Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. Hermann Ney

Seminar Selected Topics in Human Language Technology and Pattern Recognition in WS 2018

Neural Network Based Natural Language Understanding

Simon Berger

Matrikelnummer 336 353

December 10, 2018

Supervisor: Kazuki Irie

Contents

List of Tables

List of Figures

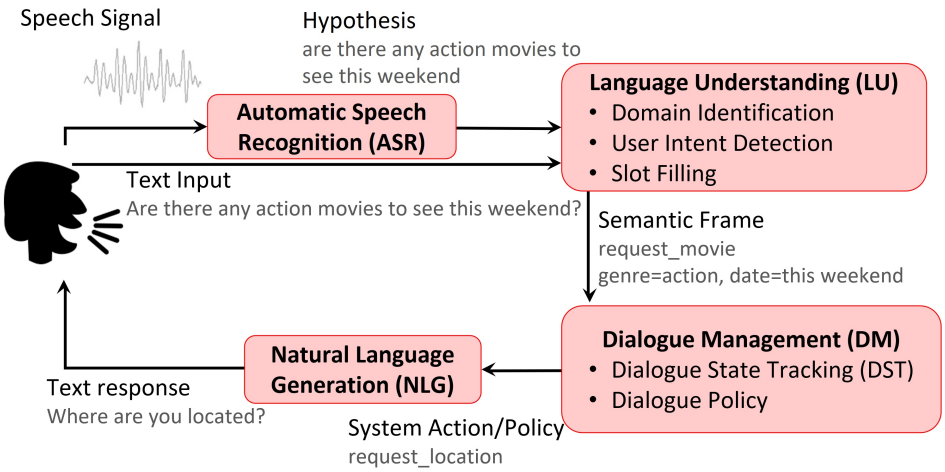

Figure 1: Dialogue-system pipeline.
Source: https://www.csie.ntu.edu.tw/ yvchen/s105-icb/project.html.

1 Introduction

2 The task of Natural Language Understanding

Natural language understanding (NLU) is a key component in any dialogue-system between human and machine. It consists of the task of *'understanding'* a human utterance given in natural language, i.e. forming a semantic frame that captures the semantic meaning of the utterance that the machine can then use. In figure 1 we see the usual pipeline of handling and responding to a natural language utterance in a dialogue-system.

The first step is parsing the raw acoustic-speech signal to written text using an *Automatic Speech Recognition* module. For example the speech-signal might be parsed to the sentence 'Are there any action movies to see this weekend?' If the human query is given in text-form this step can of course be skipped. Afterwards we need to assign semantic meaning to the given text. This is the step of *language understanding* (LU) that we focus on in this article. For example we might classify the domain of the request as *'movies'*, the intent of the query as a *'movie-request'* and extract the genre-information *'action'* and the date-information *'this weekend'*. Using this semantic information the *Dialogue-Manager* (DM) can then decide on the machine's course of action like for example retreiving the sought-after information from the web or requesting additional input from the user. The results then get parsed back into natural language using a *Natural Language Generator* to make them understandable for the user.

As indicated in the above example, NLU usually consists of three major subtasks:

- Domain Detection: Finding the domain of the request (e.g. *movies, travel, weather*, etc.)
- Intent Determination: Determining the domain-specific intent of the user (e.g. *request_movie, book_flight, get_weather*, etc.)
- Slot-filling: Extracting associated semantic slots for the intent (e.g. *departure_city, destination_city, date*, etc. for intent *book_flight*).

In the following subsections we will describe the mathematical formulation of the three subtasks in detail.

2.1 Domain Detection task formulation

Domain detection is typically handled as a semantic utterance classification problem [8] which means that it takes the word-sequence

$$x_1^T = (x_1, x_2, \ldots, x_T) \in V^T$$

for some vocabulary-set V as an input and assigns a single domain-label y^D as the output. The possible domain-labels must be designed manually before the classification.

