深度學習 作業三

學號：312831002　姓名：廖健棚

LSTM

Encoder 程式碼

```python
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()
        self.hid_dim = hid_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout = dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src):
        embedded = self.dropout(self.embedding(src)) # Defaults to zeros if (h_0, c_0) is not provided.
        outputs, (hidden, cell) = self.rnn(embedded)
        return hidden, cell
```

Decoder 程式碼

```python
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()
        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(output_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout = dropout)
        self.fc_out = nn.Linear(hid_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, cell):
        input = input.unsqueeze(0)
        embedded = self.dropout(self.embedding(input))
        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))
        prediction = self.fc_out(output.squeeze(0))
        return prediction, hidden, cell
```

Seq2Seq 程式碼

```
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device
    def forward(self, src, trg, teacher_forcing_ratio = 0.5):
        batch_size = trg.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim
        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)
        hidden, cell = self.encoder(src)
        input = trg[0,:]
        for t in range(1, trg_len):
            output, hidden, cell = self.decoder(input, hidden, cell)
            outputs[t] = output
            teacher_force = random.random() < teacher_forcing_ratio
            top1 = output.argmax(1)
            if torch.equal(top1, torch.ones_like(top1)):
                break
            input = trg[t] if teacher_force else top1
        return outputs
```

my_data_loader 程式碼

```
def my_data_loader(data_file, batch_size=4):
    embed = dict(zip(string.ascii_lowercase, range(2, 28)))
    # data read from json file
    with open(data_file) as f:
        data = json.load(f)
    # mapping text from a-z to 1-27, 0 for SOS, 28 for EOS, 28 for PAD shared with EOS
    input_tensors = []
    target_tensors = []
    batch_pairs = []
    for item in data:
        input_ids = []
        # {'train': [A, B], 'target': [a]} => [A, a], [B, a]
        target_ids = [SOS_token] + [embed[c] for c in item["target"]] + [EOS_token]*(MAX_LENGTH-
                    len(item["target"])-1)
        target = torch.tensor(target_ids).view(21, 1)
        for text in item['input']: # multiple words
            input_ids= [SOS_token] + [embed[c] for c in text] + [EOS_token]*(MAX_LENGTH-len(text)-1)
            input_tensors.append(torch.tensor(input_ids).view(21, 1))
            target_tensors.append(target)
    for i in range(0, len(input_tensors)-batch_size, batch_size):
        if i+batch_size > len(input_tensors):
            break
        input_tensor = torch.cat(input_tensors[i:i+batch_size], dim=1)
        target_tensor = torch.cat(target_tensors[i:i+batch_size], dim=1)
        batch_pairs.append((input_tensor, target_tensor))
    return batch_pairs
```

LSTM 架構程解說

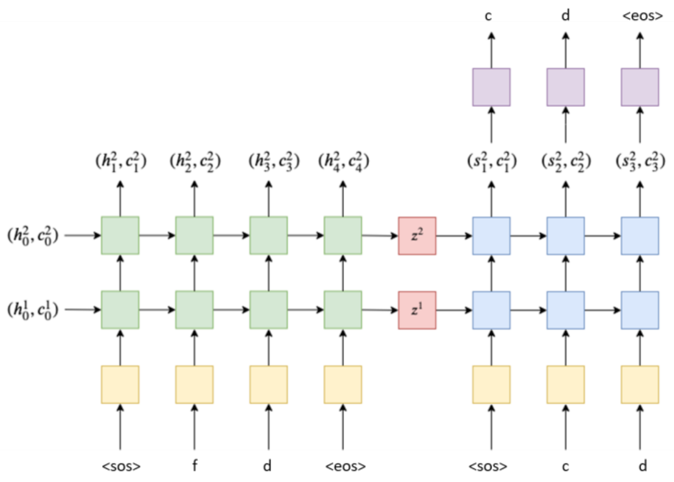


圖 1. Seq2Seq 架構: 左邊 Encoder、右邊 Decoder

Encoder 負責將輸入單字轉換成 latent 表示。其中使用 LSTM 網絡來處理輸入序列。

- embedding layer: 將輸入文字轉到設定的 256 維度，但通常是被拿來降低維度。
- dropout layer: 避免太過依賴資料
- LSTM layer: LSTM 是一種具有記憶單元的循環神經網絡（RNN），其特點是能夠有效地處理序列數據，其中 cell 負責記錄長期記憶，hidden 負責記錄短期記憶。
- forward 函數輸入單字經過 embedding 後，通過 LSTM 層得到輸出 outputs 為後的隱藏狀態 hidden、單元狀態 cell。

