Applied Deep Learning - Homework 1

Written By: Wu-Jun Pei (B06902029)

> *(1) This document is edited with Typora. In case some markdown extensions (e.g. math mode) are not supported by native markdown editor, you can read report.pdf for better reading experience.*
>
> *(2) Instead of copying-and-pasteing from the sample code, I implement the entire homework on my own.*

Q1: Data Preprocessing

For the two tasks, intent classification and slot tagging, I use the same method to preprocess the input text into trainable data. The procedures are listed below.

1. (Only for intent classification) First of all, we need to tokenize the input sentence (a `str`) into several tokens. I implement this simply using `str.split()`, a Python built-in function that splits a string into a list of strings with spaces.
2. Now, we have several tokens (`str`-type). We need to convert each token into its embedding. I implemented the function in two steps,
   1. `convert_token_to_id`: convert each token to its id. If we don't have such token, it's converted to `1`, representing it's out-of-vocabulary (OOV). `0` is reserved to be the padding token.
   2. `embedding`: a 2D array of shape $(N_{token} + 2, D_{emb})$, each row stores corresponding embedding for the token. Note that the embeddings are trainable in later training phase.

In this homework, I directly use the token dictionary and the initial embeddings from glove's `glove-wiki-gigaword-300`, a well-known and public word embedding model. The embedding dimension D_emb is thus set 300. Here I initialize the embedding of the OOV token to be the average of all other tokens and the embedding of padding token to be zero. Note that I only extracted the vocabulary used in the two tasks.

> *Relevant code can be found in `src/models/tokenizer.py`. I implement the tokenizer following Hugging Face's interfaces. We often implement the **embedding layer** in the model in PyTorch. Here the tokenizer simply return the `index` of the token (step 2-1).*

Q2: Describe your intent classification model.

Common modules between the two tasks

My models are composed of three sub-modules, including embedding layer, a RNN module and a fully-connected layer.

Embedding Layer

The embeddings are default trainable.

- Input: a `long`-type vector t of length L representing the tokens of the sentence.
- Output: e, a tensor of shape (L, D_{emb})

RNN Module

In this homework, I only used the bidirectional GRU as the RNN cell.

- Hyperparameters
  - D_{hidden}: the hidden size (default=4)
  - N_{layers}: number of RNN layers (default=2)
- Input: a tensor e of shape (L, D_{emb})
- Output:
  - o, a tensor of shape $(L, 2 \cdot D_{hidden})$
  - h, a tensor of shape $(N_{layers} \cdot 2, D_{hidden})$

Fully-connected Layer

The output size is set to be the size of the output domain, 150 for intent classification and 9 for slot tagging.

Model Flow

1. Tokenizer: get the tokens t_i of sentence s_i with the tokenizer. Let the length of t_i be l_i.
2. Embedding layer: get the embeddings of each token, $e_i = \text{EmbeddingLayer}(t_i)$. The shape of e_i is $(L, 300)$
3. RNN module: $o_i, h_i = \text{RNN}(e_i)$. We will only use h_i in the next steps. The shape of h_i is $(N_{layers} \cdot 2, D_{hidden})$
4. Fully connected layer: $p_i = \text{FullyConnectedLayer}(h_i)$. The shape of p_i is $(150,)$ in order to match the number of classes in this task.
5. The prediction will be the class with largest score, $\text{prediction} = \arg\max p_i$

Trainer

The model is trained under the following configuration:

- Loss: standard cross entropy loss
- Optimizer: Adam with learning rate=5e-3 and weight decay=1e-5
- Batch size: 256
- 10 epochs

Final Configuration

After finetuning my model using the validation set, I have the final hyperparameters:

- number of RNN layers N_{layers} = 4
- hidden dimension D_{hidden} = 768

Results

My model achieved 0.9918 accuracy on training set, 0.9037 accuracy on validation set. As for the testing set, my model achieved 0.9000 accuracy on the public set and 0.89733 on the private set. They should be high enough to pass the baseline.

Q3: Describe your slot tagging model.

Model Flow

1. Tokenizer: get the tokens t_i of sentence s_i with the tokenizer. Let the length of t_i be l_i.
2. Embedding layer: get the embeddings of each token, $e_i = \text{EmbeddingLayer}(t_i)$. The shape of e_i is $(L, 300)$
3. RNN module: $o_i, h_i = \text{RNN}(e_i)$. We will only use o_i in the next steps. The shape of o_i is $(l_i, 2 \cdot D_{hidden})$
4. Fully connected layer: For each tag j, we have $p_{i,j} = \text{FullyConnectedLayer}(o_{i,j})$, where $p_{i,j}$ is the vector representing score of each tag. The shape of each $p_{i,j}$ is $(9,)$
5. For each tag j, the tag the model outputs is $\text{tag}_j = \arg\max p_{i,j}$

Trainer

The model is trained under the following configuration:

- Loss: standard cross entropy loss
- Optimizer: Adam with learning rate=5e-3 and weight decay=1e-5
- Batch size: 256
- 25 epochs

Final Configuration

After finetuning my model using the validation set, I have the final hyperparameters:

- number of RNN layers N_{layers} = 4
- hidden dimension D_{hidden} = 768

Results

My model achieved 0.99112 token accuracy / 0.93430 joint accuracy on the training set, 0.96369 token accuracy / 0.77990 joint accuracy on validation set. As for the testing set, my model achieved 0.78498 joint accuracy on the public set and 0.79635 on the private set. They should be high enough to pass the baseline.

