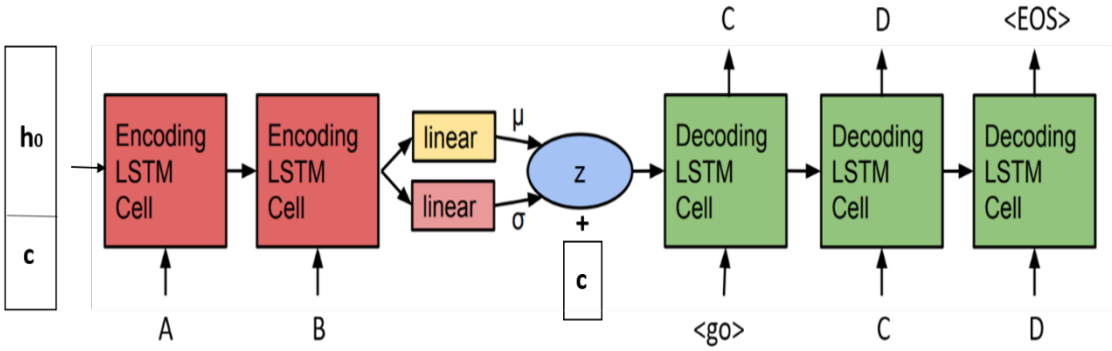
Lab3: Conditional Sequence-to-sequence VAE

系所：多媒體工程研究所
學號：0756616
姓名：周冠伶

1. Introduction

本次實驗需完成深度學習 RNN 網路的延伸架構 Conditional Sequence-To-Sequence Variational Auto-Encoder，並且以四種英文動詞型態(現在簡單式、第三人稱、過去簡單式、現在進行式)之間的互相轉換作為目標。

簡易的架構如下圖，大致上為 Input Layer、Encoder Layer、Variational Auto-Encoder Layer、Decoder Layer。在 Input Layer 將輸入原始資料編碼(h0)和原始狀態(c)編列成同一 Vector 後，逐一將其作為 Encoder Layer 的輸入，資料會被壓縮成 Latent Code；將 Latent Code 經過 VAE Layer 轉換的結果(z)、目標狀態(c)和 Flag(<go>)編列成同一 Vector 後，逐一將其作為 Decoder Layer 的輸入，並將輸出解碼即可得到轉換目標。



2. Implementation details

A. Describe how implement model

```
class customClass:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        self.index2word = {0: "SOS", 1: "EOS"}
        self.n_words = 2  # 字典

    # 新增單字
    def addSentence(self, sentence):
        for word in sentence.split(' '):
            self.addWord(word)

    # 更新字典
    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
        else:
            self.word2count[word] += 1
```

自定義的字典類別 customClass：包含文字轉換成編碼、編碼轉換成文字、新增單字等功能。

```
# dataloader
# select file and set fileName
MAX_LENGTH = 30
Pairs = []

root = tkinter.Tk()
root.withdraw()
root.wm_attributes("-topmost", 1)
fileName = filedialog.askopenfilename(
    parent=root,
    title="Choose Data File",
    filetypes=[("Data", "*.txt")],
    multiple=False)

# read fileData
fileData = open(fileName, 'r')
InputClass = customClass('inputClass')
OutputClass = customClass('targetClass')
for tenseData in fileData.readlines():
    tense = tenseData.replace('\n', '').split(' ')
    for i in range(len(tense)):
        data = tense[i]
        Pairs.append([data, data])
        InputClass.addSentence(data)
        OutputClass.addSentence(data)
#         if len(data) > MAX_LENGTH:
#             MAX_LENGTH = len(data)
fileData.close()

# MAX_LENGTH += 2
# print('MAX_LENGTH', MAX_LENGTH)
print('[[', InputClass.name, ',', OutputClass.name, ']]')
Pairs
```

自定義的 Dataloader：從檔案中讀取資料後放入對應的 customClass 中，資料型態為[[原始資料，目標資料]]。

```
# 轉換資料型態成Model可以使用的Tensor
def pairToTensor(pair):
    # 輸入完整句子(pair) >>> je suis dans le petrin .
    # 切割並轉換成字典編碼 >>> ['je', 'suis', 'dans', 'le', 'petrin', '.']
    # 加入EOS標籤，轉換型態(list->Tensor)，並Reshape((N)->[N*1]) >>> [6, 11, 296, 44, 972, 5]
    inputTensor = [InputClass.word2index[word] for word in pair[0].split(' ')]
    inputTensor.append(EOS)
    inputTensor = torch.tensor(
        inputTensor, dtype=torch.long, device=device).view(-1, 1)

    targetTensor = [
        OutputClass.word2index[word] for word in pair[1].split(' ')
    ]
    targetTensor.append(EOS)
    targetTensor = torch.tensor(
        targetTensor, dtype=torch.long, device=device).view(-1, 1)
    pairTensor = (inputTensor, targetTensor)

    return pairTensor
```

