ADL 2021 Homework 1 Report

學號: B06902125 姓名: 黃柏瑋 系級: 資工四

1. Data Processing

First, input sentences are tokenized based on spaces. Then, the representation of each token is its corresponding embedding in GloVe (http://nlp.stanford.edu/data/glove.840B.300d.zip) if such token is included; otherwise, random vector is assigned. Statistically, there are about 83% tokens covered by GloVe in intent classification, and there are 72% covered in slot tagging task.

2. Describe your intent classification model (basic)

a. Model

First, 2-layer bidirectional GRU is used for feature extraction. Below, $h_t^{(i)}$ is the hidden state (output) of layer i at time t, and $\rightharpoonup$ means forward direction while $\leftharpoonup$ means backward direction. Note that $i \in \{0, 1\}$ and $h_t^{(0)} = w_t$ where w_t is the word embedding of the token at time t.

$$\overrightharpoon{h}_t^{(i)} = GRU(\overrightharpoon{h}_t^{(i-1)}, \overrightharpoon{h}_{t-1}^{(i)})$$
$$\overleftharpoon{h}_t^{(i)} = GRU(\overleftharpoon{h}_t^{(i-1)}, \overleftharpoon{h}_{t+1}^{(i)})$$

Then, two hidden states (outputs) of the last layer, $\overrightharpoon{h}_T^{(1)}$ and $\overleftharpoon{h}_0^{(1)}$, are concatenated and regarded as the features of MLP classifier, where T is the length of the input sentence. Finally, the outputs of the classifier, i.e. y, are the emission scores of intent classes, which represent the likelihoods of the sentence being each certain class. Apparently, one with the highest emission score, i.e. y^*, is the predicted intent class of the input sentence.

$$y = (y(0), \ldots, y(149)) = MLP(concat(\overrightharpoon{h}_T^{(1)}; \overleftharpoon{h}_0^{(1)}))$$
$$y^* = \mathrm{argmax}(y)$$

Here are some configurations of GRU model:

1. Hidden size: 128
2. Number of layers: 2
3. Dropout: 0.5

Here are some configurations of MLP model:

1. Hidden sizes: 256 $\times$ 256 $\rightarrow$ 256 $\times$ 150
2. Dropout: 0.5

b. Loss function

Cross Entropy Loss with L2 regularization is used, where weight of the regularization is 10^{-5}. Below, $\hat{y}(i) \in \{0, 1\}$ indicates whether the input sentence belongs to intent class i (1 means positive otherwise 0), $softmax(y(i))$ is the predicted probability of intent class i, and w represents the weights of the whole model.

$$L = -\sum_{i=0}^{149} \hat{y}(i)\ \log(softmax(y(i))) + 10^{-5} \cdot ||\mathrm{w}||_2^2$$

c. Other details for training

Here are some configurations for training:

1. Optimizer: Adam
2. Batch size: 32
3. Learning rate: 0.001
4. Epoch: 100

d. Performance

Name	Validation[†]	Public Test	Private Test
Basic GRU	0.9320	0.9133	0.9120

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

3. Describe your slot tagging model (basic)

a. Model

First, 2-layer bidirectional GRU is used for feature extraction. Below, $h_t^{(i)}$ is the hidden state (output) of layer i at time t, and $\rightharpoonup$ means forward direction while $\leftharpoonup$ means backward direction. Note that $i \in \{0, 1\}$ and $h_t^{(0)} = w_t$ where w_t is the word embedding of the token at time t.

$$\overrightharpoon{h}_t^{(i)} = GRU(\overrightharpoon{h}_t^{(i-1)}, \overrightharpoon{h}_{t-1}^{(i)})$$
$$\overleftharpoon{h}_t^{(i)} = GRU(\overleftharpoon{h}_t^{(i-1)}, \overleftharpoon{h}_{t+1}^{(i)})$$

Then for each t, two hidden states (outputs) of the last layer, $\overrightharpoon{h}_t^{(1)}$ and $\overleftharpoon{h}_t^{(1)}$, are concatenated and regarded as the features of MLP classifier.
Finally, the outputs of the classifier, i.e. y_t, are the emission scores of slot tags at time t, which represent the likelihoods of the token being each certain tag. Apparently, one with the highest emission score, i.e. y_t^*, is the predicted slot tag for the token at time t.

$$y_t = (y_t(0), \dots, y_t(9)) = MLP(concat(\overrightharpoon{h}_t^{(1)}; \overleftharpoon{h}_t^{(1)}))$$
$$y_t^* = \mathrm{argmax}(y_t)$$