2.2 Intent Determination task formulation

Intent determination is similar to domain detection in the sense that is also treated as a semantic utterance classification problem [8]. After the domain was determined, another label y^I must be assigned to the sequence x_1^T, representing the domain-specific intent of the user. Again the set of possible intents have to be manually predetermined before performing the classification.

2.3 Slot Filling task formulation

In constrast to the other two subtasks, slot filling is treated as a sequence-labeling problem [9], which means that we want to assign an output-sequence

$$y^S = (y_1^S, y_2^S, \ldots, y_T^S)$$

of semantic slot-labels to the input x_1^T. As before the intent-specific possible slots have to be manually crafted before. Since in many cases multiple words can comprise a single semantic entity as for example in *'New York'* or *'President of the United States'*, it is popular to define the slot-labels in an *Inside-Outside-Beginning* (IOB) representation. The first word in a semantic unit has a B-label, each following word belonging to the same semantic slot gets an I-label and each word not belonging to any semantic slot gets an O-label. An example of such slot-labeling in IOB-representation can be found in figure 2.

W	find	comedies	by	james	cameron
	↓	↓	↓	↓	↓
S	O	B-genre	O	B-dir	I-dir
I	find_movie				

Figure 2: Example sentence.
An example sentence of words **W** with slot-labels **S** in IOB-representation and Intent **I**.
Source: [2].

Here, the words *'find'* and *'by'* don't belong to any semantic slot. The word *'comedies'* belongs to the semantic concept *'genre'* and has a B-tag since it is the first (and only) word in it's chunk. At last, both words *'james'* and *'cameron'* belong to the same semantic concept *'dir'* (director) with the first word *'james'* having a B-tag and the second word *'cameron'* having an I-tag.

3 State of the art sequence-chunking model for Slot Filling

As we can see from the task formulation, all of the three subtasks involve sequences of words as input, so it quickly suggests itself to try and tackle these problems using *Recurrent Neural Networks* (RNNs). And indeed, although the best results used to be produced by *Conditional Random Fields* (CRFs) [11], in recent years different RNN-models proved very successful and improved the state of the art further and further (c.f. [2], [5], [8], [9], [15], [16], [17]). We will now present one of the most successful models for Slot Filling, from the paper *'Neural Models for Sequence Chunking'* by Zhai et al. [16], in detail.

Traditionally, Neural Network (NN) based approaches to Slot Filling have no explicit way of identifying semantic chunks in a sentence and rather infer these implicitly by the IOB-labels explained in section 2.3. The idea of the model in [16] is to divide the Slot-Filling-task up into two sub-tasks:

- Segmentation of the sentence into semantic chunks;
- Labeling each chunk as a single unit.

We can think of this as not labeling the words with both their IOB-label and slot-label simultaneously but instead *first* identifying the segments and then *afterwards* assigning a single slot-label to each one. For example in the sentence *'find comedies by james cameron'* from figure 2, the words *'find', 'comedies'* and *'by'* would be labeled as single-word-chunks and the phrase *'james cameron'* as a two-word semantic chunk. Then in the second phase the slot-label *'genre'* would be assigned to the chunk *'comedies'*, the label *'director'* to the chunk *'james cameron'* and empty slot-labels to *'find'* and *'by'*.

To achieve this, the paper proposed three different models. For this article we will focus on the third model, which produced the best results in all regards. But first, we have to explain the tools needed to build the model, beginning with the concept of *Long Short Term Memory* (LSTM) networks in the next section.

3.1 Long Short Term Memory Networks

Traditional RNNs - which have recurrent connections from their own hidden states (for Elman-type) or output-states (Jordan-type) to the hidden states at the next time-step [8], as represented in figure 3 - have a problem of vanishing gradients. This means - roughly explained - that although such a network is theoretically able to capture long-term dependencies of input-words, the gradients during gradient-descend-training become vanishingly small since they decrease exponentially with the number of time steps, which slows down training or even prevent weights from changing at all due to machine-inaccuracy [1].

