Abstract Meaning Representation Parsing with Rich Linguistic Features

by

Wei-Te Chen

B.S., National Tsing-Hua University, Taiwan, 2005

M.S., National Tsing-Hua University, Taiwan, 2007

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2017

This thesis entitled:
Abstract Meaning Representation Parsing with Rich Linguistic Features
written by Wei-Te Chen
has been approved for the Department of Computer Science

Prof. Martha Palmer

Prof. James Martin

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Chen, Wei-Te (Ph.D., Computer Science)

Abstract Meaning Representation Parsing with Rich Linguistic Features

Thesis directed by Prof. Martha Palmer

Lexical and syntactic information have been shown to play important roles in semantic parsing. However, there is still no solid research on the relationship between semantic parsing and different types of linguistic knowledge that support this, e.g., lexical cues, dependency structures, semantic roles, etc. It is also known that dependency structures provide rich syntactic information for various NLP applications. Yet, few applications use dependency structures in an underlying neural network framework. This dissertation introduces a complete framework designed to parse Abstract Meaning Representations (AMRs), a semantic representation that expresses the meaning of a sentence as a directed acyclic graph. To enhance our AMR parser, we first develop a light verb construction (LVC) detector using a SVM. We also link input dependency parses to AMR concepts taking an EM-based approach to generate alignment pairs.

The main parser is split into three sub-components: a frame identifier, a concept identifier, and a transition action identifier. To support these components, we develop a Recursive Neural Network (RevNN) based model as the underlying framework of all three components. RevNN is based on dependency structures combined with distinct linguistic features. RevNN generates a corresponding vector representation for each dependency node, passing these vectors to the three identifiers as the underlying framework. By integrating all the above components, we design a transition-based parser which generates AMR graphs from input dependency parses.

Results show that our LVC detector surpasses comparable systems by 3 to 4% in F_1 score, and that this LVC detector supports the AMR parser. Our aligner improves F_1 score by 2 to 5% with LVCs information. Moreover, the resulting AMR parser achieves the best Smatch scores among other transition-based AMR parsers. We also show that the RevNN framework helps to integrate different linguistic features for improvement in accuracy of individual components.

Acknowledgements

I have received support and encouragement from a great number of individuals. Prof. Martha Palmer has been a tremendous mentor for me. Her guidance has made my PhD study a thoughtful and rewarding journey. I would like to thank my dissertation committee of Prof. James H. Martin, Prof. Wayne Ward, Dr. Mans Hulden, and Dr. Jinho D. Choi, for their encouragement, brilliant comments, and suggestions. In addition, I would like to express my special thanks to Michael Regan, who spends countless hours correcting and editing my proposal and dissertation write-up.

I would like to thank my labmates in Colorado CLEAR Lab: Shumin Wu, Tim O'Gorman, Will Styler, Steven Bethard, Lee Becker, Claire Bonial, Dimitriy Dligach, Skatje Myers, Riyaz Ahmad and Kristin Wright-Bettner. It is a wonderful experience to work and discuss with you all on either researches or projects. All of you have been their to support me when I need all kind of helps. I would also like to thank to my advisor, Jason S. Cheng, in National Tsing-Hua University in Taiwan, who guides me to the road of understanding NLP

Last but not the least, I would like to thank my family: my parents Dr. Liang-Kuang Chen and Chin-Hsiu Szu, for giving birth to me at the first place and supporting me spiritually throughout my life. Also my elder sister, Dr. Wei-Ting Chen, who encourages me to earn a PhD degree, and assists me when I first arrive in US. Special thanks to my girl friend, Fan-Yin Cheng, who supports and encourages me on everything.

Contents

Tables

Figures

Chapter 1

Introduction

1.1 Motivation

As natural language understanding becomes more popular, the demand for semantic parsing in natural language continues to grow. Meanwhile, with the availability of large annotated corpora and comprehensive lexical resources, as well as advances in statistical machine learning techniques, people are interested in how these resources and techniques can help semantic parsing. Traditionally, there are several different semantic representations, e.g., First-Order Logic, Abstract Meaning Representation (AMR) (Banarescu et al., 2013), and semantic role labeling (SRL), i.e., predicate-argument structure representation, etc. Each representation focuses on different linguistic aspects. The first two aim to express sentence level semantics, and the last one focuses more narrowly on predicate-argument structures, which are closely tied to syntax. Natural Language Processing (NLP) applications, e.g., question answering, temporal resolution, and entailment, all benefit from high quality, quick and automatic semantic parsing systems.

Nonetheless, although sentence-level semantic analysis is deeply connected to predicate-argument structures, and hence syntactic structure, recent research on semantic parsing favors the use of contextual information, e.g., distributional semantic features, over the use of syntactic structural information, e.g., dependency parses. As deep learning methods have become more popular, researchers tend to prefer methods involving lexical and distributional features as these are able to insert more data into their model. Recently, there have been attempts to apply a joint model which uses both lexical and syntactic information on semantic parsing tasks, both sentence-level semantic parsing and semantic role labeling (SRL). Due to the scarcity of gold standard semantic annotation data, most semantic parsing systems need to address the data sparsity problem. We believe that predicate-argument structure can play an influential role in semantic parsing since it supplies more coarse-grained patterns than lexical information.

AMR depends heavily on PropBank (PB) (Palmer et al., 2005) predicate-argument information since it uses semantic role labels as the relation labels between AMR concepts. This sharing aspect leads to speculation that a model that improves the performance of a semantic role labeler system may improve the accuracy and performance of an AMR parser as well. One way of doing this is to integrate semantic role information directly into a graph structure that also includes co-reference. A dependency structure is a natural choice for such integration; it is possible to replace some syntactic dependencies with semantic roles without altering the properties of the dependency structure. Once the integration is accomplished, an AMR parser can learn from this semantically-enriched dependency structure and its semantic roles, thus improving the identification of AMR concept and relation labels.

Certain linguistic phenomena that have not drawn much attention with respect to sentence-level semantic parsing may turn out to also be useful sub tasks in this process. For example, both nominal predicates and light verb constructions (LVCs), which consist of semantically general verbs and a noun that denotes an event or state, can function as the main predicate of a sentence and have been shown in recent studies to be important characteristics for semantic role labeler systems (Gerber and Chai, 2012; Do et al., 2017). We believe that these linguistic phenomena can assist the sentence-level semantic parsing task as well.

Meanwhile, deep learning has recently shown much promise for NLP research. Convolutional networks, recursive networks, recurrent networks, long short-term memory networks, sequence-to-sequence models, and attention mechanisms, all have achieved success in different NLP applications, including in machine translation (Devlin et al., 2014), dependency parsing (Chen and Manning, 2014), and semantic role labeling (FitzGerald et al., 2015; Zhou and Xu, 2015). In general, most applications use word vector representations, or what are called "word embeddings", as training features, and process each sentence at a surface form level. Word embeddings allow a model to share meaning between similar words. Syntactic information contributes less in these frameworks. However, dependency relations and dependency paths have proven to be important features in some tasks, e.g., SRL. The inclusion of syntax-based structures in a neural network provides not only a syntax-rich framework for NLP tasks but can also supply more general features than word-based systems, thus addressing data sparsity problems. On the other hand, there have also been attempts to incorporate syntactic and dependency information into the word embeddings directly (Levy and Goldberg, 2014; Melamud et al., 2015). We are specifically interested in seeing to what extent syntax-enriched word embeddings can aid the overall performance of semantic parsing.

1.2 Background

1.2.1 Semantic Role Labeling

Semantic role labeling is a process that identifies the participants that are involved in sentence-level semantic analysis. This analysis represents the roles of "who" did "what" to "whom", "when," and "where." For instance, in Sentence (1)

(1) Every day, [$_{Rider}$ Martin] **rode** [$_{Vehicle}$ the bus] [$_{Road}$ on highway 880] .

the predicate **ride** profiles a *ride_vehicle* event, with participant *Martin* as a rider, *the bus* as a vehicle, and *on highway 880* as the road. The predicate is usually a verb. In some cases, however, the predicate can be a nominal, an adjective or even an adverb phrase. Multi-word expressions (MWEs) can also function as predicates (i.e., LVCs (see Section 1.2.2).) With an individual predicate, the participants that are involved in the event are called arguments.

In this thesis, PropBank (PB) (Palmer et al., 2005) standards are used to represent each predicate and its semantic role relations. PB uses numbered arguments to express the core roles of the participants for each predicate. Generally, *Arg0* exhibits features of a prototypical agent, causer, or experiencer. *Arg1* is a prototypical patient or theme; *Arg2* is usually a prototypical beneficiary. For example, in Sentence (1), PB annotates *Martin* as *Arg0* and *the bus* as *Arg1*. In addition

Table 1.1: PropBank frame file for *free*.

free.01:	*be available*	free.03:	*cost nothing*	free.04:	*be unrestricted*
Arg0:	*causer*	**Arg1**:	*thing that costs nothing*	**Arg1**:	*source*
Arg1:	*source*	**Arg2**:	*of charge*	**Arg2**:	*free from*
Arg2:	*theme*			**Arg3**:	*resulting freedom*

to core arguments, PB uses *ArgM* to represent non-core modifier roles, e.g., time (*ArgM-TMP*), location(*ArgM-LOC*), cause (*ArgM-CAU*), etc. In Sentence (1), *on highway 880* is annotated as *ArgM-LOC* in PB. Each predicate in PB defines a set of senses and corresponding role sets. For example, the predicate *free* allows three different word senses: *be available*, *cost nothing*, and *be unrestricted*. (Table 1.1)

With the increasing availability of corpora annotated with semantic roles, people have turned their focus to developing automatic SRL systems. Given a target sentence with its predicate labeled for word sense, an automatic SRL system identifies the arguments with their proper roles. Since annotated arguments are always in the form of phrasal units, SRL processing benefits from syntactic structure which provides the phrasal units of a given sentence. Traditionally, there are two syntactic representations for SRL: constituent-based parse trees and dependency parse trees. Both constituent-based SRL (Gildea and Jurafsky, 2002) and dependency-based SRL (Johansson and Nugues, 2008; Choi and Palmer, 2011) approaches label arguments on the tree vertices of the phrase analysis. An example of semantic roles on both constituent and dependency parse trees is shown in Figure 1.1.