使用 Encoder 之前，需先將資料做處理：讀取單字、切割分字成逐個字母、字母轉換成編碼、加入 EOS 標籤、型態轉換(list 到 Tensor)、Reshape。

```
# conditions
conditions = torch.Tensor(numpy.zeros((4, 4)))
for i in range(len(conditions)):
    for j in range(len(conditions[i])):
        conditions[i][i] = 1
conditions
```

後續架構中會使用到的狀態 c：使用 One-Hot 方式進行編碼。

```python
# 編碼器：
# 輸入：一次一個字母(sourceTense) & 前一個隱藏資訊
# 輸出：壓縮後的資料 & 隱藏資訊
class EncoderRNN(torch.nn.Module):
    def __init__(self, inputSize, hiddenSize):
        super(EncoderRNN, self).__init__()
        self.hiddenSize = hiddenSize

        self.embedding = torch.nn.Embedding(inputSize, hiddenSize)
        self.gru = torch.nn.GRU(hiddenSize, hiddenSize)

        self.linear = torch.nn.Linear(self.hiddenSize + 4, self.hiddenSize)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)

        output, hidden = self.gru(embedded, hidden)
        return output, hidden

    # concatenate the condition part with the initial hidden part as input
    def initHidden(self, condition):
        initHidden = torch.zeros(1, 1, self.hiddenSize, device=device)
        initCondition = conditions[condition].view(1, 1, -1).to(device)

        initHidden = torch.cat((initHidden, initCondition), dim=2)
        initHidden = self.linear(initHidden)
        return initHidden
```

Encoder：用於將輸入資料進行編碼。Hidden vector 由前次的 Hidden vector 與給定的 Condition vector 定義，經由 Embedding 將每一個 Vector 進行編碼後，再使用 GRU 進行資料編碼與學習。

```python
# Latent Layer
class LatentLayer(torch.nn.Module):
    def __init__(self, hiddenSize, latentSize):
        super(LatentLayer, self).__init__()
        self.hiddenSize = hiddenSize
        self.latentSize = latentSize

        self.linear1 = torch.nn.Linear(hiddenSize + 4, 128)
        self.linear2 = torch.nn.Linear(128, 64)
        self.linear3 = torch.nn.Linear(64, latentSize)

        self.linear4 = torch.nn.Linear(latentSize, 64)
        self.linear5 = torch.nn.Linear(64, 128)
        self.linear6 = torch.nn.Linear(128, hiddenSize + 4)

    def forward(self, latent):
        # encoder hidden layer
        latent = self.linear1(latent)
        latent = self.linear2(latent)
        latent = self.linear3(latent)

        # VAE
        standardDeviation = latent
        standardDeviation = torch.exp(0.5 * standardDeviation)
        eps = torch.rand_like(standardDeviation)
        mu = latent
        z = eps.mul(standardDeviation).add_(mu)

        # decoder hidden layer
        latent = self.linear4(z)
        latent = self.linear5(latent)
        latent = self.linear6(latent)
        latent = torch.sigmoid(latent)

        return latent
```

Re-parameterization trick：用於 Encoder 與 Decoder 之間的 Latent layer，使用 Encoder 的輸出計算出 μ 和 σ 後，再將其二的線性組合作為 Decoder 之輸入。其中 KL Loss 需要用到此層。

```python
# Attention解碼器：將解碼器的輸入乘上一組權重，以幫助解碼器選擇正確的輸出類別
# soft輸入：依基準選擇部分解碼器輸出的隱藏資訊


class AttnDecoderRNN(torch.nn.Module):
    def __init__(self, hiddenSize, outputSize):
        super(AttnDecoderRNN, self).__init__()
        self.hiddenSize = hiddenSize
        self.outputSize = outputSize
        self.dropout_p = DROPOUT
        self.maxLength = MAX_LENGTH  # 最長的句子

        self.embedding = torch.nn.Embedding(self.outputSize, self.hiddenSize)
        self.attn = torch.nn.Linear(self.hiddenSize * 2, self.maxLength)
        self.attnCombine = torch.nn.Linear(self.hiddenSize * 2,
                                           self.hiddenSize)
        self.dropout = torch.nn.Dropout(self.dropout_p)
        self.gru = torch.nn.GRU(self.hiddenSize, self.hiddenSize)
        self.out = torch.nn.Linear(self.hiddenSize, self.outputSize)

        self.linear = torch.nn.Linear(self.hiddenSize + 4, self.hiddenSize)

    def forward(self, input, hidden, encoderOutputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        attnWeights = torch.nn.functional.softmax(
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
        attnApplied = torch.bmm(
            attnWeights.unsqueeze(0), encoderOutputs.unsqueeze(0))

        output = torch.cat((embedded[0], attnApplied[0]), 1)

        output = self.attnCombine(output).unsqueeze(0)

        output = torch.nn.functional.relu(output)
        output, hidden = self.gru(output, hidden)

        output = torch.nn.functional.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attnWeights

    # concatenate the condition part with the initial hidden part as input
    def initHidden(self, encoderHidden, condition):
        encoderHidden = encoderHidden
        initCondition = conditions[condition].view(1, 1, -1).to(device)

        initHidden = torch.cat((encoderHidden, initCondition), dim=2)
        initHidden = self.linear(initHidden)
        return initHidden
```