A common solution to this problem comes in the form of so called Long Short Term Memory (LSTM) networks. These special type of RNNs consist of several layers interacting in a certain way so that information can flow along it unchanged. This allows gradients to flow nicer over time. In detail, a LSTM-module consists of an *input-gate* i, a *forget-gate* f and an *output-gate* o, as well as a memory cell c and a hidden state h. Their values at

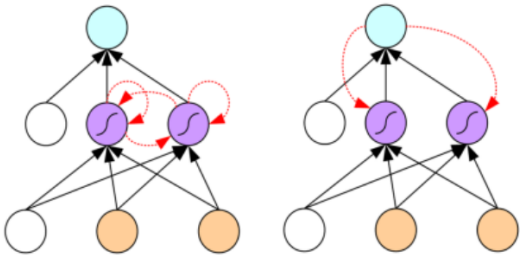

Figure 3: RNN architecture.
Traditional RNNs of Elman-Type (left) and Jordan-Type (right). Source: [8].

timestep t are computed as follows (see [16]):

$$\begin{aligned} i_t &= \sigma(W^i x_t + U^i h_{t-1} + b^i) \\ f_t &= \sigma(W^f x_t + U^f h_{t-1} + b^f) \\ o_t &= \sigma(W^o x_t + U^o h_{t-1} + b^o) \\ g_t &= \tanh(W^g x_t + U^g h_{t-1} + b^g) \\ c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\ h_t &= o_t \otimes \tanh(c_t) \end{aligned}$$

Hereby, $\otimes$ denotes element-wise multiplication, $\sigma(\cdot)$ is the element-wise sigmoid-function and $\tanh(\cdot)$ the element-wise hyperbolic-tangent function. Also $W^i, W^f, W^o, W^g U^i, U^f, U^o$ and U^g are shared weight-matrices and b^i, b^f, b^o and b^g are shared bias-vectors. A diagram illustrating the function of such a module is illustrated in figure 4. Because of their ability to capture long-term dependencies very well, LSTMs or similar modules such as *Gated Recurrent Units* (GRU) (see for example [17]) have been the preferred choice for many researchers of NLU in recent years ([5], [10], [14], [16], [17])

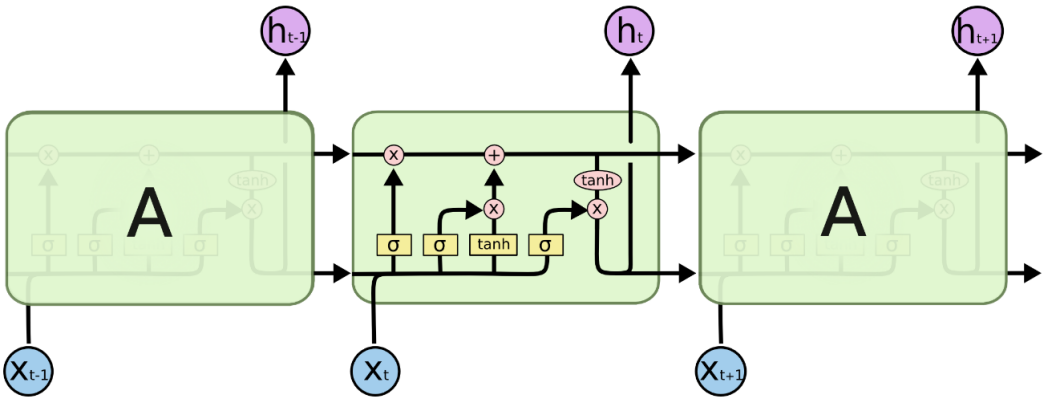

Figure 4: LSTM-Module.
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

3.2 Bidirectional LSTMs

Since in the task of NLU we have the full sentence at our disposal before starting the classification, instead of only encoding information about the previous words $x_1, \dots, x_{t-1}$ of x_t into the LSTM, we can also go backwards through the sentence to encode information about the following words $x_{t+1}, \dots, x_T$. To account for information about the whole sentence when computing the hidden-state for the word x_t, we can combine these two approaches to a so called *bi-directional* LSTM (Bi-LSTM): First, the word-sequence $x_1, \dots, x_T$ gets run through the LSTM to compute hidden-states $\overrightarrow{h_1}, \dots, \overrightarrow{h_T}$. Then, the word-sequence again gets run through the LSTM but in reverse order $x_T, \dots, x_1$ to compute hidden-states $\overleftarrow{h_T}, \dots, \overleftarrow{h_1}$. The hidden-state $\overleftrightarrow{h_t}$ of the Bi-LSTM at time-step t is then generated as the concatenation $\overleftrightarrow{h_t} = [\overleftarrow{h_t}, \overrightarrow{h_t}]$ (c.f. [14],[16]).

