1. (1%)選擇的題目

   Listen and Translate

2. (1%)Team name, members and your work division

   ■ Team name：NTU_r05942017_ㄏㄏㄏㄏㄏㄏㄏㄏㄏ

   ■ members：
   學號：r05942017 系級：電信碩二 姓名：黃梓鳴
   學號：r05942148 系級：電信碩一 姓名：鄭立晟

   ■ work division：
   處理 feature 的部分:兩者同時
   研究 Seq2Seq model 的部分:兩者同時
   研究 retrieval model 的部分:黃梓鳴
   研究 ensemble 的方法:鄭立晟

3. (3%)Preprocessing/Feature Engineering

   ■ Seq2Seq

   → Audio data (encoder input data)：
   總共為 3 dimension 的資料，first dimension 為總資料量，筆數為 45036、second dimension 為最大聲音長度，長度為 246，長度不足 246 的聲音則補 0 補至 246、third dimension 為音符的向量長度，長度為 39。因此 encoder input data 的 numpy array shape 為 (45036,246,39)。

   → Caption data (Decoder input data)：
   總共為 3 dimension 的資料，first dimension 為總資料量，筆數為 45036、second dimension 為最大句子長度+1=14，+1 是為了在每個句子前面加入 begin of sentence(BOS)符號，而句子長度+1 不足 14 的句子則補 0 補至 14、third dimension 為 one hot encoding 的字典大小，大小為 2391，2391 包含原句子中文字的種類個數 2389，再加上 begin of sentence(BOS)與 end of sentence(EOS)，而單一中文字的向量表示為在該字典對應的中文字的 index 下，標上 1，而其餘則標 0。因此 Decoder input data 的 numpy array shape 為 (45036,14,2391)。

→ Caption data (Decoder label data)：
總共為 3 dimension 的資料，first dimension 為總資料量，筆數為 45036、second dimension 為最大句子長度+1=14，+1 是為了在每個句子後面加入 end of sentence(EOS)符號，而句子長度+1 不足 14 的句子則補 0 補至 14、third dimension 為 one hot encoding 的字典大小，大小為 2391，2391 包含原句子中文字的種類個數 2389，再加上 begin of sentence(BOS)與 end of sentence(EOS)，而單一中文字的向量表示為在該字典對應的中文字的 index 下，標上 1，而其餘則標 0。因此 Decoder label data 的 numpy array shape 為 (45036,14,2391)。

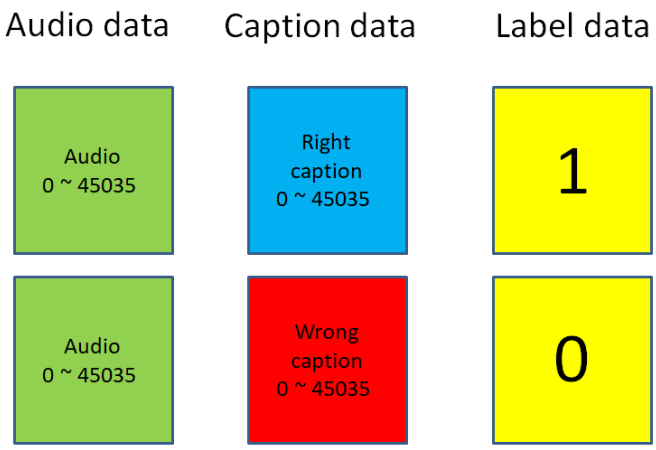
■ Retrieval model

→ Audio data：
總共為 3 dimension 的資料，first dimension 為總資料量，筆數為 45036*2，前 45036 是用來對應正確的中文句子，後 45036 是用來對應錯誤的中文句子、second dimension 為最大聲音長度，長度為 246，長度不足 246 的聲音則補 0 補至 246、third dimension 為音符的向量長度，長度為 39，而每一筆音符向量有做 normalization：(音符向量-平均值)/標準差。因此 Audio data 的 numpy array shape 為 (90072,246,39)。

→ Caption data：
總共為 3 dimension 的資料，first dimension 為總資料量，筆數為 45036*2，前 45036 為 Audio data 所對應的正確中文句子，後 45036 為 Audio data 所對應的錯誤中文句子，而錯誤的中文句子的挑選方法為：將全部配對正確的中文句子往後位移一個 index，最尾端的句子則移至第一個，如此可確保後 45036 為 Audio data 所對應的錯誤中文句子、second dimension 為最大句子長度 13，而句子長度不足 13 的句子則補 0 補至 13、third dimension 為中文字 pretrained 好的 model 產生的中文向量，大小為 300，而每一筆中文向量有做 normalization：(中文向量-平均值)/標準差。因此 Caption data 的 numpy array shape 為 (90072,13,300)。