Decoder：用於將輸入資料進行解碼。步驟與 Encoder 相似，但是有使用到激勵函式。使用的是 Attention decoder，輸入編碼可以是非固定長度。

B. Specify the hyper-parameters

LEARNING_RATE = 0.05

TEACHER_FORCING_RATIO = 1.0

ITERS = 50000

HIDDEN_SIZE = 256

VOCABULARY_SIZE = 28

DROPOUT = 0.1

BATCH_SIZE = 1

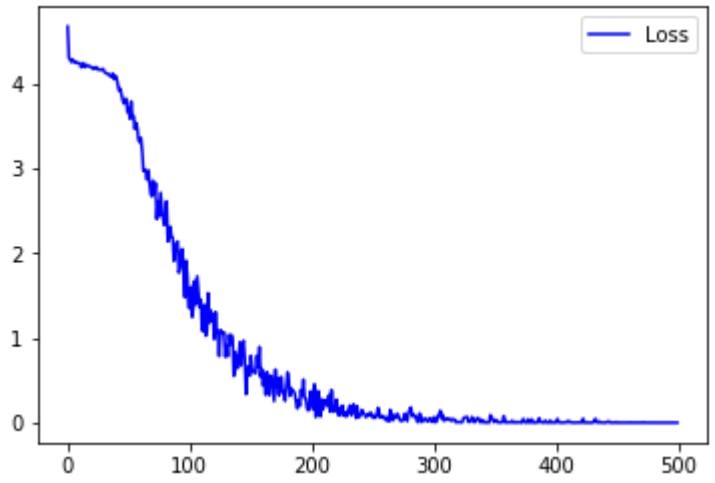
3. Results and discussion

結果為四種型態的 Encoder 與 Decoder，但是只能在同一形態中做 Encoder 與 Decoder，無法做不同型態的轉換。

A. Results of tense conversion and generation

```
True    exhorted            exhorted            exhorted
True    exhilarate          exhilarate          exhilarate
True    exhilarates         exhilarates         exhilarates
True    exhilarating        exhilarating        exhilarating
True    exhilarated         exhilarated         exhilarated
True    exculpate           exculpate           exculpate
True    exculpates          exculpates          exculpates
True    exculpating         exculpating         exculpating
True    exculpated          exculpated          exculpated
True    exasperate          exasperate          exasperate
True    exasperates         exasperates         exasperates
True    exasperating        exasperating        exasperating
True    exasperated         exasperated         exasperated
True    exacerbate          exacerbate          exacerbate
True    exacerbates         exacerbates         exacerbates
True    exacerbating        exacerbating        exacerbating
True    exacerbated         exacerbated         exacerbated
Accuracy: 100.0 %
```

B. Plot the Cross-entropy Loss, KL loss, BLEU-4 score curves during training and discuss the results according to setting of teacher forcing ratio, KL weight, and learning rate.



