A Mean Field Theory of Binary Neural Networks

Zhichao Wang
wangzc@tamu.edu

December 12, 2018

1 Introduction and Motivation

As we all know, the initialization of weights and biases in deep neural networks can dramatically affect the learning speed. For example, the exponential vanishing or exploding of gradients during back-propagation may occur if the selections of the weight initialization and the activation function are not appropriate. Therefore, it is important to understand the theoretical properties of untrained random networks using some mathematical and statistical tools. There are different statistical properties of random networks people have studied recently: the singular value distribution of a deep neural network's input-output Jacobian, which can be considered as dynamical isometry of the system (see [7]); the propagation of the variance and correlation through the networks (see [3]); some isometric property of the expectation and variance to maintain some numerical stability (see [8]). By studying these properties and analyzing the asymptotic behaviors of the random neural networks, we can find some conditions on the selections of weight initialization and activation function for neural networks.

For a fully-connected random neural network of depth L, with widths $\{N_l\}_{1\leq l\leq L}$, we always set weights W^l_{ij} and bias b^l_i have normal distributions $\mathcal{N}(0, \frac{\sigma_w^2}{N_{l-1}})$ and $\mathcal{N}(0, \sigma_b^2)$. For any input $x \in \mathbf{R}^d$, the propagation of this input is given by

$$y^1_i(x) = \sum_{j=1}^{d} W^1_{ij} x_j + b^1_i \tag{1}$$

and

$$y^l_i(x) = \sum_{j=1}^{N_{l-1}} W^l_{ij} \phi(y^{l-1}_j(x)) + b^l_i. \tag{2}$$

Considering $y^l_i(\cdot)$ as a Gaussian process, Saufiane Hayou et al. established a theory in [3] to study the propagation of variance and correlation of the Gaussian process through this fully-connected network. Later, a theoretical understanding of convolutional neural networks with random parameters is built using similar methods in [2]. Recent work, [1], reveals such theoretical result for recurrent neural networks. We would like to follow this trend to study other architectures under a similar setting and develop their theories in different situations.

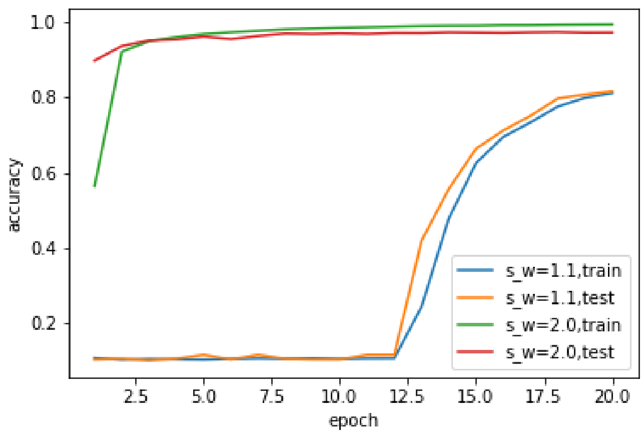


Figure 1: Different kinds of initialization for deep fully-connected neural networks show very different performance.

2 Binarized Neural Networks

In this project, we mainly focus on studying Binarized Neural Network (BNN), which is introduced by Courbariaux et al. in [13].

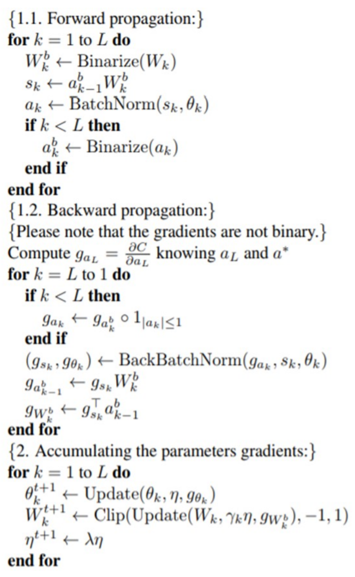
As we know, each iteration of training a deep neural network involve three steps: forward propagation of feature information, back propagation and parameters update (See Figure 1). The mean field theory mainly deal with the initialization of the neural networks, considering the propagation through the random neural networks. Therefore, this theory main study the first two steps, forward propagation and back propagation of gradients at the beginning of the training. We want to use the mean field theory to study the forward propagation of statistical information through the deep BNN and also study the back propagation of gradients. The goal of studying forward propagation of statistical information is to ensure that the random neural networks have a high expressivity to keep information through the deep networks. For the study of the back propagation of gradients, we want to avoid the vanishing and exploding of gradients when training the deep neural networks, which is necessary for updating the parameters and getting a good accuracy after training. The setup for BNN is stated as below. We define a entry-wise function $B(\cdot)$ as the binarized function, which may have different definitions in different situations. We will introduce it next. As stated in [13], the activation function is also the binarized function which means we also use binary input for computations.

