Final Project: Exploring Batch Normalization

Zony Yu

Dec 1, 2021

Abstract

Batch Normalization is a method of standard normalization applied to the weighted sums of each layer in a neural network. Developed in 2015, Batch Normalization helps significantly speed up the training of deep neural networks by reducing the *internal covariate shift* in each layer, which describes the changes to the mean and variance of layer distributions during the training process. This paper goes into a detailed explanation of the backpropagation process which includes derivations that were not presented in the original paper, as well as exploring through experimentation the effects Batch Normalization has on training and hyperparameter search. All code used in this experiment can be found in my GitHub repo: `https://github.com/zony249/Batch-Norm`

1.0: Introduction

In the field of supervised learning, a mathematical model is fit to a training dataset using various optimization schemes such that the model can be used to make predictions on unseen data. However, due to the nature of most training data, optimization can often be difficult and un-optimal. In many multivariate learning problems, the input features are often on completely different scales. This leads to a highly eccentric cost function where gradients with respect to certain weights are much steeper than with respect to other weights. This requires learning rates to be very small in order not to overshoot in any dimension.

To solve this problem, many feature scaling techniques were developed to ensure that all input features are roughly on the same scale. One example is **standardization** [1] where raw features are processed by subtracting by the mean of the data and scaling by the standard deviation:

$$\hat{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

Here, each training feature j of training example i is standardized separately, where $\mu_j = \frac{1}{M}\sum_i x_j^{(i)}$ and $\sigma_j^2 = \frac{1}{M}\sum_i (x_j^{(i)} - \mu_j)^2$. Note that M represents the number of examples in the training set. This method is often considered to be a sufficient solution for linear models to mitigate highly eccentric cost functions. However, standardization is much less effective for neural networks.

To understand its shortcomings when applied to neural networks, consider the following example: suppose we model the linear transformation of a single layer within the NN:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

We may assume that the input features are standardized such that $\mathbf{x} \sim \mathcal{N}(0,1)$, however this by no means guarantee that $\mathbf{z} \sim \mathcal{N}(0,1)$. This is problematic because $\mathbf{z}$ (after passing it through a nonlinearity) becomes the input to the next layer. A non-standardized $\mathbf{z}$ introduces different behaviours on the succeeding layer's inputs depending on the nonlinearity used – ReLU functions will preserve all features of $\mathbf{z}$ that are positive, of which features could be on vastly different scales, and Sigmoid and Tanh functions will saturate on certain features of $\mathbf{z}$ if the magnitudes are large.

1.1: Batch Normalization

In the 2015 paper "*Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*" [2], the researchers propose a method to standardize the weighted sum of each layer, ensuring that their distributions remain stable during training. This elimination of "*Internal Covariate Shift*" allows the use of much larger learning rates without the gradients diverging, thus greatly accelerating convergence.

Since Batch Normalization (BN) is applied layer-wise in a neural network, there needs to be a way to backpropagate through each BN layer, which was not needed in prior linear models using input feature standardization. While the aforementioned paper presented the equations for the backpropagated gradients, in this paper we will discuss the derivation of backpropagation in greater detail. This paper will also present qualitative and quantitative differences between experiments carried out on neural networks with and without BN.

2.0: Formulation

Batch Normalization applies feature standardization to the weighted sum of each layer, which then the normalized outputs are fed into the nonlinearity. A concrete description is shown below:

Algorithm 2.0.0: Forward Propagation for a particular layer

```
Input: $\mathbf{x}^{(i)}$
Output: $\mathbf{a}^{(i)}$

1 Function forward($\mathbf{x}^{(i)}$) is
2     $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)}$
3     $\hat{\mathbf{y}}^{(i)} = \mathrm{BN}(\mathbf{z}^{(i)})$
4     $\mathbf{a}^{(i)} = f(\hat{\mathbf{y}}^{(i)})$
5     return $\mathbf{a}^{(i)}$
6 end
```

In the above algorithm, we use $\mathbf{x}$ to denote the inputs to the layer, $\mathbf{z}$ to denote the output of the weighted sum, $\hat{\mathbf{y}}$ to denote the standardized weighted sum, $\mathbf{a}$ to denote the nonlinearity output, and (i) represents index of the training element. It's worth mentioning that a common area of contention is the placement of BN with respect to the activation function. Although this topic has been researched [3], the results depended on the model architecture, so for this paper we will stick with the implementation of the original 2015 paper [2], placing BN **before** the nonlinearity.

Applying feature standardization on the weighted sum, we get the following:

$$\hat{\mathbf{z}}^{(i)} = \frac{\mathbf{z}^{(i)} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \tag{1}$$

Note that we add a small ϵ in the denominator for numeric stability. Here, $\boldsymbol{\mu} = \mathrm{E}[\mathbf{z}]$ and $\boldsymbol{\sigma}^2 = \mathrm{Var}[\mathbf{z}]$. BN computes the standardization of every example within a batch, so we should specify that $\mathbf{z}$ represents the output of the weighted sum with dimensions $M_{\text{batch}} \times N_x$, Where M_{batch} represents the batch size and N_x represents the number of features in $\mathbf{x}$.

