基於卷積神經網路訓練之批量標準化高效率電路設計

New Efficient Hardware Design for Batch Normalization Based on Convolution Neural Network Training

作者：丁昱升、鄧宇凡

指導教授/實驗負責教授：闕志達

摘要(Abstract)：

在卷積神經網路的訓練中，常常加入批次標準化層以提高訓練的速度以及效能，然而過去針對批次標準化層的硬體設計大部分皆不支援訓練的過程，或是在硬體資源的利用上，以及與其他層運算單元的配合上有許多可以改進的空間。因此本研究提出了新的資料流以及演算法，透過管線化設計，讓卷積運算與批次標準化運算能夠同時進行，並且利用前向傳播與反向傳播在理論上的對稱性，增加硬體資源的重複使用率。最後，透過 MNIST 手寫數字辨識的神經網路訓練模擬，顯示此設計能在不損失精準度的條件下，完成批次標準化層的硬體加速。

一、 簡介

批次標準化(Batch normalization)是一種用於提高神經網路訓練速度、效能和穩定性的技術。在深層網路的訓練中，由於網路的參數變化而引起內部結點數據分布發生變化的過程被稱作內部協變量偏移(Internal Covariate Shift)。此現象會造成上層網路需要不停調整來適應輸入數據分布的變化，進而造成網路學習速度的降低。另外，網路的訓練過程也容易陷入梯度飽和區，減緩網路的收斂速度。因此 Sergey Ioffe 與 Christian Szegedy 提出了批次標準化的技術來解決這個問題[1]。以下考慮一個批次的訓練，假設訓練樣本數為 N，而$x^{(i)}$表示第 i 個小批次(mini-batch)的訓練數據，並關注在輸入的第 j 個維度，可以透過式(3)對該維度進行標準化：

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} \quad \text{式(1)}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} (x_j^{(i)} - \mu_j)^2 \quad \text{式(2)}$$

$$\hat{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \quad \text{式(3)}$$

其中ε是為了防止分母為 0 產生無效計算。

此時雖然減緩了內部協變量偏移的問題，讓每一層網路的輸入數據分布都變得穩定，但卻因為轉換操作改變了原本數據的訊息，造成數據表達能力的缺失。除此之外，如果下一層的激活函數為 sigmoid 函數或是 tanh 函數，如此的資料分布容易陷入這些非線性激活函數的線性區域。因此，批次標準化引入了兩個可學習的參數γ與β。透過這兩個參數可以對標準化後的數據進行線性轉換，如式(4)：

$$y_j{}^{(i)} = \gamma_j \hat{x}_j^{(i)} + \beta_j \quad \text{式(4)}$$

透過上述步驟，即可以確保了輸入資料的表達能力。

而在訓練過程中，需要根據損失函數(Loss function)，並透過梯度下降法(Gradient descent)求出每個權重的更新量。根據鏈鎖律，假設下一層傳入的損失梯度值為$\frac{\partial \mathrm{L}}{\partial y^{(i)}}$，可以推導出批次標準化層反向傳播(Backpropagation)的損失梯度函數如式(5)、式(6)以及式(10)：

$$\frac{\partial \mathrm{L}}{\partial \gamma} = \sum_{i=1}^{N} \frac{\partial \mathrm{L}}{\partial y^{(i)}} \hat{\mathrm{x}}^{(i)} \quad \text{式(5)}$$

$$\frac{\partial \mathrm{L}}{\partial \beta} = \sum_{i=1}^{N} \frac{\partial \mathrm{L}}{\partial y^{(i)}} \quad \text{式(6)}$$

$$\frac{\partial \mathrm{L}}{\partial \hat{\mathrm{x}}^{(i)}} = \frac{\partial \mathrm{L}}{\partial y^{(i)}} \gamma \quad \text{式(7)}$$

$$\frac{\partial \mathrm{L}}{\partial \sigma^2} = \sum_{i=1}^{N} \frac{\partial \mathrm{L}}{\partial \hat{\mathrm{x}}^{(i)}} \left(x^{(i)} - \mu\right) \frac{-1}{2} \left(\sigma^2 + \varepsilon\right)^{-\frac{3}{2}} \quad \text{式(8)}$$