In order to speed up SRL processing time and reduce the number of candidate tree vertices, several constraints are proposed (Xue and Palmer, 2004; Zhao et al., 2009). For constituent-based SRL, candidate arguments are the constituents that are siblings of either the predicate or the predicate's ancestors. Additionally, the direct children of prepositional phrase arguments also become candidates. For dependency-based SRL, only the direct dependency children are considered to be argument candidates.

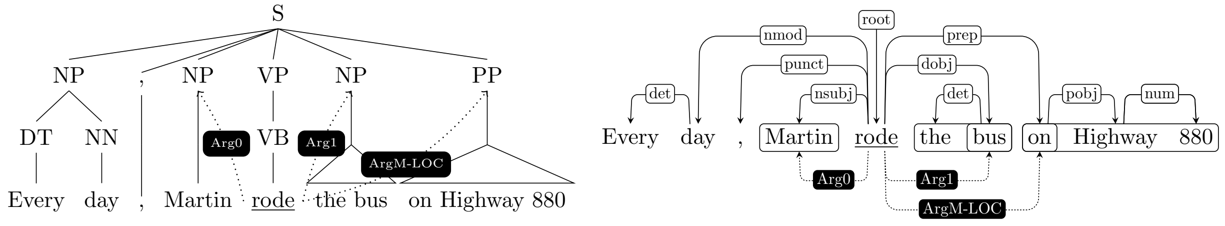


Figure 1.1: Semantic roles of Sentence (1) on a constituent parse tree(left) and dependency parse tree(right). The semantic roles for the three participants are `Arg0`, `Arg1`, and `ArgM-LOC` (labeled with black tags)

1.2.2 Light Verb Constructions

As a linguistic construction for which the relational semantics of verbs can be extended in novel ways, English light verb constructions represent a powerfully expressive resource. These constructions, such as *make an offer* and *give a groan*, consist of a semantically general verb and a noun that denotes an event or state. However, an exact definition of LVCs, which would precisely delimit these constructions from either idiomatic expressions or compositional, "heavy" usages of the same verbs (e.g. *She made a dress; He gave me a present;* etc.) remains under debate. In this thesis, we follow the definition for LVCs given by Butt (2003). LVCs are characterized by a light verb and a predicate complement that "combine to predicate as a single element." In LVCs, the verb is considered semantically lightened in such a way that the verb does not fulfill its full predicating power. Thus, the light verb plus its true predicate can often be paraphrased by a verbal form of the true predicate without loss of the core meaning of the expression. As we compare LVCs with nominalizations, a slight difference between these two constructions appears. The meaning of an LVC can be affected by both the light verb plus the eventive noun. However, the nominal dominates the whole meaning of the nominalization predicate. Also, the roles that are involved in an LVC depend on both the light verb and the eventive noun, while the roles of a nominalization are affected by the nominal predicate only. As we go through the OntoNotes (in Section 2.1.3) 4.99 data release, there are 15,349 nominal gold annotations across all genres of text. Of these, 1,659 are LVCs.

As a result of its complicated construction, it is no surprise that the automatic detection of English LVCs remains challenging, especially given the semi-productive nature of LVCs which allows for novel LVCs to enter the language. Novel LVCs are particularly difficult to detect, yet their identification and interpretation is key for NLP systems. For example, NLP systems need to recognize that *She made an offer to buy the house for $1.5 million* is an *offering* event, rather than a *making* or *creation* event.

1.2.3 Abstract Meaning Representation

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic representation that expresses the meaning of an English sentence with a rooted, directed, acyclic graph. An example AMR in both PENMAN(Matthiessen and Bateman, 1991) format and GRAPH format is shown in Figure 1.2. AMR associates semantic *concepts* with the nodes on a graph, such that "*(p/person)*" refers to an instance of the concept person. In addition, a semantic *relation* is the label edge between concept nodes, such that "*(j / join-01 :time (d / date-entity))*" means there is a "*join*"(*j*) event in time unit "*date-entity*"(*d*). Meanwhile, AMR relies heavily on predicate-argument structures from PropBank, which share several core arguments as edge labels with AMR, i.e., all the *ArgN* tags. When one concept is involved in multiple roles, AMR employs a re-entrancy to represent co-reference. For example, in Figure 1.2, the concept *person* is related to two concepts, *join-01* and *have-org-role-91*. Moreover, the representation also encodes additional rich information, including named entities (NEs) (e.g., "*person*" in Figure 1.2), wiki-links, and discourse connectives. The design principle of AMR is to abstract away from syntactic idiosyncrasies. This means that when two sentences express the same semantic meaning with different syntactic structures, their AMR representations remain identical.

The AMR sembank provides a large whole sentence corpus with AMR annotations. With the availability of this corpus, researchers are focusing on AMR parsing. Basically, there are two approaches. The first approach is to treat the concept-relation matching problem as that of finding a maximum spanning graph (Flanigan et al., 2014; Werling et al., 2015). After identifying the corresponding word span of the sentence for the concept and learning a scoring function for a given

```
(j / join-01
  :ARG0 (p / person :wiki -
    :name (p2 / name
      :op1 "Pierre" :op2 "
         Vinken")
    :age (t / temporal-
       quantity
      :quant 61
      :unit (y / year)))
  :ARG1 (b / board
    :ARG1-of (h / have-org-
       role-91
      :ARG0 p
      :ARG2 (d2 / director
        :mod (e / executive
          :polarity -))))
  :time (d / date-entity
    :month 11 :day 29))
```

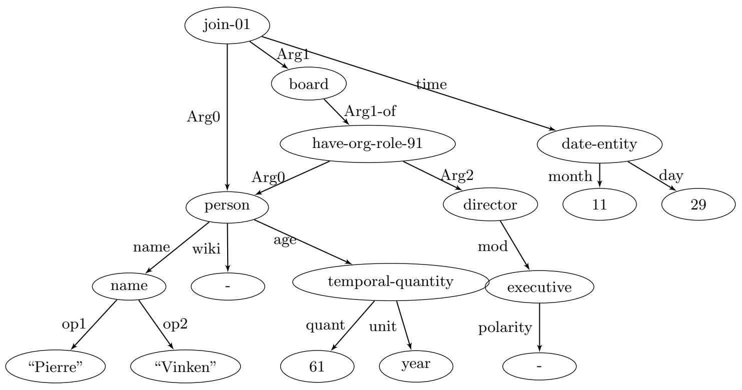


Figure 1.2: The AMR annotation of sentence "Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29." in PENMAN format (left) and GRAPH format (right).

pair of concepts and their relations, this approach determines an AMR graph by maximizing the sum of the score. A second approach uses a transition-based framework(Wang et al., 2015a,b) that defines a group of actions to convert from the English sentence in the form of a syntax parse tree to its corresponding AMR graph. As we mentioned above, AMR depends heavily on different types of linguistic information, e.g., semantic roles, named entities, etc., in its graph structure. Therefore, a transition-based parser benefits from the availability of these linguistic features in a dependency tree when determining its parsing actions.

The design of an AMR-to-English-sentence aligner is the first step in implementing an AMR parser, since AMR annotation does not contain links between AMR concepts and their corresponding span of words. One basic alignment strategy is to link the AMR tokens (either concepts or edge labels) with their corresponding span of words. Another strategy is to find the alignment from an AMR concept to the word node in the dependency parse tree. Since a dependency parse tree is a good structure for attaching rich information, e.g., NE tags, on its tree nodes, this rich linguistic information can be provided to develop a more accurate aligner. An alignment between an AMR concept and a dependency node represents a correspondence between the meaning of this concept and its child concepts and the phrase governed by this dependency node (i.e., head word).

1.2.4 Syntax-Based Vector Representation

Deep learning has become one of the most popular topics in machine learning in recent years. One important application of deep learning in NLP is representing words in vector space, or what we call “word embeddings”. This representation maps words or phrases to a vector of real numbers, and aims to assign close vectors to words with similar meanings. The main advantage of word embeddings over word classes is that embeddings incorporate various aspects of semantic and syntactic information in a vector in which each dimension corresponds to a latent feature of the word. As a result, a distributed representation is compact and less susceptible to data sparsity. Furthermore, some real world knowledge is learned implicitly for word embeddings with vector geometry operations. For example, the property “gender” is learned if we examine the following equation: $V_{king} - V_{male} + V_{female} \approx V_{queen}$, where V is the vector representation of a given word, and $+/-$ is the plus or minus vector operation. To train a general purpose word representation, unsupervised techniques, such as a skip-gram model (Mikolov et al., 2013; Collobert and Weston, 2008), are adopted. Word embeddings, when used as an underlying input representation, have demonstrated the ability to improve several NLP applications, including part-of-speech (POS) tagging (Collobert and Weston, 2008), syntactic parsing (Collobert and Weston, 2008; Socher et al., 2010) and sentiment analysis (Socher et al., 2013; Tai et al., 2015).

