# ADL Hw1 Report

學號：B08502141　　姓名：石旻翰　　系級：電機四

# 1 Data Processing

(a). I use the sample code from the link of Github, and following is the details of the implementation of data processing for Intent Classification and Slot Tagging, respectively. The process can be broken down to the following steps:

1. Collect all words in train set and validation set.
2. Collect labels for Intent and Slot Tagging, respectively.
3. Link the labels to the indices, making training and testing process more convenient.
4. Build embeddings with:
    (1). Filter widely used words.
    (2). Tokenize the words into integers.
    (3). Calculate Glove embeddings based on the tokenized words.

(b). The pretrained embedding I used is GloVe provided by TA.

# 2 Intent Classification

## 2.1 Model

My model in this part consists of embedding layer (**Embedding**), recurrent network(including RNN, GRU, or LSTM) layer, and classifier layer. Let the input be **x**, and the output .

We first get **x** through embedding layer.

$$x_{embed} = \textbf{Embedding}(x) \tag{2.1}$$

Next, we have to get $x_{embed}$ through recurrent network layer. The recurrent network layer I used is bidirectional GRU, and hidden size is set to 512, so the output shape is [N, l, 1024], where N is the number of sample in each batch, and l is the max length which is set to 128 in my implementation. For each layer, the reset $r_t$, the new gate $n_t$, and the update $z_t$ are calculated by the following equations:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \tag{2.2}$$

$$n_t = tanh(W_{in} + b_{in} + W_{in}h_{(t-1)} + b_{hz}) \tag{2.3}$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{iz}h_{(t-1)} + b_{hz}) \tag{2.4}$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)} \tag{2.5}$$

Since the output shape is (N, l, 1024) now, the $x_{GRU}$, which has to be in shape of (N, 1024), have to be calculated by: ($x_l$ is the feature in last phase of the layer)

$$x_{GRU} = \Sigma_{i=0}^{l-1} x_l \tag{2.6}$$

Next, the classifier layer is as following:

$$x_{N,150} = \text{LeakyReLU}(W_{1024,1024}x_{GRU} + b)W_{1024,150} + b \tag{2.7}$$

Lastly, the prediction is obtained by:

$$\hat{y} = \text{argmax}(x_{N,150}) \quad [2.8]$$

## 2.2 Performance

| Train Accuracy | Validation Accuracy | Public Score | Private Score |
|---|---|---|---|
| 1 | 0.85 | 0.88266 | 0.88488 |

Table 1: All Performance of Intent Classification

## 2.3 Loss Function & Optimizer

Loss Function: I use **CrossEntropyLoss** with L2 regularization ($\lambda = 10^{-5}$ )provided by pytorch. The loss function is: (W denotes the parameters of model)

$$\mathcal{L}(x, y) = \frac{1}{N}[-\Sigma_{n=1}^{N} ln(\frac{exp(\hat{y_{n,y_n}})}{\Sigma_{i=1}^{150} exp(\hat{y_{n,i}})}) + \lambda W^T W] \quad [2.9]$$

Optimizer: I use **AdamW** provided by pytorch, and the learning rate is set to 0.001, the batch size is 128.

# 3 Slot Tagging

## 3.1 Model

My model in this part is basically similar to the model of Intent Classification. The only difference is the classifier layer. The classifier in this part is:

$$x_{N,10} = \text{LeakyReLU}(W_{1024,512} x_{GRU} + b) W_{512,10} + b \quad [3.1]$$

And the prediction can be obtained by:

$$\hat{y} = \text{argmax}(x_{N,10}) \quad [3.2]$$

## 3.2 Performance

| Train Accuracy | Validation Accuracy | Public Score | Private Score |
|---|---|---|---|
| 0.998 | 0.785 | 0.72064 | 0.74169 |

Table 2: All Performance of Slot Tagging

## 3.3 Loss Function & Optimizer

Loss Function: I use **CrossEntropyLoss** with L2 regularization ($\lambda = 10^{-5}$ )provided by pytorch. So the loss function is: (W denotes the parameters of model)

$$\mathcal{L}(x, y) = \frac{1}{N}[-\Sigma_{n=1}^{N} ln(\frac{exp(\hat{y_{n,y_n}})}{\Sigma_{i=1}^{9} exp(\hat{y_{n,i}})}) + \lambda W^T W] \quad [3.3]$$

Optimizer: I use **AdamW** provided by pytorch, and the learning rate is set to 0.001, the batch size is 128.

# 4 Sequence Tagging Evaluation

## 4.1 Evaluation Result

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| date | 0.79 | 0.79 | 0.79 | 206 |
| first_name | 0.91 | 0.84 | 0.88 | 102 |
| last_name | 0.83 | 0.56 | 0.67 | 78 |
| people | 0.73 | 0.75 | 0.74 | 238 |
| time | 0.86 | 0.86 | 0.86 | 218 |
| micro avg | 0.81 | 0.78 | 0.80 | 842 |
| macro avg | 0.82 | 0.76 | 0.79 | 842 |
| weighted avg | 0.81 | 0.78 | 0.79 | 842 |

