An Extensive Exploration of Chinese Spoken Language Understanding Models

Yixin Huang, Yufei Wu, Kai Yang
Department of Computer Science and Engineering
Shanghai Jiao Tong Univeristy

Abstract

This study delves into Natural Language Understanding (NLU), specifically focusing on Spoken Language Understanding (SLU) tasks. Our aim is to predict act-slot-value triples from input utterances. We explore various models including the baseline BiLSTM, the History Modeling strategy, CNN-Transformer-based networks, and Hierarchical Decoders, integrating conversation history and pointer networks to tackle challenges like out-of-vocabulary (OOV) terms. Results indicate significant accuracy improvements over baselines, with SDEN-Bert showcasing a 6% accuracy boost.

1 Introduction

Spoken language understanding (SLU), the technique of extracting meaning from speech utterance, is an key component in the task-oriented diaglogue system, providing information for downstream modules. SLU starts with automatic speech recognition (ASR), the task of taking the sound waves and transcribing the audio input to text. In this project, we focuses on what follows ASR, the task of natural language understanding (NLU). We define a semantic unit to be consisting of three parts: action (act), semantic slot (slot), and slot value (value). The goal is to predict a set of act-slot-value triples (in the form of *act(slot=value)*) given an input utterance.

SLU is commonly formulated as a slot tagging task, seeking to attach a class or a label to each of the tokens in the utterance. The simple baseline model adopts BiLSTM to tackle this slot tagging problem. In this work, multiple other utterance encoders and decoders are implemented and discussed. Furthermore, conversation history is carefully incorporated into the models to aid training. We also explore other ways to format the SLU problem with the pointer network. The pointer network also serves to tackle the out-of-vocabulary (OOV) problem, improving the generalization ability of the proposed model.

2 Methodology

2.1 History Modeling

In both training dataset and development dataset, it usually takes multiple utterances to form a complete conversation. However, in the original model, history is not taken into account in slot tagging. Intuitively, utterances in the same conversation can provide additional information to improve the accuracy of tagging.

Memory Network[2] and Sequential Dialogue Encoder Network[1] are proposed to leverage history conversation.

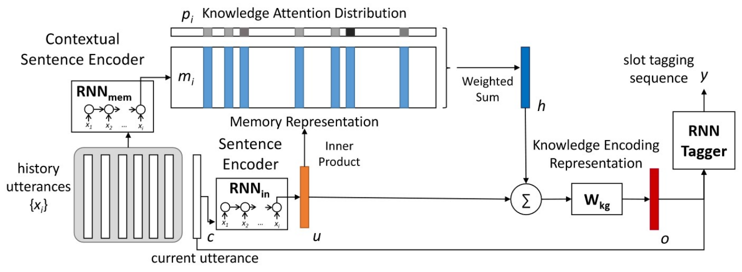
p_i Knowledge Attention Distribution
Contextual Sentence Encoder
RNN_mem
m_i
Weighted Sum
h
slot tagging sequence y
Memory Representation
Sentence Encoder
Inner Product
RNN Tagger
history utterances $\{x_i\}$
RNN_in
Knowledge Encoding Representation
W_kg
c
u
o
current utterance

Figure 1: The architecture of Memory Network

2.1.1 Memory Network

For every current utterance c, it is embedded respectively through an RNN for current use into a vector u and an RNN for memory into a vector m. u and m are of the same dimension. The memory vector m is stored in a memory pool for tagging of future utterances.

