.

The standard initialization that we have used (eq.1) gives rise to variance with the following property:

$$n Var[W] = \frac{1}{3} \tag{15}$$

where n is the layer size (assuming all layers of the same size). This will cause the variance of the back-propagated gradient to be dependent on the layer (and decreasing).

The normalization factor may therefore be important when initializing deep networks because of the multiplicative effect through layers, and we suggest the following initialization procedure to approximately satisfy our objectives of maintaining activation variances and back-propagated gradients variance as one moves up or down the network. We call it the **normalized initialization**:

$$W \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\Big] \tag{16}$$

4.2.2 Gradient Propagation Study

To empirically validate the above theoretical ideas, we have plotted some normalized histograms of activation values, weight gradients and of the back-propagated gradients at initialization with the two different initialization methods. The results displayed (Figures 6, 7 and 8) are from experiments on **Shapeset-**3×2, but qualitatively similar results were obtained with the other datasets.

We monitor the singular values of the Jacobian matrix associated with layer i:

$$J^i = \frac{\partial \mathbf{z}^{i+1}}{\partial \mathbf{z}^i} \tag{17}$$

When consecutive layers have the same dimension, the average singular value corresponds to the average ratio of infinitesimal volumes mapped from $\mathbf{z}^i$ to $\mathbf{z}^{i+1}$, as well as to the ratio of average activation variance going from $\mathbf{z}^i$ to $\mathbf{z}^{i+1}$. With our normalized initialization, this ratio is around 0.8 whereas with the standard initialization, it drops down to 0.5.

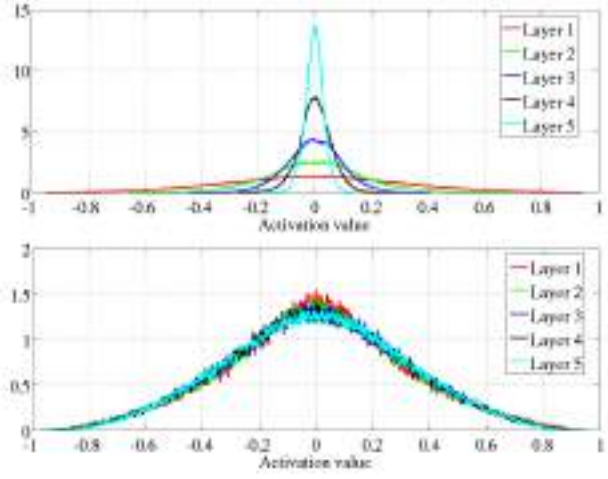

Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

4.3 Back-propagated Gradients During Learning

The dynamic of learning in such networks is complex and we would like to develop better tools to analyze and track it. In particular, we cannot use simple variance calculations in our theoretical analysis because the weights values are not anymore independent of the activation values and the linearity hypothesis is also violated.

As first noted by Bradley (2009), we observe (Figure 7) that at the beginning of training, after the standard initialization (eq. 1), the variance of the back-propagated gradients gets smaller as it is propagated downwards. However we find that this trend is reversed very quickly during learning. Using our normalized initialization we do not see such decreasing back-propagated gradients (bottom of Figure 7).

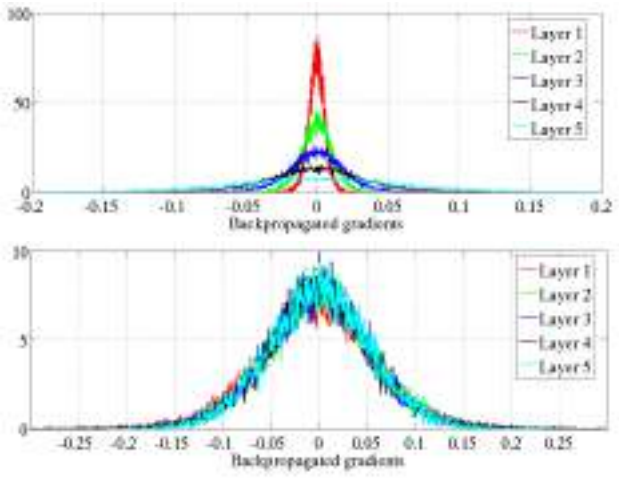

Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

What was initially really surprising is that even when the back-propagated gradients become smaller (standard initialization), the variance of the weights gradients is roughly constant across layers, as shown on Figure 8. However, this is explained by our theoretical analysis above (eq. 14). Interestingly, as shown in Figure 9, these observations on the weight gradient of standard and normalized initialization change during training (here for a tanh network). Indeed, whereas the gradients have initially roughly the same magnitude, they diverge from each other (with larger gradients in the lower layers) as training progresses, especially with the standard initialization. Note that this might be one of the advantages of the normalized initialization, since having gradients of very different magnitudes at different layers may yield to ill-conditioning and slower training.

Finally, we observe that the softsign networks share similarities with the tanh networks with normalized initialization, as can be seen by comparing the evolution of activations in both cases (resp. Figure 3-bottom and Figure 10).

