A study on Abstract Meaning Representation parsing

Lai Dac Viet

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
March, 2018

Master's Thesis

A study on Abstract Meaning Representation parsing

1610204 Lai Dac Viet

Supervisor :	Nguyen Le Minh
Main Examiner :	Nguyen Le Minh
Examiners :	Nguyen Le Minh
	Satoshi Tojo
	Kiyoaki Shirai
	Shogo Okada

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science

February, 2018

Abstract

Text parsing is a core research topic in natural language processing (NLP). The output representation of text parsing is considered as vital seeds for NLP applications. Although text parsing has been studied comprehensively and intensively for decades, it is still a tough but attractive field. This research focuses on developing a novel parser for Abstract Meaning Representation (AMR). AMR is a powerful semantic representation at the sentence level. This representation features many aspects of semantic such as name entity, co-reference, and semantic relations. AMR represents the meaning of a text in form of a labeled root directed acyclic graph. Many applications based on AMR have been introduced with great improvement thank the robustness of AMR. Therefore, improving the accuracy of AMR parser would be a great contribution to the success of those applications including text generation, text extraction.

Abstract Meaning Representation parsing is a challenging task due to the complicated combination of many subtasks and the structure of AMR graph. AMR covers a wide range of semantic representation from co-reference, named entity and semantic relations. Hence, facing many tough subtasks in a single combined problem is the most arduous difficulty in AMR parsing. Additionally, the requirement of the graph-structure output is the second problem which restricts the utilization of many advanced deep learning model. Moreover, the existing AMR corpora are considered as small however it contains so large vocabulary that make the corpora sparse. The sparsity of the dataset is an addressed tough issue for any learning system.

Prior successful works utilized transition-based system which converts the dependency tree into AMR graph, however, this method is approaching the limit because of the simplicity of the model. Recent studies in AMR parsing turn to deep learning which has proved its robustness in many tasks. This thesis presents our study on developing an AMR parser using deep learning. First, we introduced a framework for text-to-AMR (AMR parsing) and AMR-to-text (AMR generation). Second, we combined convolutional sequence to sequence with our proposed linearization algorithm for AMR parsing. Third, we published an AMR dataset which we expect to encourage studies in the legal domain.

Firstly, we generalized AMR parsing and AMR generation tasks into a framework which would be an essential tool for addressing problem and evaluation in AMR-related works. This framework contains two components: a graph conversion algorithm and a neural machine translation model. This framework separates graph conversion and translation model so that we can easily assess the contribution of these factors. Secondly, we introduced an AMR parsing model based on our proposed framework. We proposed a

graph linearization algorithm with the reverse path as the core of graph conversion. Three sequence to sequence models is investigated in this research. They are bidirectional long short-term memory(LSTM) encoder-decoder, fully convolutional model and a combination of convolutional encoder and LSTM decoder. Finally, we put our effort to build an AMR test set for legal documents which we retrieve from the English version of Japan Civil Code.

Our evaluations were conducted with standard SMATCH score and various task-specific metrics which are widely used in other works. Our first experiment of upper bound proved that reverse path linearization method efficiently rewrites a graph as a sequence. Hence, we can totally confident to approach parsing task with neural machine translation method. Our parsers achieved state-of-the-art performance on two golden standard datasets LDC2014T12 and LDC2017T10 in the second evaluation. Our additional analysis indicates that graph linearization method dominates the performance of the parser. The benchmark of throughput confirms that convolutional sequence to sequence model delivers extremely higher throughput than LSTM-based models. In the evaluation of task-specific semantic parsing, the models with reverse path linearization outperformed existing linearization method with parenthesis.

Keywords: abstract meaning representation, convolutional sequence to sequence, graph linearization.

Acknowledgements

Spending two years at Japan Advanced Institute of Science & Technology is the most fruitful decision in my life. My vision and knowledge have been expanded at highest ability thank the advanced educational system at JAIST. I have received brilliant instruction from the professors and enthusiastic support from the staffs. In addition, the facilities at JAIST are considered as the energetic muscle of the research life. All of those compounded into my achievement during my master program.

I would like to thank my main supervisor, Associate Professor Nguyen Le Minh, for his continuous instruction and supervision. He contributes the largest part in my extraordinary development including knowledge, research vision, motivation and academic skills. I cannot imagine the extent of struggle in research without his guidance.

I appreciate the comment and recommendation from Professor Satoshi Tojo, Associate Professor Kiyoaki Shirai and Associate Professor Shogo Okada. Their feedbacks have increased the importance of my research. I also benefit from the support from my minor research supervisor, Associate Professor Razvan Beuran. He has taught me many academic skills and I admire his clearness and kindness during our conversation. The lesson from professor Ho Tu Bao and Associate Professor Dam Hieu Chi have built up my standard of the characteristics of data science and data scientist.

My smooth research witnessed the support from all the members of Nguyen Laboratory. Their discussion and knowledge were so helpful in speeding up my research. I would like to thank all members of JAIST Hanoi Team and Nhau Team. Thanks to their sharing and encouragement, my research life was far from stress.

From the bottom of my heart, I give my greatest respect to my family and my dear, they are always at my back to support me regardless my decision.

Lai Dac Viet
February 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we provide an overview of the research problem, our motivation, our objective, and the contribution of this study.

1.1 Background

The Internet is an extraordinary media which helps people share enormous information in form of text such as newspaper, open knowledge base, blog, and report. Those electronic documents have been exponentially increasing day after day and becoming a resourceful knowledge database of almost public fields. However, integrating those resources into daily applications has yielded many problems because of insufficient modeling techniques. These difficulties relate to clear extraction of information from text, well-structuring information, and logical inference from that basis. Therefore, extracting and building high-level knowledge representation take important roles in mining advantageous values from text documents.

Text parsing is a research field in natural language processing which has been investigated for decades. Parsing problems can be categorized into two main classes: *structural parsing* and *semantic parsing*. Structural parsing includes phrase structure (Ramshaw and Marcus, 1999), syntactic tree (Munoz et al., 1999), and dependency tree (McDonald et al., 2005; Nivre, 2003). Studies in this subfield focused on converting a sentence-like sequence into the tree-based structure. Those tree structures have provided valuable features contributing to many solutions in NLP such as open information extraction (Schmitz et al., 2012) and sequence labeling (Jie et al., 2017). On the contrary, semantic parsing aims at extracting rules for a dedicated grammar (e.g. context-free grammar (CFG), the structured query language (SQL), and the combinatory categorial grammar (CCG)).

Abstract Meaning Representation (AMR) is a robust semantic representation which can express the abstract meaning at the sentence level. Each AMR graph fully generalizes the meaning of a sentence as a directed graph. Nevertheless, AMR features semantic relations, named entity, predicate-argument structure and so on. Therefore, AMR parsing can be considered as both structural parsing and semantic parsing. Although AMR

was designed for single sentence representation (Banarescu et al., 2013), it supports the representation of multiple sentences.

Comparing recent AMR parser, AMR parsing task has been approached in two main different ways depending on the consideration of the problems. *Structural-oriented* approaches prioritize building the shape of AMR graph. Hence, the main advantages of this method are its abilities to deal with a long sentence and to take advantage of existing structures such as dependency tree. *Semantic-oriented* approaches prefer the semantic aspect which is generated directly from the sentence before applying appropriate method to restructure it to AMR graph. As the result, this method is good at delivering word sense which is the content of every vertex while it suffers from poor graph construction.

According to the designation, AMR revealed many difficulties have been addressed in recent works including:

- **Word-sense disambiguation** which is a tough problem in natural language processing stands as a core problem of AMR parsing. In an AMR graph, the content of each vertex in AMR graph is constructed from a word or a number itself or by generalizing an English word into a sense regarding the list of word sense in the Propbank (Banarescu et al., 2013). Thus, solving AMR parsing is also solving word sense disambiguation problem.

- **Semantic graph construction** is the second challenge in AMR parsing. This representation was designed as an abstract representation that features many aspects of semantic such as name entity, co-reference, and semantic relations (Banarescu et al., 2013). Therefore, the concern of an AMR parser relates to those perspectives in a single task. Moreover, outputting a graph requires more complicated data structure and algorithm.

- **Data sparsity** is the common issue for every AMR parser. Machine learning system is arguably data-driven system. A system may behave nicer if we feed more data for the same set of learning parameters. Recent works in neural machine translation have benefited from the large-scale dataset which contains millions of samples. However, parsing AMR is harder than that because the vocabulary much larger whereas the data is much smaller (Gildea et al., 2017).

1.2 Motivation

Abstract meaning representation is a promising text representation because of following reasons. Firstly, the readability of AMR is clearly conceded (Banarescu et al., 2013; Bos, 2016) since the tree structure of AMR is a precise and straightforward way to show the relationship between concepts and events. Secondly, semantic representations are exceedingly chaotic because of enormous separated representations such as temporal entities, named entities, semantic relations, discourse connectives, co-reference and so on. Moreover, each representation has a particular meaning and evaluation method. That makes

the integration of those into application hard and complicated. However, AMR introduces a whole-sentence homogeneous semantic representation for semantic. As the result, AMR is expected to support many natural language processing tasks such as text generation (Banarescu et al., 2013).

Additionally, AMR provides a systematic foundation which are human-level gold-standard resources and evaluation methodology. Firstly, the directed graph of AMR enables the automatic evaluation of the computer-generated graph. A standard evaluation score (Cai and Knight, 2013) was introduced to calculate the precision, recall, and F-score of machine-generated output toward a gold-standard AMR graph. Secondly, the annotation of AMR graph is simpler and faster than the former formal representations, which encourages the production of large manually annotated corpus [1] consisting of nearly 40.000 samples. Due to those reasons, studying AMR is methodological and resourceful to obtain a great understanding of the natural language.

Even though AMR parsing has experienced increasing improvement in parsing accuracy, the achieved performance is far from the expected one. The recorded numbers are around 0.70 F1-score which is much lower than syntactic parsing performances, which is more than 0.90 F-score. Hence, there is a room for revolutionizing AMR parsing. Moreover, since AMR revealed many attractive features, there is an increasing trend in employing AMR in event extraction (Garg et al., 2016; Rao et al., 2017), text summarization (Dohare and Karnick, 2017; Liu et al., 2015) and text generation (Song et al., 2016; Takase et al., 2016). Hence, abstract meaning representation parsing has a tremendous impact on those applications. Therefore, in this research, we would like to develop an AMR parser that generates highly accurate outputs.

Currently, the most advanced AMR parsers were built based on two main approaches: *transition system* and *recurrent neural network*. Firstly, transition system (Wang et al., 2015b) once introduced has leveraged the accuracy to a much higher level that time. However, researchers who solve AMR problem using transition system are facing many difficulties: the feature mining and the simplicity of the model. Feature engineering is the conventional border which requires expertise. The simplicity of the model is unable to create remarkable improvement. Second, several researchers have adjusted the model of machine translation to the AMR parsing problem (Konstas et al., 2017; Takase et al., 2016). These studies indicate that RNN can deliver a good parser using existing neural machine translation model. However, this method comes with disadvantages of computational complexity and time consumption. Therefore, we would like to proposed convolutional neural network to create a faster and more precise parser.

Finally, despite the existing large AMR corpus, there is a lack of publicly accessible AMR corpus. The very high cost of the corpus, mentioned above, limits the popularity of AMR in solving natural language processing problems. Hence, this obstacle may discourage many types of research involving in AMR. In addition, the existing AMR corpus is in the domain of public news whereas there is no domain-specific AMR corpus. We target at producing a corpus for the legal document because of the difficulty in dealing

[1] https://catalog.ldc.upenn.edu/LDC2017T10

with legal documents. The extremely long and complicated legal documents cannot be treated as same as general documents. By building a new AMR dataset, we believe that the legal document processing community has a new tool for their research interest.

1.3 Objectives

In this study, our research mainly targets at building a *highly accuracy abstract meaning representation parser*. Our objectives are:

- We approach the AMR parsing problem by neural machine translation method. We implement three advanced attention-based sequence to sequence models with the cores are either recurrent neural network or convolutional neural network. We hope to leverage the performance of AMR parsing and text generation.
- We propose a graph linearization and want to investigate the contribution of graph linearization and advanced models to the problem by comparing this linearization method with an existing method.
- We put my effort into building an AMR corpus of Japan Civil Code for testing AMR parser in the legal domain. This is the first study representing the legal document as AMR graph.

Chapter 2

Literature review

In this chapter, we provide a background of the abstract meaning representation and several models in abstract meaning representation parsing. The characteristics, applications and potential difficulties of AMR are presented in section 2.1. Then, we introduce two main promising approaches to solve AMR parsing problem in section 2.2 and 2.3. Finally, in the section 2.4, we present successful works which have been proposed recently.