Once we get the standardized output $\hat{\mathbf{z}}^{(i)}$, we observe that directly using this result can limit the hypothesis capacity of the neural network. Suppose we compare two weighted sums: $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)}$ and $\mathbf{z}'^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$. We can compute the means...

$$\boldsymbol{\mu} = \mathrm{E}[\mathbf{z}]$$

$$\boldsymbol{\mu}' = \mathrm{E}[\mathbf{z}'] = \mathrm{E}[\mathbf{z} + \mathbf{b}] = \boldsymbol{\mu} + \mathbf{b}$$

... As well as the variances:

$$\boldsymbol{\sigma}^2 = \mathrm{E}[\mathbf{z} - \boldsymbol{\mu}]^2$$

$$\boldsymbol{\sigma}'^2 = \mathrm{E}[\mathbf{z}' - \boldsymbol{\mu}']^2$$

$$= \mathrm{E}[\mathbf{z} + \mathbf{b} - \boldsymbol{\mu} - \mathbf{b}]^2$$

$$= \boldsymbol{\sigma}^2$$

We can then show that:

$$\hat{\mathbf{z}}'^{(i)} = \frac{\mathbf{z}'^{(i)} - \boldsymbol{\mu}'}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}$$

$$= \frac{\mathbf{z}^{(i)} + \mathbf{b} - \boldsymbol{\mu} - \mathbf{b}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}$$

$$= \hat{\mathbf{z}}^{(i)}$$

Here we observe that standardizing the weighted sum will ignore the bias term. BN can be made more flexible by scaling the normalized sum and reintroducing the bias term:

$$\hat{y}_j^{(i)} = \gamma_j \hat{z}_j^{(i)} + \beta_j$$

$$\hat{\mathbf{y}}^{(i)} = \boldsymbol{\gamma} \odot \hat{\mathbf{z}}^{(i)} + \boldsymbol{\beta} \tag{2}$$

The rescaling factor $\boldsymbol{\gamma}$ and bias term $\boldsymbol{\beta}$ are parameters learnable through backpropagation. One example of the improved flexibility is the ability to learn the identity function. Suppose $\boldsymbol{\gamma} = \sqrt{\boldsymbol{\sigma}^2 + \epsilon}$ and $\boldsymbol{\beta} = \boldsymbol{\mu}$, then we observe that $\hat{\mathbf{y}}^{(i)} = \mathbf{z}^{(i)}$.

2.1: Backpropagation

In the beginning of **section 2.0**, we formulated the order of which operations are performed in forward propagation through a single layer in *Algorithm 2.0.0.* Backpropagation [4] involves computing the gradients in the reverse order of computations in forward propagation, which requires computing several gradients with respect to the parameters of the BN layer. This includes calculating $\frac{\partial J}{\partial \mathbf{z}^{(i)}}$ for the i^{th} input to BN, as well as $\frac{\partial J}{\partial \gamma}, \frac{\partial J}{\partial \beta}$ to learn the scaling and shifting parameters.

We will refer extensively to *Figure 2.1.0* in the appendix to explain the process. This dependence graph shows the relations between each variable. For example, J directly dependent on all $\hat{\mathbf{y}}^{(i)}, 1 \leq i \leq M$. The three bolded vectors points to all variables that $\mathbf{z}^{(i)}$ has direct contributions to, namely $\boldsymbol{\mu}, \boldsymbol{\sigma}^2$, and $\hat{\mathbf{z}}^{(i)}$. Therefore, we can formulate the gradients from here. Note that for this portion of the paper, all all multiplication operations are assumed to be pointwise rather than matrix, in order to reduce clutter. Also, it's worth mentioning that $\boldsymbol{\gamma}$ and

$\boldsymbol{\beta}$ are intentionally left out of this particular diagram for the same reason.

$$\frac{\partial J}{\partial \mathbf{z}^{(i)}} = \frac{\partial J}{\partial \hat{\mathbf{z}}^{(i)}}\frac{\partial \hat{\mathbf{z}}^{(i)}}{\partial \mathbf{z}^{(i)}} + \frac{\partial J}{\partial \boldsymbol{\sigma}^2}\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}} + \frac{\partial J}{\partial \boldsymbol{\mu}}\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}^{(i)}} \tag{3}$$

Starting from the first term, we first compute $\frac{\partial J}{\partial \hat{\mathbf{z}}^{(i)}}$:

$$\frac{\partial J}{\partial \hat{\mathbf{z}}^{(i)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}^{(i)}}\frac{\partial \hat{\mathbf{y}}^{(i)}}{\partial \hat{\mathbf{z}}^{(i)}}$$

Referring to equation (2),

$$\frac{\partial \hat{\mathbf{y}}^{(i)}}{\partial \hat{\mathbf{z}}^{(i)}} = \boldsymbol{\gamma}$$

Therefore,

$$\frac{\partial J}{\partial \hat{\mathbf{z}}^{(i)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}^{(i)}}\boldsymbol{\gamma}$$

Referring to equation (1), we can compute $\frac{\partial \hat{\mathbf{z}}^{(i)}}{\partial \mathbf{z}^{(i)}}$:

$$\frac{\partial \hat{\mathbf{z}}^{(i)}}{\partial \mathbf{z}^{(i)}} = \frac{1}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}$$