Decoder 負責根據 Encoder 的隱藏狀態和輸入生成單字。

- embedding layer: 與 Encoder 一樣
- LSTM layer: 與 Encoder 一樣，但 Decoder 會輸出 output
- Dropout layer: 與 Encoder 一樣
- fc_out: 全連接層生成最終預測
- forward 函數中，將輸入到 embedding 及 LSTM 處理，最後通過全連接層得到預測結果 prediction、新的 hidden 和 cell。

Seq2Seq 整合了 Encoder 和 Decoder

- encoder: 經過 encoder 後得到 hidden 及 cell 用於 Decoder
- decoder: seq2seq 會將 target 逐字丟入 decoder，decoder 將接收 Encoder 輸出的 hidden 及 cell 進行預測，output 將會是預測的下個字母，並且會將預測當作下個輸入。
- teacher_force: 透過 teacher_forcing 可以讓模型學得比較沒壓力，teacher_force 會固定機率給下個字母的答案，如果 teacher_forcing_rate=1，100%輸出都是答案，teacher_forcing_rate=0.5，50% 會是答案，這是會影響模型效率及結果。

Dataloader 將資料整理成 model 可以處理的格式，其中可以實作 batch_size 做加速

- 先將字母 a-z(1-27)編碼，其中需要考慮 sos(0)及 eos(28)
- 輸入的單字長度要一致，且不足最大長度(21)需要 padding(這邊使用 eos)
- 單字頭尾需要加上 sos 及 eos
- train.json 中的 input 有多個單字，需要將他分開
- 最後要將資料轉為 tensor

Train 訓練/Evaluate 驗證

資料批次輸入模型，訓練時會開啟 teacher_forcing，驗證時則會關掉。

在計算 bleu-4 時需要將 tensor 轉回文字，從 id 轉回 char

訓練結果

```
| Test Loss: 1.900 | Test Bleu: 0.919 | Accuracy: 0.8776

| New Test Loss: 2.240 | New Test Bleu: 0.642 | Accuracy: 0.4490
```

Transformer

Encoder 程式碼

```
class Encoder(nn.Module):
    def __init__(self, num_emb, hid_dim, n_layers, n_heads, ff_dim, dropout, max_length=100):
        super(Encoder, self).__init__()
        self.tok_embedding = nn.Embedding(num_emb, hid_dim)
        self.pos_embedding = PositionalEncoding(hid_dim, dropout, max_length, batch_first=True)
        self.layer = nn.TransformerEncoderLayer(d_model=hid_dim, nhead=n_heads, batch_first=True,
dim_feedforward=ff_dim, dropout=dropout)
        self.encoder = nn.TransformerEncoder(self.layer, num_layers=n_layers)
    def forward(self, src, src_mask):
        src = self.tok_embedding(src)
        src = self.pos_embedding(src)
        enc_src = self.encoder(src, src_key_padding_mask=src_mask)
        return enc_src
```

Decoder 程式碼

```
class Decoder(nn.Module):
    def __init__(self, num_emb, hid_dim, n_layers, n_heads, ff_dim, dropout, max_length=100):
        super(Decoder, self).__init__()
        self.tok_embedding = nn.Embedding(num_emb, hid_dim)
        self.pos_embedding = PositionalEncoding(hid_dim, dropout, max_length, batch_first=True)
        self.layer = nn.TransformerDecoderLayer(d_model=hid_dim, nhead=n_heads, batch_first=True,
dim_feedforward=ff_dim, dropout=dropout)
        self.decoder = nn.TransformerDecoder(self.layer, num_layers=n_layers)
    def forward(self, tgt, memory, src_pad_mask, tgt_mask, tgt_pad_mask):
        tgt = self.tok_embedding(tgt)
        tgt = self.pos_embedding(tgt)
        tgt = self.decoder(tgt, memory, memory_key_padding_mask=src_pad_mask, tgt_mask=tgt_mask,
tgt_key_padding_mask=tgt_pad_mask)
        return tgt
```

