深度學習

HW9

學號 : B103012002

姓名 : 林凡皓

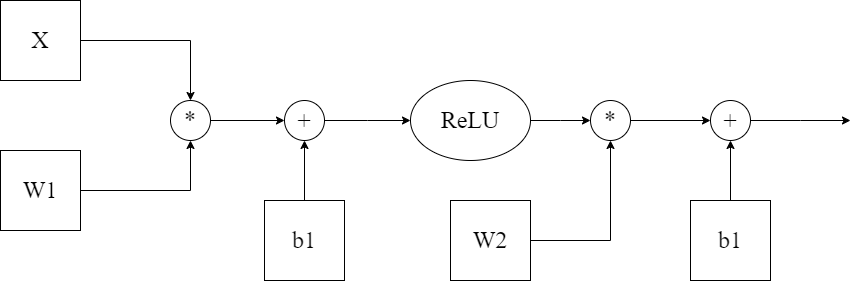
1. Barebones PyTorch

PyTorch提供high-level的API來讓使用者方便定義模型。這部分會從一個簡單的全連接ReLU網路架構開始，利用PyTorch tensor的操作來計算forward path，並用PyTorch autograd來計算梯度。

在創建PyTorch tensor時，如果將參數requires\_grad設定成True，則在計算與該tensor相關部分的時候PyTorch會建立一個computational graph來計算梯度，並將梯度存到x.grad中。

* Barebones PyTorch: Two-Layer Network

Two-Layer Network架構如下



根據此架構來完成forward path。

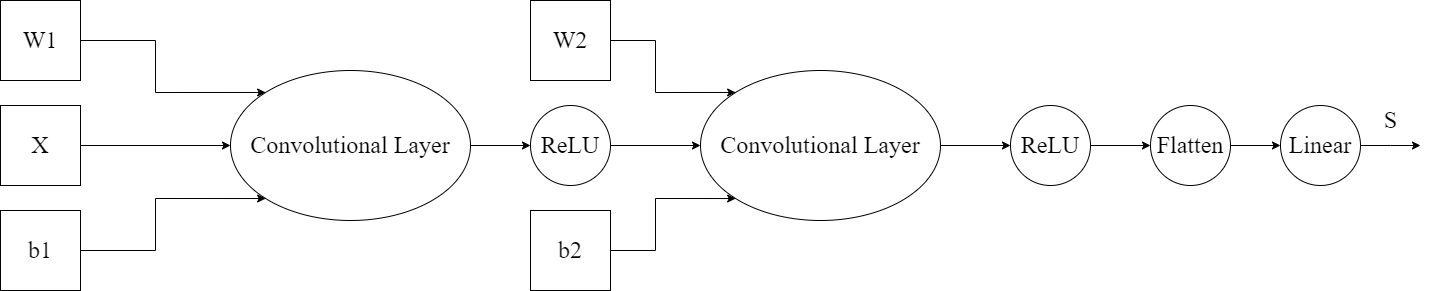
為了確認網路架構的正確性，我們將輸入與參數全部設定成0之後傳入網路中，並將輸出的形狀顯示出來，結果如下



此結果為正確結果，代表說網路可以正常運作。

* Barebones PyTorch: Three-Layer Network

Three-Layer Network架構如下



根據此架構完成forward path。

關於Convolutional layer，利用torch.nn.functional.conv2d來實作。

為了確認網路架構的正確性，我們將輸入與參數全部設定成0之後傳入網路中，並將輸出的形狀顯示出來，結果如下



* Barebones PyTorch: Kaiming Initialization

對於W的初始化，採用Kaiming initialization，我們可以直接利用PyTorch幫我們定義好的函數torch.nn.init.kaiming\_normal\_( )來完成。

對於b的初始化，我們一律將他初始化為0。

這邊嘗試利用nn.init.kaiming\_normal\_( )與nn.init.zeros\_( )來查看輸出結果，結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 黑色 的圖片

