The Effect of Pre-ReLU Input Distribution on DNN

Some formulations of Batch-Normalization

Shaojie Bai [1]

Advisor: J. Zico Kolter [1]

[1] Carnegie Mellon University School of Computer Science

10 May 2017

Table of Contents

Backgrounds

Recent years have witnessed great success in the usage of deep neural network (DNN) in various tasks.

- VGG, AlexNet, AlphaGo, etc.

So what is a DNN ? The idea is straightforward.

1. Layers
2. Links
3. Non-linearities

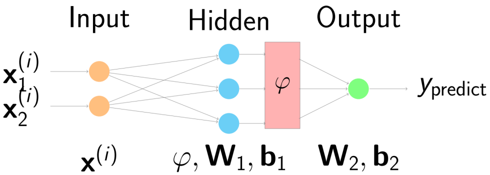


FIGURE – A simple 2-layer network

In this example :

$$y_{\text{predict}} = \mathbf{W}_2\left(\varphi(\mathbf{W}_1(\mathbf{x}^{(i)}) + \mathbf{b}_1)\right) + \mathbf{b}_2$$

Backgrounds

A neural network is able to update its parameters to make itself better, as long as there is a metric for what's good/bad— a loss function ℓ :

$$\mathbf{W}^{+} \leftarrow \mathbf{W} - \alpha \cdot \nabla_{\mathbf{W}} \ell(\mathbf{x}^{(i)}, \mathbf{W}, \mathbf{b}, \dots) \qquad \text{(GD)}$$

Backpropagation

- GD, Adam, Adagrad, RMSProp, etc.

Activation functions (nonlinearities)

- ReLU (Rectified Linear Unit), Sigmoids, etc.

Input distribution

- Batch-normalization

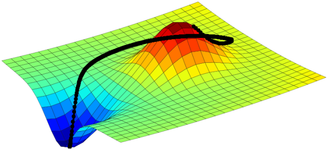


FIGURE – Example of a GD trace

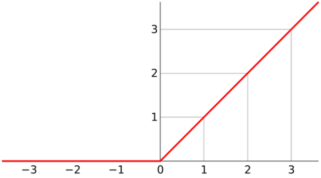


FIGURE – ReLU = max(0, x)

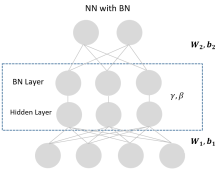
Backgrounds

In late 2015, Szegedy and Ioffe proposed the method of adding a **batch-normalization (BN)** layer use before ReLU to accelerate training and testing convergences.

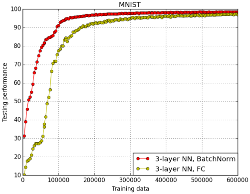
Idea : normalize the input minibatch's data so that it has zero mean and unit variance :

$$\hat{x}^{(i)} = \gamma^{(i)} \frac{x^{(i)} - \mu^{(i)}}{\sqrt{{\sigma^{(i)}}^2 + \epsilon}} + \beta^{(i)}$$

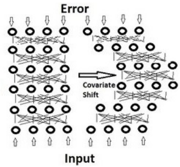
where $\mu^{(i)}$ is the mean of this minibatch and $\sigma^{(i)}$ is the standard deviation.



(a) BN layer



(b) Convergence : BN vs. no-BN



(c) Internal covariate shift

.

Learnable Transformations

Compare the distribution pattern for pre-BN data vs. post-BN (pre-ReLU) data :

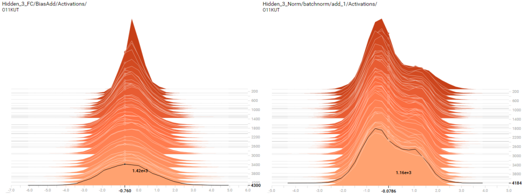


FIGURE – Pre-ReLU distribution in a network using BN (setting 1) on MNIST. Left : pre-BN. Right : post-BN.

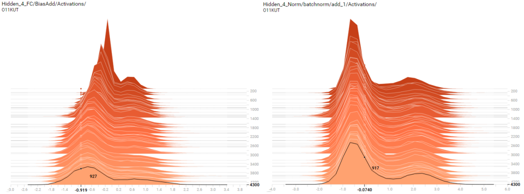


FIGURE – Pre-ReLU distribution in a network using BN (setting 2) on MNIST. Left : pre-BN. Right : post-BN.

Learnable Transformations

In general, we observed that surprisingly, BN+ReLU tends to transform the data into a bimodal shape. In particular, a higher, narrower peak at the near-zero negative side, as well as a shorter, wider peak at the farther positive side.

Idea : if this is the shape DNN prefers... we can try to simulate it !

