Convolutional layer weights initialization methods

Guilhem Garnier and **Louis Malosse**
Polytechnique Montréal - Génie Informatique and Génie Physique

Abstract

Weight initialization is a crucial step of every neural network training procedure. Proper weight initialization can significantly improve the convergence rate, prevent vanishing or exploding gradients, and enhance the overall performance of the network by avoiding poor local minima. In this article, we highlight the importance of a good initialization by comparing different techniques. We discuss the pros and cons of each approach, and put them to the test on an image classification task, using ResNet9 architecture. More specifically, we will focus on the initialization of convolutional layers.

1 Introduction

1.1 Different kinds of initializations

Narkhede et al. [NBS22] divide weight initialization methods into three categories: random initialization, data-driven initialization, and initialization with pre-training. Although these categories are relevant to give a general idea of the methods, the authors highlight some methods that don't fall strictly into those categories.
In particular, some methods are hybrid and use tools from several categories. For example, Das et al. [DBP21] initialize the weights by fitting the filters as linear auto-encoders. This kind of data-driven methods is very similar to transfer learning and pre-training methods.

1.2 Related work

As weight initialization is an ubiquitous problematic in machine learning, there are numerous paper either theorizing weight initialization methods or discussing and comparing these methods. We can cite the two key papers in weight initialization from which we took many results :

- *Understanding the difficulty of training deep feedforward neural networks* (X.Glorot et al. [GB10])
- *Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification* (K. He et al. [He+15])

There are numerous papers discussing weight initialization methods, as the comparison has to be updated regularly. We can cite the paper that inspired this project :

- *A review on weight initialization strategies for neural networks* (M. V. Narkhede et al. [NBS22])

2 Random Initialization

First of all, it is important to notice that randomness, or at least heterogeneity is *necessary* in weight initialization. A first thought might be to initialize all the weights to zero or any other value. However, the training algorithms based on backpropagation, as introduced by Rumelhart et al. [RHW85], would propagate the same update to every weight in a layer, making a proper training impossible.

In this section, we present techniques that rely on random distribution to initialize the weights.

2.1 Choice of distribution law

Uniform initialization

The most common method to initialize weights of a layer is to draw them according to a uniform distribution $\mathcal{U}([-bound, bound])$ A naive approach would be to always use a bound value constant like $\mathcal{U}([-1, 1])$. But this leads to great issues both in forward and backward passes.
In forward pass, a layer of bigger output dimension would increase the variance and norm of outputs.
In backward pass, a layer of bigger input dimension would increase the variance of the computed gradients. This can lead to both exploding or vanishing gradient. To solve this issue, we use bounds that depend on the layer size.

Normal initialization

We can also use a centered normal distribution instead of a uniform distribution. The bound parameter is replaced by the distribution std: $\mathcal{N}([0, \sigma^2])$ Like for the uniform distribution, a constant σ parameter would modify the variance of inputs in forward pass and gradients in backward pass. It is interesting to note that after the success of Krizhevsky et al. paper in 2012, initialization with a normal distribution centered on zero and standard deviation set to 0.01 and adding a bias of one on certain layers was very popular ; indeed, the results derived in this project, although they feel natural, are not obvious at all.

Hence, as for uniform distribution, we will use a σ parameter that depends on the layer size.

2.2 Choice of the law parameter

In the next sections, we will adopt the following notations :

- $W^{[L]}$ is the weight matrix of the L^{th} layer
- $a^{[L]} = f(W^{[L]}a^{[L-1]} + b^{[L]})$ with f the activation function and b the biases
- $n^{[L]}$ is the number of nodes in the L^{th} layer
- Δ_L is the gradient computed for the L^{th} layer

Xavier Initialization

Xavier initialization, or Glorot initialization, was established by Glorot et al. using several assumptions. The most important one is used to derive the evolution of the variance of the data as it passes through the layers of the network ; the activation functions are approximated as linear functions, which may seem unrealistic for a non-linear classifier, yet this is a good approximation for tanh close to zero. The purpose of this mathematical derivation is to find the correct weight initialization so that the variance of the data does not vary exponentially as it crosses the layers, for both the forward and the backward pass. An exponential decrease translates to a vanishing gradient, resulting in a network unable to adjust ; an exponential increase translates to an exploding gradient, preventing the network from converging to a minimum. The key idea behind this derivation is that, in a linear regime, the variance at layer L is given by :

$$\begin{aligned} Var(a^{[L]}) &= n^{[L-1]}Var(W^{[L]})Var(a^{[L-1]}) \\ &= \left[\prod_{l=1}^{L} n^{[l-1]}Var(W^{[l]})\right] Var(x) \end{aligned} \tag{1}$$

