Understanding the difficulty of training deep feedforward neural networks

Blanca Cano Camarero

Universidad Autónoma de Madrid

March 1, 2023

Overview

Understanding the difficulty of training deep feedforward neural networks

- Authors: Xavier Glorot and Yoshua Bengio.
- Publication year: 2010.
- Conference: Proceedings of the thirteenth international conference on artificial intelligence and statistics.
- Cited by: 19886 (I have found is quite a lot!)

Reference: Glorot and Bengio (2010)

Abstract

Context

- Before 2006 deep multilayer neural networks were not successfully trained.
- But after, by new initialization or training mechanisms experimental results showed the superiority of deeper vs less deep architectures.

Article objetive

- Understand why standard gradient descent from random initialization is doing so poorly with deep neural networks.

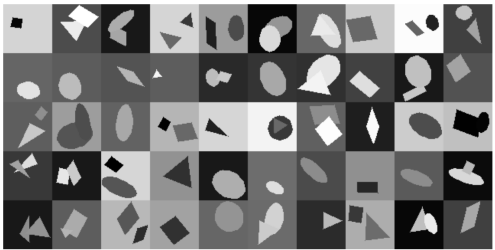
Experiments

- Observe the influence of the non-linear activations functions:
  - Logistic sigmoid activation is unsuited for deep networks for deep networks with random initialization due to its mean value, which can drive especially the **top hidden layer into saturation**.
  - Saturated units can move out of saturation by themselves, albeit slowly (and explain the plateaus sometimes seen when training neural networks).
  - Found a new non-linearity that saturares less can often be beneficial.
- They **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 one.
- They proposed a new initialization scheme that brings substantially faster convergence.

Experimental setting and Datasets

Online learning on a synthetic images dataset: Shapeset- 3 $\times 2$ dataset.

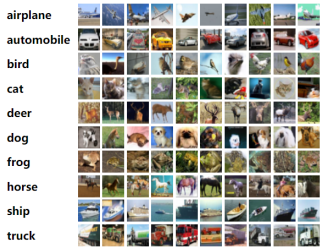
- Focus on optimization issues rather than on the small-sample regularization effects.
- The learner tries to predict which objects (parallelogram, triangle, or elipse) are presented, and one or two objects can be present yield into 9 possible classification.



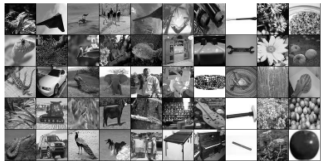
Finite Datasets

- The MNIST digits.
  - 50,000 training images,
  - 10,000 validations images.
- CIFAR-10 labeled subset of the tiny images.
  - 50,000 training images,
  - 10,000 validations images,
  - 10 classes corresponding to he main object: airplane, au- tomobile, bird, cat, deer, dog, frog, horse, ship, or truck.
  - Classes are balanced.
- Small-ImageNet.
  - 90,000 training images,
  - 10,000 validations images,
  - 10 classes corresponding to he main object: reptiles, vehicles, birds, mammals, fish, furniture, instruments, tools, flowers and fruits.
  - Classes are balanced.

(a) MNIST



(b) CIFAR-10



(c) Small-ImageNet

Figure 1: Three simple graphs

Some observations

- Only classification??
- Obsolete architecture.
- What about convolutional neuronal networks.
- And LTSM?

Some updated articles:
See Hartmann et al. (2021) and its art state.

Experimental Setting

- They optimized feedforward neural networks:
  - With one to five hidden layers.
  - One thousand hidden units per layer.
  - Softmax logistic regression for the output.
- Cost function is the negative log-likelihood:

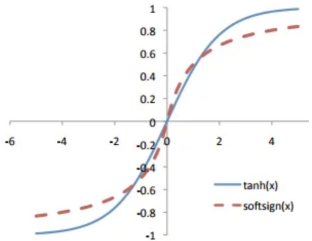
$$-\log P(y|x), \tag{1}$$

  where (x, y) is the (input image, target class).
- NN optimized with stochastic back-propagation on mini-batches of size ten.
- Learning rate is a hyperparameter that is optimized based on validation set error after a large number of updates (5 million).

Experimental setting

- The non-linear activation functions in the hidden layers varied on:
  - The sigmoid
  - the hyperbolic tangent,
  - the *newly proposed* softsign:

$$\frac{x}{1+|x|} \tag{2}$$



Experimental setting

- They search for the best hyperparameters (learning rate and depth) separately for each model.
- Initialization:
  - Biases to 0,
  - the weight W_{ij} at each layer with the following commonly used heuristic

$$W_{ij} \sim U\left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right), \tag{3}$$

where U is the uniform distribution and n the size of the previous layer.