→ Label data：
總共為 2 dimension 的資料，first dimension 為總資料量，筆數為 45036*2，前 45036 為 1， Audio data 與 Caption data 內積為 1，則代表兩向量有高相似度，後 45036 為 0， Audio data 與 Caption data 內積為 0，則代表兩向量有低相似度、second dimension 為 1 與 0。

→ draw of Audio data + Caption data + Label data to explain total data：



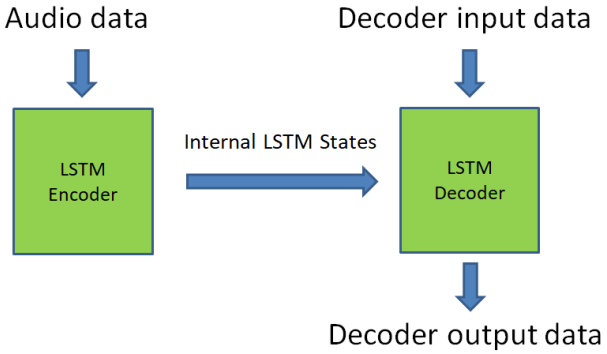
→ Data Shuffling：

將 Audio data、Caption data、Label data 同時 shuffling，使用 sklearn shuffle function 可以同時 shuffle 3 個 numpy array，不讓 Audio data、Caption data、Label data 的相對應錯亂。

4. (7%)Model Description (At least two different models)

■ Seq2Seq

→ Training Flow Diagram：



→ Train Model：

```
### Seq2Seq Model Start ###
# Encoder Model
encoder_inputs = Input(shape=(None,audio_vector_length))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
encoder_states = [state_h, state_c]

# Decoder Model, using `encoder_states` as decoder initial state
decoder_inputs = Input(shape=(None,caption_vector_length))
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,initial_state=encoder_states)
decoder_dense = Dense(caption_vector_length, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
### Seq2Seq Model End ###
```

→ Detailed Explanation (Training)：

1. 先將每個讀進來的中文句子在頭跟尾加記號，我是在句子頭加” 在句子尾加’。

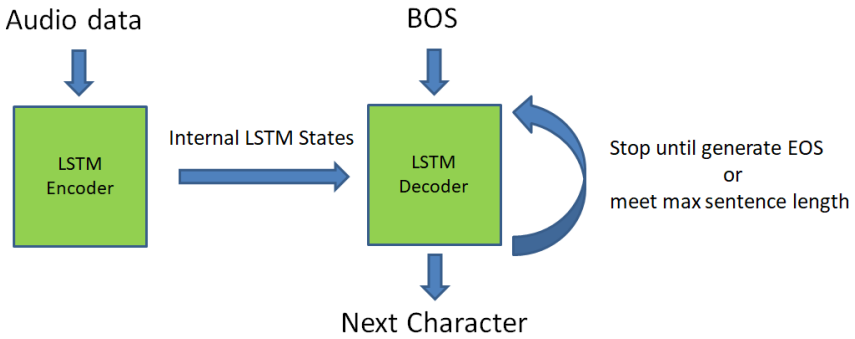
2. 將中文字建一個字典，總共有 2391 種字，用 one-hot encoding 的方式將中文字轉成向量。

3. 製造 input 的 format,input 有兩個，一個是中文字用 one hot encoding 的轉成的向量，格式為(訓練資料中最長句子的長度,句子總類數目),另一個為 train data 的聲音檔向量，格式為(聲音最大長度:246,音符向量長度:13)。

4. 製造 output data，output data 的格式跟 input data 的中文字向量一樣，但是時間格上領先一格，即 output[t-1]=input[t]，且沒有包含起始符號。

5. 訓練時的模型是把 encoder 和 decoder 接在一起訓練，encoder 是一個 LSTM 層，input shape 為 (246,13)，這層 LSTM 除了一般的 output 之外還要回傳 state，將這 state 設為 decoder 的初始 state，decoder 的 input shape 為 (13,2391)，decoder 是回傳整個句子，之外還會回傳 state，之後在將回傳的句子進入 dense 層(activation='softmax')，output shape 為(13,2391)，跑約 20 個 epoch。

