Report for Question 2

Xufeng Cai
Shanghai Jiao Tong University

2019-3-31

Abstract

In this report, I illustrate some of my thoughts for the question 2. However, I think it is a quite open question, so I have to admit that my solution is quite incomplete and limited.
Keywords: Batch Normalization, Convolutional Layer, Implicit Problem

1 Background Settings

Question: Denote Conv as a convolutional layer, and BN as a batch normalization layer. Usually, for a forward pass (for these two modules), it is defined as

$$z = BN(Conv(x)). \tag{1.1}$$

However, instead of doing so, here, we want to solve the following implicit formula, i.e. x is given but z is unknown,

$$x = BN(Conv(z)). \tag{1.2}$$

Particularly, for this problem, you do not need to train a model, but just solve the above two-module implicit problem. You may need to look at Generalized Minimal Residual Method.

My Understanding:
Here I will try to solve the above implicit problem (1.2), where x is given but z is unknown. Hence, my understanding of the question is to go through two modules inversely to get the input z. Moreover, I assume that **the parameters of two modules BN and Conv are given**, only in this way can I start to solve the problem.

Then we recall some ideas of convolutional layer and batch normalization.

- **Convolutional Layer:** As is known, we can regard the operation of convolution as a linear transformation, then we have

  $$Conv(z) = w * z + b, \tag{1.3}$$

  where w is the kernel matrix, and b is the bias. I have to mention that here z in (1.3) is a column vector or a matrix other than the original data after data-processing operation, but it doesn't matter when we derive the solution.

- **Batch Normalization:** We just consider the situation of mini-batch here. The algorithm is shown below:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Since we do scale and shift in the last step of batch normalization, we can omit the bias in the linear transformation of convolutional layer, then we obtain that

$$BN(Conv(z)) = BN(w * z + b) = BN(w * z). \tag{1.4}$$

Thus,

$$BN(Conv(z)) = \gamma\frac{y - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta I, \tag{1.5}$$

where

  - $y = w * z$
  - $\mu = \frac{1}{m} I^T y I$,
  - $\sigma^2 = \frac{1}{m}(y - \mu)^T(y - \mu)$,
  - I is the unit column vector,
  - γ and β are known scale and shift parameters in batch normalization.

Therefore, we can finally construct the equation

$$x = \gamma\frac{y - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta I. \tag{1.6}$$

2 Attempt

2.1 Motivation

In order to apply Generalized Minimal Residual Method, we want to approximate the equation (1.6) with the form

$$x = Az, \tag{2.1}$$

where A is a matrix. Since the Conv is a linear transformation, we only need to derive the linear approximation of Batch Normalization.

2.2 Method

Obviously, the expression of BN Module with respect to y (or z) is nonlinear. So I attempted to find a linear approximation for this expression. I tried Taylor Expansion, and I expanded the expression with respect to $y - \mu$ about the origin. But it turns out that the coefficient of the linear item is ϵ, which is definitely a very bad approximation.

Except for Taylor Expansion, I did not think up other proper linear approximation methods, but I think it is direction worthy of trying to solve this implicit problem. Of course, doing linear approximation surely omits much information and brings about much error. Maybe there are some fancy approximation methods which can well solve this problem. But due to my limited knowledge, I can not illustrate a such proper method here.

Therefore, I change my direction and make some assumptions to deduct this problem.

3 Theoretical Method

3.1 Motivation

Similar to the thoughts in the Attempt Section, I want the Batch Normalization module to be linear, then I can apply Generalized Minimal Residual Method to solve this problem.

This time I first fix σ to make BN module linear, then I derive the condition solution should satisfy to be the distribution with fixed σ.

3.2 Theoretical Solution

Firstly, we fix σ, and let $b = x - \beta I$ and $c = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$. Obviously, b and c are known, and the equation (1.6) becomes

$$b = c(y - \mu). \tag{3.1}$$

Since $\mu = \frac{1}{m} I^T y I$, then we have

$$y - \mu = My, \tag{3.2}$$

where M is a matrix and

$$M_{ij} = \begin{cases} 1 - \frac{1}{m}, & i = j; \\ -\frac{1}{m}, & \text{otherwise.} \end{cases} \tag{3.3}$$

Moreover, as $y = w * z$, we let $A = cMw$, then we obtain

$$b = Az. \tag{3.4}$$

It is a nonsymmetric system of linear equations, and b and A are all known. Here we apply **Generalized Minimal Residual Method** to solve (3.4) to obtain a numerical solution iteratively.