$$y_i^l = \sum_{j=1}^{N_l} B(W_{ij}^l) B(y_j^{l-1}) + b_i^l, \tag{3}$$

$$s_j^l = B(y_j^l), \quad 0 \le l \le L-1, \tag{4}$$

and

$$y_i^L = s_i^L = \sum_{j=1}^{N_L} B(W_{ij}^L) s_j^{L-1} + b_i^L. \tag{5}$$



```
{1.1. Forward propagation:}
for $k = 1$ to $L$ do
  $W_k^b \leftarrow \text{Binarize}(W_k)$
  $s_k \leftarrow a_{k-1}^b W_k^b$
  $a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)$
  if $k < L$ then
    $a_k^b \leftarrow \text{Binarize}(a_k)$
  end if
end for
{1.2. Backward propagation:}
{Please note that the gradients are not binary.}
Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$
for $k = L$ to $1$ do
  if $k < L$ then
    $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}$
  end if
  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$
  $g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$
end for
{2. Accumulating the parameters gradients:}
for $k = 1$ to $L$ do
  $\theta_k^{t+1} \leftarrow \text{Update}(\theta_k, \eta, g_{\theta_k})$
  $W_k^{t+1} \leftarrow \text{Clip}(\text{Update}(W_k, \gamma_k \eta, g_{W_k^b}), -1, 1)$
  $\eta^{t+1} \leftarrow \lambda \eta$
end for
```

Figure 2: The general Algorithm of training a BNN (see [13]). To train a deep neural network with binary weights we only binarize the weights during the forward process. Then we use such binarized network to do back propagation and get the upgrade for the original weights. After updating the parameters, we binarize the weights agian to the next training process. We can use any kinds of initialization, but we need to binarize them first before propagation.

Here we consider the fully-connected deep neural network with binary inputs and binary weights. We do not include dropout and batch normalization, which we will discuss separably later. For general convolutional neural networks, our methods and results also hold and we can get more restrictions on the initialization of weights. For more details, we refer to [2].

For BNN, the function $B(\cdot)$ should be the sign function ideally, which means we constrain both weights and activation to either $+1$ or -1. The advantage of doing that is from a hardware perspective. See [9, 10, 12, 13]. The binarized function $B(x)$, defined by

$$\bar{x} := Sign(x) = \begin{cases} -1, & x < 0 \\ 1, & 0 \leq x, \end{cases}$$

is not differentiable at zero and has zero derivatives otherwise. This is a bad property for back propagation so we need to modify it in practice. Also, notice that there is another stochastic version of binarization function defined by

$$B(x) = \begin{cases} -1, & \text{with probability } p = \sigma(x) \\ 1, & \text{with probability } 1 - p, \end{cases}$$

where σ is the hard sigmoid function defined by $\sigma(x) := \max(0, \min(1, \frac{x+1}{2}))$. This stochastic binarization is more interesting and reasonable but harder to implementation in practice. So we will not consider the second case in this project. Instead, in our project, we mainly consider the binarized function defined by

$$B(x) = \begin{cases} -1, & x \le -r \\ \frac{x}{r}, & -r < x < r \\ 1, & r \le x, \end{cases}$$

which is a hard tahn function, and we require $r \to 0$. In this sense, we have derivative of $B(x)$, which is $B^{'}(x) = \frac{1}{r}\mathbf{1}_{[-r,r]}(x)$, a simple function. However, our setting is a little bit different from the original BNN in [13], which only consider $r = 1$ in there algorithm. In that case, the derivative of "sign" function is $\mathbf{1}_{[-1,1]}(x)$. We will explain this in the following two sections.