Here are some configurations of GRU model:

1. Hidden size: 128
2. Number of layers: 2
3. Dropout: 0.5

Here are some configurations of MLP model:

1. Hidden sizes: 256 $\times$ 256 $\rightarrow$ 256 $\times$ 10
2. Dropout: 0.5

Note that for advanced loss function, please refer to Question 5.

b. Loss function

Cross Entropy Loss with L2 regularization is used, where weight of the regularization is 10^{-5}. Below, $\hat{y}_t(i) \in \{0, 1\}$ indicates whether the token at time t belongs to slot tag i (1 means positive otherwise 0), $softmax(y_t(i))$ is the predicted probability of slot tag i at time t, T is the length of each input sentence, and w represents the weights of the whole model.

$$L = -\sum_{t=0}^{T-1}\sum_{i=0}^{9} \hat{y}_t(i)\ \log(softmax(y_t(i))) + 10^{-5} \cdot ||\mathrm{w}||_2^2$$

c. Other details for training

Here are some configurations for training:

1. Optimizer: Adam
2. Batch size: 32
3. Learning rate: 0.001
4. Epoch: 100

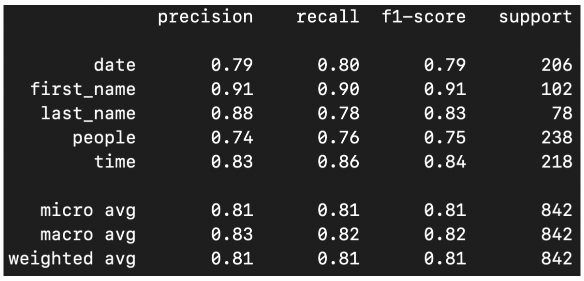
d. Performance

Name	Validation†	Public Test	Private Test
Basic GRU	0.8210	0.7962	0.7937

(† means *train.json* is used for training and *eval.json* is used for validation. Otherwise, both *train.json* and *eval.json* are used for training.)

4. Sequence Tagging Evaluation

Classification Report from *seqeval*



```
              precision    recall  f1-score   support

        date       0.79      0.80      0.79       206
  first_name       0.91      0.90      0.91       102
   last_name       0.88      0.78      0.83        78
      people       0.74      0.76      0.75       238
        time       0.83      0.86      0.84       218

   micro avg       0.81      0.81      0.81       842
   macro avg       0.83      0.82      0.82       842
weighted avg       0.81      0.81      0.81       842
```

Metrics Comparison

Token Accuracy

Token accuracy is the fraction of true-predicted tokens among all tokens. In slot tagging task, the token accuracy of the basic model is 0.9688.

The main problem of this metric occurs when it comes to imbalanced counts of tag classes. For example, tag “O” is an overwhelming majority, but it is the least important one. However, even if one tagging model outputs tag “O” for all tokens, the token accuracy might still be high. That is, in this case, token accuracy overestimates the performance of tagging model.

Joint Accuracy

Joint accuracy is the fraction of true-predicted sentences among all sentences. Tags of tokens in a true-predicted sentence must be all correctly estimated. In slot tagging task, the joint accuracy of the basic model is 0.8210.

The joint accuracy looks more reasonable than token accuracy. However, just like the problem of token accuracy, joint accuracy might still be stuck in extreme imbalanced dataset. For example, if there are plenty of sentences with all "O"-tagged tokens, the joint accuracy might overestimate the performance of tagging model.

Precision, Recall and F1-score

For each tag class, precision is the fraction of true-positive tags among all predicted-positive tags, while recall is the fraction of true-positive tags that were predicted positive. Generally speaking, a good model should achieve high scores on both precision and recall. Therefore, an unified metric called F1-score, the harmonic mean of precision and recall, appears.

When the counts of tag classes are imbalanced, token accuracy (and even joint accuracy) might become misleading. At this time, F1-score comes in handy. Just like classification report from *seqeval* above, people can focus on the performance of some important classes, and then calculate a reasonable average score for their tasks. However, F1-score is not sentence-based, so those with good F1-score are not guaranteed to get reasonable joint accuracies; that is, there might be only a few sentences with all correctly predicted tags.

5. Compare with different configurations

RNN / GRU / LSTM

Introduction

Different models have different numbers of parameters and gating mechanisms. In this experiment, all parameters are mentioned in the basic model and fixed except the model type; RNN, GRU and LSTM are considered.

