SLUBert: A Simple yet Effective Framework for Chinese Spoken Language Understanding

Fan Nie, Shuyuan Zhang, Yikai Zhang*
Department of Computer Science and Engineering (IEEE Honor Class)
Shanghai Jiao Tong University

Abstract

Despite the remarkable success of spoken language understanding (SLU) in dialog system, it's generally studied based on English. However, this time we focus on Chinese spoken language and propose a new framework for solving this task. Various experiments done by us represents our new method's outperforms all the baselines by a large margin. Besides, we explore how to use historical dialogue information and analyze all the experimental results in detail. We also have an open discussion on the bottleneck of models' performance and the theoretical validity of different methods.

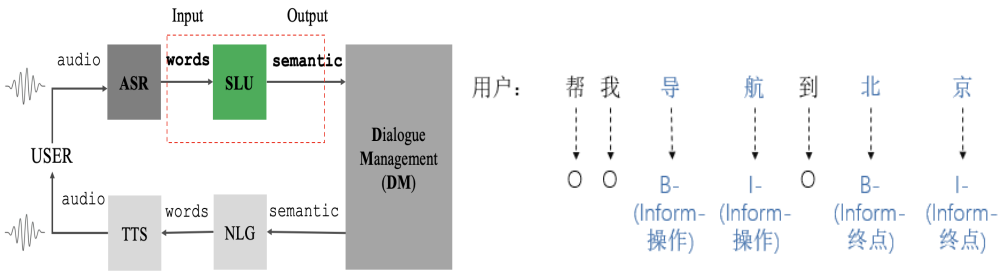
1 Introduction

Spoken language understanding (SLU) is an emerging field in between the areas of speech processing and natural language processing. It's an critical part of conversation system, which aims to extract users' intention and feeds those information into latter model which finally completes the task.

In this work, we mainly focus on how to improve the final accuracy. Besides sophisticatedly fine-tuning on the given baseline model, we try the transformer model: the most heated architecture recent years in NLP domain. We then explore the usage of chinese words tokenizing and bert pretrained model. To support tokenizing, we rewrite the whole pipeline(data preprocessing, testing, validation and the main function). After that, we tried to leverage history conversation which we think may further enhance our model. First we tried it on baseline in a batched uni-direction manner. Then we implement it in the foundation of tokenized bert in another manner mentioned in [1]. Beyond all these, we jump out of the tagging model to explore the usage of pointer network, which we think may aid achieving better result with extraction from the raw test.

The contributions are:

- We finetune all the baseline and proposed models and provide detailed experiment settings and hyperparameters in our paper.
- We evaluate the performance of baseline models with different RNN encoders, compare and analyze their results.
- We propose SLUBert, a simple yet effective framework for SLU tasks, and justify their performance with solid experiments, whose results surpass all the baselines' accuracy by nearly 10%.
- We discuss the usage of history conversation in our model and implement it in two distinct ways
- We explore another kind of modeling to solve SLU: Pointer Network, and discuss detailly how we can implement it



(a) A diagram of Spoken dialogue system (b) An example of "BIO" Encoding

Figure 1: In **(a)**, ASR means Automatic Speech Recongnition, NLG means Natural Language Generation, and TTS means Text to Speech. In **(b)**, Act-slot can be combined as a tag and then we can use the B-(act plot) and I-(act plot) tags to determine the value

2 Problem Formulation

Fig. 1(a) shows a diagram of Spoken dialogue system, which includes five parts. Our task focuses on the SLU part, which takes words as input and output their semantics. In general, users's intention is defined with some semantic units, each consists of a (act, slot, value) triplets.

In the next few sections, we will introduce different algorithms, model architectures, and original improvements. Next is the introduction of datasets, experimental settings and result analysis.

3 Methodology

3.1 The Baseline Model

To solve the problem of spoken language understanding, one easy way is to transform it into a sequence labeling problem, which is used by our baseline method. The standard way to do this is the "BIO" encoding[2], where we label each word by "B-X", "I-X" or "O". Here, "B-X" means "begin a phrase of type X", "I-X" means "continue a phrase of type X" and "O" means "not of any type." There are 2 types of action and 18 types of slots in our dataset, so X has 36 types totally. Then we're faced with the problem of assigning the label to each word in a sequence, which can then be seen as a multi-class classification task.