5 Error Curves and Conclusions

The final consideration that we care for is the success of training with different strategies, and this is best illustrated with error curves showing the evolution of test error as training progresses and asymptotes. Figure 11 shows such curves with online training on **Shapeset-**3×2, while Table 1 gives final test error for all the datasets studied (**Shapeset-**3×2, MNIST, CIFAR-10, and Small-ImageNet). As a baseline, we optimized RBF SVM models on one hundred thousand Shapeset examples and obtained 59.47% test error, while on the same set we obtained 50.47% with a depth five hyperbolic tangent network with normalized initialization.

These results illustrate the effect of the choice of activation and initialization. As a reference we include in Fig-

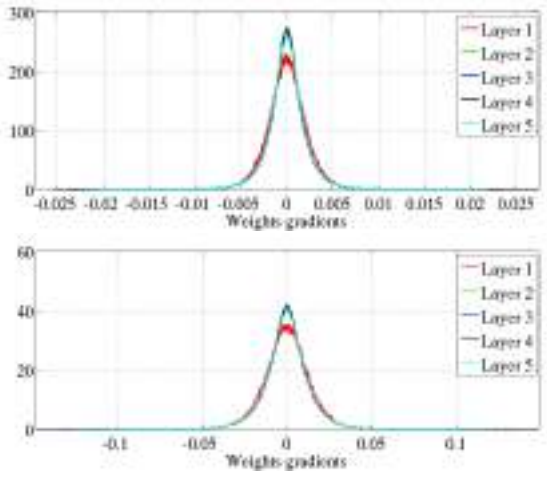

Figure 8: *Weight gradient normalized histograms with hyperbolic tangent activation just after initialization, with standard initialization (top) and normalized initialization (bottom), for different layers. Even though with standard initialization the back-propagated gradients get smaller, the weight gradients do not!*

Table 1: *Test error with different activation functions and initialization schemes for deep networks with 5 hidden layers. N after the activation function name indicates the use of normalized initialization. Results in bold are statistically different from non-bold ones under the null hypothesis test with* $p = 0.005$.

TYPE	Shapeset	MNIST	CIFAR-10	ImageNet
Softsign	**16.27**	**1.64**	55.78	**69.14**
Softsign N	**16.06**	**1.72**	**53.8**	**68.13**
Tanh	27.15	**1.76**	55.9	70.58
Tanh N	**15.60**	**1.64**	**52.92**	**68.57**
Sigmoid	82.61	2.21	57.28	70.66

ure 11 the error curve for the supervised fine-tuning from the initialization obtained after unsupervised pre-training with denoising auto-encoders (Vincent et al., 2008). For each network the learning rate is separately chosen to minimize error on the validation set. We can remark that on **Shapeset-**3×2, because of the task difficulty, we observe important saturations during learning, this might explain that the normalized initialization or the softsign effects are more visible.

Several conclusions can be drawn from these error curves:

- The more classical neural networks with sigmoid or hyperbolic tangent units and standard initialization fare rather poorly, converging more slowly and apparently towards ultimately poorer local minima.
- The softsign networks seem to be more robust to the initialization procedure than the tanh networks, presumably because of their gentler non-linearity.
- For tanh networks, the proposed normalized initialization can be quite helpful, presumably because the layer-to-layer transformations maintain magnitudes of activations (flowing upward) and gradients (flowing backward).

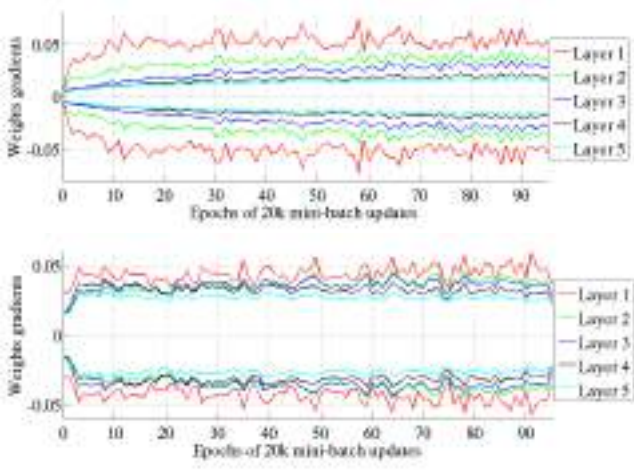

Figure 9: *Standard deviation intervals of the weights gradients with hyperbolic tangents with standard initialization (top) and normalized (bottom) during training. We see that the normalization allows to keep the same variance of the weights gradient across layers, during training (top: smaller variance for higher layers).*

