Contents

1 Initialization and Normalization in DNN and CNN 3
1.1 Data normalization in DNNs and CNNs 3
1.1.1 Normalization for input data of DNNs 3
1.1.2 Data normalization for images in CNNs 4
1.1.3 Comparison of $\sqrt{[\sigma_X]_j}$ and $\sqrt{[\tilde{\sigma}_X]_j}$ on CIFAR10. 5
1.2 Initialization for deep neural networks 5
1.2.1 Xavier's Initialization 5
1.2.2 Kaiming's initialization 7
1.2.3 Initialization in CNN models and experiments 8
1.3 Batch Normalization in DNN and CNN 10
1.3.1 Recall the original DNN model 10
1.3.2 "Real" Batch Normalization and "new" model 10
1.3.3 BN: some "modified" SGD on new batch normalized model 12
1.3.4 Testing phase in Batch-Normalized DNN 13
1.3.5 Batch Normalization for CNN 13

References 15

1

Initialization and Normalization in DNN and CNN

1.1 Data normalization in DNNs and CNNs

1.1.1 Normalization for input data of DNNs

Consider that we have the all training data as

$$(X, Y) := \{(x_i, y_i)\}_{i=1}^N, \tag{1.1}$$

for $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}^k$.

Before we input every data into a DNN model, we will apply the following normalization for all data x_i for each component. Let denote

$$[x_i]_j \longleftrightarrow \text{ the j-th component of data } x_i. \tag{1.2}$$

Then we have following formula of for all $j = 1, 2, \cdots, d$

$$[\tilde{x}_i]_j = \frac{[x_i]_j - [\mu_X]_j}{\sqrt{[\sigma_X]_j}}, \tag{1.3}$$

where

$$[\mu_X]_j = \mathbb{E}_{x\sim X}[[x]_j] = \frac{1}{N}\sum_{i=1}^N [x_i]_j, \quad [\sigma_X]_j = \mathbb{V}_{x\sim X}[[x]_j] = \frac{1}{N}\sum_{i=1}^N ([x_i]_j - [\mu_X]_j)^2. \tag{1.4}$$

Here $x \sim X$ means that x is a discrete random variable on X with probability

$$\mathbb{P}(x = x_i) = \frac{1}{N}, \tag{1.5}$$

for any $x_i \in X$.

For simplicity, we rewrite the element-wise definition above as the following compact form

$$\tilde{x}_i = \frac{x_i - \mu_X}{\sqrt{\sigma_X}}, \tag{1.6}$$

where

$$x_i, \tilde{x}_i, \mu_X, \sigma_X \in \mathbb{R}^d, \tag{1.7}$$

defined as before and all operations in (1.6) are element-wise.

Here we note that, by normalizing the data set, we have the next properties for new data $\tilde{x} \in \tilde{X}$ with component $j = 1, 2, \cdots, d$,

$$\mathbb{E}_{\tilde{X}}[[\tilde{x}]_j] = \frac{1}{N}\sum_{i=1}^{N}[\tilde{x}_i]_j = 0, \tag{1.8}$$

and

$$\mathbb{V}_{\tilde{X}}[[\tilde{x}]_j] = \frac{1}{N}\sum_{i=1}^{N}([\tilde{x}_i]_j - \mathbb{E}_{\tilde{X}}[[\tilde{x}]_j])^2 = 1. \tag{1.9}$$

Finally, we will have a "new" data set

$$\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_N\}, \tag{1.10}$$

with unchanged label set Y. For the next sections, without special notices, we use X data set as the normalized one as default.

1.1.2 Data normalization for images in CNNs

For images, consider we have a color image data set $(X, Y) := \{(x_i, y_i)\}_{i=1}^N$ where

$$x_i \in \mathbb{R}^{3\times m\times n}. \tag{1.11}$$

We further denote these the (s, t) pixel value for data x_i at channel j as:

$$[x_i]_{j;st} \longleftrightarrow (s, t) \text{ pixel value for } x_i \text{ at channel } j, \tag{1.12}$$

where $1 \leq i \leq N, 1 \leq j \leq 3, 1 \leq s \leq m$, and $1 \leq j \leq n$.