2.1 Abstract Meaning Representation

Abstract Meaning Representation is a semantic representation language (Banarescu et al., 2013) which is designed to be easily readable for ordinary people. AMR forms a rooted acyclic directed graph which is convenient for traveling in a computer program. Each AMR graph represents the content of a set of analogous sentences which share the basic meaning, however, differ from surface forms. Every vertex and edge of a graph are labeled according to the sense of the words in a sentence. AMR parsing is the task of converting a given sentence to a corresponding graph. At the time of designation, AMR is orientationally developed for English. However, AMR has been expanding to several other languages such as Chinese (Li et al., 2016), French, German, Spanish and Japanese (Vanderwende et al., 2015) and is expected to be developed to other languages in the future.

Three different ways are widely utilized to demonstrate AMR graphs. Figure **??** illustrates these three typical format of AMR. First, AMR can be formally written as the conjunction of logical triples. Secondly, the PENMAN notation is used in such occasions that are related to human reading and writing. Thirdly, computer programs commonly store AMRs as graph structure in memory. In an AMR graph, nodes are managed by unique identification called **variable** which refers to an semantic **concept** representing the content of a vertex. For shorten the expression of graph, the notation of slash defines a vertex: $g/girl$ means that g is an instance (vertex) of the abstract concept *girl*. The concept can be either an English word (e.g. *girl*), a PropBank frameset (e.g. *want-01*) or a special keyword. The keywords are varied including special entity type (e.g. *date-entity, organization, ordinal-entity*), quantities (e.g. *distance-quantity*), and logical conjunction(e.g. *and, or*). The label connecting two AMR nodes is made up by more

than 100 relation types including frameset arguments (e.g.*:arg0, :arg1*), general semantic relations (e.g. *:age, :condition, :domain*), relations for quantities (e.g. *e.g. :quant, :unit, :scale*), relations for date-entities, relations for listing (e.g. *:op1, :op2, :op3*).

AMR also provides the inverse form of all relations by concatenating *-of* to the original relation (e.g. *:location* vs *:location-of*). Hence, if R is a directed relation of x and y, we have: $R(x, y) \equiv R\text{-}of(y, x)$

Table 2.1: Conjunction format and PENMAN format of an AMR graph.

$\exists w, e, d, b:$ $\textbf{instance}(w, want\text{-}01) \wedge \textbf{instance}(e, eat\text{-}01)$ $\wedge\textbf{instance}(d, dog) \wedge \textbf{instance}(b, bone)$ $\wedge\textbf{arg0}(w, d) \wedge \textbf{arg1}(w, e)$ $\wedge\textbf{arg0}(e, d) \wedge \textbf{arg1}(e, b)$	(w/want-01 :ARG0 (d/dog) :ARG1 (e/eat-01 :ARG0 d :ARG1(b/bone)))

To measure the similarity of two semantic graphs, Cai et al (Cai and Knight, 2013) introduced the **SMATCH** score. This score measures the level of element overlapping between two structures. SMATCH score has been widely applied in measuring the accuracy of AMR parser. Table 2.2 shows the process to calculate the SMATCH score from two given graphs.

Table 2.2: Example of calculating SMATCH score.

Assume that a machine generates the AMR graph of the sentence *"The dog ate the bone"* and the gold standard AMR graph is the graph of the sentence *"The dog wants his bone"*.

Sentence	The dog ate the bone.		The dog wants his bone.	
PENMAN Format	(a/eat-01 :ARG0 (b/dog) :ARG1 (c/bone))		(x/want-01 :ARG0 (y/dog) :ARG1 (z/bone) :poss(y))	
Conjunction Format	$\exists a, b, c: instance(a, eat-01)$ $\wedge instance(b, dog)$ $\wedge instance(c, bone)$ $\wedge arg0(a, b) \wedge arg1(a, c)$		$\exists x, y, z: instance(x, want-01)$ $\wedge instance(y, dog)$ $\wedge instance(z, bone)$ $\wedge arg0(x, y) \wedge arg1(x, z)$ $\wedge poss(z, y)$	
Alignments	Matches	Precision	Recall	F-score
a=x, b=y, c=z	4	4/5	4/6	0.73
a=x, b=z, c=y	0	0/5	0/6	0.00
a=y, b=x, c=z	1	1/5	1/6	0.18
a=y, b=z, c=x	0	0/5	0/6	0.00
a=z, b=x, c=y	0	0/5	0/6	0.00
a=z, b=y, c=x	1	1/5	1/6	0.18
SMATCH Score				0.73

2.2 Transition-system-based method

A transition system is a collection of configurations and transitions that were widely used to describe a dynamic process, in which the configurations represent feasible states of the process and the transitions describe the way to shift from a state to another state. A transition system is ordinarily defined with a set of initial states and terminal states. Chess can be considered as a simple transition system where the positions of pieces on the chess board is a state of the table and the chess rules are the set of transitions.

The transition-based parser has made notable achievements in text parsing tasks such as dependency tree (Chen and Manning, 2014; McDonald et al., 2005; Nivre, 2003). Currently, the best AMR parser also uses transition system as the core of the parser (Wang et al., 2016, 2015a,b). Those works consider an existing structure of a sentence (e.g. dependency tree) as the initial state and train an transition system to restructure into AMR graph. The transition system (S, T, s_0, S_t) in these studies contains:

- $S(\sigma, \beta, G)$ is a set of parsing configurations (states). Each state contains two buffers σ, β which are used to determine the transformation actions applied to the graph G.
- T is the collection of parsing transitions (actions). Each action is a function that maps: $S_i \leftarrow S_j (i \neq j)$.
- $s_0(w, D)$ is a function that initializes the initial state from sentence w and its dependency tree D. This model employs a dependency tree parser to parse the given sentence into a dependency tree which is the input of the initialization function s_0.
- S_t is the set of terminal state, hence, S_t is a subset of S.

At the state $s \in S$, a scoring function $score(s, t)$ estimates the score of action $t \in T$ from a weight vector $\vec{w}$ and a feature vector generated from function $\Phi(s, t)$:

$$score(s, t) = \vec{w} \cdot \Phi(s, t) \tag{2.1}$$

The greatest advantage of this method is that regardless the level of accuracy of the *score* function, the output of the system is guaranteed to be an AMR graph. The main consideration while designing this transition system is the transition set and building the function $score(s, t)$. Designing the set of transition is the process of drawing the map where we travel from the dependency tree to AMR graph and constructing the *score* function is as important as the search algorithm in finding the best path to travel. In this research, we do not follow this method to build our parser. However, since this method currently initiated the best parser, we compare our method against this competitive method.

2.3 Neural-translation-based method

Another approach in AMR parsing is based on neural machine translation. In this method, the graph is rewritten in an *intermediate language* which is fit the neural machine translation model. Hence, the parsing process is separated into two stages: (1) translating the given sentence into an intermediate language and (2) converting the intermediate representation into PENMAN notation using a rule-based algorithm.

2.3.1 Theoretical framework

Recurrent neural network

Recurrent neural network (RNN) is a variant of the neural network. They are designed to tackle sequential data such as a sequence of text, numerical time-series data, genome, the stock market, and so on. It is based on an assumption that there is a strong relationship among values in a period in term of time point. RNN solves them by introducing a hidden state maintaining storing the observed information, then the information is delivered back to itself through recurrent connections as an input at the next time step. In an RNN, the smallest component calculating hidden state from the previous hidden state and current input is called the *RNN cell*.

Given a sequential input $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ of length n where $x_i \in \mathbb{R}^r$ is a r-dimensional representation vector, the sequence of the hidden states is recursively computed as follow:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b_h) \tag{2.2}$$

where: $W_h \in \mathbb{R}^{d \times d}$ and $W_x \in \mathbb{R}^{d \times r}$ are the learnable weight matrices of the network; $b \in \mathbb{R}^d$ are bias terms; h_t are the hidden states of the model at a time step t; σ is a non-linear activation function (e.g. *sigmoid* and *tanh* function). Applying this formula n times on the input X, we can derive the concatenation of n hidden states $H = \{h_1, \ldots, h_n\}$ with $h_i \in \mathbb{R}^d$. The final states h_n is considered to be the aggregated representation of the sequence because it is computed from every parts of the sequence. Figure 2.1 illustrates the way to unroll a RNN.

Long Short-Term Memory

Long Short-Term Memory (LSTM) is a successful extension of RNN. LSTM features three-gate mechanism which contains input gate, forget gate, and output gate. Gating mechanism is a kind of information filtering that keeps some parts of the information while reducing or even wiping out the others through element-wise multiplication of the input. Similar to the RNN, LSTM updates its hidden memory at time step t from the

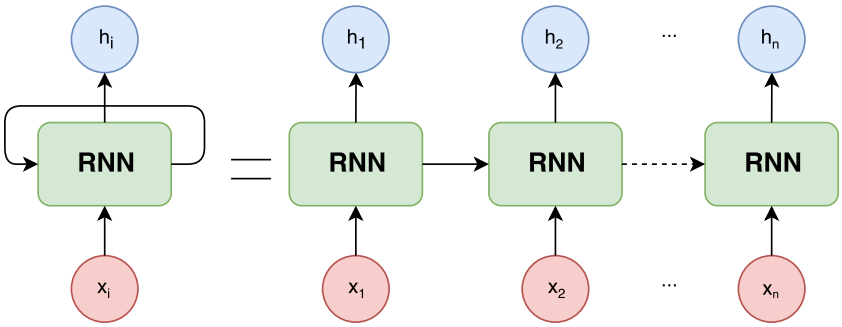

Figure 2.1: Computational process of recurrent neural network.
The RNN cell (the green box) computes the hidden state (the blue circle) from the previous hidden state and the input (the red circle).

input x_t, the previous hidden state h_{t-1}, and the previous memory state c_{t-1} as follow:

$$\begin{aligned} i_t &= \sigma(W_{hi}h_{t-1} + W_{xi}x_t + b_i) \\ f_t &= \sigma(W_{hf}h_{t-1} + W_{xf}x_t + b_f) \\ o_t &= \sigma(W_{ho}h_{t-1} + W_{xo}x_t + b_o) \\ \tilde{c}_t &= \tanh(W_{hc}h_{t-1} + W_{xc}x_t + b_c) \\ c_t &= f_t \otimes c_{t-1} + i_t \otimes \tilde{c}_t \\ h_t &= o_t \otimes \tanh(c_t) \end{aligned} \tag{2.3}$$

where $W_j \in \mathbb{R}^{d\times d}$ are the weight matrices of the network; $b_j \in \mathbb{R}^d$ are bias vectors; i, f, o indicate the input gate, forget gate, and output gate; $\otimes$ is the element-wise multiplication. Analogously, the component performing the computation in formula 2.3 is named the *LSTM cell*.

Thanks to the gating mechanism, LSTM also reduces the vanishing and exploding gradient issue of RNN (Hochreiter and Schmidhuber, 1997) because it provides a constant error back-propagation during training. Therefore, LSTM is better at modeling the long-term dependency between steps than that RNN does.

Bidirectional RNN & LSTM

Mentioned networks including RNN and LSTM model the dependencies in positive time direction which is forward states. However, in most of the natural language processing task, unidirectional modeling is not sufficient to design the relations. For examples, an English noun can be supported by both the front adjectives and the relative clause in the back. Hence, ones cannot catch the full idea of a word with only the former part of the sentence. This phenomenon is also very common in most of the languages and other tasks such as time series processing. Hence, it is better to make bidirectional constraints by using bidirectional RNN and LSTM.

Bidirectional RNN (or LSTM) consists of two RNN (or LSTM) network: *forward* network and *backward* network. The forward network processes from the head to the tail

of the input while the backward processes in the reverse direction. Then the hidden states and outputs of two networks are aggregated corresponding to the index of the input. The hidden state h_t at input x_t is calculated as follow:

$$\begin{aligned} \overrightarrow{h}_t &= ForwardCell(\overrightarrow{h}_{t-1}, x_t) \\ \overleftarrow{h}_t &= BackwardCell(\overleftarrow{h}_{t+1}, x_t) \\ h_t &= \left[\overrightarrow{h}_t, \overleftarrow{h}_t\right] \end{aligned} \tag{2.4}$$

where $ForwardCell, BackwardCell$ are the cells (RNN or LSTM) of the forward network and backward network; $\overrightarrow{h}_{t-1}$ is the previous hidden state of the forward network corresponding to the input x_{t-1} ; $\overleftarrow{h}_{t+1}$ is the previous hidden state of the previous state of the backward network corresponding to the input x_{t+1}.