The first term of equation (3) can be put together:

$$\frac{\partial J}{\partial \hat{\mathbf{z}}^{(i)}}\frac{\partial \hat{\mathbf{z}}^{(i)}}{\partial \mathbf{z}^{(i)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}^{(i)}}\boldsymbol{\gamma}\frac{1}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \tag{4}$$

Moving onto the second term, we compute $\frac{\partial J}{\partial \boldsymbol{\sigma}^2}$. Note that J's dependency on $\boldsymbol{\sigma}^2$ is split among many paths, so the gradient through each path must be accounted for.

$$\frac{\partial J}{\partial \boldsymbol{\sigma}^2} = \sum_{m=1}^{M} \frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\frac{\partial \hat{\mathbf{y}}^{(m)}}{\partial \hat{\mathbf{z}}^{(m)}}\frac{\partial \hat{\mathbf{z}}^{(m)}}{\partial \boldsymbol{\sigma}^2}$$

Recall that $\frac{\partial \hat{\mathbf{y}}^{(m)}}{\partial \hat{\mathbf{z}}^{(m)}} = \boldsymbol{\gamma}$. Now we calculate $\frac{\partial \hat{\mathbf{z}}^{(m)}}{\partial \boldsymbol{\sigma}^2}$, referring to equation (1):

$$\frac{\partial \hat{\mathbf{z}}^{(m)}}{\partial \boldsymbol{\sigma}^2} = -\frac{1}{2}(\mathbf{z}^{(m)} - \boldsymbol{\mu})(\boldsymbol{\sigma}^2 + \epsilon)^{-3/2}$$

Now we can complete $\frac{\partial J}{\partial \boldsymbol{\sigma}^2}$:

$$\frac{\partial J}{\partial \boldsymbol{\sigma}^2} = -\frac{1}{2}\sum_{m=1}^{M} \frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}(\mathbf{z}^{(m)} - \boldsymbol{\mu})(\boldsymbol{\sigma}^2 + \epsilon)^{-3/2} \tag{5}$$

We will now compute the second half of the second term in equation (3), namely $\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}}$. From the dependencies graph we note that there are two paths that lead from $\mathbf{z}^{(i)}$ to $\boldsymbol{\sigma}^2$, so the gradient must account for both contributions. The point-wise variance $\boldsymbol{\sigma}^2$ is represented by the following equation:

$$\boldsymbol{\sigma}^2 = \frac{1}{M}\sum_{m=1}^{M}(\mathbf{z}^{(m)} - \boldsymbol{\mu})^2 \tag{6}$$

And below is the gradient $\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}}$:

$$\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}} = \left[\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}}\right]_{\text{Direct}} + \frac{\partial \boldsymbol{\sigma}^2}{\partial \boldsymbol{\mu}}\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}^{(i)}} \tag{7}$$

Here $\left[\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}}\right]_{\text{Direct}}$ is the gradient resulting from the direct contribution of $\mathbf{z}^{(i)}$ to $\boldsymbol{\sigma}^2$, and the second term of (7) represents the contribution of $\mathbf{z}^{(i)}$ chained through $\boldsymbol{\mu}$. First, we compute $\left[\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}}\right]_{\text{Direct}}$ from equation (6):

$$\left[\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}}\right]_{\text{Direct}} = \frac{2}{M}(\mathbf{z}^{(i)} - \boldsymbol{\mu})$$

Now we can compute the second term of equation (7), starting with $\frac{\partial \boldsymbol{\sigma}^2}{\partial \boldsymbol{\mu}}$:

$$\frac{\partial \boldsymbol{\sigma}^2}{\partial \boldsymbol{\mu}} = -\frac{2}{M}\sum_{m=1}^{M}(\mathbf{z}^{(m)} - \boldsymbol{\mu})$$

Then, we compute $\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}^{(i)}}$, knowing that the pointwise mean $\boldsymbol{\mu}$ is represented by the equation $\boldsymbol{\mu} = \frac{1}{M}\sum_{m=1}^{M}\mathbf{z}^{(m)}$:

$$\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}^{(i)}} = \frac{1}{M} \tag{8}$$

Now, we can complete equation (7) by piecing together the gradients we just calculated:

$$\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}} = \frac{2}{M}(\mathbf{z}^{(i)} - \boldsymbol{\mu}) - \frac{2}{M^2}\sum_{m=1}^{M}(\mathbf{z}^{(m)} - \boldsymbol{\mu})$$

Combining (7) with (5), we put together the second term of equation (3). The next step is to simplify the term.

$$\frac{\partial J}{\partial \boldsymbol{\sigma}^2}\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}} = \left[-\frac{1}{2}\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}(\mathbf{z}^{(m)} - \boldsymbol{\mu})(\boldsymbol{\sigma}^2 + \epsilon)^{-3/2}\right]$$
$$\left[\frac{2}{M}(\mathbf{z}^{(i)} - \boldsymbol{\mu}) - \frac{2}{M^2}\sum_{m=1}^{M}(\mathbf{z}^{(m)} - \boldsymbol{\mu})\right]$$

$$= \frac{2}{M}(\mathbf{z}^{(i)} - \boldsymbol{\mu})\left[-\frac{1}{2}\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}(\mathbf{z}^{(m)} - \boldsymbol{\mu})(\boldsymbol{\sigma}^2 + \epsilon)^{-3/2}\right]$$
$$-\frac{2}{M^2}\sum_{m=1}^{M}(\mathbf{z}^{(m)} - \boldsymbol{\mu})\left[-\frac{1}{2}\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}(\mathbf{z}^{(m)} - \boldsymbol{\mu})(\boldsymbol{\sigma}^2 + \epsilon)^{-3/2}\right]$$