3 Gaussian Initialization

We consider each $W_{ij}^l \sim \mathcal{N}(0, \sigma^2)$ and $b_i^l \sim \mathcal{N}(0, \sigma_b^2)$ for initialization and assume all different random variables are independent with each other. And then we use the mean field theory to study the propagation of statistical information through such BNNs. Recall equations (3), (4) and (5). If we analyze this network from the first layer, it is easy to see that y_i^1 is a Gaussian variable when width of the first layer goes to infinity by the Central Limit Theorem. In fact, y_i^1 is a sum of some independent random variables. Therefore, according to [3, 4, 5], we can conclude that for all layers, the processes $y_i^l(\cdot)$ are independent centred Gaussian processes of covariance K^l. Here, $y_i^l(a)$ and $y_i^l(b)$ represent different inputs of this Gaussian Process. Then, the mean field theory is a good way to study the forward propagation of such Gaussian Process. Our goal is to study the propagation of the information stored in the covariance of different inputs. Considering two different inputs, we want to the propagation of the covariance, whether it can propagate infinitely far away through this BNN or it will be collapsed.

We define the covariance q_{ab}^l at l-th layer by $q_{ab}^l = \mathbb{E}[y_i^l(a)y_i^l(b)]$. Then,

$$q_{ab}^l = \sigma_b^2 + N_{l-1}\mathbb{E}[B(W_{ij}^l)^2]\mathbb{E}[B(y_j^{l-1}(a))B(y_j^{l-1}(b))]. \tag{6}$$

Based on different definitions of binarized function, we can compute the above equation differently.

3.1 Case 1

If the function is a pure Sign function, then $B(W_{ij}^l)^2 = 1$ since W_{ij}^l is centered regardless with any distribution. Then, whatever the distribution of the weight entry is, the covariance q_{ab}^l is a function of N_{l-1}, which means as the width increasing, the covariance will explode exponentially. This a really bad situation comparing with general fully-connected neural networks and our following analysis. Therefore, we believe, from the point view of the mean field theory, this BNN setting is not good for propagation of the information and keep statistical information through deep neural networks. Table 1 shows the differences of case 1 and case 2 which we will consider in the following sections. Case 1 employs the Sign function for the forward propagation, while in Case 2 we use the $B(x)$ as the binary function. In these experiments, we use the same architecture, width 300 with 10 layers and batch size 64. DNN represents the original fully-connected neural network and we can see the accuracy will be best among these three after training several epochs because it keeps the total information of the weight matrices while training.

epoch	1	2	3	4
Case 1	0.8361	0.9330	0.9234	0.9473
Case 2	0.9421	0.9516	0.9560	0.9609
DNN	0.8990	0.9379	0.0.9526	0.9630

Table 1: Training accuracy with different parameters and training epochs.

3.2 Case 2

If the function is a hard tahn function defined in section 2, then we can go on the analysis of our mean field theory. In this case,

$$N_{l-1}\mathbb{E}[B(W_{ij}^l)^2] = \frac{N_{l-1}}{\sqrt{2\pi}}\int_{\mathbb{R}} B(\sigma y)^2 e^{\frac{-y^2}{2}} dy.$$

If

$$\sigma = \frac{r\sigma_w}{\sqrt{N_{l-1}}}, \tag{7}$$

then the above integral converges σ_w^2 by the Dominated Convergence Theorem. Then, when the weight initialization is

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{r^2\sigma_w^2}{N_{l-1}}\right),$$

the covariance is simplified as

$$q_{ab}^l = \sigma_b^2 + \sigma_w^2\mathbb{E}[B(y_j^{l-1}(a))B(y_j^{l-1}(b))]. \tag{8}$$

