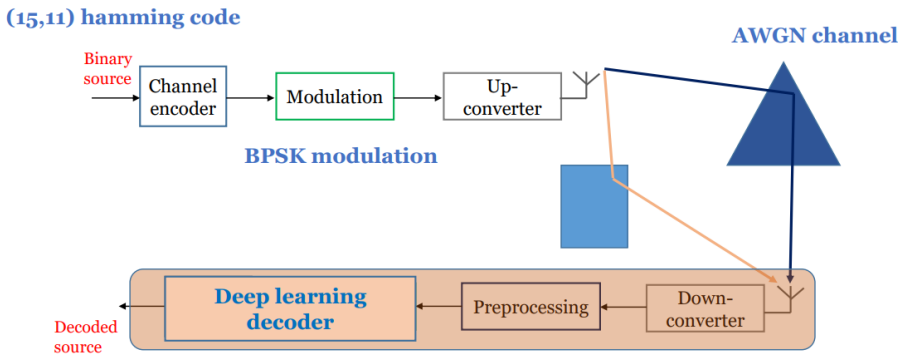
AI 無線通訊系統實

Module 1 Report

賴昱凱 111511141

一、Description and final results of mini project



本實驗主要利用(15, 11) hamming code 將訊號 encode、BPSK 進行 modulation，在加上高斯雜訊後，利用 deep learning 的方式將其 decode 回原訊號。

1. 訊號產生

```python
def message_gen(dataset_size):

  # my code
  m = np.array(np.random.randint(0, 2, (dataset_size, 11)))

  return m
```

產生隨機於[0, 1]的整數，即二進位，並指定輸出形狀為(dataset_size, 11)，代表共有 dataset_size 筆資料，每筆資料有 11 個位元。

2. (15, 11) hamming code

```python
def encoding(m):
    # my code
    # (15,11) Hamming Code Generator matrix
    P_T = np.array([
        [1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1],
        [1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1],
        [1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0],
        [1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0]
    ])

    # 取得 P 矩陣 (P 是 P^T 的轉置)
    P = P_T.T

    # 建立 11x11 單位矩陣 I_11
    I_11 = np.eye(11, dtype=int)

    # 拼接 G 矩陣 [ I_11 | P ]
    G = np.hstack((I_11, P))

    c = np.mod(np.dot(m, G), 2)  # 編碼後的結果
    return c
```

(n, k)-linear Block Codes 是一種 encode 方式，他可以皆由額外的位元去驗證接收到的訊號是否正確，若有少數位元遺失，可以將其回復，具體方式如下：

我們有 k-bit message $\boldsymbol{m}$, n-bit codeword $\mathbf{c}$，兩者將符合以下關係：

$$\boldsymbol{c = mG}$$

G 為一個$(k \times n)$的矩陣，稱為 generator matrix。

並存在一個$(n-k \times n)$的 H 矩陣，符合以下關係：

$$\boldsymbol{cH^T = Hc^T = 0}$$

Rows of $\boldsymbol{H}$ span the nullspace of $\boldsymbol{G}$，我們稱 $\boldsymbol{H}$ 為 parity check matrix。而其中 generator matrix 以及 parity check matrix 又符合：

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

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

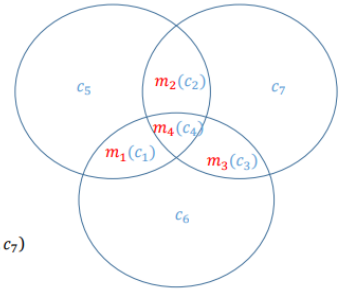
可以更容易的做出 generator matrix。

因此，若收到的訊號用 parity check matrix 驗證後等於零，就可以知道這個訊號沒有錯誤。而不同的 linear Block Codes 有不同的糾正方式，以比較簡單的(7, 4) hamming code 為例，他利用下圖進行糾正：

• $c = mG$

$$\mathbf{G} := \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

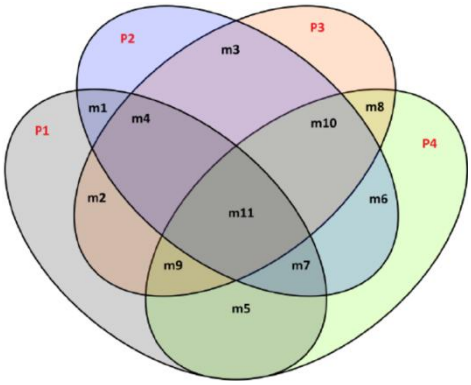
where $\boldsymbol{m} = (m_1, m_2, m_3, m_4)$ and $\boldsymbol{c} = (c_1, c_2, c_3, c_4, c_5, c_6, c_7)$



5

簡單來說，它是由三個圓組成，若在圓中的 m 為 1 的總數為奇數，那該圓的 c 就是 1，反之總數為偶數就是 0，組成 7 位元的 codeword。當有錯誤出現時，若只有一個 bit 出錯也可以很輕易地找到錯誤並修正他。