3.3 Convolutional Neural Networks

Another tool that is used in the model of [16] are *Convolutional Neural Networks* to compute feature vectors for each chunk to use for labeling. In this article we will not explain CNN-architecture in detail (for that see for example [4]), just briefly how it is used: a CNN with m filters and a filter-size of n (and a stride of 1) is used to compute m-dimensional feature-vectors for each n-gram in the chunk. Afterwards a max-pooling-layer over all of these feature-vectors gets us a single feature-vector for the whole chunk that can be used for labeling.

By using a CNN for feature-extraction we don't have to be concerned about the number of words in each chunk and instead have a single feature-vector of fixed size.

3.4 Pointer Networks

In the model, the chunks will be identified in a greedy way, i.e. one chunk after the other. Since all chunks are adjacent, this means we will only ever have to identify the end-point of the next chunk because the start is given by the end-point of the last one. This behaviour can be realized by the use of so called *Pointer Networks* (PN) which were proposed by Vinyals et al. in [13]. The goal of a PN is to match sequences $\mathcal{P} = \{P_1, \dots, P_n\}$ of vectors to sequences of indices $\mathcal{C}^{\mathcal{P}} = \{C_1, \dots, C_m\} \subseteq \underline{n}^m$ of these vectors, i.e. *pointers* to vectors in $\mathcal{P}$. The probability of a pointer-sequence $\mathcal{C}^{\mathcal{P}}$ given $\mathcal{P}$ is calculated as

$$p(\mathcal{C}^{\mathcal{P}}|\mathcal{P}) = \prod_{i=1}^{m} p(C_i|C_1, \dots, C_{i-1}, \mathcal{P})$$

so Vinyals et al. designed a parametric model (a RNN with parameters θ) to estimate the terms of the product

$$p_\theta(C_i|C_1, \dots, C_{i-1}, \mathcal{P}; \theta) \approx p(C_i|C_1, \dots, C_{i-1}, \mathcal{P}).$$

To estimate this probability they proposed an encoder-decoder-framework, i.e. using one *encoder*-RNN that runs over $\mathcal{P}$ to produce a hidden-state sequence $e_1, \dots, e_n$, whose final hidden state e_n is then used to initialize a second *decoder*-RNN that produces the sequence $d_1, \dots, d_m$ of states. They then calculated the probability via

$$\begin{aligned} u_j^i &= v^{tr} \tanh(W_1 e_j + W_2 d_i) \qquad \text{for } j \in \{1, \dots, n\} \\ p_\theta(C_i|C_1, \dots C_{i-1}, \mathcal{P}; \theta) &= \text{softmax}(u^i) \end{aligned}$$

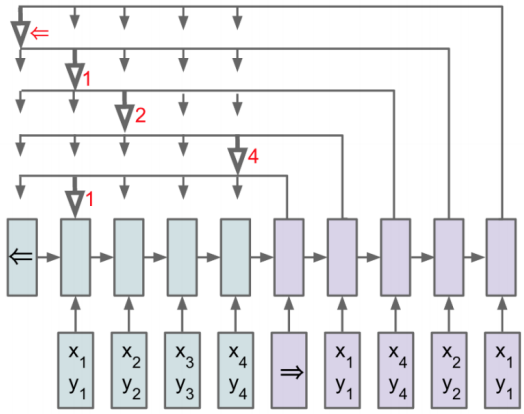

Figure 5: PN architecture.
The input-sequence is first encoded with an encoder-network (blue) to initialize the decoder-network (purple). The outputs of the decoder are then used to point to one of the input-words via softmax. Source: [13].