Based on this equation, we can directly apply the methods and results in [3, 5] to analyze the forward propagation of BNNs in this settings. Let $q_a^l = q_{aa}^l$ which is the variance of the Gaussian Process. Let correlation function be $c_{ab}^l = \frac{q_{ab}^l}{\sqrt{q_a^l a_b^l}}$. Then the variance function is determined by the following recursion formula:

$$q_a^l = \sigma_b^2 + \sigma_w^2 \int_{\mathbb{R}} B^2\left(z\sqrt{q_a^l}\right) Dz, \tag{9}$$

where Dz is the standard Gaussian density $\frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx$. We want know the fixed point of the above recursion formula. If this recursion formula has a non-trivial fixed point, then this network will eventually converge to this fixed point and keep this variance through the propagation. Let q_* be the fixed point of the recursion formula (9). Then, if the inputs a and b have the same variance $q_a^1 = q_b^1 = q_*$, then the correlation function has the following recursion formula:

$$c_{ab}^l = \frac{1}{q_*}\left(\sigma_b^2 + \sigma_w^2\int_{\mathbb{R}^2} B\left(u_1\right)B\left(u_2\right)D(z_1)D(z_2)\right), \tag{10}$$

where $u_1 = \sqrt{q_a^l}z_1$ and $u_2 = \sqrt{q_b^l}z_1\left(c_{ab}^l z_1 + \sqrt{1-(c_{ab}^{l-1})^2}z_2\right)$. By the definition of q_*, we can see $c = 1$ is a fixed point for the above recursion formula. Let

$c^l = 1 - \epsilon^l$, where c^l_{ab} is denoted by c^l. Then we have the asymptotic recursion formula of ϵ^l:

$$\epsilon^{l+1} = \epsilon^l \cdot \chi_1, \quad \chi_1 = \sigma_w^2 \int_{\mathbb{R}} B'\left(z\sqrt{q_a^l}\right)^2 Dz. \tag{11}$$

$\chi_1 = 1$ is called the edge of chaos. This is because when $\chi_1 < 1$, then the difference of one and correlation function c^l will decrease exponentially to zero, which means the correlation information of inputs a and b will disappear quickly after going through several layers. This is a bad case that we would not want to see. However, if $\chi_1 > 1$, then the difference would increase and become chaotic, which is even a worse situation for forward propagation and will interfere the training process dramatically. Therefore, keeping the parameter χ_1 staying one is a good choice to avoid the loss of some information of the input data. Therefore, the existence of the fixed point q_* and the restriction on χ_1 give us two equations for selecting the appropriate σ_w and σ_b given the parameter r.

$$q_* = \sigma_b^2 + \sigma_w^2 \int_{\mathbb{R}} B\left(z\sqrt{q_*}\right)^2 Dz \tag{12}$$

and

$$\frac{r^2}{q_*\sigma_w^2} = \int_{\frac{r}{\sqrt{q_*}}}^{\frac{r}{\sqrt{q_*}}} Dz = \Phi(\frac{r}{\sqrt{q_*}}) - \Phi(-\frac{r}{\sqrt{q_*}}), \tag{13}$$

where Φ is the distribution of the standard Gaussian random variable. Based on equation (12) and equation (13), we can compute the fixed point q_* or select suitable (σ_w, σ_b). As we can see in Figure 3, 4 and 5, the line represents the

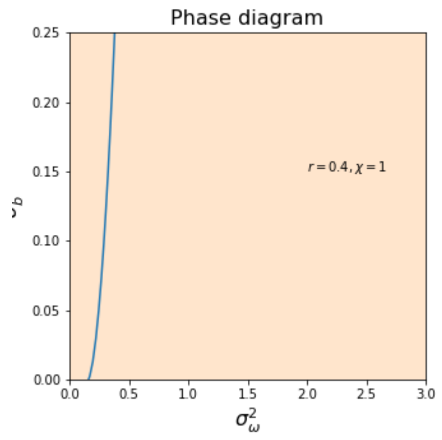


Figure 3: Phase diagram when r=0.4.