→ Testing Flow Diagram：



→ Test Model：

```
### Seq2Seq Model Start ###
# Encoder Model
encoder_model = Model(encoder_inputs, encoder_states)
# Decoder Model
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([decoder_inputs] + decoder_states_inputs,[decoder_outputs] + decoder_states)
### Seq2Seq Model Start ###
```

→ Detailed Explanation (Testing)：

1. 將 train 好的模型另外在疊成 decoder 和 encoder 兩個模型，encoder 的模型為輸入整串音訊檔(246,13)，輸出為經過 LSTM 後的 state。

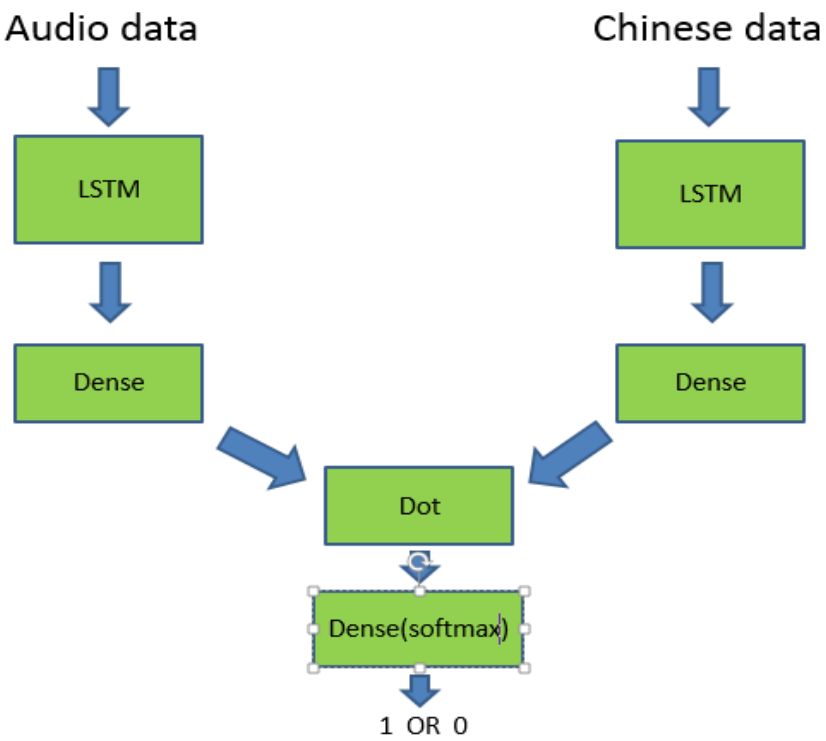
2. decoder 的模型輸入為 state 以及一個中文字的向量，輸出為一個 state 以及中文字的向量。

3. 將整句音訊向量丟到 encoder 模型裡，encoder 會回傳一個 state，將這 state 和"這符號的向量(起始符號)一起丟入 decoder 中，decoder 會回傳一個機率分布的向量及 state2，將那機率分部轉成最像它的向量，此向量的字即為翻譯句子的一個字，接著再把這向量和 state2 丟到 decoder 中，重複一樣的動作，直到 decoder 的 output 為結束符號為止。

4. 將這中間產生的所有機率分佈依序排列，和每題的每個選項各別作內積，內積最高的即為最相似的，即為答案。

■ Retrieval

→ Training & Testing Flow Diagram：



→ Train / Test Model：

```
encoder_inputs = Input(shape=(max_encoder_seq_length, num_encoder_tokens))
LSTM_encoder_layer=LSTM(latent_dim,implementation=1,go_backwards=True)
encoder_outputs= LSTM_encoder_layer(encoder_inputs)
encoder_outputs=Dense(256)(encoder_outputs)
encoder_outputs=LeakyReLU(0.25)(encoder_outputs)

decoder_inputs = Input(shape=(max_decoder_seq_length, num_decoder_tokens))
LSTM_decoder_layer=LSTM(latent_dim,implementation=1,go_backwards=True)
decoder_outputs= LSTM_decoder_layer(decoder_inputs)
decoder_outputs=Dense(256)(decoder_outputs)
decoder_outputs=LeakyReLU(0.25)(decoder_outputs)
DOT=dot([encoder_outputs,decoder_outputs],axes=1)

den=Dense(1,activation='sigmoid')(DOT)

model = Model([encoder_inputs,decoder_inputs], den)
model.summary()
model.compile(optimizer='rmsprop', loss='binary_crossentropy',metrics=['accuracy'])

model.fit([total_shuffle_encoder_data, total_shuffle_decoder_data],hinge_shuffle,
          batch_size=batch_size,callbacks=callbacks,
          epochs=epochs,
          validation_split=0.05)
```

