深度學習

HW8

學號 : B103012002

姓名 : 林凡皓

接續上一次作業，將後半部分 (Update rules) 完成。本次作業內容從七、update rules開始。

1. Linear layer

* Linear.forward :

1. 解題思路 : 先利用view( )將輸入x變成shape為(N, D)的tensor。接著透過forward propagation的公式來計算forward propagation的輸出結果。
2. 執行結果 :



由此結果可以看出，計算結果與預期結果相差很小，代表說這個function功能正確。

1. 額外討論 :

Forward propagation的公式為。之所以是xW是因為考慮到tensor的形狀。out、x、W的形狀分別為(N, M)、(N, D)、(D, M)，因此透過xW的形狀為(N, M)滿足out的形狀可以得知，矩陣乘法順序應該為xW。

* Linear.backward :

1. 解題思路 : 透過computational graph來幫助我們計算微分，computational graph如下

一張含有 文字, 字型, 行, 圖表 的圖片

自動產生的描述

根據上圖推倒結果，完成dx、dw、db即可。

1. 執行結果 :

一張含有 文字, 字型, 螢幕擷取畫面, 印刷術 的圖片

自動產生的描述

由上面結果可以看出，跟預期結果相比，誤差很小，代表說此function功能正確。

1. 額外討論 :

透過computational graph計算出來的dx為，這其實是reshape後的dx，並不是我們要的。因此計算出此結果後，還需要透過reshape將形狀變成 (N, d1, …, d\_k)。

1. ReLU activation

* ReLU.forward :

1. 解題思路 :

透過torch.relu( )即可完成此題。

1. 執行結果 :



由上面結果可以看出，跟預期結果相比，誤差很小，代表說此function功能正確。

* ReLU.backward :

1. 解題思路 :

由可以知道，對此函數做微分的結果為，當x > 0，微分結果會是upstream derivatives dout，而當x<=0，微分結果為0。因此我先將dx令成dout，再透過index將x<=0的部分改成0。

1. 執行結果 :



由上面結果可以看出，跟預期結果相比，誤差很小，代表說此function功能正確。

1. “Sandwich” layers

在神經網路中，通常會出現一些常見的層模式，像是linear layer後面會接上ReLU。為了定義這些層模式，我們可以定義一個convenience layer，也就是說定義一個layer為linear layer和ReLU的組合。透過這種抽象化的概念，我們可以更方便的去管理code架構。接下來就是將剛剛實現的Linear和ReLU做結合，變成一個叫做Linear\_ReLU的convenience layer。

* Linear\_ReLU.forward :

1. 解題思路 : 將input的x、w、b送進剛剛定義好的Linear.forward。接著再將經過Linear.forward的輸出送進ReLU.forward即可。要注意的是，我們需要將過程中的參數記錄下來，以便在backward path中使用。

* Linear\_ReLU.backward :

1. 解題思路 : 先接收從Linear\_ReLU.forward的過程中存下來的參數，接著分別將這些參數送進先前定義好的ReLU.backward和Linear.backward。由於是back propagate，因此要先經過ReLU.backward再經過Linear.backward。
2. 執行結果 :

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

由上面結果可以看出，跟預期結果相比，誤差很小，代表說此function功能正確。

1. Two-layer network

* \_\_init\_\_ :

1. 解題思路 : 這個為TwoLayerNet物件的初始化。由於是兩層的神經網路，因此會有四個參數要做初始化，分別為W1、W2、b1、b2(他們都是tensor)。W1和W2我選擇利用randn(normal distribution)的方式做隨機的初始化，b1、b2則是初始化為0。
2. 額外討論 :

為甚麼可以直接將b初始化為0但是W卻不行，甚至還要特別使用weight\_scale來控制初始化大小?

b可以初始化為0是因為它其實只是在做平移，並不需要做過多的變化或是隨機性。但是W就不一樣了。W會直接的影像到神經網路的收斂性，在做gradient descent的時候，W設置不好可能會造成所謂的梯度消失或是梯度爆炸，因此通常在做W的初始化都會採用一些隨機初始化的方法。