Here we observe that $(\boldsymbol{\sigma}^2+\epsilon)^{-3/2}$ is invariant with respect to m, so we can factor that out of the sums. We also observe that $(\boldsymbol{\sigma}^2+\epsilon)^{-3/2} = \frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}$. Taking this into consideration, we continue the simplification:

$$= \frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg[-\frac{1}{M}\frac{(\mathbf{z}^{(i)}-\boldsymbol{\mu})}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\frac{(\mathbf{z}^{(m)}-\boldsymbol{\mu})}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg] + \frac{1}{M^2}\sum_{m=1}^{M}\frac{(\mathbf{z}^{(m)}-\boldsymbol{\mu})}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\frac{(\mathbf{z}^{(m)}-\boldsymbol{\mu})}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg]\Bigg]$$

Here, we notice that $\hat{\mathbf{z}}^{(i)} = \frac{\mathbf{z}^{(i)}-\boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}$, so we can substitute that in, completing the simplification.

$$\frac{\partial J}{\partial \boldsymbol{\sigma}^2}\frac{\partial \boldsymbol{\sigma}^2}{\partial \mathbf{z}^{(i)}} = \frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg[-\frac{1}{M}\hat{\mathbf{z}}^{(i)}\Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\hat{\mathbf{z}}^{(m)}\Bigg] + \frac{1}{M^2}\sum_{m=1}^{M}\hat{\mathbf{z}}^{(m)}\Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\hat{\mathbf{z}}^{(m)}\Bigg]\Bigg] \tag{9}$$

Finally, we calculate the final term of equation (3). Starting off with $\frac{\partial J}{\partial \boldsymbol{\mu}}$, we observe that $\boldsymbol{\mu}$ is a dependency of $\hat{\mathbf{z}}^{(1)}$,..., $\hat{\mathbf{z}}^{(M)}$ as well as $\boldsymbol{\sigma}^2$. Note that we have already accounted for the contribution of $\boldsymbol{\mu}$ to J through $\boldsymbol{\sigma}^2$ in equation (7), so we do not need to compute that a second time.

$$\frac{\partial J}{\partial \boldsymbol{\mu}} = \sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\frac{\partial \hat{\mathbf{z}}^{(m)}}{\partial \boldsymbol{\mu}}$$

Next, we calculate $\frac{\partial \hat{\mathbf{z}}^{(m)}}{\partial \boldsymbol{\mu}}$ referencing equation (1):

$$\frac{\partial \hat{\mathbf{z}}^{(m)}}{\partial \boldsymbol{\mu}} = -\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}$$

Thus, we get the full equation for $\frac{\partial J}{\partial \boldsymbol{\mu}}$:

$$\frac{\partial J}{\partial \boldsymbol{\mu}} = -\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}$$

The latter half of the third term $\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}^{(i)}}$ is already calculated in equation (8), so we will reuse that calculation. The last term of equation (3) is as follows:

$$\frac{\partial J}{\partial \boldsymbol{\mu}}\frac{\partial \boldsymbol{\mu}}{\partial \mathbf{z}^{(i)}} = -\frac{1}{M}\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma} \tag{10}$$

putting together $\frac{\partial J}{\partial \mathbf{z}^{(i)}}$, we sum together equations (4), (9), and (10) and simplify:

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{z}^{(i)}} &= \frac{\partial J}{\partial \hat{\mathbf{y}}^{(i)}}\boldsymbol{\gamma}\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}} \\ &\quad + \frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg[-\frac{1}{M}\hat{\mathbf{z}}^{(i)}\Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\hat{\mathbf{z}}^{(m)}\Bigg] \\ &\quad + \frac{1}{M^2}\sum_{m=1}^{M}\hat{\mathbf{z}}^{(m)}\Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\hat{\mathbf{z}}^{(m)}\Bigg]\Bigg] \\ &\quad - \frac{1}{M}\frac{1}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\end{aligned}$$

$$\begin{aligned} &= \frac{1}{M\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\Bigg(M\frac{\partial J}{\partial \hat{\mathbf{y}}^{(i)}}\boldsymbol{\gamma} \\ &\quad + \Bigg[\sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\hat{\mathbf{z}}^{(m)}\Bigg]\Bigg[\Bigg[\frac{1}{M}\sum_{m=1}^{M}\hat{\mathbf{z}}^{(m)}\Bigg] - \hat{\mathbf{z}}^{(i)}\Bigg] \\ &\quad - \sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\boldsymbol{\gamma}\Bigg)\end{aligned} \tag{11}$$