Reasoning with back propagation, and also considering that the activation function is linear, we can derive the following result :

$$Var(\Delta_{L-1}) = n^{[L]}Var(W^{[L]})Var(\Delta_L) \tag{2}$$

According to equations 1 and 2, in order to maintain a constant variance through forward and backward pass, the variance of the weights needs to follow :

$$\begin{cases} Var(W^{[L]}) = \frac{1}{n^{[L-1]}} \\ Var(W^{[L]}) = \frac{1}{n^{[L]}} \end{cases}$$

As it is not possible unless all the layers are the same size, the compromise chosen by Glorot et al. sets the variance of each layer to be :

$$Var(W^{[L]}) = \frac{2}{n^{[L-1]} + n^{[L]}}$$

Hence, Xavier initialization can correspond to the following initialization strategies :

$$\begin{cases} W^{[L]} \sim \mathcal{N}\left(0, \frac{2}{n^{[L-1]}+n^{[L]}}\right) \\ W^{[L]} \sim \mathcal{U}\left(\pm\sqrt{\frac{6}{n^{[L-1]}+n^{[L]}}}\right) \end{cases}$$

He Initialization

In their 2015 article, He et al. compute the "variance propagation" in a CNN that uses ReLU nonlinearities. Because of the simplicity of the ReLU function, they can bypass the linear assumption made by Glorot et al., deriving the following result :

$$\begin{aligned} Var(a^{[L]}) &= \frac{1}{2}n^{[L]}Var(W^{[L]})Var(a^{[L-1]}) \\ &= \left[\prod_{l=1}^{L} \frac{1}{2}n^{[l-1]}Var(W^{[l]})\right] Var(x) \end{aligned} \tag{3}$$

The backpropagation derivation is similar to the one given by Glorot et al., the simplicity of ReLU allowing again for a computation without any linearity assumption. The gradients' variance varies as follows :

$$Var(\Delta_{L-1}) = \frac{1}{2}n^{[L]}Var(W^{[L]})Var(\Delta_L) \tag{4}$$

Looking at equations 3 and 4, there is still a problem about $n^{[L]}$ and $n^{[L-1]}$ not necessarily being equal, but He et al. argue that it should not be really important given the regular CNN architectures. Consequently, the weights are initialized according to :

$$Var(W^{[L]}) = \frac{2}{n^{[L]}}$$

Using these results, the weights can be initialized with a normal or a uniform random distribution :

$$\begin{cases} W^{[L]} \sim \mathcal{N}\left(0, \frac{2}{n^{[L]}}\right) \\ W^{[L]} \sim \mathcal{U}\left(\pm\sqrt{\frac{6}{n^{[L]}}}\right) \end{cases}$$

This is known as the He initialization, and it has been shown by its creators that it is superior to Xavier initialization in the case of very deep neural networks. With a 22 layers CNN, He was showing faster convergence than Xavier, and with 30 layers, Xavier was even unable to converge due to vanishing gradients while He converged.

Gain

The use of different activation functions can modify the theoretical results written above.
Hence, in practice, it is common to multiply the found values by a certain constant g, that we call the gain. It follows that after including the $\sqrt{6}$ factor in the constant g, we get the formula:

$$bound = g * \frac{1}{\sqrt{n^{[L]}}}$$

And for normal initialization, similarly :

$$std = g * \frac{1}{\sqrt{n^{[L]}}}$$

3 Data driven initialization

Since every dataset needs a unique solution (architecture, training methods, weights, etc.), there are good reasons to think that initializing the weights specifically for a certain dataset will bear results. This key idea that the solution needs to be adapted to the problem from the very beginning - that is, weight initialization - led researchers like Lehtokangas et al. [LS98] or Yam et al. [Yam+02] to initialize the weight based on the data to infer, ensuring that the neurons outputs stay within the activation function active region.

Certain methods employ an "auto-encoder based greedy pre-training" [HS06], which consists in an optimization problem in which each layer of weights is optimized to encode and decode the activations preceding it, using subsets of the dataset. While the previous methods can be considered as regular pre-training (as they require a using gradient descent) and not clearly weight initialization, a method proposed by Das et al. [DBP21] achieves the same goal by solving a convex optimization problem. If we define the latent code $\mathbf{S}$, the activations $\mathbf{X}$, the weight matrix $\mathbf{W}$ can be initialized by solving :

$$\min_{\mathbf{W}} = \underbrace{\|\mathbf{X} - \mathbf{W}^T\mathbf{S}\|_F^2}_{\text{Decoding Loss}} + \lambda \underbrace{\|\mathbf{W}\mathbf{X} - \mathbf{S}\|_F^2}_{\text{Encoding Loss}} \tag{5}$$