A word embedding represents word meaning based on the word or phrase itself and its surrounding words in the same sentence, omitting syntactic information. Researchers have attempted to introduce syntactic features into a neural network framework for NLP applications through different approaches. One idea is to include syntactic parse tree features in the framework. In this strategy, each tree node (either a constituent or dependency node) holds a distributed vector representing the syntactic information from the tree node itself and all its child nodes. Another idea is to use the path feature, which concatenates all the syntactic and direction labels through the tree path between two target tree nodes. The path feature has been shown to be a useful feature for semantic role labeling (Gildea and Jurafsky, 2002). However, neither the number of child nodes nor the number of nodes along the tree path are fixed. To gather all the feature vectors into one fixed n-dimensional vector, we need a technique that merges different vectors without losing essential information. For instance, a convolutional neural network (Waibel et al., 1990) is a traditional feed-forward network which processes portions of input elements and encloses them sequentially, "wrapping" the generated vector and the rest of the elements into a new vector. A recurrent neural network (RNN) (Elman, 1990) is designed to model sequential data and contains the memory units to record status from previous inputs. The information from previous elements is compressed and sent to the next neural unit. Then the next neural unit processes the output from the previous neural unit and the current input, generates a new information vector, and feeds it to the next neural unit.

RNNs have achieved significant success in different domains, including speech recognition, language modeling, image processing, etc. However, RNNs cannot control the flow of the information that the neural units really needs. RNNs tend to maintain too much information from nearby neural units while forgetting too much information from faraway neural units. To overcome this issue, long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) have been designed. The LSTM network introduces three different gates to control the information: a *forget gate*, which controls how much information to throw away from previous neural units; an *input gate*, which decides how much information from the current input should be kept; and, an *output gate*, which determines how much information should be transferred to the next neural unit. The benefit of LSTMs for NLP applications is great, especially in the context of sentence sequential processing, e.g., machine translation and POS tagging.

A common assumption about word embeddings is that neighboring words in the window size affect the target word sense the most. However, there are several arguments that question this assumption, including about how a dependency relation with the head word may affect the word sense more than that of any other dependency relation. By reducing the weight of the relations between a word and its surrounding context, the effects of function words and prepositional phrases are lowered, thus allowing long distance relations, e.g. relative clauses, to be better modeled. To achieve this, dependency grammars are incorporated into the model, adding syntactic information between a word and its head word to the word embeddings itself. As such, the dependency-based embeddings are less topical and exhibit more functional similarity than word embeddings generated in linear contexts.

Another attempt to inject syntactic knowledge into vector representations is to add syntactic information into the model itself. A recursive neural network is designed to extract vector representations for each node along a parse tree. By giving a syntax tree (i.e., constituent or dependency parse tree), the representation of a node is inferred by the lexical and syntactic information of the node itself and by the representation of its child tree nodes. This model keeps the syntactic information on the tree node representation. Since the representation is composed of nodes with direct syntax relations (e.g., head word and child nodes), a recursive neural network is helpful when dealing with long distance relations within a sentence.

1.3 Research questions and aims

This thesis sets out to answer three questions, 1) "Does better dependency parse to AMR alignment help the AMR parser?", and 2) "Do syntactic and semantic knowledge improve the performance of AMR parsing?" and 3) "Are syntax-based recursive neural networks useful, general approaches for AMR parsing?" Until now, AMR parsing and SRL have been treated as independent tasks. However, if our final goal is to find the correct AMR, and the elements of the AMR and semantic roles highly overlap, can we use SRL more effectively in the AMR parsing process?

On the other hand, although deep learning techniques have shown encouraging results in different NLP tasks, the application of syntactic structures in a neural network based framework has not been widely used. We have implemented a general recursive neural network based framework that supports AMR parsing well and which also uses powerful features.

Figure 1.3 shows the overall framework of the thesis. The computational systems that have been implemented are described in double-lined boxes (i.e., AMR parser, AMR-to-Dependency

Parse aligner, Frame Identifier, and Semantic Role Labeler). The primary four goals of this work are: First, to incorporate a LVC detector to support frame identification (Chapter 3); Second, to demonstrate the benefits of using dependency parses in AMR alignment for AMR parsing (Chapter 4); Third, to build a memory-unit-based AMR parser with rich lexical and syntactic information that generates correct AMRs (Chapter 5); Fourth, to develop a dependency node-based embedding feature and dependency-based word embeddings which are useful for the general purpose of various NLP applications (Section 5.2).

1.4 Thesis statement

The goal of this thesis is to improve the overall performances of AMR parsing components, including a frame identifier, concept identifier, and relation identifier, by leveraging the syntactic and semantic information from dependency parses and semantic roles. By constructing a recursive neural network based framework for AMR parsing with rich lexical and syntactic information (including syntactic dependencies, word embeddings, dependency node-based embeddings, named entities, semantic roles, and light verb construction information), we can improve all the various AMR parsing components. An essential prerequisite for this task is shown to be the alignment of dependency parses to AMRs.

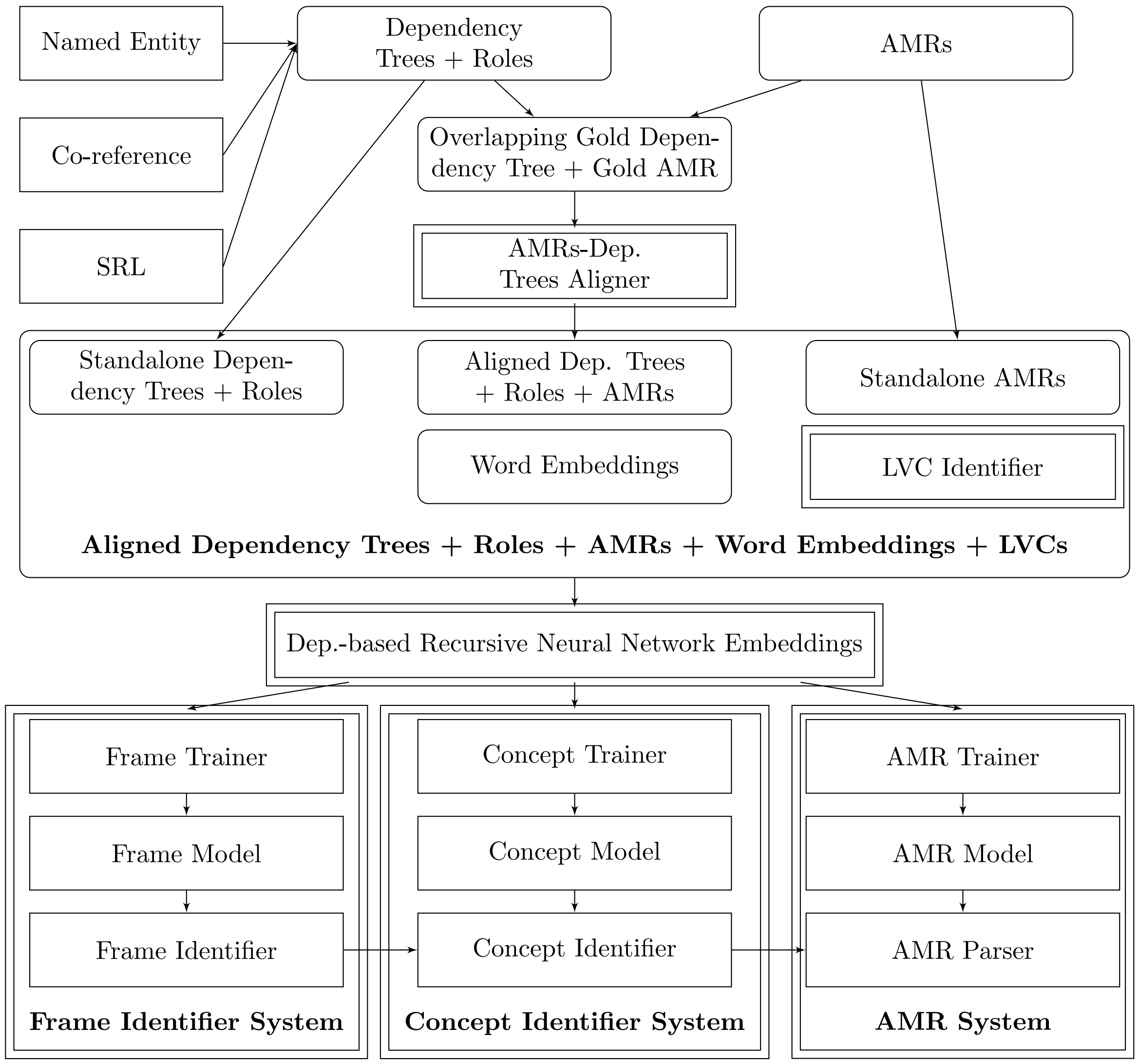


Figure 1.3: Overall framework. Systems to be implemented are described in double-lined boxes, and the data sets are described in rounded corner boxes. The plain rectangles are external systems that can automatically procide additional enrichments to dependency trees when gold annotations are not available.

Chapter 2

Related Work

2.1 Corpora

2.1.1 Semantic Corpus

The main semantic corpus we use in this thesis is PropBank (PB) (Section 1.2.1). The primary goal of PB is the development of an annotated corpus to be used as training data for supervised machine learning systems. The first PB release consists of 1M words of the Wall Street Journal portion of the Penn Treebank II (Marcus et al., 1994), annotated with predicate-argument structures for verbs, using semantic role labels for each verb argument.

In the past, PB annotation had been restricted to verb relations, but recent work has extended coverage to noun relations and complex relations like LVCs. In current practice, annotators identify light verbs and the main noun predicate in an initial verb pass of annotation. In a second pass, annotation is completed for the full span of the complex predicate, using the roleset of the noun. Consider the following example sentence,

(2) [$_{ArgM-temporal}$ Yesterday], [$_{Arg0}$ John] [$_{LightVerb}$ made] an [$_{rel}$ offer] [$_{Arg1}$ to buy the house] [$_{Arg2}$ for \$350,000]

, which uses the *offer* roleset but not the roleset of the light verb *make*. PB ensures that the complete argument structure of the complex predicate receives annotation, regardless of whether the argument is within the domain of locality of the noun or verb, and ensures that the roles assigned reflect the event semantics of the noun. The latest PropBank (Unification version) consists of 5,556

English frame files and 8,690 role sets. The latest PropBank (Unification version) consists of 5,556 English frame files and 8,690 role sets. And it contains 390,266 predicates in total, including 349,352 verb, 40,163 noun, and 2,191 adjective predicates. Among the verb predicates, there are 2,191 light verb constructions in the corpus.