- Steeper slope at around 0
- Gradually flattened slope as we move farther from the origin
- At least twice differentiable

Example :

$$f(x,a,b) = \begin{cases} \sqrt{(ax)^b + \frac{1}{4}} - \frac{1}{2} & x \geq 0 \\ -\sqrt{(-ax)^b + \frac{1}{4}} + \frac{1}{2} & x < 0 \end{cases}$$

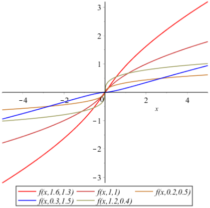


FIGURE – Learnable transformation example : *square-root-shift* functions ; *a* and *b* are learnables

Moment-Matching

A more careful and statistical formulation of Batch-Normalization.

- Think of BN as an attempt to match the empirical data distribution so that its 1st moment [1] becomes 0 and 2nd moment becomes 1.
- Target : $\mathcal{N}(0, 1)$ (which has moments 0,1,0,3,0,15,...)
- What if we match the moments to degrees higher than 2 ? For instance, $k = 6$?

Challenge : Matching 2 moments is very easy (affine transformation). Generally no reliable one-step method for higher-degree moment match.

1. In general, the i^{th} statistical moment is defined as $\mathbb{E}[X^i] = \int x^i p(x)\, dx$

Moment-Matching

Generally, given input data **x** the goal of this method is two-fold :

1. Find optimal distribution for certain moments

- Find distributions $p_1(x)$ and $p_2(x)$ that satisfy the empirical and target moments, respectively :

$$\mathbb{E}_*[x^i] = \int x^i p_*(x)\, dx = \begin{cases} \hat{\mu}_i = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j^i & \text{empirical} \\ \mu_i & \text{target} \end{cases} \quad \text{for } i = 1, \ldots, k$$

2. Transform the data

- Match the quantiles of the two distributions, and transform the input **x** by :

$$\hat{\mathbf{x}}_j = F_2^{-1}(\underbrace{F_1(\mathbf{x}_j)}_{\text{the quantile of } \mathbf{x}_j \text{ in } F_1}) \iff F_2(\hat{\mathbf{x}}_j) = F_1(\mathbf{x}_j)$$

where F_1 and F_2 are the cumulative distribution functions (cdf) of p_1 and p_2.

Moment-Matching

Using maximum Shannon entropy $H(p)$ as the metric for the optimal p_1 and p_2, we can re-formulate the problem by considering its dual problem, which must be convex :

$$\max_p H(p) = -\int p(x)\log p(x)dx \ \underline{\text{s.t.}} \ \mathbb{E}_*[x^i] = \mu_i, i \in [k]$$

$$\iff \min_\gamma \mathcal{L}(\gamma) = \int_S \underbrace{\exp\left(\sum_{i=0}^k \gamma_i T_i(x) - 1\right)}_{\text{Optimal } p_* \text{ from KKT}} dx - \sum_{i=0}^k \gamma_i T_i(x) \quad (\lambda \text{ are dual variables})$$

This allows us to use Newton's method with backtracking line search :

$$\gamma^+ = \gamma - t(\nabla^2_\gamma \mathcal{L})^{-1}\nabla_\gamma \mathcal{L} \qquad (t \text{ from line search})$$

As for the second phase, $F_2(\hat{\mathbf{x}}_j) = F_1(\mathbf{x}_j)$ can also be solved using Newton's method.

However, the detailed steps is much more complicated :

- Inefficient integration
- Sensitivity of γ_i corresponding on higher degrees compromises stability of convergence
- Hard to backpropagate

Copula Transform

We can impose an even stronger requirement on the target of the transformation.

- **Learnable transformations** : observation-based
- **Moment-matching** : match to some target moments up to a finite degree k
- **Copula transform** : a "perfect" transform to match a specific distribution

Definition (informally) : For a random vector $(X_1, \ldots, X_d)$ where each X_i has a continuous probability distribution function, the random vector of its cdfs :

$$(U_1, \ldots, U_d) = (F_1(X_1), \ldots, F_d(X_d))$$

has uniformly distributed marginals. The joint cdf of $(U_1, \ldots, U_d)$ is the copula of $(X_1, \ldots, X_d)$.

A bit too abstract ?

Copula Transform

Basic ideas/steps for copula transform :

Copula transform

- SortIndex($\mathbf{x}$) : For each $\mathbf{x}_i$ in the input, find its index in the sorted version of $\mathbf{x}$. Name this index t_i.