* TwoLayerNet.loss :

1. 解題思路 :

一張含有 文字, 行, 字型, 圖表 的圖片

自動產生的描述

根據上面的推導的結果以及先前定義好的function來完成。在計算dw的時候，由於先前並沒有加上regularization的影響，因此經過Linear.backward和Linear\_ReLU.backward後還需要加上的部分。

1. 執行結果 :

一張含有 文字, 螢幕擷取畫面, 字型, 黑與白 的圖片

自動產生的描述

可以看出不管有沒有考慮regularization，計算出來的結果誤差都很小，代表說功能正確。

1. Solver

Solver為一個將訓練分類器所需要的東西都封裝起來的類別。要使用Solver，我們需要先創建一個Solver類別，並將dataset、learning rate、batch size……等參數輸入進去。接著可以透過呼叫train方法來訓練模型。訓練完成後會將參數存放到model.params中，訓練過程的loss、training accuracy、validation accuracy會分別存放到solver.loss\_history、solver.train\_acc\_history和solver.val\_acc\_history中。

* \_\_init\_\_ :

初始化權重的想法和TwoLayerNet相同。不一樣的是這次要針對多層的神經網路做初始化。對於每一層來說，W的形狀為(dim\_of\_last\_layer, dim\_of\_current\_layer)，而b的形狀為(dim\_of\_current\_layer, )。

* loss :

loss的部分分為forward path和backward path。

Forward path的部分就是透過for loop不斷的將計算出來的東西送進Linear\_ReLU.forward，除了最後一層是送入Linear.forward。與先前不同的是，這一次還要考慮使用Dropout layer的情況。Dropout的部分會使用到Dropout.forward，因此先說明Dropout.forward的實現。

* Dropout.forward : 利用torch.rand\_like(x)生成與x相同形狀的tensor，並對這個tensor使用mask，讓大於dropout機率的位置設成1，其餘設成0。

接著是backward path的部分。Backward path的主要實現邏輯與forward path相同，都是透過for loop來進行多次的propagation，差別在於backward path是不斷的經過Linear\_RuLU.backward。由於也要多考慮到Dropout的部分，因此這邊需要多實現Dropout.backward。

* Dropout.backward : Dropout.backward的主要概念就是將在forward path過程中，被dropout的部分設成0，其餘為1。
* create\_solver\_instance :

創建一個Solver物件並將要設定的參數輸入進去。這邊嘗試使用的參數如下 : hidden\_dim = 200、update\_rule = sgd、learning\_rate = 0.4、lr\_decay = 0.96、batch\_size = 512、num\_epochs = 15。

* 訓練結果 :

使用create\_solver\_instance所設置的參數後，訓練結果如下

一張含有 文字, 螢幕擷取畫面, 功能表, 黑與白 的圖片

自動產生的描述

最佳的準確度為52.69 %。

將結果視覺化如下

一張含有 文字, 螢幕擷取畫面, 繪圖, 行 的圖片

自動產生的描述

1. Multilayer network

* get\_three\_layer\_network\_params :

採用三層的神經網路對50筆資料訓練。這次可以調整的參數為weight\_scale和learning\_rate。先將weight\_scale設成1，learning rate設成1e-2。訓練結果如下

一張含有 文字, 功能表, 螢幕擷取畫面, 圖書 的圖片

自動產生的描述

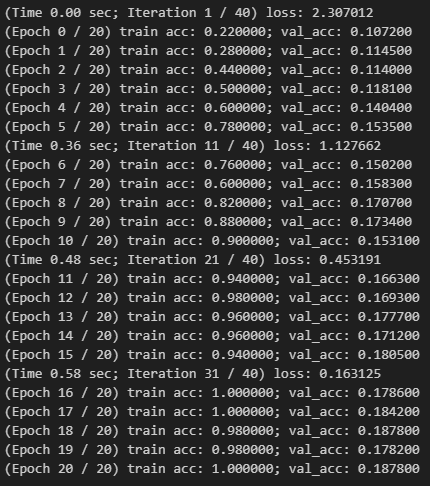
一張含有 文字, 螢幕擷取畫面, 陳列, 數字 的圖片

自動產生的描述

可以看出在20epochs內，training accuracy就可以來到100%，overfitting很嚴重。

* get\_five\_layer\_network\_params :