$\mathbf{S}$ can be derived using Principal Components Analysis (PCA) or any other convenient method. By taking the derivative of equation 5, solving the problem above is equivalent to finding $\mathbf{W}$ so that :

$$\underbrace{\mathbf{S}\mathbf{S}^T}_{\mathbf{A}}\mathbf{W} + \mathbf{W}\underbrace{\lambda\mathbf{X}\mathbf{X}^T}_{\mathbf{B}} = \underbrace{(1+\lambda)\mathbf{S}\mathbf{X}^T}_{\mathbf{C}} \tag{6}$$

By defining the matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ as above, this is a Sylvester equation $\mathbf{AW} + \mathbf{WB} = \mathbf{C}$ whose resolution is well known [BS72].

This initialization method is applied to every layer before the network is trained. In our experiments, we calculated $\mathbf{S}$ using PCA. The results are presented in the Experiments section.

4 Normalization layers

Most modern architecture use normalization layers that improve performances, such as described by Ioffe and Szegedy [IS15]. These layers improve performance and lower the issue of gradient vanishing or exploding, making the weight initialization less critical.
We will test this assumption on the ResNet9, using both full architecture and architecture without batch normalization layers.

5 Experiments

Our experiments were performed using a ResNet9 on the image classification task of CIFAR-10 dataset.
We compare the results when varying the gain of random Kaiming initialization.
We first test the ResNet9 with several gains and without batch normalization layers.
We then add batch normalization layers and rerun the same test.
We finally initialize the weights with a data driven method, the method described by Das et al. in Data-driven Weight Initialization with Sylvester Solvers.
All runs are made through 20 epochs, with a batch size of 256 and Adam Optimizer with learning rate 1e-3 and weight decay 1e-5.

5.1 Architecture

We used the architecture ResNet9, composed of 8 convolutional layers with ReLU activations and 1 fully connected layer at the end with softmax activation for classification.
In addidation, there are two residual connections as shown in the Figure 1.
Before ReLU activations, there is a Batch Normalization Layer. We will remove this layer in a first time to compare the behaviour in both situations.

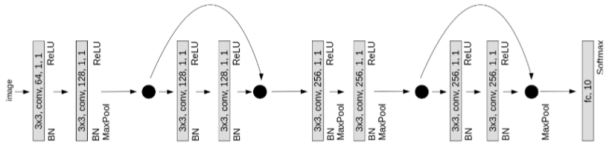

Figure 1: ResNet9 architecture (**github.com/matthias-wright/cifar10-resnet**)

5.2 Gain value variation

We first train the network without batch normalization layer, using a He uniform random initialization with different gains. That is, weights are sampled according to $\mathcal{U}\left(\pm\frac{gain}{\sqrt{c_{in} * kernelsize^2}}\right)$
A common random seed is used for all generations gain value, meaning $W_{gain=10} = 10 * W_{gain=1}$.

Results

gain	min test loss	max test accuracy
0.01	2.303	0.100
0.1	0.766	0.776
$\sqrt{6}^{-1}$	0.713	0.784
$\sqrt{3}^{-1}$	0.713	0.790
1	0.653	0.802
$\sqrt{3}$	0.649	0.802
$\sqrt{6}$	0.666	0.789
10	2.539	0.415
100	$8E9$	0.392

Figure 2: Best results without normalization and with He uniform initialization when training for 20 epochs.

We see on Figure 2 and Figure 3 that the gain value has a great impact on performance. When the gain is too small ($= 0.01$), the network doesn't learn at all. When the gain is

too large (> 10) the network learns but very slowly and seems to converge to a largely suboptimal local minimum.