suitable (σ_w, σ_b) to obtain the edge of chaos in our BNN architecture. Comparing with these figures, it is easy to see when r goes to zero, which means the function $B(x)$ becomes a sign function, then the edge of chaos of σ_w also goes to zero. In other world, we need to choose the parameter σ_w smaller enough when r is small to ensure the conservation of the forward propagation. For example, when $r = 0.1$, $(\sigma_w, \sigma_b) = (0.1, 0.0)$ is on the edge of chaos. Also, we can

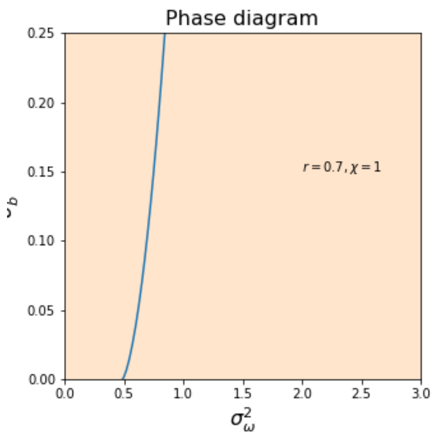


Figure 4: Phase diagram when r=0.7.

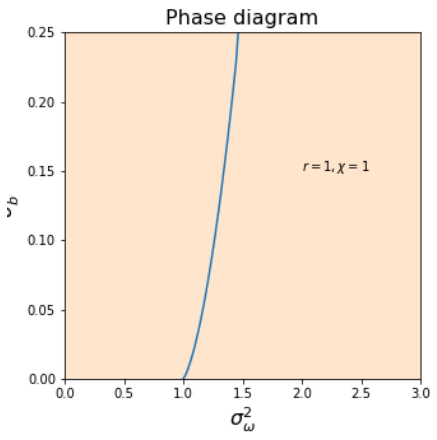


Figure 5: Phase diagram when r=1.0.

use Hermitian expansion of the activation function, $B(x)$, to get a mathemtical restriction for $\chi_1 = 1$. To get a stable dynamic system near the $c = 1$, we need to have the following conditions:

$$0 = \int_{\mathbb{R}} B\left(z\right) Dz \quad \text{and} \quad \sigma_b = 0. \tag{14}$$

Obviously, $B(x)$ satisfies this condition and we only need to choose $\sigma_b = 0$ and corresponding σ_w on the edge of chaos. Therefore, for instance, $(\sigma_w, \sigma_b) = (0.1, 0.0)$ is a good initialization according to our analysis. Besides, these conditions indicate that binarized function is a little bit better that ReLU in this sense. However, there may be other aspects indicating the advantages of ReLU.

3.3 Gradients of Back Propagation

Considering a loss function E, we now study the back propagation. Our goal is analyzing the back propagation to avoid the exploding and vanishing of the gradients since these situations may ruin the upgrade of the learnable parameters during training and make the learning process inefficient. We only update the original real-valued weights and need to consider the recursion formula of $\delta_i^l = \frac{\partial E}{\partial y_i^l}$. This is because

$$\frac{\partial E}{\partial W_{ij}^l} = \frac{\partial E}{\partial B(W_{ij}^l)} \frac{\partial B(W_{ij}^l)}{\partial W_{ij}^l} = \delta_i^l B(y_j^{l-1}) B'(W_{ij}^l). \tag{15}$$

This computation shows that our architecture will never avoid the vanishing and exploding of the gradients if we want r to go to zero asymptotically. In other words, if r is too small, then either $\frac{\partial E}{\partial W_{ij}^l}$ will be almost zero when W_{ij}^l is not close to zero, or $\frac{\partial E}{\partial W_{ij}^l}$ will be huge for updating when W_{ij}^l is pretty close to zero. This conclusion is actually consistent with our experiments, which shows when r is small, the training process is almost frozen and cannot move in any direction along the loss manifold. However, when $r = 1$, which is the setting for [13], then the learning speed is fast and the algorithm can quickly find a local minimal point in the loss manifold since the gradients will not be vanishing or exploding in this situation. When $r = 1.0$, the suitable parameters are $(\sigma_w, \sigma_b) = (1.0, 0.0)$, as we discussed before (see Figure 5). Table 2 shows the training accuracy corresponding to the analysis of back propagation of gradients. As we see, when r is small, the learning process is dead and no improvement at all, while the learning accuracy is increasing and precise when r is not small.