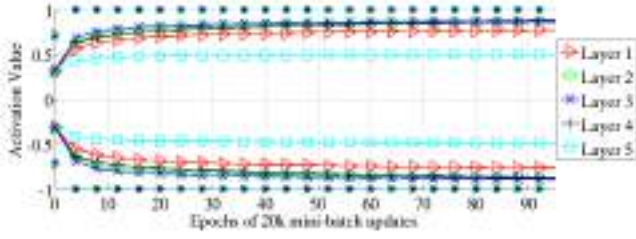

Figure 10: *98 percentile (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for hyperbolic tangent with normalized initialization during learning.*

Others methods can alleviate discrepancies between layers during learning, e.g., exploiting second order information to set the learning rate separately for each parameter. For example, we can exploit the diagonal of the Hessian (LeCun et al., 1998b) or a gradient variance estimate. Both those methods have been applied for **Shapeset-**3×2 with hyperbolic tangent and standard initialization. We observed a gain in performance but not reaching the result obtained from normalized initialization. In addition, we observed further gains by combining normalized initialization with second order methods: the estimated Hessian might then focus on discrepancies between units, not having to correct important initial discrepancies between layers.

In all reported experiments we have used the same number of units per layer. However, we verified that we obtain the same gains when the layer size increases (or decreases) with layer number.

The other conclusions from this study are the following:

- Monitoring activations and gradients across layers and

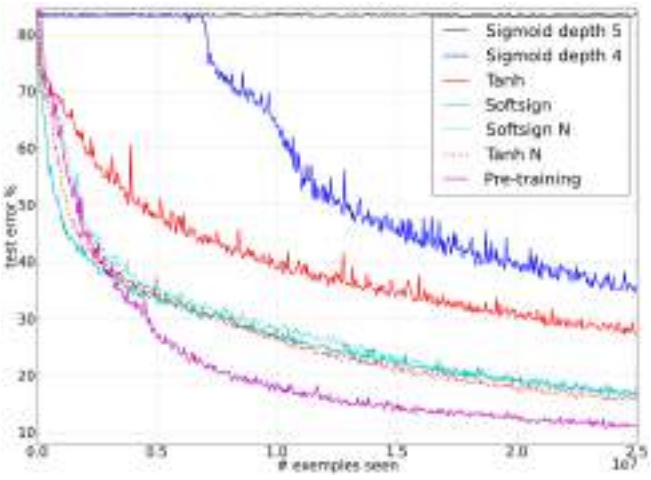

Figure 11: *Test error during online training on the* **Shapeset-**3×2 *dataset, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

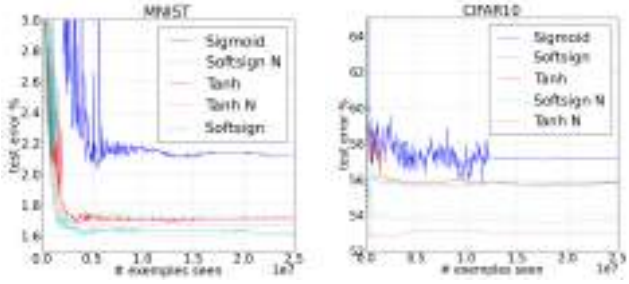

Figure 12: *Test error curves during training on MNIST and CIFAR10, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

training iterations is a powerful investigative tool for understanding training difficulties in deep nets.

- Sigmoid activations (not symmetric around 0) should be avoided when initializing from small random weights, because they yield poor learning dynamics, with initial saturation of the top hidden layer.
- Keeping the layer-to-layer transformations such that both activations and gradients flow well (i.e. with a Jacobian around 1) appears helpful, and allows to eliminate a good part of the discrepancy between purely supervised deep networks and ones pre-trained with unsupervised learning.
- Many of our observations remain unexplained, suggesting further investigations to better understand gradients and training dynamics in deep architectures.

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*, 1–127. Also published as a book. Now Publishers, 2009.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *NIPS 19* (pp. 153–160). MIT Press.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*, 157–166.

Bergstra, J., Desjardins, G., Lamblin, P., & Bengio, Y. (2009). *Quadratic polynomials learn better image features* (Technical Report 1337). Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Bradley, D. (2009). *Learning in modular systems*. Doctoral dissertation, The Robotics Institute, Carnegie Mellon University.

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML 2008*.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., & Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. *AISTATS'2009* (pp. 153–160).

Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images* (Technical Report). University of Toronto.

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, *10*, 1–40.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. *ICML 2007*.

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

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998b). Efficient backprop. In *Neural networks, tricks of the trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag.

Mnih, A., & Hinton, G. E. (2009). A scalable hierarchical distributed language model. *NIPS 21* (pp. 1081–1088).

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2007). Efficient learning of sparse representations with an energy-based model. *NIPS 19*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Solla, S. A., Levin, E., & Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, *2*, 625–639.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *ICML 2008*.

Weston, J., Ratle, F., & Collobert, R. (2008). Deep learning via semi-supervised embedding. *ICML 2008* (pp. 1168–1175). New York, NY, USA: ACM.

Zhu, L., Chen, Y., & Yuille, A. (2009). Unsupervised learning of probabilistic grammar-markov models for object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*, 114–128.