$$\frac{\partial \mathrm{L}}{\partial \mu} = \sum_{i=1}^{N} \frac{\partial \mathrm{L}}{\partial \hat{\mathrm{x}}^{(i)}} \frac{-1}{\sqrt{\sigma^2 + \varepsilon}} \quad \text{式(9)}$$

$$\frac{\partial \mathrm{L}}{\partial x^{(i)}} = \frac{\partial \mathrm{L}}{\partial \hat{\mathrm{x}}^{(i)}} \frac{1}{\sqrt{\sigma^2 + \varepsilon}} + \frac{\partial \mathrm{L}}{\partial \sigma^2} \frac{2(x^{(i)} - \mu)}{N} + \frac{\partial \mathrm{L}}{\partial \mu} \frac{1}{N} \quad \text{式(10)}$$

而針對神經網路訓練的加速硬體，現有的研究大多是針對卷積運算去進行電路設計，批次標準化則仍多以軟體的方式進行運算。而在少數有提到批次標準化電路設計的研究當中[2][3]，大多數又都是針對推論(Inference)的過程進行加速。然而在批次標準化的演算法當中，推論時的統計參數是常數，不會隨著輸入資料變動，因此在電路設計上較為容易。也因為如此，在上述提到的研究中，都是將標準化的運算以及後續的激活函數，甚至是卷積層的運算結合。然而，如果是要針對訓練的過程做加速，每個批次的輸入資料都會有不同的統計參數，因此需要另外設計電路對此進行運算。除此之外，從上述梯度函數的推導中可以看出，在反向傳播的過程當中需要重複使用到一些前向傳播(Forward)已經計算過的數據(如標準化過後的數據$\hat{\mathrm{x}}^{(i)}$)。

而對於上述研究當中把批次標準化與其他運算結合的設計來說，將無法在前向傳播時儲存這些數據以提供給反向傳播時使用，因此如果基於此類電路設計進行反向傳播電路的延伸設計，將會有計算效率不佳的問題存在。

另外也有研究是針對可以同時支援推論以及訓練的批次標準化層進行電路設計[4][5]。在 Yang 等人的研究當中，前向傳播與反向傳播的電路是分開設計的。然而其實反向傳播的運算經過化簡之後，可以觀察到與前向傳播有非常高的對稱性，因此可以藉由透過相同的資料流(Dataflow)以及部分電路的共用達到較有效率的硬體設計。另外，在 Shuang 等人的研究當中，是使用 L1 範數(L1-norm)的版本進行批次標準化演算法的電路實作，然而此研究中只有針對批次標準化層本身進

行設計，而未顧及與網路中前一層運算電路的互動關係，導致在整個神經網路訓練的過程中，不同運算層之間的搭配有可以改善的空間。事實上，在許多神經網路的架構設計當中，批次標準化層都會接在卷積層（Convolution layer）後面，因此本研究想要設計出可以與卷積運算電路搭配的批次標準化電路，並且透過管線化設計(Pipeline Design)，讓卷積運算單元以及批次標準化電路都盡量同時在工作，而減少任何硬體資源閒置的時間。

二、 實驗原理與設計架構

1. 實驗原理

在本研究中，假設前一層的卷積運算電路是採用脈動陣列(systolic array)的設計方式，由 16 個處理元件(processing element)組成 4x4 的二維核心運算陣列，如圖 2.1 所示。

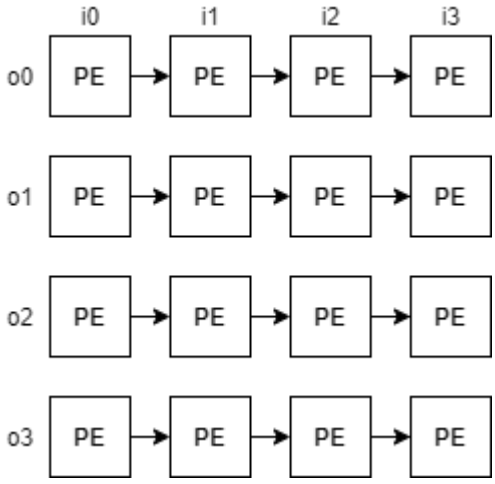

圖 2.1 核心運算陣列

一個處理元件可以處理 9 組權重與輸入資料乘加，並以 FP32 的型態輸出。在此假設下，一個處理元件剛好可以完成一個卷積核(kernel)大小為 3x3 的影像資料的卷積運算。在此 4x4 的二維核心運算陣列當中，同一欄的處理元件會輸入相同的影像資料，而不同列的處理元件會輸入不同輸出通道(output channel)的卷積核權重。如此一來，一個處理元件列的最右側將會得到 4 個不同輸出通道的輸入通道(input channel)與卷積核卷積結果的累加。為

