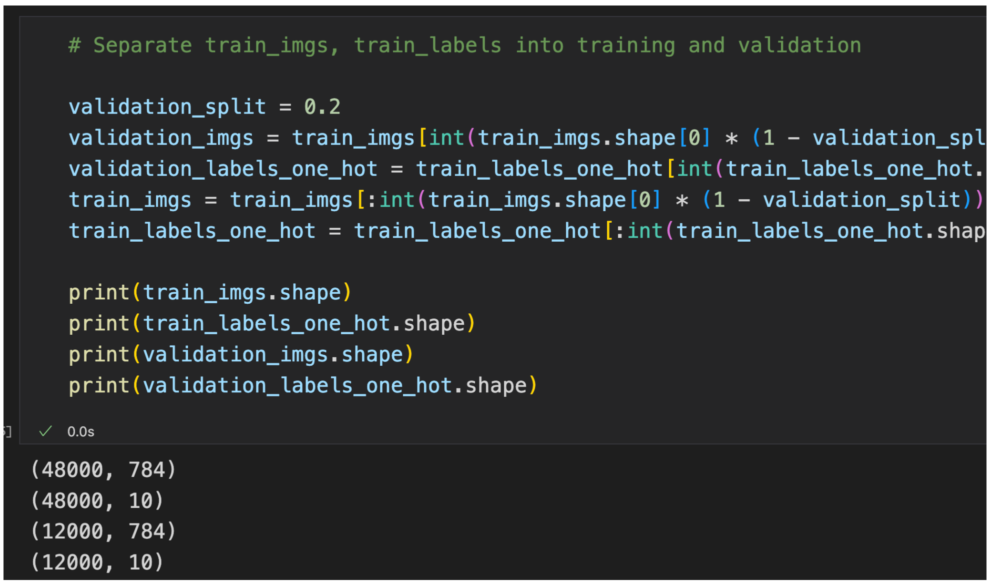
人工智慧模型設計與應用 Lab1

NM6121030 余振揚

1. **Cross-Validation:**

   Use 20% of training set as validation set. (Original shape: 60000, 784)

   ➔ Training set: (48000, 784), Validation set: (12000, 784)



```python
# Separate train_imgs, train_labels into training and validation

validation_split = 0.2
validation_imgs = train_imgs[int(train_imgs.shape[0] * (1 - validation_spl
validation_labels_one_hot = train_labels_one_hot[int(train_labels_one_hot.
train_imgs = train_imgs[:int(train_imgs.shape[0] * (1 - validation_split))
train_labels_one_hot = train_labels_one_hot[:int(train_labels_one_hot.shap

print(train_imgs.shape)
print(train_labels_one_hot.shape)
print(validation_imgs.shape)
print(validation_labels_one_hot.shape)
```

✓ 0.0s

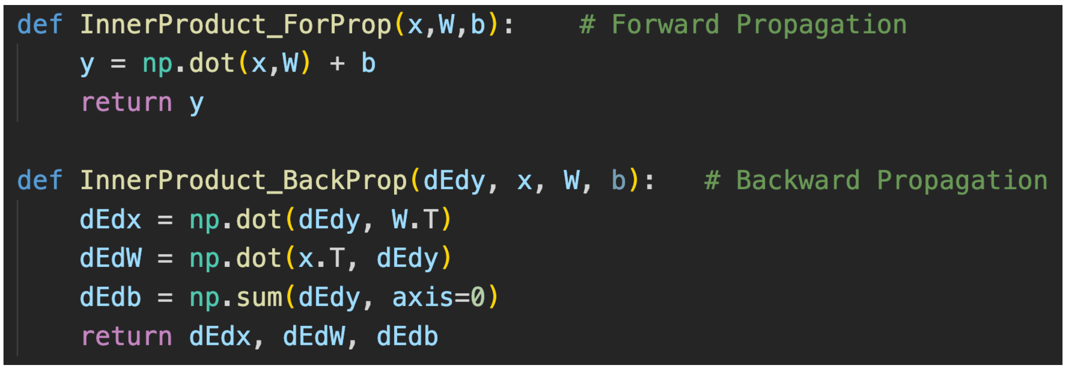
```
(48000, 784)
(48000, 10)
(12000, 784)
(12000, 10)
```

2. **Hyper Parameters:**
   - Learning Rate: 0.001
   - Epochs (Iteration): 100

3. **Hidden and Output Layer Definition:**

   Use one hidden layer with 64 neurons and one output layer with 10 neurons for the 10 classes (0 through 9).