自動產生的描述

* Barebones PyTorch: Check Accuracy

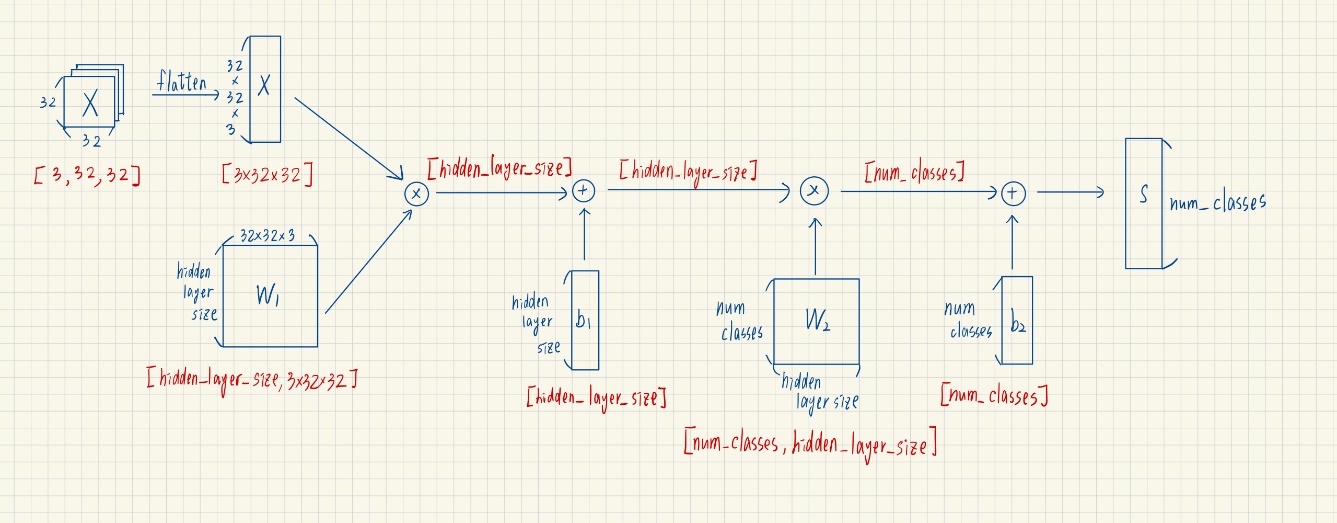
建立check\_accuracy\_part2( )，用來計算準確度。此函數會接收三個參數，分別為data loader、model、parameters。先利用data loader來讀取資料，在訓練階段利用validation set來做驗證，在測試階段使用test set做測試。接著根據model和parameters來完成模型，並利用此模型來計算準確度，最後將準確度回傳。

* Barebones PyTorch: Training Loop

定義train\_part2( )來做訓練。此函數會接收三個參數，分別為model、parameters、learning rate。此函數主要為一個for loop，先透過forward path預測出結果，並利用cross entropy來計算出loss。利用.backward( )自動幫忙計算梯度，然後根據梯度與learning rate來更新權重。完成一個epochs後，利用剛才定義好的check\_accuracy\_part2( )來協助計算validation accuracy。

* Barebones PyTorch: Train a Two-Layer Network

利用先前建立好的Two-Layer Network來訓練模型。訓練一開始，我們需要先初始化參數。W的部分利用torch.nn.init.kaiming\_normal\_( )來初始化，b則利用torch.nn.zeros( )來做初始化。關於各參數的形狀，參考以下推導



執行結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 筆跡 的圖片

自動產生的描述

可以看到在調整參數之前，準確度大約落在42 %左右。

* Barebones PyTorch: Train a ConvNet

這一部分要實作initialize\_three\_layer\_conv\_part2函數，這個函數主要是在對參數(W、b)做初始化。

對於W的初始化，採用Kaiming initialization，我們可以直接利用PyTorch幫我們定義好的函數torch.nn.init.kaiming\_normal\_( )來完成。

對於b的初始化，我們一律將他初始化為0。

關於torch.nn.init.kaiming.normal\_( )，他需要傳入一個tensor，這題我們傳入一個空的tensor。

由於第一層convolutional layer的參數有W1和b1，根據kernel\_size\_1和channel數量，W1的shape = (channel\_1, C, kernel\_size\_1, kernel\_size\_1)，中間會加上一個C的主要原因在於輸入X有rgb三個顏色。b1的shape = (channel\_1)。第二層convolutional layer的參數有W2和b2，根據前一層的shape，W2的shape = (channel\_2, channel\_1, kernel\_size\_2, kernel\_size\_2)，b2的shape = (channel\_2)。最後還要通過一層全連接層，全連接層的參數有fc\_w與fc\_b。由於有先經過flatten的關係，fc\_w的shape = (num\_classes, channel\_2\*H\*W)，而fc\_b的shape = (num\_classes)。