了達到資料重複利用，每一欄的處理元件將會抓取當下 3x3 影像資料右側的 3 個新像素，形成右移一個像素單位的新 3x3，並對此 3x3 新的輸入影像重複上述的卷積運算，持續右移直到輸入影像的此三列影像的前 4 個輸入通道都已加總完成。如果卷積的輸入通道大於 4，核心運算陣列將重新載入下 4 個輸入通道的輸入影像資料以及卷積核權重，並重複上述步驟，直到此三列影像的所有輸入通道的卷積結果均加總完成。以此推類，接著依序對列、批次、輸出通道進行迭代完成運算，即圖 2.2 所示的資料流。

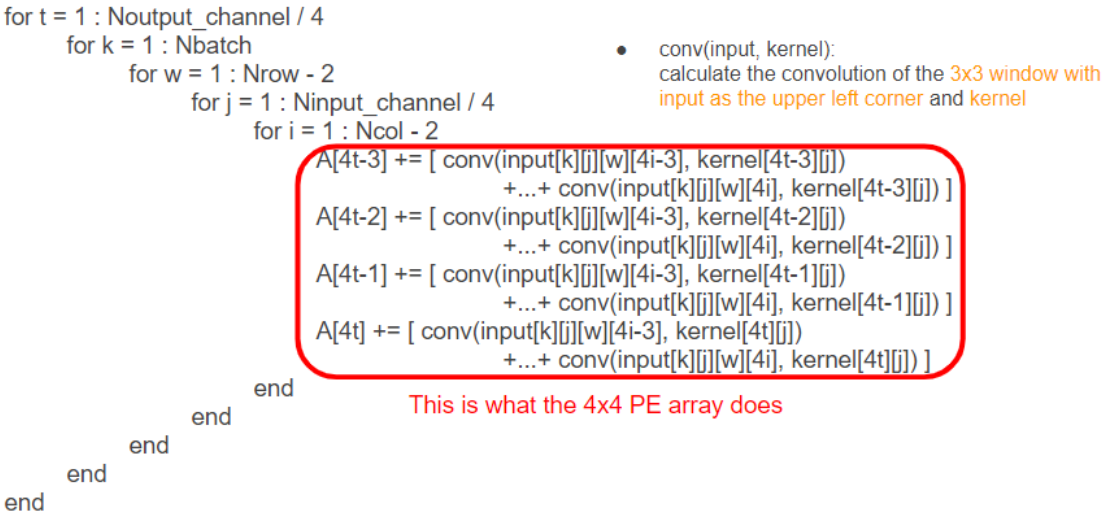

圖 2.2 核心運算陣列資料流

在此過程中，這些加總值會被存入批次標準化模組當中的暫存器(詳細電路圖會在後面說明)，而此加總的階段則稱為 Stage 1。

對於反向傳播來說，要計算出γ與β的損失梯度值，同樣要對輸入的損失梯度值進行加總，也就是說此部份(Stage 1)的電路設計以及資料流可以完全與前向傳播共用。

Stage 1 完成後，前向傳播需要對資料進行統計參數的計算以及資料的標準化以及縮放(乘以γ，再加β)，此部分的運算即為前向傳播的 Stage 2。而反向傳播則需要計算出輸入端的損失梯度值，此值可由式(10)整理成式(11)：

$$\frac{\partial L}{\partial x^{(i)}} = \frac{\gamma}{\sqrt{\sigma^2+\varepsilon}}\left(\frac{\partial L}{\partial y^{(i)}} - \frac{1}{N}\frac{\partial L}{\partial \beta} - \frac{\hat{x}^{(i)}}{N}\frac{\partial L}{\partial \gamma}\right) \quad \text{式(11)}$$