Another available semantic corpus is Berkeley FrameNet (Baker et al., 1998), a computational lexicon which gathers a collection of annotated corpus attestations with examples of "frame elements" and their syntactic realizations. In frame semantics, a frame involves an interaction (frame) and its participants (roles). Currently there are 1,161 frames in the FrameNet database. A total of 12,609 lexical units are listed. Around 4,861 verbs, 5,135 nouns and 2,253 adjectives are attached to different frames. The British National Corpus (BNC) (Clear, 1993) was chosen as the FrameNet project example corpus. This balanced corpus with broad domains of data provides more than 100 million words and nearly 10,000 example sentences have been annotated with FrameNet. There is no overlap between AMR annotations and FrameNet so it was not used in this thesis.

2.1.2 Lexical Corpus

WordNet (WN) is a large electronic database of English words[1] , which was in part inspired by work in psycholinguistics investigating how and what type of information is stored in the human mental lexicon (Miller, 1995). WN is divided first into syntactic categories: nouns, verbs, adjectives and adverbs, and second by semantic relations. The semantic relations that organize WN are: synonymy (given in the form of "synsets"), antonymy, hyponymy (e.g. a Maple it is a tree; therefore, *tree* is a hypernym of *Maple*), and meronymy (part-whole relations). These relations make up a complex network of associations that is useful for both computational linguistics and NLP, and are also informative in situating a word's meaning with respect to other words.

Another interesting part of WN is the noun's "type", as indicated by the lexical file information. For each noun in WN, lexicographers have coded the noun with one primary superordinate, or lexical file, given forty-five numbered options. This thesis also focuses on this noun type since it

[1] http://wordnet.princeton.edu/wordnet/

can possibly denote events or states of a noun, which can theoretically combine with a light verb to form an LVC. The type designations that may denote eventive and stative nouns are listed in Table 2.1, and proved helpful in LVC detection.

Table 2.1: WordNet lexical file information types of interest for eventive and stative nouns

Name	**Nouns denoting...**	**Name**	**Nouns denoting...**
noun.act	acts or actions	noun.motive	goals
noun.cognition	cognitive process	noun.phenomenon	natural phenomena
noun.communication	communicative process	noun.possession	possession and transfer
noun.event	natural events	noun.process	natural processes
noun.feeling	feelings and emotions	noun.relation	relations between things
noun.location	spatial position	noun.state	stable states of affairs

2.1.3 OntoNotes

The OntoNotes project (ON) (Hovy et al., 2006) corpus integrates several layers of different annotation types in a single corpus, making it ideal training data for semantic analysis (Pradhan et al., 2007). The five layers of annotation include: 1) the syntactic parse from the Penn Treebank, 2) proposition structure from PB, 3) coarse-grained word senses from the ON sense grouping inventory, 4) named entity types, and 5) anaphoric coreference. The latest release, ON 5.0 (Weischedel et al., 2013), contains five various genres of text (newswire, broadcast news, broadcast conversation, telephone conversation, and web data) with 1.4 million English words.

ON sense groupings can be thought of as a more coarse-grained view of WN senses because these sense groupings were based on WN senses, which were successively merged into more coarse-grained senses, based on the results of inter-annotator agreement (Duffield et al., 2007). Essentially, where two annotators were consistently able to distinguish between two senses, the distinction was kept. Where annotators were not able to consistently distinguish between two senses, the senses were reorganized and tagged again. It was found that sense distinctions with this level of granularity can be detected automatically at 87-89% accuracy, making them effective for NLP applications (Dligach and Palmer, 2011). This sense inventory was used to annotate all ON verbs with more than three WN senses and some nouns. Unfortunately, the sense tagging is not complete for all of the ON corpus: there are about one million verbs and nouns in ON 5.0, but only 293,359 of these have sense tags (although many are surely monosemous), including 120,400 nouns with sense tags, 2,500 verbs, almost all verbs in WN with 3 or more senses, have tagged senses, comprising over 150,000 tagged instances. Each ON sense also lists which WN senses it includes, providing a mapping between ON annotations and WN senses.

2.1.4 Abstract Meaning Representation Corpus

A manually annotated AMR corpus in English is currently available. The LDC DEFT Phase 2 AMR Annotation Release 2[2] consists of AMRs with English sentence pairs. Annotated selections from various genres (including newswire, discussion forum, other web logs, and television transcripts) are available, for a total of 39,260 sentences. Among these gold standard AMRS, 446,214 concepts, 847,286 relations, and 401,372 attributes are included. This release follows the PropBank Unification frame file. In addition, the Little Prince Corpus, which annotates the novel *The Little Prince* by Antoine de Saint-Exupry, is available for pilot study. It consists of English and Chinese translations of 1,562 sentences in total.

2.2 English Light Verb Identification

There are two main approaches for the automatic identification of LVCs: contextually-based and statistically-based. Contextually-based approaches detect surrounding tokens and decide whether the verb-noun pair with the identified context words should be considered an LVC. Vincze et al. (2013) propose a contextually-based model, with a conditional random field machine learning method, for detecting English and Hungarian LVCs. Evaluation showed that their model performs well in various domains of LVCs and in detecting low-frequency LVCs. On the other hand, the statistically-based approach, which computes the degree of cohesion or association between target terms (e.g., log-likelihood ratio (Dunning, 1993) and mutual information (Church and Hanks, 1990)), finds LVCs among verb-noun pairs from a well-defined set of verbs and eventive

[2] https://catalog.ldc.upenn.edu/LDC2016E25

nouns (nouns denoting events, like *declaration*), with a classifier function deciding whether a pair should be considered an LVC or not. Van de Cruys and Moirón (2007) propose a statistically and semantically-based method for recognizing verb-preposition-noun dependency relation combinations of LVCs. Furthermore, Gurrutxaga and Alegria (2012) detect idiomatic and light verb-noun pairs from Basque using statistical methods.

To compare these two approaches, Tu and Roth (2011) propose a Support Vector Machine (SVM) based classifier to identify LVCs. They develop their system using both contextual and statistical features and analyze the deep interaction between them. They select 2,162 sentences with LVCs drawn from the BNC data as experimental data. Their SVM-based identification performances are 85.33% and 86.46% F_1 score on contextual and statistical feature models, respectively, for positive LVC instances, and 86.46% and 87.06% for negative LVC instances, respectively. They concluded that local contextual features perform better than statistical features on ambiguous examples, and combining them did not give better performance.

On the other hand, in The PARSEME Shared Task on Automatic Identification of Verbal Multiword Expressions (VWMEs) (Savary et al., 2017), several teams participated in the task of identifying multiword expressions, including idioms, light verbs, verb-particle constructions, and inherently reflexive verbs. A set of multilingual corpora in European languages is provided as experimental data. Most systems use shallow-based machine learning models, e.g., SVM and CRF, to detect multiword expressions. Further, the MUMULS system (Klyueva et al., 2017) uses a bidirectional recurrent neural network with single layer of GRUs. The output vector maps to the label ID with a softmax layer.

The author proposes a LVC identification system (Chen et al., 2015) based on contextual features with an SVM as well Tu and Roth (2011). On the other hand, although the experimental results show that shallow-based systems perform better than neural network-based systems in the VWME shared task, we are specifically interested in how a neural network-based model can improve LVC detection. Details about this research and experimental results are described in Chapter 3.

2.3 Semantic Role Labeling

In general, the processing of semantic role labeling is done in three steps (Gildea and Jurafsky, 2002). Upon giving the input as an English sentence with its parse tree, the SRL first identifies the predicate word. If the predicate is nominative or adjectival, an extra pass is needed to identify and label the token with the correct frame ID. The second step is to identify arguments. By following Xue and Palmer (2004), SRL extracts the candidate parse tree nodes denoting the arguments. Then a binary classifier determines whether the candidate is an argument or not. The final step is to match the role label to the argument. Here we need another multiclass classifier to decide the role label according to the frame file. This approach can be applied to either constituent-based trees or dependency-based trees, with both reaching comparable performances.

Pradhan et al. (2008) propose ASSERT, a constituency-based automatic SRL system. By using an SVM algorithm for argument identification and classifier, ASSERT demonstrates state-of-the-art results in CoNLL'04 - 05 shared tasks (Carreras and Màrquez, 2005). A set of rich linguistic features are used, including a parse tree path (see details in 2.6.1), phrase type, position, head word, etc. On the other hand, Johansson and Nugues (2008) built a dependency-based SRL system that reaches state-of-the-art results for CoNLL'08 - 09 shared tasks (Hajič et al., 2009). Choi and Palmer (2011) also propose a dependency-based SRL system with transition-based shift-reducing algorithm. To improve performance on unseen data, they use a clustering technique to group verb predicates, and suggest this clustering information as feature. Their experimental results show that their system achieves state-of-the-art performance in CoNLL'08 - 09 shared tasks.

Furthermore, several joint-inference SRL systems are proposed to improve SRL performance. Basically, the SRL models mentioned above take a pipeline approach. An SRL system with a pipeline approach means the target sentence must have a syntactic parse before it is submitted to the SRL system. A joint-inference approach takes the inference step for both syntactic parsing and SRL synchronously. The syntactic parser adds the arguments as external features, while the argument identification task benefits from more accurate syntactic features as well. The CoNLL'08

- 09 shared task profiles the challenge of learning both syntactic dependency parsing and SRL jointly. One advantage of using the joint-inference approach is that an SRL system might improve performance since the essential errors in SRL arise from incorrect syntactic structures (Pradhan et al., 2005). Several joint-inference approaches (Henderson et al., 2008; Titov et al., 2009) are proposed to learn the dependency parser and SRL simultaneously and achieve comparable results against the pipeline approach SRL system.