採用五層的神經網路對50筆資料訓練。這次可以調整的參數為weight\_scale和learning\_rate。先將weight\_scale設成1，learning rate設成0.1。訓練結果如下



一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述

可以看出在20epochs內，training accuracy就可以來到100%，overfitting很嚴重。

1. Update rules

* sgd + momentum :

1. 解題思路 : 根據Nesterov Momentum的公式，如下圖，來完成此函數。

一張含有 文字, 字型, 行, 螢幕擷取畫面 的圖片

自動產生的描述

1. 測試結果 :



1. 比較sgd與sgd + momentum :

分別透過sgd與sgd + momentum來訓練一個六層的神經網路。執行結果如下 :

一張含有 文字, 螢幕擷取畫面, 字型, 黑與白 的圖片

自動產生的描述

一張含有 文字, 行, 螢幕擷取畫面, 圖表 的圖片

自動產生的描述

由結果可以很明顯地看到，sgd + momentum 的收斂速度快很多。

* RMSProp :

1. 解題思路 : 根據RMSProp的公式來完成此題。有關RMSProp更加詳細說明，請參考八、額外嘗試中的optimizer部分。

一張含有 文字, 字型, 筆跡, 白色 的圖片

自動產生的描述

一張含有 文字, 字型, 筆跡, 行 的圖片

自動產生的描述

1. 執行結果 :



* Adam :

1. 解題思路 : 根據Adam的公式來完成本題。有關Adam更加詳細說明，請參考八、額外嘗試中的optimizer部分。

一張含有 字型, 文字, 筆跡, 書法 的圖片

自動產生的描述

一張含有 字型, 圖表, 白色, 數字 的圖片

自動產生的描述

一張含有 字型, 白色, 文字, 行 的圖片

自動產生的描述

1. 執行結果 :

一張含有 文字, 字型, 螢幕擷取畫面 的圖片

自動產生的描述

* Optimizer的比較 :

比較sgd、sgd\_momentum、adam與rmsprop對於訓練過程的影響，結果如下圖

一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述

由結果可以看出，在引入momentum的概念後，收斂速度有大幅提升。在加入隨梯度大小調整learning rate的概念後(rmsprop、adam)，收斂速度會再次提升，此外，也可以注意到除了速度上的提升，loss與accuracy都可以優化的更好，超越沒有調整learning rate時的表現。

1. Dropout

Dropout會在forward path中隨機將部分神經元的輸出結果設成0，以防止overfitting的發生。

* Dropout : forward

1. 解題思路 :

dropout在訓練階段與測試階段的表現有所不同，因此需要根據這兩種模式寫入不一樣的方法。

在訓練階段，可以透過遮罩的方式。生成一個隨機tensor (tensor中的數值需要在[0, 1)中)，將tensor中大於p的值設成1，其餘為0。在讓輸入經過這個mask即為輸出。

在測試階段，由於不需要執行dropout的動作，因此只需要將輸入回傳即可。

1. 執行結果 :

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

由上面結果可以看出，fraction of train-time output set to zero的數值會根據dropout機率p的設置而改變，且兩者的數值會幾乎相等，代表說這個函數的公正確。

* Dropout : backward

1. 解題思路 :

Backward path和forward path一樣會在訓練與測試時有不一樣的表現。在測試時和forward path相同，之需要將dx設成dout即可。

訓練的部分要根據下圖推導的結果來完成。

一張含有 文字, 筆跡, 字型, 數字 的圖片

自動產生的描述

1. 執行結果 :