→ Detailed Explanation (Training & Testing)：

1. 利用 pretrain 好的中文 model，先將中文字都轉成大小為 300 的向量

2. 將音訊資料和中文字資料都輸入進去，一句一句輸入

3. 之所以要用 Leakyrelu 是因為在後面有 DOT，假設是用 sigmoid 或是 relu，則 dot 起來的值必大於 0，不是我們想要的，在更後面的 sigmoid 會恆大於 0.5

4. 最後一層用 sigmoid 是要把輸出控制在 0~1 之間，因為我們的 label 是 0 跟 1，要把它控制在 0 和 1 之間

5. train 好之後將 test 每題的音訊向量和每個選項中文句子作 predict，取四個選項中 predict 出來最高的那一個當成答案

5. (8%)Experiments and Discussion:

■ **Seq2Seq:**

→ Data Preprocessing

目前這模型中文字只想到用 one hot encoding 的方法才 train 的出來，而 one hot encoding 的向量太大，我電腦 16G 記憶體沒辦法用到所有資料來 train，會 memory error。

→ Training

training 的時間較久，而且模型在一開始建構時也較難理解，特別是在將聲音訊號經過 LSTM 後，把當下的 encoder hidden state 當作 decoder 的 initialize state，去預測下一個所產生的字，這種方法如果是用在當 encoder input data shape 不等於 decoder input data shape 的時候，將會 loss 掉很多資料。EX：聲音訊號的 max length 長度為 246，而一個中文字的 max length 為 14，如果直接將聲音訊號一對一對應至中文字，會發現無法全部對齊，因為 max length 的長度不一樣，因此這樣會造成對應錯誤。

→ Testing

Testing 的時候，會發現吐出的中文句子在前面的字特別容易出現常見的字，如”我們”、”你們”之類的，不管是輸入什麼音訊。

```
[['我 們 都 不 可 以 了 ']
 ['我 們 不 可 以 去 找 ']
 ['我 不 會 說 ']
 ...,
 ['好 了 ']
 ['我 不 可 以 讓 妳 去 ']
 ['你 們 現 在 是 我 的 ']]
```

→ Conclusion

Seq2Seq 較 Retrieval 模型有意義，因為它是直接吐句子出來，並非像 retrieval model 要有預先的選項才可以做配對，是一個字一個字對應目前的輸入來回答適當的句子。

■ Retrieval:

→ Data Preprocessing

要產生出對應正確的句子與對應錯誤的句子，訓練出來的 model 才可以選擇選項，不然如果只有給 model 對應正確的句子，他無法判讀何謂對應錯誤的句子，這是 Retrieval model 相當重要的一個資料預處理環節。

→ Training

1. 語音與中文字在跑 LSTM 的時候，記得 activation function 要使用含有 output 為負數的 function，因為之後要做 dot，如果正數跟正數做 dot，出來一定是正數，之後再經過 sigmoid function，出來的值一定會>0.5，而這不是我們所預期的，我們預期的是 dot 出來要介於 0~1 之間，如此才與我們的 label 相符合。

2. LSTM 的 go_backwards 要使用 True，原因是因為我們猜測如果直接使用 forwards 會受到後面補 0 的影響，造成一開始讀取的音符或中文字被改變，而如果使用 go_backwards 先將補 0 的地方看完，再來看之後的句子，應該會等於直接看整個句子而不受 0 所影響，而經過 kaggle 上傳證實，go_backwards 確實比 forwards 的 performance 要來的好許多。

forwards：Kaggle Public：**0.27300**
go_backwards：Kaggle Public：**0.48700**

→ Testing
將語音與 4 個選項分別丟入 model 中，output 最高的即為相對應的選項，另外我們發現只用一個 model 無法超過 strong，因此我們採用 ensemble 的方式，將兩個 model predict 出來的值做相加/2，取最高的值當作最後的選項預測。

without ensemble： Kaggle Public：**0.48700**
with ensemble： Kaggle Public：**0.52300**

→ Conclusion
Retrieval model 比 Seq2Seq 來的簡單很多，但這只能回答選擇題，如果事先沒有選項則無法給出相對應的台語中文翻譯，這是 Retrieval model 的缺點。另外，FINAL 好累，但終於寫完了!!!