2.3.1 Semantic Role Labeling with Neural Networks

With the raising popularity of deep learning, more and more SRL models have begun to involve neural network techniques. Zhou and Xu (2015) propose an end-to-end SRL system which labels predicate-argument structure from English sentences directly without an intermediate level, including syntactic parse trees. To enfold the varied number of words in the target sentence into the network, they use a deep bi-directional LSTM (DB-LSTM) network, which transits the information in forward and backward passes and forms its subsequent hidden layer. At the top of the network, an IOB classifier is trained to identify the arguments by a Conditional Random Field model. Instead of using a predefined feature template on syntax parse trees, they choose only the distributed representation of a word, predicate context, and region mark of predicate as input features. They report comparable experimental results to other SRL systems which operate on syntactic parse trees with rich linguistic features.

FitzGerald et al. (2015) propose another SRL approach to integrate neural networks into the model. Their approach aims to generate distributed representations for predicates and arguments and train hidden layers to obtain vector representations for predicate-argument pairs. On the other hand, their approach maps the features of the target span of word into another vector representation. Their goal is to embed these two distributed representations into the same vector space. In the final layer, the score function, which denotes how likely it is that the span of the word is the correct argument given the predicate, is learned from the product of these two vectors. At inference time, a graphical model is used to represent global assignment of arguments to their semantic roles, subject to linguistic constraints. Their experimental results achieve a state-of-the-art SRL system on CoNLL'04 - 05 and CoNLL'08 - 09 datasets. An inspirational finding is that different frame file systems can improve the performance of frame identification for other systems. As they add more CoNLL'04-05 data (in PropBank) to a FrameNet-based identification system and retrain it, they gain better identification results.

2.4 Abstract Meaning Representation Parsing

There are three well-known AMR parsing models. One is a graph-based model (Flanigan et al., 2014; Werling et al., 2015) which searches for the highest score of an edge from the fully connected graph, a process which tends to optimize the AMR graph with global relations. The second well-known model is transition-based (Wang et al., 2015a,b), determining the highest score of an action which converts syntactic parse trees to AMR graphs greedily. In comparison with the graph-based model, the transition-based model tends to optimize AMR graphs with local relations. The third one is a sequence-to-sequence model (Peng et al., 2017; Barzdins and Gosko, 2016; Konstas et al., 2017) which is based on an encoder-decoder model and is inspired by neural machine translation. The encoder accepts context-aware representations as inputs and map them into hidden states. The decoder converts the hidden states to AMRs in linear form.

Meanwhile, although an AMR corpus is available, an AMR-to-English sentence aligner should be designed first before we start to implement an AMR parser. The current AMR corpus does not contain links between each AMR concept and its original span of words. Two different approaches, a heuristic method and an unsupervised method, are proposed to design aligners. The heuristic method (Flanigan et al., 2014) incorporates several straightforward alignment rules. Most of them are based on word forms and lemmas. The unsupervised method (Pourdamghani et al., 2014; Werling et al., 2015) aims to align AMR tokens to word span pairs based on co-occurrence probabilities in a corpus. Generally, the unsupervised method captures more underlying and reasonable alignment pairs than the heuristic method. All in all, a good aligner potentially improves AMR parser performance.

2.4.1 Graph-based AMR Parser

A graph-based AMR parser aims to find a connected graph with a maximum sum of edge scores. JAMR (Flanigan et al., 2014) is the first system for AMR parsing that implements a graph-based approach. This model first identifies concepts from a sequence of word spans in a given English sentence. The identification function is a linear parameterized function that optimizes the score generated by the product of the weight vector and feature vector of the word span. Features include word span length, named entity, etc. In decoding time, a dynamic programming (DP) algorithm is used to search for the best segmentation of a word span into concepts by maximizing the sum of the identification function score. Concept identification has a time-complexity of $O(n^2)$.

The next stage is to identify relations between concept pairs by a maximum spanning connected subgraph (MSCG) algorithm. This search algorithm is similar to a maximum spanning tree algorithm. There are four constraints for posting a relation between two concepts:

- **Preserving**: All the concepts identified by the previous stage are part of the final graph.
- **Simple**: At most one edge between any two concepts.
- **Connected**: Every vertex is reached from another vertex regardless of edge direction.
- **Deterministic**: Outgoing edge labels from the same vertex are never duplicated.

An **acyclicity** constraint is not included here, hence JAMR uses Lagrangian relaxation (Geoffrion, 1974) to supplement the MSCG algorithm, which assures the final connected graph is acyclic. The scoring function of relation identification is decomposed by edges and with linear parameterization. The search algorithm is initialized with concepts and empty connected edges. Then the highest scoring edge is added to the graph heuristically until all concepts are reached by another concept. Relation identification runs in $O(|V|^2 \log |V|)$. In 2014 JAMR reaches a 58% (and 56% in SemEval task 8) Smatch score (Cai and Knight, 2012) on automatic concept and relation identification data, and 80% on gold concept and automatic relation identification data.

Werling et al. (2015) extends JAMR and improves JAMR's concept identification. They claim that the concept identification task is more difficult and important than the relation identification task. Rather than combining span detection and label identification into one dynamic programming processing, they separate concept identification with an action-type classifier task. By determining the span of text in a given sentence with their aligner (described in Section 2.5), their action-type classifier assigns an action to the given word span according to features and action reliability. In the relation identification stage, their system reuses the MSCG algorithm from JAMR. This system achieves a Smatch score of 62.3%.

Foland and Martin (2016;2017) propose another AMR parser based on bidirectional long short term memory (BiLSTM). This approach is similar to the graph-based AMR parser, which separates the parsing processing into concept identification and relation identification steps. To express the underlying knowledge of sentence, they use the BiLSTM network to create an output vector to represent each sentence, and use this output for further usage. They design their concept identification, and four other sub-relation identification (argument, non-argument, leaf attribute, and wikipedia category) using the BiLSTM network. Their experimental results lightly surpass JAMR and CAMR in SemEval Task 8. In their latest publication, their Smatch score achieves 64.6%, which is the state-of-the-art result.

2.4.2 Transition-based AMR Parser

A transition-based method intends to generate AMR graphs through conversion from syntactic parse trees. Wang et al. (2015a) designed the CAMR system, which parses dependency parse tree structures into AMRs. CAMR first defines eight different actions (transitions), which transfer dependency tree segments into AMR sub-graphs (See Appendix A for the details of transition actions.) By traversing dependency parse tree nodes from leaf nodes first (in-order traversal), CAMR uses a greedy algorithm which selects the action with the highest score, and applies this action to the current sub-graph. An extended version of CAMR is proposed by Wang et al. (2015b), which acquires further features, including semantic roles, coreference, and word clusters. Their latest version of CAMR (Wang and Xue, 2017) improves their concept identifier by designing a bi-directional LSTM, and a character-level Embedding. Unlike a graph-based system that separates the parsing processing into two stages, the transition-based method collapses these two identification tasks into their transition-based framework. An average perceptron learning algorithm (Collins and Roark, 2004) is selected for learning the action score function. Their evaluation shows that CAMR reaches a Smatch score 5% better than JAMR for the LDC AMR 2013 release (63% vs. 58%.)

When compared to a graph-based method, a transition-based method shows two advantages. First, it provides richer syntactic information attached to dependency tree nodes than a graph-based model. This information potentially provides more hints in parsing. Second, the time-complexity of a transition-based system is faster than a graph-based system. The worst-case running time for CAMR is $O(n^2)$.

A comparison model that is worth mentioning is to use a transition-based model to parse Universal Conceptual Cognitive Annotation (UCCA) (Hershcovich et al., 2017). UCCA is a sentence level semantic representation which contains reentrance, discontinuous units, and coordination constructions. Their transition-based parser inheritances from Nivre's shift-reduce algorithm (Nivre, 2008). By applying a multi-layers bi-directional LSTM as their underlying structure, their parsr reaches the best F_1 score on in-domain and out-domain datasets.

2.4.3 Sequence-to-Sequence AMR Parser

Sequence-to-sequence models (Sutskever et al., 2014) have shown strong performance recently on various domains of NLP applications, e.g., machine translation and constituent parsing (Luong et al., 2015). A sequence-to-sequence model can be used for predicting future tokens when given previous ones. To build a sequence-to-sequence model, an encoder and a decoder are two essential components. An encoder receives a sequence of inputs (e.g., English sentence) and processes them into an intermediate vector representation. Then, in the next step, a decoder is trained on both the output sequence (e.g., the translated word in an MT system) as well as the fixed representation from the encoder. Since the decoder receives the intermediate vector representing the context of input elements, the decoder predicts future words based on the current word and the intermediate vector with greater accuracy. In general, a neural network model with memory units (e.g., RNN or LSTM) is adopted to implement the encoder and decoder. The final outputs of the memory unit in the encoding step are passed to the decoder as the context representation. These outputs of the memory unit in the decoding step are the tokens we want to predict. Although the full context representation is generated in the encoder step, the decoder just needs part of the context information to predict the future word based on the current word. An attention mechanism is designed to make the decoder focus on the relatively important part of the context representation while filtering out irrelevant information. In implementation, a simple feed-forward network is used to generate the attention. This feed-forward layer receives both the internal status from the encoder and the output of the decoder. Then the attention layer weights the hidden state of the encoder states. In the decoding step, the output vector of the attention feed-forward network is provided as part of the decoder inputs to predict the following tokens.

Due to the success of applying the sequence-to-sequence model to various NLP domains, researchers have turned their focus to using sequence-to-sequence models for AMR parsing. The idea is the encoder receives a target sentence word-by-word as inputs, and then the decoder predicts the AMR in a linear form. However, some issues that arise in this implementation have yet to be resolved. First, an efficient linearization, which converts the AMR graph into a series of concepts and relation tokens, is necessary. The linearization needs to either preserve the entire hierarchy between entities, or recovery from the linear form to the original AMR graph. On the other hand, the data sparsity problem is another critical issue that needs to be taken care of. The lack of gold annotations is always a major difficulty for AMR parsing. This issue is more crucial for sequence-to-sequence models which require extensive data for training an efficient encoder/decoder and attention mechanism.

