Comparing Gaussian, Copula, and Batch Normalization Schemes in Fully Connected Neural Networks

Junqing Qiao
UMass Amherst
140 Governors Dr, Amherst, MA 01002
jqiao@umass.edu

Abstract

Batch normalization has shown great results improving the training and accuracy of neural networks. In this paper, we implement and analyze the performance of two different forms of normalization on randomized data using a fully connected architecture.

1. Introduction

The normal distribution is commonly used in modeling and statistics for its nice properties and ability to approximate many real world phenomenon. Given this motivation, we implement and analyze the effects normalizing the outputs of each layer of a fully connected feed forward neural network by the Gaussian distribution has on its accuracy on an arbitrary data set.

2. Background/Related Work

The main inspiration of this paper came from the idea of Copula normalization presented in [1] who's algorithm is very similar to the one outlined in this paper. Additionally, we borrow many of the same notation choices. Being a paper on normalization in neural networks, we must of course also give credit to the original batch norm paper which pioneered it all [2]. The code for this project was adapted from the code provided in the Stanford cs231n course [3].

3. Approach

The two normalization methods we are testing are one which normalizes to the uniform [-0.5, 0.5] distribution and another which normalizes to the Gaussian with mean 0 and variance 1. The implementation for the first normalization method is detailed in [1]. In this paper we show how to implement the second normalization method. Originally it was planned to test these normalization methods on the CIFAR-10 dataset however, due to results outlined in section 4, such tests were determined to be not worth implementing and have been subsequently omitted. Instead, we provide insights on the differences between batch normalization versus Copula and Gaussian normalization and why batch normalization performs so much better.

The objective of Gaussian normalization is to transform an arbitrary random variable produced as output by a layer in a feedforward network into a random variable that has a Gaussian distribution with $\mu = 0, \sigma^2 = 1$. We accomplish this by first estimating the CDF of $X : F(X)$ from the minibatch and then transforming it by feeding the result into the inverse CDF of the standard normal distribution Φ^{-1}.

3.1. Minibatch Statistics

We start by estimating the CDF of $X : \hat{F}(x)$ by calculating a piecewise $\hat{F}^{-1}(x)$. This section is almost identical to section 1.1 and 1.2 of [1]

First estimate the CDF on M values, where each percentile $F^{-1}(\frac{k}{M})$ will be calculated according to the batch samples $x_1, x_2, ..., x_m$. We make the assumptions that

- Each x_i is iid
- The true CDF of x can be reasonably approximated with the Empirical CDF or ECDF.
- $M << m$

Let $z_1, z_2, ..., z_m$ be the sorted set of the batch samples where z_1 is the smallest value. There are 3 cases to consider:

$$\hat{F}^{-1}(\frac{k}{M}) = \begin{cases} z_1 - (z_2 - z_1), & k = 0 \\ (1-d)z_i + d * z_{i+1}, & 0 < k < m \\ z_m + (z_m - z_{m-1}), & k = m \end{cases}$$

i and d in the third case are obtained by solving the equation $\frac{k}{M} = \frac{i+d}{m+1}$ where i is an integer and $0 \leq d < 1$.
We then estimate:

$$\hat{F}(x) = \begin{cases} \epsilon, & x < \hat{F}^{-1}(0) \\ \frac{1}{M+1}[k + \frac{x_i - \hat{F}^{-1}(\frac{k}{M})}{\hat{F}^{-1}(\frac{k+1}{M}) - \hat{F}^{-1}(\frac{k}{M})}], & \hat{F}^{-1}(\frac{k}{M}) \leq x < \hat{F}^{-1}(\frac{k+1}{M}) \\ 1 - \epsilon, & x > \hat{F}^{-1}(1) \end{cases}$$