而本實驗使用的(15, 11) hamming code 就比較複雜了，圖片如下。



3. BPSK modulation

```
def modulation(c,SNR):
  # type your own code
  # assign the signal power based on the given SNR
  P = np.power(10,SNR/10)
  # implement the BPSK modulation
  x = np.sqrt(P)*(2*c - 1)
  return x
```

因為 SNR 是以分貝為單位，因此要先將其轉換為線性比例才是訊號的功率：

$$P = 10^{\frac{SNR}{10}}$$

BPSK modulation 則是將 0, 1 映射到 -1, +1，利用$2 \times c - 1$就可以達成這個目的，而最後再乘上剛剛算出來的$\sqrt{P}$，讓輸出訊號能量與 SNR 匹配。

至於為什麼要開根號，這是因為訊號功率為振幅的平方，因此訊號本身應該要乘以根號 P 才會讓其功率為 P 倍。

4. AWGN channel

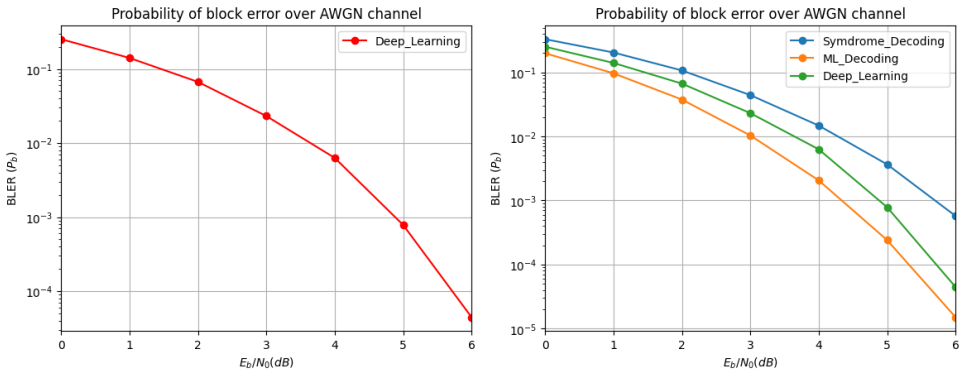
```
def AWGN_Channel(x):
    # assign the normalized noise
    n = np.random.normal(0,np.sqrt(1/2), x.shape)
    # return the receive signal y by transmitted signal x plus the noise n
    y = x + n
    return y
```

這個步驟就是讓傳出去的訊號加上高斯雜訊以模擬現實中訊號被影響的問題。其中參數分別為(平均值μ, 標準差σ, 形狀)。

5. Deep Learning

```
model = keras.models.Sequential([
        keras.layers.Dense(128, activation='gelu', input_shape=(15,)),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.1),
        keras.layers.Dense(256, activation='gelu'),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.1),
        keras.layers.Dense(128, activation='gelu'),
        keras.layers.BatchNormalization(),
        keras.layers.Dropout(0.1),
        keras.layers.Dense(64, activation='gelu'),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(unique_category_count, activation='softmax')
])
```

利用一個 input dimension=15、output dimension=2^{11}的模型進行訓練。輸入即為加上雜訊的 codeword，因此為 15 位元，而輸出結果為一個機率分布，代表每一種 code 可能的機率，因為有 11 個位元，且每個位元都有可能是 0 或 1 兩種選項，因次共有2^{11}種可能。而預測結果即為機率最高的那個選項。訓練結果如下：



二、Description and discussion of all decoding method in Module 1

1. Hard decision

這是最簡單的一種解碼方式，簡單來說就是經過 BPSK modulation

並加上雜訊的訊號若大於 0，就將其判斷為 1，反之若小於 0，就判斷為 0。這個方式計算量小、解碼速度快，不過性能較差。

2. Syndrome decoding

這個方式主要用於 linear block codes，例如前面提到的 hamming code。利用 Parity-Check Matrix 來檢查是否出現錯誤，$s = Hr^T$，若 s = 0 就代表沒有錯誤，不等於 0 就代表有錯誤，並可以根據 s 來判斷錯誤的地方，本實驗直接利用查表的方式將每一種錯誤都有對應的錯誤位元。這個方式可以處理錯誤，但可處理的數量不多。

3. Maximum Likelihood (ML) decoding

這個方式是將所有可能的組合都通過 generator matrix 後得到他們的 codewords，並經過 BPSK modulation，計算所有經過調變的可能結果與收到的訊號 y 的距離，最短的那個即為預測的結果。這個方式雖然會需要極大量的計算資源以及計算時間，但他也是最佳的解碼方式，並適用於任何 encode 方式。

4. Auto-encoder

這個方式並不是像其他的方式是利用已知的 encode 方式再想辦法去解碼，而是讓模型自行尋找最適合的 encode 以及 decode 方式，也可以算是一種 un-supervised learning，我們只需事先告知模型他的 input dimension、encoder 和 decoder 之間的 channel dimension，並讓 decoder 將加上雜訊的 channel 資料想辦法解碼回 input 即可。整個過程都是利用模型自行尋找相關特徵，因此我們不必手動設計編碼及解碼方式，不過這個方式需要大量數據進行訓練，同時訓練過程也需要大量的計算資源，硬體需求高。