All of the single words in the sequences will be embedded as vectors, and denoted by indexes. Since all the types of X can be the beginning or middle of the phrase, and we add a type of 'PAD' to represent words that should be ignored, there're totally 74 tags denoted by their indexes as well.

Then we feed the input data into RNN encoders to extract their features. The encoders we choose are bi-LSTM, bi-GRU and bi-RNN.

Bi-RNN. A multi-layer Elman RNN with $tanh$ can be applied to the model input as encoder layer. For each word in the input sequence, each layer computes the following function:

$$h_t = tanh(w_{ih}x_t + b_{ih} + w_{hh}h_{t-1} + b_{hh}) \tag{1}$$

where h_t is the hidden state at time t, x_t is the input at time t, and $h(t1)$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time o.

Bi-GRU. For each word in our model input, each layer computes the following function:

$$\begin{aligned} r_t &= sigmoid(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\ i_t &= sigmoid(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\ n_t &= tanh(W_{in}x_t + b_{in} + rt(W_{hn}h_{(t-1)} + b_{hn})) \\ h_t &= (1 - i_t)nt + i_t * h(t-1) \end{aligned} \tag{2}$$

*The three authors are sorted by name.

where h_t is the hidden state at time t, x_t is the input at time t, and $h(t1)$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time o. And r_t, z_t, n_t are the reset, update, and new gates, respectively. $*$ is the Hadamard product.

Bi-LSTM. A multi-layer long short-term memory (LSTM) RNN has been chosen as one of the encoders. For each word in the input sequence, each layer computes the following function:

$$\begin{aligned} i_t &= sigmoid(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\ f_t &= sigmoid(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\ o_t &= sigmoid(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\ g_t &= tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\ c_t &= f_t c_{t-1} + i_t g_t \\ h_t &= o_t * tanh(c_t) \end{aligned} \tag{3}$$

where $h_t$is the hidden state at time t, $c_t$is the cell state at time$t$, $x_t$is the input at time t, $h_{t-1}$is the hidden state of the layer at time $t-1$ or the initial hidden state at time o. i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively.

3.2 Leveraging History Conversation

According to the format of the dataset, short sentences are grouped together to form conversations. In each conversation, sentences tend to be strongly related with each other, which means it's possible to aid training with history information!

We implement models that utilize history conversation in two distinct ways.

1. To remain the effectiveness brought about by batchfying, we group each conversation together. Specifically, we concatenate all the utterances in each conversation, separating them by semicolon, and feed each concatenated long utterance to the baseline model. In this way, the model can still leverage the effectiveness brought out by concurrency but also learn from the past for downstream utterances. However, there exists a main drawback that we cannot use bi-LSTM since it's not allowed to see the information in the future.
2. We also tried a method based on [1]. We call it *MultiTurn* model. This model utilizes a mechanism similar to the attention mechanism. For each episode of talks, the model maintains a memory pool that stores the past utterance. When predicting the new utterance, the model computes the similarity between the new utterance and the utterance in the memory. The weighted sum of knowledge in the memory is then utilized to help improve the accuracy of prediction.

3.3 Proposed Model

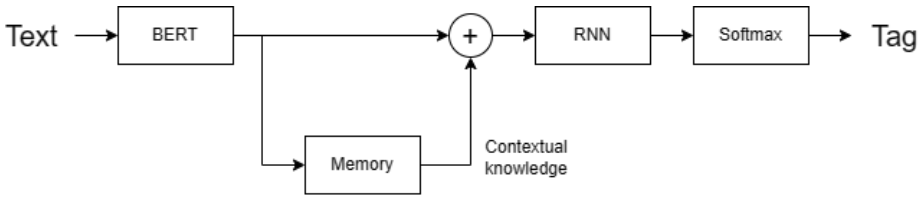


Figure 2: The flow of our proposed model.

Our proposed model first use a BERT model to encode the input text to word vectors. The word vectors are then fed into a RNN (GRU or LSTM) and a softmax classifier to obtain the corresponding tag. As mentioned before, we use the memory and attention mechanism to make use of the contextual information. This is shown in Figure 2 as the "Memory" part. During each episode, every round of utterance is encoded again by a neural network and stored in the memory, denoted by $m_1, m_2, \ldots$. The similarity of two vectors can be measured by their inner product. The similarity between the current utterance and m_i is denoted by p_i. All p_i is processed by a softmax function so that their sum

is 1. Then we obtain the contextual knowledge vector

$$c = \sum_i p_i m_i$$

This vector can be added to the original word vector to introduce the contextual information.