1. Full-connected nets with dropout

* 在FullyConnectedNet中加入dropout功能

先前執行FullyConnectedNet都沒有使用到dropout，現在試著使用dropout來確認dropout可以正常運作。執行結果如下

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

由結果可以看到，對於不同dropout機率，各個參數的誤差都非常小，且會根據不同的dropout機率做改變，代表說dropout的運作會根據dropout機率變動而調整，且整體功能正確。

* Regularization experiment

對不同的神經網路做訓練，來查看dropout是否真的有regularization的功能。這邊對三個神經網路做訓練，網路架構分別為

1. Hidden size = 256，dropout = 0
2. Hidden size = 512，dropout = 0
3. Hidden size = 512，dropout = 0.5

訓練結果如下

一張含有 文字, 螢幕擷取畫面, 黑與白, 功能表 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 黑與白, 字型 的圖片

自動產生的描述

由上面結果可以看到幾個現象 :

1. Hidden size的影響 :

比較hidden size = 256與hidden size = 512結果可以發現，hidden size = 512的準確度會高於hidden size = 256，代表說較大的hidden size可以做到更多的非線性來更好的擬和訓練資料，使準確度提升。

1. Dropout 的影響 :

比較hidden size = 512以及相同hidden size但是dropout = 0.5的模型，會發現到對於訓練資料來說，dropout = 0會有比較高的準確度，但是在validation set上，反而是dropout = 0.5的模型有比較高的準確度。會造成這樣的現象是因為hidden size = 512、dropout = 0的模型發生嚴重的overfitting，導致說在validation上的表現不如預期。而在加入dropout後，overfitting的程度被稍微改善了，因此在validation上的表現稍微提升了一些。由這次嘗試也可以看到，dropout的功能就如同regularization，可以防止overfitting的發生。

將訓練過程視覺化後結果如下

一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述

由視覺化後結果可以更明顯看出，在training accuracy上，加入dropout表現最差，因為要防止overfitting所以對訓練資料的擬和程度不會像沒有dropout一樣高。但是在validation accuracy上，有dropout的模型表現最好，這便是regularization產生的效果。

1. 額外嘗試

* 嘗試重新訓練模型 :

在regularization experiment的地方，使用三種不同模型做訓練，最終最佳的準確度落在49 %，我嘗試重新調整參數，試圖提升準確度。

這次我嘗試調整hidden size和dropout機率。Optimizer使用adam，learning rate = 5e-3，並對40000張訓練資料做訓練。從訓練過程，我觀察到幾個現象

1. Dropout機率對訓練的影響

跟先前說明的一樣，dropout確實有regularization的作用，剛才只有測試dropout = 0.5的部分，因此我嘗試加大dropout機率，結果如下

一張含有 文字, 螢幕擷取畫面, 黑與白, 字型 的圖片

自動產生的描述

由上圖可以看到，在一樣是hidden size = 512的情況下，持續加大dropout機率到0.7之後，training accuracy來到只有73.9 %，但是validation卻可以到50 %左右，代表說加大dropout後overfitting的現象又再更進一步改善了。

Dropout這麼大確實可以讓training accuracy和validation accuracy更加相近，但是這樣的結果就會有最高的validation accuracy嗎?經過我的測試後，答案是否。我嘗試使用dropout = 0.3對相同的網路架構做訓練，結果如下

一張含有 文字, 螢幕擷取畫面, 黑與白, 字型 的圖片

自動產生的描述

由結果可以發現到，降低dropout機率後，validation accuracy可以來到52.3 %，比dropout機率為0.7時來的高。經由這次實驗可以得知，dropout其實也是一種超參數，需要經過不斷嘗試來找尋最適合的數值。

1. 訓練最佳結果 :

