深度學習

Lab Assignment 3

繳交日期：2025 年 12 月 11 日 23:59（星期四）

1. 目的

本作業延伸自 Lab Assignment 2，目的在於：

- 比較單層模型（Lab2）與多層模型（Lab3）的分類性能差異
  - 針對 Lab2 準確率低的 Group B 資料集，此資料集含有四種外觀類似的上衣類。
- 手動實作多層神經網路（Multilayer Perceptron, MLP）與反向傳播演算法（Backpropagation Algorithm）
  - 透過多層神經網路理解反向傳播演算法如何更新多層神經網路的權重與偏差值。
- 探討不同超級參數，對模型性能的影響，如：
  - **必要實驗項目：至少比較兩種不同激活函數（Activation Functions）：**
    1. ReLU（必選）
    2. Sigmoid（必選）
    3. Tanh（可加選）

    → **在相同的超參數組合下，討論激活函數性能與訓練穩定性**。
  - 其他建議實驗項目：嘗試其他超參數組合：
    - 隱藏層數（1–2 層）
    - 神經元個數（如 64 / 128 / 256）
    - 學習率（如 0.001、0.01、0.05）
    - Mini-Batch GD（建議）；批次大小（如 32、64、128）

    → 自行設定以上超參數，找到最佳超參數組合。

2. 課程章節與實驗環境

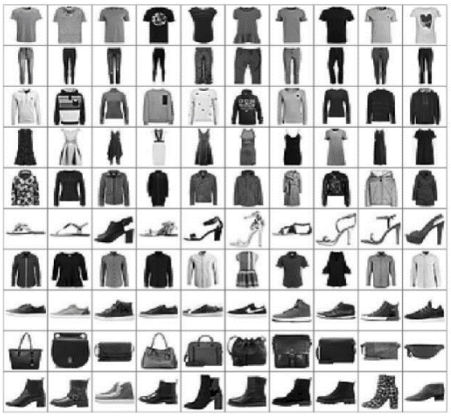
- 課程章節: Week 7 - Multilayer Networks and Backpropagation
- 實驗環境: 限定使用 Python 程式語言

3. 實驗與要求

- 程式撰寫要求：
  - 更新方法可使用 Stochastic / Batch / Mini-Batch，建議使用 Mini-Batch。
  - 本次作業著重於手動實作，不允許使用現成深度學習框架（如 TensorFlow 或 PyTorch），也不可使用任何 AI 輔助工具，例如 GitHub Copilot、ChatGPT 等。
- 網路架構：
  - 輸入層包含 784 個節點
  - **隱藏層**
    - **1–2 層**
    - **每層可選：64 / 128 / 256/其他 個神經元**
    - **激活函數，必須比較 ReLU 與 Sigmoid，（可加選）Tanh**

- 輸出層包含四個神經元
  - 激活函數為 **Softmax**
  - 損失函數，請使用 k 類交叉熵損失函數: $\boldsymbol{E}(\mathbf{a}) = -\sum_{j=1}^{k} y_j \log a_j$
    其中y_j為 One-Hot Encoding 標籤，其值為 0 或 1，a_j為神經元輸出值，k 為類別數。

- **輸入資料**
  - 本實驗採用 Lab2 的 Group B 資料集，即採用部分 **Fashion-MNIST** 資料。**Fashion-MNIST** 數據集，包含 10 個類別，每筆資料為 28x28 灰階圖像（784 維特徵向量），共有 60,000 筆訓練資料和 10,000 筆測試資料。



  - Lab2 的 Group B 資料集含有以下四種類別:
    0: Tshirt/top
    2: Pullover
    4: Coat
    6: Shirt
    有標籤資料(Group_B_train.csv)：每類 2,500 筆，共 10,000 筆，以供訓練及驗證。
    無標籤測試資料(Group_B_test.csv)：每類約 500 筆，共 2,000 筆，以供預測。

- **輸出資料**
  - 程序停止時，請顯示以下模型資訊：
    1. 隱藏層激活函數（ReLU / Sigmoid / Tanh）
    2. 隱藏層結構（層數、每層神經元個數）
    3. 其他超級參數：學習率、停止世代數與批次大小（若使用 Mini-Batch）
    4. 訓練與驗證集之準確率（accuracy）與損失值（loss）
    5. 測試資料的預測結果（每筆資料之預測類別編號）
    6. 繪製訓練與驗證集 loss / accuracy 曲線

    請依照助教指示之輸入/輸出格式要求。

- **訓練/驗證流程**
  - 建議使用 80% 訓練集、20% 驗證集。
  - 訓練停止條件可為：
    - 達到設定世代數（epoch）上限。此條件為避免無窮迴圈，不可視為找到合理模型。
    - 訓練損失不再改善
    - 驗證準確率下降（出現 overfitting）

4. 實驗報告討論（額外加分；最多 20 分）：

使用表格或圖表總結實驗結果，加以討論和分析，例如：

- 單層（Lab2） vs 多層（Lab3）的性能差異
  - 多層是否改善？改善多少？原因是什麼？

- 隱藏層激活函數比較，如：
  - 收斂速度
  - 是否有梯度消失
  - 那個激活函數對 Lab2 的 Group B 資料集較有效
- 隱藏層結構的影響
  - 隱藏層數（1－2 層）的性能差異
  - 不同神經元個數（64、128、256）的表現
- 那一種神經網路架構（即不同層數/數量的神經元）獲得了最佳訓練/驗證準確率。分析並解釋你的觀察。
- 其他超參數比較，例如：
  - 改變學習率（如 0.001、0.01、0.05）
  - 改變批次大小（如 32、64、128）
- 其他心得討論報告，例如：觀察比較訓練與驗證結果/訓練曲線與驗證曲線。