The bidirectional neural network is widely used in a wide range of research including machine translation (Bahdanau et al., 2015), sequence labeling (Huang et al., 2015; Lai et al., 2017).

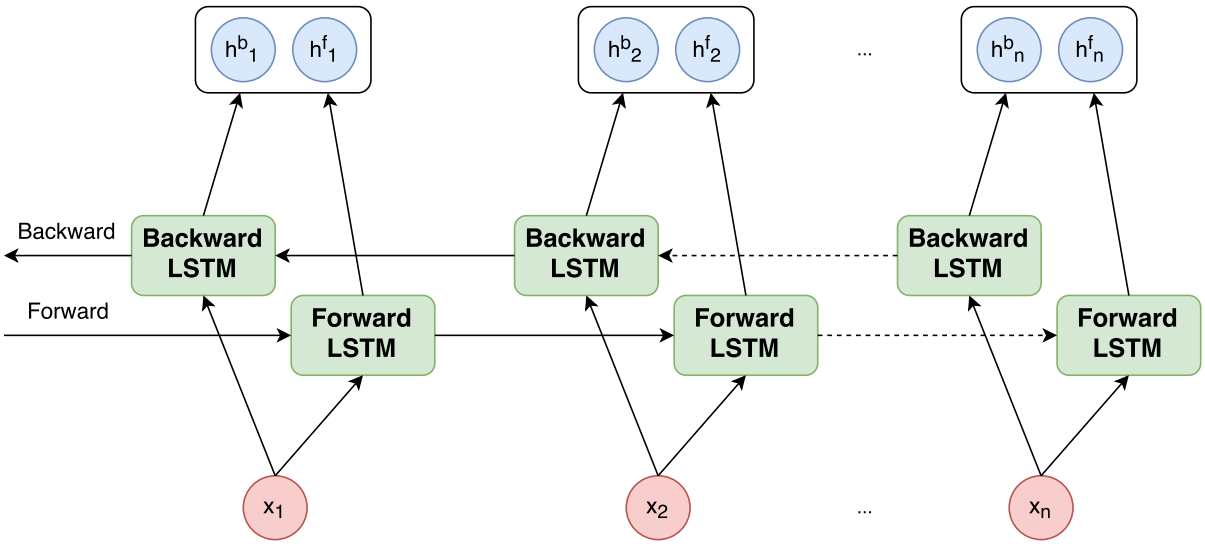

Figure 2.2: Computational process of bidirectional LSTM.
x_i are input vectors; h_i^b are outputs of the backward network;
h_i^f are outputs of the forward network

Convolutional neural network

Convolutional neural network (CNN) is another variant of neural network featuring the *convolution layer* which performs the convolution operation. Apart from convolution layer, fully convolutional network may contain *pooling layer*, and *fully connected layer*. A fully CNN is built by stacking these layers on top of the others before ending up by multiple fully connected layers.

In a CNN, **convolution layer** is the core component which dominates the behaviors and performance of the network. CNN extracts local features using a *filter* (or a *kernel*) which is a learnable matrix. The local features are extracted based on the local connection among regional features inside the kernel. The convolution layer performs convolution operation regarding the kernel to output an abstract feature. The kernel slides on the matrix of input features to map the input features into a feature map. The distance between two consecutive operations is called the *stride*. *Kernel size* is the size of the sliding window in which the operation is active. Conventionally, multiple kernels which share a kernel size are used to extract multiple patterns from the input. Various kernel sizes are put into a CNN to capture features in both short dependency and long dependency.

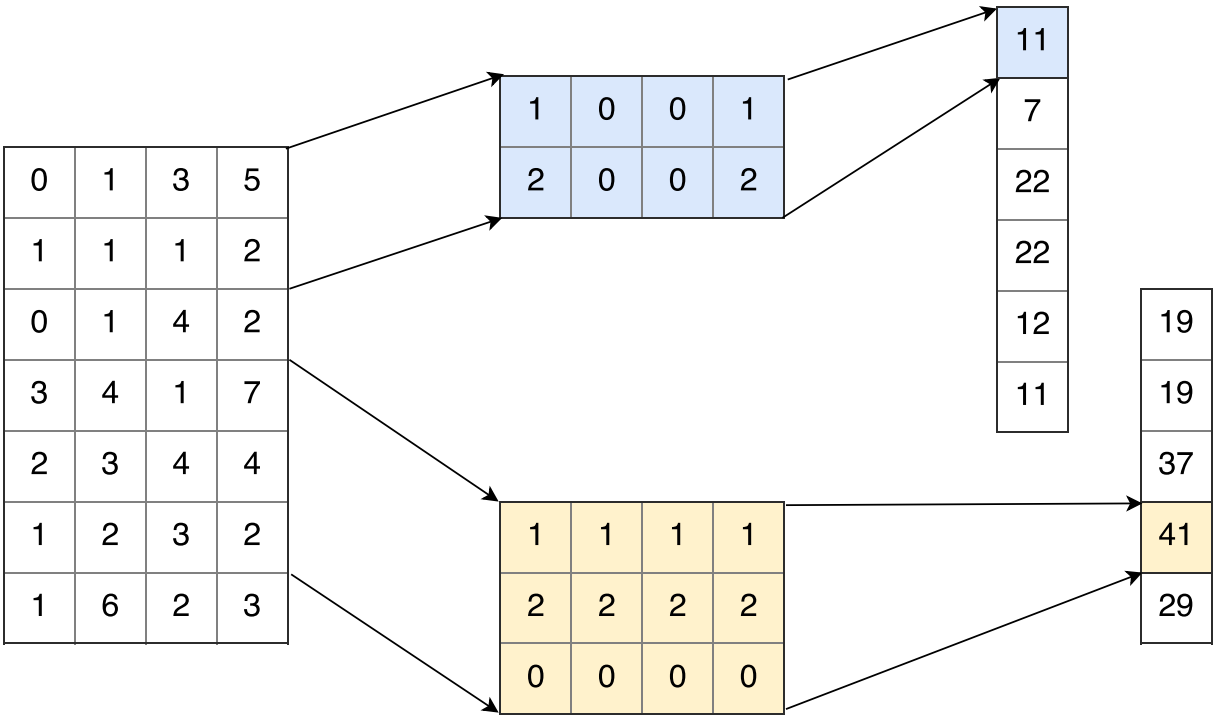

Figure 2.3: An example of convolution layer without padding.
The 7×4 input feature matrix (left side) represents a sentence of of 7 words into 4-*dimensional* space, two kernels whose *kernel_sizes* are 2 and 3 (middle), and two feature maps (right side).

Originally, CNN was invented for image processing task (e.g. image classification, image segmentation, etc) in which the kernel is a small square matrix. It slides vertically and horizontally along the height and the width of the image. However, many studies have modified CNN for natural language processing. The width of the kernel is fixed as same as the number of dimensions of input feature space which is much bigger than those in image processing. The kernel slides vertically only to perform 1D convolution operator. In other words, the kernel slides along the sentence length hence, the kernel size is also considered as n-gram of the model. The values of kernel size, the dimension of input space, the number of kernels are varied in different tasks, computational capacity, and

n-gram models. The kernel size is commonly chosen in the range from 2 to 5 according to the conventional n-gram models. Figure 2.3 illustrates the behaviors of the convolution layer.

In a CNN, **pooling layer** frequently locates between two convolution layers to down-sample the spatial size of the representational features. As the results, it reduces the number of parameters in higher CNN layer, hence, consequently reduces the computation load and control the overfitting of the network. The pooling layer combines values of a feature map at a layer into a value of the next layer. The pooling layer is not utilized on the output of different feature maps. There are several types of pooling layer. Each concrete pooling layer features an operation (e.g. *max pooling* performs MAX operation, *average pooling* calculate the average value of the output of the prior layer). Figure 2.4 presents the down-sampling of max pooling layer.

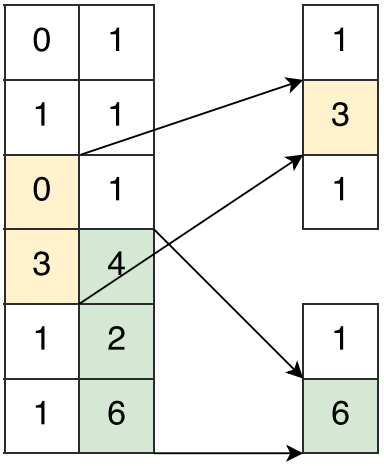

Figure 2.4: An example of max pooling layer.
Two max pooling layers performs on two distinguishing feature maps (two column matrix on the left); the sizes of the filter of the layer are 2×1 (top, yellow) and 3×1 (bottom, green).

Fully-connected layer locates at the top of the CNN architecture. This layer fully pairwise connects neurons of two adjacent layers whereas neurons in the same layer are not connected. This layer works as the output layer of the CNN. The simplest CNN compound of a convolution layer, a max pooling layer, and a fully-connected layer is illustrated in figure 2.5.

Gating mechanism

Gating mechanism is a dominant architecture in long short-term memory in which it controls the flow of information. Thank the management of the gates, LSTM reduces vanishing and exploding gradient and enables long-term dependencies (Hochreiter and Schmidhuber, 1997). Moreover, gating mechanism has been upgraded to a higher level to be applied in a non-recurrent neural network such as convolutional neural network (Gehring et al., 2017b). The main advantage of gating mechanism is its ability to learn

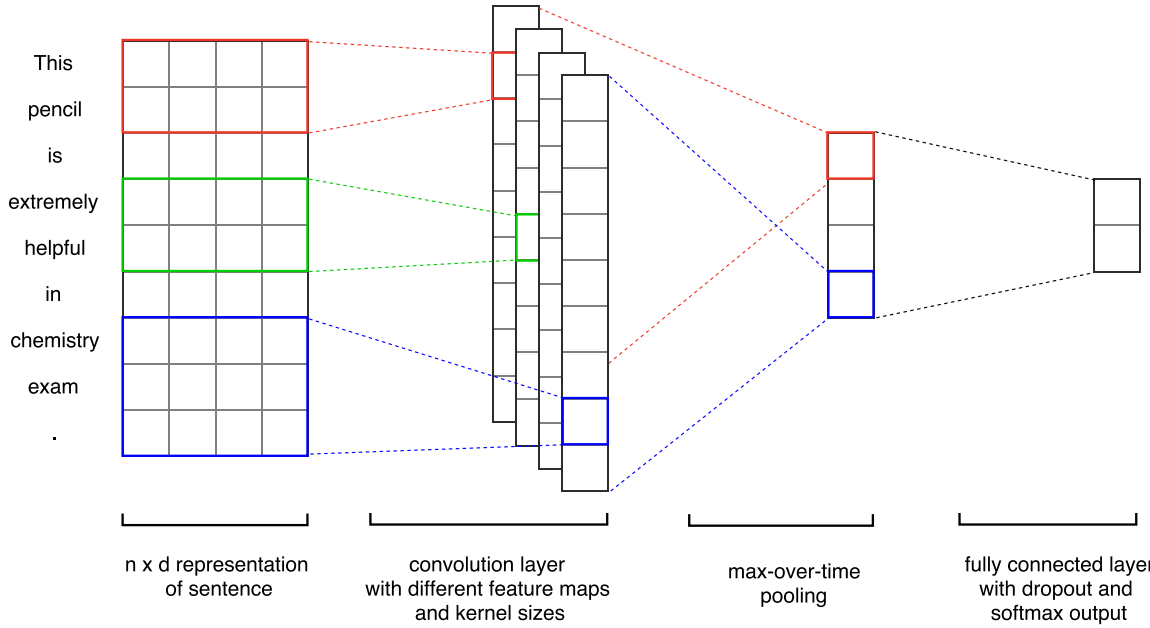

Figure 2.5: The example of CNN using in sentence classification.

and to distill unrelated information while enhancing essential information. Intuitively, to refine information of an input $X^{r\times d}$, a gate uses a filter $G^{r\times d}$ and performs matrix element-wise multiplication operation as follow:

$$\bar{X} = \phi(X) \otimes \psi(G)$$

where ϕ and ψ are transformation functions including linear and non-linear functions (e.g. *tanh* and *sigmoid*). To recent years, several variants have been introduced and achieved state-of-the-art performance such as *rectified linear unit* (ReLU), *gated tanh unit*(van den Oord et al., 2016), *gated linear unit* (GLU) (Dauphin et al., 2017; Gehring et al., 2017b). The calculations of ReLU, GTU, GLU are presented in formula 2.6, 2.7, 2.8, respectively.

