NLP Assignment 1 - Named Entity Recognition

資工碩一 112526001 許仁覺

Data Preprocessing:

train_loader:

1. 從 train.txt 取得 data_list 和 label_list 用於後續組成句子，word_index_table, label_index_table 用於後續查表。
2. 組成句子後，將 sentence 透過 token2index 轉換成電腦可懂數字 index
3. 將成對打包好的 data_content 和 label_content 餵入 DataSet，並使用特徵前處理的 Dataloader 來做 dataloader，因為是訓練故設定 shuffle=True

```
def get_train_loader(train_file_path, embedding_vector, batch_size):
    train_data_list, train_label_list, word_index_table, label_index_table = get_word_index_table(train_file_path, embedding_vector)
    data_content = compose_sentences(train_data_list)
    label_content = compose_sentences(train_label_list)
    index_content = [token2index(sentence, word_index_table) for sentence in data_content]
    #build dataset and dataloader
    data = list(zip(data_content, label_content))
    train_dataset = NerDataset(data)
    collate_fn = partial(customed_preprocess, word2index=word_index_table, label2index=label_index_table)
    loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, collate_fn=collate_fn)
    return loader
```

extract_data_label:

1.將找出來 data 若是 stopword 話跳過不放入後面的訓練 data_list，並將所有 data 轉成小寫

```
def extract_data_label(input_list):
    '''extract data and label from input_list,
       if data in stopword, pass it, and lower the other data.
    '''
    data_list = []
    label_list = []
    for item in input_list:
        match = re.search(r'([^\t\n]+)\t([^\t\n]+)\n', item)
        if match:
            if match.group(1) in stopword:
                continue
            else:
                data_list.append((match.group(1)).lower())
                label_list.append(match.group(2))
        else:
            if item == '\n':
                data_list.append('\n')
                label_list.append('\n')
            else:
                print(f'Error: no data and label found: {item}')
    return data_list, label_list
```

token2index:

1.因為使用是 gensim 訓練好的 twitter-25 embedding，針對未見字簡單使用 twitter-25 一少見字 ""的 embedding index 做為替代

```
def token2index(data_list, word_index, use_lower=False):
    if use_lower:
        data_list = [word.lower() for word in data_list]
    #unkown word use rare case <span> to instead #twitter-25
    return [word_index[word] if word in word_index else word_index["<span>"] for word in data_list]
```

customed_preprocess:

1.將 x(dat)和 y(label)轉換成 index, 若遇到 batch 中需要補 pad 狀況，x 使用 twitter-25 一少見字'<lol>'的 index，y 使用最多的'O'的 index

```
def customed_preprocess(batch, word2index, label2index):
    x, y = zip(*batch) #x:tuple
    x = [token2index(sentence, word2index) for sentence in x]
    y = [token2index(sentence, label2index) for sentence in y]

    #pad x to same size in the batch, and convert x long tensor for rnn.pad_sequencce format
    #pad use the rare case <lol> instead of <pad>
    pad_token_idx = word2index['<lol>'] #twitter-25
    x = [torch.LongTensor(sentence) for sentence in x]
    x_pad = torch.nn.utils.rnn.pad_sequence(x, batch_first=True, padding_value=pad_token_idx)

    pad_label_idx = label2index['O']
    y = [torch.LongTensor(sentence) for sentence in y]
    y_pad = torch.nn.utils.rnn.pad_sequence(y, batch_first=True, padding_value=pad_label_idx)
    return x_pad, y_pad
```

###未完後面還有###

Model Architecture:

Class LSTM_NER:

1.建立使用 pretrain embedding 的 LSTM 模型，並最終可以輸出各類的機率

```python
class LSTM_NER(nn.Module):
    def __init__(self, hyperparameters, vocab_size, output_size, embedding_model):
        super(LSTM_NER, self).__init__()

        """ Instance variables """
        self.embed_dim = hyperparameters["embed_dim"]
        self.hidden_dim = hyperparameters["hidden_dim"]
        self.freeze_embeddings = hyperparameters["freeze_embeddings"]
        self.bidirectional = hyperparameters["bidirectional"]

        """ Embedding Layer
        """
        self.output_size = output_size
        self.embedding_model = embedding_model
        weights = torch.FloatTensor(self.embedding_model.vectors)
        self.embeds = nn.Embedding.from_pretrained(weights, padding_idx=0)
        if self.freeze_embeddings:
            self.embeds.requires_grad = False
        """ LSTM Layer
        """
        num_lstm_layers = 2
        self.lstm = nn.LSTM(input_size=self.embed_dim, hidden_size=self.hidden_dim, num_layers=num_lstm_layers, batch_first=True, bidirectional=self.bidirectional)

        if self.bidirectional:
            D = 2
        else:
            D = 1

        """ Output Layer
        """
        self.output_layer = nn.Linear(D*self.hidden_dim, self.output_size)
        """ Probabilities
        """
        if self.output_size == 1:
            self.probabilities = nn.Sigmoid()
        else:
            self.probabilities = nn.Softmax(dim=-1)
```