此式的運算則為反向傳播的 Stage 2。

2. 設計架構

圖 2.3 為本研究之整體電路設計架構圖，此架構主要分為三部分：核心運算陣列（PE array）、批次標準化運算處理器（BN_processor）和靜態隨機存取記憶體（SRAM），其中批次運算處理器架構再依照本報告之實驗原理所區分出的 Stage 1 和 Stage 2，細分為 Forward/Backward Stage 1 (FBS1)和分別針對前向傳播和反向傳播時的批次標準化運算所需的模組 Forward Stage 2 (FS2)和 Backward Stage 2 (BS2)。

在整個訓練過程中，只要卷積層後面有接批次標準化層，我們就會重複使用此架構。上一層的輸入資料會先經過核心運算陣列做卷積運算後一次輸出 4 個輸出通道的值至批次標準化運算處理器，再根據前向傳播或反向傳播的運算後傳至下一層，而 FBS1 和 FS2（前向傳播時）或 BS2（反向傳播時）為同時運作，以加速訓練。以下將詳細介紹此架構的細節和整個訓練過程在此架構下的資料流。

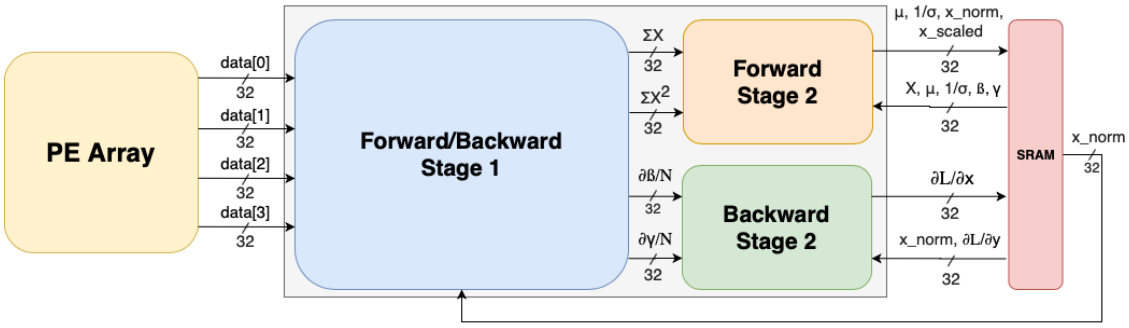

圖 2.3、整體電路架構

a. 前向傳播：

(i) Stage 1

此階段將核心運算陣列做卷積運算後之輸出值累加，再送入下一階段計算平均（Mean）和標準差（Standard deviation）。

圖 2.4 為 FBS1 架構的其中一組，總共有四組，每組分別對應核心運算陣列一次輸出的 4 個值。每組由 2 個 4x32bit 的暫存器（A_mean, B）、1 個

32x32bit 的暫存器 (A_x)、平方器、乘法器、加法器所組成。在此階段，核心運算陣列會根據輸入資料大小、輸入通道、批次不斷做卷積運算後輸入 FBS1 並累加存入 A_mean 和 A_x，而 A_x 的值會進一步輸出至靜態隨機存取記憶體，並做平方運算再累加存入 B（此階段不會用到乘法器），當累加完成後就進入 FS2，同時下一筆輸入資料會送進來繼續做運算。

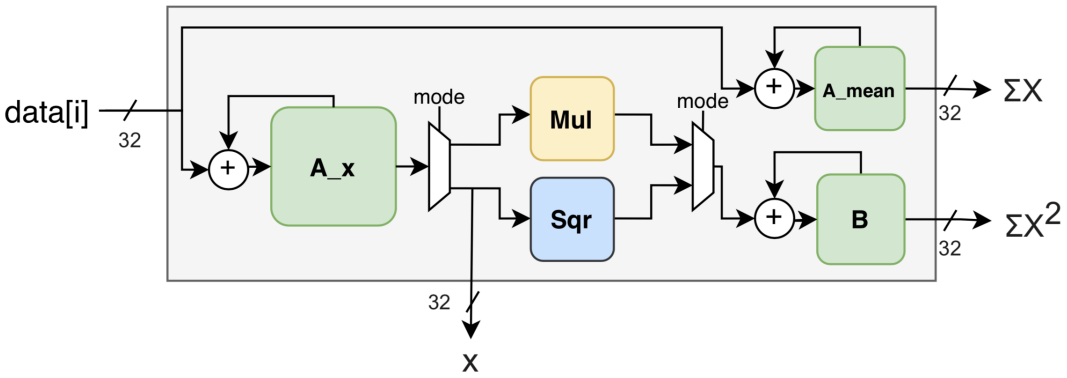

圖 2.4、Forward/Backward Stage 1 電路架構

(ii) Stage 2:

