Matrix Arithmetic and Batch Normalization

Ronald Yu

January 18, 2018

1 Matrix Arithmetic

Let us review some matrix arithmetic with the simple exercise shown in class.

Suppose we have a three-layer toy network with no biases or non-linearities that regresses a scalar y such that

$$y = W_3 W_2 W_1 x$$

where $W_3 \epsilon \mathbb{R}^{1x100}$, $W_2 \epsilon \mathbb{R}^{100x100}$, $W_1 \epsilon \mathbb{R}^{100x10}$, $x \epsilon \mathbb{R}^{10}$, $||x|| \approx 1$, and all weights are sampled i.i.d from $N(0,1)$.

Exercise 1

Problem:

$$\frac{\partial y}{\partial W_1} =? \qquad \frac{\partial y}{\partial W_2} =? \qquad \frac{\partial y}{\partial W_3} =?$$

Hints:

$$tr(ABC) = tr(BCA) = tr(CAB) \tag{1}$$

$$\nabla_A (AB) = B^T \tag{2}$$

Hint 1 applies for an arbitrary number of matrices (not just 3). Recall that the trace of a matrix is the sum of its diagonals.

Solution

First, let us treat y as a $1x1$ matrix and take the trace of both sides of the equation:

$$tr(y) = tr(W_3 W_2 W_1 x)$$

Since y is a $1x1$ matrix, the trace of y is also the value of y. Combining this information with Hint 1 we have:

$$y = tr(W_3 W_2 W_1 x) = tr(W_2 W_1 x W_3) = tr(W_1 x W_3 W_2)$$

We can then directly apply Hint 2 to obtain:

$$\frac{\partial y}{\partial W_1} = (xW_3W_2)^T$$
$$\frac{\partial y}{\partial W_2} = (W_1xW_3)^T$$
$$\frac{\partial y}{\partial W_3} = (W_2W_1x)^T$$

Exercise 2

Problem:

$$E[||W_1||_F] = ? \qquad E[||W_2||_F] = ? \qquad E[||W_2||_F] = ?$$

Recall that $||M||_F$ denotes the Frobenius Norm of a mxn matrix M and is defined by:

$$||M||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|M_{ij}|^2}$$

Hint:

$$Var(X+Y) = Var(X) + Var(Y)$$

Recall that the variance of a distribution X is defined by $Var(X) = E[(X - E(X))^2] = E[X^2] - E[X]^2$.

Solution: Since the weights are sampled from a normal distribution of mean 0 and variance 1, for the distribution X of the elements in each of the weight matrices, we have $1 = Var(X) = E[X^2] - E[X]^2 = E[X^2] - 0 = E[X^2]$.

Using the hint we can determine the expected value of the Frobenius norm of each weight matrix M of size mxn by:

$$\begin{aligned} E[||M||_F] &= E[\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}}] \\ &= \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} E[|M_{ij}|^2]} \\ &= \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} 1} \\ &= \sqrt{mn} \end{aligned}$$

Thus,

$$E[||W_1||_F] = \sqrt{1000} \qquad E[||W_2||_F] = 100 \qquad E[||W_2||_F] = 10$$

Exercise 3

Problem:

Let $y_1 = W_1 x$ and $y_2 = W_2 W_1 x$.

$$E[||y_1||_2] \leq ? \qquad E[||y_2||_2] \leq ? \qquad E[||y||_2] \leq ?$$

Hint:

$$||A||_2 \leq ||A||_F$$

Solution:

Using the Cauchy-Schwartz Inequality, we have:

$$\begin{aligned} E[||y_1||_2] &= E[||W_1 x||_2] \\ &\leq E[||W_1||_2 ||x||_2] \end{aligned}$$

From Example 2 and the hint we know that $E[||W_1||_2] \leq E[||W_1||_F] = \sqrt{1000}$, and we are given that $||x|| \approx 1$, so:

$$\begin{aligned} E[||y_1||_2] &\leq E[||W_1||_2 ||x||_2] \\ &= E[||W_1||_2|] \\ &\leq \sqrt{1000} \end{aligned}$$

Similarly, since for $y2$ we have:

$$\begin{aligned} E[||y_2||_2] &= E[||W_2 y_1||] \\ &\leq E[||W_2||_2 ||y1||_2] \\ &\leq \sqrt{1000} E[||W_2||_2] \\ &\leq 100\sqrt{1000} \end{aligned}$$

$$\begin{aligned} E[||y||_2] &= E[||W_3 y_2||] \\ &\leq E[||W_3||_2 ||y2||_2] \\ &\leq 100\sqrt{1000} E[||W_3||_2] \\ &\leq 1000\sqrt{1000} \end{aligned}$$

2 Batch Normalization

2.1 Motivation

One takeaway from these exercises is that as we increase the number of layers, the magnitude of the the upper-bounds of the outputs of the deeper layers and the gradients of each layer increase exponentially. We see that the effects of small changes in the shallower layers are amplified on deeper parts of the network. Hence, we must carefully tune our learning rate so that it is not too high, which would lead to higher magnitude weight values and could easily cause output values to explode and the network to diverge. However, this leads to an excessively slow learning rate, causing the network to take an extremely long time to converge.