Hereby W_1, W_2 and v as well as the parameters of the decoder and the encoder-RNN are trainable. A visualization of the PN is given in figure 5.

We can now easily see how the concept of PNs applies to the problem of identifying chunks: the sequence $\mathcal{P}$ becomes our input-word-sequence and the index-sequence $\mathcal{C}^{\mathcal{P}}$ is the sequence of start/end-points of the chunks. How exactly PNs are applied in the chunking model will be explained in the next section.

3.5 The Model

Now that we have explained all tools needed, we will describe the full model of Zhai et al. [16] in detail. Note that some of the notation was changed to make for better consistency.

The model follows an encoder-decoder framework similar to the one described in section 3.4. We have an encoder-network in the form of a Bi-LSTM and a decoder-network in the form of a usual LSTM. After the whole sentence is encoded, the information is used to greedily identify semantic chunks with a PN which are then immediately used as input to the decoder. The output of the decoder is used for labeling the current chunk.

The encoder uses word-embeddings $(Ex_1, \dots, Ex_T)$ of the words $(x_1, \dots, x_T)$ of the input sequence, which are provided in *one-hot-representation* (i.e. vectors of size $|V|$ with a 1 at the correct index and 0's otherwise) and produces the hidden-state sequence $\overleftrightarrow{h_1}, \dots, \overleftrightarrow{h_T}$ for a matrix $E \in \mathbb{R}^{m \times |V|}$. The sentence-representation $[\overrightarrow{h_T}, \overleftarrow{h_1}]$ (i.e. the concatenation of the final hidden states of the forwards- and of the backwards-LSTM) is used to initialize the decoder-LSTM, together with an special starting input token. The process of continually identifying chunks and labeling them now works as follows:

For a chunk beginning at time step b (for the first chunk this is the start of the sentence, for later chunks the beginning is set at the end of the previous chunk) the probabilities of all possible endpoints are calculated in a way that is similar to the Pointer Networks from

section 3.4. First, we define the vector

$$u_j^i = v_1^{tr} \tanh(W_1 \overleftrightarrow{h_j} + W_2 E x_j + W_3 E x_b + W_4 d_i) + v_2^{tr} LE(j - b + 1)$$

with $j \in \{b, \ldots, b + l_m - 1\}$. Here, i is the index of the chunk, v_1 and v_2 are trainable vectors, W_1, W_2, W_3 and W_4 trainable matrices, l_m is a hyperparameter for the maximum chunk length and $LE(j-b+1)$ is a chunk-length embedding that is not precisely specified in [16] but is also a trainable parameter. This is comparable to the feature-vector for PNs but instead of only having the encoder hidden-state $\overleftrightarrow{h_j}$ and the previous decoder hidden-state d_i as the input, we add the word embeddings of the beginning word x_b and the word at the current position x_j, as well as an embedding for the length of the chunk. Now the probabilities of chunk i ending at position j are given by

$$p_i(j) = \frac{exp(u_j^i)}{\sum_{k=b}^{b+l_m-1} exp(u_k^i)},$$

i.e. $p_i \equiv softmax(u^i)$. The ending point e can now be greedily chosen as the point with maximum probability

$$e := \underset{j \in \{b, \ldots, b+l_m-1\}}{\operatorname{argmax}} (p_i(j)).$$

After identifying the scope of the chunk, it will now be used to compute the input C_{i+1} to the decoder for the next time-step. This input consists of several parts:

- First, we compute the feature extraction of the current chunk via a CNN:

  $$C_{i+1,x} := g(Ex_b, \ldots, Ex_e)$$

  where $g(\cdot)$ gives us the output of the CNN (in the paper [16], this step is depicted with using the x_j directly as input to the CNN but this is presumably inaccurate, or the embedding was included in g).