Training Losses (Intent)

Name	epoch 20†	epoch 40†	epoch 60†	epoch 80†	epoch 100†
RNN	0.0574	0.0341	0.0399	0.0399	0.0362
GRU	0.0102	0.0080	0.0096	0.0077	0.0071
LSTM	0.0103	0.0047	0.0047	0.0027	0.0033

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Intent)

Name	Validation†	Public Test	Private Test	Avg Test
RNN	0.8797	0.8311	0.8453	0.8382
GRU	**0.9320**	0.9133	0.9120	0.9127
LSTM	0.9297	**0.9196**	**0.9213**	**0.9205**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Training Losses (Slot)

Name	epoch 20†	epoch 40†	epoch 60†	epoch 80†	epoch 100†
RNN	0.0146	0.0080	0.0064	0.0064	0.0050
GRU	0.0065	0.0036	0.0025	0.0024	0.0022
LSTM	0.0069	0.0029	0.0026	0.0024	0.0018

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Slot)

Name	Validation†	Public Test	Private Test	Avg Test
RNN	0.7960	0.7582	0.7444	0.7513
GRU	**0.8210**	0.7962	**0.7937**	0.7950
LSTM	0.8150	**0.8075**	0.7926	**0.8001**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Discussion

According to the training losses and performance reported above, in both tasks, RNN is always the underfitting one, whose training losses, validation score and test scores are both much worse than GRU and LSTM.

As for GRU and LSTM, they are comparable and each has its own advantages. Note that LSTM has more number of parameters than GRU; therefore, although LSTM can converge to a lower training loss faster, it might be stuck in overfitting when dataset isn't large enough. This is the possible reason why, in both tasks, GRU outperforms LSTM by about 0.3% to 0.5% when it comes to validation score, where only *train.json* are used in training; however, LSTM has much better averaged test score, where both *train.json* and *eval.json* are used for training, than GRU by about 0.5% to 1%.

Different number of GRU layers

Introduction

Different numbers of GRU layers cause different model complexities. In this experiment, all parameters are mentioned in the basic model and fixed except the number of GRU layers; 1, 2 and 3 are considered.

Training Losses (Intent)

Name	epoch 20†	epoch 40†	epoch 60†	epoch 80†	epoch 100†
GRU (1)	0.0102	0.0081	0.0055	0.0052	0.0038
GRU (2)	0.0102	0.0080	0.0096	0.0077	0.0071
GRU (3)	0.0127	0.0080	0.0079	0.0070	0.0079

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Intent)

Name	Validation†	Public Test	Private Test	Avg Test
GRU (1)	**0.9357**	**0.9182**	**0.9271**	**0.9227**
GRU (2)	0.9320	0.9133	0.9120	0.9127
GRU (3)	0.9280	0.9088	0.9147	0.9118

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Training Losses (Slot)

Name	epoch 20†	epoch 40†	epoch 60†	epoch 80†	epoch 100†
GRU (1)	0.0081	0.0030	0.0013	0.0003	0.0006
GRU (2)	0.0065	0.0036	0.0025	0.0024	0.0022
GRU (3)	0.1005	0.0040	0.0044	0.0025	0.0027

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Slot)

Name	Validation†	Public Test	Private Test	Avg Test
GRU (1)	0.8170	0.7898	0.7942	0.7920
GRU (2)	0.8210	**0.7962**	0.7937	0.7950
GRU (3)	**0.8300**	0.7920	**0.7996**	**0.7958**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Discussion

Generally speaking, more number of layers increases model complexity, and training loss will thus get lower. However, the training losses reported above don't reflect this statement. Specifically, in both tasks, GRU with one layer can converge to a much lower training loss than those with more than one layers. The possilble reason might be the influence of dropout mechanisms between GRU layers, which is implemented not in 1-layer GRU but in 2 or 3-layer one.

Note that the dropout mechanism can sacrifice some training accuracy for better generalization. However, if dropout ratio is set too large, too much training accuracy will be dropped and lead to underfitting. Back to the results of this experiment, for intent classification task, dropout ratio (0.5) might be too much, causing accuracy crash when it comes to multi-layer GRU. On the contrary, for slot tagging task, the dropout ratio (0.5) is moderate so that multi-layer GRU can keep enough information from training data without losing generalization.

p.s. Due to the approach of deadline, comparison of different dropout ratios, especially in intent classification task, will be left as future work.

Different number of hidden dimensions

Introduction

Different numbers of hidden dimensions offer different model complexities. In this experiment, all parameters are mentioned in the basic model and fixed except GRU hidden size; 64 and 128 are considered.