TransformerAutoEncoder 程式碼

```
class TransformerAutoEncoder(nn.Module):
    def __init__(self, num_emb, hid_dim, n_layers, n_heads, ff_dim, dropout, max_length=100,encoder=None):
        super(TransformerAutoEncoder, self).__init__()
        if encoder is None:
            self.encoder = Encoder(num_emb, hid_dim, n_layers, n_heads, ff_dim, dropout, max_length)
        else:
            self.encoder = encoder
        self.decoder = Decoder(num_emb, hid_dim, n_layers, n_heads, ff_dim, dropout, max_length)
        self.fc = nn.Linear(hid_dim, num_emb)
    def forward(self, src, tgt, src_pad_mask, tgt_mask, tgt_pad_mask):
        enc_src = self.encoder(src, src_pad_mask)
        out = self.decoder(tgt, enc_src, src_pad_mask, tgt_mask, tgt_pad_mask)
        out = self.fc(out)
        return out
```

SpellCorrectionDataset 程式碼

```
class SpellCorrectionDataset(Dataset):
    def __init__(self, root, split:str = 'train', tokenizer=None, padding:int=0):
        super(SpellCorrectionDataset, self).__init__()
        self.data = self.load_data(os.path.join(root, split+".json"))
        self.tokenizer = tokenizer
        self.padding = padding
    def load_data(self, file_name):
        inputs = []
        targets = []
        parse_data = []
        with open(file_name) as f:
            data = json.load(f)
        for d in data:
            inputs += [i for i in d['input']]
            targets += [d['target']]*len(d['input'])
        for i in range(len(inputs)):
            parse_data.append({'input': inputs[i],
                               'target': targets[i]})
        return parse_data
    def pad_sequence(self, sequence):
        if len(sequence) < self.padding:
            padded_sequence = sequence + [PAD_token] * (self.padding - len(sequence))
        else:
            padded_sequence = sequence[:self.padding]
        return padded_sequence
    def tokenize(self, text:str):
        ids = [SOS_token] + [self.tokenizer[c] for c in text] + [EOS_token]
        return ids
    def __len__(self):
        return len(self.data)
    def __getitem__(self, index):
        input_ids = torch.tensor(self.pad_sequence(self.tokenize(self.data[index]['input'])))
        target_ids = torch.tensor(self.pad_sequence(self.tokenize(self.data[index]['target'])))
        return input_ids, target_ids
```

mask 程式碼

```
def gen_padding_mask(src, pad_idx):
    return src.eq(pad_idx)
def gen_mask(seq):
    seq_len = seq.shape[-1]
    mask = torch.triu(torch.ones((seq_len, seq_len)), diagonal=1).bool()
    return mask
```

validation 程式碼

```
tgt_input = torch.full_like(tgt, PAD_token).to(device)
tgt_input[:,0] = SOS_token
...
src_pad_mask = gen_padding_mask(src, PAD_token)#generate the padding mask
tgt_pad_mask = gen_padding_mask(tgt_input, PAD_token)#generate the padding mask
tgt_mask = gen_mask(tgt).to(device)#generate the mask
pred = model(src, tgt_input, src_pad_mask, tgt_mask, tgt_pad_mask)
pred_idx = get_index(pred)[:, i]  # get the prediction idx from the model for the last token
tgt_input[:, i + 1] = pred_idx  # assign the prediction idx to the next token of tgt_inpu
```

Transformer 架構程解說

Encoder 經過 tok_embedding 及 pos_embedding 獲取 token 及 position 的資訊

- encoder: src_padding_mask 可以不去計算 pad_token

Decoder 跟 Encoder 很像，但會輸入 Encoder 輸出的 memory 及 memory mask

- decoder: memory 為 Encoder 的輸出，tgt 為 Decoder 的輸出目標

TransformerAutoEncoder 結合 Encoder 及 Decoder，為一種 Seq2Seq model

- 最後經過 fc layer，將輸出映射到字母維度

Mask 因為 Transformer 使用 attention 可以一次看到全部資訊，所以需要 mask 精準的把資料遮蓋，避免一次看到所有資訊

- gen_padding_mask: 將 pad_token 設為 True，不去計算輸入中的 padding
- gen_mask: 為一個上三角矩陣，模擬逐個預測字母