We assume that the numerical solution obtained is $\hat{z}$, and denote $\hat{\sigma}^2 = Var(\hat{z})$. As we fix σ at first, the numerical solution we obtained should satisfy some restriction. That is the

distribution of data processed by Conv Module should have std σ. Thus, we have to check that whether

$$\sigma^2 = w\hat{\sigma}^2 w^T. \tag{3.5}$$

Combining the (3.4) and (3.5), we obtain that for $\forall\sigma$, follow the process below to solve the problem:

- Solve (3.4) to get numerical solution $\hat{z}$ by GMRES,
- Check whether $\hat{z}$ satisfy the condition (3.5),
- If satisfied, then $\hat{z}$ is one solution.

In this way, we can solve this implicit problem.

4 Implementation in Pytorch

4.1 New Understanding of the Problem

Besides the above theoretical analysis, this implicit problem may be much easier with the implementation with Pytorch. In fact, the Batch Normalization Module in Pytorch offers operations called **BN.running_mean** and **BN.running_var**, which give out a statistical estimates of activations we want to normalize. Such estimates are computed during training (usual forward process in this problem) by exponential moving average.

Since in this implicit problem two modules applied to input z and the output x are known, I suppose the estimates of mean and var given by the above two operations are also known. Indeed, we can regard them as knowledges that two modules learn from the input data, which is also a part of two-module. Moreover, I suppose that the above two operations give out a proper estimates for mean and var, so I can use them to solve this implicit problem.

With such estimates for the mean and var, the Batch Normalization Module also simply becomes a linear transformation, and we can implement it in Pytorch with directly using GMRES to solve the linear system.

4.2 Detailed Description of Implementation

We use the similar notation with Section 1 in this subsection. Given a feature map F with size of $C * H * W$ (channel, height, width), we can obtain its normalized version $\hat{F}$ with the freezed batchnorm, by computing the following linear transformation for each spatial position i, j:

$$\begin{pmatrix} \hat{F}_{1,i,j} \\ \hat{F}_{2,i,j} \\ \vdots \\ \hat{F}_{C-1,i,j} \\ \hat{F}_{C,i,j} \end{pmatrix} = \begin{pmatrix} \frac{\gamma_1}{\sqrt{\hat{\sigma}_1^2+\epsilon}} & 0 & \cdots & & 0 \\ 0 & \frac{\gamma_2}{\sqrt{\hat{\sigma}_2^2+\epsilon}} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \frac{\gamma_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2+\epsilon}} & 0 \\ 0 & & \cdots & 0 & \frac{\gamma_C}{\sqrt{\hat{\sigma}_C^2+\epsilon}} \end{pmatrix} \cdot \begin{pmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{C-1,i,j} \\ F_{C,i,j} \end{pmatrix} + \begin{pmatrix} \beta_1 - \gamma_1 \frac{\hat{\mu}_1}{\sqrt{\hat{\sigma}_1^2+\epsilon}} \\ \beta_2 - \gamma_2 \frac{\hat{\mu}_2}{\sqrt{\hat{\sigma}_2^2+\epsilon}} \\ \vdots \\ \beta_{C-1} - \gamma_{C-1} \frac{\hat{\mu}_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2+\epsilon}} \\ \beta_C - \gamma_C \frac{\hat{\mu}_C}{\sqrt{\hat{\sigma}_C^2+\epsilon}} \end{pmatrix}$$

Let $W_{BN} \in R^{C\times C}$ and $b_{BN} \in R^C$ denote the matrix and bias in the above equation, and $W_{Conv} \in R^{C\times(C_{prev}\times k^2)}$ and $b_{Conv} \in R^C$ the parameters of the convolutional layer precedes the batchnorm layer, where C_{prev} is the number of channels of the feature map F_{prev} input to the convolutional layer and $k \times k$ is the filter size.