There have been several attempts to adopt a sequence-to-sequence model for AMR parsing. Barzdins and Gosko (2016) propose an integrated model which mixes the results from a wrapped CAMR system and the results of a sequence-to-sequence model. The wrapped CAMR system contributes the most to AMR parsing, while the sequence-to-sequence model results in a 44.3% Smatch score. Their integrated model ties CAMR in the SemEval task, with a 62.0% Smatch score. Later on, Peng et al. (2017); Konstas et al. (2017) propose the first two successful sequence-to-sequence AMR parsers. Their models are built upon similar ideas. For linearization, they both use depth-first search to traverse the graph, then generate a series of AMR tokens. A rendering function which marks the scope of the AMR concept is also provided in their linearization. They both use anonymization of named entities and categorization of concept entities to reduce sparsity and account for unseen entities. However, the approaches of the two groups then diverge. Peng et al. (2017) use a categorizing approach to generate a compact vocabulary set for the data, while Konstas et al. (2017) use a large set of unlabeled sentences as an external data set. By taking a self-learning approach which uses its initial model to parse the external dataset, they expand their vocabulary size and reduce the unseen word rate. Konstas et al. (2017) achieve an AMR parsing result of a 62.1% Smatch score without using external semantic resources, e.g. dependency parsing and semantic role labeling.

2.4.4 Other AMR Parser Approaches

In addition to the previous two AMR parser models, there are other AMR parsing approaches that treats the parsing question in different ways. Pust et al. (2015) conceive the AMR parsing task as a string-to-tree syntax-based machine translation problem. They first convert AMR graph to a flat style tree, which create fake concept node as syntactic tree node. Then they reconstruct flat syntax trees with concept label and role label as intermediate tree node. This reconstructing step aim to transform a flat syntax trees to deeper length and more natural syntax tree which fit the syntax-based machine translation model better. Then tree relabeling step is applied. By following traditional MT approach, a syntax-tree to sentence aligner is used here (Pourdamghani et al., 2014). After the system obtain alignments between syntax tree node and phrase, the final step is to reorder the tree structure, which follows the original word order. After taking previous steps, they submit the resulting trees to a string-to-tree synta-based statistical machine translation system. Their experiment results surpass JAMR and CAMR system, and achieve 67% Smatch score in LDC AMR 2013 release.

On the other hand, Artzi et al. (2015) propose a Combinatory Categorial Grammar(CCG) (Steedman, 1996, 2000) based AMR parser. They first use CCG to construct lambda-calculus representations of the compositional aspects of AMR. Several following steps, include a joint representation of compositional and non-compositional semantics, and a CCG grammar induction algorithm to scale to longer sentences, are taken as well. Their experimental results achieve 66.3% Smatch score in LDC AMR 2013 release. In a similar work in Misra and Artzi (2016), the authors modify the CCG approach to a neural-based one. Their Smatch score reaches 65.3% on the newswire portion of LDC AMR 2013 release.

2.5 AMR-to-Sentence Aligner

In order to obtain links between AMR concepts and word spans before parsers learn their models, we need an automatic aligner to estimate alignments. JAMR provides a heuristic aligner between AMR concepts and words or phrases from the original sentence. They use a set of aligner rules, like NE, fuzzy NE, data entity, etc., with a greedy strategy to match the alignments. This aligner achieves a 90% F_1 score on hand aligned AMR-sentence pairs. On the other hand, the ISI Aligner (Pourdamghani et al., 2014) presents a generative model to align AMR graphs to sentence strings. They propose a string-to-string alignment model which transfers the AMR expression to a linearized string representation as the initial step. Their training method is based on the IBM word alignment model (Brown et al., 1993) by modifying the objective function of the alignment model. The IBM Model-4 with a symmetric method reaches the highest F_1 score, 83.1%. When separating the alignments into roles (edge labels) and non-roles (concepts), F_1 scores are 49.3% and 89.8%, respectively. In Werling's AMR parser (Werling et al., 2015), they conceive of the alignment task as a linear programming relaxation of a boolean problem. The objective function is to maximize the sum of action reliability. Each concept is constrained to align to exactly one token in a sentence. This ensures that only adjacent nodes or nodes that share the same title refer to the same token.

They hand-annotate 100 AMR parses, and their aligner achieves an accuracy of 83.2. By providing different alignments to their graph-based AMR parser, their aligner achieves a better Smatch score than JAMR's aligner.

However, two transition-based parsers, e.g., CAMR (Wang et al., 2015b) system and the RIGA system (Barzdins and Gosko, 2016), tie for the best results in SemEval 2016 task 8 (May, 2016). These transition-based parsers rely heavily on a dependency parser for AMR alignment. It is important to note that the JAMR aligner was not designed for this task where its alignment F_1 score is only 69.8%. In order to deal with this problem, we designed a dependency parse to AMR aligner which estimates alignments by learning the feature probabilities of lexical (surface) forms, relations, NEs and semantic roles jointly (Chen, 2015; Chen and Palmer, 2017). The goal for estimating these feature probabilities is to further develop the AMR parser with these initial models. Details and extensions of this aligner are discussed in Chapter 4.

2.6 Deep Learning with Syntactic Information

As neural networks have become popular in recent years, and neural network language models achieve great success in various types of NLP applications, researchers hope to integrate syntactic information into different neural networks. We group these approaches into three categories: a parse tree path method, a syntax-based emebedding method, and a syntactic structure method.

2.6.1 Parse Tree Path Feature with Neural Networks

A parse tree path is the concatenation of tree node symbols and direction tags (upward or downward movement) from one tree node through a parse tree to another node. In Figure 1.1, the constituent parse tree path between *predicate* rode and *Arg0* Martin is $\underline{VB \uparrow VP \uparrow S \downarrow NP}$, while the dependency parse tree path between the same two words is $\underline{nsubj \downarrow}$. A parse tree path is a useful feature for extracting relations between any two tree nodes, e.g., SRL (Gildea and Jurafsky, 2002) and relation extraction (Bunescu and Mooney, 2005; Kambhatla, 2004; Xu et al., 2015). Take the SRL task that uses Figure 1.1 as an example here. The parse tree path $\underline{VB \uparrow VP \uparrow S \downarrow NP}$ is a reliable indicator to determine Martin is *Arg0* of *predicate* rode, since it is capable of finding the subject of rode, which is most likely an *Arg0*. The parse tree path not only captures the governing category, e.g., subject or direct object, on parse trees, but it also captures the underlying knowledge from the combination of syntax labels and directions. However, the number of variations of tree paths is huge, which makes this feature more susceptible to data sparseness. With the development of neural network techniques, a parse tree path is designed to map to a single vector representation that sustains tree path information and overcomes data sparseness.

For instance, Hermann et al. (2014) design a dependency parse tree path feature in the form of a distributed representation that composes words and dependency labels. This method is used to identify the frame of each predicate. By giving the target predicate and argument, its parse tree path feature is denoted as the average of word embeddings along the parse tree path. To learn a mapping function that reduces the dimension of a parse tree path vector, the W_{SABIE} algorithm (Weston et al., 2011) is used here. The same technique is used in their argument identification task as well. However, one problem for this approach is that the average of word embeddings cannot maintain useful information and does not throw away irrelevant words along the path. To address this issue, Xu et al. (2015) propose a shortest dependency path - long short-term memory model (SDP-LSTM), which aims to capture and classify relations between any two dependency tree nodes. After separating the dependency parse tree path by the common ancestor node of the two nodes, these two sub-paths are mapped to vectors that are composed of four channels (i.e., word distributed representation, POS, dependency label, and WordNet hypernyms) with the LSTM network. The LSTM network can control information by three different control gates (Section 1.2.4.) They also design a dropout strategy to alleviate the overfitting problem, which chooses to throw away word embeddings, the gates in the LSTM unit, and the middle hidden layers of LSTM network. SDP-LSTM is the first model to use LSTM and dependency parses on the relation classifier task.

2.6.2 Syntax-based Embeddings

Word embeddings based on a linear bag-of-words model have achieved excellent success in recent NLP research. Still, other attempts to incorporate both lexical and syntactic knowledge into the word embeddings have been proposed. Dependency-based word embeddings (Levy and Goldberg, 2014), which derive contexts based on the syntactic relations of each word, are designed as an alternative to the bag-of-words approach. Using a corpus of dependency parse annotations, this model extracts the relation label between a head word and its child words. These word pairs replace the context words in a linear context model as input data. At training time, the model reuses the approach from the skip-gram model (Mikolov et al., 2013), which uses a negative-sampling objective function. With each pair (w, C) of word w and its context (i.e., dependency child) words C from the dependency parse, the model generates a second pair (w', C) as an extra training instance, where w' is a randomly selected word along with the original C. Both the training pair (w', C), which is the negative sample, and the positive sample, (w, C), are then provided for the model to learn latent parameters.

One important advantage of this approach to word representations is that it captures the relations between words that are far apart but are associated via long distance dependency relations. Also, this model avoids "coincidental" contexts which are within the window but not directly related to the target word. In addition, linear-context word embeddings tend to categorize words into the related domain, while dependency-based word embeddings prefer to gather words with similar syntactic functions.