Avoidable things

- Excessive saturation of activation functions (affects gradient propagation).
- Overlay linear units (will not compute something interesting).

Experiments with the Sigmoid

The sigmoid non-linearity has symptomatic behaviour.

- Its none-zero mean that induces import singular values in the Hessian.
- Saturation levels.

Experiments considerations

- looking at the evolution of activations during training on the Shapeset-3 x 2 data.
- 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.

Evolution ot the activations during training

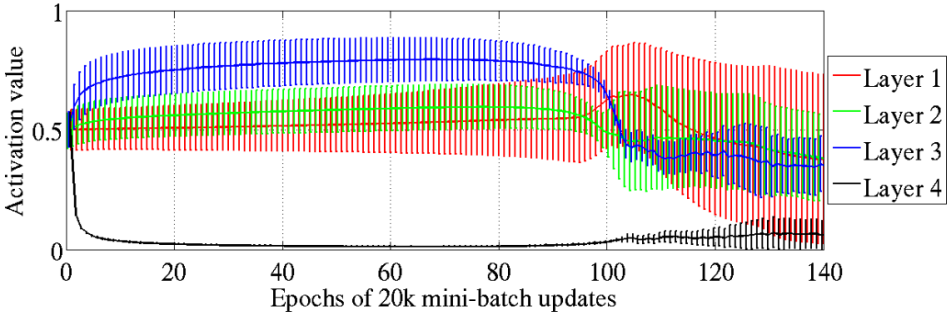


Figure 2: Mean and standard deviation (vertical bars) of the activation values (output of the sigmoid) during supervised learning, for the different hidden layers of a deep architecture. The top hidden layer quickly saturates at 0 (slowing down all learning), but then slowly desaturates around epoch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

Observations

- 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.
- 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.
- 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.
- The top hidden layer moves out of saturation, the first hidden layer begins to saturate and therefore to stabilize.

Hypothesis I

- 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 since deep networks with sigmoids but initialized from unsupervised pre-training (e.g. from RBMs) do not suffer from this saturation behavior. **(And what about ReLu now?)**
- 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 $softmax(b + Wh)$ 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).

Hypothesis II

- Would the error gradient wold tend to push Wh towards 0, which can be achieved by pushing h towards 0.
- Symmetric activation functions like the hyperbolic tangent and the softsign, sitting around 0 is good because it allows gradients to flow backwards.
- 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. However that, even after this, the network moves into a solution that is of poorer quality.

Test error during online training

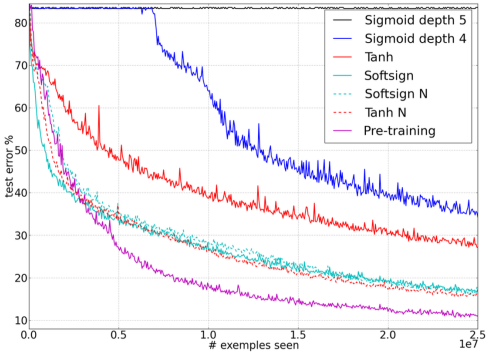


Figure 11: *Test error during online training on the* **Shapeset-**3×2 *dataset, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

Experiments with the Hyperbolic tangent

- 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.
- With the standard weight initialization, 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.**

Activation values for hiperbolic tangent and softsign

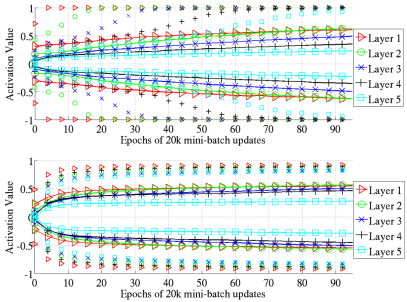


Figure 3: Top:98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of the activation values for the hyperbolic tangent networks in the course of learning. We see the first hidden layer saturating first, then the second, etc. Bottom: 98 percentiles (markers 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.

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.

Histogram of activation values at the end of the learning

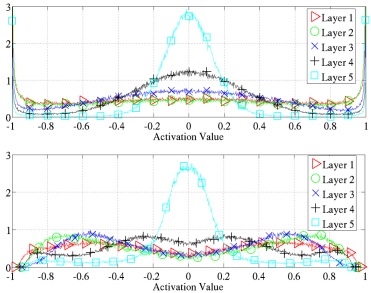


Figure 4: Activation values normalized histogram at the end of learning, averaged across units of the same layer and across 300 test examples. Top: activation function is hyperbolic tangent, we see important saturation of the lower layers. Bottom: activation function is softsign, we see many activation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

Query

Why are ReLU good?

Effect of the Cost Function

The logistic regression function

$$-\log P(y|x) \tag{4}$$

coupled with softmax output worked much better for classification problems since the plateaus in the training criterios are less present with the log-likelihood cost function.