Then, the normalization for x_i is defined by

$$[\tilde{x}_i]_{j;st} = \frac{[x_i]_{j;st} - [\mu_X]_j}{\sqrt{[\sigma_X]_j}}, \tag{1.13}$$

where

$$[x_i]_{j;st}, [\tilde{x}_i]_{j;st}, [\mu_X]_j, [\sigma_X]_j \in \mathbb{R}. \tag{1.14}$$

Here

$$[\mu_X]_j = \frac{1}{m\times n\times N}\sum_{1\le i\le N}\sum_{1\le s\le m, 1\le t\le n}[x_i]_{j;st}. \tag{1.15}$$

and

$$[\sigma_X]_j = \frac{1}{N\times m\times n}\sum_{1\le i\le N}\sum_{1\le s\le m, 1\le t\le n}([x_i]_{j;st} - [\mu_X]_j)^2. \tag{1.16}$$

In batch normalization, we confirmed with Lian by both numerical test and code checking that BN also use the above formula to compute the variance in CNN for each channel.

Another way to compute the variance over each channel is to compute the standard deviation on each channel for every data, and then average them in the data direction.

$$\sqrt{[\tilde\sigma_X]_j} = \frac{1}{N}\sum_{1\le i\le N}\left(\frac{1}{m\times n}\sum_{1\le s\le m, 1\le t\le n}([x_i]_{j;st} - [\mu_i]_j)^2\right)^{\frac{1}{2}}, \tag{1.17}$$

where

$$[\mu_i]_j = \frac{1}{m\times n}\sum_{1\le s\le m, 1\le t\le n}[x_i]_{j;st}. \tag{1.18}$$

1.1.3 Comparison of $\sqrt{[\sigma_X]_j}$ and $\sqrt{[\tilde\sigma_X]_j}$ on CIFAR10.

They share the same μ_X as

$$\mu_X = \begin{pmatrix}0.49140105 & 0.48215663 & 0.44653168\end{pmatrix}. \tag{1.19}$$

But they had different standard deviation estimates:

$$\begin{aligned}\sqrt{[\sigma_X]_j} &= \begin{pmatrix}0.24703284 & 0.24348499 & 0.26158834\end{pmatrix}\\ \sqrt{[\tilde\sigma_X]_j} &= \begin{pmatrix}0.20220193 & 0.19931635 & 0.20086373\end{pmatrix}\end{aligned} \tag{1.20}$$

1.2 Initialization for deep neural networks

1.2.1 Xavier's Initialization

The goal of Xavier initialization [1] is to initialize the deep neural network to avoid gradient vanishing or blowup when the input is white noise.

Let us denote the DNN models as:

$$\begin{cases} f^1(x) &= W^1x + b^1\\ f^\ell(x) &= W^\ell\sigma(f^{\ell-1}(x)) + b^\ell \quad \ell = 2:L\\ f(x) &= f^L\end{cases}, \tag{1.21}$$

with $x \in \mathbb{R}^{n_0}$ and $f^\ell \in \mathbb{R}^{n_\ell}$. More precisely, we have

$$W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}. \tag{1.22}$$

The basic assumptions that we make are:

- The initial weights W^ℓ_{ij} are i.i.d symmetric random variables with mean 0, namely the probability density function of W^ℓ_{ij} is even.
- The initial bias $b^\ell = 0$.

Now we choose the variance of the initial weights to ensure that the features f^L and gradients don't blow up or vanish. To this end we have the following lemma.