Given a $k \times k$ neighborhood of F_{prev} unwrapped into a $k^2 \times C_{prev}$ vector $f_{i,j}$, we illustrate the whole linear transformation process below:

$$\hat{f}_{i,j} = W_{BN} \cdot (W_{Conv} \cdot f_{i.j} + b_{Conv}) + b_{BN} \tag{4.1}$$

Therefore, in fact, we can replace the two modules by a single convolutional layer with the following parameters:

- **filter weights:** $W = W_{BN} \cdot W_{Conv}$
- **bias:** $b = W_{BN} \cdot b_{Conv} + b_{BN}$

Since the convolutional layer is a linear transformation, we then do the following expansion. We expand the kernel matrix and input data, with respect to rows, to column vectors, and the convolutional operator can be represented as matrix multiplication and a shift $C^T z + \hat{b}$, where C^T and $\hat{b}$ are dependent on W and b. In other words, we can regard C^T and $\hat{b}$ as expansions of W and b. Take 3×3 kernel matrix as example:

$$\begin{pmatrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & w_{0,0} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ 0 & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0} & 0 \\ w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\ w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\ 0 & w_{2,2} & 0 & w_{1,2} \\ 0 & 0 & w_{2,0} & 0 \\ 0 & 0 & w_{2,1} & w_{2,0} \\ 0 & 0 & w_{2,2} & w_{2,1} \\ 0 & 0 & 0 & w_{2,2} \end{pmatrix}^T$$

Moreover, when applying GMRES, we want the matrix to be square, so the output and input have the same dimension, and we need to do padding in the convolutional layer. The padding process is also a linear transformation, and denote that transformation matrix as P, we then only need to solve the following linear system by the GMRES method to solve the implicit problem:

$$C^T P z = x - \hat{b} \tag{4.2}$$

4.3 Error Analysis

In the experiment part, I apply my code to solve the implicit problem for the randomly generated input data of order $3 \times 8 \times 8$ for fast testing to get numerical error. Indeed, I also

implement the solver for CIFAR10, but the number of data and the size of image are kind of large, which costs much time for testing.

For two modules of Conv and BN, I just use their initial generated parameters, and I didn't train the model. Moreover, I use 1-norm, 2-norm and max-norm to define the numerical errors as below

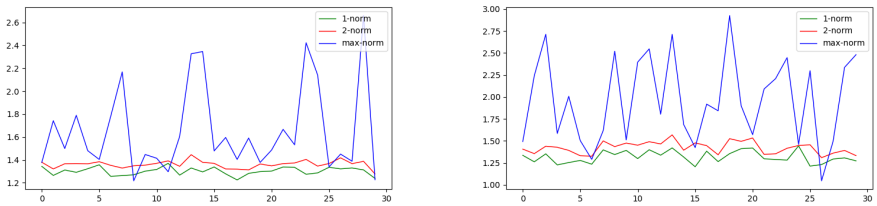
$$error_1 = norm(z_{real} - z_{cal}) \div norm(z_{real}) \tag{4.3}$$

$$error_2 = norm(BN(Conv(z_{cal})) - x) \tag{4.4}$$

$$error_3 = norm(BN(Conv(z_{cal})) - x) \div norm(x) \tag{4.5}$$

Since I have to test the method by computing x with generating z, I know the real value of z, then I calculate the error (4.3) here as another estimate, even though it does not make sense for implicit problems. In fact, for implicit problems, it is almost impossible to know the exact values of input data. Hence, we take error (4.4) and (4.5) as our main estimates in this part.

Also, since data are randomly generated, different try may have different results of error, but the trend and magnitude won't differ a lot. I run 30 batches of $8 \times 3 \times 8 \times 8$ and $16 \times 3 \times 8 \times 8$ input data, and plots for $error_2$ are in the zip and $error_3$ are below:



(a) 8 feature maps (b) 16 feature maps

Figure 1: $error_3$ for 8 and 16 feature maps

From figures we can see that there is a not small error when solving this problem, but it is acceptable. Since we are solving a quite large linear system, it is not much realistic to get each element in the solution accurate. Instead, we can see that $1 - norm$ and $2 - norm$ are not so large, which means that the whole solution can be a reasonable approximation. Also, the error increases to some extent with feature maps increasing, which is quite natural as we will solving a larger linear system and it will be harder to get an exact solution.

4.4 Improvements

Since our method to solve the implicit problem is by regarding the two modules as a linear transformation and solving the linear system, it is quite likely that the matrix $C^T P$ above is ill-conditioned. Hence, doing the preconditioning to decrease the conditional number is of necessity.