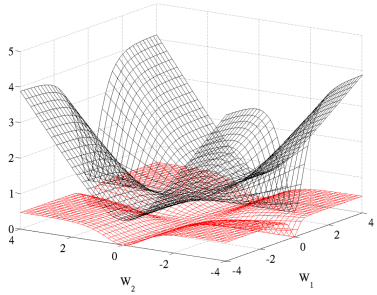


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W1 respectively on the first layer and W2 on the second, output layer.

Some question about this statement

- How the find this result? By computing gradients of a data set? ...
- There are any math proof?

Theoretical Considerations and a New Normalized Initialization

Brandley:

- Back propagated gradients were smaller as one moves from the output layer towards the input layer.
- With linear activation at each layer, the variance of the back propagated gradients decreases as we go backward in the network.

For dense neural neworks

We are going to use linear symmetric activation functions f with unit derivative at 0. (What about reLU? Still true this result?)

Mathematical model:

- Activation vector of layer i is z^i.
- The argument vector of the activation function at layer s^i.

$$s^i = z^i W^i + b^i \tag{5}$$

$$z^{i+1} = f(s^i). \tag{6}$$

Backpropagation

The error function of the neural network h is $Cost(h)$ so:

$$\frac{\partial Cost}{\partial s_k^i} = f'(s_k^i) W_{k,*}^{i+1} \frac{\partial Cost}{\partial s^{i+1}}, \tag{7}$$

$$\frac{\partial Cost}{\partial w_{l,k}^i} = z_l^i \frac{\partial Cost}{\partial s_k^i}. \tag{8}$$

Variance

- The variance will be expressed with respect to the input, output and weight initialization randomness.
- The weights are initialized independently.
- Input features variance ($var(x)$) are the same.

$$f'(s_k^i) \approx 1. \tag{9}$$

Using Taylor expansions for the moments of functions of random variables:

$$Var(f(X)) \approx \left(f'\left(E[X]\right)\right)^2 Var[X] \tag{10}$$

1

Since $f'' = 0$ under our linearity assumption the approximation is equality.

[1] See: https://en.wikipedia.org/wiki/Variance and https://en.wikipedia.org/wiki/Taylor_expansions_for_the_moments_of_functions_of_random_variables

Expressing the variance with respect to the input

Hypothesis:

- We are in a linear regime at the initialization.
- The weights are initialized independently.
- The inputs features variances are the same.

Let x be the network input and n_i the size of the layer i,

$$f'(s_k^i) \approx 1, \tag{11}$$

$$Var\left[z^i\right] = Var[x] \prod_{j=0}^{i-1} n_j Var\left[W^j\right]. \tag{12}$$

Variace of a product

Let X, Y be independent random variables.

$$Var(XY) = E[X]^2 Var[Y] + E[Y]^2 Var[X] + Var[X]Var[Y]. \tag{13}$$

Under our assumptions:

- $E[W] = 0$.
- Abuse of notation: W and x include bias and $Var[W^j]$ for the share of all the weights at layer j.
- For an induction proof the **base case**:

$$Var\left[z^0\right] = Var[x]. \tag{14}$$

Proof using induction I

Hypotheses induction (12) is true for i-1

$$Var[z^i] = Var\left[\sum_j f\left(s_j^{i-1}\right)\right] \tag{15}$$

$$= n_{i-1} Var\left[f\left(W^{i-1}z^{i-1}\right)\right] \tag{16}$$

$$= f'\left(W^{i-1}z^{i-1}\right) Var\left[W^{i-1}z^{i-1}\right] \tag{17}$$

$$= Var\left[W^{i-1}z^{i-1}\right] \tag{18}$$

$$= E\left[z^{i-1}\right]^2 Var\left[W^{i-1}\right] + E\left[W^{i-1}\right]^2 Var\left[z^{i-1}\right] + Var\left[W^{i-1}\right] Var\left[z^{i-1}\right] \tag{19}$$

$$= n_{i-1} Var\left[W^{i-1}\right] Var\left[z^{i-1}\right]. \tag{20}$$

Note: $E\left[z^i\right] = 0$ using second-order Taylor expansions and that f is linear.

Variance of a the cost function

For a network with d layers,

$$Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^d}\right]\prod_{j=i}^{d} n_{j+1} Var\left[W^j\right]. \tag{21}$$

(Proof using 7)

$$Var\left[\frac{\partial Cost}{\partial w^i}\right] = \left(\prod_{j=i}^{i-1} n_i Var\left[W^j\right]\right)\left(\prod_{j=i}^{d-1} n_{j+1} Var\left[W^j\right]\right) \times Var[x] Var\left[\frac{\partial Cost}{\partial s^d}\right]. \tag{22}$$

(Proof using 8 and 21).