執行結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 功能表 的圖片

自動產生的描述

由結果可以看到，準確度最高可以到47%，比起前面訓練的Two-Layer Network，加入convolutional layer後準確度提升大約5%。

1. PyTorch Module API

前面採用barebone PyTorch時，我們需要自行追蹤參數的shape，這對於大型神經網路說是不可能的，因此PyTorch有提供nn.Module API來幫助我們自動追蹤所有參數。此外，在前兩次作中我們自行實作了許多optimizer，在PyTorch中有提供torch.optim來讓我們不用每一次都要自己寫optimizer。

* Module API: Two-Layer Network

建立一個Python繼承類別名為TwoLayerFC，該類別繼承torch.nn.Module的方法。

TwoLayerFC為一個兩層全連接層的神經網路。網路中參數W都是以kaiming\_normal做初始化，b則是初始化為0。

此外，雖然說backward path可以透過PyTorch中的autograd計算，但是forward path仍然需要自行定義，因此還需要寫一個函數來定義forward path。

呼叫TwoLayerFC的建構子並將建立好的模型顯示出來，結果如下

一張含有 文字, 字型, 黑色, 螢幕擷取畫面 的圖片

自動產生的描述

* Module API: Three-Layer ConvNet

定義一個ThreeLayerConvNet的繼承類別。

ThreeLayerConvNet為一個三層的神經網路，其中前兩層為convolutional layer，最後一層為全連接層。第一層convolutional layer channel size為channel\_1、filter size為5\*5、padding為2。第二層convolutional layer channel size為channel\_2，filter size為3\*3、padding為1。最後一層全連接層，input features為channel\_2\*H\*W(考慮到flatten的作用)、output features為num\_classes。

此外，跟TwoLayerNet相同的是，一樣要定義forward path。

呼叫TwoLayerNet的建構仔並將建立好的模型顯示出來，結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 黑色 的圖片

自動產生的描述

* Module API: Check Accuracy

定義check\_accuracy\_part34( )來幫助我們確認模型的準確度。此函數與在Barebones PyTorch時建立的check\_accuracy\_part2( )大致上相同，差別只在於這次不需要手動傳入參數。

* Module API: Training Loop

定義兩個函數，分別為adjust\_learning\_rate( )和train\_part345( )來幫助我們訓練模型。

adjust\_learning\_rate( )為調整learning rate的函數，他會根據使用者設定好的schedule來調整learning rate，調整幅度為參數lrd (learning rate decay)。

train\_part\_345( )為訓練模型的函數。此函數會利用到剛才定義的adjust\_learning\_rate( )來優化訓練過程。計算梯度時也是利用PyTorch的autograd來協助計算。與方才在Barebones PyTorch時定義的training\_loop不一樣的是，這次我們不手動更新權重，而是利用torch.optim來更新權重。

* Module API: Train a Two-Layer Network

利用剛才定義好的函數來訓練Two-Layer Network。這次訓練我們不再需要自行分配參數tensor，只需要傳入input size、hidden layer size、number of class即可。透過剛才定義好的TwoLayerNet類別來方便建立神經網路，並利用torch.optim來選取optimizer(這邊使用SGD)。訓練結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 功能表 的圖片

自動產生的描述

最好的準確度落在47.5%。

* Module API: Train a Three-Layer ConvNet

利用剛才定義好的函數來訓練Three-Layer ConvNet。

這邊會用到ThreeLayerConvNet類別中的方法，initialize\_three\_layer\_conv\_part3( )。

Initialize\_three\_layer\_conv\_part3( )為初始化模型的方法，該方法會建立一個Three-Layer ConvNet並將optimizer設定成SGD。訓練結果如下