4. **Forward Propagation and Backward Propagation Implementation:**
   - Inner-Product:



```python
def InnerProduct_ForProp(x,W,b):    # Forward Propagation
    y = np.dot(x,W) + b
    return y

def InnerProduct_BackProp(dEdy, x, W, b):   # Backward Propagation
    dEdx = np.dot(dEdy, W.T)
    dEdW = np.dot(x.T, dEdy)
    dEdb = np.sum(dEdy, axis=0)
    return dEdx, dEdW, dEdb
```

- Sigmoid (In this lab, I use Sigmoid as my activation function):

```python
def Sigmoid_ForProp(x):  # Forward Propagation: 1/(1+exp(-x))
    y = 1 / (1 + np.exp(-x))
    return y

def Sigmoid_BackProp(dEdy,x):   # Backward Propagation: y * (1 - y)
    # Compute the gradient of sigmoid function with respect to the input x
    y = Sigmoid_ForProp(x)
    dydx = y * (1 - y)
    # Compute the gradient of sigmoid loss function with respect to the input x
    dEdx = dEdy * dydx
    return dEdx
```

- ReLU:

```python
def Rectified_ForProp(x):   # Forward Propagation: max(0,x)
    y = np.maximum(0, x)
    return y

def Rectified_BackProp(dEdy,x):   # Backward Propagation
    # Compute the gradient of rectified linear unit function with respect to the input x
    dEdx = dEdy * (x > 0)
    return dEdx
```

- Softmax:

```python
def Softmax_ForProp(x):  # Forward Propagation
    # Compute the unnormalized probabilities
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    # Normalize them for each example
    y = exp_x / exp_x.sum(axis=1, keepdims=True)
    return y

def Softmax_BackProp(y,t):  # Backward Propagation
    # y is the output of softmax function (class probabilities)
    # t is the one-hot representation of the label

    # Compute the gradient of softmax loss function with respect to the input x
    dEdx = y - t
    return dEdx
```

5. **Forward and Backward Propagation Description:**

- **Forward Propagation (前項傳播):**

  在神經網絡的訓練過程中，前向傳播是從輸入數據到模型輸出的過程。它包括了將數據通過不同層（例如 Inner-Product Layer、Activation Layer、Softmax Layer）的運算，每一層都對數據進行變換和處理。

  - **Inner-Product Layer:**

    將輸入數據與權重矩陣相乘，並添加偏差項，產生輸出。

  - **Activation Layer:**

    對內積層的輸出進行非線性變換，例如 Sigmoid、ReLU 等。

  - **Softmax Layer:**

    計算每個類別的概率分佈，用於多類別分類問題。

- **Backward Propagation (反向傳播):**
  反向傳播是訓練過程中最重要的一部分，它用於計算梯度(Gradient Computation)，以便調整模型的權重和偏差(Update weight and bias)，以最小化損失函數(Minimize the Loss)。
  以下是反向傳播的一些主要步驟：
  - **Sofmax Layer:**
    計算模型的預測概率(Prediction Probability)和真實類別(Ground Truth)之間的誤差(Loss or Cost)，這個誤差被用來計算後一層(Activation Layer)的梯度。
  - **Activation Layer:**
    計算 Activation Function 對於誤差的梯度，這個梯度被用來計算前一層(指的是位於當前 Activation Layer 之前的 layer)的梯度。
  - **Inner-Product Layer:**
    計算內積層的權重和偏差的梯度，這個梯度被用來更新模型參數。

  在每一層的運算中，都需要計算梯度，以便向後傳播誤差。這些梯度是通過應用鏈式法則（chain rule）來計算的，並用於更新模型參數，以改進模型的性能。

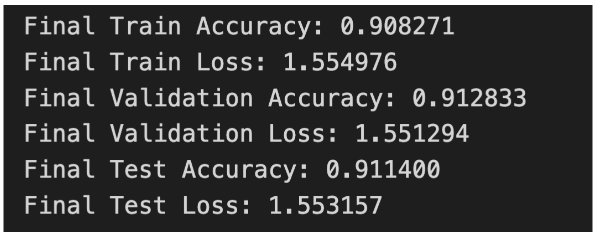
6. Loss Function (Cross Entropy):

```python
def CrossEntropy(y,t):  # Cross Entropy Loss Function
    # y is the output of softmax function (class probabilities)
    # t is the one-hot representation of the label

    # Compute the cross entropy loss function
    loss = -np.sum(t * np.log(y + 1e-8)) / y.shape[0]
    return loss
```