Desirable variances

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

$$\forall i, j \quad Var\left[z^i\right] = Var\left[z^j\right]. \tag{23}$$

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

$$\forall i, j \quad Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^j}\right]. \tag{24}$$

Requirements for achieve desirable variance

Using 12 and 14 condition 23 is equivalent to

$$\forall i, \quad n_i Var\left[W^i\right] = 1. \tag{25}$$

Using 21 condition 24 is equivalent to

$$\forall i, \quad n_{i+1} Var\left[W^i\right] = 1. \tag{26}$$

Both constraints are satisfied when all layers have the same width.
As a compromise between 25 and 24constraints, we might want to have

$$\forall i \quad , Var\left[W^i\right] = \frac{2}{n_i + n_{i+1}}. \tag{27}$$

The varianze of the backpropagation gradient might vanish or explode in deeper networks

If

$$\forall i, j \quad Var\left[W^i\right] = Var\left[W^j\right] \tag{28}$$

substituting on 21 we obtain

$$\forall i, Var\left[\frac{\partial Cost}{\partial s^i}\right] = [nVar[W]]^{d-i} Var[x] \tag{29}$$

$$\forall i, Var\left[\frac{\partial Cost}{\partial w^i}\right] = [nVar[W]]^{d} Var[x] Var\left[\frac{\partial Cost}{\partial s^d}\right]. \tag{30}$$

(Reminiscence of RNN see Bengio et al., 1994)

Variance of the experiments

$$Var[W] = \frac{1}{n3}, \tag{31}$$

where n is the layer size and this causes the variance of the back propagated gradient to be dependent on the layer and decreasing.

Initialization procedure: Normalized initialization

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.

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]. \tag{32}$$

This continuos being the trent?

Other initialization techniques

See Boulila et al. (2021)

TABLE I. COMPARISON BETWEEN WEIGHT INITIALIZATION TECHNIQUES

Initialization method	Pros.	Cons.
All-zeros / constant	Simplicity	Symmetry problem leading neurons to learn the same features
Random	Improves the symmetry-breaking process	- A saturation may occur leading to a vanishing gradient - The slope or gradient is small, which can cause the gradient descent to be slow
LeCun	Solves growing variance and gradient problems	- Not useful in constant-width networks - Takes into account the forward propagation of the input signal - This method is not useful when the activation function is non-differentiable
Xavier	Decreases the probability of the gradient vanishing/exploding problem	- This method is not useful when the activation function is non-differentiable - Dying neuron problem during the training
He	Solves dying neuron problems	- This method is not useful for layers with differentiable activation function such as ReLU or LeakyReLU

In Table I, a comparison between weight initialization techniques is depicted. In this table, we compared the main weight initialization techniques according to their advantages and limitations.

Gradient Propagation Study I

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

$$J^i = \frac{\partial z^{i+1}}{\partial z^i} \tag{33}$$

Gradient Propagation Study II

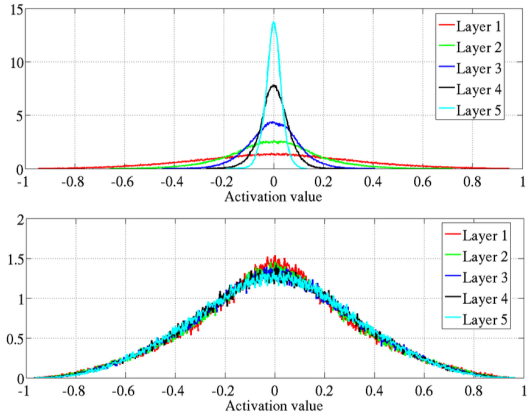


Figure 6: Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.

Gradients backpropagation Study

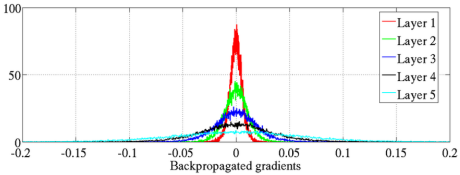


Figure 7: Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.

Conclusion

- Monitoring activations and gradients across layers and 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 I

Wadii Boulila, Maha Driss, Mohammed Al-Sarem, Faisal Saeed, and Moez Krichen. Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives, 02 2021.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/glorot10a.html`.

David Hartmann, Daniel Franzen, and Sebastian Brodehl. Studying the evolution of neural activation patterns during training of feed-forward relu networks. *Frontiers in Artificial Intelligence*, 4, 2021. ISSN 2624-8212. doi: 10.3389/frai.2021.642374. URL `https://www.frontiersin.org/articles/10.3389/frai.2021.642374`.