- Second, we compute a representation of the hidden states of the encoder for the words in the chunk by

  $$C_{i+1,h} := \text{Average}(\overleftrightarrow{h_b}, \ldots, \overleftrightarrow{h_e})$$

- Third, we use a context-window to capture context information for the chunk. With context-window size d (which is a tunable hyperparameter) this means

  $$C_{i+1,w} := [Ax_{b-d}, \ldots, Ax_{b-1}, Ax_{e+1}, \ldots, Ax_{e+d}]$$

  is the concatenation of the d words preceeding and the d words following chunk i.

- Now $C_{i+1} := [C_{i+1,x}, C_{i+1,h}, C_{i+1,w}]$ is the concatenation of the three components.

The output of the decoder can be used for labeling, for example if the outputs are viewed as probabilities for the slot labels via a softmax-layer, we can assign the label with maximum probability.

A diagram depicting the model can be found in figure 6.

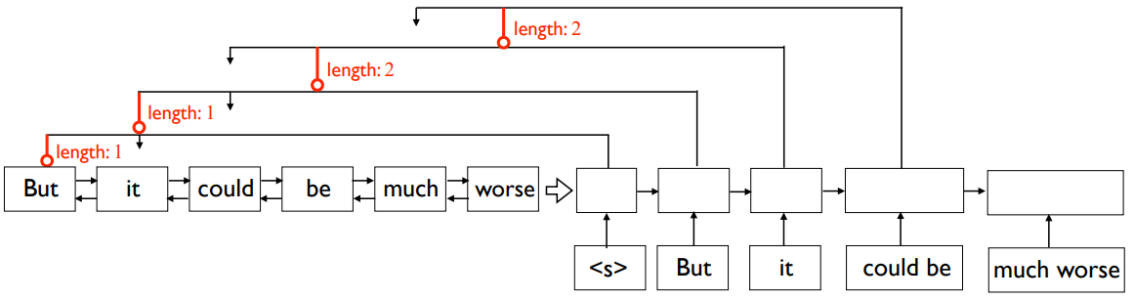

Figure 6: Encoder-decoder-pointer framework.
The presented framework used on the example sentence *'But it could be much worse'*. On the left is the encoder-BiLSTM, which is used to encode the sentence for the decoder-LSTM on the right. Inputs to the decoder are the chunks identified in the previous timestep. A PN with a maximum chunk length of 2 calculates the probabilities of the possible ending points. The chosen ending point is pointed to in red color. Source: [16].

3.6 Training criterion

Since the task essentially is split into two subtasks which are solved simultaneously (segmentation and labeling), there are also two loss functions to be considered. Zhai et al. used the sum of the cross-entropy loss for both subtasks as loss function for the whole model:

$$L(\theta) := L_{\text{segmentation}}(\theta) + L_{\text{labeling}}(\theta)$$

where $L_{\text{segmentation}}(\theta)$ is the cross-entropy loss for the segmentation task and $L_{\text{labeling}}(\theta)$ the cross-entropy loss for the labeling task with trainable parameters θ. The model is then trained by stochastic gradient descent on the loss function $L(\theta)$.

3.7 Performance on Slot Filling task

In this section we will describe, how well the model performed at the Slot Filling task of NLU. Zhai et al. also reported the performance on the sequence chunking task but since this is not as relevant for NLU we will omit these results for now.

The performance was evaluated on two different datasets:

- The *Airline Travel Information System* (ATIS) corpus. This dataset consists of 4,978 training and 893 test sentences from the airline travel domain. It has a vocabulary size if 572 and contains 84 different slot-labels (127 if counting IOB prefixes). This dataset is widely regarded as one of the standard datasets for evaluating NLU performance and thus has been used by many researches (e.g. in [2], [5], [8], [9], [10], [15], [16], [17]). For more information on how this dataset was collected and transcribed, see [3].

- A larger dataset (LARGE) prepared by merging the ATIS corpus with the MIT Restaurant Corpus and the MIT Movie Corpus [6] [7]. In total this large dataset contains 30,229 training and 6,810 testing sentences. The vocabulary size is 16,049 and the number of slot labels is 116 (194 if counting IOB prefixes). This dataset is

obviously much larger than the ATIS dataset and has three different domains so it poses more of a challenge. However it is not as widely used so there are few other models to compare the score on this dataset to (we only compare to [5]).

