Individual Final Report

Xiao Qi

1. Introduction

The objective of the final project is to classify emotions after listening to a speech or a short voice note and do a comparative analysis of the different dynamic models like RNN or LSTM and a convolutional model like CNN.

2. Description of my individual work

(1) pre-processing

I changed the original pre-processing code to make it more clear and fast to run. Meanwhile, oversampling and data augmentation are implemented in a concise way in my code. I also helped to debug my team members' codes to make them run successfully.

(2) Model Built

I built the Elman RNN model individually, tried different methods, like changing input size, learning rate, optimizers, ReduceLROnPlateau, to improve the results, and optimized the code by using functions.

(3) Report and slide writing

Finished the report and slides with team members.

3. Describe the portion of the work

(1) Oversampling and data augmentation for new added data

```
audio_tab_new = audio_tab.copy()
audio_tab_new['origin'] = False

audio_natural = audio_tab_new[(audio_tab_new['emotion'] == 0)].copy()

pd_audio = pd.concat([audio_tab, audio_natural, audio_tab_new, audio_natural, audio_tab_new, audio_natural, audio_tab_new, audio_natural])
```

```
if pd_audio.iloc[i].origin is False:
    fn1_data = transform_audio(y, fns, sample_rate)
    y = transform_audio(fn1_data, fns, sample_rate)
```

(2) Padding

```
# padding
padded_list_of_lists = []
for row in log_spectrogram:
    padding_length = max_row_length - len(row)
    if padding_length > 0:
        # avg_value = np.mean(row[4:])
        avg_value = 0
        padded_row = np.pad(row, (0, padding_length), mode='constant', constant_values=avg_value)
        padded_list_of_lists.append(padded_row[4:])
    else:
        padded_list_of_lists.append(row[4:])
```

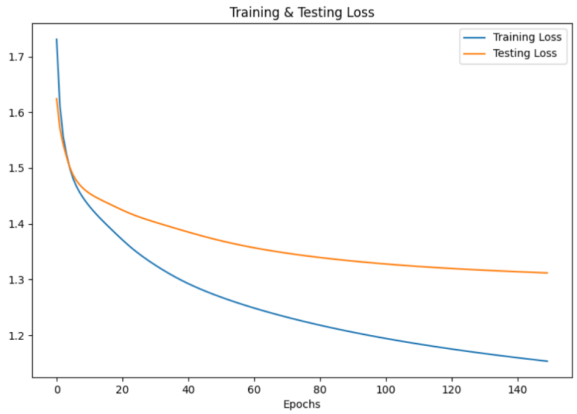
(3) Model built

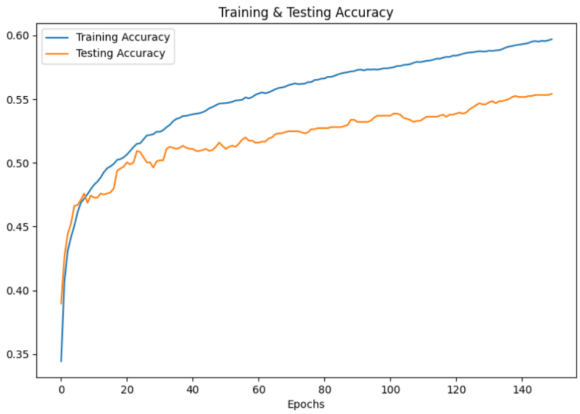
```
class RNN(nn.Module):
    # implement RNN from scratch rather than using nn.RNN
    Akshat Saini +1
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o1 = nn.Linear(input_size + hidden_size, 64)
        self.i2o2 = nn.Linear(64, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    Qi-Xiao +1
    def forward(self, input_tensor, hidden_tensor):
        combined = torch.cat((input_tensor, hidden_tensor), 1)

        hidden = self.i2h(combined)
        output = self.i2o1(combined)
        output = self.i2o2(output)
        output = self.softmax(output)
        return output, hidden

    Akshat Saini +1
    def init_hidden(self):
        return torch.zeros(1, self.hidden_size)
```

(4) Train and test

```
for epoch in range(1, max_epochs + 1):

    train_loss, steps_train = 0, 0
    test_loss, steps_test = 0, 0
    pred_train = []

    for i in range(X_train.size(0)):
        category_tensor = y_train[i].reshape(1).type(torch.LongTensor)
        line_tensor = X_train[i]
        output, loss = train(line_tensor, category_tensor)
        output_item = torch.argmax(output).item()
        pred_train.append(output_item)

        train_loss += loss
        steps_train += 1

    avg_train_loss = train_loss / steps_train

    pred = []
    for i in range(X_test.shape[0]):
        output, loss = predict(X_test[i], y_test[i].reshape(1).type(torch.LongTensor))
        output_item = torch.argmax(output).item()
        pred.append(output_item)

        test_loss += loss
```

4. Results





In our project, we set the epoch as 150 to train the model. For the training process, we can find that the Elman RNN's accuracy is very low. Before the data augmentation, the test set accuracy is only 39% on vanilla dataset. After doing data augmentation, the test accuracy increases to 55%. Compared to the Lstm and Cnn model, the accuracy of Elman RNN is still very slow.

5. Summary and conclusions

The project focused on the classification of Ryerson Audio-Visual Database of Emotional Speech. We built three models separately to do this task and compare those three models. It shows that the CNN model is most suitable for this task, which has a test accuracy of 0.85. And the Elman RNN model has the least accurate result between the three models.

This could be because of its simple architecture and also because this particular dataset did not require the benefits or the usage of a feedback loop or delay, which models like RNN and LSTM can provide.

6. References

https://paperswithcode.com/dataset/ravdess

https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

https://github.com/gabrielloye/RNN-walkthrough/blob/master/main.ipynb