Lemma 1. *Under the previous assumptions f^ℓ_i is a symmetric random variable with $\mathbb{E}[f^\ell] = 0$. Moreover, we have the following identity*

$$\mathbb{E}[(f^\ell_i)^2] = \sum_k \mathbb{E}[(W^\ell_{ik})^2]\mathbb{E}[\sigma(f^{\ell-1}_k)^2]. \tag{1.23}$$

Now, if $\sigma = id$, we can prove by induction from $\ell = 1$ that

$$\mathbb{V}[f^L_i] = \left(\Pi^L_{\ell=2} n_{\ell-1} \mathrm{Var}[W^\ell_{st}]\right)\left(\mathbb{V}[W^1_{st}] \sum_k \mathbb{E}[([x]_k)^2]\right). \tag{1.24}$$

We make this assumption that $\sigma = id$, which is pretty reasonably since most activation functions in use at the time (such as the hyperbolic tangent) were close to the identity near 0.

Now, if we set

$$\mathbb{V}[W^\ell_{ik}] = \frac{1}{n_{\ell-1}}, \quad \forall \ell \geq 2, \tag{1.25}$$

we will obtain

$$\mathbb{V}[f^L_i] = \mathbb{V}[f^{L-1}_j] = \cdots = \mathbb{V}[f^1_k] = \mathbb{V}[W^1_{st}] \sum_k \mathbb{E}[([x]_k)^2]. \tag{1.26}$$

Thus, in pure DNN models, it is enough to just control $\sum_k \mathbb{E}[([x]_k)^2]$.

A similar analysis of the propagation of the gradient ($\frac{\partial L(\theta)}{\partial f^\ell}$) suggests that we set

$$\mathbb{V}[W^\ell_{ik}] = \frac{1}{n_\ell}. \tag{1.27}$$

Thus, the **Xavier's initialization** suggests to initialize W^ℓ_{ik} with variance as:

- To control $\mathbb{V}[f^\ell_i]$:

$$\mathrm{Var}[W^\ell_{ik}] = \frac{1}{n_{\ell-1}}. \tag{1.28}$$

- To control $\mathbb{V}[\frac{\partial L(\theta)}{\partial f_i^\ell}]$:

$$\mathrm{Var}[W_{ik}^\ell] = \frac{1}{n_\ell}. \tag{1.29}$$

- Trade-off to control $\mathbb{V}[\frac{\partial L(\theta)}{\partial W_{ik}^\ell}]$:

$$\mathrm{Var}[W_{ik}^\ell] = \frac{2}{n_{\ell-1} + n_\ell}. \tag{1.30}$$

Here we note that, this analysis works for all symmetric type distribution around zero, but we often just choose uniform distribution $\mathcal{U}(-a, a)$ and normal distribution $\mathcal{N}(0, s^2)$. Thus, the final version of Xavier's initialization takes the trade-off type as

$$W_{ik}^\ell \sim \mathcal{U}(-\sqrt{\frac{6}{n_\ell + n_{\ell-1}}}, \sqrt{\frac{6}{n_\ell + n_{\ell-1}}}), \tag{1.31}$$

or

$$W_{ik}^\ell \sim \mathcal{N}(0, \frac{2}{n_\ell + n_{\ell-1}}). \tag{1.32}$$

1.2.2 Kaiming's initialization

In [2], Kaiming He and others extended this analysis to get an *exact* result when the activation function is the **ReLU**.

We first have the following lemma for symmetric distribution.

Lemma 2. *If $X_i \in \mathbb{R}$ for $i = 1 : n$ are i.i.d with symmetric probability density function $p(x)$, i.e. $p(x)$ is even. Then for any nonzero random vector $Y = (Y_1, Y_2, \cdots, Y_n) \in \mathbb{R}^n$ which is independent with X_i, the following random variable*

$$Z = \sum_{i=1}^n X_i Y_i, \tag{1.33}$$

is also symmetric.

Then state the following result for ReLU function and random variable with symmetric distribution around 0.