此階段對上一階段輸入之資料進行統計參數的計算（平均、標準差）以及對每個資料做標準化以及縮放(乘以 γ ，再加 β)。

FS2 的架構共分為兩部分：一部分（如圖 2.5）為計算上述之平均和標準差，其設計由乘法器、平方器、開根號計算器、倒數計算器所組成；另一部分（如圖 2.6）為計算上述之資料標準化以及縮放。

在圖 2.5 架構中，我們在計算平均前先將計算平均的參數（N），即輸入資料大小乘以批次參數，先花 10 個週期做倒數運算，由於在一整個批次標準化層中，N 都是固定的，故此方法不僅可用來替代計算平均所需的除法、減少原本除法器所需的面積，亦能在之後對每一個資料做標準化時，都只需要花 1 個週期就能完成，此方法可大量減少原本除法所需的計算時間。而在圖 2.5 架構計算完平均和標準差的倒數後，傳入圖 2.6 架構計算資料標準化以及縮放。我們先利用平均和標準差計算資料標準

化，再用先前就傳進來暫存的 γ 、 β 做縮放，最後傳至靜態隨機存取記憶體，完成前向傳播的運算。此階段本研究為了將週期時間壓低至 5ns 內，在此架構的輸入和輸出和中間訊號都用安插一個暫存器，藉此管線化（Pipeline）來壓低訊號計算時間。

由於在 FS2 的統計參數計算時間相較 FSB1 少很多（詳細時間分析會在後面說明），因此相較於現今一些研究將前向傳播和反向傳播運算分開設計電路[4]，本研究更於此階段先計算反向傳播所需的每一個輸入資料做標準化以及縮放(乘以 γ ，再加 β)的值，此方法不僅能增加核心運算陣列與批次標準化運算器之間的運算對稱性，更能有效消除反向傳播運算重新計算標準化及縮放所需的資料存取次數和運算時間。

