Module1 final report

一、 (7,4) hamming code 四種解碼方式討論與比較

Channel decoding 是恢復原始訊息經過雜訊通道傳輸的過程。(7,4) hamming code 是一種廣泛使用的錯誤更正碼，可以更正傳輸數據中 bit 的錯誤。Module 1 中使用了 4 種方法解碼 Hamming code，包括 Syndrome Decoding、Maximum Likelihood、Support Vector Machine 和 Deep Learning。以下將比較這些方法：

1. Syndrome Decoding：

Syndrome Decoding 是一種常用的 hamming code 解碼方法。在這種方法中，先將接收到的經過雜訊的訊號做 hard decision 變成 binary code。

$$\hat{z}_i = sign(y_i) \;\; and \;\; \hat{d}_i = (\hat{z}_i + 1)/2$$

接著將 binary code 與 parity check matrix (H) 相乘為 syndrome vector。

$$\widehat{\boldsymbol{d}}\boldsymbol{H^T} = \boldsymbol{s}\ ,\ \boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

然後將 syndrome vector 與所有可能出現 error pattrern 表進行比較，更正相應的 error 以獲得原始訊號 bit。

Syndrome $\boldsymbol{s} = \widehat{\boldsymbol{d}}\boldsymbol{H^T}$	Error pattern $\hat{\boldsymbol{e}}$
(0, 0, 0)	(0, 0, 0, 0, 0, 0, 0)
(0, 0, 1)	(0, 0, 0, 0, 0, 0, 1)
(0, 1, 0)	(0, 0, 0, 0, 0, 1, 0)
(0, 1, 1)	(0, 0, 1, 0, 0, 0, 0)
(1, 0, 0)	(0, 0, 0, 0, 1, 0, 0)
(1, 0, 1)	(0, 1, 0, 0, 0, 0, 0)
(1, 1, 0)	(1, 0, 0, 0, 0, 0, 0)
(1, 1, 1)	(0, 0, 0, 1, 0, 0, 0)

這種方法相對簡單高效，但因為只考慮最有可能出現的錯誤，若有多個錯誤時，判別的正確率會較低。

2. Maximum Likelihood：

Maximum Likelihood 旨在將所有可能收到的訊息 mapping 到最有可能的原始訊息。在 (7,4) hamming code 的情況下，可以透過計算接收到的 code 與所有可能的 code 之間的 distance 來實現。然後選擇距離最小的 code 作為發

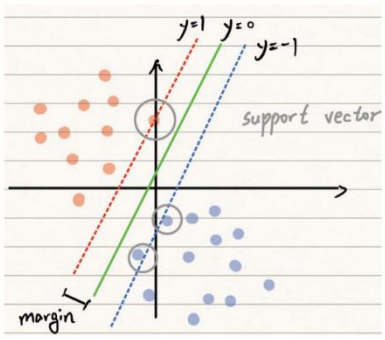
送 code。

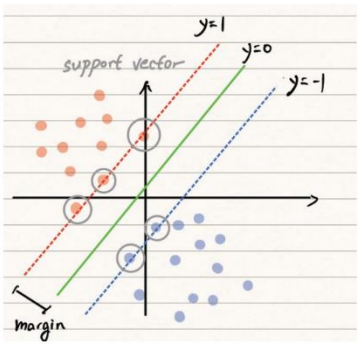
$$\hat{\boldsymbol{c}} = argmin_{\boldsymbol{c}\in C}\|\boldsymbol{y} - \boldsymbol{x}\|^2$$

這種方法可以更正多個錯誤，可以達到最佳的解碼性能，但因為需要考慮所有可能性，所以計算較 syndrome decoding 複雜。

3. Support Vector Machine：

Support Vector Machine 是一種監督式的學習方法，用統計風險最小化的原則來分類，其基礎就是找到一個決策邊界 (decision boundary) 讓兩類之間的邊界 (margins) 最大化，使其可以完美區隔開來。SVM 用於解碼就是將接收到的 code 分類為特定 codeword。





SVM 需要預先用 training data 來訓練，因此沒有 Syndrome decoding、Maximum Likelihood 來得及時，也容易受 data 品質影響，不過若已知雜訊模型，這種方法便有良好的解碼性能。因為有考慮多個 bit 錯誤的情況，所以表現較 Syndrome decoding 好，但不會像 maximum likelihood 有高準確性。