Q4: Sequence Tagging Evaluation

Output of `classification_report`

```
              precision    recall  f1-score   support
        date       0.77      0.76      0.77       206
  first_name       0.96      0.80      0.88       102
   last_name       0.83      0.83      0.83        78
      people       0.72      0.71      0.71       238
        time       0.80      0.85      0.82       218

   micro avg       0.79      0.78      0.79       842
   macro avg       0.82      0.79      0.80       842
weighted avg       0.79      0.78      0.79       842
```

Difference between different evaluation methods

- `seqeval`
    - Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

    - Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

    - F1-score

$$\text{F1-score} = \frac{2}{\text{Recall}^{-1} + \text{Precision}^{-1}}$$

- Token accuracy: the token-level accuracy, each token is independent
- Joint accuracy: a sentence is considered correct only when all the tokens are classified correct

Q5: Compare with different configurations.

The validation set is actually the `eval` set in the given dataset. Since the model is trained with fixed training epochs (25), it's possible model starts to overfit after several epochs. Thus, the model is evaluated on the checkpoint with the lowest validation loss.

1. Different hidden dimension

In this experiment, I want to examine the impact of different hidden size.

Settings: the number of layers N_{layers} is set 4. Other settings are the same as described in Q2 and Q3 except the hidden dimension D_{hidden} is the control variable

Intent classification

Hidden Size	Best Epoch	Train Loss	Train Acc.	Val. Loss	Val. Acc.
16	25	0.13559	0.98110	0.80867	0.81927
64	15	0.05819	0.99247	0.48669	0.88428
256	10	0.04604	0.99108	0.42132	0.89731
1024	7	0.05599	0.98501	0.39666	0.90421

Slot tagging

Hidden Size	Best Epoch	Train Loss	Train Token Acc.	Train Joint Acc.	Val. Loss	Val. Token Acc.	Val. Joint Acc.
16	25	0.11982	0.97012	0.81091	0.17132	0.95241	0.71164
64	15	0.07704	0.97403	0.83201	0.12230	0.96017	0.75946
256	9	0.08248	0.96948	0.80365	0.11872	0.95725	0.74794
1024	7	0.79194	0.96900	0.79658	0.11501	0.96049	0.75673

Findings

We can find that the models are easily overfitted in these two tasks. Models with small hidden size fits the training data well (0.98 accuracy on intent classification and 0.97 token accuracy on slot tagging). Also, we can see that models with large hidden size stops improving on validation set after short epochs, and it's another indicator showing that models are getting overfitted. Blindly increasing the hidden size will only make the situation worse.

2. Different number of RNN layers

In this experiment, I want to examine the influence of different number of RNN layers and see if we can tune the model with better hyperparameter.

Settings: the hidden dimension D_{hidden} is set 256. Other settings are the same as described in Q2 and Q3 except the number of RNN layers N_{layers} is the control variable.

Intent classification

#RNN Layers	Best Epoch	Train Loss	Train Acc.	Val. Loss	Val. Acc.
1	13	0.09170	0.98472	0.52171	0.87420
2	12	0.05639	0.99024	0.43037	0.89146
4	10	0.04604	0.99108	0.42131	0.89730
8	8	0.08302	0.97821	0.49763	0.88807

Slot tagging

#RNN Layers	Best Epoch	Train Loss	Train Token Acc.	Train Joint Acc.	Val. Loss	Val. Token Acc.	Val. Joint Acc.
1	17	0.07396	0.97430	0.83076	0.11888	0.95870	0.74168
2	13	0.06869	0.97472	0.83062	0.11425	0.96149	0.76454
4	9	0.08248	0.96948	0.80365	0.11872	0.95725	0.74794
8	11	0.07726	0.97108	0.81196	0.12441	0.95655	0.74656

Findings

First of all, similar to experiment 1, we can find that as N_{layers} increases, and the model is more powerful, it's more likely the model gets overfitted. We can observe the trend from the *"Best Epoch"* column. On the other hand, we can see that the model has no significant improve when we add an additional RNN layers when N_{layers} is more than 4, indicating that we should not blindly adding more RNN layers to prevent overfitting.

3. Trainable embedding or not

In this experiment, I want to examine whether making embedding trainable or not affects the models.

Setting: the hidden dimension D_{hidden} is set 256 and the number of RNN layers N_{layers} is set 4. Other settings are the same as described in Q2 and Q3.

Intent classification

Embedding Trainable	Best Epoch	Train Loss	Train Acc.	Val. Loss	Val. Acc.
True	10	0.04604	0.99108	0.42131	0.89730
False	18	0.01626	0.99781	0.45315	0.89068

Slot tagging

Embedding Trainable	Best Epoch	Train Loss	Train Token Acc.	Train Joint Acc.	Val. Loss	Val. Token Acc.	Val. Joint Acc.
True	9	0.08248	0.96948	0.80365	0.11872	0.95725	0.74794
False	17	0.058677	0.97801	0.84840	0.11324	0.96372	0.78212

Findings

Making embedding layer not trainable will slow down the training process, we can find evidence from the *"Best Epoch"* column. The models whose embedding layers are not trainable converge slower. Whether the embedding layer is trainable has less impact on intent classification model, while it shows significant improve on slot tagging model when the embedding layer is not trainable.