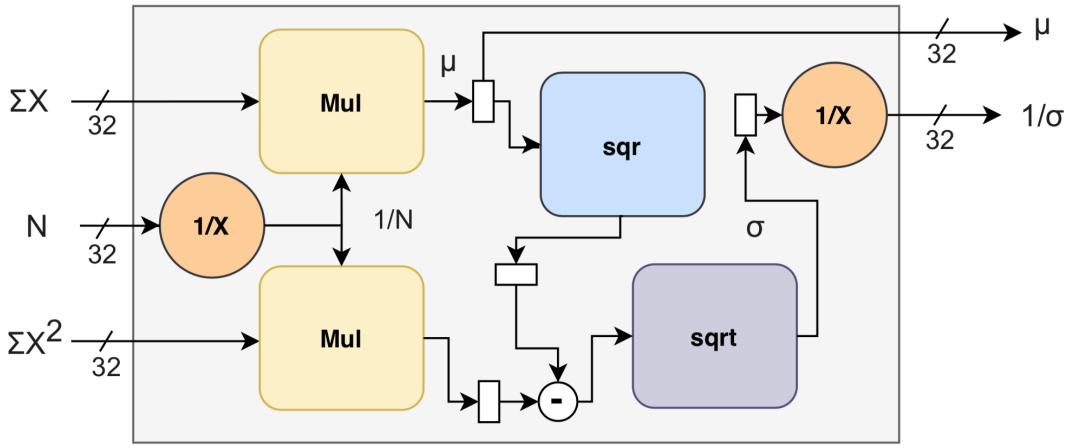

圖 2.5、Forward Stage 2 計算統計參數電路架構

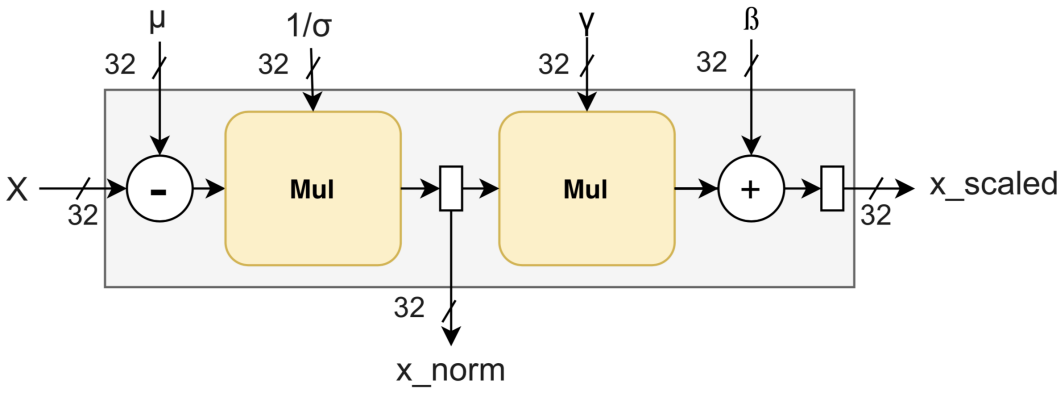

圖 2.6、Forward Stage 2 標準化及縮放電路架構

b. 反向傳播：

(i) Stage 1:

本研究利用前向傳播和反向傳播在 stage 1 的運算對稱性，設計出可共用的電路架構，相較現今一些研究將前向傳播和反向傳播運算分開設計電路[4]，本研究的做法可減少將近 50%的面積，因此在此階段之運算我們使用 FBS1 架構（如圖 2.4）。

在此階段，核心運算陣列會根據上一層輸入資料大小、輸出通道、批次不斷做卷積運算後輸入 FBS1 並累加存入 A_mean 和 A_x，其運算過程與前向傳播的運算相同，只是 A_x 的值不會輸出至靜態隨機存取記憶體，而是繼續做乘法運算再累加存入 B（此階段不會用到平方器）。與前向傳播運算相異的是，當累加完成後，在進入 BS2 之前我們會先將 A_mean 和 B 中的值與參數 N 的倒數花 10 個週期相乘後，才傳入 BS2，此舉之目的與 FS2 先計算倒數相同，可用來替代計算$\partial\beta$/N、$\partial\gamma$/N 所需的除法、減少原本除法器所需的面積，且之後不須再花 10 個週期對每個資料都除以 N。

(ii) Stage 2:

由於在 FS2 已先計算了標準化和縮放的資料，因此此階段僅需做簡單運算計算出輸入端的損失梯度值。

如圖 2.7，此架構由乘法器和減法器所組成，從靜態隨機存取記憶體取得計算過的∂y^((i))、標準化縮放資料和 FBS1 計算出的值做運算後得到輸入端的損失梯度值並傳至靜態隨機存取記憶體，完成反向傳播的運算。而此階段本研究也為了將週期時間壓低至 5ns 內，在此架構的輸入和輸出和中間訊號都用安插一個暫存器，藉此管線化來壓低訊號計算時間。