This gradient calculation for $\frac{\partial J}{\partial \mathbf{z}^{(i)}}$ is required to propagate gradients through the BN layer to earlier layers. There are two other gradients to calculate, namely $\frac{\partial J}{\partial \gamma}$ and $\frac{\partial J}{\partial \boldsymbol{\beta}}$. Luckily, these two gradients are much easier to compute, and do not require pages of derivation. Referring to *Figure 2.1.1* in the appendix (which is modelled after equation (2)), scaling and shifting parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are dependencies of every $\hat{\mathbf{y}}^{(1)}, \ldots, \hat{\mathbf{y}}^{(M)}$, therefore when computing these gradients, we need to sum every contribution:

$$\begin{aligned}\frac{\partial J}{\partial \gamma} &= \sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\frac{\partial \hat{\mathbf{y}}^{(m)}}{\partial \gamma} \\ &= \sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\hat{\mathbf{z}}^{(m)}\end{aligned} \tag{12}$$

$$\begin{aligned}\frac{\partial J}{\partial \boldsymbol{\beta}} &= \sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\frac{\partial \hat{\mathbf{y}}^{(m)}}{\partial \boldsymbol{\beta}} \\ &= \sum_{m=1}^{M}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(m)}}\end{aligned} \tag{13}$$

2.2: Inference

Batch Normalization learns the scaling and shifting factors $\boldsymbol{\gamma}, \boldsymbol{\beta}$ during the training process and uses the learned parameters at inference time. While these are the only parameters in BN that are learned via the backpropagation process, these are not the only parameters carried forward from training. During the inference process, we use the population mean and variance of the training set for the standardization calculations.

$$\hat{\mathbf{z}}^{(i)} = \frac{\mathbf{z}^{(i)} - \mathrm{E}[\mathbf{z}_{train}]}{\sqrt{\mathrm{Var}[\mathbf{z}_{train}] + \epsilon}} \tag{14}$$

The original papers [2] mention why the population statistics were used rather than the inference mini-batch statistics – we want the inference output to depend only on the input, rather than other inputs in the mini-batch. This is especially apparent if mini-batch sample is small. The sample statistics of batch sizes smaller than 30 [5] can vary wildly from the population statistics. There are also applications of inference where the mini-batch size of 1 is desired. In these cases, sample variance is unobtainable.

The population mean and variance are computed at train time, where an exponentially weighted average of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are kept. The exponentially weighted average of $\boldsymbol{\mu}$ is shown below:

$$\mathrm{E}[\mathbf{z}_{train}]_t = \alpha \mathrm{E}[\mathbf{z}_{train}]_{t-1} + (1-\alpha)\boldsymbol{\mu}_t$$

We chose $\alpha = 0.97$, which roughly translates to averaging over the last 30 sample means. The same is done to keep track of the second moment.

3.0: Batch Normalization in Practice

In the previous section, we went through pages of derivations to formulate the forward propagation, backpropagation, and inference behaviours of Batch Norm. This section goes through the experimental implementation of BN, discussing about training performance, validation and testing performance, as well as inference computational penalties. All metrics will be compared to the same fully-connected neural network without BN.

3.1: Testing Methodology

Firstly it's important to talk about the dataset used to conduct the tests. The Higgs Dataset [6] is produced by Monte Carlo simulations of particles in a particle accelerator. The dataset contains 11 million data points with 28 features each, as well binary labels representing whether the signals produced Higgs bosons, or if they are simply background signals. The full Higgs Dataset contains 11 million data points and takes up 7.5 GB of storage, however due to the computational deficiencies of training the model on CPU, we took a subset of the full dataset containing only 495,000 data points.

We implemented two 5-layer neural networks, one with batch normalization, and one without as a control. Other than the BN layers, the model architecture are identical, with 4 hidden layers with 1000 neurons each, terminating with a binary classifier. For the full model diagrams, refer to *Figure 3.1.0* in the appendix. Both networks were implemented in NumPy (i.e. no high-level ML libraries were used), and the BN implementation strictly follows the formulation described in **section 2**.

The training heuristics are identical for each model. The hyperparameter search process used a genetic algorithm [7] to find the best performing hyperparameters for each model. While the hyperparameters will not be identical between the control and batch normalized models, we are still subjecting both models to the same heuristics, thus both models are given equal opportunities to search for ideal hyperparameters that maximize validation accuracy.

For each generation, the genetic algorithm trains 10 models with randomly selected hyperparameters within a set of bounds, and sorts the models by validation accuracy. The two hyperparameters being optimized are the L2 regularization coefficient and the learning rate. Then, the top 3 models within each generation are selected and using gaussian mutation [8], the bounds of each hyperparameter are updated. We train for 10 generations total, with 10 models per generation, where each model is trained for 10 epochs. Refer to *Algorithm 3.2.0* in the appendix for the hyperparameter search algorithm.

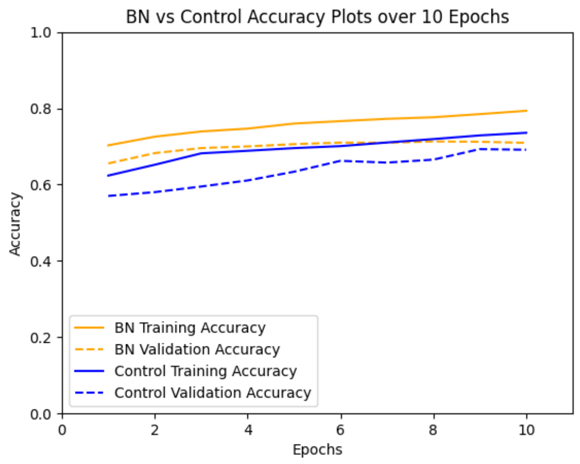
After the ideal hyperparameters are found, the models are then trained one last time for 10 epochs before evaluating them on the test set.