4 Experiments

In this section, we conduct extensive experiments to answer the following questions:

- **Q1:** How effective are our baseline models or SLUBert models in SLU prediction tasks? And how does the use of history conversation affects the results?
- **Q2:** How does the noise of input affect the performance of SLUBert? Besides, what's the training bottleneck of the models?

4.1 Experiment Setup

Data We use the given training and validation datasets which groups utterances in the same conversion together in json format.

Validation We validate our model in the dataset which shares the format with the training set. The output of our model on an utterance is considered right only when all the outputs are exactly match. We compute the accuracy, precision, recall rate and F1 score for validation.

Table 1: Main prediction results of baseline models and our proposed models. Because testing samples are unlabelled, the accuracy and F scores displayed in the table are all results on the validation dataset called 'development'. All the models are trained under ASR input, which may be inconsistent with manual input.

Model	**Accuracy**↑	**Precision**↑	**Recall**↑	**F1 Score**↑	**Ranking**
SLU-Baseline-RNN	69.77±0.13	78.17±1.18	72.87±0.50	75.42±0.39	7
SLU-Baseline-LSTM	70.79±0.29	79.12±1.41	74.58±0.46	76.78±0.59	6
SLU-Baseline-GRU	70.91±0.19	80.17±1.23	74.52±0.37	77.23±0.38	5
SLUBert-RNN	76.31±0.49	82.51±0.98	80.02±0.49	81.25±0.51	4
SLUBert-LSTM	77.21±0.57	83.45±1.11	80.52±0.44	81.96±0.56	3
SLUBert-GRU	**80.67±0.44**	**85.50±0.97**	**85.80±0.23**	**85.65±0.29**	**1**
SLUBert-MultiTurn	78.67±0.56	84.14±1.28	84.02±0.34	84.08±0.42	2

4.2 Main Results (Q1)

We train and evaluate each model under the settings as described in the last section (more hyper-parameter and training/testing details in Appendix A). All the experiments are run over 5 different random seeds. We make the following observations:

SLU Results. Tab. 1 reports SLU-Baseline and SLUBert's prediction performance. In this setting, we treat the baseline models and pretrained models with three different rnn encoders and fairly compare them under the same setting. The results show that the SLUBerts consistently outperform all the baselines by a large margin along with low variance. Such improvements strongly suggest that SLUBert can achieve better prediction performance for spoken language understanding tasks. Besides, the results also show that Baseline-GRU beats all the other baselines and attain an accuracy of 70.91%.Similarly, SLUBert-GRU outperforms the other two SLUBert models as well.

Usage of History Conversation. Fig. 3(a) shows the performance of baseline models without or with our implementation of batched history conversation. To validate if history information really aid the result, we used uni-direction LSTM to compare with our model. As is shown in the bar graph, there isn't much difference whether history information is included or not. The result is quite reasonable because after we carefully inspect the given dataset, we find that each conversation tends to contain only one utterance. Moreover, in those multi-turn conversations, which takes up only a tiny

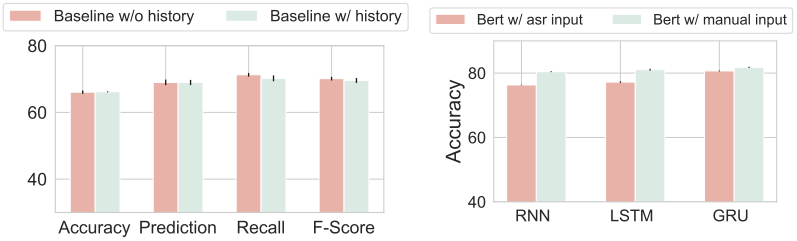


Figure 3: In **(a)**, the histogram represents the performance of baselines with or without history information. In **(b)**, the histogram shows the validation accuracy of SLUBerts with ASR inputs(inputs with noise) or manual input(inputs without noise).Note that the validation data are all ASR inputs for the sake of comparative fairness.