Rectified linear unit (in formula 2.5) is partially considered as a simplification of a general ReLU unit in formula 2.6. ReLU maintains a linear path but non-linear path for back-propagation. Therefore, the gradient of the network containing ReLU does not vanish while passing layer to layer. However, the inactiveness to the negative input and the trainability limit the adaptiveness of ReLU in filtering information.

$$ReLU(x) = max(0, x) \tag{2.5}$$

$$ReLU(X) = X \otimes (X > 0) \tag{2.6}$$

Gated tanh unit is an LSTM-like gating mechanism which features two non-linear paths. Since it does not contain any linear path, model which stacks multiple GTU units may easily wipe out gradient during training with the back-propagation algorithm.

$$GTU(X) = tanh(X * W + b) \otimes \sigma(X * V + c) \tag{2.7}$$

Gated linear unit has the profitable properties of both ReLU and GTU. The unit features both a linear path and a non-linear path for the flow of gradient. Therefore, it is unnecessary to scale the gradient (Dauphin et al., 2017). GLU unit contains two sets of filters $W = (w_1, w_2, \cdots, w_k)$ and $V = (v_1, v_2, \cdots, v_k)$ where $w_i \in \mathbb{R}^{m_i \times d}$ and $v_j \in \mathbb{R}^{n_j \times d}$. Given a sequence of input $X = (x_1, x_2, \cdots, x_n)$ where $x_i \in \mathbb{R}^d$, the GLU unit compute the sequence of hidden states $H = (h_0, \cdots, h_n)$ as shown in formula 2.8.

$$GLU(X) = (X * W + b) \otimes \sigma(X * V + c) \tag{2.8}$$

Highway connection

Highway connection (or residual connection, shortcut connection) is connection that skips one or more layers. The connection makes a highway flow X which is an identity mapping of the input X. The output of the connection is added to the output $F(X)$ of the skipped layers to make the output $H(X)$ of the residual block.

$$H(X) = F(X) + X \tag{2.9}$$

Highway connection is an extremely simple technique which is easily implemented in one line of code on common deep learning frameworks (e.g. Tensorflow, Caffe). Moreover, identity highway connection keeps the simplicity of the model because neither additional parameters nor computational load is added. Enhanced convolutional neural network and recurrent neural network with highway connection have delivered higher performance in recent works (He et al., 2016; Zilly et al., 2017). Figure 2.6 presents the illustration of a residual block.

2.3.2 Sequence-to-sequence model

Sequence-to-sequence (seq2seq) is a learning model that converts a sequence of data from one domain into another domain. For example, a seq2seq model can be trained to perform translation of a sequence of text from one language to another language (Bahdanau et al., 2015; Luong et al., 2015; Sutskever et al., 2014). Seq2seq have been applied successfully in many application such as machine translation (Sutskever et al., 2014), text generation (Konstas et al., 2017), text summarization (Lai et al., 2017), and structural parsing (Konstas et al., 2017; Takase et al., 2016).

Given an input sequence $(x_1, x_2, \ldots, x_n)$, the goal of a seq2seq model is to estimate the conditional probability $P(y_1, y_2, \ldots, y_m | x_1, x_2, \ldots, x_n)$ where $(y_1, y_2, \ldots, y_m)$ are the corresponding output whose length m may differ from the length of the input n (Sutskever et al., 2014). The seq2seq model first embeds the input sequence $(x_1, x_2, \ldots, x_n)$ into an vector v which is the last hidden states of the model. Then it computes the conditional probability of the output sequence follow the decomposition formula of conventional language model:

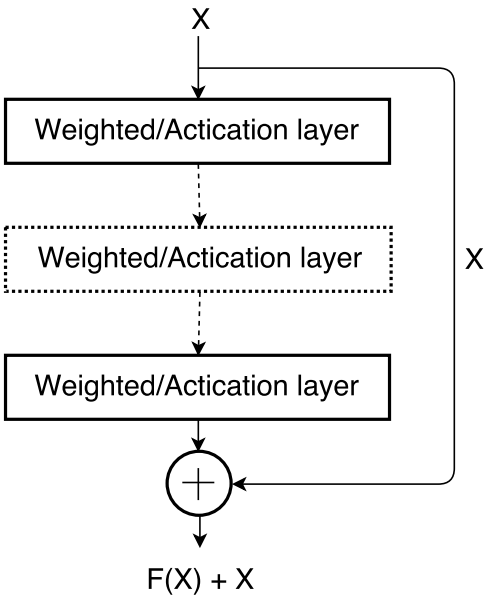

Figure 2.6: Illustration of highway connection.

$$P(y_1, \ldots, y_m | x_1, x_2, \ldots, x_n) = \prod_{t=1}^{m} P(y_t | v, y_1, \ldots, y_{t-1}) \tag{2.10}$$

Encoder - Decoder

Sequence-to-sequence model commonly consists of two main components which are the **encoder** and the **decoder** regarding two main phases in seq2seq model:

- In the **encoding** phase, the encoder reads the input sequence then transforms and stores the information in its internal states which is an intermediate representation. This form can be either a vector or a sequence of vectors which compress the fed information.
- During the **decoding** phase, the decoder recursively predicts the next item (e.g. a character, a word, and so on) of the target sequence from a special initial item (e.g. $<GO>$ character). The internal states of the encoder are shipped to the decoder in order to transfer understandable knowledge from the encoder to the decoder. To provide a sense of context, the decoder also uses its generated sequence as its own input. Thank two informative sources of information, the decoder can both sufficiently deliver information of the source input and generate expectedly fluent output.

The most common seq2seq models utilize RNN and LSTM as the core of the encoder and decoder. Naturally, RNN and LSTM reveal a sequential input and output which are

extremely convenient for the sequence to sequence. Figure 2.7 illustrates the overall idea of sequence to sequence model.

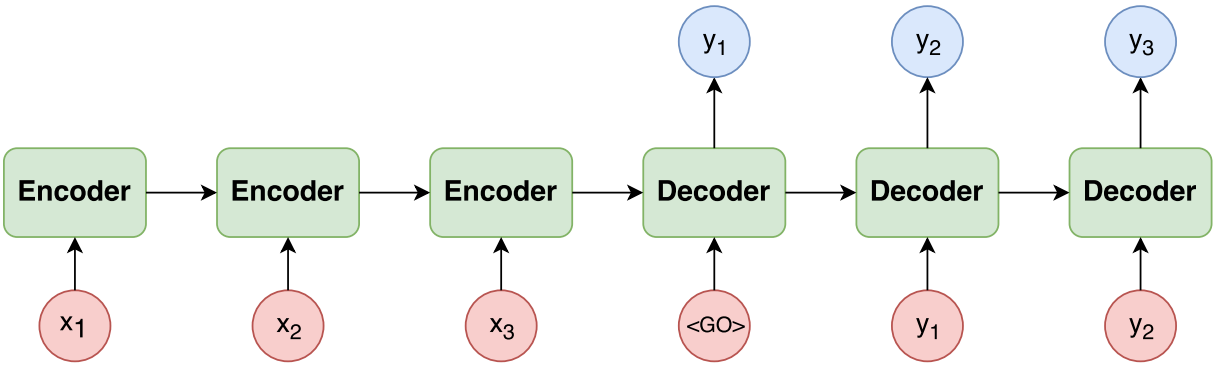

Figure 2.7: Computing flow of a sequence to sequence model

In the vanilla encoder-decoder model, the decoder is given the last hidden states of the encoder as the representation of the whole sequence. This mechanism reveals two main obstacles due to its static context encoding mechanism. Firstly, the provided information might be partially lost due to the limited ability of encoding of LSTM and RNN cell. Secondly, at a certain decoding step, redundant information existing in the encoded information may interfere the behavior of the decoder.

Attention mechanism

Attention mechanism is one of the breakthrough advancements in deep learning. This mechanism is analogous to the human visual attention. Given a picture, a person focuses on a sub-region of the picture which is most attractive regarding his/her attentive thought. The sequence-to-sequence models have delivered state-of-the-art performance such as those in machine translation (Luong et al., 2015) thank the attention mechanism.

In the vanilla sequence-to-sequence model, the encoder encodes full sentence into a single fixed-dimension vector before delivering to the decoder. This method suffers from the low accuracy of the very long sentence even LSTM seems to be better at building such constraints. In contrast, the decoder in the attention-based seq2seq model is provided full access to the hidden states at every time steps. The model is trained to attend to several particular hidden states based on the input and its previous generated outputs. Therefore, the output of a certain time step is generated as similar as those created by human. For example, a generated sentence should be as grammatical as possible in machine translation and text generation.

Figure 2.8 illustrates a sequence-to-sequence model with Bi-LSTM encoder and attention-based decoder. Given an input sequence $X = (x_1, x_2, \cdots, x_n)$, the encoder generates its hidden states $H = (h_1, h_2, \cdots, h_n)$ where h_i is the concatenation of forward and backward hidden states $[h_i^f, h_i^b]$). The decoder compute the alignment probability a_{ti} among its last hidden states h_{t-1}^d and hidden states h_i. Then it computes the context vector at the time step t using weighted sum as follow:

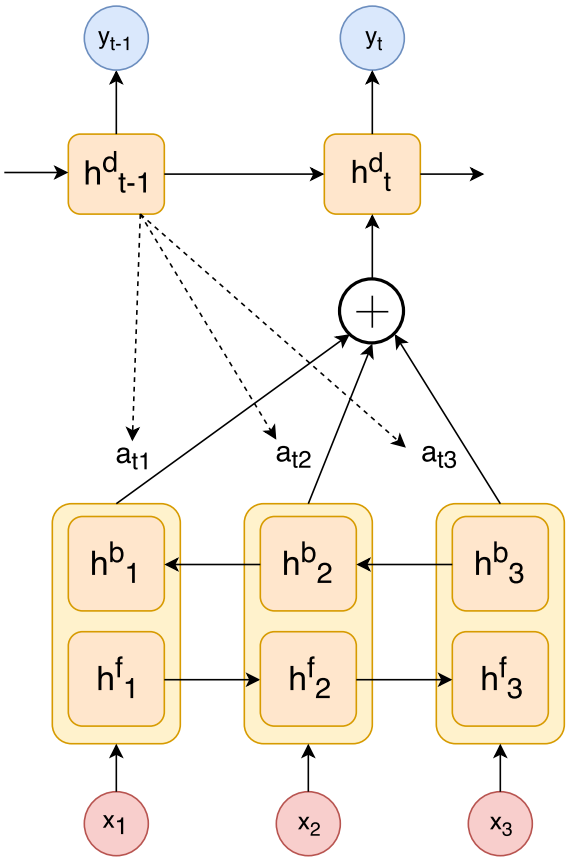

Figure 2.8: Illustration of attention mechanism with Bi-LSTM encoder.

$$
\begin{aligned}
s_{ti} &= tanh(W_d h^d_{t-1} + W_e h_i) \\
a_{ti} &= \frac{\exp(s_{ti})}{\sum_{j=1}^{n} \exp(s_{tj})} \\
c_t &= \sum_{i=1}^{n} a_{ti} h_i
\end{aligned}
\tag{2.11}
$$

The RNN/LSTM of the decoder generates its hidden state by combining its previous hidden state h^d_{t-1}, the computed context vector c_t, and input x'_t:

$$h^d_t = LSTM(h^d_{t-1}, c_t, x'_t)$$

2.3.3 Graph linearization

Sequence-to-sequence model requires sequential representation of features and labels, therefore, the AMR graph must be presented as a sequence. However, the raw AMR text cannot be an appropriate format due to its imbalance of tokens. Raw AMR text

contains too many round brackets and variables which present less semantic information than other components such as concepts and arguments.

2.4 Prior works on abstract meaning representation

The first learning parser and aligner for AMR (namely **JAMR**) divide the problem into two stages: concept recognition and relation recognition (Flanigan et al., 2014). The parser utilizes a sequence labeling algorithm to identify the concept which is defined as a substring of a word in the sentence. Then, a subgraph maximum-score algorithm finds an optimal subgraph whose vertices have been collected in the first stage.

Motivated by the similarity between dependency tree and AMR graph, Wang et al. (2015b) proposed the first transition system for parsing AMR graph. Figure 2.9 illustrates the dependency tree and the AMR graph of the sentence: *"The domicile of a juridical person shall be at the location of its principal office"*. As can be seen, these two structures share some nodes, (e.g. *domicile, person, juridical*) and their node interrelations (e.g. *person - juridical*). They defined a two-stage process for their system: (1) parsing a sentence into a dependency tree using existing parsers such as Stanford parser and Charniak parser; (2) converting the obtained tree to AMR graph by an eight-action transition system. Their later works have investigated a richer feature set including co-reference, semantic role labeling, word cluster (Wang et al., 2015a); rich name entity tag, and ISI verbalization list(Wang et al., 2016).