3.2. Normalizing

There is no closed form solution to the inverse Gaussian CDF so we will use the python scipy.stats.norm.ppf function as an estimate for $\Phi^{-1}(p)$. Call this function $\hat{\Phi}^{-1}(p)$

$$\hat{x_i} = \hat{\Phi}^{-1}(\hat{F}(x_i))$$

Unlike Copula and batch normalization, It makes less sense to modify our output y as $y_i = \gamma * \hat{x_i} + \beta$ because $\hat{x_i}$ is not a linear transformation of x_i. However, to keep the number of trainable parameters constant for all normalization methods, we utilize β and γ the same as in the other two methods. We borrow from the notation of [1] and define $g_k := \hat{F}^{-1}(\frac{k}{M})$

Below are the derivatives needed during backpropagation:

$$\frac{dL}{d\beta} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_i}$$

$$\frac{dL}{d\gamma} = \sum_{i=1}^{N} \frac{\partial L}{\partial y_i} * \hat{x_i}$$

$$\frac{dL}{d\hat{x_i}} = \frac{\partial L}{\partial y_i}\gamma$$

$$\frac{d\hat{x_i}}{d\hat{F}(x_i)} = \frac{1}{\hat{\Phi}'(\hat{\Phi}^{-1}(\hat{F}(x_i)))} = \frac{1}{\hat{\Phi}'(\hat{x_i})}$$

$$\frac{d\hat{F}(x_j)}{dx_i} = \frac{\partial \hat{F}(x_j)}{\partial x_i} + \sum_{k=0}^{M} \frac{\partial \hat{F}(x_j)}{\partial g_k}\frac{dg_k}{dx_i}$$

$$\frac{\partial \hat{F}(x_j)}{\partial x_i} = \frac{1}{M+1}\frac{\frac{dx_j}{dx_i}}{g_{k+1} - g_k}$$

$$\frac{\partial \hat{F}(x_i)}{\partial g_k} = \frac{1}{M+1}(\frac{x_i - g_k}{(g_{k+1} - g_k)^2} - \frac{1}{g_{k+1} - g_k}), g_k \leq x_i < g_{k+1}$$

$$\frac{\partial \hat{F}(x_i)}{\partial g_{k+1}} = -\frac{1}{M+1}\frac{x_i - g_k}{(g_{k+1} - g_k)^2}, g_k \leq x_i < g_{k+1}$$

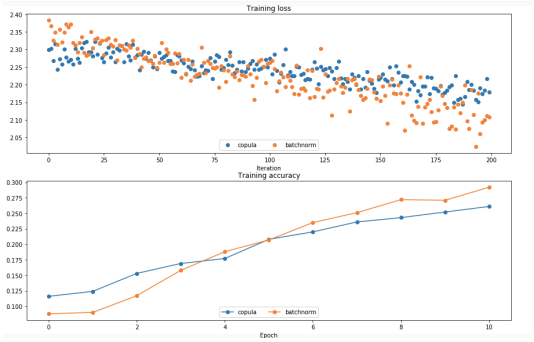
$$\frac{dL}{dx_i} = \sum_{j=1}^{N} \frac{\partial L}{\partial y_j}\gamma\frac{d\hat{x_j}}{d\hat{F}(x_j)}\frac{d\hat{F}(x_j)}{dx_i}$$

4. Experiment

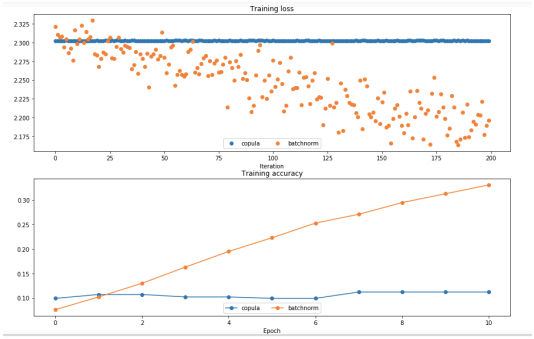
Both Copula and Gaussian normalization have a similar problem during backpropagation when dealing with test values which exceed the bounded estimates created during training. These outliers will always be mapped to the same value ($[-0.5, 0.5]$ in Copula normalization and $[\epsilon, 1-\epsilon]$ in Gaussian normalization). Samples who's value exceeds the estimated value of the inverse CDF are all assigned to the same value. For this reason, backpropagation is not an effective method to learn about these samples through these normalization layers. This issue becomes apparent when we look at the affect the effects Copula and Gaussian normalization has when used on trained using batch sizes much smaller than the actual dataset. We first present each case individually and then analyze the results together.