epoch	1	2	3	4
r=0.1, $(\sigma_w, \sigma_b) = (0.01, 0.0)$	0.097400	0.097400	0.097400	0.097400
r=1.0, $(\sigma_w, \sigma_b) = (1.0, 0.0)$	0.942100	0.951600	0.956000	0.960900

The recursion formula of $\tilde{q}_a^l := \mathbb{E}[(\delta_i^l)^2] = \mathbb{E}[(B'(y_i^l))^2] \sum_{j=1}^{N_{l+1}} \tilde{q}_a^{l+1} \mathbb{E}[(B(W_{ji}^l))^2]$ can be simplified in different cases. If $B(x) = Sign(x)$, then

$$\tilde{q}_a^l = N_{l+1} \tilde{q}_a^{l+1} \chi_1, \tag{16}$$

which means variance of the gradient is not small and will explode through the back propagation. However, if we use our definition of hard tanh function as $B(x)$, then

$$\tilde{q}_a^l = \frac{N_{l+1}}{N_l} \tilde{q}_a^{l+1} \chi_1, \tag{17}$$

which means the gradients is very stable along the back propagation.

Again, let us study the gradient $\frac{\partial E}{\partial W_{ij}^l}$, based on the computation of $B'(x) = \frac{1}{r} \mathbf{1}_{[-r,r]}(x)$. To make $B'(W_{ij}^l)$ none zero, we need the probability

$$P(|W_{ij}^l| < r) = P(|\frac{r\sigma_w}{\sqrt{N_{l-1}}} Z| < r) = P(|\frac{r\sigma_w}{\sqrt{N_{l-1}}} Z| < r) \approx 1.$$

This can be made when the width is very large. Therefore, vanishing of the gradients is not the actual case in our algorithm. But exploding may appear when r is too small.

4 Uniform Initialization

The reason we want include the uniform initialization is that in [13] they use such Golort Uniform initialization. And the performance on the BNNs is good enough to train a deep neural networks as fast as other architectures. Also, such initialization has not been studied by the mean field theory before. Consider the initialization $W_{ij}^l \sim \text{Uniform}(-A, A)$. We want to find a suitable in our mean field theory. Since we still set different entries of weight matrices are independent, we can apply Central Limit Theorem again to get the Gaussian Process in the forward propagation when the width goes to infinity. Notice that Golort Uniform initialization is Uniform$(-\frac{\sqrt{6}}{\sqrt{N_l+N_{l-1}}}, \frac{\sqrt{6}}{\sqrt{N_l+N_{l-1}}})$, which shows A should not be too large. Therefore, based on equation (6) and Case 2 in section 3, we can get

$$N_{l-1}\mathbb{E}[B(W_{ij}^l)^2] = 2N_{l-1}\int_0^A (x/r)^2 dx = \frac{2N_{l-1}}{3r^2}A^3.$$

In order to have the same equation as equation (8), we need

$$A = \left(\frac{3\sigma_w^2 r^2}{2N_{l-1}}\right)^{1/3}. \tag{18}$$

This parameter we got is a little bit different from Golort Uniform initialization and with lower order in the width. Using this setting, we can then apply the mean field theory here to get a better estimation for A to obtain a good forward propagation. Since the width is large, such A should be small. Then, when considering the back propagation, in terms of equation (15), we can still guarantee that the gradients will not vanish through the back propagation.

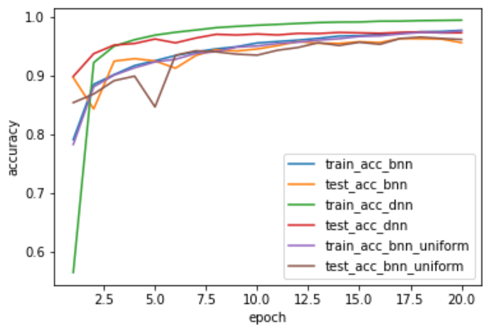


Figure 6: Comparing.