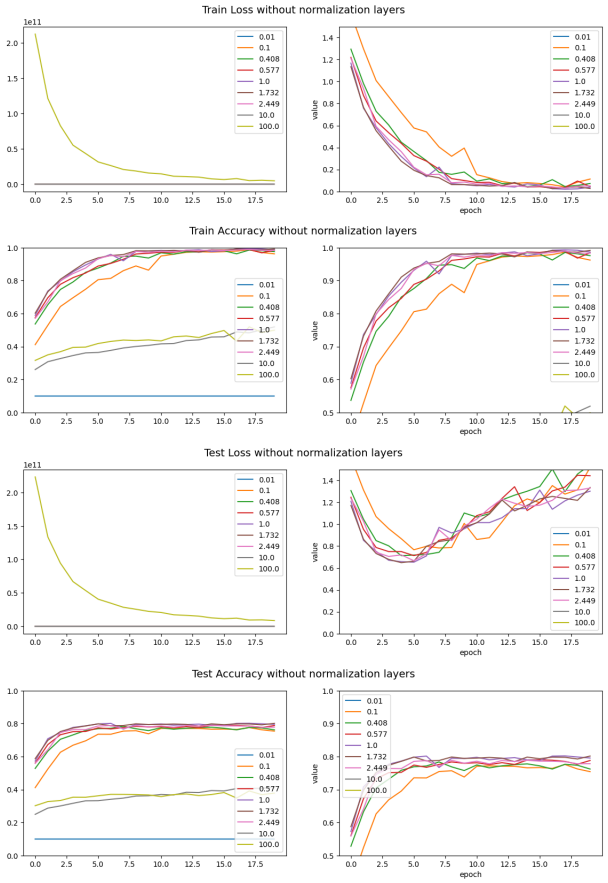

Figure 3: Train and test loss and accuracy without batch normalization layer (right figures are a re-scaling of the left ones)

5.3 Adding Normalization Layers

We then add normalization layers after each convolutional layer (before ReLU activation) and we rerun the training. We still use a He uniform random initialization with different gains.
That is, weights are sampled according to $\mathcal{U}\left(\pm\frac{gain}{\sqrt{c_{in} * kernelsize^2}}\right)$
A common random seed is used for all generations gain value, meaning $W_{gain=10} = 10 * W_{gain=1}$. The random seed is actually the same as in the previous test (without normalization layer), so for a certain gain the weights are the same

Results

We see on figure 4 and 5 that the addition of Batch Normalization layers lowered the issues encountered with the architecture without Batch Normalization. Still, we see that a too large gain value slows the convergence and leads to a largely suboptimal local minimum.
We can also see that the overfitting is reduced, and the best

gain	min test loss	max test accuracy
0.01	0.513	0.855
0.1	0.509	0.870
$\sqrt{6}^{-1}$	0.458	0.866
$\sqrt{3}^{-1}$	0.477	0.871
1	0.465	0.885
$\sqrt{3}$	0.497	0.878
$\sqrt{6}$	0.519	0.861
10	0.733	0.802
100	0.962	0.695

Figure 4: Best results when adding batch normalization and with He uniform initialization when training for 20 epochs.

test accuracy is greatly improved in comparison to the previous case. Adding batch normalization layer has then improved the generalization of the model.

5.4 Data Driven initialization

We test the driven initialization method from Das et al.
This time needed to complete this initialization increases fast with the number of inputs considered, as well with the number of image parts considered. We then used only 256 images as first input for the first layer and 3000 image parts in total for each subsequent layer. We used PCA to get S (target latent representation of X). Because the number of channels improve a lot in the first layer (3 to 64), we couldn't apply PCA so we initialized the first layer randomly using He initialization and a gain = 1.

Results

The results in Figure 6 and Figure 7 don't show a clear improvement using the data driven method. Still, the results are comparable with the best we got with the random initialization method.
We only used a really small part of the data to compute the weights, so it actually is already kind a of good result. Using more data is time consuming, and wouldn't make much sense if it increases the global training time.

5.5 Limitations of the approach

Due to computation resources constrains, we weren't able to run the experiments using several seeds to get more reliable results.
We partly compensate that by using the same random seed when varying the gain, but it makes it difficult to compare with the data driven initialization method.
We clearly see that the addition of a normalization layer improves the performance (from 0.802 to 0.885 test accuracy) and reduces the need of a perfect initialization. A good gain value still improves the convergence rate and should be in a limited range around 1.

6 Conclusion

There are a myriad of ways to initialize weights for training a CNN, and we presented some of them in this article. First,