Deep Learning：

Deep Learning 使用 neural network 來學習接收到的 code 和相應的傳輸 code 之間的對應關係。此方法需要給定 neural network layer 和 nodes 的數量，以及 activate function、learning rate 等等，並使用大量 training data 來訓練，可能非常耗時。

```
def model_compile(SNR):
    learning_rate = 0.001/(SNR+1)
    opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)

    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Dense(128, activation='relu', input_shape=(7,)))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dense(16, activation='softmax'))

    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

然而，若參數調適得當，此方法可以獲得良好的解碼性能並且可以很好地應用到新的 data，表現可以接近 Maximum likelihood。

總之，這四種方法都有其優點和局限性。Syndrome decoding 簡單高效，但可能無

法更正多個錯誤。Maximum likelihood 是最佳解但計算複雜。Support vector machine 和 Deep learning 是基於機器學習的方法，可以實現良好的解碼性能，但需要訓練並且可能無法很好地應用到新數據。方法的選擇取決於具體的應用要求，包括所需的解碼性能、計算複雜度和可用的 training data。

二、 (15,11) hamming code 程式說明

在 module 1 的 mini project 中，將使用 Deep Learning 的方法解碼，不過這次原始訊息的編碼方法是使用(15, 11) hamming code。整個程式將與之前一樣比較 SNR=0~6，過程從訊息產生、編碼、modulate，到加入雜訊，最後建立並訓練模型，解碼訊息。

```
%%time
SNR_start = 0
SNR_end = 6
step_size = 1
SNR = np.arange(SNR_start,SNR_end+1,step_size)
length_SNR = len(SNR)
BLER = np.zeros(length_SNR)
rate = 0.7 # training data rate
datasetsize4SNR = np.array([1e5, 1e5, 1e5, 1e5, 1e5, 1e5, 3e5, 8e5, 8e5],dtype=int)
for i in range(length_SNR):

  #------------------------------------#
  # transmitter part
  #------------------------------------#
  # generate message
  message = message_gen(datasetsize4SNR[i])  產生訊息
  # generate codeword
  c, m = encoding(message)  用 (15,11) hamming code 編碼
  # BPSK modulation
  x = modulation(c, SNR[i])  modulate
  # go through AWGN channel
  y = AWGN_Channel(x)  加入雜訊

  #------------------------------------#
  # receiver part
  #------------------------------------#
  # decoding by deep learning
  # split data
  [train_y,train_m,test_y,test_m,train_size,test_size] = splitting_data(y,m,rate,datasetsize4SNR[i])

  # define model
  unique_category_count = 2048
  train_m_OneHot=tf.one_hot(train_m, unique_category_count)

  model = model_compile(SNR[i])  建立 neural network

  if i >= 4:
    num_epochs = 30
  elif i >= 1:
    num_epochs = 50
  else:
    num_epochs = 25

  # training and fitting model
  history = model.fit(train_y, train_m_OneHot, epochs=num_epochs, validation_split=0.2)
  # predict by the model                                                    訓練模型
  plt.plot(history.history['accuracy'])
  plt.plot(history.history['val_accuracy'])
  plt.legend(['training', 'validation'], loc = 'upper left')
  plt.show()

  pred = model.predict(test_y)  預測 (解碼)
  pred_m = np.argmax(pred, axis=1)

  # calculate error
  BLER[i] = caculate_error(pred_m, test_m)
```

1. 產生訊息:

```
def message_gen(dataset_size):
  m = np.zeros((dataset_size, 11))
  for i in range(dataset_size):
    m[i] = np.array(np.random.randint(0, 2, size=11))
  return m
```

隨機產生 11-bit 的 binary 訊息。

2. 用(15, 11) hamming code 編碼:

```python
# encoding function
def encoding(m):
    dataset_size = m.shape[0]
    G = np.array([
      [1,0,0,0,0,0,0,0,0,0,0,1,1,1,1],
      [0,1,0,0,0,0,0,0,0,0,0,0,1,1,1],
      [0,0,1,0,0,0,0,0,0,0,0,1,0,1,1],
      [0,0,0,1,0,0,0,0,0,0,0,1,1,0,1],
      [0,0,0,0,1,0,0,0,0,0,0,1,1,1,0],
      [0,0,0,0,0,1,0,0,0,0,0,0,1,1,1],
      [0,0,0,0,0,0,1,0,0,0,0,0,1,0,1],
      [0,0,0,0,0,0,0,1,0,0,0,0,1,1,0],
      [0,0,0,0,0,0,0,0,1,0,0,1,0,1,0],
      [0,0,0,0,0,0,0,0,0,1,0,1,0,0,1],
      [0,0,0,0,0,0,0,0,0,0,1,1,1,0,0]  Generator matrix
    ])

    c = np.dot(m, G)
    c = c % 2

    m_modified = np.zeros(dataset_size)
    for i in range(dataset_size):
      m_modified[i] += m[i][0]*1024 + m[i][1]*512 + m[i][2]*256 + m[i][3]*128 + m[i][4]*64
      + m[i][5]*32 + m[i][6]*16 + m[i][7]*8 + m[i][8]*4 + m[i][9]*2 + m[i][10]
                                                          轉換成 decimal
    m_modified = np.asarray(m_modified, dtype = 'int')
    return c, m_modified
```

將原始訊息$\boldsymbol{m}$乘上 generator matrix$\boldsymbol{G}$，得到 15-bit 的編碼訊息$\boldsymbol{c}$ ($\boldsymbol{c} = \boldsymbol{mG}$)。這裡將 11-bit 的 binary message 轉換成 decimal，以便之後 one hot 分類。

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = [P^T \quad I_4]$$

$$G = [I_{11} \quad P]$$

3. Modulate、加入雜訊:

Modulation (BPSK)

