Using Batch Normalization to Accelerate AlexNet[1]

Jinfei Zhu, MA in Computational Social Science

Why we need Batch Normalization?

Current neural networks divide the training dataset into mini-batches for gradient descent, and the model is only updated after all training examples have been evaluated. In each mini-batch, the gradient of loss is calculated as an estimate of the gradient over the whole training set. The larger the batch size is, the better the model performance. Additionally, the computation over mini batches can be more efficient, due to the parallelism offered by Graphic Processing Units.

Stochastic gradient descent is often used along with mini-batches, since it's simple and effective. However, this method often requires careful tuning of the model hyper-parameters, such as learning rate and initial value. What's more importantly, the inputs to each layer are affected by the parameters of all preceding layers. Therefore, small changes in the previous layer would be amplified in the deeper layers, resulting instability of the model and longer time to train.

[1] Codes can be found at: https://colab.research.google.com/drive/1sHsUcxpCxD_D9YaRqh-uo87H7sPVh_GA?usp=sharing

Besides this sensitivity of deep layer's parameters, two problems are also pointed out by the original papers. The first one is the saturation problem, for example, for sigmoid activation function $g(x) = \frac{1}{1+\exp(-x)}$, as $|x|$ increases to a certain level, $g'(x)$ would approximate zero, which resulting vanishing gradient problems. This saturation problem nowadays is usually solved by using Rectified Linear Units, careful initialization, and smaller learning rates. However, if we can stabilize the distribution of inputs throughout the training, then the optimizer would be less likely to encounter this saturated situation and training can go faster.

The second problem is the Internal Covariate Shift, which is the change in the distribution of network activations due to change in network parameters during training.

To reduce these two problems, Batch Normalization ensures that for any parameter values, the network could always produce activations with the desired distribution.

What does the original paper do?

The original paper and the PyTorch reimplementation codes used LeNet model, which is one of the earliest convolutional neural network models, to test the performance of adding batch normalization layer, with MNIST dataset. The codes are from the book Dive Into Deep Learning, and it used some customized functions from the book instead of PyTorch `nn.BatchNorm2d()` API and train the model.

Since LeNet is designed for black and white pictures, it is no longer suitable in this case. I know ResNet is preferable in this case, but we have been working with ResNet in our Problem Set 2, so I decide to use AlexNet, which is another CNN models designed for

colorful pictures and faster to run than ResNet, to test if I could use Batch Normalization to improve the model's performance.

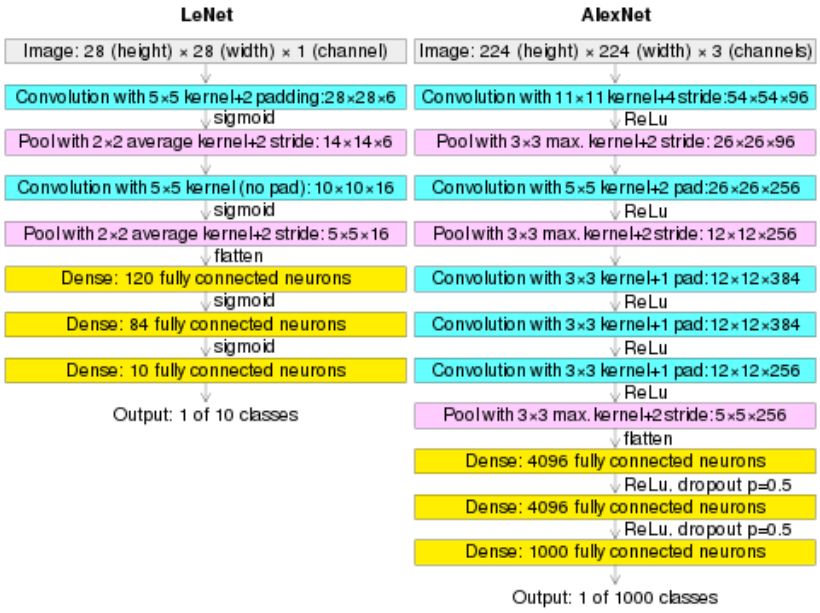


Figure 1 Comparison of the LeNet and AlexNet convolution, pooling and dense layers (source)

What can I extend?