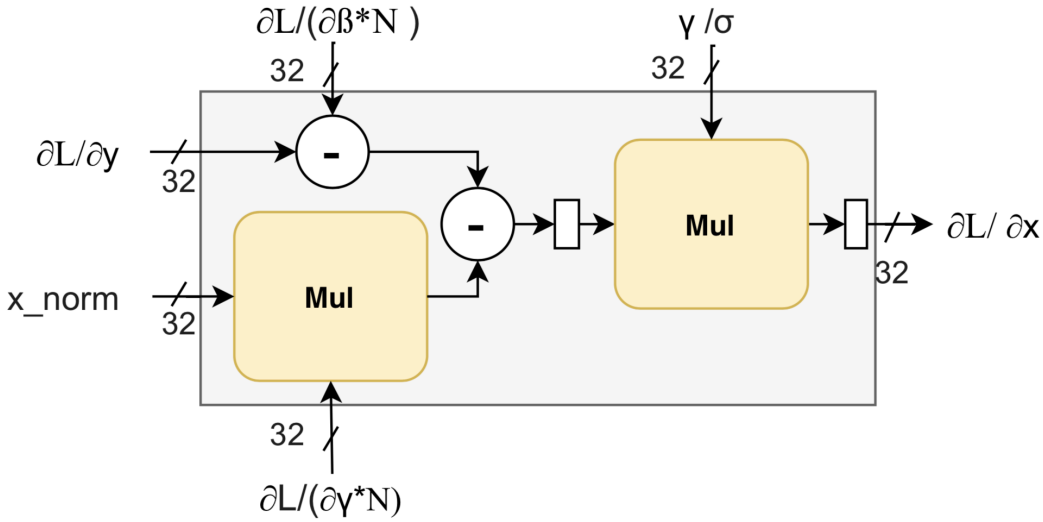

圖 2.7、Backward Stage 2 電路架構 Forward Stage 2 電路架構 1

三、 結果與討論(Result and Discussion)

1. 時間分析

在前向傳播的過程中，因為 4x4 的卷積運算單元陣列可以同時計算 4 個輸入通道的卷積運算值，所以在 Stage 1 所花費的週期數如式(12)所示：

$$\text{Batch size} \times \text{image size} \times \left[\frac{N_{input\ channel\ (pre-conv)}}{4}\right] \qquad \text{(式 12)}$$

而因為批次標準化電路要完成 4 個輸出通道的計算，所以在 Stage 2 所花費的週期數約如式(13)所示：

$$\text{Batch size} \times \text{image size} \times 4 \qquad \text{(式 13)}$$

根據式(12)與式(13)，可以推得在前向傳播的運算過程當中，當前一層的輸入通道數大於 16，則批次標準化電路將會需要等待卷積運算單元陣列；當前一層的輸入通道數小於 16，則卷積運算單元陣列需要等待批次標準化電路。

反向傳播的結果也是如此，Stage 1 所花費的週期數如式(14)所示：

$$\text{Batch size} \times \text{image size} \times \left[\frac{N_{output\ channel\ (post-conv)}}{4}\right] \qquad \text{(式 14)}$$

而 Stage 2 所花費的週期數同樣大約為式(15)所示：

$$\text{Batch size} \times \text{image size} \times 4 \qquad \text{(式 15)}$$

因此可以推得類似的結論，在反向傳播的運算過程當中，當後一層的輸出通道數大於 16，則批次標準化電路將會需要等待卷積運算單元陣列；當後一層的輸出通道數小於 16，則卷積運算單元陣列需要等待批次標準化電路。

而在大部分的卷積神經網路當中，輸入通道或輸出通道的數量都會大於 16，也就是說，此設計之下的批次標準化電路在大部分情況中，都會在卷積運算單元陣列完成運算前即計算完畢。因此可以在不增加原本卷積所需運算時間的條件下完成批次標準化。

2. 運算精準度

為了測試此電路的可行性，此研究進行了暫存器傳輸級(RTL)的電路模擬，測資是透過如下圖的 MNIST 手寫數字辨識的神經網路生成。如圖 3.1，此網路中有包含兩個批次標準化層，分別為 BN1 以及 BN2，兩者都個別接在兩個卷積層後面。