3.2: Experimental Results and Discussion

The hyperparameter search process is where we see the first differences between the batch-normalized model and the control. Referring to *Table 3.2.1* from the appendix, we see that the ideal learning rate for the batch-normalized model is 2.57, significantly **higher** than the 0.62 for the control model. This corroborates with the reportings from the original paper, where BN allows for larger learning rates without risking the model diverging.

The effects of the larger learning rate are shown when we look at the training loss and accuracy at the final epoch – the BN model shows lower training loss (0.3379 vs 0.4846) and higher training accuracy (0.8927 vs 0.7586) than the control model. While we often do not compare training metrics between models, this shows that the BN model can perform more aggressive gradient steps than the control model.

We also notice that the L2 coefficients obtained from hyperparameter search are **lower** for the BN model than the values for the control model (1.26×10^{-8} vs. 3.98×10^{-8}), which suggests that BN also provides slight regularization effects. The original papers [2] also arrived on the same conclusion, stating that the strength of Dropout [9] (or other methods of regularization) can be reduced or removed altogether in a Batch Normalized network.



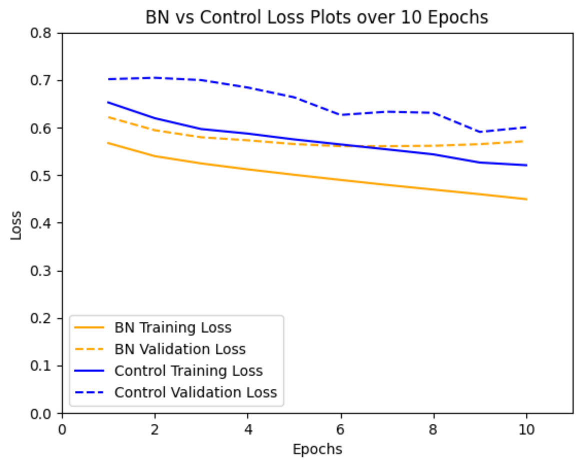


Figure 3.2.2: Accuracy and Loss comparisons between batch-normalized and control models

The validation performance of the BN model also shows improvements over the control model, achieving a peak validation accuracy of 0.7130 after training for 8 epochs vs. 0.6929 after 9 epochs. Referring to *Figure 3.2.2*, we can observe that the metrics of the BN model performs consistently better than the control model, and arrives at peak validation performance in less gradient steps than the control model. In the end, the batch-normalized network achieved a test accuracy of 0.7127, compared to 0.6872 of the control model.

3.3: Limitations of Batch Normalization

So far, we've seen the benefits of Batch Normalization as it standardizes and stabilizes the layer distributions in neural networks, allowing for much faster training as well as higher peak validation and test performance. That said, implementing BN comes with a performance penalty. We benchmarked the training and inference performance of the batch normalized and control model, of which the performance values can be found in *Table 3.2.1* (Appendix). During inference, the BN model needs to compute equations (14) and (2) during forward propagation at each layer, leading to a 25.3% slowdown for the BN model. During the training process, the performance penalty is even greater, as in addition to the forward propagation overhead of BN, we also need to calculate equations (11), (12), and (13) at every layer during backprop, leading to an overall slowdown of 35.6%. The performance penalty during inference may be an issue for tasks that are time-sensitive, such as real time object detection, while the performance penalty during training is often not a concern, as BN already speeds up training significantly.

4.0: Conclusion and Final Remarks

Batch Normalization extends on the idea of feature standardization, applying standardization operations on the weighted sums of each layer over a mini-batch, then scaling and shifting by parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$ learnt through gradient descent. These operations reduce the internal covariate shift during training, allowing much larger gradient steps to be performed without diverging.

In this paper we presented the in-depth derivation of backpropagation of BN and implemented the derived formulation in a simple fully-connected model. The BN model was then compared against architecturally-identical control model on the Higgs Dataset [6]. From the experimentation we found that the BN model supported higher learning rates and did not need as much regularization as the control. While BN introduces runtime performance penalties when training and predicting, for our experiment, the benefits of faster convergence far outweighed the drawbacks of slower gradient iterations. **All in all, the BN model trained faster and performed better on the test set than the control model**.

Finally, as a follow-up to this paper, we can explore implementing BN on different model architectures, as well as experimenting with the placement of BN within a layer given different nonlinearities and weight initializations.