Forward:

1. 有比較 V1:正常 LSTM, V2:正常 LSTM+Dropout, V3:residul LSTM, 但相差不大，故最後用 V1 正常版
2. 特別的是做 Cross-entropy output 需轉換成(Batch_size, num_class, length), 需在最後用 permute 去轉換軸，若使用 view 去轉換 output 會造成內容有誤而無法收斂

```python
    def forward(self, inputs):
        """
        Let B:= batch_size
            D:= self.embed_dim
            H:= self.hidden_dim

        inputs: a (B, L) tensor of token indices
        """
        B, L = inputs.shape
        embeds = self.embeds(inputs)

        hidden = None
        lstm_output, hidden= self.lstm(embeds, hidden)

        #v1
        output = self.output_layer(lstm_output)
        #v2
        #use dropout to prevent overfitting
        # output = self.dropout(output)
        #v3
        # output1= self.fc1(lstm_output)
        # residual = output1 + embeds
        # output = self.fc2(residual)
        output = self.probabilities(output)
        #for cross-entropy format, cannot use view.(B, self.output_size, -1)
        output = output.permute(0, 2, 1)
        return output
```

Training process

train_flow:

取得訓練需要的 component: embedding model, stopword, dataloader, model, optimizer, schedular 後，Call train function 訓練。

```python
def train_flow(train_file_path, val_file_path, batch_size, epochs, learning_rate, model_hyperparameters, model_path):
    #get glove embedding model
    embedding_model = KeyedVectors.load_word2vec_format("./glove-twitter-25.txt")
    #get stopword
    get_stopword()

    #data preprocess and build dataloader
    loader = get_train_loader(train_file_path, embedding_model, batch_size)
    val_loader = get_val_loader(val_file_path, batch_size)

    #build model, optimizer, scheduler
    vocab_size = len(word_index_table)
    output_size = len(label_index_table)
    model = LSTM_NER(model_hyperparameters, vocab_size, output_size, embedding_model)

    # model.load_state_dict(torch.load(r"./weights/model_test_latest_0448.pth"))
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    scheduler = StepLR(optimizer, step_size=10, gamma=0.8)

    #start training
    train(loss_function, optimizer, model, loader, epochs, val_loader=val_loader, scheduler=scheduler, model_path=model_path)
```

train:

1.除了計算 loss 外，也計算 acc 和呈現 lr， 可以更好了解訓練狀況，並也計算 val part 了解模型真實表現，是否有 overfitting/underfitting 狀況。

```python
def train(loss_function, optimizer, model, loader, epochs, val_loader=None, scheduler=None, model_path="./weights/model_test.pth"):
    for epoch in range(epochs):
        train_num, train_correct = 0, 0
        total_loss, total_acc = 0, 0
        for batch_inputs, batch_labels in loader:
            optimizer.zero_grad()
            outputs = model(batch_inputs)
            loss = loss_function(outputs, batch_labels)
            loss.backward()
            optimizer.step()
            #summary
            total_loss += loss.item()
            predicted = torch.argmax(outputs, dim=1)
            train_correct += (predicted == batch_labels).sum().item()
            train_num += math.prod(batch_labels.shape)
        total_loss = total_loss / len(loader)
        total_loss = total_loss
        total_acc = train_correct / train_num
        if val_loader and (epoch+1) % 5 == 0:
            model = model.eval()
            total_val_loss, total_val_acc = evaluate(model, val_loader)
            model = model.train()
            print(f'epoch: {epoch}, train-loss: {total_loss:.5f}, train-acc: {total_acc:.3f}, val_loss: {total_val_loss:.5f}, val_acc: {total_val_acc:.3f}')
        else:
            print(f'epoch: {epoch}, train-loss: {total_loss:.5f}, train-acc: {total_acc:.3f}, lr: {optimizer.param_groups[0]["lr"]:.5f}')
        scheduler.step()
        torch.save(model.state_dict(), model_path)
```