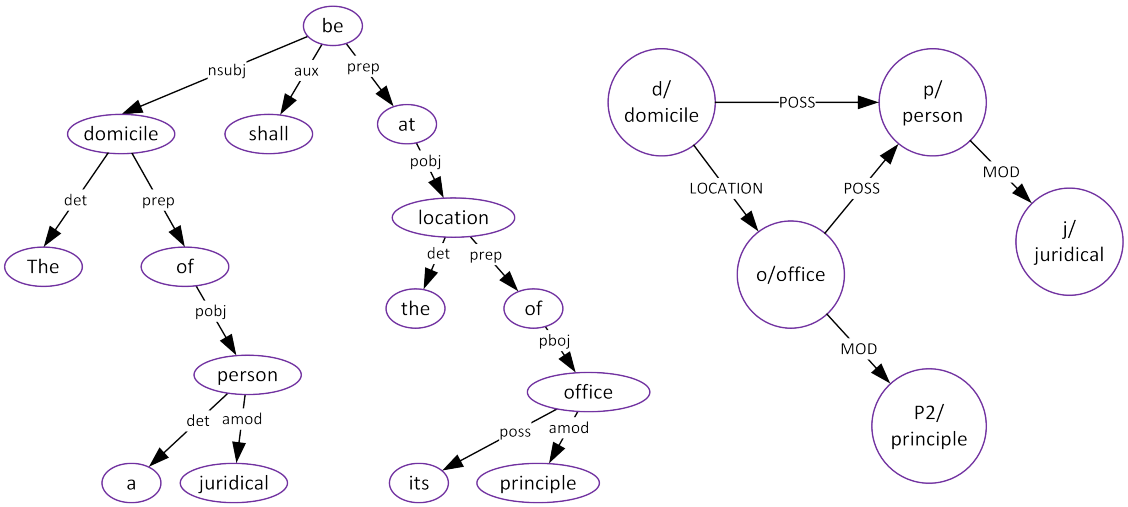

Figure 2.9: Dependency tree and AMR graph

Konstas et al. (2017) presented a joint system of parsing and generation with bootstrapping training strategy. Barzdins and Gosko (2016) showed an efficient adaptation of machine translation to AMR parsing. Their work indicates that character-based features are better than word-based features in this task. Targeting at the sparsity of the AMR graph data, Gildea et al. (2017) limited the vocabulary to 2000 and applied LSTM with

attention mechanism. These aforementioned approaches investigated only recurrent neural network and its variants. The work of Ballesteros and Al-Onaizan (2017) has combined recurrent neural network and transition system into a deep transition model. Instead of searching for a large number of features as done by the conventional transition method, they encoded the information of every state in LSTM hidden state. The input of the model consists of embedding vector and syntactic features.

Although recent studies have utilized LSTM in AMR parsing (Ballesteros and Al-Onaizan, 2017; Konstas et al., 2017), there are several disadvantages of employing LSTM:

- LSTM models long dependency which might be noise to generate a linearized graph whereas CNN provides a shorter dependency which is fit to graph traversal path.
- LSTM requires a chronologically computing process that reduces the parallelization; on the contrary, CNN enables to run model simultaneously.

Chapter 3

AMR parsing and generation model

In this chapter, we first present a universal framework for translation-based AMR parsing in section 3.1. Then, we show the process of graph conversion including graph simplification and graph linearization which are essential to enable translation-based model in section 3.2. In section 3.3, we present the employed attention-based sequence-to-sequence model in this research.

3.1 Framework

We introduce a universal framework of structural parsing using neural machine-translation method. The framework is presented in figure 3.1. First, the sequence-to-sequence model obligates the input and the output of the model to be shaped in a linear sequence of elements.

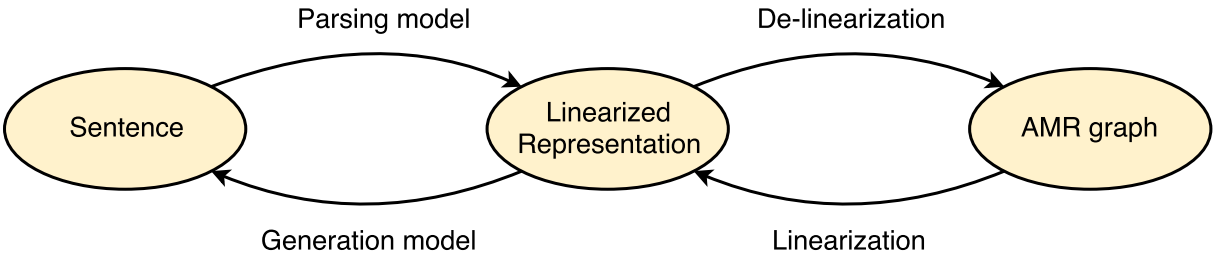

Figure 3.1: An universal framework for AMR parsing and generation task

In the AMR parsing, the parsing process follows the upper left-to-right flow. Given a sentence, a text-to-graph model translates a raw sentence into an intermediate linearized representation. A de-linearization algorithm structures the text into a graph. This recovery process considers tokens as a concept of AMR graph and reconnect.

On the contrary, the generation process starts with an AMR graph. This graph is converted into the intermediate linearized representation which is a sequence of text. Then a graph-to-text model generates a concrete sentence from the linearized representation.

3.2 Graph conversion

3.2.1 Graph simplification

In this research, we apply a series of graph simplification in order to reduce the sparsity of the AMR graph. They consist of variable removal, anonymization of named entities such as date, the name of person, organization, and country.

Removing variables

AMR graph contains a huge number of less informative information such as the variable of a node and the *instance-of* relation which is written by a slash character("/"). We remove all of these from the PENMAN representation of the AMR graph. In the case of reentrancy, the variable referring to an existing concept is replaced by its concept.

There are advantages and disadvantages of cutting out variables from the graph. Firstly, eliminating variable shorten the sequential representation of the AMR graph. Suppose an AMR node is written by 5 tokens including open and close parentheses, a variable, a slash relation and a concept, we can reduce 40% of the tokens representing a node. Second, variable elimination also reduces noise from the graph. A variable can be a representative of thousands of distinguishable words hence the inconsistency of the variable might misleads learning models and makes them hard to converge.

The removal of variable comes at a cost of information loss. The variable helps to distinguish different nodes which share the same concept. Once the variable is wiped out, nodes wrapping the same concept are collapsed into a single node. We present an experimental estimation of information loss in the section 4.4.1.

Anonymizing names

Names such as human name, organization name, the country name frequently appear in AMR, however, each name appears at a low-frequency (Konstas et al., 2017). We apply an extensive replacement of private names by name of the category to reduces the sparsity of the data. Additionally, collapsing a subgraph into an indexed node decreases the length of the text representation of AMR graph. However, this method also reveals an issue of understanding indexed nodes. The index of the collapsed node is meaningless to the model and even makes noise to the model.

We replace a sub-graph whose roots are in fine-grained entity types by a special node. The concept of this node contains the fine-grained type and an index i indicating the *i-th* appearance of that type in the graph. Table 3.1 shows the replacement of less frequent subgraph.

Anonymizing dates

We replace date-related nodes by anonymized tokens for day-number, day-name, month-number, month-name, and year. Similar to the name entities, this token contains the

Table 3.1: Anonymizing name entities in AMR graph.

	Sentence	**Graph**
Original	Mollie Brown, who slays orcs	**(p / person** :wiki "Margaret_Brown" **:name (n / name** **:op1 "Mollie"** **:op2 "Brown")** :ARG0-of (s / slay-01 :ARG1 (o / orc)))
Anonymized	**person_0**, who slays orcs	(p / **person_0** :wiki "Margaret_Brown" :ARG0-of (s / slay-01 :ARG1 (o / orc)))

categorical token and an index i indicating the i-th ordinal occurrence of that type in the graph.

Although substitution of the date-related nodes does not change the length of AMR text, it reduces the vocabulary in the AMR graph. Hence, the substitution may encourage the convergence of translation model. An example of replacement of date-related nodes is presented in table 3.2.

Table 3.2: Anonymizing date entities in AMR graph.

	Sentence	**Graph**
Original	Wednesday, February 29, 16:30 PST	(d / date-entity **:month 2** **:day 29** :weekday (w / wednesday) :time 16:30 :timezone (z / PST))
Anonymized	Wednesday, **month_0 day_0**, 16:30 PST	(d / date-entity **:month month_0** **:day day_0** :weekday (w / wednesday) :time 16:30 :timezone (z / PST))

3.2.2 Linearization & De-linearization

Linearization algorithm contains two main parts: (1) a graph traversal order which defines the order of appearance of a node in the intermediate sequence (2) a rendering function

which generates the sequence that satisfies the recoverability criteria. The criteria ensure that we can rebuild the graph from the generated text.

In this research, we employ the depth-first-search traversal with pre-order because this traversal with scope markers ensure the recoverability of the graph and the generated text is analogous to the original graph.

Linearization by parentheses

AMR graph written in PENMAN notation uses parentheses as the scope markers. The first linearization method which we employ also use brackets because of the nature of bracket in AMR format. However, we eliminate every redundant bracket (both the left bracket "(" and the right bracket ")") which wrap the node with only one child, therefore, we can lessen the number of generated tokens.

Linearization by reverse path

We propose a linearization sharing the same traversal strategy but marking scope differently. Instead of using two parentheses for marking a subgraph, we use the reverse path to structure the graph. The detail of the traversal algorithm is presented in figure 1. Table 3.3 shows an example of two linearization methods. The bold tokens present the reverse path.

Function `dfs(`T, $node_t$, L`)`**:**
  $append(L, node_t)$
  for $node_j$ *in* $child(T, node_t)$ **do**
    $append(L, relation(node_t, node_j)$
    if $is\_leaf(node_j)$ **then**
      $append(L, node_j)$
      $append(L, node_j)$ `// Reverse path`
    else
      $dfs(T, node_j, S)$
    end
  end
  $append(L, node_t)$ `// Reverse path`

Function `linearize(`G`)`**:**
  $T := preprocess(G)$
  Initialize list L
  $dfs(T, root(T), L)$
  return S

Algorithm 1: Linearization with reverse path.

Given a graph G, we pre-process it as presented in section 3.2. After that, the graph G is converted to tree-format T. Then we follow depth-first search traversal on the tree. Once we exit a node, we add the concept of the node to signal that is the end of the traversal at that node. The added concept is the reverse path to travel back to the root node when all nodes are visited.

Table 3.3: Graph conversion process in detail.

Original sentence	No abuse of rights is permitted.
AMR in PENMAN notation	(p / permit-01 :polarity - :ARG1 (a / abuse-01 :ARG1 (r / right-05)))
Removing variable	(permit-01 :polarity - :ARG1 (abuse-01 :ARG1 (right-05)))
Linearization with parentheses	permit-01 (:polarity -) :ARG1 abuse-01 :ARG1 right-05
Linearization with reverse path	permit-01 :polarity - **-** :ARG1 abuse-01 :ARG1 right-05 **right-05 abuse-01 permit-01**

3.3 Convolutional attention-based sequence-to-sequence model

We implement the fully convolutional sequence-to-sequence model proposed by Gehring et al. (2017b) which achieved the state-of-art performance in machine translation. We present three significant components of this model: the convolutional block, the attention encoder, and multi-step attention decoder.

There are three reasons why we choose convolutional sequence-to-sequence model for AMR parsing:

- AMR graph written in PENMAN notation is an abstract representation which contains generalized tokens and relations. Most of the relations between entities in AMR graph are spanned in short paths on the graph therefore, a model featuring local features are most suitable for AMR parsing. The convolutional neural network is well-known in representing local feature with short-term dependency. Therefore, we expected that convolutional sequence-to-sequence model would leverage the accuracy of AMR parsing.

- Based on the mechanism of RNN and CNN, we hypothesize that CNN-based models can deliver higher throughput than LSTM-based models. Recently, deep learning models have taken benefits from graphics processing unit (GPU) card which features CUDA technology with thousands of processing unit. Hence, parallelization is the most important factor to increase the usage of thousand cores. The obstacle of RNN is its sequential computation from one step to another, as the result, it can not benefit from the increasing number of cores in GPU. In contrast, the slicing of RNN is parallelized entirely because the independent of the computation of separate slices. In other words, the number of iteration of RNN is linear with the length of the sequence but those of CNN.

- Although convolutional sequence-to-sequence models have just been proposed re-

cently by Gehring et al. (2017a) and Gehring et al. (2017b), they achieved the highest performance on a large-scale benchmark dataset. This proved a promising chance to improve AMR parsing.