Then, compute the attention distribution p_i between u and its history m_i, which is drawn from the memory pool, with an inner product followed by a softmax. To encode the history information, a history vector h is the sum over m_i weighted by the attention distribution p_i.

$$h = \sum_i p_i m_i$$

The current embedding u and the history vector h are combined and passed through a fully-connected neural network W to generate a knowledge vector o.

$$o = W(h + u)$$

Finally, the current utterance c and the knowledge vector o are combined to go through the original tagger to predict slots and values.

$$y = Tagger(o + c)$$

2.1.2 Sequential Dialogue Encoder Network

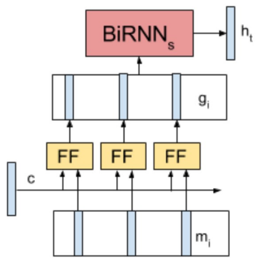

Figure 2: The architecture of Sequential Dialogue Encoder Network. The feed forward layers share weights across all memories.

SDEN is proposed based on MN. Hence, the whole workflow is similar, except the difference in generating history vector h.

The current embedding u and the history m_i are concatenated and passed through a feed forward layer to produce a context encoding vector g_i. Since the number of utterances in a conversation is not certain, g_is are summed weighted by the attention distribution p_i to generate an overall context vector G, which is different from the original algorithm in [1]. Then, G is passed through a BiGRU to produce a history vector h. Except for this, the workflow is the same to that in Memory Network.

2.2 CTRAN: CNN-Transformer-based network

This study in [5] introduces a new encoder–decoder CNN-Transformer-based architecture called CTRAN designed for intent-detection and slot-filling.

The model consists of five modules: a pre-trained language model Bert, a several convolutional layers with different kernel sizes, a following window feature sequence, a stacked Transformer encoders, and an intent-detection decoder with self-attention layer. The specific architecture diagram is as follows in Figure.3.

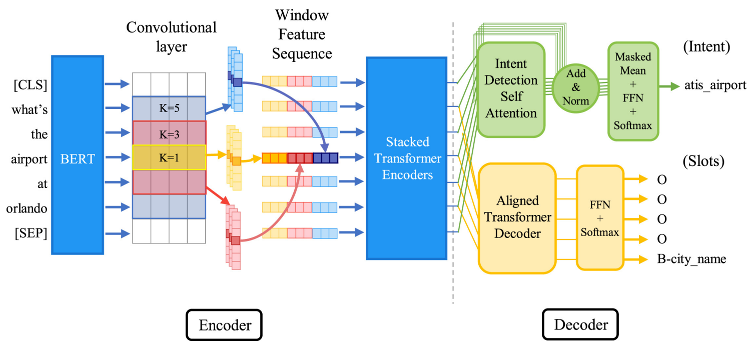

Figure 3: The architecture of CTran

2.2.1 Pre-trained language model

Pre-trained language models have been proven to be effective in improving the performance of the downstream NLP tasks as in [3]. We achieved three pre-trained language models(Bert, WWM, and Roberta) as recommended in our architecture and compare the results to the given word vector.

2.2.2 Convolutional layer

Subsequent to the pre-trained model, we use a transformer encoder instead of an LSTM network. Additionally, we have implemented a CNN to extract features from a token span, which includes extracting features from a single token. This approach places emphasis on the meaning of adjacent tokens, resulting in a more comprehensive local word representation.

Especially,we utilize several convolutional layers with different kernel sizes. Assume the embedding dimension as d, the input sentence has L tokens, and would be defined as $x \in \mathbb{R}^{L\times d}$. Then $x_i \in \mathbb{R}^d$ is the embedding vector for the i-th token. Let k be a filter size and $f \in \mathbb{R}^{k\times d}$ denote a filter-map. Then for each position i in the sentence, there is a window w_i containing k token vectors as $w_i = [x_i, x_{i+1}, x_{i+2}, ..., x_{i+k-1}]$. As $\odot$ is element-wise multiplication, b is bias and A is an activation function. Hence, the convolution operation c_i for each window w_i is given by:

$$c_i = A(w_i \odot f + b)$$

2.2.3 Window feature sequence

Since discontinuous pooling operations can result in the loss of information and disrupt sequential data, we use a window feature sequence to preserve the original word order.