附錄 A： 權重初始化（Weight Initialization）

在多層神經網路中，權重初始化方式對訓練是否穩定有關鍵影響。若初始化不當，模型可能在訓練初期就遭遇：梯度爆炸（數值發散）、梯度消失（參數幾乎不更新），甚至導致 loss 變為 NaN，使訓練無法順利進行。因此，現代深度學習通常使用 Xavier Initialization 或 He Initialization。

(a) Xavier Initialization（Sigmoid / Tanh 適用）

Sigmoid 與 Tanh 在輸入值過大或過小時容易進入飽和區域，使其導數接近 0，導致梯度消失。Xavier 初始化希望在 forward 與 backward 兩個方向都保持訊號變異數穩定，因此同時考慮前一層的輸入元個數（n_{in}），與下一層的輸出元個數（n_{out}）。

兩種常見的實現方式:

1. Xavier 標準分佈 (Normal Distribution)

   從一個平均值 $\mu = 0$且標準差 $\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$ 為特定值的常態分佈中抽取權重，即$w_{ji} \sim N(0, \sigma^2)$。

2. Xavier 均勻分佈 (Uniform Distribution)，可用於淺層網路

   從 [–limit, limit] 的範圍內，均勻分佈抽取權重，即$w_{ji} \sim U(-\text{limit}, \text{limit})$，其中$\text{limit} = \sqrt{\frac{6}{n_{in}+n_{out}}}$。

(b) He Initialization（ReLU / Leaky ReLU 適用）

對於對稱分布(如常態分布)的輸入， ReLU 大約會將一半的輸入設為零（即激活值有一半機會為零），導致 ReLU 輸出變異數約為輸入的一半。為了抵消這種衰減，He 初始化將權重縮放得稍大一些，並且只考慮前一層的輸入元個數（n_{in}）。

兩種常見的實現方式:

1. He 標準分佈 (Normal Distribution)

   從一個平均值$\mu = 0$且標準差 $\sigma = \sqrt{\frac{2}{n_{in}}}$為特定值的常態分佈中抽取權重，即，即$w_{ji} \sim N(0, \sigma^2)$。

2. He 均勻分佈 (Uniform Distribution)，可用於淺層網路

   從 [limit, limit] 的範圍內，均勻分佈抽取權重，即$w_{ji} \sim U(-\text{limit}, \text{limit})$，其中$\text{limit} = \sqrt{\frac{6}{n_{in}}}$。

(c) 若初始權重的絕對值過大（如使用標準常態 $N(0, 1)$），則每一層的線性組合 $\mathbf{n}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ 的絕對值也變大，導致 Sigmoid/Tanh 進入飽和區、ReLU 訊號逐層放大，使得整個網路在訓練初期就出現梯度消失、梯度爆炸，甚至 loss 變成 NaN。

附錄 B：**Stochastic Backpropagation for Multi-Classification**

// Use **Activation Functions** $\boldsymbol{f}$ in hidden layers
// $\boldsymbol{f}$ can be ReLU, Sigmoid, and Tanh

// Use ***Softmax Function*** in the output layer with the ***cross-entropy loss:***
// $-\sum_{j=1}^{k} y_j \log a_j$

Initialize all network weights/biases to small random numbers

UNTIL a stopping condition is met, DO
// e.g., max epochs, validation accuracy decreases (overfitting), loss no longer decreases

FOR each (**x**, **y**) in the training dataset, DO

1. **Feedforward:**

   // Use $\boldsymbol{f}(\mathbf{n}^l)$ in hidden layers
   Input the instance **x** $(= \mathbf{a}^0)$
   For each $l = 1, 2, \ldots, L-1$
   Compute $\mathbf{n}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ and $\mathbf{a}^l = \boldsymbol{f}(\mathbf{n}^l)$

   // Apply ***Softmax Function*** in the output layer
   Compute $\mathbf{n}^L = \mathbf{W}^L \mathbf{a}^{L-1} + \mathbf{b}^L$ and $\mathbf{a}^L = \boldsymbol{Softmax}(\mathbf{n}^L)$
   // Use stabilized Softmax to avoid overflow or NaN

2. **Backward:**

   Step 2.1 Output layer error
   $\Delta^L = (\mathbf{a}^L - \mathbf{y})$

   Step 2.2 Hidden layer errors
   For each $l = L-1, L-2, \ldots, 1$
   Compute the error vector at layer l:
   $\Delta^l = ((\mathbf{W}^{l+1})^T \Delta^{l+1}) \odot \boldsymbol{f}'(\boldsymbol{n}^l)$
   // $\odot$: Element-wise product
   // (also known as the Hadamard product)

   Step 2.3 Update weights and biases
   For each $l = L-1, L-2, \ldots, 1$
   Compute $\mathbf{W}^l = \mathbf{W}^l - \eta \Delta^l (\mathbf{a}^{l-1})^T$ and $\mathbf{b}^l = \mathbf{b}^l - \eta \Delta^l$

Notes:

- ReLU: $f'(n) = 1$ if $n > 0$ else 0
- Sigmoid: $f'(n) = \sigma(n)(1-\sigma(n))$
- Tanh: $f'(n) = 1 - \tanh(n)^2$