In short, for a target distribution with cdf Φ :

$$\hat{\mathbf{x}} = \Phi^{-1}\left[\frac{\text{SortIndex}(\mathbf{x}) + 0.5}{n = \text{len}(\mathbf{x})}\right]$$

(Add 0.5 to balance indexing bias)

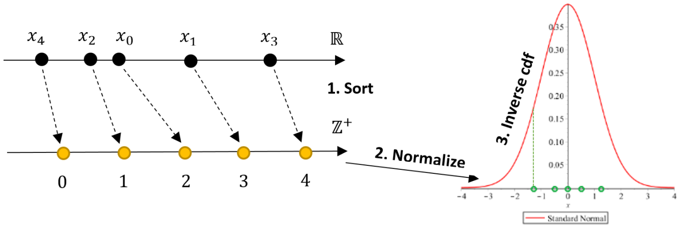


FIGURE – An example of a copula transform on a length-5 input data $\mathbf{x}$

Copula Transform

However, sorting is not differentiable. This makes backpropagation impossible. Instead, we can approximate the sort.

Q : What does it mean when we say $\mathbf{x}_i$ is at index 2 in the sorted version of $\mathbf{x}$?

Note that

$$\mathbf{x}\mathbf{1}^T - \mathbf{1}\mathbf{x}^T = \begin{bmatrix} x_1 & x_1 & \dots & x_1 \\ x_2 & x_2 & \dots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_n & \dots & x_n \end{bmatrix} - \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_1 & x_2 & \dots & x_n \end{bmatrix} = [x_i - x_j]_{i,j}$$

Therefore, the # of positive values in row i is exactly the index of $\mathbf{x}_i$:

$$\begin{aligned} \textbf{SortIndex}(\mathbf{x}) &= 1_{\{y>0\}}(\mathbf{x}\mathbf{1}^T - \mathbf{1}\mathbf{x}^T) \cdot \mathbf{1}^{n\times 1} \\ &\approx \left[\sigma(\gamma(\mathbf{x}\mathbf{1}^T - \mathbf{1}\mathbf{x}^T)) - \frac{1}{2}I\right] \cdot \mathbf{1}^{n\times 1} \end{aligned}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the sigmoid function and γ is a strength factor for the approximation.

Copula Transform

The approximation is good in most of the cases, assuming we have a reasonable value for γ— which is learnable through backpropagation.

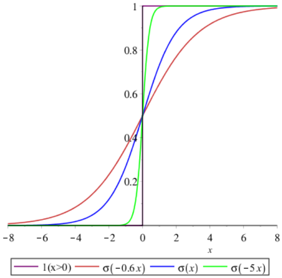


FIGURE – Sigmoid approximation of the indicator function with different strengths

Copula transform formula with the learnables θ, β, γ

$$\hat{\mathbf{y}} = \theta\hat{\mathbf{x}} + \beta = \theta\Phi^{-1}\left[\frac{\sigma(\gamma(\mathbf{x}\mathbf{1}^T - \mathbf{1}\mathbf{x}^T)) \cdot \mathbf{1}^{n\times 1}}{n = \text{len}(\mathbf{x})}\right] + \beta$$

Learnable transformations

Two example functions :

$$\text{log-shift} : g(x, a, b, c) = \begin{cases} a \cdot \log((bx)^c + 1) & x \geq 0 \\ -a \cdot \log((-bx)^c + 1) & x < 0 \end{cases}$$

$$\text{sqrt-shift} : h(x, a, b) = \begin{cases} \sqrt{(ax)^b + \frac{1}{4}} - \frac{1}{2} & x \geq 0 \\ -\sqrt{(-ax)^b + \frac{1}{4}} + \frac{1}{2} & x < 0 \end{cases}$$

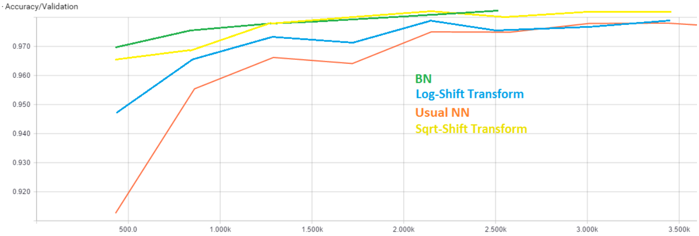


FIGURE – Simulating the bimodal shape using transformations proposed yielded pretty good results. With learnable transformations, the convergence is slightly worse than BN, but still much better than usual DNN training.

Moment-Matching

- The most difficult part of moment-matching formulation of BN is the stability of the convergence. This problem was mentioned in some prior work [Abramov 2010], but no completely reliable method was found.
- We used Hermite polynomial basis for $T_i(x)$ instead of standard basis for stablization.
- The support of the integration is on $3\times$ range of the input data.
- We used Gaussian quadrature to estimate the integration.

