Lab1
Backpropagation and Basic Pytorch

313510164 陳緯亭 電子碩一

September 23, 2024

Contents

1 My network **2**

2 Loss function **3**

3 Activation function **3**

3.1 ReLU 4

3.2 Sigmoid 4

4 Hyperparameters **5**

4.1 Epoch 5

4.2 Batch size 5

4.3 Learning rate 5

5 Optimizer **6**

5.1 Adam Optimization 6

5.2 SGD 6

6 What differences between the results of Task1 and Task2 **6**

7 Results **7**

7.1 Task1 7

7.2 Task2 7

8 References **8**

1 My network

本次設計了一層隱藏層的全連接神經網絡，如 Fig. 1 所示。經過對一至三層的嘗試，我發現，在其他條件保持不變的情況下，增加層數對於此訓練集的準確度並未帶來提升，甚至呈現出下降的趨勢，我猜測是因為我的 Activation Function 是使用 ReLU。Relu 值域區間為 $[0, \infty]$ 不會對數據做幅度壓縮，所以數據的幅度會隨著模型層數的增加不斷擴張。準確率的變化可參考圖 Fig. 2、Fig. 3和 Fig. 4。

基於單層隱藏層的設計，我進一步嘗試了不同權重矩陣大小的組合，並調整了批量大小（Batch size）和學習率（Learning rate），以期提高準確率。最終，在 Colab 平台上的驗證準確度達到 96%。根據參考資料，卷積神經網絡（CNN）的效果通常比全連接層（Fully Connected Layer）更佳，但由於 CNN 的實現相對複雜，故本次並未考慮以 CNN 作為架構。

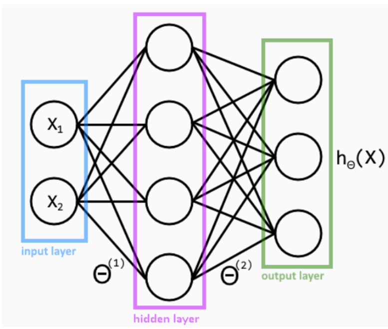

Fig. 1: 2-layer fully-connected neural network

```
Task1  | Epoch:  1  |Train Loss:  0.5042  |Train Acc:83.8360  |Val Loss:  0.2543  |Val Acc:92.1000
Task1  | Epoch:  2  |Train Loss:  0.1962  |Train Acc:93.9700  |Val Loss:  0.1872  |Val Acc:94.2700
Task1  | Epoch:  3  |Train Loss:  0.1245  |Train Acc:96.2200  |Val Loss:  0.1739  |Val Acc:94.6500
Task1  | Epoch:  4  |Train Loss:  0.0812  |Train Acc:97.7440  |Val Loss:  0.1711  |Val Acc:94.6200
Task1  | Epoch:  5  |Train Loss:  0.0517  |Train Acc:98.7040  |Val Loss:  0.1696  |Val Acc:94.8900
Task1  | Epoch:  6  |Train Loss:  0.0328  |Train Acc:99.3100  |Val Loss:  0.1673  |Val Acc:95.3800
Task1  | Epoch:  7  |Train Loss:  0.0204  |Train Acc:99.6180  |Val Loss:  0.1676  |Val Acc:95.5800
Task1  | Epoch:  8  |Train Loss:  0.0123  |Train Acc:99.7940  |Val Loss:  0.1684  |Val Acc:95.7600
Task1  | Epoch:  9  |Train Loss:  0.0081  |Train Acc:99.8660  |Val Loss:  0.1682  |Val Acc:95.9400
Task1  | Epoch: 10  |Train Loss:  0.0056  |Train Acc:99.8880  |Val Loss:  0.1702  |Val Acc:95.9100
```

Fig. 2: 2-layer result
[input, hidden, output] = [28*28, 360, 10]

```
Task1  | Epoch:  1  |Train Loss:  1.9167  |Train Acc:29.9000  |Val Loss:  1.5167  |Val Acc:51.7700
Task1  | Epoch:  2  |Train Loss:  1.2381  |Train Acc:57.2160  |Val Loss:  1.0216  |Val Acc:69.6500
Task1  | Epoch:  3  |Train Loss:  0.8744  |Train Acc:72.3400  |Val Loss:  0.6529  |Val Acc:78.2600
Task1  | Epoch:  4  |Train Loss:  0.7188  |Train Acc:78.0060  |Val Loss:  0.5991  |Val Acc:82.2900
Task1  | Epoch:  5  |Train Loss:  0.5913  |Train Acc:82.7740  |Val Loss:  0.5564  |Val Acc:83.4000
Task1  | Epoch:  6  |Train Loss:  0.5361  |Train Acc:84.4040  |Val Loss:  0.5148  |Val Acc:84.7800
Task1  | Epoch:  7  |Train Loss:  0.5039  |Train Acc:85.1160  |Val Loss:  0.5225  |Val Acc:85.2400
Task1  | Epoch:  8  |Train Loss:  0.4636  |Train Acc:86.6640  |Val Loss:  0.5125  |Val Acc:86.4500
Task1  | Epoch:  9  |Train Loss:  0.4244  |Train Acc:88.2520  |Val Loss:  0.4889  |Val Acc:87.5000
Task1  | Epoch: 10  |Train Loss:  0.3941  |Train Acc:89.1680  |Val Loss:  0.4876  |Val Acc:87.8200
```

