1. Barebones PyTorch: Three-Layer ConvNet
2. three\_layer\_convnet
   * 在此區塊使用了torch.nn.functional內已經被implement的各式neuro network函數(像是Linear、convolution等)來組合成我們要的model。這邊是透過Convolution-> ReLu -> Convolution -> ReLu -> flatten(將Convolution產生的output轉換成linear可以接受的input) -> Linear。最後產生的score就是各類別的分數。
   * A screenshot of a computer

     Description automatically generated with medium confidence在這個區塊我們也必須要自己定義weight及bias的大小，並且以他們為nn.functional函式的input。

Text

Description automatically generated

1. Kaiming Initialization
   * 在這個部分透過Kaiming Initailization的方法去初始化參數值，這邊不用像前面一樣自己去計算每個參數的值。Torch.nn.init裡面提供了多種initialization的方法，這邊會根據batch size, input channel, output channel, kernel size去建立各個layer的weight及 bias tensor，將weight透過nn.init.kaiming\_normal\_()來初始化，bias則用nn.init.zeros\_()來初始化為0。
   * 最後每個參數都是一個tensor，將這些參數(tensor)裡面的requires\_grad設為True。其目的是當運算包含到此tensor時，會去建立一個computational graph，使得未來能夠進行back propagation的運算。

Text

Description automatically generated

1. Training Loop
   * 在training 時可以直接使用前面所建立的model，model\_fn就是上面自訂的model，params是每個Layer的weight及Bias。
   * 使用model\_fn(x, params)這個function來執行forward的動作。
   * 透過nn.functional.cross\_entropy來計算scores和標籤值y的entropy值，也就是loss。
   * 透過loss.backward()來計算每個參數內grad的值。最後需要針對每個parameters一一的去更新，也就是減去learning\_rate \* grad。
   * Text

     Description automatically generatedTorch.no\_grad()代表以下的運算都不要建立進computational graph。grad.zero\_()則是將每個參數的grad都設為0，才不會影響下次的backprop。

­­

1. PyTorch Module API
2. ThreeLayerConvNet
   * 此Class繼承了nn.Module，且在此區塊nn.Conv2d是一個class，前面nn.functional.conv2d是一個function，他們2個的運作方式不同。
   * 使用nn.Module，會在class的\_\_init\_\_裡面去建立我們需要使用到的layer並且去初始化所有的參數。這邊不需要自己去建立參數的tensor，他會自己包含在nn.Conv2d裡面。
   * Text

     Description automatically generated在forward的部分，就依照model的需求去呼叫在\_\_init\_\_設置的layer或一些輔助的activation function
3. Train a Three-Layer ConvNet
   * 首先，只需要透過對設計好的model class建立一個instance ，其中C代表input channel，channel\_1及channel\_2代表此model第1、2層的hidden layer， num\_classes為linear layer輸出大小。
   * 在opimizer的部分可以使用toch.optim內建的多種optimizer，不需要自己手動去更新。
   * 從training的部分可以看到只要呼叫model(x)就會透過\_\_call\_\_()呼叫forward產生score。這邊一樣透過cross\_entropy來產生loss。
   * A screenshot of a computer

     Description automatically generated with medium confidenceText

     Description automatically generatedText

     Description automatically generated在update gradient時，這邊先透過optimizer.zero\_grad()先將每個參數的grad值設為0。接下來，透過loss.backward計算每個參數的grad。最後，直接透過optimizer.step()去更新參數的值。

Text

Description automatically generated

1. PyTorch Sequential API
2. Use sequential API

* 在一些較簡單的Model可以透過Sequential API來建立model，透過這樣的方式就不需要先建立一個class。
* 這個方式只需要透過nn.Sequential()這個class，將model的每個layer依序放進nn.Sequential的參數就會自動建立model所需的所有東西。其他地方則跟Module API一樣。

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

1. ResNet for CIFAR-10
2. PlainBlock

* ResNet會有多層相同的Block也就是PlainBlock，它的組成為BatchNorm->ReLu->Conv->BatchNorm->ReLu->Conv
* Text

  Description automatically generated其中在第一個Convolution layer會根據設定來決定是否要downsampe，在ResNet是使用stride而不是Pooling。

1. ResidualBlock

* ResNet除了上面的PlainBlock外，它會在加入原本的input避免vanishing gradient。
* Text

  Description automatically generatedself.shortcut需要用if來判斷主要是因為input的size可能會與output的size不同。如果PlainBlock有做downsample，則input一樣也做downsample; 如果單純是input和output channel不一樣的話就使用1\*1的filter去產生一樣的size。

1. Residual Stem

* Text

  Description automatically generated此部分用在model的一開始，用來增加convolution channel的數量

1. Residual stage

* 用來一次建立多個residual block

Text

Description automatically generated

1. ResNet

* 主要的model，用前面的components來建立。
* 首先在\_\_init\_\_建立model，第一個block使用ResNetStem來提升input channel。
* 接下來，透過ResNetStage建立多個ResNetBlock
* 最後透過nn.Sequential來將這些block組合起來並指派給self.cnn
* 由於self.cnn的output為(batch size, output channel, H, W)，因此在forward的地方透過average pooing的方式將每個channel上的值給平均，並且將每個batch flatten。
* 在linear layer就可以使用最後一個ResNetBlock的output channel當作input，output則為num\_class。

Text

Description automatically generated

1. ResNet (Plain Block vs Residual Block)

* 透過前面所建立的ResNet來測試Residual Block和PlainBlock的準確度。
* 從圖中可以看到Residual Block的準確度明顯高於PlainBlock。

Chart, line chart

Description automatically generated

1. Residual bottleneck block

* 透過將Block中間的channel減少使其增加效率。透過這樣的方式能夠讓我們deploy在性能較差的設備上。
* 其程式碼跟Residual Block幾乎一樣，但他多了一層且第一層的output channel是原本output channel的1/4，第2層也都是1/4，最後一層才轉回output channel。
* Text

  Description automatically generated透過測試Residual bottleneck block所產生的ResNet( resnet47 )，其效能可以從圖中得知，它比使用Residual block ( resnet32 ) 的稍微好一點。

Chart, line chart

Description automatically generated