Lemma 3. *If X is a random variable on $\mathbb{R}$ with symmetric probability density $p(x)$ around zero, i.e.,*

$$p(x) = p(-x). \tag{1.34}$$

Then we have $\mathbb{E}X = 0$ and

$$\mathbb{E}[[\mathrm{ReLU}(X)]^2] = \frac{1}{2}\mathrm{Var}[X]. \tag{1.35}$$

Based on the previous Lemma 1, we know that $f_k^{\ell-1}$ is a symmetric distribution around 0. The most important observation in Kaiming's paper [2] is that:

$$\mathbb{V}[f_i^\ell] = n_{\ell-1}\mathbb{V}[W_{ij}^\ell]\mathbb{E}[[\sigma(f_j^{\ell-1})]^2] = n_{\ell-1}\mathbb{V}[W_{ik}^\ell]\frac{1}{2}\mathbb{V}[f_k^{\ell-1}], \tag{1.36}$$

if $\sigma = \text{ReLU}$. Thus, Kaiming's initialization suggests to take:

$$\mathbb{V}[W_{ik}^\ell] = \frac{2}{n_{\ell-1}}, \quad \forall \ell \geq 2. \tag{1.37}$$

For the first layer $\ell = 1$, by definition

$$f^1 = W^1 x + b^1, \tag{1.38}$$

there is no ReLU, thus it should be $\mathbb{V}[W_{ik}^1] = \frac{1}{d}$. For simplicity, they still use $\mathbb{V}[W_{ik}^1] = \frac{2}{d}$ in the paper [2]. Similarly, an analysis of the propagation of the gradient suggests that we set $\mathbb{V}[W_{ik}^\ell] = \frac{2}{n_\ell}$. However, in paper [2] authors did not suggest to take the trade-off version, they just chose

$$\mathbb{V}[W_{ik}^\ell] = \frac{2}{n_{\ell-1}}, \tag{1.39}$$

as default.

Thus, the final version of Kaiming's initialization takes the forward type as

$$W_{ik}^\ell \sim \mathcal{U}(-\sqrt{\frac{6}{n_{\ell-1}}}, \sqrt{\frac{6}{n_{\ell-1}}}), \tag{1.40}$$

or

$$W_{ik}^\ell \sim \mathcal{N}(0, \frac{2}{n_{\ell-1}}). \tag{1.41}$$

1.2.3 Initialization in CNN models and experiments

For CNN models, following the analysis above we have the next iterative scheme in CNNs

$$f^{\ell,i} = K^{\ell,i} * \sigma(f^{\ell,i-1}), \tag{1.42}$$

where $f^{\ell,i-1} \in \mathbb{R}^{c_\ell \times n_\ell \times m_\ell}$, $f^{\ell,i} \in \mathbb{R}^{h_\ell \times n_\ell \times m_\ell}$ and $K \in \mathbb{R}^{(2k+1)\times(2k+1)\times h_\ell \times c_\ell}$. Thus we have

$$[f^{\ell,i}]_{h;p,q} = \sum_{c=1}^{c_\ell}\sum_{s,t=-k}^{k} K_{h,c;s,t}^{\ell,i} * \sigma([f^{\ell,i-1}]_{c;p+s,q+t}). \tag{1.43}$$

Take variance on both sides, we will get

$$\mathbb{V}[[f^{\ell,i}]_{h;p,q}] = c_\ell(2k+1)^2\mathbb{V}[K_{h,o;s,t}^{\ell,i}]\mathbb{E}[([f^{\ell,i-1}]_{o;p+s,q+t})^2], \tag{1.44}$$

thus we have the following initialization strategies:

Xavier's initialization

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{2}{(c_\ell + h_\ell)(2k+1)^2}. \tag{1.45}$$

Kaiming's initialization

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{2}{c_\ell(2k+1)^2}. \tag{1.46}$$

Here we can take this Kaiming's initialization as:

- Double the Xavier's choice, and get

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{4}{(c_\ell + h_\ell)(2k+1)^2}. \tag{1.47}$$

- Then pick c_ℓ or h_ℓ for final result

$$\mathbb{V}[K^{\ell,i}_{h,o;s,t}] = \frac{4}{(c_\ell + h_\ell)(2k+1)^2} = \frac{2}{c_\ell(2k+1)^2}. \tag{1.48}$$

And they have the both uniform and normal distribution type.