However, what if in Exercise 3 we normalized the outputs y_1 and y_2 to have a distribution of $N(0, 1)$? If we did so for each layer then small changes in the shallower layers would not be amplified in the deeper parts of the network, and the upper bounds of the outputs deeper parts of the would not explode. If the output of each layer is normalized to mean 0 and variance 1 anyway, then the magnitude of the output of each layer is less dependent on the magnitude of the weights of each layer, so weights with large magnitudes would not easily lead to exploding outputs and gradients. This would allow us to safely *increase the learning rate* with lessened risk of divergence, leading to much faster convergence. This is one of the core motivations behind Batch Normalization.

Another problem that Batch Normalization addresses is *internal covariate shift*. What is internal covariate shift? First let us define covariate shift. Covariate shift is when the input distribution to a machine learning system changes. For a more severe example of covariate shift, if we train a network to classify animated face models and then give the network real face images as input, the network will have poor performance as the input distribution is different. We can imagine that if a network were constantly receiving new input distributions during training time, the network would continually have to adapt to the new input distributions and training would be slow.

If we think of each layer and all its following layers in a network as a subnetwork (i.e. Layer 0 and onwards is a subnetwork, Layer 1 and onwards is a subnetwork, Layer 2 and onwards is a subnetwork, etc.), then we can see how the input distribution to a subnetwork beginning at Layer i constantly changes as the weights of the previous layers and therefore the output distribution of Layer $i-1$ change. Each subnetwork suffers from such a form of covariate shift, which the authors of Batch Normalization dub as internal covariate shift, throughout the whole training process, leading to slower convergence and slightly worse performance. Batch Normalization seeks to reduce internal covariate shift by forcing the input distribution of each subnetwork (i.e. each layer) to remain as a normal distribution throughout the training process.

If you take a look at the figure below from the original paper, you can see that batch normalization increases convergence speed, that internal covariate shift is present among the inputs of the activation layers (especially in the earlier stages of training), and how Batch Normalization resolves internal covariate shift.

2.2 Implementation Details

Typically a Batch Normalization step is done before each non-linear activation function to ensure that each activation function gets a clean and normalized

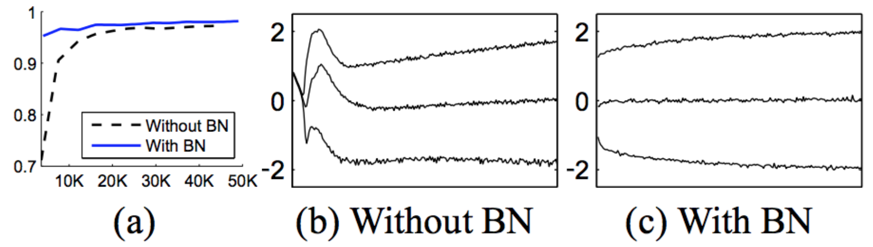

Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

input. Recall that activation functions are not learnable, so as long as the input distribution is nice, they won't exhibit divergent behavior regardless of the learning rate. Because of this, we don't need to apply batch normalization to the output of the non-linearity.

As in the case with gradients, estimating the mean and variance of the entire distribution of inputs for each subnetwork is costly, so at each iteration Batch Normalization estimates the mean and variance of the entire input distribution by calculating the mean and variance of the mini-batch (hence the name). Using the estimated mean and variance values μ and σ^2, the input x to each subnetwork is normalized to $\hat{x} = \frac{x-\mu}{\sigma}$.

You may be concerned that such a normalization step may reduce the representation power of the network. For example, if our non-linear activation function is some fat sigmoid function, then if the input distribution is normalized to have mean 0 and variance 1, then we would restrict the inputs to only the linear part of the function. To make sure that Batch Normalization is able to do at least as well as a network without Batch Normalization, the final output of each Batch Normalization layer is $y = \gamma\hat{x} + \beta$, where γ and β are learnable parameters. This way, if the optimal thing to do is indeed to have not normalize x at all, then the network can learn to set γ equal to σ and β equal to μ such that $y = x$.

A Batch Normalization layer keeps a running average of μ and σ^2, so at test time the running averages and the learned parameters are used to transform the input to each Batch Normalization layer.