我嘗試調整的hidden size為128、256、512、1024，dropout機率為0、0.3、0.5、0.7、0.9，經過多次訓練，最佳結果如下

一張含有 文字, 螢幕擷取畫面, 黑與白, 字型 的圖片

自動產生的描述

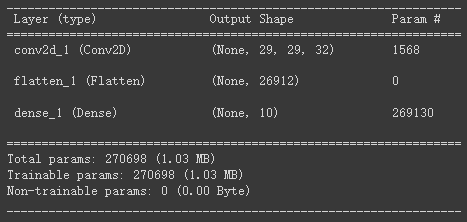
最佳結果發生在hidden size = 1024，dropout機率為0.5。醉雞的準確度為52.91%。

* 嘗試使用一層CNN + 一層全連接層 :

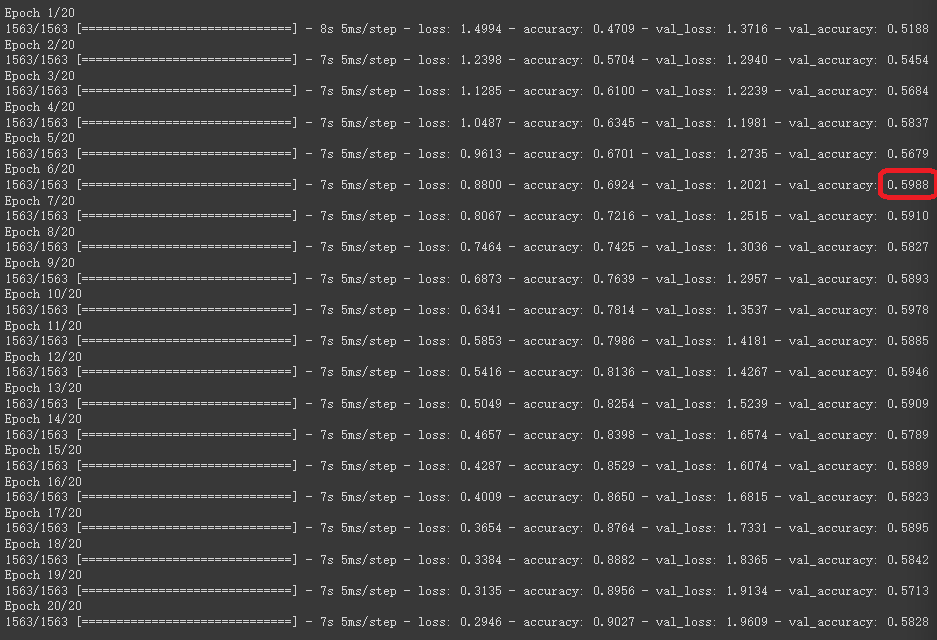
由於教授上課已經交到CNN的部分，因此我嘗試利用tensorflow來建立一個CNN，並與這次作業中的的模型做比較。

1. CNN架構 :

這次嘗試建構的CNN架構為一層卷積層加上一層全連接層。詳細架構如下圖



1. 訓練結果 :



由結果可以看出，最佳的準確度為59.88 %，將近有六成的準確度。

1. 結果討論 :

由訓練結果可以看出，CNN的準確度比一層全連接層的準確度高上許多，此外，教授上課時有提到，使用卷積層的話通常不會加上dropout layer，因為卷積層本商就具有對overfitting的抵抗力，經過這次嘗試也證明了教授所言。在我自己建立的CNN架構中，沒有加入dropout layer，但是最後training accuracy只有90.27%，而validation accuracy卻來到58.28%。與全連接層加上dropout的結果進行比較，overfitting的現象甚至還沒那麼嚴重。因此我認為在實際應用中，dropout並非那麼實用，與其使用dropout加上全連接層，不如直接使用卷積層。

* Optimizer :