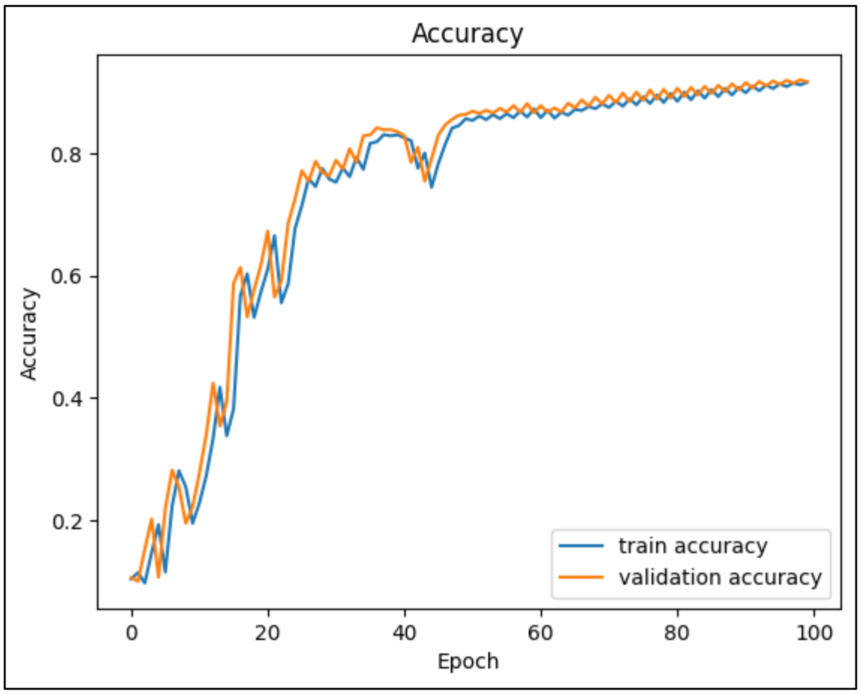
7. **Accuracy and Loss for Training, Validation, and Testing set:**

- Accuracy for testing (0.91):

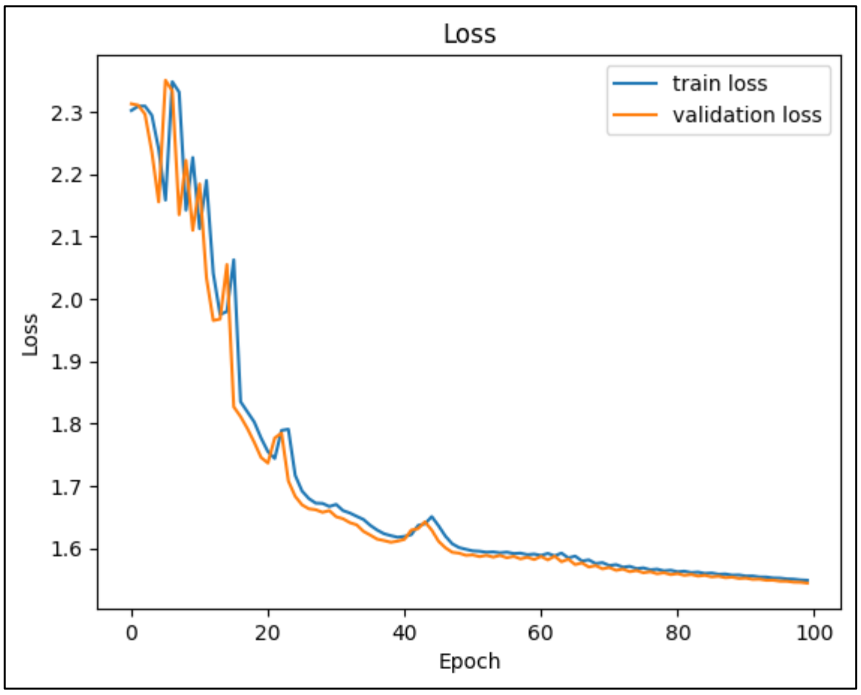


```
Final Train Accuracy: 0.908271
Final Train Loss: 1.554976
Final Validation Accuracy: 0.912833
Final Validation Loss: 1.551294
Final Test Accuracy: 0.911400
Final Test Loss: 1.553157
```

- Accuracy for training and validation per epoch:



- Loss for training and validation per epoch:



8. **Problem Encounter (ReLU VS Sigmoid):**

Most of the time, ReLU performs better than Sigmoid. However, in this lab, I observed that when using ReLU as the activation function, the accuracy consistently hovered around 0.11. It seems that the dataset is not complex enough for ReLU to effectively learn or extract meaningful features.

Finally, I decided to use Sigmoid as the activation function for this classification task.

大多數情況下，ReLU 的表現優於 Sigmoid。然而，在這個實驗中，我觀察到當使用 ReLU 作為激活函數時，準確度始終在 0.11 左右徘徊。看來數據集不夠複雜，無法讓 ReLU 有效地學習或提取有意義的特徵。

最終，我決定在這個分類任務中使用 Sigmoid 作為激活函數。

Reference: 為什麼深度學習模型準確率不會提昇？