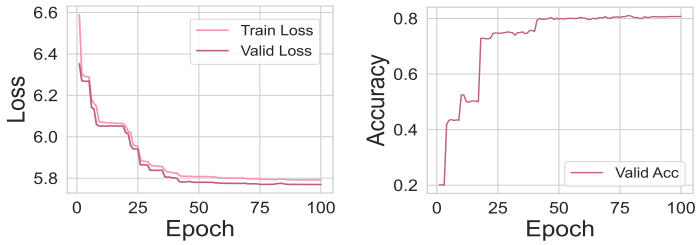


Figure 4: **(b)** and **(c)** show the training process of SLUBert-GRU. Specifically, they represent the changes of training and validation loss and validation accuracy with the increase of epochs. Due to space limitation, we omit the results of SLUBert-RNN and SLUBert-LSTM since they're similar to those of SLUBert-GRU.

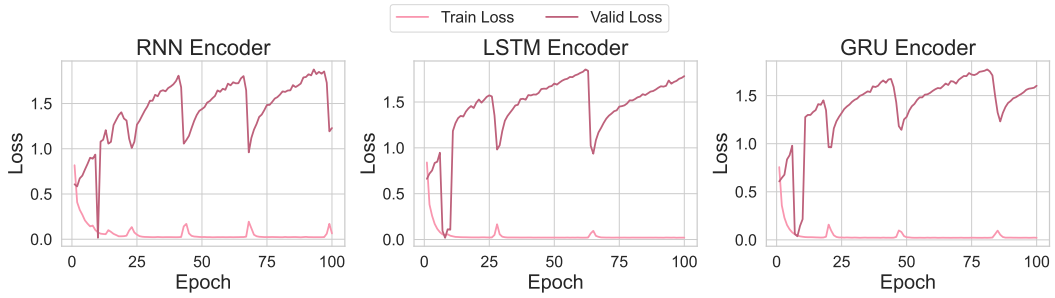


Figure 5: The training and validation loss of the baseline models with three different RNN encoders.

percentage, relationship between utterances in the same conversation is not strong enough, which may explain why history information doesn't really help get better results.

4.3 Further Study (Q2)

In this part, we analyze the training procedures of the models to find the training bottleneck of the models. Besides, we'll answer the question of how the noise of input will affect the performance of SLUBert.

Training procedure. According to the training and validation curve of the baseline model shown if Fig. 5, the training loss is decreasing while the validation loss is oscillating. Also, as is shown if Fig. 6, the validation accuracies are oscillating as well. It's clear that the baseline model is overfitted on the training data. After carefully inspecting the data preprocessing code and the format of the

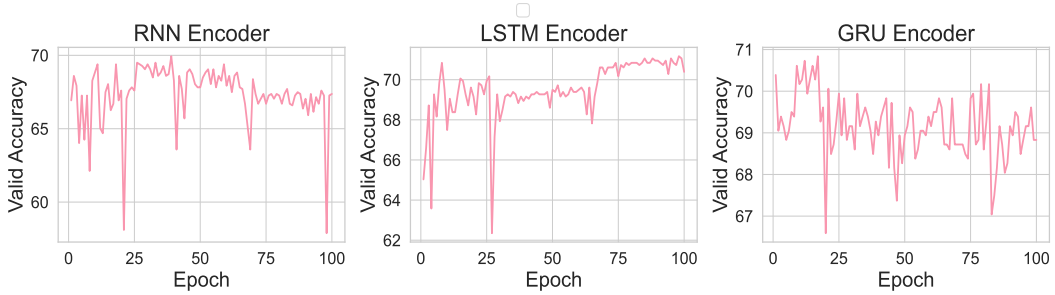


Figure 6: The validation accuracy of the baseline models with three different RNN encoders.

training data, we came to the conclusion that this might be due to the poor quality of the dataset. The true label which we train our model on is derived from tagging the noisy input with the ground-truth tags, while we found that chances were that the true label value didn't even exists in the noisy input, which could result in large amounts of empty labels and thus inhibited the training process. Moreover, there was no doubt that the model could learn nothing from empty labels. So it's very reasonable that the model is overfitted on the training set where vast labels are empty sets. However, the training procedure shown in Fig. 4(a) and Fig. 4(b) represent that our proposed model can effectively alleviate the impact of insufficient dataset on training since our models are quite stronger.

Impact of Noise. In Fig. 3(b), we train the SLUBert-GRU with ASR input or manual input. For fairness of comparison, in the validation process, both models use input with noise. It's clear that the noise in input will weaken the model's performance.