訓練過程如下：

```
epoch: 0, train-loss: 0.31224, train-acc: 0.943, lr: 0.01000
epoch: 1, train-loss: 0.15134, train-acc: 0.957, lr: 0.01000
epoch: 2, train-loss: 0.13464, train-acc: 0.961, lr: 0.01000
epoch: 3, train-loss: 0.12061, train-acc: 0.964, lr: 0.01000
epoch: 4, train-loss: 0.11074, train-acc: 0.965, val_loss: 0.15334, val_acc: 0.955
epoch: 5, train-loss: 0.10205, train-acc: 0.965, lr: 0.01000
epoch: 6, train-loss: 0.09783, train-acc: 0.966, lr: 0.01000
epoch: 7, train-loss: 0.09090, train-acc: 0.967, lr: 0.01000
epoch: 8, train-loss: 0.08543, train-acc: 0.968, lr: 0.01000
epoch: 9, train-loss: 0.08089, train-acc: 0.969, val_loss: 0.14873, val_acc: 0.952
```

Loss:

1. 因為訓練資料為不平衡資料(大量的 O)，故除使用 normal 的 cross-entropy 外
2. 也探索 cross-entropy+class weight，O 類權重 1，其他類權重為 100
3. Focal loss 會改善一些，但未到很多，最終選擇 Focal loss

```
def loss_function(input, target):
    """focal loss
    """
    gamma = 2
    logpt = torch.log(input)
    pt = torch.exp(logpt)
    logpt = (1-pt)**gamma * logpt
    loss = F.nll_loss(logpt, target)
    return loss

def loss_function_normal(batch_outputs, batch_labels):
    """cross entropy loss
    """
    log_softmax_outputs = torch.log(batch_outputs)
    loss = F.nll_loss(log_softmax_outputs, batch_labels)
    return loss

def loss_function_class_weight(batch_outputs, batch_labels):
    """cross entropy loss with class weight
    """
    log_softmax_outputs = torch.log(batch_outputs)
    class_weights = torch.FloatTensor([100 for i in range(20)] +[1] )
    loss = F.nll_loss(log_softmax_outputs, batch_labels, weight=class_weights)
    return loss
```

Test process

result_parsing:

1. 讀取 data_list 和 result 的結果，並拼出最終 test-submit 的 string 格式
2. 若遇到超過 length 長度的 result 跳過，若遇到如果是 stopword 話，自動補'O'，增加準確率

```
def result_parsing(data_list, result, lengths, is_stopword_list):
    text = ""
    index_label_table = {index:label for label, index in label_index_table.items()}
    idx = 0
    for i,batch_result in enumerate(result):
        for j,word_result in enumerate(batch_result):
            if j >= lengths[i]:
                #pass the <pad>
                break
            predict_name = index_label_table.get(word_result)

            if is_stopword_list[i][j] == True:
                predict_name = 'O'
            text += f'{data_list[i][j]}\t{predict_name}\n'
        text += '\n'
    return text
```

最終使用 dev.txt 去測試：

```
λ python conlleval.py < dev_output.txt
processed 17261 tokens with 661 phrases; found: 593 phrases; correct: 166.
accuracy:  23.67%; (non-O)
accuracy:  93.49%; precision:  27.99%; recall:  25.11%; FB1:  26.48
          company: precision:  29.41%; recall:  25.64%; FB1:  27.40  34
         facility: precision:   8.06%; recall:  13.16%; FB1:  10.00  62
          geo-loc: precision:  33.09%; recall:  38.79%; FB1:  35.71  136
            movie: precision:   0.00%; recall:   0.00%; FB1:   0.00  0
      musicartist: precision:   0.00%; recall:   0.00%; FB1:   0.00  0
            other: precision:   0.88%; recall:   0.76%; FB1:   0.81  114
           person: precision:  42.02%; recall:  58.48%; FB1:  48.90  238
          product: precision:   0.00%; recall:   0.00%; FB1:   0.00  1
       sportsteam: precision:  62.50%; recall:   7.14%; FB1:  12.82  8
           tvshow: precision:   0.00%; recall:   0.00%; FB1:   0.00  0
```