Let V be the total number of filters with different kernel sizes and initial values for the convolutional layer, and j denotes the index referring to the j-th filter. As R_i indicates final embedding vector of the word i and C_i^j indicates i-th element of the j-th filter, we have:

$$R_i = C_i^1 \oplus C_i^2 \oplus C_i^3 \oplus ... \oplus C_i^j \oplus ... \oplus C_i^V$$

Where $\oplus$ represents concatenation, therefore R_i concatenates all of the convolutional values belonging to index i for all of the filter-maps.

2.2.4 Intent-detection decoder

As shown in Fig.3, intent-detection decoder consists of a self-attention layer with residual connection and layer normalization, followed by a linear feed-forward network with softmax applied for classification. Accordingly, the output for the ID self-attention layer is determined by following:

$$S = e + LayerNorm(MultiHead(e))$$

Where $e \in \mathbb{R}^{L \times d_{model}}$ is encoder output, and $LayerNorm$ denotes Layer-Normalization. Moreover, $MultiHead(e)$ denotes the $MultiHead$ attention function where Q, K, V are transformed from e. Furthermore, S represents the output of the operation. Finally, probability distribution between intents is computed as below:

$$P_{intent} = softmax(W \cdot S + b)$$

Note that b and W are parameters to be learned at training time.

2.2.5 Exploration of Connection

During the implementation of the model architecture, we experimented with different connection methods, including serial, parallel, and direct stacking as shown in Fig.4. The experiments showed that the serial connection method resulted in varying accuracies between 40-60%, which were relatively low. On the other hand, parallel connections allowed for the simultaneous preservation of multiple semantic information, resulting in better performance. Stacking prevented semantic redundancy from serial and parallel connections, demonstrating optimal performance with final accuracy stabilising at around 70%.

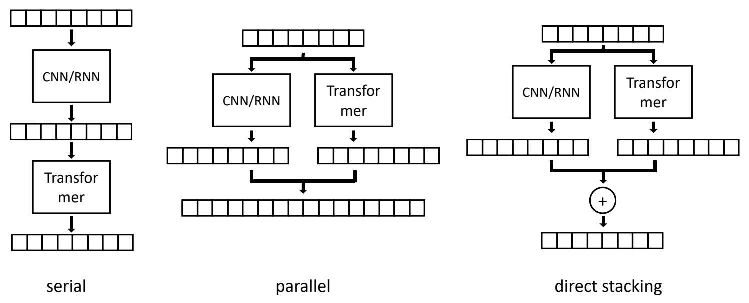

Figure 4: The Exploration of CTran

2.3 Hierarchical Decoder

Different from treating SLU as a slot tagging task as is done in previous sections, [6] proposes to handle the problem as an act-slot-value triple generation task. Since act, slot and the corresponding value are strongly correlated, a hierarchical decoder is built in this paper to better utilize contextual information.

The overall model consists of four modules: a shared utterance encoder, an act type classifier with utterance as input, a slot type classifier with both the utterance and an act type as inputs and a valude decoder with the utterance and an act-slot type pair as inputs.

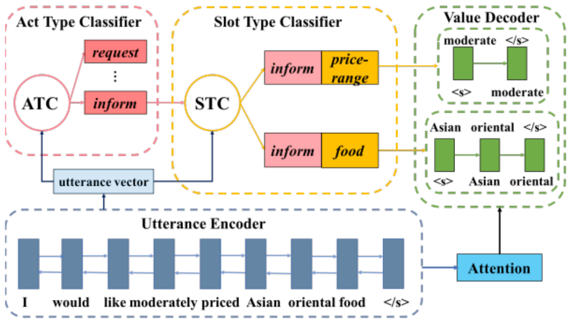

Figure 5: The workflow model of Hierarchical Decoding Model for Spoken Language Understanding . ATC denotes act type classifier and STC denotes slot type classifier