The performance is measured by the F_1-Score which is the harmonic mean of *Precision* and *Recall* value:

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

where

$$Precision = \frac{\#\text{correctly labeled words}}{\#\text{total labeled words}}$$

and

$$Recall = \frac{\#\text{correctly labeled words}}{\#\text{words with nonempty label as ground truth}}$$

([12]). The best possible score is 1 (perfect labeling) and the worst is 0 (no correct label). This F_1 score is commonly used for evaluating classification tasks. Table 1 shows the F_1 score of the presented model compared to different Neural Network based models from some other papers of the past few years on the ATIS and the LARGE dataset.

Model	ATIS F_1	LARGE F_1
Bi-dir. Jordan RNN (Mesnil et al., 2013) [9]	93.98%	-
RNN (Yao et al., 2013) [15]	94.11%	-
Hybrid RNN (Mesnil et al., 2015) [8]	95.06%	-
Slot Gated (Goo et al., 2018) [2]	95.20%	-
Joint ID + SF (Zhang et al., 2016) [17]	95.49%	-
Enc.-Dec. LSTM (Kurata et al., 2016) [5]	95.66%	74.41%
Sequence Chunking (Zhai et al., 2017) [16])	**95.86%**	**78.49%**

Table 1: F_1-Score comparison of different models on ATIS and LARGE dataset

As we can see from table 1, the presented sequence chunking model by Zhai et al. produces state of the art results and stands as one of the most successful models for the Slot Filling task today.

4 Models for Intent Determination

5 Joint models for multiple subtasks

6 Conclusion

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[2] C.-W. Goo, G. Gao, Y.-K. Hsu, C.-L. Huo, T.-C. Chen, K.-W. Hsu, and Y.-N. Chen. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 753–757, June 2018.

[3] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington. The atis spoken language systems pilot corpus. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '90, pages 96–101. Association for Computational Linguistics, 1990.

[4] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751, Oktober 2014.

[5] G. Kurata, B. Xiang, B. Zhou, and M. Yu. Leveraging sentence-level information with encoder LSTM for natural language understanding. *CoRR*, abs/1601.01530, 2016.

[6] J. Liu, P. Pasupat, S. Cyphers, and J. Glass. Asgard: A portable architecture for multilingual dialogue systems. In *Proc. ICASSP*, pages 8386–8390, 2013.

[7] J. Liu, P. Pasupat, Y. Wang, S. Cyphers, and J. Glass. Query understanding enhanced by hierarchical parsing structures. In *Proc. ASRU*, pages 72–77, 2013.

[8] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, and G. Zweig D. Yu. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, March 2015.

[9] G. Mesnil, X. He, L. Deng, and Y. Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. pages 3771–3775. Proc. Interspeech, January 2013.

[10] S. Ravuri and A. Stolcke. Recurrent neural network and lstm models for lexical utterance classification. In *Proc. Interspeech*, pages 135–139. International Speech Communication Association, September 2015.

[11] C. Raymond and G. Riccardi. Generative and discriminative algorithms for spoken language understanding. In *Proceedings of the Eigth Annual Conference of the International Speech Communication Association*, 2007.

[12] J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[13] O. Vinyals, M. Fortuanto, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28*, pages 2692–2700, September 2015.

[14] X. Yang, Y.-N. Chen, D. Hakkani-Tür, P. Crook, X. Li, J. Gao, and L. Deng. End-to-end joint learning of natural language understanding and dialogue manager. *CoRR*, 2016.

[15] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu. Recurrent neural networks for language understanding. Interspeech, August 2013.

[16] F. Zhai, S. Potdar, B. Xiang, and B. Zhou. Neural models for sequence chunking. January 2017.

[17] X. Zhang and H. Wang. A joint model of intent determination and slot filling for spoken language understanding. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2993–2999. AAAI Press, 2016.