3.3.1 Word embedding

We equip two type of distributed embedding to the input of the network including conventional word embedding and position embedding. The word embedding layer projects the input sequence $\mathbf{X} = (x_1, \ldots, x_n)$ into distributed space as $\mathbf{W} = (w_1, \ldots, w_n)$, where $w_i \in \mathbb{R}^r$. This layer maintains an embedding matrix $D \in \mathbb{R}^{V \times r}$ where V is the size of the vocabulary of the language. In order to equip the sense of the position in the sentence to the model, we add an external position embedding layer which maps the input sequence $\mathbf{X}$ into $\mathbf{P} = (p_1, \ldots, p_n)$ where $p_j \in \mathbb{R}^r$. The final representation of the input is the combination of word embedding and the position embedding $\mathbf{S}(s_1 = w_1 + p_1, \ldots, s_n = w_n + p_n)$.

3.3.2 Convolutional block

The architecture of this convolutional block is integrated into both the encoder and the decoder of the model. Each block contains a non-linearity layer on top of a 1-D convolutional layer. The non-linear function chosen is gated linear unit (Dauphin et al., 2017) because of its diversity of linearity and non-linearity paths. We add a residual network connecting the input of the convolutional layer to the output of non-linearity layer.

At each step, the convolutional layer slides on k input tokens which are projected to r-dimensional space. In other word, the input at step t is $\mathbf{X}[j - k + 1 : j] \in \mathbb{R}^{k \times d}$ (the value of j depends on whether it is part of the encoder or decoder). Each convolution layer contains a kernel $W \in \mathbb{R}^{2d \times kd}$ and bias term $b_w \in \mathbb{R}^{2d}$. The convolutional operation projects input $\mathbf{X}$ into an output $\mathbf{Y} = [AB] \in \mathbb{R}^{2d}$. The non-linearity layer glu maps the output $\mathbf{Y}$ back to $\mathbb{R}^d$:

$$glu([AB]) = A \otimes \sigma(B)$$

Stacking multiple blocks

We stack several blocks to equip model the sense of hierarchical understanding, hence, we expect the model can translate in both implicit and explicit ways. The number of parameters in the model can be controlled in several ways. We can adjust the number of stacked blocks and the number of kernels inside each convolutional block.

$\mathbf{E}^l(e_1^l, e_2^l, \ldots, e_n^l)$ denotes the output of the block l-th in the encoder. The output of the block $l + 1$-th at time step t is calculated as:

$$e_t^{l+1} = glu\Big(W_{encoder}^l [e_{t-\frac{k}{2}}^l, \ldots, e_{t+\frac{k}{2}}^l]\Big) + e_t^l \tag{3.1}$$

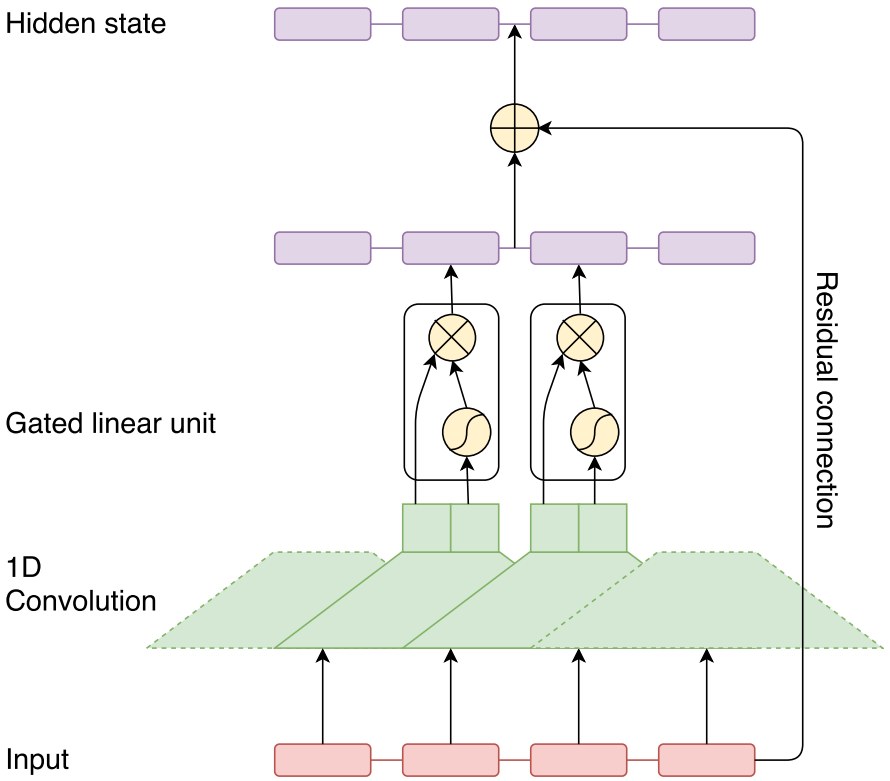

Figure 3.2: Architecture of the convolutional block.

However, during decoding process, the future information must not be fed to the decoder, so the kernel covers $k-1$ steps back to the time step t. The output of block $l+1$-th at time step t of the decoder is calculated from the output of block l-th $\mathbf{D}^l(d_1^l, d_2^l, \ldots, d_n^l)$:

$$d_t^{l+1} = glu\Big(W_{decoder}^l[d_{t-k+1}^l, \ldots, d_t^l]\Big) + d_t^l \tag{3.2}$$

3.3.3 Convolutional encoder

The encoder contains three main parts, an embedding layer which is presented in section 3.3.1 and a stack of convolutional blocks presented in section 3.3.2. We add dropout layer between convolutional blocks to trim the overfitting issue. We also project the output of the last block back to the embedding space. Then we add with the word embedding through a residual connection to produce the output of the encoder. Figure 3.3 presents the architecture of the encoder.

During decoding process which will be presented in detail in section 3.3.4, we use the output of the last convolutional block e^l for computing the proportional distribution of the attention mechanism whereas the output of the encoder o^e is used as the embedding source of weighted sum.

$$o_t^e = We_t^l + s_t \tag{3.3}$$

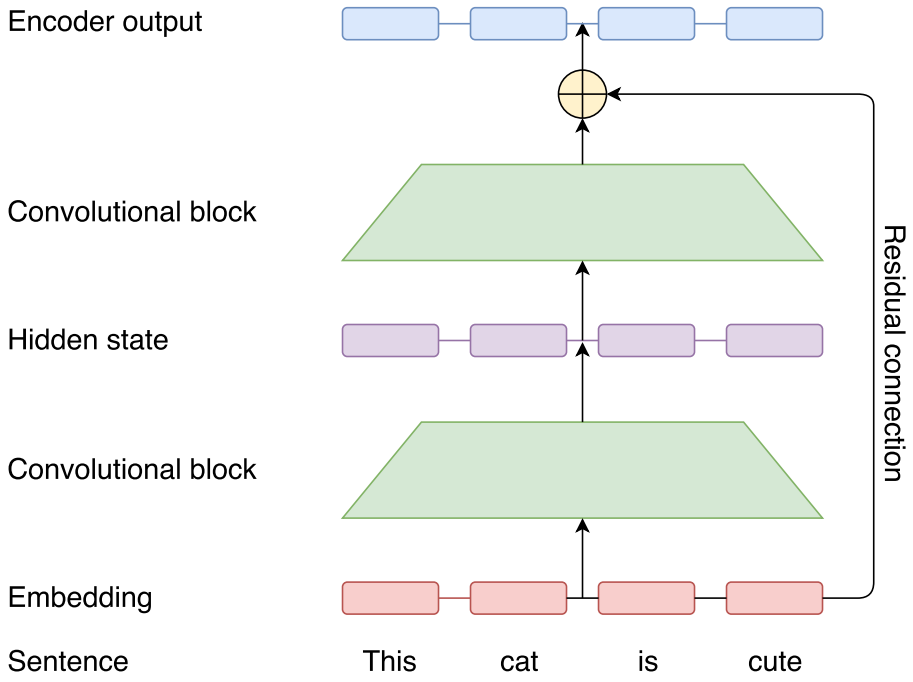

Figure 3.3: Architecture of the encoder.

3.3.4 Attention-based decoder

The decoder generates the output step by step. At each step, it generates a distribution of the next token given the source sequence and on-going generated sequence:

$$P(y_{t+1}) = P(y_{t+1}|y_1, \ldots, y_t, X) \tag{3.4}$$

We encode the generated sequence using the same architecture of the encoder. However, we use a different set of parameters in word embedding layer, position embedding layer, projection layer and convolution layer. The output of a convolutional block is presented in the formula 3.2.

Multilayer attention mechanism

We equip the decoder a sense of hierarchical decoding through the multilayer attention mechanism. We utilize attention mechanism at every layer of the decoder.

At time step $t+1$, the generated sequence $\mathbf{Y}_{1:t} = (y_1, \ldots, y_t)$ is mapped into distributed representation as $\mathbf{G}_{1:t} = (g_1, \ldots, g_t)$. We project the output d^l of the l-th layer back to the target embedding space. Then we combine it with the embedding of the generated sequence g_i.

$$h_i^l = (Wd_i^l + b_i) + g_i \tag{3.5}$$

The attention coefficient a_{ij}^l of the state i toward source element j at the decoder

layer l are computed:

$$a_{ij}^{l} = \frac{\exp(h_i^l \cdot e_j^u)}{\sum_{u=1}^{t} \exp(h_i^l \cdot e_u^u)} \tag{3.6}$$

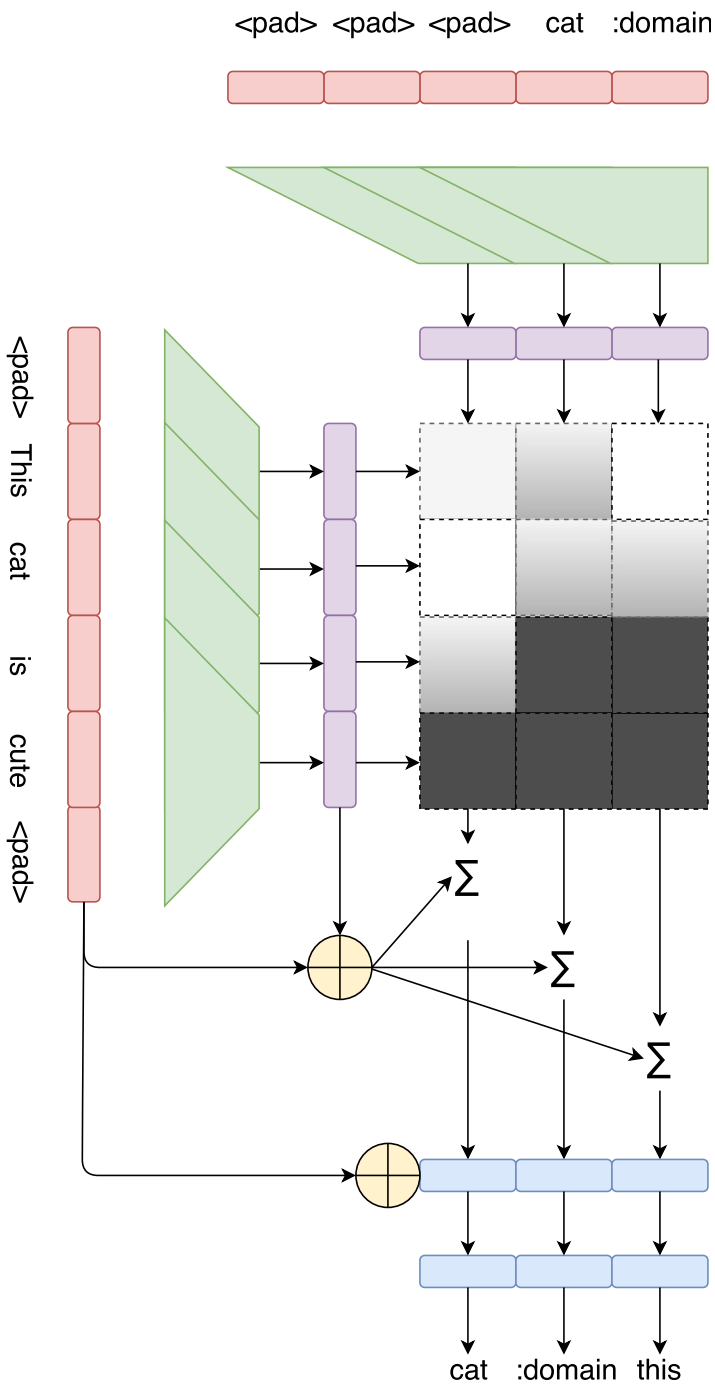

Figure 3.4: Architecture of the decoder.