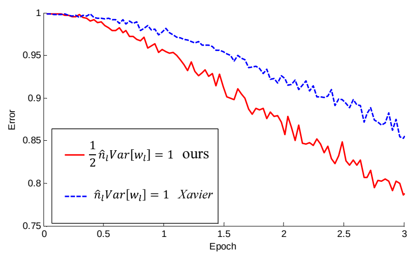

Fig. 1.1. The convergence of a **22-layer** large model. The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. Use ReLU as the activation for both cases. Both Kaiming's initialization (red) and "*Xavier's*" (blue) [1] lead to convergence, but Kaiming's initialization starts reducing error earlier.

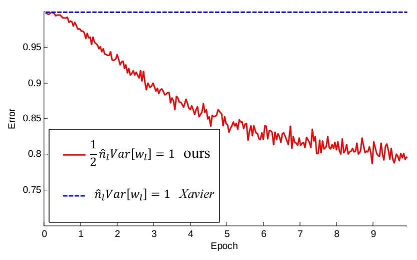

Fig. 1.2. The convergence of a **30-layer** small model (see the main text). Use ReLU as the activation for both cases. Kaiming's initialization (red) is able to make it converge. But "*Xavier's*" (blue) [1] completely stalls - It is also verified that that its gradients are all diminishing. It does not converge even given more epochs.

Given a 22-layer model, in cifar10 the convergence with Kaiming's initialization is faster than Xavier's, but both of them are able to converge and the validation accuracies with two different initialization are about the same(error is 33.82,33.90).

With extremely deep model with up to 30 layers, Kaiming's initialization is able to make the model convergence. On the contrary, Xavier's method completely stalls the learning.

1.3 Batch Normalization in DNN and CNN

1.3.1 Recall the original DNN model

Consider the classical (fully connected) artificial deep neural network (DNN) f^L,

$$\begin{cases} f^1 &= \theta^1(x) := W^1 x + b^1, \\ f^\ell &= \theta^\ell \circ \sigma(f^{\ell-1}) := W^\ell \sigma(f^{\ell-1}) + b^\ell,\ \ell = 2, ..., L. \end{cases} \tag{1.49}$$

where $x \in \mathbb{R}^n$ is the input vector, σ is a non-linear function (activation).

1.3.2 "Real" Batch Normalization and "new" model

Definition of BN operation based on the batch

Following the idea in normalization, we consider that we have the all training data as

$$(X, Y) := \{x_i, y_i\}_{i=1}^N. \tag{1.50}$$

Since the normalization is applied to each activation independently, let us focus on a particular activation $[f^\ell]_k$ and omit k as f^ℓ for clarity. We have N values of this activation in the batch,

$$X = \{x_1, \cdots, x_N\}.$$

Let the normalized values be $\hat{f}^\ell$, and their linear transformations be $\tilde{f}^\ell$.

(1.51)

$$\begin{aligned} &\mu_X^\ell \leftarrow \mathbb{E}_{x\sim X}[f^\ell(x)] = \frac{1}{N}\sum_{i=1}^N f^\ell(x_i) && \text{batch mean} \\ &\sigma_X^\ell \leftarrow \mathbb{E}_{x\sim X}\left[(f^\ell(x) - \mathbb{E}_{x\sim X}[f^\ell(x)])^2\right] = \frac{1}{N}\sum_{i=1}^N (f^\ell(x_i) - \mu_X)^2 && \text{batch variance} \\ &\hat{f}^\ell(x) \leftarrow \frac{f^\ell(x) - \mu_X^\ell}{\sqrt{\sigma_X^\ell + \epsilon}} && \text{normalize} \\ &\tilde{f}^\ell(x) \leftarrow \gamma^\ell \hat{f}^\ell(x) + \beta^\ell && \text{scale and shift} \end{aligned}$$