一張含有 文字, 螢幕擷取畫面, 字型, 功能表 的圖片

自動產生的描述

可以看到，最佳的準確度為50.6%，比起Two-Layer Network高了3.1%。

1. PyTorch Sequential API

透過PyTorch Module API的協助，建立一個神經網路比起之前一個函數一個函數慢慢寫方便許多，但是我們仍然會需要去定義一個類別，並將建構子與forward path完成。

PyTorch提供一個module叫做torch.nn.Sequential，此module將剛才定義一個類別、建構子、forward path這幾個步驟合併成一個。雖然說torch.nn.Sequential很方便，但是他只提供一些基本功能，更加複雜的功能仍然需要透過PyTorch Module API來完成。

* Sequential API: Two-Layer Network

一開始先利用nn.Sequential將模型定義好。

nn.Sequential可以接收一個ordered dictionary，dictionary中從放layer的名字與layer的種類。nn.Sequential會將ordered dictionary中的layer照著順率串接起來，成為一個神經網路。

定義好模型之後，利用torch.optim來選擇optimizer，這邊使用SGD。

利用剛才定義好的train\_part345( )來訓練模型，結果如下

一張含有 文字, 功能表, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

最佳的準確度為47.4 %。

* Sequential API: Three-Layer ConvNet

步驟基本上與Two-Layer Network相同，一開始先利用nn.Sequential將模型定義好，接著利用torch.optim來選擇optimizer，這邊選擇使用SGD。訓練結果如下

一張含有 文字, 字型, 螢幕擷取畫面, 功能表 的圖片

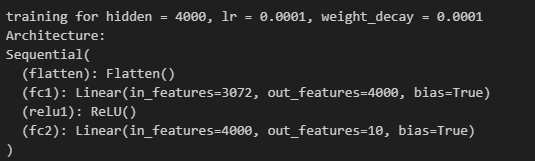
自動產生的描述

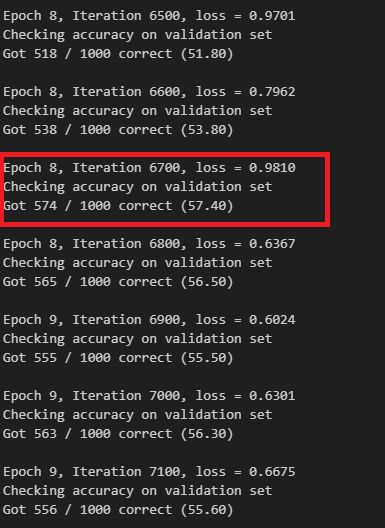
最好的準確度為54.6 %。

1. 額外嘗試

* 嘗試重新訓練Two-Layer Network

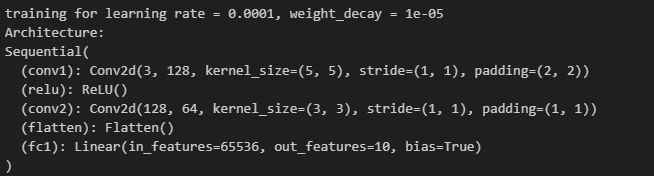
剛才訓練的Two-Layer Network都是未經調整參數訓練出來的結果，我嘗試迭代多組參數來尋找最佳結果。首先我將optimizer更改為adam，並將訓練epochs提升到10。超參數設置部分，hidden layer size我嘗試[4000, 5000, 6000]、learning rate [1e-1, 1e-2, 1e-3, 1e-4]、weight decay [1e-3, 1e-4, 1e-5]。訓練後最佳結果發生在hidden layer size = 4000、learning rate = 0.0001、weight\_decay = 0.0001時，準確度為57.4 %，結果如下圖



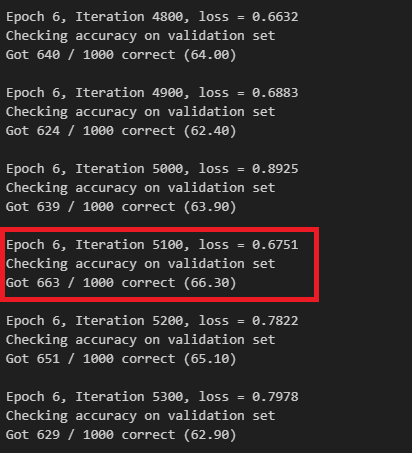


* 嘗試重新訓練Three-Layer ConvNet

剛才訓練的Two-Layer Network都是未經調整參數訓練出來的結果，我嘗試迭代多組參數來尋找最佳結果。首先，我將optimizer調整為adam，我嘗試使用不同channel size、learning rate、weight decay，並將epochs設為10來做做訓練。訓練後，最佳結果發生在learning rate = 0.0001、weight decay = 1e-5時，神經網路架構如下