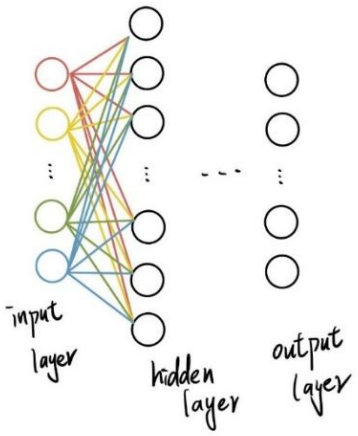
5. Deep Learning decoding

這個方式與上一個 auto-encoder 有異曲同工之妙，不過他是利用已知的 encode 方式，僅將 Decoder 的部分應用深度學習進行解碼。深度學習有很多種的架構，包括 CNN、RNN/LSTM、transformer 等，而本實驗是選擇使用最基礎的 DNN，以下簡單介紹 DNN model。

我使用的模型架構如下：

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                512

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 16)                1040

=================================================================
Total params: 18,128
Trainable params: 18,128
Non-trainable params: 0
```

本模型是由 2 層 hidden layer 以及 input、output layer，共 4 層所組成的 DNN 模型，DNN 模型的全稱為 Deep Neural Network，顧名思義代表所有的節點都與前後層的所有節點相連，利用權重做相乘$a_j = \sum_{i=1}^{D} W_{ji}x_i + w_0$往下一層前進，因此整個每層共有$D_{in} \times D_{out} + bias$個參數，以本模型為例：



$$7 \times 64 + 64 = 512$$

$$64 \times 128 + 128 = 8320$$

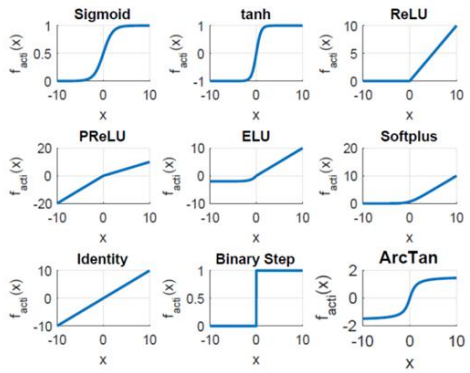
$$128 \times 64 + 64 = 8256$$

$$64 \times 16 + 16 = 1040$$

全部相加即為本模型的總參數：18128

除了 linear 以外，還有一個部分稱為 activation function，他是非線性的 layer，因為若整個 model 都是 linear 的 layer，會很難處理現實中的問題，因為現實中的問題基本上都不會是 linear 的，因此需要在 model 中加上 activation function 來讓模型更加靈活。

本模型皆使用 ReLU 這個最常見的 activation function，他會將正數原封不動的送給下一層，而遇到負數就將其設為 0，這是一種最不耗計算資源的 non-linear 方式。下圖還有更多常見的 activation function。

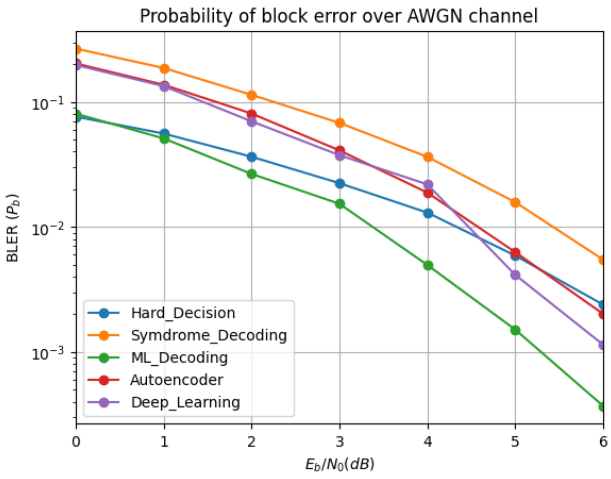


本模型的輸出層為有 16 個節點的神經層，最後一層利用 softmax 作為 actiavtion function，利用指數化確保輸出皆為正數，再讓所有輸出的值都介於 0~1 且相加為 1，代表各種類的可能性，最高的那個就代表此次模型預測的結果，這是幾乎所有分類問題處理輸出最基本的方式。

Softmax:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Simulation results (BLER of all decoding method) is necessary!



BLER_hard_decision = [0.07629, 0.05586, 0.03645, 0.02239, 0.01302, 0.0059, 0.00239, 0.00095, 0.00017]

BLER_SymdromeDecoding = [0.26725, 0.187, 0.114, 0.068, 0.036375, 0.01578, 0.005475, 0.00147, 0.000282]

BLER_ML = [8.05625e-02, 5.10625e-02, 2.65625e-02, 1.53125e-02, 4.96250e-03, 1.50875e-03, 3.70000e-04, 5.62500e-05]

BLER_autoencoder = [0.20328333, 0.13721667, 0.08078333, 0.04095, 0.01881667, 0.00628333, 0.00201667]

BLER_DL = [0.1975, 0.13392857, 0.07, 0.0375, 0.02189286, 0.00413571, 0.00114286, 0.00022857]