Training Losses (Intent)

Name	epoch 20[†]	epoch 40[†]	epoch 60[†]	epoch 80[†]	epoch 100[†]
GRU (64)	0.0186	0.0107	0.0097	0.0077	0.0090
GRU (128)	0.0102	0.0080	0.0096	0.0077	0.0071

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Intent)

Name	Validation[†]	Public Test	Private Test	Avg Test
GRU (64)	0.9276	0.9071	**0.9151**	0.9111
GRU (128)	**0.9320**	**0.9133**	0.9120	**0.9127**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Training Losses (Slot)

Name	epoch 20[†]	epoch 40[†]	epoch 60[†]	epoch 80[†]	epoch 100[†]
GRU (64)	0.0127	0.0058	0.0054	0.0024	0.0023
GRU (128)	0.0065	0.0036	0.0025	0.0024	0.0022

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Slot)

Name	Validation[†]	Public Test	Private Test	Avg Test
GRU (64)	0.8120	0.7866	0.7856	0.7861
GRU (128)	**0.8210**	**0.7962**	**0.7937**	**0.7950**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Discussion

According to the training losses and performance reported above, in both tasks, GRU with hidden size 64 gains higher training losses and lower accuracies. That is, GRU with hidden size 128 is more informative, without losing generalization, than that with 64.

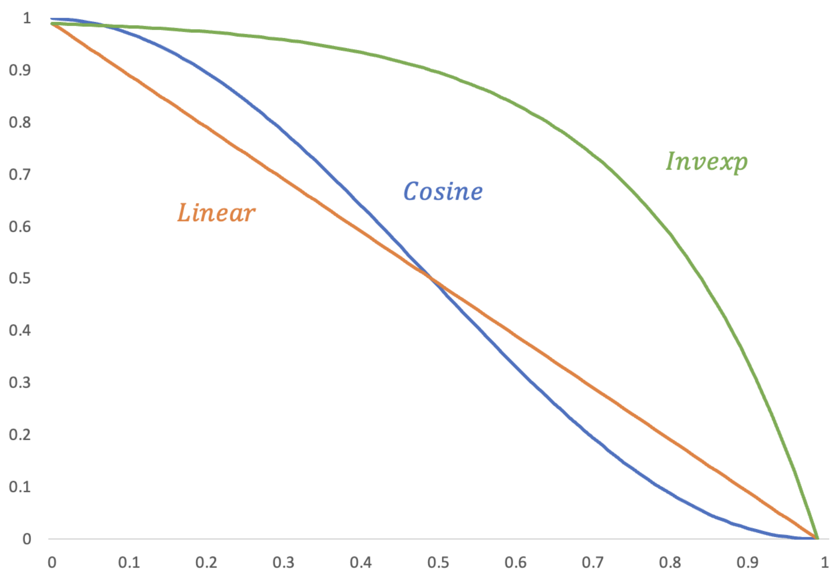
Learning rate schedulers

Motivation and Introduction

In training progress, small learning rate is usually required for stable convergence in the latter epochs. Therefore, different learning rate decay schedulers will be compared here.

In this experiment, all parameters are mentioned in the basic model except adding learning rate scheduler. There are three kinds of schedule functions, and their formulas and graphs are as follows, where x is training progress rate, i.e. $\frac{current\ epoch}{total\ epoch}$, and y is learning rate scale, i.e. $\frac{current\ lr}{initial\ lr}$:

$$\begin{aligned} Cosine &: y = 0.5 \cdot (1 + \cos(\pi x)) \\ Linear &: y = 1 - x \\ Invexp &: y = 1 - 0.01^{(1-x)} \end{aligned}$$



Training Losses (Intent)

Name	epoch 20†	epoch 40†	epoch 60†	epoch 80†	epoch 100†
Basic GRU	0.0102	0.0080	0.0096	0.0077	0.0071
GRU + Cosine	0.0104	0.0040	0.0022	0.0002	0.0008
GRU + Linear	0.0076	0.0039	0.0025	0.0011	0.0004
GRU + Invexp	0.0099	0.0091	0.0056	0.0026	0.0004

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Intent)

Name	Validation†	Public Test	Private Test	Avg Test
Basic GRU	0.9320	0.9133	0.9120	0.9127
GRU + Cosine	**0.9350**	**0.9302**	**0.9387**	**0.9345**
GRU + Linear	**0.9350**	0.9240	0.9333	0.9287
GRU + Invexp	0.9317	0.9227	0.9320	0.9274

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Training Losses (Slot)

Name	epoch 20†	epoch 40†	epoch 60†	epoch 80†	epoch 100†
Basic GRU	0.0065	0.0036	0.0025	0.0024	0.0022
GRU + Cosine	0.0081	0.0030	0.0013	0.0003	0.0006
GRU + Linear	0.0079	0.0028	0.0014	0.0006	0.0007
GRU + Invexp	0.0080	0.0034	0.0022	0.0010	0.0005

(† means *train.json* is used for training and *eval.json* is used for validation.)