5.0: References

[1] Sebastian Raschka, “About feature scaling and normalization – and the effect of standardization for machine learning algorithms,” Jul. 11, 2014. [Online]. Available: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

[2] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html

[3] M. Hasani and H. Khotanlou, “An empirical study on position of the batch normalization layer in convolutional neural networks,” in *2019 5th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, 2019, pp. 1–4.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[5] S. G. Kwak and J. H. Kim, “Central limit theorem: the cornerstone of modern statistics,” *Korean journal of anesthesiology*, vol. 70, no. 2, pp. 144–156, Apr 2017, 28367284[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/28367284

[6] P. Baldi, P. Sadowski, and D. Whiteson, “Searching for exotic particles in high-energy physics with deep learning,” *Nature Communications*, vol. 5, no. 1, p. 4308, Jul 2014. [Online]. Available: https://doi.org/10.1038/ncomms5308

[7] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, ser. Engineering Design and Automation. Wiley, 1999. [Online]. Available: https://books.google.ca/books?id=U7MuV1q6P1oC

[8] N. Higashi and H. Iba, “Particle swarm optimization with gaussian mutation,” in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS’03 (Cat. No.03EX706)*, 2003, pp. 72–79.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, jan 2014.

6.0: Appendix

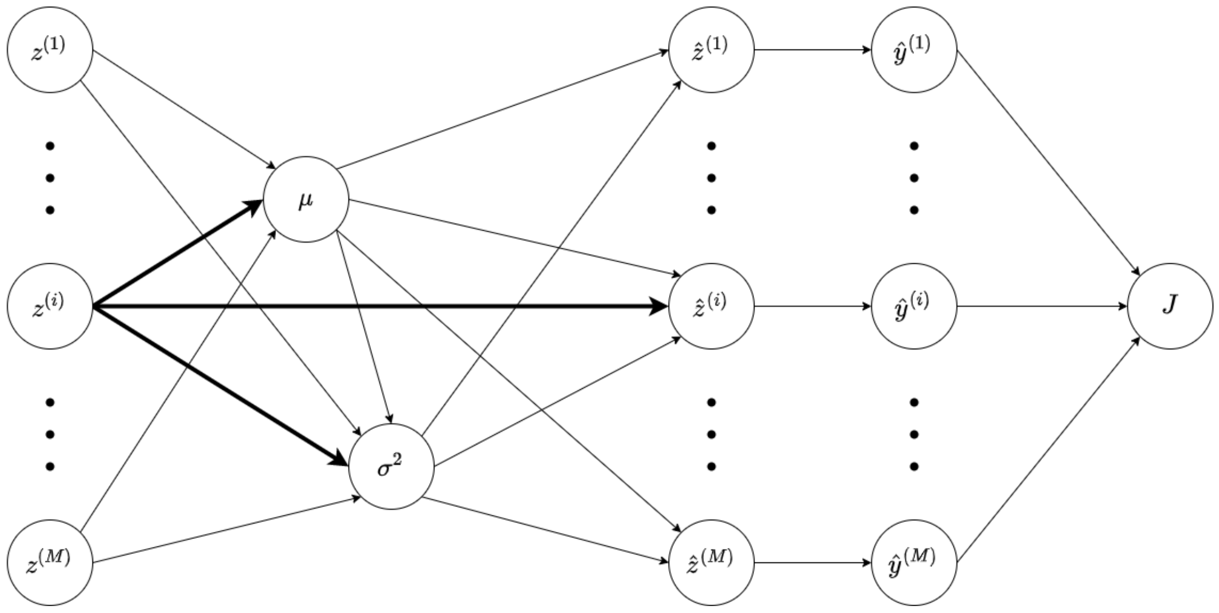


Figure 2.1.0: Dependencies graph of J with respect to $z^{(i)}$

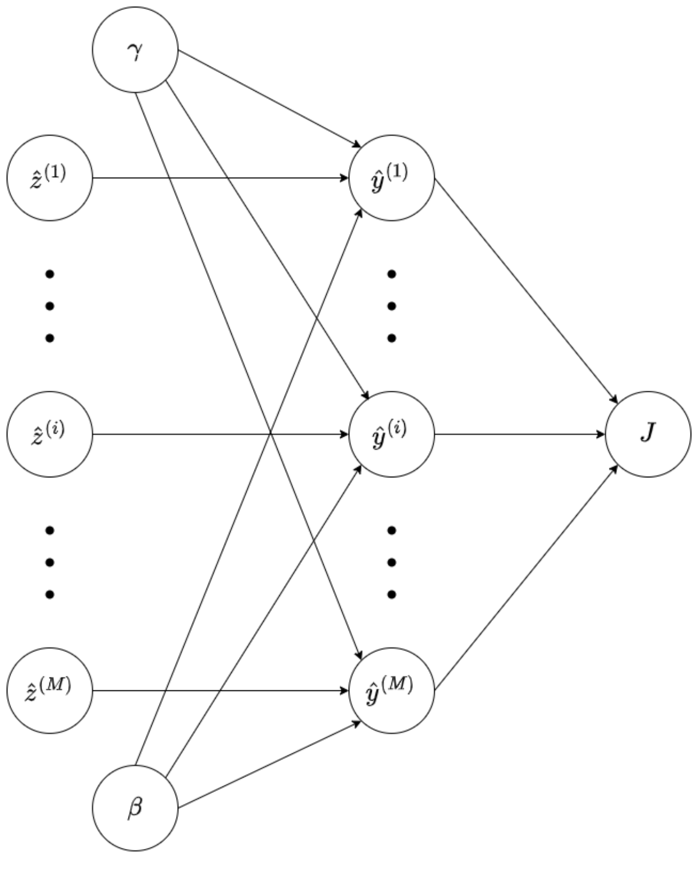
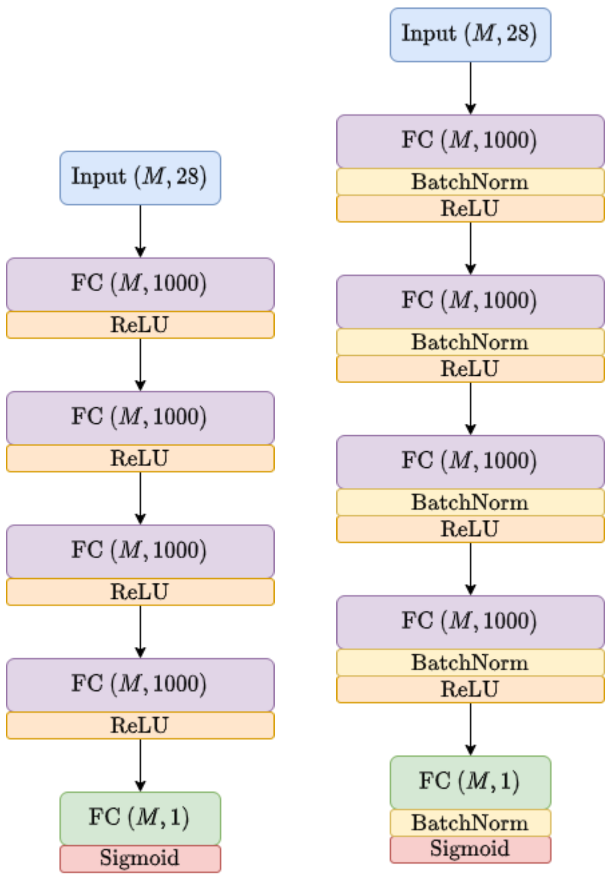


Figure 2.1.1: Dependencies graph of J with respect to $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$



(a) NN control (b) NN with Batch Norm

Figure 3.1.0: The control model and the Batch Normalized model

Algorithm 3.2.0: Hyperparameter Search

```
1  $\lambda_{upper} \leftarrow 0$        // L2 bounds
2  $\lambda_{lower} \leftarrow -10$
3  $\alpha_{upper} \leftarrow 0$        // Learning Rate bounds
4  $\alpha_{lower} \leftarrow -10$