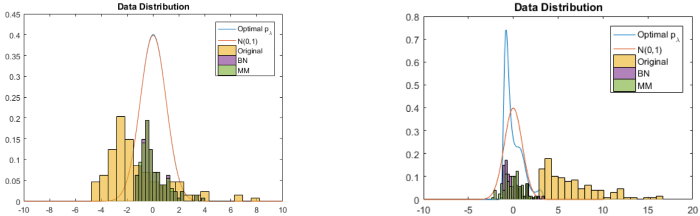
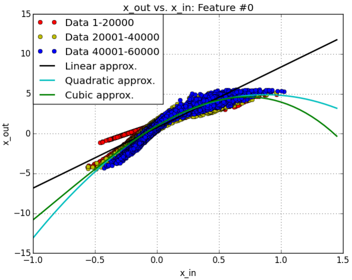


FIGURE – Moment-matching on some sample MNIST data inputs

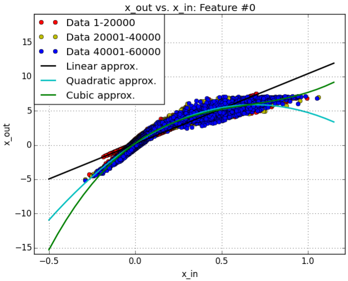
Status : Already achieved stable convergence (yay). Ongoing research to optimize backprop and simplify test-phase behavior.

Copula Transform

We used linear regression in the testing phase to approximate the copula transformation function. A choice by observation :



(a) After 3 epochs of training



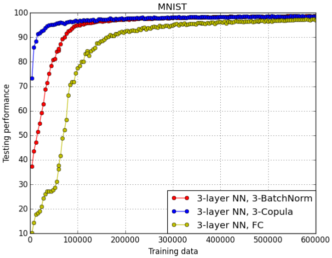
(b) After 6 epochs of training

FIGURE – Data input vs. output for copula transformation training

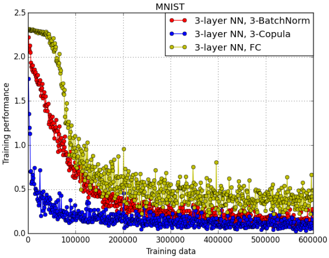
Copula Transform

Training and testing on MNIST and CIFAR-10 datasets revealed encouraging results. In terms of convergence in both phases, copula transform is able to outperform batch-normalization :

- MNIST dataset (handwritten digit) :



(a) Testing accuracy convergence



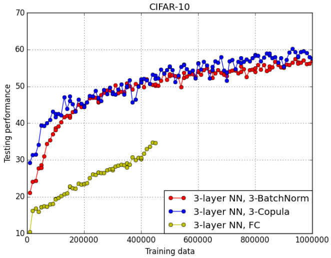
(b) Training loss convergence

FIGURE – Convergence comparison on MNIST : CT vs. BN vs. Regular

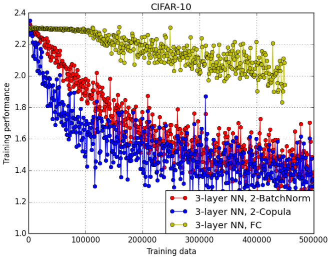
Copula Transform

Training and testing on MNIST and CIFAR-10 datasets revealed encouraging results. In terms of convergence in both phases, copula transform is able to outperform batch-normalization :

- CIFAR-10 dataset (image classification) :



(a) Testing accuracy convergence



(b) Training loss convergence

FIGURE – Convergence comparison on CIFAR-10 : CT vs. BN vs. Regular

Copula Transform

Tables for comparison on MNIST dataset performance :

MNIST-training (loss)					
	25K	50K	75K	100K	125K
BatchNorm	1.738	1.243	1.050	0.632	0.634
Copula	**0.717**	**0.270**	**0.246**	**0.167**	**0.361**
Regular	2.292	2.135	1.577	0.764	1.034

MNIST-testing (accuracy)					
	25K	50K	75K	100K	125K
BatchNorm	62.84%	80.84%	91.59%	95.01%	95.90%
Copula	**92.54%**	**95.15%**	**96.03%**	**96.48%**	**96.84%**
Regular	24.41%	28.78%	56.44%	77.54%	82.49

Future work

- Further optimize the copula transform so that it is
  1. more efficient to run on convolutional layers (i.e. image inputs that can have muliple channels) ;
  2. parallelizable on CUDA to further speed up training.

- Optimize the backprop in moment-matching, and define a testing behavior for it.

- So far the focus has been on $\mathcal{N}(0, 1)$. But from learnable transformations and copula transform we can see that many shapes and distributions are worthy of further explorations.

Acknowledgements

Many thanks to Zico and Brandon for their guidance on my senior thesis !

FIGURE – Zico Kolter

FIGURE – Brandon Amos