Here we note that all these operations in the previous equation are defined by element-wise. Then at last, we define the BN operation based on the batch set as

$$\mathrm{BN}_X(f^\ell(x)) = \tilde{f}^\ell(x) := \gamma^\ell \frac{f^\ell(x) - \mu_X^\ell}{\sqrt{\sigma_X^\ell + \epsilon}} + \beta^\ell, \tag{1.52}$$

where $\tilde{f}^\ell(x)$, μ_X^ℓ and σ_X^ℓ are given above.

"New" model for BN

In summary, we have the new DNN model with BN as:

$$\begin{cases} \tilde{f}^1(x_i) & = (\theta^1(x_i)), \\ \tilde{f}^\ell & = \theta^\ell \circ \sigma \circ \mathrm{BN}_X(\tilde{f}^{\ell-1}), \quad \ell = 2, ..., L. \end{cases} \tag{1.53}$$

For a more comprehensive notation, we can use the next notation

$$\sigma_{\mathrm{BN}} := \sigma \circ \mathrm{BN}_X. \tag{1.54}$$

Here one thing is important that we need to mention is that because of the new scale γ^ℓ and shift β^ℓ added after the BN operation. We can remove the basis b^ℓ in θ^ℓ, thus to say the real model we will compute should be

$$\begin{cases} \tilde{f}^1(x_i) & = W^1 x_i, \\ \tilde{f}^\ell & = W^\ell \sigma_{\mathrm{BN}}(\tilde{f}^{\ell-1}), \quad \ell = 2, ..., L. \end{cases} \tag{1.55}$$

Combine the two definition, we note

$$\tilde{\Theta} := \{W, \gamma, \beta\}, \tag{1.56}$$

where $W = \{W^1, \cdots, W^l\}$, $\gamma := \{\gamma^2, \cdots, \gamma^L\}$ and $\beta := \{\beta^2, \cdots, \beta^L\}$.

Finally, we have the loss function as:

$$\mathcal{L}(\tilde{\Theta}) = \mathbb{E}_{(x,y)\sim(X,Y)} \approx \frac{1}{N}\sum_{i=1}^{N} \ell(\tilde{f}^L(x_i; \tilde{\Theta}), y_i). \tag{1.57}$$

A key observation in (1.57) and the new BN model (1.55) is that

$$\begin{aligned} \mu_X^\ell &= \mathbb{E}_{x\sim X}[f^\ell(x)], \\ \sigma_X^\ell &= \mathbb{E}_{x\sim X}\left[(f^\ell(x) - \mathbb{E}_{x\sim X}[f^\ell(x)])^2\right], \\ \mathcal{L}(\tilde{\Theta}) &= \mathbb{E}_{(x,y)\sim(X,Y)}\left[\ell(\tilde{f}^L(x_i; \tilde{\Theta}), y_i)\right]. \end{aligned} \tag{1.58}$$

Here we need to mention that

$$x \sim X$$

means x subject to the discrete distribution of all data X.

1.3.3 BN: some "modified" SGD on new batch normalized model

Following the key observation in (1.58), and recall the similar case in SGD, we do the the sampling trick in (1.57) and obtain the mini-batch SGD:

$$x \sim X \approx x \sim \mathcal{B}, \tag{1.59}$$

here $\mathcal{B}$ is a mini-batch of batch X with $\mathcal{B} \subset X$.

However, for problem in (1.57), it is very difficult to find some subtle sampling method because of the composition of μ_X^ℓ and $[\sigma_X^\ell]^2$. However, one simple way for sampling (1.57) can be chosen as taking (1.59) for all the expectation case in (1.57) and (1.58).