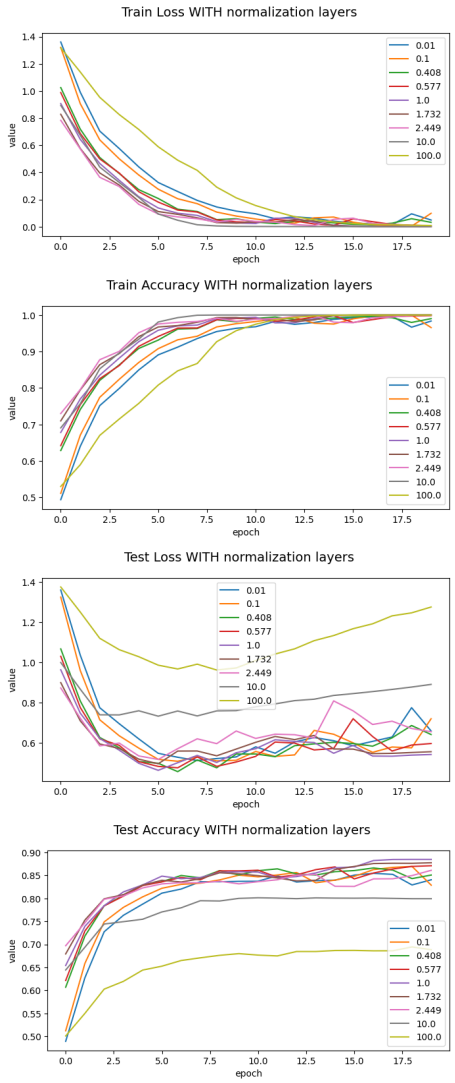

Figure 5: Test loss and accuracy with batch normalization layer

we discussed random weight initialization and the two main initialization methods that are widely used for this purpose : Xavier and He initialization. While it remains not well understood, it seems that He's is able to perform better than Xavier's. Then, we explained a data driven method that uses the solution of a Sylvester equation.

Our study has demonstrated the crucial role of weight initialization in the performance deep neural networks. It also has shown that using batch normalization method reduces the importance of weight initialization, but without removing it. In addition, it showed that batch normalization leads to way better generalization results.

Finally, our implementation of the data driven method didn't lead to improved results, but still to results comparable with the best ones of the random initialization methods. We think that these results could be improved using more data to initialize the weights, but at a high cost in computational time.

alpha	min test loss	max test accuracy
0.1	0.475	0.881
1.0	0.501	0.884
10.0	0.482	0.883

Figure 6: Best results when using sylvester computed weights when training for 20 epochs.

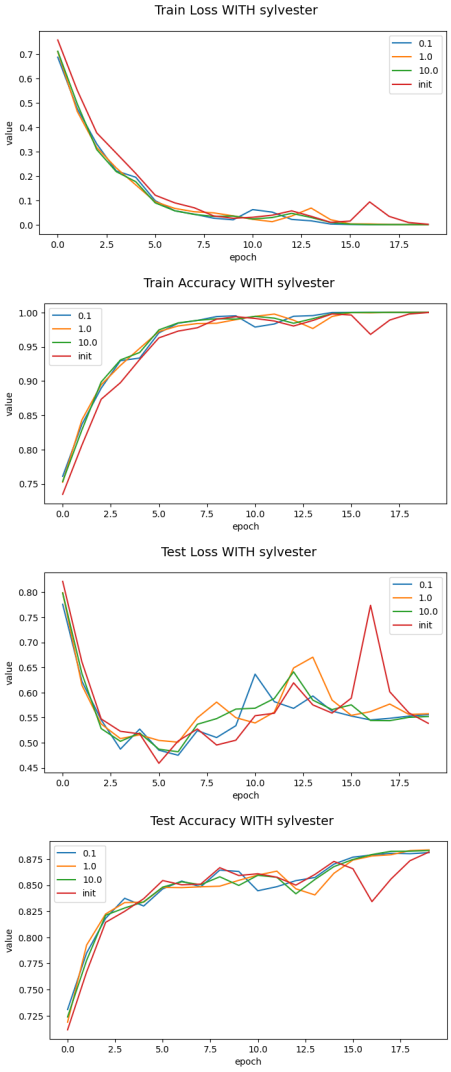

Figure 7: Train and test loss and accuracy with datadriven initialization using sylvester solvers. The alpha value represents the weight of the encoding error (to compare with the decoding error) and the init is the randomly initialized with He uniform and gain = 1

References

[BS72] Richard H. Bartels and George W Stewart. "Solution of the matrix equation AX+ XB= C [F4]". In: *Communications of the ACM* 15.9 (1972), pp. 820–826.

[DBP21] Debasmit Das, Yash Bhalgat, and Fatih Porikli. "Data-driven weight initialization with sylvester

solvers". In: *arXiv preprint arXiv:2105.10335* (2021).

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[He+15] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[LS98] Mikko Lehtokangas and Jukka Saarinen. "Weight initialization with reference patterns". In: *Neurocomputing* 20.1-3 (1998), pp. 265–278.

[NBS22] Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. "A review on weight initialization strategies for neural networks". In: *Artificial intelligence review* 55.1 (2022), pp. 291–322.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[Yam+02] Yat-Fung Yam et al. "An independent component analysis based weight initialization method for multilayer perceptrons". In: *Neurocomputing* 48.1-4 (2002), pp. 807–818.