2.3.1 Shared Utterance Encoder

In the original paper, a bidirectional LSTM model is exploited to encode the utterance. e_w denotes the word embedding of the word w. The word embedding is done using Bert as in previous models. The output of BiLSTM could be denoted as:

$$h_i = \overleftarrow{h_i} \oplus \overrightarrow{h_i}; \overleftarrow{h_i} = f_l(\overleftarrow{h_{i+1}}, e_{w_i}); \overrightarrow{h_i} = f_r(\overrightarrow{h_{i-1}}, e_{w_i})$$

where $\overleftarrow{h_i}$ is the hidden vecotr of the backward pass in BiLSTM and $\overrightarrow{h_i}$ is the one of the forward pass. f_l and f_r are both LSTM units. The utterance representation is defined as:

$$\tilde{h} = \overleftarrow{h_1} \oplus \overrightarrow{h_T}$$

The utterance vector $\tilde{h}$ will be used for act and slot classification and the hidden vectors $h = \{h_i, \ldots, h_T\}$ will be used for value sequence generation. To gather more information and sentence features for the complex value sequence generation task, we add a transformer here to enhance the encoding. Plus, the BiLSTM module could be replaced with a bidirectional GRU or RNN. The efficiency will be later tested.

$$h^{trans} = \text{Transformer}(W_e e + b)$$
$$h = \text{LayerNorm}(h + h^{trans})$$

The transformer itself could also act as an utterance encoder. The first token in the sentence input is a special token [CLS] and its encoding best represents the sentence information.

2.3.2 Act and Slot Type Classfiers

Act and slot prediction is formulated as two multi-label classification problem here. As of act type prediction, the input is solely the utterance vector. The existence score for each possible label is calculated as:

$$r = \text{ReLU}(W_u u + b_u)$$
$$p = \sigma(W_r r + b_r)$$

During training, Binary Cross Entropy loss (BCE) is used. In the testing stage, classes with score higher than 0.5 is predicted. If no valid semantics could be extracted from the utterance, the existence scores for all labels should be below 0.5. The ground truth act type vector for this kinds of input is hence a zero vector.

Slot type prediction is formatted in a similar way. To take more contextual information into consideration, the corresponding act vectors are fed into the slot classifier:

$$u = \tilde{h} \oplus e_a$$

It is worth noting that, in the training stage, the act vectors inputted are the ground truth act vectors. This helps to alleviate the cumulative loss in the hierarchical decoder. When performing validation and testing, the act vectors filtered using the 0.5 criterion mentioned above are passed to the slot classifier.

2.3.3 Value Decoder

In the original paper, the value decoder is composed of an LSTM-based sequence-to-sequence model with attention and a pointer network. Following the implementation in [4], the LSTM-based value generator is replaced with a transformer-based one.

The pointor network is implemented to tackle possible OOV problem. The idea is to generate words not only from the train vocabulary, but also the input utterance. The word generation probability is calculated on the extended vocabulary (the basic vocabulary and words in the input utterance). The weights between the basic vocabulary and words in the input utterance is determined by the input sentences and the act-slot pair previously predicted by the classifier. The act-slot pair is encoded using stacked cross-attention layers.

3 Experiments

3.1 Experiment Setup

Both the groundtruth manual transcript and raw outputs of the automatic speech recognition(asr) model are provided in the training dataset, whereas only asr results are available in the test set. To study the influence of input data on the model performance, each model is trained on asr input and manual transcript input respectively. Accuracy, precision, recall rate and F1 score are computed for model evaluation.