Cross-entropy Loss：EPOCH = 50000、REMIDER = 100、Teacher forcing ratio = 0.05

4. Discussion

以下是實作過程中，嘗試的內容：

A. 語言轉換與單字型態轉換：

兩者的資料的前處理方式不同，對於字典的編碼方式差異很大。語言轉換是單字對應單字，但是單字型態轉換是字母對應字母。

Training 的輸入資料為[Input data, Target data]，在語言轉換中兩者是不相同的，Input data 為輸入語言、Target data 為目標語言；在資料型態轉換中 Input data 與 Target data 是相同的，但是必需在使用 Encoder 和 Decoder 之前，將 Input data 與 Target data 與 Condition

vector 做結合，模型會透過 Condition vector 學習如何做不同型態的轉換。

在實作過程中，最主要的變動為資料的前處理方式，以及字典索引的內容。Training 時，一次會讀取一整個單字，但是會逐序訓練各個字母。

B. Encoder 與 Attention Encoder：

在 Attention Encoder 中，除了 Encoder 既有的輸出資料外，還會生成一個 Attention Weight，用於表示輸入資料維度的權重，使模型可以著重於高權重資料的解析。

兩者對於輸入資料的長度要求不同，一般的 Encoder 要求輸入資料必需長度一致，雖然可以使用 Padding 進行輸入資料的填充，但是過多的 Padding 會導致模型的訓練不容易；Attention Encoder 的輸入資料長度可以不相同，不會出現 Padding 過多的問題。

在實作中，雖然使用的是 Attention Encoder，但是為了方便撰寫程式碼，一樣有使用 Padding 對齊輸入資料長度。

C. Epoch 與 Loss：

在 Training 時，除了 Learning rate、Teacher forcing rate 等參數會影響 Training 結果外，隨機的初始化參數也會影響到其結果。

在實作過程中，大多數的情況下，Epoch < 10000 時 Loss 無法收斂、Epoch ~ 20000 時 Loss 收斂並趨於穩定、Epoch > 40000 時 Loss 不減反升；少部分時候會在 Epoch < 20000 時出現 Overfitting。

D. Tensor Shape：

Encoder 和 Decoder 的輸入是 Hidden 與 Condition 的串接，其中 Hidden 第一次為 Zero Vector、其它為前一次的輸出 Hidden，Condition 為該次輸入資料型態的 One-Hot Vector。

在實作過程中，將兩者做串接後會導致輸入資料的 Tensor Shape 改變，導致所有連動函式的參數都必需調整，解決方法可以選擇使用 Linear 函式改變輸入資料的維度，或是調整所有連動函式的參數。

E. GRU 與 LSTM：

GRU 與 LSTM 都是為了解決 RNN 因為過長的反向傳播導致梯度消失，無法長期記憶所衍伸出的模型。在 GRU 與 LSTM 中，都有引入 Gate 的想法，透過 Gate 控制該次是否要保留資訊，以維持模型儲存到較為重要的資訊。

兩者都透過 Gate 為資料做過濾，在 LSTM 中使用 Memory Cell 表示資料的傳輸過程，在 GRU 中透過 Hidden State 表示；LSTM 中的 Forget Gate、Input Gate 對應到 GRU 的 Update Gate。其中 GRU 因為參數較少、使 Tensor 的操作較少，因此訓練速度較 LSTM 快；但是在數據集較大的情況下，LSTM 的表現較佳。因此選擇使用 GRU 作為本次

實驗的 RNN 模型。

F. VAE 與 CVAE：

VAE 與 CVAE 都是 Auto-Encoder 的延伸應用，透過 Encoder-Decoder 的架構為資料進行編碼與解碼還原。VAE 會在 Latent Space 中應用到高斯分佈，為 Encoder 和 Decoder 做更多的變化；CVAE 也會為 Encoder 和 Decoder 做更多的變化，但是此變化不由高斯分佈產生，而是自行給定 Conditional 到模型中，讓使用者可以自行控制變化狀況。

在實作過程中，沒有使用 Condition 時，可以做到同一形態做 Encoder 和 Decoder；加入 Condition 到 Hidden 時，會導致 Loss 無法下降，可能原因為 Encoder 和 Decoder 中間層佈局錯誤、Encoder 和 Decoder 中間的 Latent Space 計算錯誤、或是 Hidden 與 Condition 的串接方式錯誤等。

G. Batch Size、Padding 與 End of String：

輸入資料長度不足時，會使用 Padding 標籤為資料填充長度。在輸入資料時，雖然會包含 Padding 標籤的資訊，但是模型輸出的結果不會包含 Padding 標籤。

在 Decoder 的資料中，使用 End of String 標籤作為資料結束的標籤。當 Batch Size 不為 1 時，End of String 標籤將會作為切割不同筆資料的分隔點；但是當 Batch Size 為 1 時，是否加入 End of String 標籤並不會影響模型的預測結果。

在實作中，Batch Size 為 1，且有加入 Padding、End of String 標籤。顯示模型預測結果時，不會有 Padding 標籤、但是必需另外處理 End of String 標籤。其中 Padding 與 End of String 的置入順序，不會影響預測結果。