2.6.3 Interaction between syntactic information and neural networks

To further develop grammatical relations and structures in the framework, a syntax-based distributed representation is designed to either capture the relation between two nodes or train the network hierarchically, based on the parse tree. Socher et al. (2010) propose a recursive neural network (ReVNN) which learns a syntactic tree parsing model and word embedding jointly. The model aims to learn distributed representations for each constituent-based binary tree node by folding vectors of its two child nodes. Then the corresponding vector is used to jointly learn 1) a parse tree parsing model, and 2) distributed representations of phrases/words at the bottom of the syntax tree by back-propagation during training. Based on ReVNNs, several alternative networks, e.g., matrix-vector ReVNNs (MV-RNN) (Socher et al., 2012) and Recursive Neural Tensor Networks (RNTN) (Socher et al., 2013), are presented as a general model for NLP applications. For example, in Socher et al. (2013), the researchers compare these three different recursive-based neural networks on a sentiment analysis task. On each tree node, the composition function outputs two items. The first one is a distributed representation of the node itself (by giving the two vector representations of its child and the composition function). The second one is a label to indicate positive/negative sentiment of the text segment corresponding to the tree node.

Moreover, some researchers use ReVNNs for relation identification and similar tasks. Hashimoto et al. (2013) propose a RNN-based model for semantic relation classification. Extending from ReVNNs, they change the composition function by introducing weights on different child nodes. In general, the syntactic head tends to be assigned with a higher weight. Syntactic paths are another factor that affects the weight of different child nodes. Also, they obtain model parameters by averaging these parameters from all rounds of training. In addition, Tai et al. (2015) propose a LSTM-based model (Tree-Structured LSTMs) which applies a recurrent network technique to the parse tree node composition function. Like regular ReVNNs, each tree node corresponds to one composition function. In this, the composition function is a LSTM unit. Each LSTM unit receives inputs from the vector representations of its child nodes. The biggest difference from traditional LSTMs is that the forget gate in their LSTM unit controls information from its child nodes. Then the information from more important child nodes (e.g., the syntactic head word) can be preserved in the parent tree node. The Tree-Structured LSTM is fitted to both constituent and dependency parse trees by using various models (N-ary Tree-LSTMs and Child-Sum Tree-LSTMs, respectively.) They experiment with Tree-Structured LSTMs on two NLP applications: sentiment classification and semantic relatedness, showing significant improvements on both tasks. They conclude that Tree-Structured LSTMs have advantages over other networks especially with respect to long distance relations.

Chapter 3

Light Verb Construction Identifier

3.1 Overview

Before we introduce our LVC identifier, we want to examine the important role that LVCs play in AMR. We examine the overlapping sentences that we use in our aligner experiment (in Section 4.2.), and count how many concepts are aligned to eventive nouns in the training data. Results are shown in Table 3.1. As we can see, among 9,100 AMRs, there are 295 LVCs that appear in original sentences. Among these 295 LVCs, we find only 59 of them (20%) use light verbs as concepts in AMRs. The remaining AMRs (80%) use eventive nouns as concepts. Given this observation, we have high confidence in asserting that if an LVC appears in a sentence, the eventive noun should be used as the concept rather than the light verb.

An LVC identifier determines what combinations of potential light verbs and eventive nouns should be labeled as LVCs. For a given dependency tree T, the system first checks if T meets certain syntactic criteria in order to decide if T should be put into the candidate set (these criteria are described in more detail in Section 3.2.1). Next, the light (or "Support") verb V_S and eventive noun N_E pair is submitted to an LVC binary classifier, which labels the V_S-N_E pair as an LVC or

Table 3.1: Number of LVC appearing in AMR.

# of sentence	9,100
# of predicate concept	40,102
# of LVC	295
# of light verbs in AMR	59

not. This supervised classifier is trained with the LibLinear (Fan et al., 2008) algorithm.

3.2 Light Verb Construction Classifier

3.2.1 Candidate Identification

The first step in LVC recognition is to select the candidate dependency trees for the training of the classifier. Here, the PB layer of the ON 4.99 corpus is used as a starting point. For this research, we chose to exploit LVCs that are composed of a limited set of the most frequent light verbs, focusing on the six most frequent light verbs in the data, which cover 99.26% of the V_S-N_E pairs.

The second step is to select the eventive nouns on which to focus. Our starting point for this process was to make use of a list of eventive and stative nouns drawn from WN (initial list provided by Christiane Fellbaum, personal communication). For the next step, only the dependency trees containing the previously mentioned six light verbs and eventive nouns on the candidate list are selected.

The last stage is to extract the dependency relation between the light verb V_S and eventive noun N_E. The following three cases are considered:

(1) The N_E is the direct child of the V_S with a direct object relation between them (see Figure 3.1)

(2) The N_E follows a quantity or partitive expression, and the quantity is the direct object of the V_S (see Figure 3.2)

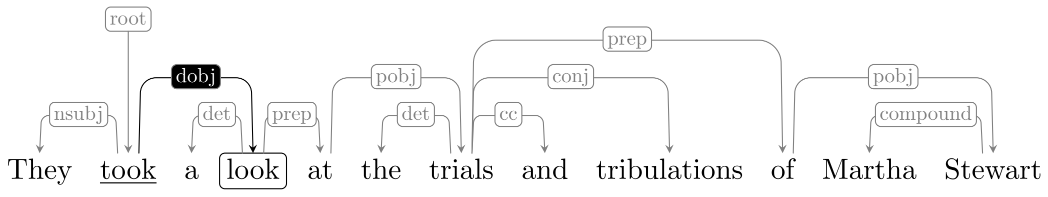


Figure 3.1: A dependency relation where nominal is the direct child of the verb. In this example, the eventive noun N_E = "look" is the direct object of the verb V_S = took.

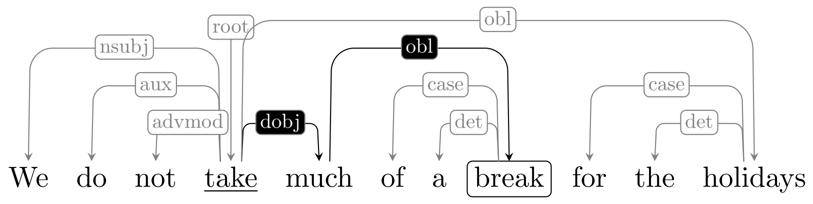
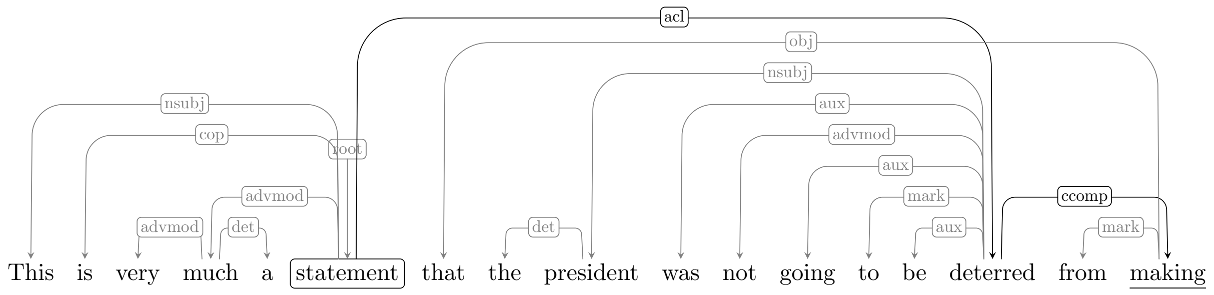


Figure 3.2: A quantity is the direct object of the verb. In this example, the quantifier "much" is the direct object of the verb V_S = take, and the eventive noun N_E = "break" follows the quantity.



acl
obj
nsubj
nsubj
aux
cop
advmod
root
aux
advmod
mark
ccomp
advmod
det
det
aux
mark
This is very much a statement that the president was not going to be deterred from making

Figure 3.3: A dependency relation where the eventive noun is the head of the clause. In this example, the eventive noun N_S = "statement" is the head word of the clause that contains the verb V_S = making.

(3) The N_E is the head word of the clause (see Figure 3.3)

Only the dependency trees containing the V_S and N_E in one of the previous three syntactic relation combinations could be considered as candidate sets. By following these steps, we extract 1,739 LVCs among the 1,768 LVC instances in the entirety of the ON 4.99 data. Thus, we detect 98.42% of the LVCs in the gold annotations. Error analysis of the 28 instances that were not detected revealed that they were missed either because of long-distance dependences and/or intervening relative clauses, and a few were annotation mistakes. The distribution of the three relation combinations is displayed in Table 3.2. The most frequent relation type is the direct object relation, which makes up 87.44% of all the LVCs.

3.2.2 Classifier

In LVC recognition, our classifier assigns a binary label (+1/-1) to an individual V_S-N_E pair on the dependency tree. Here, we use the LibLinear machine learning algorithm, adopt-

Table 3.2: Distribution of Dependency Relation Type of LVCs in ON 4.99 data

Relation Type	**Numbers**	**Portion**
I. Direct Chlid	1,546	87.4%
II. Quantity	16	0.9%
III. Head of the clause	178	10.1%
Sub-Total	**1,740**	**98.4%**
Other Type	**28**	**1.6%**
Total	**1,768**	**100.0%**

ing L2-regularized logistic regression. This algorithm uses the following approach: given a set of instance-label pairs (x_i, y_i), where $x_i \in R^d$, $y_i \in$ {1,-1}, a function $f : x_i \rightarrow y_i$ is found that maximizes the likelihood estimation of the classifier's parameters, which assumes that x_i was generated by a binomial model that depends on y_i (McCullagh and Nelder, 1989). One advantage of using LibLinear over a Support Vector Machine (SVM) is the training and prediction speed. The dimensionality of our features is often very high, but the training sample size is small. LibLinear performs only logistic regression without using a kernel. Results show that LibLinear reduces both training and decoding times, while maintaining the accuracy of the prediction.

Several features are used by the classifier. We categorize them into three different types: Basic Features, OntoNotes Word Sense Features, and WordNet Features.

Basic Features Basic features include the word and its lemma, the part of speech (POS) tag, and the dependency relation of V_S and N_E. Two paths, the sequence of POS tags and the sequence of dependency relation labels, are included as well. Additionally, a subcategorization frame which concatenates the dependency labels of V_S and N_E is adopted. These features are used either individually or jointly (e.g., POS of V_S and lemma of N_E make another new feature). The basic features are listed in Table 3.3.