4.1. Copula Normalization

Below are the results of a test comparing Copula to batch normalization using a shallow 2 layer fully connected network with 100 units in each hidden layer on a randomized dataset.

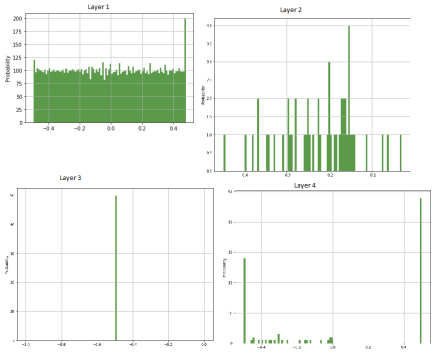


In this example we see that batch norm seems to converge faster than Copula normalization in the case of shallow networks. Things look even worse when we look at the effects on a deeper network:



The result is that in deeper (10 layer) networks, copula normalization fails to learn at all. The reason for this becomes apparent when we analyze the distribution of out-

puts at each layer of the network after copula normalization. The diagram below showcases the output distributions of the first 4 layers in our feedforward network after training:



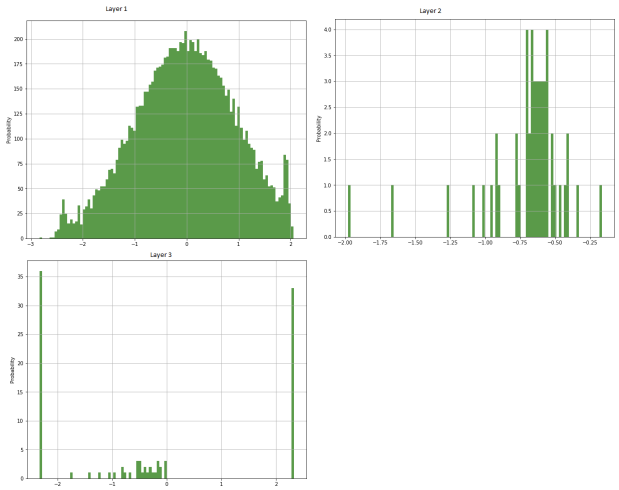
At the very start of the network, normalization is working correctly but already we see some problems with the distribution, namely a lot of values are clustered around 1 and 0. This property quickly propagates itself as we go through the layers and it becomes apparent that, since the input data becomes so skewed towards the two extremes, when a new copula layer is trained, its estimated CDF will be heavily influenced by the large amount of values greater than the expected mean. Since we are converting the outputs of each layer into a uniform distribution, this effectively shifts the whole CDF in one direction.

Shifting the CDF of the inputs away from the true center of the distribution causes large ranges of values to be evaluated in the same range of values after applying the inverse CDF to the uniform distribution in the following layer. This is apparent in the transition from layer 2 to layer 3 where we see how there only seems to be one group that all the values fall into. When passing over to the next layer, this causes the estimated CDF of the input each subsequent layer to mimic a step function in the most extreme cases and causes the normalization layer to strip inputs of critical features. There are several possible solutions to this problem of which discussions are deferred until after we provide similar results for Gaussian normalization.

4.2. Gaussian Normalization

Gaussian normalization performs even more poorly than Copula normalization when it comes to learning in deep networks in that it fails to reduce the loss at all. When first training very small networks (2 layers and less than 50 hidden units) with Gaussian normalization, accuracy would quickly converge and stay at a remain constant while the losses exploded towards infinity. Instead of looking at a graph of accuracy (which is just a straight line), we instead analyze solely the distributions of its hidden layers to see what the cause of this failure to learn is.