Figure 1: Result of sequence evaluation on Slot validation set

## 4.2 Evaluation Metrics in seqeval

TP: True Positive; TN: True Negative; FP: False Positive; FN: False Negative

1. Precision:

$$\frac{TP}{TP + FP} = \frac{TP}{\text{predicted positive}} \tag{4.1}$$

2. Recall:

$$\frac{TP}{TP + FN} = \frac{TP}{\text{actually positive}} \tag{4.2}$$

3. F1-score:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

4. Token Accuracy: It is not necessary that all tokens have to be correct in a sample. However, it would still count if a token is done so.

$$\frac{1}{N\Sigma_{i=1}^{N}|y_i|}\Sigma_{i=1}^{N}\Sigma_{j=1}^{|y_i|}\left(y_{i,j} == \hat{y}_{i,j}\right) \tag{4.4}$$

5. Joint Accuracy: Sample is considered correct iff all tokens are correct.

$$\frac{1}{N}\Sigma_{i=1}^{N}\left(y_i == \hat{y}_i\right) \tag{4.5}$$

# 5 Different Configuration

## 5.1 Intent Classification

From Fig. 2, we can see that in the same number of layer, and other hyperparameters are all the same, GRU outperforms the other two recurrent network(RNN, LSTM).

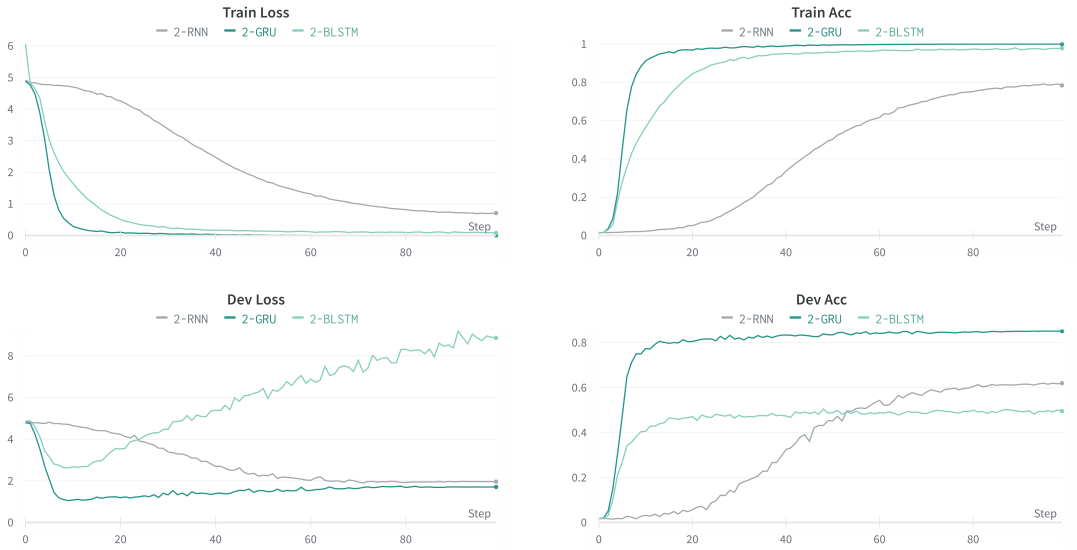


Figure 2: Different types of recurrent network

## 5.2 Slot Tagging

From Fig. 3, we can see that adding the number of layer did not help a lot to further improve the performance of GRU in this task. However, though 2-layer GRU seems better in train and dev set, when I put the result of inference on test set to kaggle, the result of 4-layer GRU is the better one.

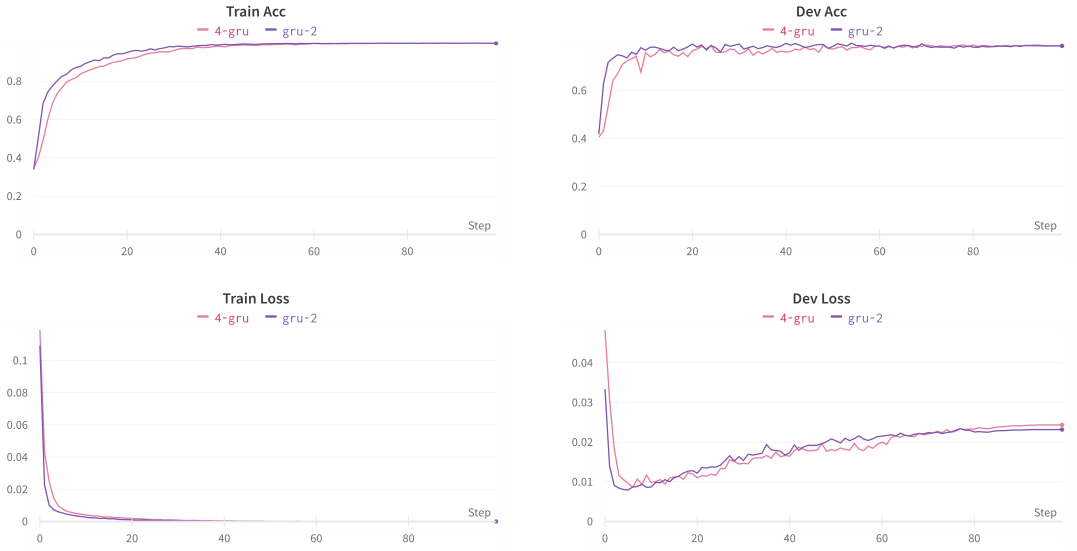


Figure 3: Different numbers of layer of GRU

# References

[1] https://github.com/ntu-adl-ta/ADL21-HW1