由於這次作業有做到sgd + momentum的實現，因此我希望可以多加了解一下optimizer的原理與種類。

1. Gradient Descent : gradient descent是一種基於凸函數的優化演算法。他透過沿著負gradient方向來降低loss。

一張含有 文字, 字型, 筆跡, 白色 的圖片

自動產生的描述

* 優點 : 很好理解，實做起來簡單。
* 缺點 : 由於他一次要計算所有訓練資料的梯度並一次更新所有參數，因此計算速度很慢，而且需要花費很多記憶體空間。

1. Stochastic Gradient Descent (SGD) : 主要概念與gradient descent相同，差別在於他一次只更新一個參數。

一張含有 圓形, 寫生, 字型, 筆跡 的圖片

自動產生的描述

* 優點 : 他可以對大型的資料做計算，且需要比較少的記憶體空間。
* 缺點 : 由於SGD更新參數時只有使用一個樣本，因此可能有noise，使模型更新不穩定。此外，計算成本也比較高。

1. Mini-Batch Gradient Descent : 將訓練資料分成batches並對這些batches做參數更新。

一張含有 圓形, 行, 圖表, 設計 的圖片

自動產生的描述

* 優點 : 收斂過程會更佳穩定，計算複雜度也比較低。
* 缺點 : 容易受到learning rate的影響。

1. SGD with Momentum : 在SGD的基礎上加上momentum。Momentum使參數在更新一定程度的保留前一次更新時的方向，並利用當前的梯度對該方向做調整。

一張含有 文字, 字型, 筆跡, 白色 的圖片

自動產生的描述

一張含有 圓形, 寫生, 設計 的圖片

自動產生的描述

* 優點 : momentum可以幫助減少noise，而且exponential weighted average可以讓曲線更平滑。
* 缺點 : 需要額外的超參數。

1. AdaGrad (Adaptive Gradient Descent ) : 先前討論的所有方法中，learning rate都是固定的。AdaGrad使我們可以對不同的神經元、迭代次數使用不一樣的learning rate。

一張含有 文字, 字型, 筆跡, 行 的圖片

自動產生的描述

* 優點 : learning rate可以根據迭代次數做調整。對於稀疏的資料也可以有不錯的表現。
* 缺點 : 對於深層的神經網路來說，learning rate會變得很小，導致一些神經元權重更新很慢，也就是所謂的dead neuron problem。

1. RMS-Prop (Root Mean Square Propagaion) : 為一種特殊版本的AdaGrad。Learning rate為梯度的exponential average。通常會結合momentum一起使用。

一張含有 文字, 字型, 筆跡, 白色 的圖片

自動產生的描述

* 優點 : learning rate可以自動更新，而且對於每一個參數都可以有不一樣的learning rate。
* 缺點 : 訓練比較慢。

1. Adam (Adaptive Moment Estimation) : 為目前最流行的optimizer。他是一種可以為每一個參數都計算adaptive learning rate的方法。他結合了momentum和RMS-Prop的使用。

一張含有 字型, 行, 圖表, 筆跡 的圖片

自動產生的描述

* 優點 : 計算效率高、需要比較少的記憶體空間。

1. 如何選擇optimizer :

宗傑以上幾種optimizer的特性，可以歸納出幾個選擇標準。

* 對於稀疏的資料，採用adaptive learning rate的optimizer。
* 在很多情況下，Adam、RMSprop有差不多的表現。
* Adam通常會比RMSprop表現更好一點，因為Adam可以透過momentum來保持更新的方向的一致性。
* 對於較大型的模型，建議使用Adam，因為Adam具有更好的收斂性能。

1. Reference

[1] Musstafa “Optimizers in Deep Learning” <https://musstafa0804.medium.com/optimizers-in-deep-learning-7bf81fed78a0>

[2] ML Glossary “Optimizers” <https://ml-cheatsheet.readthedocs.io/en/latest/optimizers.html>

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

[4] Devashree Madhugiri “Using CNN for Image Classification on CIFAR-10 Dataset” <https://devashree-madhugiri.medium.com/using-cnn-for-image-classification-on-cifar-10-dataset-7803d9f3b983>