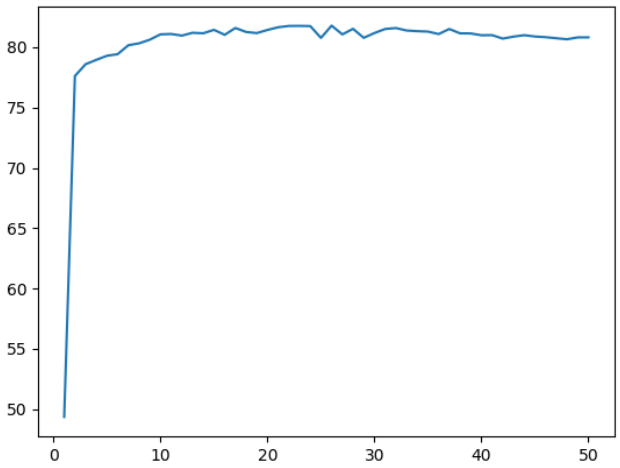
學號：B06902006　　系級：資工三　　姓名：王俊翔

1. 請說明你實作的 RNN 的模型架構、word embedding 方法、訓練過程 (learning curve)和準確率為何？

模型架構如下圖:

```
class LSTM_Net(nn.Module):
    def __init__(self, embedding, embedding_dim, hidden_dim, num_layers, dropout=0.5, fix_embedding=True):
        super(LSTM_Net, self).__init__()
        # 製作 embedding layer
        self.embedding = torch.nn.Embedding(embedding.size(0),embedding.size(1))
        self.embedding.weight = torch.nn.Parameter(embedding)
        # 是否將 embedding fix 住，如果 fix_embedding 為 False，在訓練過程中，embedding 也會跟著被訓練
        self.embedding.weight.requires_grad = False if fix_embedding else True
        self.embedding_dim = embedding.size(1)
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.dropout = dropout
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=num_layers, batch_first=True, dropout=dropout)
        self.classifier = nn.Sequential( nn.Dropout(0.2),
                                         nn.Linear(hidden_dim, 1024),
                                         nn.Dropout(dropout),
                                         nn.PReLU(),
                                         nn.Linear(1024, 1),
                                         nn.Sigmoid() )
    def forward(self, inputs):
        inputs = self.embedding(inputs)
        x, _ = self.lstm(inputs, None)
        # x 的 dimension (batch, seq_len, hidden_size)
        # 取用 LSTM 最後一層的 hidden state
        x = x[:, -1, :]
        x = self.classifier(x)
        return x
```

```
class LSTM_Net_BI(nn.Module):
    def __init__(self, embedding, embedding_dim, hidden_dim, num_layers, dropout=0.5, fix_embedding=True):
        super(LSTM_Net_BI, self).__init__()
        # 製作 embedding layer
        self.embedding = torch.nn.Embedding(embedding.size(0),embedding.size(1))
        self.embedding.weight = torch.nn.Parameter(embedding)
        # 是否將 embedding fix 住，如果 fix_embedding 為 False，在訓練過程中，embedding 也會跟著被訓練
        self.embedding.weight.requires_grad = False if fix_embedding else True
        self.embedding_dim = embedding.size(1)
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.dropout = dropout
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=num_layers, batch_first=True, dropout=dropout, bidirectional=True)
        self.classifier = nn.Sequential( nn.Dropout(0.2),
                                         nn.Linear(hidden_dim*2, 1024),
                                         nn.Dropout(dropout),
                                         nn.PReLU(),
                                         nn.Linear(1024, 1),
                                         nn.Sigmoid() )
    def forward(self, inputs):
        inputs = self.embedding(inputs)
        x, _ = self.lstm(inputs, None)
        # x 的 dimension (batch, seq_len, hidden_size)
        # 取用 LSTM 最後一層的 hidden state
        x = x[:, -1, :]
        x = self.classifier(x)
        return x
```



上下皆有使用，word embedding 的方法則是使用 word2vec(size = 250, window = 5, min_count = 5, iter = 10, sg = 1)，learning rate 起始為 0.0001，後來根據準確率去做調整，運用 scheduler (optim.lr_scheduler.ReduceLROnPlateau)，準確率一降就調整 learning rate，得出的曲線如上圖(x : epoch, y : accuracy)

2. 請比較 BOW+DNN 與 RNN 兩種不同 model 對於"today is a good day, but it is hot"與"today is hot, but it is a good day"這兩句的分數(過 softmax 後的數值)，並討論造成差異的原因。

	Sen1	Sen2
BOW+DNN	0.7801814675331116	0.7801814675331116
RNN	0.08650051057338715	0.9969922304153442

BOW 只在乎裡面的字是甚麼，因此會發現這兩句的文字其實是一樣的，所以輸出的結果也會是一樣，至於 RNN 會考慮一個字與前後的關係，因此在這種情況下 RNN 的判斷應該較為準確。

3. 請敘述你如何 improve performance（preprocess、embedding、架構等等），並解釋為何這些做法可以使模型進步，並列出準確率與 improve 前的差異。（semi supervised 的部分請在下題回答）

Embedding 的部分拿所有的 data 一起做(包含 unlabeled 的)，這樣的原因是為了讓 embedding 變的更 general，且之後有做 semi，架構的部分嘗試了單向的 lstm 三層及雙向的兩層，參數的增加也讓準確率有所上升，另外我在 linear 層也多加了一層(由上圖可知)，也是因為參數增加有所進步，preprocessing 基本上照著助教的 code，embedding 的參數於第一題以說明使用 stepgram 的原因是 twitter 有許多奇奇怪怪少量的字，使用 stepgram 會比 CBOW 好，此外，最後做了多個 model 的 ensemble，效果蠻顯著的。

4. 描述你的 semi-supervised 方法是如何標記 label，並比較有無 semi-supervised training 對準確率的影響並試著探討原因（因為 semi-supervise learning 在 labeled training data 數量較少時，比較能夠發揮作用，所以在實作本題時，建議把有 label 的 training data 從 20 萬筆減少到 2 萬筆以下，在這樣的實驗設定下，比較容易觀察到 semi-supervise learning 所帶來的幫助）。

- 沒減少 data 時做 semi 其實就有不錯的準確率增加了(增加了 0.5%)，有嘗試過 70%以上 30%以下標記及 80%以上 20%以下標記，兩種準確率差不多。
- 將 data 減少至 20000 筆且在 85%以上 15%以下做標記，做完 semi 後，準確率在同一個 validation set 增加了 0.2%，我認為由於我是對同一個 validation set 產生的結果，導致就算加了 semi 後，那些辨認的不好的仍沒有得到好的處裡，上一段的就有重新切 validation，效果就挺不錯的。這裡的 validation data 拿了 20000 筆，在同一份 validation 下效果就不顯著。