Table 1: The experiments result with networks trained and validated both on asr1_best

Model	Accuracy	Precision	Recall	F1 Score
Baseline-RNN	70.53	75.72	74.45	73.76
Baseline-GRU	70.67	79.46	74.83	76.12
Baseline-LSTM	70.84	80.96	73.62	77.12
MN	70.84	78.48	73.41	75.86
MN-Bert	76.65	81.43	78.62	**80.00**
MN-Transformer	71.51	80.41	73.20	76.64
MN-Bert-Transformer	76.42	78.91	78.83	78.87
SDEN	71.06	79.44	73.72	76.47
SDEN-Bert	**76.98**	80.72	**79.04**	79.87
SDEN-Transformer	71.73	80.02	75.60	77.75
SDEN-Bert-Transformer	75.64	77.62	78.83	78.22
CNN-Transformer	72.63	76.21	75.81	76.01
Bert-CNN-Transformer	74.08	77.60	76.96	77.28
Bert-LSTM-Transformer	74.53	78.93	76.96	77.93
WWM-LSTM-Transformer	73.74	**81.61**	76.33	78.88
HD-LSTM	53.07	56.99	56.10	56.54
HD-Transformer-RNN	70.61	76.15	73.93	75.03
HD-Transformer-GRU	66.59	71.78	69.24	70.49
HD-Transformer-LSTM	68.04	72.73	70.91	71.81

3.2 History Modeling

The original MN and SDEN slightly outperform the baseline in accuracy when trained and validated both on asr1_best. The performance of SDEN is consistently better than MN in most settings because of its further extraction of memory features.

Table 2: The experiments result with networks trained on manual transcript and validated on asr1_best. The best performances (recall, F1 score) of models implemented by us are marked in blue.

Model	Accuracy	Precision	Recall	F1 Score
Baseline-RNN	74.75	77.33	78.62	77.97
Baseline-GRU	76.54	79.67	81.75	80.70
Baseline-LSTM	76.65	79.78	**81.86**	**80.80**
MN	76.20	79.28	80.60	79.94
MN-Bert	75.87	77.67	80.50	79.06
MN-Transformer	76.98	79.24	79.98	79.61
MN-Bert-Transformer	76.20	77.71	**81.44**	79.53
SDEN	76.09	78.46	80.50	79.46
SDEN-Bert	75.53	77.25	80.40	78.79
SDEN-Transformer	**77.32**	79.78	**81.44**	**80.60**
SDEN-Bert-Transformer	75.98	77.86	81.02	79.41
CNN-Transformer	76.09	77.86	80.29	79.06
Bert-CNN-Transformer	76.42	78.70	80.92	79.79
Bert-LSTM-Transformer	76.20	78.67	81.13	79.88
WWM-LSTM-Transformer	75.53	77.70	81.02	79.33
HD-Transformer-RNN	74.86	**79.80**	76.23	77.97
HD-Transformer-GRU	72.18	75.87	74.77	75.32
HD-Transformer-LSTM	73.07	77.47	75.29	76.36

Bert is applied to replace the provided word2vec-768.txt, and the parameters of Bert are not updated during training. MN and SDEN with Bert outperforms the original ones by a large margin. Besides, transformer is added to the tagger to combine with RNN. When trained and validated both on asr1_best, transformer can only slightly improve the performance.

The same models are then trained on manual transcripts, which bear less noise, and validated on asr1_best. Models without Bert gain a dramatic improvement on performance, while 3 of the models with Bert suffer an obvious decrease in performance. Hence, using training dataset with less noise is double-edged. The model can fit the task better, but the risk of overfitting may increase.

Memory Network and Sequential Dialogue Encoder Network enables the model to process multiple utterances in a single conversation. For instance, the 3 utterance inputs in one conversation is 导航, 凤凰湖私家菜 and 第一个 with corresponding semantic triples inform-操作-导航, inform-poi名称-凤凰湖私家菜 and inform-序列-第一个 can be correctly predicted. Besides, MN and SDEN don't result in changes in tagger, making it possible to insert memory structures into most kinds of SLU models. Therefore, they can be flexibly applied in various models to improve performances on multiple utterances.

3.3 CTran

The original CTran architecture integrates a shared encoder, a Slot-Filling (SF) decoder, and an Intent-Detection (ID) decoder, fostering joint learning by utilizing vectors produced by the encoder for both SF and ID tasks. However, since our task has no sufficient data for ID, we depart the joint decoder as single self-attention decoder.