5  for 10 Generations do
6      for 10 models do
7          $\lambda_{model} \leftarrow 10^{\text{Rand}(\lambda_{lower}, \lambda_{upper})}$
8          $\alpha_{model} \leftarrow 10^{\text{Rand}(\alpha_{lower}, \alpha_{upper})}$

9          model $\leftarrow$ new NN using $\lambda_{model}, \alpha_{model}$
10         Train model for 10 epochs
11         Save best validation accuracy
12     end
       /* ************ Gaussian Mutation ***********  */
13     Select top 3 models based on val accuracy

       // Mean and stddev of L2 bounds of top 3 models
14     $\overline{\lambda_{bnds}} \leftarrow \text{mean}(\log_{10}(\text{top3}.\lambda_{model}))$
15     $\text{std}(\lambda_{bnds}) \leftarrow \text{stddev}(\log_{10}(\text{top3}.\lambda_{model}))$

       // Mean and stddev of learning rate bounds of
          top 3 models
16     $\overline{\alpha_{bnds}} \leftarrow \text{mean}(\log_{10}(\text{top3}.\alpha_{model}))$
17     $\text{std}(\alpha_{bnds}) \leftarrow \text{stddev}(\log_{10}(\text{top3}.\alpha_{model}))$

       // update bounds with corresponding first and
          second moments
18     $\lambda_{upper}, \lambda_{lower} \leftarrow \overline{\lambda_{bnds}} \pm 1.5\text{std}(\lambda_{bnds})$
19     $\alpha_{upper}, \alpha_{lower} \leftarrow \overline{\alpha_{bnds}} \pm 1.5\text{std}(\alpha_{bnds})$
20 end
21 $\hat{\lambda} \leftarrow 10^{\frac{\lambda_{upper}+\lambda_{lower}}{2}}$
22 $\hat{\alpha} \leftarrow 10^{\frac{\alpha_{upper}+\alpha_{lower}}{2}}$
23 return $\hat{\lambda}, \hat{\alpha}$
```

Table 3.2.1: Observations after Hyperparameter Search

Data	**Control Model**	**BN Model**
Learning Rate	0.62	2.57
L2	3.98×10^{-8}	1.26×10^{-8}
Best Epoch	9	8
Validation Accuracy (best epoch)	0.6929	0.7130
Training Accuracy (best epoch)	0.7433	0.8467
Validation Loss (best epoch)	0.6005	0.5651
Training Loss (best epoch)	0.5001	0.3864
Training Accuracy (epoch 10)	0.7586	0.8927
Training Loss (epoch 10)	0.4846	0.3379
Training step time (Batch size 1024)	0.249s	0.387s
Inference step time (Batch size 1024)	0.124s	0.166s
Test Accuracy (best epoch)	**0.6872**	**0.7127**