This is to say, in training process (t-th step for example), once we choose $B_t \subset X$ as the mini-batch, then the model becomes

$$\begin{cases} \tilde{f}^1(x_i) & = W^1 x_i, \\ \tilde{f}^\ell & = W^\ell \sigma_{\rm BN}(\tilde{f}^{\ell-1}), \quad \ell = 2, ..., L. \end{cases} \tag{1.60}$$

where

$$\sigma_{\rm BN} := \sigma \circ {\rm BN}_{\mathcal{B}_t}, \tag{1.61}$$

or we can say that X is replaced by $\mathcal{B}_t$ in this case.

Here ${\rm BN}_{\mathcal{B}_t}$ is defined by

$$\begin{aligned} \mu_{\mathcal{B}_t}^\ell &\leftarrow \frac{1}{m}\sum_{i=1}^m f^\ell(x_i) && \text{mini-batch mean} \\ \sigma_{\mathcal{B}_t}^\ell &\leftarrow \frac{1}{m}\sum_{i=1}^m (f^\ell(x_i) - \mu_{\mathcal{B}_t})^2 && \text{mini-batch variance} \\ \hat{f}^\ell(x) &\leftarrow \frac{f^\ell(x) - \mu_{\mathcal{B}_t}^\ell}{\sqrt{\sigma_{\mathcal{B}_t}^\ell + \epsilon}} && \text{normalize} \\ {\rm BN}_{\mathcal{B}_t}(\tilde{f}^\ell) := \tilde{f}^\ell(x) &\leftarrow \gamma^\ell \hat{f}^\ell(x) + \beta^\ell && \text{scale and shift} \end{aligned} \tag{1.62}$$

Here BN operation introduce some new parameters as γ and β. Thus to say, for training phase, if we choose mini-batch as $\mathcal{B}_t$ in t-th training step, we need to take gradient as

$$\frac{1}{m}\nabla_{\tilde{\Theta}} \sum_{i \in \mathcal{B}_t} \ell(\tilde{f}^L(x_i; \tilde{\Theta}), y_i), \tag{1.63}$$

which needs us the to take gradient for μ_B^ℓ or $[\sigma_B^\ell]^2$ w.r.t w^i for $i \leq \ell$.

Questions: To derive the new gradient formula for BN step because of the fact that

$$\mu_{\mathcal{B}_t}^\ell, \quad \text{and} \quad \sigma_{\mathcal{B}_t}^\ell,$$

contain the output of $\tilde{f}^{\ell-1}$.

This is exact the batch normalization method described in [3].

1.3.4 Testing phase in Batch-Normalized DNN

One key problem is that, in the BN operator, we need to compute the mean and variance in a data set (batch or mini-batch). However, in the inference step, we just input one data into this DNN, how to compute the BN operator in this situation.

Actually, the γ and β parameter is fixed after training, the only problem is to compute the mean μ and variance σ^2. All the mean $\mu_{\mathcal{B}_t}$ and variance $\sigma^2_{\mathcal{B}_t}$ during the training phase are just the approximation of the mean and variance of whole batch i.e. μ_X and σ_X^2 as shown in (1.58).

One natural idea might be just use the BN operator w.r.t to the whole training data set, thus to say just compute μ_X and σ_X^2 by definition in (1.51).

However, there are at least the next few problems:

- computation cost,
- ignoring the statistical approximation (don't make use of the $\mu_{\mathcal{B}_t}$ and $\sigma^2_{\mathcal{B}_t}$ in training phase).

Considering that we have the statistical approximation for μ_X and σ_X^2 during each SGD step, moving average might be a more straightforward way. Thus two say, we define the μ^ℓ and $[\sigma^\ell]^2$ for the inference (test) phase as

$$\mu^\ell = \frac{1}{T}\sum_{t=1}^{T}\mu^\ell_{\mathcal{B}_t}, \quad \sigma^\ell = \frac{1}{T}\frac{m}{m-1}\sum_{t=1}^{T}\sigma^\ell_{\mathcal{B}_t}. \tag{1.64}$$

Here we take Bessel's correction for unbiased variance. The above moving average step is found in the original paper of BN in [3].