Validation

- 輸入為一個最大長度的 padding list，開頭為 sos
- 將 mask 丟入 model，並將每次 model 輸出的資訊放到 tgt_output

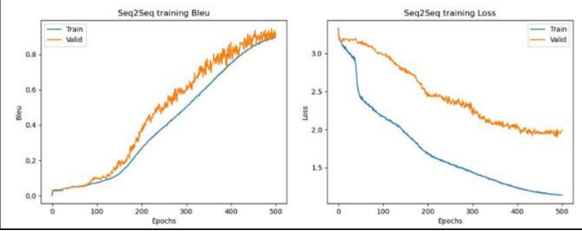
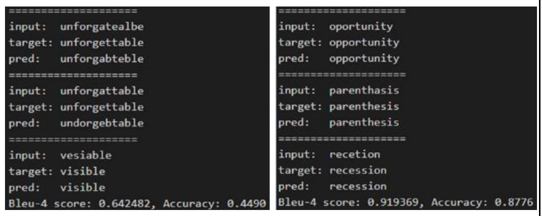
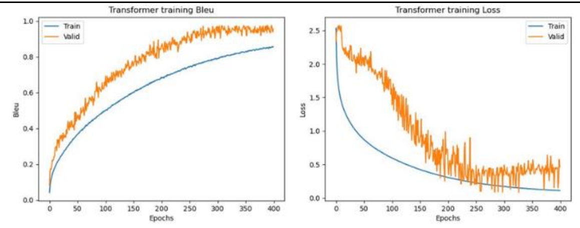
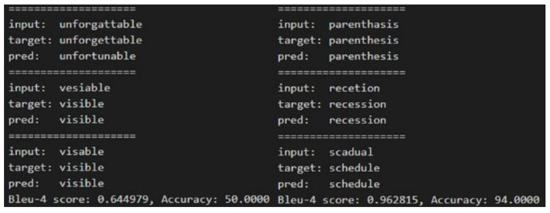
訓練結果

new_test_acc: 50.00 new_test_loss: 2.60 | [pred: appreciate target: appreciate] new_test_bleu4: 0.6450

test_acc: 94.00 test_loss: 0.08 | [pred: contented target: contented] test_bleu4: 0.9628

訓練紀錄及比較

LSTM 及 Transformer(TF)

LSTM	Loss/Bleu	new_test/test Log
Graph		
TF	Loss/Bleu	new_test/test Log
Graph		

```
====================
input:  unforgatealbe
target: unforgettable
pred:   unforgabteble
====================
input:  unforgattable
target: unforgettable
pred:   undorgebtable
====================
input:  vesiable
target: visible
pred:   visible
Bleu-4 score: 0.642482, Accuracy: 0.4490
```

```
====================
input:  oportunity
target: opportunity
pred:   opportunity
====================
input:  parenthasis
target: parenthesis
pred:   parenthesis
====================
input:  recetion
target: recession
pred:   recession
Bleu-4 score: 0.919369, Accuracy: 0.8776
```

```
====================                    ====================
input:  unforgattable                   input:  parenthasis
target: unforgettable                   target: parenthesis
pred:   unfortunable                    pred:   parenthesis
====================                    ====================
input:  vesiable                        input:  recetion
target: visible                         target: recession
pred:   visible                         pred:   recession
====================                    ====================
input:  visable                         input:  scadual
target: visible                         target: schedule
pred:   visible                         pred:   schedule
Bleu-4 score: 0.644979, Accuracy: 50.0000 Bleu-4 score: 0.962815, Accuracy: 94.0000
```

比較: LTSM 因為 RNN 架構，運算速度較慢，Transformer 因為 attention layer 可以一次看完所有輸入，速度會比較快，同時訓練的代次數也會比較少，在參數調整得當時，Transformer 的表現會比 LSTM 較佳

參考資料

i. Fong, Q. (2021, April 24). LSTM 運作原理與參數介紹，以天氣預測為例. Medium. https://qifong04.medium.com/lstm-運作原理與參數介紹-以天氣預測為例-2e1df792799e

ii. Bentrevett. (n.d.). Bentrevett/Pytorch-seq2seq: Tutorials on implementing a few sequence-to-sequence (seq2seq) models with pytorch and TorchText. GitHub. https://github.com/bentrevett/pytorch-seq2seq/tree/master

iii. OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/chat