The above figure of several curves compares the test accuracy and training accuracy given different architectures and different kinds of initialization. As we can see, the original DNN performs better than the BNN after training, but DNN needs more time to train. For different kinds of initialization, the normal initialization (which are the blue and yellow lines) is a little bit better that the uniform initialization. Here, we use the same architecture. The width is 300 for each hidden layer and with 10 layers. The batch normalization is 64 with batch normalization or dropout. There is one thing that we do not know how to explain. The DNN actually spends much less time than the others. This ruins the motivation of studying the BNNs in fact.

5 Dropout and Batch Normalization

We also consider the influences the dropout and batch normalization on the information propagation using mean field theory. The idea comes from [5] and a recent long paper [14], which we believe is a good reference for statistical deep learning.

When we consider the dropout for each layer then equation (3), (4) and (5) need to be changed:

$$y_i^l = \frac{1}{\rho}\sum_{j=1}^{N_l} B(W_{ij}^l) p_j^l B(y_j^{l-1}) + b_i^l, \tag{19}$$

where p_j^l are independent Bernoulli random variables with parameter ρ. Then the recursion formula of variance function will become

$$q_a^l = \sigma_b^2 + \frac{1}{\rho}\sigma_w^2 \int_{\mathbb{R}} B^2\left(z\sqrt{q_a^l}\right) Dz. \tag{20}$$

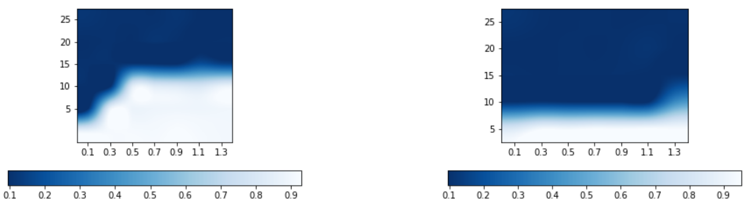
And the correlation function will be also changed. In this case, we do not have the convergence of the correlation function to one. This is because, when we input correlation function as one at the beginning, what we get will become

$$c_{ab}^l = 1 - \frac{1-\rho}{\rho q_*}\sigma_w^2 \int_{\mathbb{R}} B^2\left(z\sqrt{q_*}\right) D(z), \tag{21}$$

which means the correlation function would be strictly less than one in this case. Because of dropout, we do not have the critical point for training anymore. Therefore, in some sense, dropout limits the depth to which the information can be propagated in a deep neural network (see Figure 8 (b)).

For the batch normalization, the analysis based on mean field theory is pretty hard and needs more computations so we still do not get the final results. However, [14] has already considered this problem and has some rigorous computations. It is shown that the gradient signals grow exponentially in depth and that these exploding gradients cannot be eliminated by tuning the initial weight variances or by adjusting the nonlinear activation function. Indeed, batch normalization itself is the cause of gradient explosion. Also, batch normalization limits the depth to which the information can be propagated in a deep neural network (see Figure 8 (a)) in some sense.

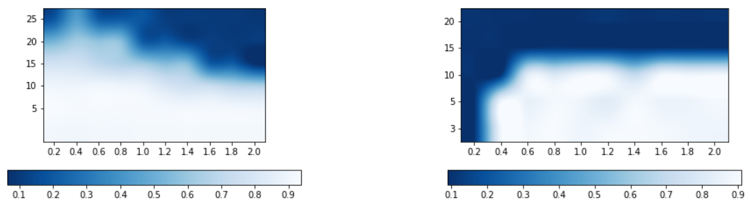
We did some experiments based on this theory, but the results do not match with such theory. We believe we need more time training such neural networks to get a more convincing result in such setting. Here, the four figures are implemented with only one epoch in very deep neural networks so it takes much time to get such figures. Figure 7 is to compare the DNNs and BNNs. The y label is the number of hidden layers. The x label is the parameter σ_w for normal initialization. For Figure 7 (a), we use pure fully-connected binary neural networks with width 300 at each layer. For Figure 7 (b), we use



(a) BNN

(b) DNN

Figure 7: Comparing BNNs and DNNs with different numbers of layer and parameter σ_w. Blue means low accuracy while white means high accuracy.