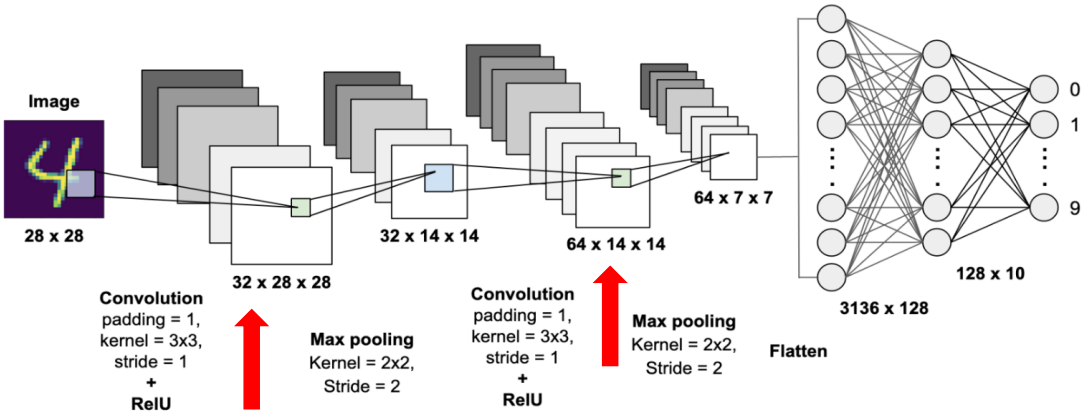

圖 3.1、MNIST 手寫數字辨識卷積神經網路

訓練的批次大小設定為 16。BN1 的通道數為 32、輸入資料尺寸為 28x28，BN2 的通道數為 16、輸入資料尺寸為 14x14。透過 python 進行此神經網路的訓練，並在第 3000 個迭代(iteration)時將兩層批次標準化層的輸入值、輸出值、γ及γ的更新值、β及β的更新值、此層輸出端的損失梯度值以及輸入端的損失梯度值寫出，並利用 verilog 進行驗證硬體運算結果與軟體運算結果是否相符。

表 3.1 與表 3.2 分別為兩個批次標準化層的運算結果與軟體結果相比之下的

L1 範數誤差：

BN1	L1-norm error
x	0.001226844
partial beta	7.721996e-07
partial gamma	1.4431578e-05
partial x	1.2775096e-08

表 3.1、BN1 的 L1 範數誤差

BN2	L1-norm error
x	0.0010662433
partial beta	3.1179752e-09
partial gamma	1.2771513e-05
partial x	4.134978e-09

表 3.2、BN2 的 L1 範數誤差

可觀察到 BN1 的誤差均略大於 BN2 的誤差，推論應該是因為在神經網路越後面的層，其值會越收斂(意即越靠近 0)，而 L1 範數誤差反映的是絕對誤差值，因此兩層誤差的差異可在 L1 範數誤差中觀察到。另一方面，為了確保這些低誤差不是因為該數值本身就很小，資料相對誤差也需要被討論。

x	Relative error
BN1	0.0028902171
BN2	0.07315532

表 3.3、x 的相對誤差

從表 3.3 可觀察到，BN1 的相對誤差(不到 0.3%)可以被接受，但 BN2 的相對誤差卻高達 7%。然而，如果去細看這些相對誤差極大的值，可發現剛好都是非常接近 0 的值。舉例來說，某一個資料點的正確答案為 1.1683e-05，而硬體的運算結果為-0.0007，這種情況將會造成相對誤差遽增，然而 L1 範數誤差

仍很小，但是這樣的情況可能是來自於數值利用 FP32 表達時本身的誤差，而不是乘法器或除法器等運算模組的精準度問題。所以綜合評估下來，此設計的運算結果已有非常高的精準度。

四、 結論

本研究提出的演算法與新的運算資料流，能夠維持運算精確度並有效降低深度卷積網路在訓練所需的時間。針對此運算資料流，本研究也設計出適用不同核心運算陣列的批次標準化運算處理器硬體電路，透過平衡兩個運算模組的計算時間，也能減少原本需等待核心運算陣列所浪費的時間，增加整體運算效率。

在未來，人工智慧技術估計將會如同近幾年繼續蓬勃發展，針對深度學習其龐大的計算量，從目前常見到的 CPU 與 GPU，改以使用特製硬體 ASIC 完成複雜的深度學習訓練任務是無可避免的；若有新的研究也旨在運用硬體加速訓練過程，本研究可提供其在卷積網路訓練中的批次標準化層之硬體電路概念，使深度學習簡化計算領域的方法更多元。

五、 參考文獻

[1] IOFFE, Sergey; SZEGEDY, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[2] GE, Jiexian, et al. BNReLU: Combine Batch Normalization and Rectified Linear Unit to Reduce Hardware Overhead. In: 2019 IEEE 13th International Conference on ASIC (ASICON). IEEE, 2019. p. 1-4.

[3] WAI, Yap June, et al. Fixed point implementation of tiny-yolo-v2 using opencl on fpga. International Journal of Advanced Computer Science and Applications, 2018, 9.10: 506-512.

[4] ZHIJIE, Yang, et al. Bactran: A Hardware Batch Normalization Implementation for CNN Training Engine. IEEE Embedded Systems Letters, 2020.

[5] WU, Shuang, et al. $ L1 $-Norm batch normalization for efficient training of deep neural networks. IEEE transactions on neural networks and learning systems, 2018, 30.7: 2043-2051.