Here I choose the method of row equivalence to deal with the case that magnitudes of weights in different rows of the combined two modules differ with each other too much. Suppose $C^T P = [a_{ij}]$, calculate the $\infty-$norm of each row:

$$S_i = max_j \mid a_{ij} \mid \tag{4.6}$$

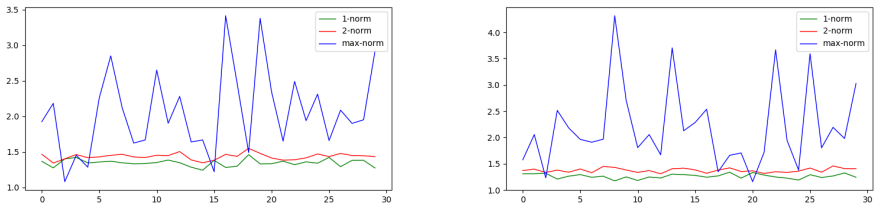
Let

$$D = diag(S_1^{-1}, S_2^{-1}, \cdots, S_n^{-1}, \cdots), \tag{4.7}$$

then the original linear system becomes

$$(DC^T P)z = D(x - \hat{b}). \tag{4.8}$$

Solving this new linear system is more likely to obtain the exact solution, and speed up convergence. Below are the results with preconditioning:



(a) 8 feature maps (b) 16 feature maps

Figure 2: $error_3$ for 8 and 16 feature maps with preconditioning

In fact, we can see that the range of errors is not much different from ones without preconditioning. However, maybe it is not so obvious here, as I just run small amount of batches, the error can be controlled more stably in an acceptable range, especially when comparing $1 - norm$ and $2 - norm$. Moreover, if a more proper preconditioning method is conducted, the error may be reduced.

5 Some Thoughts

5.1 Why we use GMRES?

- It is a kind of numerical solver quite easy to implement. Moreover, it has been implemented in Scipy modules in Python, which is convenient for us to just import and use it.

- In the process of our implementation, we see that in fact the matrix $C^T P$ is quite sparse, as C^T and P are both sparse matrices by our construction. Fortunately for GMRES, at each iteration, ordinarily the computation for a matrix-vector product A_{q_n} costs about $2m^2$ floating-point operations for general dense matrices of size m, but the cost can decrease to O(m) for sparse matrices.

- By the property of Krylov subspace, the GMRES method arrives at the exact solution. Moreover, for many situations, after a small number of iterations (relative to m), the vector $x_n \in K_n$ is already a good approximation for the exact solution, which means that the convergence of GMRES method is pretty good.

5.2 Discussion of my solution

- In practice, we usually set a threshold for the condition (3.5).
- **Preconditioning** is quite important for iterative numerical methods. It will increase the rate of convergence since it decreases the condition number of A above. In my experiment, there is no big difference whether conducting preconditioning. This may due to that I just use the initial generated parameters of two modules, which are kind of uniform, so the difference of magnitudes of weights in different rows in the final matrix is not big, and there is little need to to preconditioning. However, while training the parameters will be updated and may not likely to be uniform like beginning, at that time preconditioning may be important and necessary.
- Since the matrix $C^T P$ is sparse, the linear system may have no solution or have infinitely many solutions. Hence, such implicit problem is hard to solve to some extent. What I can deal with is just when the final matrix is well enough to have solutions. Moreover, since the GMRES method is an iterative method consisting of the least square method, I think it gives out a kind of proper approximate exact solution.
- At first, I try to obtain the inverse of Batch Normalization, but then I find that infinitely many different distributions can become the given output distribution. For example, for a distribution satisfies the condition, if we do the dilation, it still satisfies the condition. Hence, I think it is impossible to get the one exact inverse of BN module without giving any priori knowledge about the input data distribution, thus trying other methods to solve this implicit method.
- In my coding, I use many **for** combined together, which is very ugly and inefficient. I believe there must be other elegant and more efficient ways to implement it.

6 Apology & Acknowledge

Since I had some tests and courseworks to prepare and finish, I am sorry that I spent more time than I expected to implement question 2.

Last but not least, thank you for giving me a second chance to implement, and from this I have learned a lot.