First of all, I need to change the model to AlexNet and the dataset to CIFAR-10, and they've been easily accessible via PyTorch. Then I tried different learning rates of 1, 0.1 and 0.01 with the original models, find that the 0.1 learning rate is performing the best. Without Dropout or Batch Normalization, the pure AlexNet performs not very well. After 10 epochs of training, the model still doesn't converge, so I train more epochs. The result shows that only after 20 epochs does the model converge.

I am going to following modification to the model:

1. First implementing from the scratch for batch normalization and add it to AlexNet, compare the final test error rate and then add batch normalization to AlexNet to

see if the model performance increases.

2. Compare different optimizers' performance, specifically SGD and Adam
3. Test if a better initialization like Xavier Initialization is needed
4. The original implementation of AlexNet uses Dropout but no Batch Normalization. As the paper said, Batch Normalization regularizes the model. Whereas DropOut has been used to reduce overfitting, the paper found in a batch-normalized network it can be either removed or reduced in strength, but they didn't test the difference. I am going to test if the model's performance changes a lot.
5. L2 Regularization is another popular method to regularize the model by adding a weight decay rate in the optimizer. I am going to test if L2 Regularization will further improve the model's performance and accelerate the convergence.

What are the results of my experiments?

1 Adding Batch Norm to AlexNet

Without Batch Normalization or DropOut, the AlexNet converges slowly, and needs more than 20 epochs to train. However, after adding Batch Normalization, the model can usually converge in 3 epochs. The training speed is accelerated.

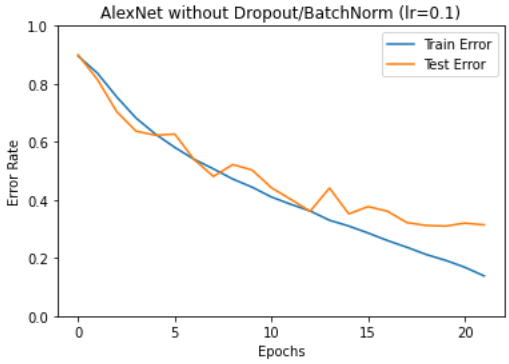


Figure 2 Performance of AlexNet without BatchNorm/DropOut

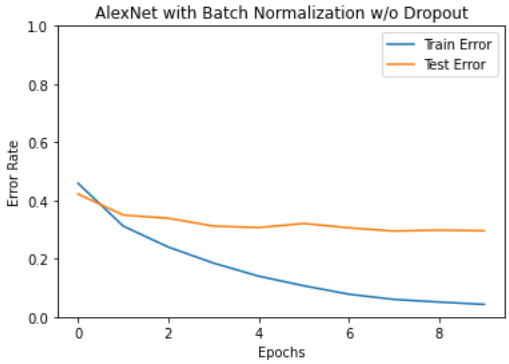


Figure 3 Adding BatchNorm

From the result, we can see that the performance is much improved and the training is accelerated.

2 Comparing Different optimizer

Instead of just Stochastic Gradient Descent, Adam has been used as a replacement optimization algorithm for SGD for training deep learning models, which combines the best properties of AdaGrad and RMSProp algorithms to provide an optimization algorithm. I did an experiment to see if Adam can provide a further improvement for the model.

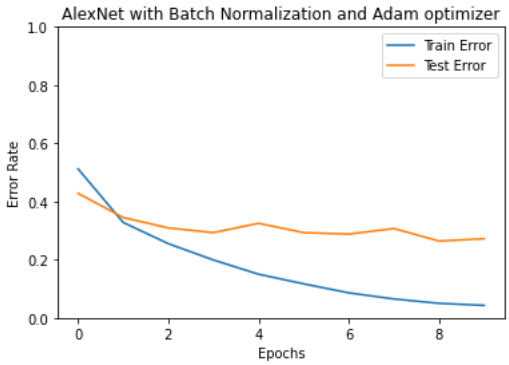


Figure 4 Try Adam Optimizer

However, since the AlexNet model is not so deep, there is no big difference between SGD and Adam.