The output of the attention layer l is computed as weighted sum of both the encoder output o^e and the last hidden state of the encoder e^u:

$$c_i^l = \sum_{j=1}^{t} a_{ij}^l (o_j^e + e_j^u) \tag{3.7}$$

Finally, we compute the distribution of the output token over V_{target} possible tokens of the intermediate representation:

$$P(y_{t+1}|y_1, \ldots, y_t, \mathbf{X}) = softmax(W_o c^u + b_o) \in \mathbb{R}^{V_{target}} \tag{3.8}$$

Figure 3.4 illustrates the computing process of the multilayer decoder.

Objective function

Since the translation-based model output a highly dimensional distribution, we minimize the negative log likelihood loss of the correct translated sequence G toward target sequence T similar to Sutskever et al. (2014) where $\mathbf{D}$ is the dataset:

$$\frac{1}{|T|} \sum_{G,T \in \mathbf{D}} \log P(G|T) \tag{3.9}$$

Chapter 4

Evaluation

This chapter first shows the annotation of JCivilCode-1.0 dataset and experiment settings used in this research. Since the graph conversion causes information loss to some AMR graphs, we present the estimation of the upper bound of the score for AMR parsing task. Finally, we present results of AMR parsing including accuracy, speed, and stability.

4.1 Annotation of JCivilCode

The Semeval competitions allowed participants to access multiple AMR corpus annotated manually but no large corpus has been made accessible to the public. Especially, there is no open AMR resource for any specific domain such as the juristic document or scientific document. Therefore, we manually annotated a corpus for the English version of the Japan Civil Code. The code is organized into multiple levels including chapter, part, article, paragraph, and sentence.

The pre-processing consists of the following steps: gathering articles, removing all article prefixes and article IDs, then splitting the article into sentences. We labeled each sentence with an ID containing the article name, the paragraph index, and the sentence index. To annotate the sentences, we used the web-based editor [1] provided by ISI group. This editor provides a combination of command line and graphical interface. The Propbank corpus is integrated into the search engine to minimize the time it takes to choose a proper meaning of the words. Two annotators are given a list of article sentences and annotate corpus independently. After finishing their own works, the annotators are invited to discuss and aggregate their outcomes into a single result. We call this dataset **JCivilCode-1.0**. The statistics of this corpus is presented in table 4.2 and table 4.4.

To evaluate the agreement of annotations, we calculate the SMATCH score between two annotations. The results are presented in table 4.1. As can be seen, the similarity of two annotations is very high at 92.25 points of SMATCH score. The precision and recall are fairly equivalent. These results suggest that the dataset was built with high quality.

[1] https://amr.isi.edu/editor.html

Table 4.1: SMATCH score between two annotations of JCivilCode-1.0.

Metric	**Score**
Precision	91.18
Recall	93.35
F-score	92.25

4.2 Datasets and Preprocessing

In this research, we evaluate our model on the AMR Annotation Release 1.0[2] (namely **LDC2014T12**) and AMR Annotation Release 2.0[3] (namely **LDC2017T10**). They were annotated by the Linguistic Data Consortium (LDC), SDL/Language Weaver, Inc., the University of Colorado's Computational Language and Educational Research group and the Information Sciences Institute at the University of Southern California. The former LDC2014T12 contains 13.051 pairs. The LDC2017T10 is a revised and expanded version of the LDC2014T12 with 39.260 AMR-sentence pairs in various domains of newswire, web blogs, and web discussion forums. Two datasets are organized into split and unsplit sets. The unsplit set categorizes the data by topics. The split set is divided into training/validation/testing sets. Table 4.2 shows the partitions of two datasets and training/valid/test setting of two datasets.

Table 4.2: Detail off AMR Annotation Releases.

Partition	Dataset	Train	Valid	Test	Totals
BOLT DF MT	♣★	1.061	133	133	1.327
Broadcast conversation	♣	214	0	0	214
Weblog and WSJ	♣★	0	100	100	200
BOLT DF English (1)	★	1.703	210	219	2.142
BOLT DF English (2)	♣	6.455	210	219	6.894
DEFT DF English	♣	19.558	0	0	19.558
Guidelines AMRs	♣	819	0	0	819
2009 Open MT	♣★	204	0	0	204
Proxy reports	♣★	6.603	826	823	8.252
Weblog	♣	866	0	0	866
Xinhua MT	♣★	741	99	86	926
JCivilCode-1.0	♠	741	99	86	926
Dataset					
LDC2014T12 ★		10.312	1.368	1.371	13.051
LDC2017T10 ♣		36.521	1.368	1.371	39.260
JCivilCode-1.0 ♠		0	0	157	157

[2]https://catalog.ldc.upenn.edu/ldc2014t12
[3]https://catalog.ldc.upenn.edu/ldc2017t10

Before putting pairs of sequences into neural machine translation model, we perform additional preprocessing to the pairs. The raw sentence is tokenized using Stanford CoreNLP tokenizer [4]. This tokenizer separates punctuations and words into tokens. After linearizing an AMR graph, we get two well-formated sequences including linearized graph and a tokenized sentence. We set a threshold of token frequency and replace less frequent tokens by an unknown token $<unk>$.

4.3 Experimental environment

We train the system using CUDA technology[5] which speeds up the parallel computation of matrix using graphic processing unit (GPU). Though the machine has multiple GPU cards, we train models separately on a single GPU. The specifications of the machine are presented in table 4.3:

Table 4.3: Specifications of the machine during training.

	Type
CPU	1 × Intel Xeon E5-2698 20 cores 2.2 GHz[6]
Graphic card	4 × Nvidia GTX 1080[7]
Memory	128GB DDR4

4.4 Result

4.4.1 Experiment 1: Upper bound of graph linearization

We estimate the maximum SMATCH score to prove the efficiency of this graph conversion method. All graphs in the official AMR corpus G are passed through a full linearization process L to get the linearized versions. These sequences were then put into a de-linearization process $D_L$to obtain the AMR graph set G'.

$$\begin{aligned} G' &= D_L(L(G)) \\ U(G, L, D_L) &= \frac{1}{n}\sum_{n} Smatch(G_i, G'_i) \end{aligned} \tag{4.1}$$

The upper bound of SMATCH score $U(G, L, D_L)$ which a graph conversion method (L and D_L) can deliver is calculated by equation 4.1. The result of the test is presented in table 4.4.

[4]https://stanfordnlp.github.io/CoreNLP/
[5]https://developer.nvidia.com/cuda-toolkit
[6]https://ark.intel.com/products/91753
[7]https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/

Table 4.4: Upper bound of SMATCH score on three datasets.

Dataset	Parenthesis	Reverse path
LDC2014T12	79	80
LDC2017T10	79	80
JCivilCode-1.0	79	79

Firstly, as can be seen from table 4.4, the experimental upper bound of all dataset are much higher than the performance of the current state-of-art systems which obtained at most 0.71 SMATCH score. Therefore, approaching AMR parsing with machine translation based method is a potential way if we can develop a robust neural machine translation model.

Secondly, the table shows a consistent boundary of SMATCH scores. As can be seen, the two linearization methods deliver a nearly constant upper bound among every dataset including standard datasets and our manually annotated dataset. The result suggests that the variety of dataset in term of size, vocabulary and annotation process have small effects on the stability of the representation of linearization process.

Finally, though the upper restriction of the SMATCH score is far from the performance of the most advanced parser, the limitation of the proposed linearization method will block the more robust sequence-to-sequence model in the future. Once the margin is narrowed down enough, we should propose another more advanced linearization method to open to a higher performance.

4.4.2 Experiment 2: AMR parsing

In order to evaluate the accuracy and throughput of the model, we train three sequence to sequence models namely **Bi-LSTM**, **Conv-Enc** and **Fully-Conv**.

Bi-LSTM model is built from a bi-directional LSTM encoder and an LSTM decoder with fast-forward connections proposed by Zhou et al. (2016). Conv-Enc model contains a convolutional encoder and a LSTM decoder proposed by Gehring et al. (2017a). The Fully-Conv detail explained in the section 3.3.

For word embedding, we use a learnable embedding layer similar to Gehring et al. (2017b) rather than a pre-trained distributed word representation (e.g. word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014)). To train the deep learning model, we employ the Momentum/Nesterov optimizer, a momentum optimizer for stochastic gradient optimization. We set up the hyper-parameters as shown in table 4.5.

Table 4.7 presents the SMATCH score of proposed models on LDC2014T12 and LDC2017T10 datasets. On the LDC2014T12, our models with reverse path linearization outperform reported system by a large margin of up to 6 points to the *CAMR*. Moreover, the performance of the Fully-Conv with parenthesis reached the highest score at 75.47 SMATCH score at 9 points higher.

[8]This paper reported the SMATCH score on the LDC2016E25 dataset which contains 39.260 AMR pairs. This dataset is an experimental dataset and was revised as LDC2017T10 dataset.

Table 4.5: Hyper parameters of experimental models.

	Bi-LSTM	**Conv-Enc**	**Fully-Conv**
Encoder			
Architecture	LSTM	CNN	CNN
No. hidden layers	4	6	4
Embedding size	512	512	512
Dropout rate	0.2	0.2	0.2
Decoder			
Architecture	LSTM	LSTM	CNN
No. hidden layers	4	4	3
Embedding size	512	512	512
Dropout rate	0.2	0.2	0.2
Attention mechanism	Luong attention	Luong attention	Multilayer
Totals			
Learnable parameters (approx)	13.000.000	10.000.000	11.000.000

Table 4.6: Performance of previous works of AMR parsing.

Method	**SMATCH score**
LDC2014T12 dataset	
NeuralAMR (Konstas et al., 2017)	62
Stack LSTM (Ballesteros and Al-Onaizan, 2017)	64
CAMR (Wang et al., 2015a)	66
CAMR (Wang et al., 2016)	66
LDC2017T10 dataset	
Character-based Translation (van Noord and Bos, 2017)	71[8]

On the LDC2017T10, the scores of models using reverse path were 1 point lower than that of the *Character-based translation model* however, models using parenthesis produced up to 74.28 SMATCH score. Overall, these combinations generate 3 points higher than the *Character-based translation model* did.

However, there is a crucial point of the result which should be investigated. The variation of SMATCH score of the model with the same linearization is tiny. Considering the difference in linearization creates a margin of 3-5 point of the score. Therefore, the table suggested that machine translation models did not contribute the accuracy of the system whereas the domination of linearization method did.

However, the consistency of score can be a critical point of this research, we perform a further experiment to test the stability of the model in the next experiment.

Table 4.7: SMATCH score on LDC2014T12 and LDC2017T10.

Linearization	**Model**	**LDC2014T12**	**LDC2017T10**
Reverse path	Bi-LSTM	72.70	69.74
Reverse path	Conv-Enc	72.70	69.74
Reverse path	Fully-Conv	72.71	69.75
Parenthesis	Bi-LSTM	75.43	74.25
Parenthesis	Conv-Enc	75.41	74.23
Parenthesis	Fully-Conv	**75.47**	**74.28**

4.4.3 Experiment 3: Testing of stability

Since there are more than ten millions learnable parameters in each experimental model, we verify the stability and convergability of the models by running ten-fold cross-validation. We split all partitions of the unsplit LDC2014T12 into ten folds. Then we calculate the mean and standard deviation of the SMATCH scores. In this experiment, we first try to test on this small dataset first because neural network models are unlikely to converge and stable on a small dataset. Moreover, training cross-validation takes so much time that we decide to try with small one first. The results are presented in table **??**

We can easily see from the table that the standard deviations of the SMATCH scores are tiny. These indicate that our models are likely to converge on different folds of the dataset. They also deliver stable performance on different distribution of the train/-valid/test set.

Table 4.8: Average and standard deviation of SMATCH scores on LDC2014T12.

Linearization	**Model**	**LDC2014T12**
Reverse path	Bi-LSTM	75.82 $\pm$ 0.82
Reverse path	Conv-Enc	75.83 $\pm$ 0.82
Reverse path	Fully-Conv	75.83 $\pm$ 0.82
Parenthesis	Bi-LSTM	74.40 $\pm$ 0.39
Parenthesis	Conv-Enc	74.39 $\pm$ 0.37
Parenthesis	Fully-Conv	74.40 $\pm$ 0.39

4.4.4 Experiment 4: More semantic evaluation

In this experiment, we target at investigating several specific aspects of semantic parsing such as named entity, negation, reentrancy, concept, semantic role labeling (SRL). **Named Ent** is measured by calculating F-score on the named nodes which have *:name* edge. There has been more attention to negation which is an important factor in semantic representation. **Negation** metric is the F-score of negated concepts which hold *:polarity* edge. Concept identification is a challenging task because it relates to word-sense disambiguation. Increasing the accuracy contributes largely to the higher accuracy of AMR

parsing. **Concepts** metric is the F-score of predicted concepts. Reentrancy is the distinguished property of AMR. The test of **Reentrancy** computes the SMATCH score on reentrancy edge. Finally, semantic role labeling which is an influential subtask of AMR is the identification of predicate-argument triplets. **SRL** metric measures the SMATCH score on *:arg* edges.