(a) Batch Normalization and dropouts.

(b) Only one dropout at first layer.

Figure 8: Comparing the influence of Batch Normalization and Dropout with different numbers of layer and parameter σ_w.

fully-connected deep neural networks with hard sigmoid function as activation function. As we can see, BNN performs better than DNN in depth and in terms of initialization. This because we only trianed one epoch. If we train these networks long time, DNN should have better accuracy. But this difference shows that BNN converges to its optimal states quicker that DNN because of binarization in training process. For Figure 8, we use the same architecture and initialization as Figure 7 (a) except the batch normalization and dropout. Figure 8 (a) shows a better performance if we add dropout and batch normalization after each hidden layer. In Figure 8 (b), we only add one dropout at the first hidden layer. This dropout actually has the performance when σ_w is not too small. Comparing Figure 8 (a), (b) and Figure (a), it seems to show that batch normalization improve the performance in depth and propagates the information deeper in a BNN when σ_w is small, while the dropout limits such propagation in depth.

The batch size of each experiment is 16. And all the experiments are based on MNIST. In future, we may need other data for training and testing to make the experiment more practical and convincing.

References

[1] Chen, Minmin and Pennington, Jeffrey and Schoenholz, Samuel S, *Dynamical Isometry and a Mean Field Theory of RNNs: Gating Enables Signal*

Propagation in Recurrent Neural Networks. arXiv preprint arXiv:1806.05394, 2018.

[2] Xiao, Lechao and Bahri, Yasaman and Sohl-Dickstein, Jascha and Schoenholz, Samuel S and Pennington, Jeffrey, *Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks.* arXiv preprint arXiv:1806.05393, 2018.

[3] S Hayou, A Doucet, J Rousseau, *On the Selection of Initialization and Activation Function for Deep Neural Networks.* arXiv preprint arXiv:1805.08266, 2018.

[4] Poole, Ben and Lahiri, Subhaneil and Raghu, Maithra and Sohl-Dickstein, Jascha and Ganguli, Surya, *Exponential expressivity in deep neural networks through transient chaos..* Advances in neural information processing systems. 2016.

[5] Schoenholz, Samuel S and Gilmer, Justin and Ganguli, Surya and Sohl-Dickstein, Jascha, *Deep information propagation.* arXiv preprint arXiv:1611.01232, 2016.

[6] Yang, Ge and Schoenholz, Samuel, *Mean Field Residual Networks: On the Edge of Chaos.* Advances in neural information processing systems, 2017.

[7] Pennington, Jeffrey and Schoenholz, Samuel and Ganguli, Surya, *Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice.* Advances in Neural Information Processing Systems 30, 2017.

[8] B Hanin, D Rolnick, *How to Start Training: The Effect of Initialization and Architecture.* arXiv preprint arXiv:1803.01719, 2018.

[9] Rastegari, Mohammad and Ordonez, Vicente and Redmon, Joseph and Farhadi, Ali, *Xnor-net: Imagenet classification using binary convolutional neural networks.* European Conference on Computer Vision, Springer, 2016.

[10] Hongyang Gao, Zhengyang Wang, Shuiwang Ji, *ChannelNets: Compact and Efficient Convolutional Neural Networks via Channel-Wise Convolutions.* arXiv preprint arXiv:1809.01330, 2018.

[11] Kim, Hyein and Yoon, Jungho and Jeong, Byeongseon and Lee, Sukho, *Rank-1 Convolutional Neural Network.* arXiv preprint arXiv:1808.04303, 2018.

[12] Howard, Andrew G and Zhu, Menglong and Chen, Bo and Kalenichenko, Dmitry and Wang, Weijun and Weyand, Tobias and Andreetto, Marco and Adam, Hartwig, *Mobilenets: Efficient convolutional neural networks for mobile vision applications.* arXiv preprint arXiv:1704.04861, 2017.

[13] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y., *Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1.* arXiv preprint arXiv:1602.02830, 2016.

[14] Anonymous, *A Mean Field Theory of Batch Normalization*, Submitted to International Conference on Learning Representations, https://openreview.net/forum?id=SyMDXnCcF7, 2018.