3 Better Initialization

By using a better Initialization, which sample from a Gaussian with a deviation related to number of input channels when initializing the weight, the model could be more trainable and get better result.

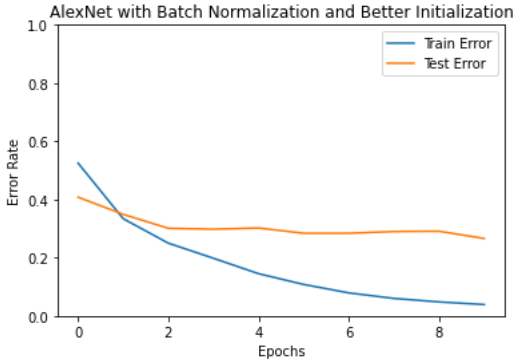


Figure 5 Batch Norm and Better Initialization

However, probably because the model is not very deep, this better initialization doesn't improve the model much.

4 Adding DropOut to interact with BatchNorm

DropOut is also a common method to avoid overfitting and increase model performance. However, its interaction with Batch Normalization is very bad. Adding it to the model makes the model take longer to converge (as the following graph shows).

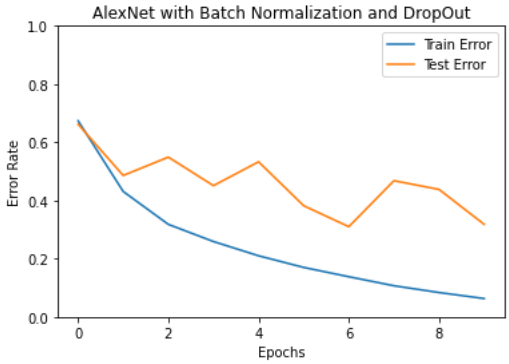


Figure 6 The Interaction of BatchNorm and DropOut is negative

If I only add DropOut to my model and don't use the Batch Normalization, the result is better than if I used both.

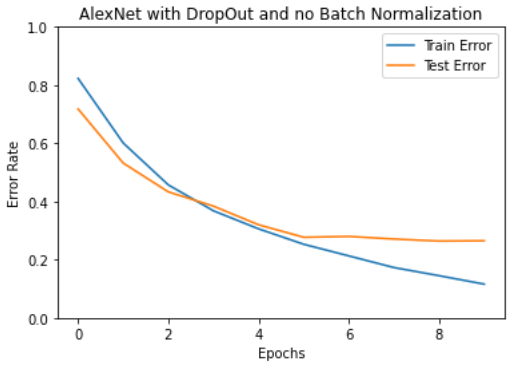
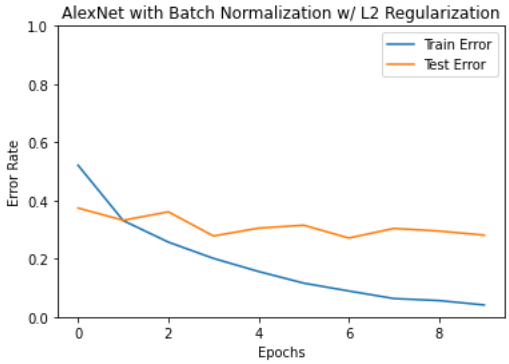


Figure 7 Only DropOut and No BatchNorm

Without Batch Normalization, it takes longer for the model to train to get the best Test Error (~6 epochs), and the model performance is worse for the first few epochs. However, it's performance is more smoother than the model with both DropOut and BatchNorm.

5 L2 Regularization

L2 Regularization is a method to add a L2 penalty to the loss function. In PyTorch, to realize this, we can simply add a weight decay argument in the optimizer to realize this.



Adding L2 Regularization to the model doesn't increase the model's performance. Probably Batch Normalization is enough so we don't need further regularization.

This summarizes all the experiments I have done to increase the accuracy and speed up the convergence of AlexNet.