*tags: **ADL***

HW1

Q1

a. In the sample code, the `Vocab` class in the `utils.py` proccesses the common words in the training and validation set and tokenize words as unique indexes.
b. The sample code uses `glove` embedding to project word index to embeddings.

Q2

> Outputs: output, h_n
>
> - **output**: tensor of shape (L, H_{in}) for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features (*h_t*) from the last layer of the GRU, for each *t*. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.
> - **h_n**: tensor of shape $(D * \text{num\_layers}, H_{out})$ or $(D * \text{num\_layers}, N, H_{out})$ containing the final hidden state for the input sequence.

- $bidirectional = True$
- $D = (1 + bidirectional) = 2$

a.

1. pass the tokenized batch into the embedding function, now the shape becomes $(B, L, embeddings.shape[-1])$
2. pass to bidirectional GRU : `nn.GRU(embeddings.shape[-1], hidden_size, num_layers, True, True, dropout, bidirectional)` , now the shape becomes $(B, L, (1 + bidirectional) * hidden\_size)$
3. pass to a `Linear` , then `dropout` , then `relu` , `nn.Linear(256, num_class)` . Now the shape is (B, L, num_class)
4. take the $mean$ of the 1^{th} $dim(L)$ as the distribution of classes.

b. achieved 0.90088 initially

c. crossentropy loss.

d. adam, lr=$1e-3$, batch_size=128

Q3

- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for the input sequence. Defaults to zeros if not provided.

vhere:

$$
\begin{aligned}
N &= \text{batch size} \\
L &= \text{sequence length} \\
D &= 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} &= \text{input\_size} \\
H_{out} &= \text{hidden\_size}
\end{aligned}
$$

a.

- $bidirectional = True$
- $D = (1 + bidirectional) = 2$

1. pass the tokenized batch into the embedding function, now the shape becomes $(B, L, embeddings.shape[-1])$
2. pass to bidirectional GRU `nn.GRU(self.idim, hidden_size, num_layers(2), True, True, dropout, bidirectional)` , now the output shape becomes $(B, L, D * hidden\_size)$, and hidden state $(D * 2, B, hidden\_size)$
3. chunk the $output$ to 2 tensors with shape $(B, L, hidden\_size)$, and final hidden state h_n to 2 tensors of shape $(D/2 * 2, B, H_{out})$
4. feed mean of the chunked tensors(both output and hidden state) to a uni-directional GRU as $input$ and h_0, the output shape becomes $(B, L, hidden\_size)$
5. feed to a classification MLP:
   ```
   nn.Sequential( nn.Linear(self.hiddim,
   self.hiddim // 2), nn.LayerNorm(self.hiddim //
   ```

```
2), nn.Dropout(dropout), nn.ReLU(),
nn.Linear(self.hiddim // 2, num_class) )
```

b. achieved 0.77050 initially
c. crossentropy loss.
d. adam, lr=$1e-3$, batch_size=128

Q4

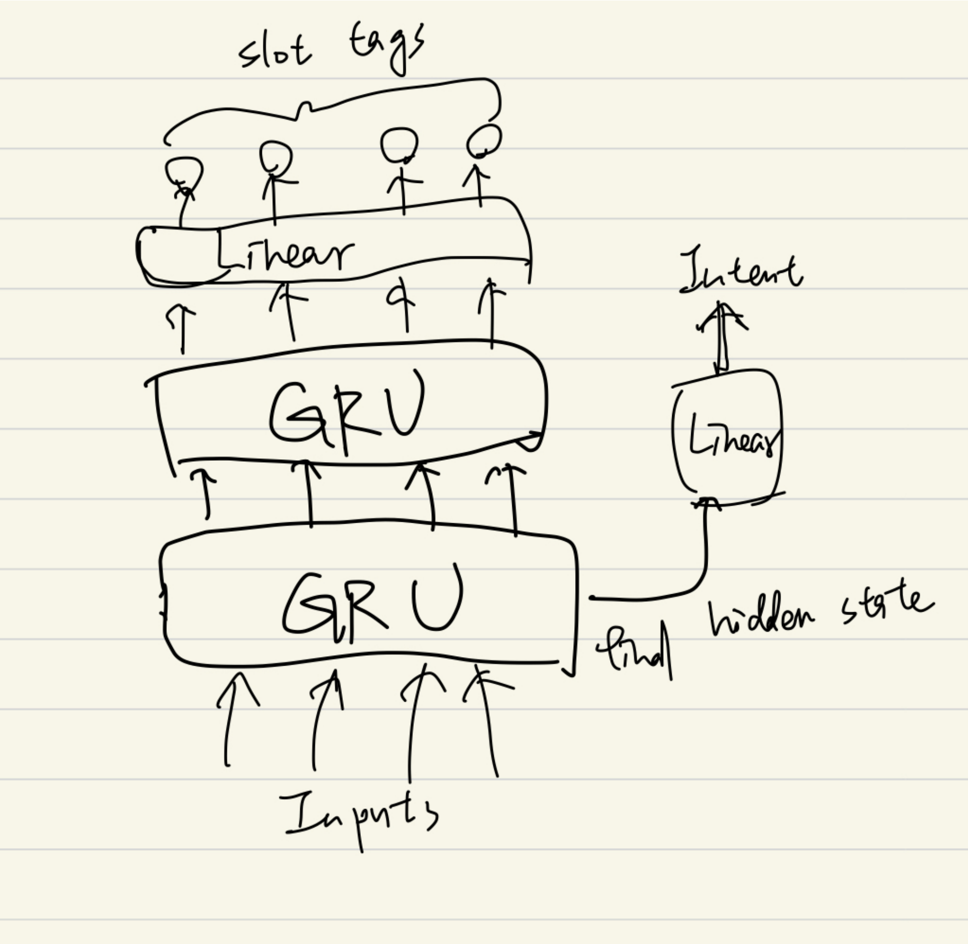
```
              precision    recall  f1-score   support

        date       0.76      0.77      0.77       206
  first_name       0.95      0.88      0.91       102
   last_name       0.77      0.71      0.74        78
      people       0.70      0.71      0.71       238
        time       0.87      0.83      0.85       218

   micro avg       0.79      0.78      0.78       842
   macro avg       0.81      0.78      0.79       842
weighted avg       0.80      0.78      0.79       842
```

- precision: among all element that we predicted to belong to that class, the ratio that we predict it right
  $\frac{tp}{(tp+fp)}$
- recall: among all element that truly belong to each class, the ratio that we predict it right
  $\frac{tp}{(tp+fn)}$
- f1-score:
  $\frac{2*Precision*Recall}{(Precision+Recall)}$
- token acc:
  the total token accuracy
- joint acc:
  the ratio of correct sentences

Q5

I tried joint training.

The model looks like this.



The two GRUs are same as that in Q3, and I feed hidden state to a MLP for intent classification.
Since more data is used to train the first GRU, the model performances better.

final result:

- intent: 0.92400
- slot: 0.78284