We followed the evaluation method proposed by Satta et al. (2017) and used their published evaluation source code[9]. The results of this evaluation are presented in table 4.9.

Table 4.9: Evaluation of semantic aspects on LDC2014T12.

Metrics	**Reverse path**			**Parenthesis**		
	BiLSTM	Conv-Enc	Fully-Conv	BiLSTM	Conv-Enc	Fully-Conv
Named Ent	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Negations	**0.92**	**0.92**	**0.92**	0.27	0.27	0.27
IgnoreVars	**0.91**	**0.91**	**0.91**	0.30	0.30	0.30
Concepts	**0.86**	**0.86**	**0.86**	0.63	0.63	0.63
Reentrancies	**0.73**	**0.73**	**0.73**	0.51	0.51	0.51
SRL	**0.91**	**0.91**	**0.91**	0.00	0.00	0.00

As can be seen from the table, there is no significant difference in accuracy of neural machine translation model which share the same linearization method. Two linearization methods reproduce nearly perfect on the named entity (reverse path at 0.9975 vs parenthesis at 0.9969). However, comparing two linearization method, *reverse path* linearization outperforms *parenthesis* on all other metrics. Reverse path creates an extremely wide margin to parenthesis method. The differences are above 0.30 on *Concepts* and *Reentrancies* and up to more than 0.60 on *Negation* and *IgnoreVars*. The reverse path also contributes to a high accuracy on semantic role labeling at 0.91 while parenthesis totally failed.

4.4.5 Experiment 5: Parallelization of CNN versus RNN

In this experiment, we aim at comparing the throughput performance of three conventional models which are bi-directional LSTM, convolutional encoder - LSTM decoder and fully convolutional encoder-decoder. We test on the LDC2017T10 dataset to evaluate the throughput of three models.

We set the batch size equally among three models. Specifically, we set $batch\text{-}size = 32$ because that is the maximum size that prevents the process from out-of-memory of the GPU card. We picked the throughput of the last epoch (in word/second) and reported in table 4.10.

As can be seen, the throughput of the *Fully-Conv* model is significantly higher than other LSTM-related models. *Fully-Conv* runs at three times faster than pure Bi-LSTM

[9]https://github.com/mdtux89/amr-evaluation

Table 4.10: Throughput of three models on LDC2017T10.

Model	**Reverse path**		**Parenthesis**	
	Word/s	Percentage	Word/s	Percentage
Bi-LSTM	6185	100%	5015	100%
Conv-Encoder	8532	138%	8464	168%
Fully-Conv	20423	330%	17209	343%

model. The speed of *Fully-Conv* doubles the speed of *Conv-Encoder* which is a mixed model of convolutional - recurrent model. *BiLSTM* model delivered the lowest throughput. Replacing the Bi-LSTM encoder by convolutional encoder boosted the throughput of the model of 50%.

The result from table 4.10 firmly indicates that convolutional neural network is faster than recurrent neural network because of the benefit of parallelization. The result is entirely in accordance with the assumption of the parallelizability of CNN.

4.5 Discussion

Linearization method might create two foreseeable issues though it significantly increased the accuracy of neural network method. First, entity redundancy occurs if the graph contains multiple nodes who share an identical concept. The second issue is the syntax error of the output because the neural network does not guarantee that the output follows the PENMAN notation. Table 4.11 shows some sample of JCivilCode-1.0 and output of our model. The bold words in the table show the error that our model generated.

Table 4.11: Two type of structural error.

Gold standard	**System**
[Node collision] A person who has become subject to the ruling of commencement of guardianship shall be an adult ward, and a guardian of an adult shall be appointed for him/her.	
(a / and :op1 (a2 / adult :domain **(p / person** :ARG1-of (s / subject-01 :ARG2 (c / commence-01 :ARG1 (g / guard-01))))) :op2 (a3 / appoint-01 :ARG1 **(p2 / person)** :ARG2 (g2 / guardian :poss p)))	(a0 / and :op1 (x0 / «unk» :domain **(p0 / person** :ARG1-of (s0 / subject-01 :ARG2 (c0 / commence-01 :ARG1 (g0 / guard-01))))) :op2 (a1 / appoint-01 :ARG1 **p0** :ARG2 x0 :poss p0))
[Syntax error] Unless otherwise provided by applicable laws, regulations or treaties, foreign nationals shall enjoy private rights.	
(e / enjoy-01 :ARG0 (n / national :mod (f / foreign)) :ARG1 (r / right-05 :ARG1-of (p / private-02)) :condition (p2 / provide-01 :polarity - :OR (o / or :op1 (l / law :mod (a / applicable)) :op2 (r2 / regulate-01) :op3 (t / treaty))))	(e0 / enjoy-01 :ARG0 (n0 / national :mod (f0 / foreign)) :ARG1 (x0 / «unk») :ARG1-of x0 :condition (p0 / provide-01 :polarity (x1 / -) **x0** **(o0 / or** **:op1 (l0 / law** **:mod x0)**

Chapter 5

Conclusion

We have presented our study on Abstract Meaning Representation Parsing using convolutional neural networks:

- We proposed a framework for AMR parsing and generation using neural machine translation method. This framework consists of two stages which are linearization/delinearization and translation.
- We estimated the upper bound of the effectiveness of representing a graph as a sequence. This proof suggests that there is room to improve the accuracy of AMR parsing by applying neural machine translation method into AMR parsing. However, it also revealed a fact that this room is becoming smaller.
- We proposed the utilization of convolutional sequence to sequence model on AMR parsing. Though comparing to LSTM-based model, this unlikely improves AMR parsing, however, convolutional architecture can remarkably increase the parallelization on GPU hardware.
- We published the first release of AMR testing set [1] for Japan Civil Code (English version). We hope that legal-specific dataset may encourage researcher to study on the legal domain.

Even though this research has proved the effectiveness of graph linearization and neural machine translation method in graph parsing, there are several limitations which we would like to investigate in the future:

- Although we claimed the reasons for the low performance of AMR generation, the experiment and observation of AMR generation were not enough to catch up with the performance of existing method (Konstas et al., 2017). More observation and modification should be conducted to outperform the NeuralAMR on text generation.

[1] https://github.com/nguyenlab/crest

- Since AMR parsing is a complex representation, we question whether SMATCH score is a good and efficient evaluation method. According to the paper of SMATCH score (Cai and Knight, 2013) and the source code published widely on their GitHub page[2], computing SMATCH score is solved using system optimization rather than exhaustive search. Therefore, the SMATCH score can be unstable once it faces very large graphs. As our observation, we raise some concerns about the reliability of SMATCH score on huge graphs.

- The high consistent of three neural machine translation cost a huge of time for training and evaluation. Therefore the evaluation of convergence and stability of model on large dataset LDC2017T10 has not been conducted as carefully as those on LDC2014T12.

[2]https://github.com/snowblink14/smatch

Publications

International conference:

1. Lai Dac-Viet, Vu Trong Sinh, Nguyen Le Minh, Ken Satoh, **ConvAMR: Abstract Meaning Representation Parsing for Legal Document**, (*Second International Workshop on Scientific Document Analysis*, SCIDOCA 2017)

2. Lai Dac-Viet, Truong-Son Nguyen and Le-Minh Nguyen, **Deletion-based sentence compression using Bi-enc-dec LSTM**, (*2017 Conference of the Pacific Association for Computational Linguistics*, PACLING 2017)

3. Minh-Tien Nguyen, Lai Dac-Viet, Huy-Tien Nguyen and Minh-Le Nguyen, **TSix: A Human-involved-creation Dataset for Tweet Summarization**, (11th edition of the Language Resources and Evaluation Conference, LREC 2018)

4. Nguyen, Minh-Tien, Lai Dac-Viet, Phong-Khac Do, Duc-Vu Tran, and Minh-Le Nguyen, **VSoLSCSum: Building a Vietnamese Sentence-Comment Dataset for Social Context Summarization** (*The 12th Workshop on Asian Language Resources*, ALR 2016)

5. Danilo Carvalho, Vu Tran, Khanh Tran, Lai Dac-Viet, Nguyen Le Minh, **Lexical to Discourse-level Corpus Modeling for Legal Questioin Answering**, (*The 10th International Workshop on Juris-Informatics*, JURISIN 2016)

Bibliography

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR*.

Ballesteros, M. and Al-Onaizan, Y. (2017). AMR parsing using stack-lstms. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1269–1275.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Barzdins, G. and Gosko, D. (2016). RIGA at semeval-2016 task 8: Impact of smatch extensions and character-level neural translation on AMR parsing accuracy. In *Proceedings of the 10th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2016, San Diego, CA, USA, June 16-17, 2016*, pages 1143–1147.

Bos, J. (2016). Expressive power of abstract meaning representations. *Computational Linguistics*, 42(3):527–535.

Cai, S. and Knight, K. (2013). Smatch: an evaluation metric for semantic feature structures. In *ACL (2)*, pages 748–752.

Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 933–941.

Dohare, S. and Karnick, H. (2017). Text summarization using abstract meaning representation. *arXiv preprint arXiv:1706.01678*.

Flanigan, J., Thomson, S., Carbonell, J. G., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of*

the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers, pages 1426–1436.

Garg, S., Galstyan, A., Hermjakob, U., and Marcu, D. (2016). Extracting biomolecular interactions using semantic parsing of biomedical text. In *AAAI*, pages 2718–2726.

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. (2017a). A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 123–135.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017b). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1243–1252.

Gildea, D., Xue, N., Peng, X., and Wang, C. (2017). Addressing the data sparsity issue in neural AMR parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 366–375.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Jie, Z., Muis, A. O., and Lu, W. (2017). Efficient dependency-guided named entity recognition. In *AAAI*, pages 3457–3465.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural AMR: sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 146–157.

Lai, D.-V., Nguyen, T.-S., and Nguyen, L. M. (2017). Deletion-based sentence compression using bi-enc-dec lstm. *2017 Conference of the Pacific Association for Computational Linguistics (PACLING)*.

Li, B., Wen, Y., Bu, L., Qu, W., and Xue, N. (2016). Annotating the little prince with chinese amrs. *LAW X*, page 7.

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. (2015). Toward abstractive summarization using semantic representations. In *NAACL*, pages 1077–1086.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119.

Munoz, M., Punyakanok, V., Roth, D., and Zimak, D. (1999). A learning approach to shallow parsing. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

Ramshaw, L. A. and Marcus, M. P. (1999). Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Rao, S., Marcu, D., Knight, K., and Daumé III, H. (2017). Biomedical event extraction using abstract meaning representation. pages 126–135.

Satta, G., Cohen, S. B., and Damonte, M. (2017). An incremental parser for abstract meaning representation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 536–546.

Schmitz, M., Bart, R., Soderland, S., Etzioni, O., et al. (2012). Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534. Association for Computational Linguistics.

Song, L., Zhang, Y., Peng, X., Wang, Z., and Gildea, D. (2016). Amr-to-text generation as a traveling salesman problem. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2084–2089.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Takase, S., Suzuki, J., Okazaki, N., Hirao, T., and Nagata, M. (2016). Neural headline generation on abstract meaning representation. In *EMNLP*, pages 1054–1059.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016). Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798.

van Noord, R. and Bos, J. (2017). Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands Journal*, 7:93–108.

Vanderwende, L., Menezes, A., and Quirk, C. (2015). An AMR parser for english, french, german, spanish and japanese and a new amr-annotated corpus. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 26–30.

Wang, C., Pradhan, S., Pan, X., Ji, H., and Xue, N. (2016). Camr at semeval-2016 task 8: An extended transition-based amr parser. In *SemEval-2016*, pages 1173–1178.

Wang, C., Xue, N., and Pradhan, S. (2015a). Boosting transition-based AMR parsing with refined actions and auxiliary analyzers. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, pages 857–862.

Wang, C., Xue, N., and Pradhan, S. (2015b). A transition-based algorithm for AMR parsing. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 366–375.

Zhou, J., Cao, Y., Wang, X., Li, P., and Xu, W. (2016). Deep recurrent models with fast-forward connections for neural machine translation. *TACL*, 4:371–383.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2017). Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 4189–4198.