Fig. 3: 3-layer result
[input, hidden1, hidden2, output] = [28*28, 360, 100, 10]

```
Task1  | Epoch:  1  |Train Loss:  2.3585  |Train Acc:9.7320  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  2  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  3  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  4  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  5  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  6  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  7  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  8  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch:  9  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
Task1  | Epoch: 10  |Train Loss:  2.3586  |Train Acc:9.7260  |Val Loss:  2.3392  |Val Acc:9.6600
```

Fig. 4: 4-layer result
[input, hidden1, hidden2, hidden3,output]= [28*28, 360, 100, 40, 10]

2 Loss function

Loss function 採用 Softmax 回歸，以及 Cross entropy 這兩層，這樣子可以輸出影像 0～9 的個別機率。

$$Softmax(x) = \frac{exp(x_i)}{\Sigma_{i=0}^{n} exp(x_i)}$$

$$CrossEntropy = -\Sigma_i t_i \log y_i$$

$$Backpropagation = y_1 - t_1$$

3 Activation function

在模型架構中，我採用了修正線性單元 rectified linear unit(ReLU) 作為激活函數。相比於 Sigmoid 函數，ReLU 在 MNIST 資料集上的收斂速度更快，且能

有效緩解過擬合問題。由於 ReLU 是一種非線性函數，非常適合處理非線性問題，應用在類神經網絡中，所訓練出的模型能夠更好地解決這類問題。Fig. 5 和 Fig. 6 為準確度的比較，很明顯 ReLU 表現較好。

3.1 ReLU

$$R(x) = max(0, x) \qquad\qquad R'(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \in \{0, 1\}$$

```
Task1  | Epoch:  1  |Train Loss:  1.9167  |Train Acc:29.9000  |Val Loss:  1.5167  |Val Acc:51.7700
Task1  | Epoch:  2  |Train Loss:  1.2381  |Train Acc:57.2160  |Val Loss:  1.0216  |Val Acc:69.6500
Task1  | Epoch:  3  |Train Loss:  0.8744  |Train Acc:72.3400  |Val Loss:  0.6529  |Val Acc:78.2600
Task1  | Epoch:  4  |Train Loss:  0.7188  |Train Acc:78.0060  |Val Loss:  0.5991  |Val Acc:82.2900
Task1  | Epoch:  5  |Train Loss:  0.5913  |Train Acc:82.7740  |Val Loss:  0.5564  |Val Acc:83.4000
Task1  | Epoch:  6  |Train Loss:  0.5361  |Train Acc:84.4040  |Val Loss:  0.5148  |Val Acc:84.7800
Task1  | Epoch:  7  |Train Loss:  0.5039  |Train Acc:85.1160  |Val Loss:  0.5225  |Val Acc:85.2400
Task1  | Epoch:  8  |Train Loss:  0.4636  |Train Acc:86.6640  |Val Loss:  0.5125  |Val Acc:86.4500
Task1  | Epoch:  9  |Train Loss:  0.4244  |Train Acc:88.2520  |Val Loss:  0.4889  |Val Acc:87.5000
Task1  | Epoch: 10  |Train Loss:  0.3941  |Train Acc:89.1680  |Val Loss:  0.4876  |Val Acc:87.8200
```