The problem lies in the high sensitivity of Gaussian layer with respect to its hyper-parameters, specifically in the estimated CDF of x, we assigned values which exceeded our initial bounds with a value of ϵ and $1-\epsilon$ respectively. When this value is transformed by the inverse CDF of the Gaussian, it can become extremely large in both directions depending on the ϵ and sample picked. In the case where the batch size is too small to reliably estimate the data, the output of the Gaussian normalization layers look as follows:



Like Copula normalization, it seems the first layer does a decent job of normalizing however, the batch was not quite good enough to capture the full range of the data distribution. This causes the output of layer 2 to be heavily skewed to the right as the estimated CDF for layer 2 has become over saturated with high inputs. This in turn causes the distribution seen in layer 3. Once the output of a layer looks like the output of layer 3, the rest of the network is almost certainly doomed to repeat the same patterns every layer down as, similar to what was seen in Copula normalization, the estimated CDF of each layer becomes almost like a step function. Coupled with the fact that if ϵ is chosen to be very small, we quickly get outputs from these layers that are enormously large in both the positive and negative directions. Additionally, the inverse CDF of the Gaussian distribution cannot be computed in a numerically stable way when values are close to 0 or 1.

Even when the batch size is more adequately chosen, the stability of the network still depends extremely closely on the hyper-parameters of the network. Most notably, the learning rate can quickly cause the output of a layer to become skewed if the weights of the previous affine layer are changed too fast.

4.3. Proposed Solutions

The main issue with both Copula and Gaussian normalization stems from the fact that they require an estimate of

the CDF of the input. This requires us to assume the input distribution to be bounded in some way. Due to this restriction both are heavily influenced by the parameters and hyperparameters of their model which are difficult to fine tune when a network becomes deep or when the data set becomes large. They are also heavily dependent on the batch from which they are trained. Both exhibit undesirable behavior when trained on deep architectures as the CDF can become skewed and subsequently map many inputs to the same output value from which it is impossible to recover.

The most obvious solution would be to increase the batch size used however, this has the drawback of making training extremely slow, especially for Gaussian normalization as the backpropagation for the Gaussian case includes computing the costly derivative of the erf function which involves taking e to a floating point power. Additionally, a vectorized solution for the gradient of both Copula and Gaussian normalization does not seem easily realizable.

Another issue is that unlike batch normalization, both Copula and Gaussian normalization do not have the option of undoing their effects on their inputs. This is again due to the estimated CDF required in their computations. Gaussian normalization has the additional issue of not having a linear CDF function.

A possible fix to this problem could be to instead of having the batches estimate the whole CDF of the input, we could have the batches only estimate a large portion of the input. For example, the batch could estimate a uniform distribution from [-0.49, 0.49] and where we assign an arbitrarily small factor to values which exceed these bounds akin to the leaky ReLU implementation. We can also do something similar for Gaussian normalization and additionally add in parameters in addition to β and γ to calculate the inverse CDF.

The drawback is twofold for this solution. The first is that we can no longer be certain that the variance will be preserved and secondly it would make computing the gradients even more costly by making the forward function even more piecewise.

5. Conclusion

Batch normalization already does a great job of speeding up and stabilizing the training process with very little overhead. Copula and Gaussian normalization on the other hand have a slew of issues ranging from numerical instability to slow training times and expensive gradient overheads. The naive fixes introduce even more problems affecting runtime and and altering desired features such as retaining a variance of 1. It seems from the tests run here that these new normalization methods are just not worth using as they do not seem to perform better with the same amount of parameters.

Some possible future directions of work could involve extending the methods experimented on here to convolutional or recurrent architectures, and possibly exploring whether a better method of vectorizing the gradient computations exists.

References

[1] Erik Learned-Miller *Copula Normalization* https://drive.google.com/file/d/0B-0OtUj$_G$j7nY0pzQ

[2] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift *In Proceedings of ICML,* pages 448 - 456, 2015.

[3] Andrej Karpathy, Fei-Fei Li, Justin Johnson, Serena Yeung. *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*