訓練結果如下



最佳的準確度來到66.3 %。

比較Two-Layer Network與Three-Layer ConvNet (調整參數後結果)，Three-Layer Network的準確度比Two-Layer Network高出11.4 %，比起調整參數前的差距，可以說是差很多。

* 參數初始化

這次作業中使用到kaimimg initialization，以下為一些initialization的說明。

1. Weight初始化為0 :

如果初始化為0，forward path計算結果都會是0。要注意的是只是輸出結果為0，模型仍然會根據back propagation做學習。

1. Random initialization :

隨機給予權重初始值。Random initialization的問題在於會根據高斯分布的變異數，造成神經網路中layer output的消失或是爆炸。數學推導如下

一張含有 文字, 字型, 收據, 螢幕擷取畫面 的圖片

自動產生的描述

可以看到當初始化權重的變異數過大，會造成layer output的變異數也過大；當初始化權重變異數很小，layer output的變異數也會很小。

1. Xavier initialization :

在PyTorch中，可以透過nn.init.xavier\_normal\_來做Xavier initialization。Xavier的主要概念在保持輸入與輸出的變異數一致。

推導如下 (d為輸入節點數量)

一張含有 文字, 字型, 白色, 代數 的圖片

自動產生的描述

假設xi與wi的平均數都是0，且idd，則

一張含有 文字, 字型, 白色, 筆跡 的圖片

自動產生的描述

如果希望Var(y) = 1，則



一張含有 字型, 行, 文字, 圖表 的圖片

自動產生的描述

利用相同方式將backward path的部分也推導一次，可以得到

一張含有 文字, 字型, 行, 白色 的圖片

自動產生的描述

為了要讓forward path與backward path的變異數相同，因此取

一張含有 文字, 字型, 行, 印刷術 的圖片

自動產生的描述

假設初始化權重的分布為Uniform~(a, b)且a = -b，計算其變異數要等於剛才推導結果

一張含有 文字, 收據, 圖表, 字型 的圖片

自動產生的描述

最後可以得到初始化權重的分布。

1. He initialization (Kaiming initialization) :

在PyTorch中，可以透過nn.init.kaiming\_normal\_來呼叫。

剛剛的Xavier initialization的問題在於使用ReLU作為activation function時，輸入輸出的變異數會有所不同。He initialization解決了這個問題，公式推導如下



由於yl - 1是對稱0的分布，再加上使用ReLU，所以

一張含有 文字, 字型, 筆跡, 書法 的圖片

自動產生的描述

照這樣的推理，將公式推導至第一層

一張含有 文字, 字型, 白色, 行 的圖片

自動產生的描述

可以看到Var(yl)為Var(wl)作連乘，為了避免變異數的改變

一張含有 字型, 文字, 白色, 行 的圖片

自動產生的描述

一樣去假設權重的分布為Uniform~(a, b)，則根據權重之變異數

一張含有 文字, 筆跡, 字型, 圖表 的圖片

自動產生的描述

最後推導出初始化權重之分布。

1. Batch Normalization

Batch normalization在訓練過程中對layer output做normalization，這樣一來就可以不用考慮如何對參數做初始化

1. 如何選擇這些方法

一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述

1. Reference

[1] Tommy Haung “深度學習:Weight initialization和Batch Normalization” <https://chih-sheng-huang821.medium.com/%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-weight-initialization%E5%92%8Cbatch-normalization-f264c4be37f5>

[2] Huili Yu “Weight Initializaiton for Deep Neural Network” <https://medium.com/@freshtechyy/weight-initialization-for-deep-neural-network-e0302b6f5bf3>

[3] Wanderer001 “一文稿懂深度網路初始化 (Xavier and Kaiming Initialization” <https://blog.csdn.net/weixin_36670529/article/details/104336598>

[4] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. <https://chat.openai.com/>