Fig. 5: The accuracy using ReLU

3.2 Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}} \qquad\qquad \sigma'(z) = \sigma(z) * (1 - \sigma(z))$$

```
Task1  | Epoch:  1  |Train Loss:  2.3585  |Train Acc:9.7820  |Val Loss:  2.3390  |Val Acc:9.6600
Task1  | Epoch:  2  |Train Loss:  2.3582  |Train Acc:9.8280  |Val Loss:  2.3384  |Val Acc:9.6600
Task1  | Epoch:  3  |Train Loss:  2.3570  |Train Acc:9.9520  |Val Loss:  2.3359  |Val Acc:9.6600
Task1  | Epoch:  4  |Train Loss:  2.3511  |Train Acc:10.2940  |Val Loss:  2.3246  |Val Acc:9.6600
Task1  | Epoch:  5  |Train Loss:  2.3312  |Train Acc:11.1480  |Val Loss:  2.2934  |Val Acc:9.6600
Task1  | Epoch:  6  |Train Loss:  2.2903  |Train Acc:12.8080  |Val Loss:  2.2420  |Val Acc:17.7500
Task1  | Epoch:  7  |Train Loss:  2.2360  |Train Acc:14.9940  |Val Loss:  2.1850  |Val Acc:17.7500
Task1  | Epoch:  8  |Train Loss:  2.1817  |Train Acc:16.6560  |Val Loss:  2.1328  |Val Acc:17.7600
Task1  | Epoch:  9  |Train Loss:  2.1360  |Train Acc:17.4380  |Val Loss:  2.0916  |Val Acc:17.8000
Task1  | Epoch: 10  |Train Loss:  2.0994  |Train Acc:17.6640  |Val Loss:  2.0564  |Val Acc:17.8700
```

Fig. 6: The accuracy using Sigmoid function

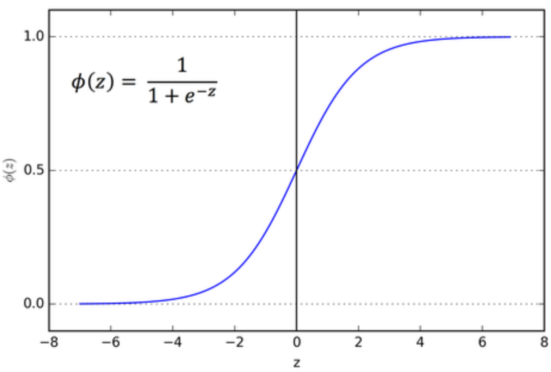

Fig. 7: Sigmoid function

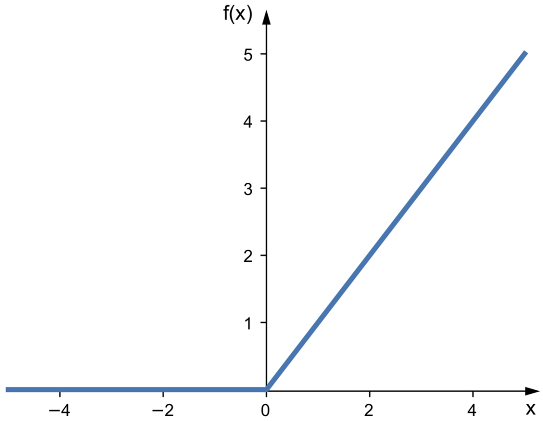

Fig. 8: rectified linear unit (ReLU)

4 Hyperparameters

主要調整的參數包括 Epoch、Batch size 和 Learning rate，這些都需要手動設定。通常我會從 Learning rate = 0.1 開始觀察，並根據訓練情況決定是否需要調大或調小。Batch size 則從 32 開始進行調整。Epoch 一開始設定為 50，除非出現 underfitting 的情況，否則通常不會變動。

4.1 Epoch

控制訓練迭代次數的參數，然而，隨著訓練精度的提高，驗證精度不一定會同步提升，可能會出現過擬合的情況。

4.2 Batch size

決定了如何將一個訓練集拆分成多個小批次，這樣神經網絡可以更頻繁地更新參數。然而，Batch size 不能設得過大，否則更新次數不足；過小則可能導致每次更新的信息量太少，影響訓練效果。

4.3 Learning rate

模型的收斂速度，學習率設得過小會使收斂過於平緩，而過大則可能導致模型無法收斂。

5 Optimizer

嘗試用了 Adam Optimization 和 stochastic gradient decent (SGD)，以準確率效果來說，SGD 的比較好，猜測可能是 Adam 參數調整的不好，所以後來不採用。

5.1 Adam Optimization

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L_t}{\partial W_t}$$
$$v_t = \beta_1 v_{t-1} + (1-\beta_2)(\frac{\partial L_t}{\partial W_t})^2$$

校正

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

更新 weight

$$W \leftarrow W - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$$

5.2 SGD

$$W \leftarrow W - \eta\frac{\partial L}{\partial W}$$

6 What differences between the results of Task1 and Task2

- Task2 在訓練速度上比 Task1 快很多。
- 如果 Task2 的 learning rate 開的跟 Task1 的 learning rate 一樣大，學習準確率會很慘，如 Fig. 9，嚴重發散。(Task 2 learning rate = 1e-3 開始調)

```
Task2  | Epoch:  1  |Train Loss:724.5391  |Train Acc:9.8920  |Val Loss:  2.3398  |Val Acc:9.6700
Task2  | Epoch:  2  |Train Loss:  2.3033  |Train Acc:9.7800  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  3  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  4  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  5  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  6  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  7  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  8  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch:  9  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
Task2  | Epoch: 10  |Train Loss:  2.3029  |Train Acc:9.7700  |Val Loss:  2.3361  |Val Acc:9.6700
```