Another way to do this is to call the similar idea in momentum. At each time step we update the running averages for mean and variance using an exponential decay based on the momentum parameter:

$$\begin{aligned} \mu^\ell_{\mathcal{B}_t} &= \alpha\mu^\ell_{\mathcal{B}_{t-1}} + (1-\alpha)\mu^\ell_{\mathcal{B}_t} \\ \sigma^\ell_{\mathcal{B}_t} &= \alpha\sigma^\ell_{\mathcal{B}_{t-1}} + (1-\alpha)\sigma^\ell_{\mathcal{B}_t}, \end{aligned} \tag{1.65}$$

α is close to 1, we can take it as 0.9 generally. Then we all take bath mean and variance as $\mu^\ell_X \approx \mu^\ell_{\mathcal{B}_T}$ and $\sigma^\ell_X \approx \sigma^\ell_{\mathcal{B}_T}$.

Many people argue that the variance here should also use Bessel's correction.

1.3.5 Batch Normalization for CNN

One key idea in BN is to do normalization with each scalar features (neurons) separately along a mini-batch. Thus to say, we need one to identify what is neuron in CNN. This is a historical problem, some people think neuron in CNN should be the pixel in each channel some thing that each channel is just one neuron. BN choose the later one. **One (most ?) important reason for this choice is the fact of computation cost.**

For convolutional layers, BN additionally wants the normalization to obey the convolutional property – so that different elements of the same feature map, at different locations, are normalized in the same way. To compute $\mu^{\ell}_{\mathcal{B}_t}$, we take mean of the set of all values in a feature map across both the elements of a mini-batch and spatial locations – so for a mini-batch of size m and feature maps of size $m_\ell \times n_\ell$ (image geometrical size), we use the effective mini-batch of size $mm_\ell n_\ell$. We learn a pair of parameters γ_k and β_k per feature map (k-th channel), rather than per activation.

For simplicity, then have the following BN scheme for CNN

(1.66)

$$
\begin{aligned}
[\mu^{\ell}_{\mathcal{B}_t}]_j &\leftarrow \frac{1}{m \times m_\ell \times n_\ell} \sum_{i=1}^{m} \sum_{1\le s\le m_\ell, 1\le t\le n_\ell} [f^\ell(x_i)]_{j;st} && \text{mean on channel } j\\
[\sigma^{\ell}_{\mathcal{B}_t}]_j &\leftarrow \frac{1}{m \times m_\ell \times n_\ell} \sum_{i=1}^{m} \sum_{1\le s\le m_\ell, 1\le t\le n_\ell} ([f^\ell(x_i)]_{j;st} - [\mu^{\ell}_{\mathcal{B}_t}]_j)^2 && \text{variance on channel } j\\
[\hat{f}^\ell(x)]_{j;st} &\leftarrow \frac{[f^\ell(x)]_{j,st} - [\mu^{\ell}_{\mathcal{B}_t}]_j}{\sqrt{[\sigma^{\ell}_{\mathcal{B}_t}]_j + \epsilon}} && \text{normalize}\\
[\mathrm{BN}_{\mathcal{B}_t}(\tilde{f}^\ell)]_{j;st} := [\tilde{f}^\ell(x)]_{j;st} &\leftarrow [\gamma^\ell]_j[\hat{f}^\ell(x)]_{j;st} + [\beta^\ell]_j && \text{scale and shift on channel}
\end{aligned}
$$

References

[1] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[4] Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller, et al. Neural networks: Tricks of the trade. *Springer Lecture Notes in Computer Sciences*, 1524(5-50):6, 1998.

[5] S. Wiesler and H. Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pages 657–665, 2011.