```python
def modulation(c,SNR):
    # type your own code
    # assign the signal power based on the given SNR
    # implement the BPSK modulation
    N0 = 1
    P = (10 ** (SNR/10) )* N0
    x = np.zeros(c.shape)
    for i, code in enumerate(c):
      x[i] = np.sqrt(P)*(2*code-1)
    return x
```

AWGN channel

```python
def AWGN_Channel(x):
    # assign the normalized noise
    # return the receive signal y by transmitted signal x plus the noise n
    N0 = 1
    sigma = np.sqrt(N0/2)
    y = np.zeros(x.shape)
    for i in range(y.shape[0]):
      y[i] = x[i] + np.random.normal(0, sigma, x[i].shape)
    return y
```

將編碼過的 message 以對應的訊雜比 (SNR) 調變，並加上 normal distribution 的雜訊，變成接收到的訊號$\boldsymbol{y}$。

4. Split data:

```
def splitting_data(y,m,rate,file_size):
  train_size = round(file_size*rate)
  test_size = file_size - train_size
  train_y = y[0:train_size,:]
  train_m = m[0:train_size]
  test_y = y[train_size:file_size,:]
  test_m = m[train_size:file_size]
  return train_y, train_m, test_y, test_m, train_size, test_size
```

將 message 和接收訊息分成 training data 和 testing data。

5. 建立 neural network:

```
def model_compile(SNR):
    learning_rate = 0.001/(SNR**1.8+1)
    opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)

    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Dense(128, activation='relu', input_shape=(15,)))
    model.add(tf.keras.layers.Dense(256, activation='relu'))
    model.add(tf.keras.layers.Dense(2048, activation='sigmoid'))
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

Activate function
不同 layer
每層 layer 中 node 數量

給定 layer、node，以及 activate function，並根據 SNR 調整 learning rate (SNR 越大，模型越容易收斂，因此 learning rate 調小)。

6. 訓練模型並預測:

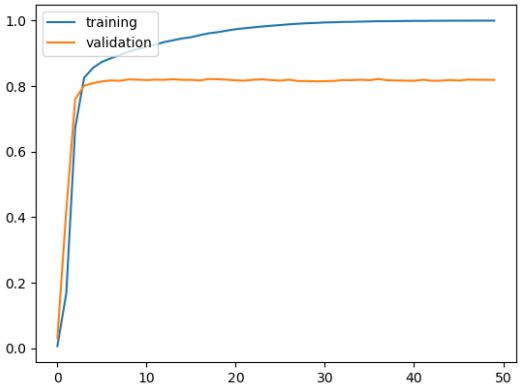
```
# define model
unique_category_count = 2048
train_m_OneHot=tf.one_hot(train_m, unique_category_count)

model = model_compile(SNR[i])

if i >= 4:
  num_epochs = 30
elif i >= 1:
  num_epochs = 50
else:
  num_epochs = 25

# training and fitting model
history = model.fit(train_y, train_m_OneHot, epochs=num_epochs, validation_split=0.2)
# predict by the model
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()

pred = model.predict(test_y)
pred_m = np.argmax(pred, axis=1)
```

2^{11}類
訓練模型
預測 (解碼)

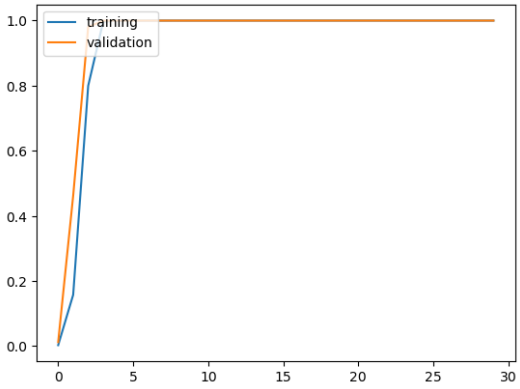
將 training data 丟入模型訓練，因為 message 為 11-bit，所以模型實際上是做 2^{11}=2048 類的分類。訓練完後丟入 test data，得到預測(解碼)後的訊息。

7. 結果

訓練過程中，SNR 越小，validation 的曲線難接近 100%，反之 fold validation 則能預測精準。



SNR = 0



SNR = 6

最後調參完的結果，表現有超過 Syndrome decoding，雖然 SNR 小時與 Syndrome decoding 差不多(learning rate 應該可以再小一點)，不過到 SNR=6 時幾乎能接近 Maximum likelihood。