Fig. 9: Task2 with the learning rate = 0.1 (as Task1)

- 在相同 Hyperparameters 和 network 架構下，Task2 的驗證準確度會比 Task1 的低大約 2～4 % 。

7 Results

7.1 Task1

Task 1 驗證準確度達 96.46 %。

```
Task1  | Epoch: 31  |Train Loss:  0.0008  |Train Acc:100.0000  |Val Loss:  0.1658  |Val Acc:96.5000
Task1  | Epoch: 32  |Train Loss:  0.0007  |Train Acc:100.0000  |Val Loss:  0.1664  |Val Acc:96.5000
Task1  | Epoch: 33  |Train Loss:  0.0007  |Train Acc:100.0000  |Val Loss:  0.1669  |Val Acc:96.5000
Task1  | Epoch: 34  |Train Loss:  0.0007  |Train Acc:100.0000  |Val Loss:  0.1675  |Val Acc:96.4900
Task1  | Epoch: 35  |Train Loss:  0.0006  |Train Acc:100.0000  |Val Loss:  0.1680  |Val Acc:96.4800
Task1  | Epoch: 36  |Train Loss:  0.0006  |Train Acc:100.0000  |Val Loss:  0.1685  |Val Acc:96.4800
Task1  | Epoch: 37  |Train Loss:  0.0006  |Train Acc:100.0000  |Val Loss:  0.1690  |Val Acc:96.4800
Task1  | Epoch: 38  |Train Loss:  0.0006  |Train Acc:100.0000  |Val Loss:  0.1695  |Val Acc:96.4800
Task1  | Epoch: 39  |Train Loss:  0.0006  |Train Acc:100.0000  |Val Loss:  0.1700  |Val Acc:96.4700
Task1  | Epoch: 40  |Train Loss:  0.0005  |Train Acc:100.0000  |Val Loss:  0.1704  |Val Acc:96.4600
```

Fig. 10: The final result of Task 1

丟到 Kaggle 上僅剩 94.06 %。

DL-test-predict (7).csv
Complete · 14s ago
0.9406

Fig. 11: Kaggle for Task 1

7.2 Task2

Task 2 驗證準確度達 94.44%。

```
Task2  | Epoch:191  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3926  |Val Acc:94.4400
Task2  | Epoch:192  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3927  |Val Acc:94.4400
Task2  | Epoch:193  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3928  |Val Acc:94.4400
Task2  | Epoch:194  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3928  |Val Acc:94.4400
Task2  | Epoch:195  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3929  |Val Acc:94.4400
Task2  | Epoch:196  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3930  |Val Acc:94.4400
Task2  | Epoch:197  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3931  |Val Acc:94.4400
Task2  | Epoch:198  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3932  |Val Acc:94.4400
Task2  | Epoch:199  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3933  |Val Acc:94.4400
Task2  | Epoch:200  |Train Loss:  0.0001  |Train Acc:100.0000  |Val Loss:  0.3934  |Val Acc:94.4400
```

Fig. 12: The final result of Task 2

8 References

1. 在深度學習中，Batch Size 是甚麼？
2. How to implement a feedforward backpropagation neural network in Python with MNIST dataset
3. 终于把神经网络算法搞懂了！
4. 卷积神经网络 (CNN) 反向传播算法
5. Dropout 原理介紹：理解深度學習中的 Dropout Layer
6. dropout 的 forward 过程及 backward 反向传播过程
7. Implement the Backpropagation with Python step by step (I)
8. 【深度學習】Hello Deep Learning! 使用 DNN 實作 MNIST
9. ReLU 为什么比 Sigmoid 效果好
10. MNIST - Deep Neural Network with Keras
11. Training a neural network on MNIST with Keras
12. 神经网络训练下降到一定程度不下降，测试集不变
13. 全連接神經網路 Fully-Connected Neural Network
14. Activation Functions —Sigmoid & ReLu & tahn & LeakyReLu & ELU
15. What are Hyperparameters ？ and How to tune the Hyperparameters in a Deep Neural Network?
16. What is the class of this image ?

17. [機器學習] Backpropagation with Softmax / Cross Entropy

18. [機器學習 ML NOTE]SGD, Momentum, AdaGrad, Adam Optimizer