The slot-filling decoder comprises an aligned Transformer decoder and a dimension-reducing linear layer. Alignment in the cross-attention segment occurs through masking, aligning keys and values transformed from encoder output. This alignment focuses the Transformer decoder on specific embedding vectors, improving token generation accuracy. The decoder further utilizes self-attention to perceive relations between previously generated tokens and cross-attention to extract information from the encoder.

CTran's mechanisms significantly elevate the extraction of semantic understanding from contextual information. The integration of the Window Feature Sequence and attention mechanisms within CTran enables a profound comprehension of context, allowing the model to grasp nuanced relationships between words and phrases within the discourse. By leveraging these mechanisms, CTran excels in capturing the contextual essence, thus enhancing its ability to decode and interpret intricate semantic nuances present in spoken language. This heightened contextual awareness facilitates more accurate predictions of act-slot-value triples by comprehensively considering the surrounding context, thereby advancing the model's proficiency in spoken language understanding tasks.

Simultaneously, CTran's dependency on context through mechanisms like the Window Feature Sequence and attention proves robust but also susceptible to noise, notably evident in the 'asr_1best' dataset. Consequently, certain predictions are prone to noise interference. For instance, phrases like "帮我导航我导航一下看到你的学段时间" lack training data, yet result in erroneous triples like "inform-导航-学段时间". Handling such noise remains a challenge, as the model struggles to differentiate and rectify such instances where context blurs into ambiguous or erroneous utterances, impacting the accurate prediction of act-slot-value triples.

3.4 Hierarchical Decoder

The original hierarchical decoder model yields inferior results when trained and validated both on asr_1best. However, the performance skyrockets with the incorporation of the transformer in the encoder. Possible explanation is that the output of the transformer at every single position takes the whole sentence into consideration, which further alleviates of the long-term forgetting problem in LSTM-related works. Further, the decoder has a far larger scale than the encoder. This imbalance may also incur problem. Due to the poor performance of the hierarchical decoder without transformer, no further experiments are conducted using this model design.

The encoder cell implemented originally using BiLSTM could be replaced with bidirectional GRU and RNN. Different from other methods implemented, the model equipped with an RNN encoder performs best. Possible reasons may that our dataset is relatively small and the utterance length is short compared to datasets in other NLP tasks. Exceeding model complexity leads to inferior model performance.

Compared with other models, hierarchical decoder hardly shows any superiority on evaluation metrics. The reason may lie in the value decoder's flexibility. The predicted value may include repetition of words in the extended vocabulary despite the conciseness of the input sentence. Also, the model handles long sentences or long objects poorly. The end of the value is signaled by the generation of the special token [SEP] by the value decoder. More corpus should be provided to let the model better learn proper ways to halt the value generation.

However, the pointer network enables the model to handle OOV values, which outstands the other models implemented. For example, the utterance input is 到临沅湘河镇 and the corresponding semantics triples is inform-终点名称-临沅湘河镇. 沅 is out of the train dataset and thus couldn't be correctly parsed using ordinary methods. The implemented hierarchical decoder successfully parses this utterance, which shows its capability of handling OOV words. However, pointer network can't handle words outside the extended vocabulary, which could possibly occurs when the input is noisy. Methods that are based on common knowledge should be adopted to tackle this issue.

4 Conclusion

We carried out wide explorations of Spoken Language Understanding, including basic architecture and further improvements. We implemented a CNN-Transformer-based network, a different architecture from the provided baseline. History in a conversation with multiple utterances was taken into consideration using memory structures, Memory Network and Sequential Dialogue Encoder Network. In order to explore architecture besides slot tagging method, Hierarchical Decoder was implemented, treating SLU as an act-slot-value triple generation task. Because of the different definition of the problem, Hierarchical Decoder can also handle out-of-vocabulary values.