5 Conclusion and Outlook

Despite the excellent performance of our proposed model, there's still much room to improve. After reading [3], we realize the existence of the OOV(Out Of Vocabulary) problem. Given the limited size of training vocabulary, chances are that we may encounter OOV words during validation or testing. Those OOV words will be recognized as <unk>. Given the probability of mistakes happening in asr, and the problem formulation as a tagging problem, OOV words will definitely be a problem towards higher accuracy.

Therefore, pointer network[4] may serve as a better suited formulation of the SLU problem since it's directly extracting sequence of words from the source text which settles the problem of OOV words.

After reading the paper Pointer Networks, the authors came up with an applicable approach to implement pointer network in SLU problem. Use LSTM as encoder to get the final hidden state as the whole sentence as the input to the decoder, which is also LSTM, leverage the attention mechanism as pointer to exact positions. Make all the 74 distinct act-slot pairs as the input to the decoder and output 2 pointer respectively points to the beginning and the end of the value corresponded with the given act-slot pair.

Despite the practicality of using pointer network in SLU, given the high complexity and the time limitation of the project, we just present it here as one promising direction where we can improve our model further.

References

[1] Y.-N. Chen, D. Hakkani-Tür, G. Tür, J. Gao, and L. Deng. End-to-end memory networks with knowledge carryover for multi-turn spoken language understanding. In *Interspeech*, pages 3245–3249, 2016.

[2] L. A. Ramshaw. Text chunking using transformation-based learning, 1995.

[3] A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[4] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

A Implementation Details

We present our implementation details here for reproducibility. Our model and all the baselines are implemented with Python 3.8, PyTorch 1.13.1. All experiments are run on a Tesla V100.

Hyperparameters. We adopt grid search to tune the hyperparameters. The learning rate is searched within $\{0.01, 0.001, 0.0001\}$, dropout is searched within $\{0.0, 0.1, 0.2\}$. Specifically, the hyperparameters are set as follows.

- For baselines, the learning rate is 0.001 and the batch size is set to 32. For BIO-Baseline-LSTM and BIO-Baseline-GRU, the dropout is 0.1, and for BIO-Baseline-RNN, the dropout is 0.
- For SLUBert with manual input, the learning rate is 0.001, the weight decay is set to 0 and the batch size is 32. The dropout is 0.
- For SLUBert with ASR input, the learning rate is first set to 0.01, the weight decay is set to 1e-3 and the batch size is 32. The dropout is 0. After the model has been trained 20 epochs, the learning rate will be decreased to 0.001 and the weight decay will be set to 0.

A.1 Details for BIO-Baseline Experiments

RNN architectures. For BIO-Baseline Experiments, the baseline encoders LSTM, GRU and RNN are implemented in the following manner:

- For LSTM, GRU and RNN, we use 2 layers with hidden size 512 by default.
- The size of word embeddings is set to 768.
- All the RNN models are bidirectional.

Training Details. We use mini-batch gradient descent to optimize our model and the baselines. More concretely, we use the cross entropy loss (or negative log likelihood) and the Adam optimizer is adopted for optimization. The total training epoch number is fixed at 100.

BIO-Baseline with History. For SLU-Baseline models with history conversation, we use a two-layer unidirectional LSTM with hidden size 512 as the encoder. Other hyperparameters are set as default.

A.2 Details for SLUBert Experiments

RNN architectures. For SLUBert Experiments, the encoders LSTM, GRU and RNN are implemented in the following manner:

- **manual input**, For GRU and RNN, we use 1 layer with hidden size 512 by default. For LSTM, the hidden size is set to 256.
- **asr input**, For GRU, RNN and LSTM, we all use 1 layer with hidden size 128 by default.
- The input size(or the dimension of Bert feautures) is 768.
- All the RNN models are bidirectional.

SLUBert-MultiTurn. For models with history conversation, the encoder and contextual_encoder all use a one layer bi-GRU with memory length 256.

Training Details. Mini-batch gradient descent is also used to optimize our SLUBert models. Specifically, we use the cross entropy loss (or negative log likelihood) as our loss function, and the Adam optimizer is adopted for optimization. The total training epoch number for SLUBerts with manual input is fixed at 150, while the max epoch for those with ASR input is set to 300.

B Limitations

Currently, we only experiment on one extremely small dataset. Besides, more complex model instantiations may be developed to achieve better prediction performance on SLU tasks, which we leave for future work.

C Individual Contribution

All three of us contribute equally to the coding, writing and experiment parts.