Performance (Slot)

Name	Validation†	Public Test	Private Test	Avg Test
Basic GRU	0.8210	0.7962	0.7937	0.7950
GRU + Cosine	0.8170	0.7989	0.7776	0.7883
GRU + Linear	0.8170	**0.8016**	0.7926	0.7971
GRU + Invexp	**0.8290**	0.7995	**0.7969**	**0.7982**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Discussion

According to the test scores reported above, in intent classification task, all schedules can improve 1.5% to 2.5%. On the other hand, in slot tagging task, only invexp schedule can improve the accuracy by 0.3%, while the others get even worse than basic model. The possible reason might be, for slot tagging task, training with a relatively large learning rate is necessary. Therefore, cosine and linear schedule, where learning rate decays faster, might hurt the performance. To sum up, learning rate schedule can indeed improve the performance; yet, different tasks might require different kinds of schedule functions.

Conditional Random Field (CRF)

Motivation and Introduction

In basic model of slot tagging task, MLP classifier is utilized to transform the outputs of BiGRU into the emission scores, max of which is the predicted tag of such token. However, this method doesn't explicitly consider the transitional dependence of predicted slot tags over time. For instance, it doesn't directly lower the probabilities of impossible results, such as *[B-date I-people O O]*, *[O I-time B-time]*, etc.

Therefore, here resorts to Conditional Random Field (CRF). The CRF layer contains an MLP classifier and a transition matrix. The former outputs the emission scores just as that in the basic model, and the latter offers the transition scores, which are the likelihoods of a token being a certain tag considering the previous token was a certain tag.

Specifically, below, for each t, two hidden states (outputs) from BiGRU's last layer, $\overrightarrow{h}_t^{(1)}$ and $\overleftarrow{h}_t^{(1)}$, are concatenated and regarded as the features of MLP classifier, whose outputs, i.e. y'_t, are the emission scores of slot tags at time t. Then the emission scores and the transition scores from transition matrix are aggregated as y_t, where $y_t(i, j)$ is the score considering both token representations and tag relations when the tag at time $t-1$ is i and that at time t is j. Finally, viterbi algorithm is used to find the path with highest sum score, i.e. y^*, as the predicted tag sequence of the input sentence.

$$y'_t = (y'_t(0), \ldots, y'_t(9)) = MLP(concat(\overrightarrow{h}_t^{(1)}; \overleftarrow{h}_t^{(1)}))$$

$$y_t = \begin{pmatrix} y_t(0,0) & \ldots & y_t(0,9) \\ y_t(1,0) & \ldots & y_t(1,9) \\ \vdots & \ddots & \vdots \\ y_t(9,0) & \ldots & y_t(9,9) \end{pmatrix} \; where \; y_t(i,j) = y'_t(j) + Trans(i,j)$$

$$y^* = \underset{i_0, \ldots, i_{T-1}}{\operatorname{argmax}} \sum_{t=0}^{T-1} y_t(i_{t-1}, i_t)$$

By the way, as for CRF's training, the loss function can be extended as follows:

$$L = -\sum_{t=0}^{T-1} \sum_{i=0}^{9} \sum_{j=0}^{9} \hat{y}_{t-1}(i)\, \hat{y}_t(j)\ \log(softmax(y_t(i,j))) + 10^{-5} \cdot ||\mathrm{w}||_2^2$$

Performance

Name	**Validation**[†]	**Public Test**	**Private Test**	**Avg Test**
Basic GRU	0.8210	0.7962	**0.7937**	0.7950
GRU + CRF	**0.8260**	**0.8064**	0.7926	**0.7995**

(† means *train.json* is used for training and *eval.json* is used for validation.
Otherwise, both *train.json* and *eval.json* are used for training.)

Discussion

According to the averaged test scores reported above, the CRF model outperforms the basic model by about 0.5%. Namely, considering tag relations can slightly improve the accuracy.