SDEN-Bert turns out to perform best when trained and validated both on asr1_best, outperforming the baseline by over 6% in accuracy. SDEN-Transformer gives the best performance when trained on manual transcript and validated on ast1_best. Our results show that all implemented model can either outperform the baseline or handle certain problems in SLU.

Future work will explore the possibility to combine the solutions to tackle different problems in SLU into a single model, and still maintain excellent performances.

References

[1] Ankur Bapna, Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. Sequential dialogue context modeling for spoken language understanding. *arXiv preprint arXiv:1705.03455*, 2017.

[2] Yun-Nung Chen, Dilek Hakkani-Tür, Jianfeng Gao, and Li Deng. End-to-end memory networks with knowledge carryover for multi-turn spoken language understanding. 2016.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Chen Liu, Su Zhu, Zijian Zhao, Ruisheng Cao, Lu Chen, and Kai Yu. Jointly encoding word confusion network and dialogue context with bert for spoken language understanding, 2020.

[5] Mehrdad Rafiepour and Javad Salimi Sartakhti. Ctran: Cnn-transformer-based network for natural language understanding. *Engineering Applications of Artificial Intelligence*, 126:107013, 2023.

[6] Zijian Zhao, Su Zhu, and Kai Yu. A hierarchical decoding model for spoken language understanding from unaligned data. *CoRR*, abs/1904.04498, 2019.

A Implementation Details

The implementation details are provided for reproducibility.

A.1 History Modeling

All models are trained for 100 epoches on the given dataset. Current encoder and memory encoder are of 1 RNN layer, and the RNN used in the tagger is of 2 layers.

Table 3: Key Hyperparameter Setting for History Modeling. The model structure is kept the same between models trained on manual transcript and those trained on asr

Model	learning rate	current encoder	memory encoder	tagger RNN	tagger dropout
MN	1e-3	LSTM	LSTM	GRU	0.2
MN-Bert	1e-3	LSTM	LSTM	GRU	0.2
MN-Transformer	1e-3	LSTM	LSTM	LSTM	0.15
MN-Bert-Transformer	3e-4	LSTM	LSTM	GRU	0.05
SDEN	1e-3	LSTM	LSTM	LSTM	0.2
SDEN-Bert	1e-3	LSTM	LSTM	LSTM	0.2
SDEN-Transformer	1e-3	LSTM	LSTM	GRU	0.1
SDEN-Bert-Transformer	1e-3	LSTM	LSTM	LSTM	0.2

A.2 CTRAN

All models are trained for 100 epoches on the given dataset.

Table 4: Key Hyperparameter Setting for CTran related models. The model structure is kept the same between models trained on manual transcript and those trained on asr

Model	RNN layers	learning rate	bert dropout	num transformer layer
CNN-Transformer	/	1e-3	0.2	2
Bert-CNN-Transformer	/	1e-3	0.2	2
Bert-LSTM-Transformer	2	1e-3	0.5	3
WWM-LSTM-Transformer	2	1e-3	0.5	2

A.3 Hierarchical Decoder

All models are trained for 150 epoches on the given dataset. The optimizer used is the BertAdam optimizer, with Adams b1, b2 and epsilon set to 0.9 and 0.999 and 1^{-6} respectively. The weight decay factor is set to 0.01.

Table 5: Key Hyperparameter Setting for Hierarchical Decoder. The model structure is kept the same between models trained on manual transcript and those trained on asr

Model	RNN layers	learning rate	bert dropout	num transformer layer
HD-Transformer-RNN	3	5×10^{-4}	0.3	3
HD-Transformer-GRU	3	5×10^{-4}	0.3	2
HD-Transformer-LSTM	3	5×10^{-4}	0.3	2

B Contribution

- Kai Yang: CTRAN
- YuFei Wu: History Modeling (memory network and sequential dialogue encoder network)
- Yixin Huang: Hierarchical Decoder