OntoNotes Word Sense Features Word sense plays an important role in recognizing LVCs. In the ON 4.99 corpus, a word sense tag is annotated on the verbs with three or more senses and many nouns. The coarse-grained sense inventory, described in Section 2.1.3, gives a definition for each word sense. Ideally, for the data to be the most effective for LVC detection, all verbs and

Table 3.3: Basic Features ($W^{-1/+1}$ refer to the left word / right word of W, and W^h refers to the head word of W).

Lemma: The lemma of V_S, N_E, V_S^{-1}, V_S^{+1}, N_E^{-1}, N_E^{+1}, V_S^h, N_E^h
POS: The part of speech tag of V_S, N_E, V_S^{-1}, V_S^{+1}, N_E^{-1}, N_E^{+1}, V_S^h, N_E^h
Dep: The dependents of V_S, N_E, V_S^{-1}, V_S^{+1}, N_E^{-1}, N_E^{+1}, V_S^p, N_E^h
RelPath: The concatenation of the relation on the dependency tree path from V_S to N_E
PosPath: The concatenation of the POS on the dependency tree path from V_S to N_E
DepSubcatSet: The subcategorization set that is derived by collecting all dependency labels of V_S and N_E
Voice: Active or Passive for the V_S
Distance: The dependency tree node distance between V_S and N_E

nouns would have sense tags. Unfortunately, not all of the subcorpora of the ON corpus are sense tagged. In the first step of our ON data experiment (in Section 3.3.2), the verbs and nouns in the automatically generated dependency trees do not contain any of the word sense tags. Hence, a Word Sense Disambiguation (WSD) model is necessary. In Lee and Ng (2002), a SVM-based WSD model that integrates the lemma, POS tag, and collocation information from nearby words is proposed. We apply this model to the WSD task with ON word senses labels and implement our WSD classifier with the LibLinear algorithm with L2-regularization and L1-loss support vector classification. This algorithm uses a linear classifier to maximize the margin instead of using a kernel. For the target word, we select ± 3 words as the window size, while we adopt the same feature list that was used in Lee and Ng (2002).

We train and test our model on the ON data for out-of-genre experiments (See Section 3.3 for the details of the data preparation). Our WSD model reaches 76.16 Precision, 71.32 Recall, and 73.66 F_1 score. Although the overall performance of our WSD model is not ideal, the predicted word sense tag is only used in the automated generated dependency trees as one feature that supports the improvement of our LVC recognition.

WordNet Features WordNet (WN) contains rich word sense information and relational information between words. In our model, several pieces of WN information are used as features:

WordNet Sense: The fine-grained WN sense inventory provides word sense information for each

Table 3.4: OntoNotes Sebse Inventory Mapping Example

	id = 1.1 *carry out, enact*	id = 1.4 *bring with, transport, move*	id = 1.5 *select, accept, come into possession of*
WordNet	*take*: 1, 15, 16, 24, 25, 28 *take apart*: 1, 2, 4 *take off*: 4 *take over*: 6	*take*: 3, 7, 27, 30 *take back*: 1, 3 *take down*: 1, 3, 7, 8 *take up*: 9	*take*: 4, 8, 10, 12, 18, 19, 20, 23, 26, 31, 32, 33 *take a joke*: 1 *take back*: 2,3 *take in*: 1, 5, 7, 10, 15, 16 *take in charge*: 1 *take lying down*: 1 *take on*: 1, 2, 3, 4, 5 *take orders*: 1, 2 *take out*: 4 *take up*: 1, 2, 3, 5, 6, 7, 13
PropBank	take.01, take.14, take.18	take.01, take.19	take.01, take.06, take.09, take.21
VerbNet	performance-26.7-2	bring-11.3, steal-10.5	characterize-29.2

verb and noun. As mentioned previously, the ON data is annotated with the ON sense inventory tag only. However, the ON sense inventory provides a mapping between each coarser-grained ON sense and the WN senses that it comprises. An example of this mapping from the On sense inventory for *take* is shown in Figure 3.4. The WN sense tag can be extracted via the ON sense tag, recalling that the WN sense inventory is more fine-grained than the ON sense inventory and that one ON sense may map to multiple WN senses. We opted to extract 1) The highest frequency sense (with the lowest WN sense label number) as the WN Sense feature, and 2) The set of WN senses mapped to the ON sense. These two features are applied to both V_S and N_E.

WordNet Noun Type (Lexical File Information): For each of the noun senses in WN, the manually assigned lexical file information is given. This can be thought of as the word's supertype. In this research, twelve basic types that indicate a noun could be either eventive or stative are selected (given in Section 2.1.2). This base type is a more generalized property for each noun, and provides more common patterns for discovering previously unattested LVCs. [1]

WordNet Hyponymy: Each word sense in WN contains the hypernym derived by the knowledge structure. The hypernym of the N_E provides a more generalized feature than the WN sense itself, but more fine-grained information than the base noun type.

3.2.3 Recursive Neural Network-based Classifier

Since neural network models have been shown to perform so remarkably in various NLP tasks, we are interested in seeing if an NN model can improve LVC identification despite the limited availability of gold training data. In this preliminary work, we use a dependency-based recursive neural network (ReVNN) model as our classifier. The fundamental idea of dependency-based ReVNN is to generate a vector representation, i.e. embeddings, for each dependency tree node. Given a dependency tree as input, a tree node representation is built that is composed of different vectors, i.e., the word embeddings of the tree node itself, the embeddings of each relation label, and the vector representation of its child nodes. Further details of this model are included in Section 5.2.

The design of our ReVNN-based LVC identification is straightforward. We first acquire the dependency tree node representations which contain the word embeddings, POS, and dependency relation label information. Then, the representation embeddings of the candidate pair of (V_S, N_E) are extracted as (h^{V_S}, h^{N_E}). The concatenated vector of h^{V_S} and h^{N_E} is then submitted to a softmax layer $f(x)$ as a binary classifier. The output of $f(x)$ indicates whether the candidate pair (V_S, N_E) is an LVC relation or not. More formally, this can be expressed as:

$$f(x_i, W) = Wx_i = W(h^{V_S} \oplus h^{N_E}) \tag{3.1}$$

where $\oplus$ refers to a vector concatenation.

In the dependency-based ReVNN model, we select distributional semantics (i.e. word embeddings), POS and semantic information as features. Lexical-rich resources like WN and ON sense

[1] The initial list is provided by Christiane Fellbaum, personal communication

tags are ignored in our model since we would like to determine if a neural network model with basic features can improve tasks with only a small amount of training data. In the experimental results below, we first present the SVM results and then compare them to the RNN results.

3.3 LVC Identifier Experiments and Results

For our experiments, we used two target corpora, the BNC LVC data provided by Tu and Roth (2011) and the ON 4.99 data. The BNC data is a collection of the *verb - noun object* patterns from the BNC corpus, and is annotated as a balanced data set. Among all the BNC LVC data, 1,039 positive examples and 1,123 negative examples are generated. We randomly sample 90% of the instances for training and the rest for testing. We also experiment with the ON 4.99 data. In order to evaluate the accuracy of our model for different genres, we split our training and test sets by randomly selecting different parts of subcorpora in each genre of ON. Portions of the following six corpora are used for the test set: MSNBC broadcast conversation, CNN broadcast news, Sinorama news magazine, WSJ newswire, CallHome telephone conversation (TC), and GALE web-text (WB). In all of the ON data, 1,768 LVCs are annotated (in Table 3.2). Among these LVCs in ON, 1,588 LVCs are listed in the training data set, and 180 LVCs are in the test data set. We first present results below using the SVM classifier, and then compare this with the ReVNN model performance.

3.3.1 BNC Data

We first train and evaluate our model with the BNC data using automatic parsers produced by ClearNLP (Choi and Mccallum, 2013). Table 3.5 shows the performance of Tu and Roth (2011)'s model and our classifier on the BNC data set at each step: precision, recall, and F_1-measure. Our baseline model involves basic features only. Our *SVM-All features* model, which includes the three WN features, gains around 3 to 4 % improvement for positive and negative examples, with respect to Tu and Roth's contextual features and statistical features.

Table 3.5: Model Comparison for BNC data. *TR-C* refers to Tu & Roth's model with contextual features, while *TR-S* refers to their model with statistical features. *SVM-Basic* model is our classifier with the basic features only, while *SVM-All Features* uses all the features. The '+' refers to the system's performance in detecting LVCs, while the '-' refers to the system's performance in detecting non-LVCs. The best results for each system are shown in bold.

Model		**P**	**R**	**F1**
TR-C	+	**86.49**	84.21	85.33
	-	86.15	**88.19**	87.16
TR-S	+	86.48	85.09	86.46
	-	86.72	87.40	87.06
SVM-Basic	+	81.13	86.00	83.50
	-	88.89	84.85	86.82
SVM-All Features	+	85.32	**93.00**	**89.00**
	-	**94.31**	87.88	**90.98**

Table 3.6: Incremental Feature Contribution for ON gold trees

Feature	**P**	**R**	**F1**	**Diff**
SVM-Basic	78.09	78.09	78.09	-
+WN-Sense	80.23	**79.78**	80.00	+1.81
+WN-Type	80.68	**79.78**	80.23	+0.23
+WN-Hyper	81.61	**79.78**	**80.68**	+0.46
+Word Sense	**81.77**	78.09	79.89	-0.80

3.3.2 OntoNotes Gold Data Evaluation

We first train and evaluate our model with automatic parse trees. The overall results are lower than on the BNC test set, because the data exhibits more variety. We achieved Precision of 54.94%, Recall of 77.22% and an F1 score of 64.20%. Then we use the identical data set with Gold Standard dependency trees, to evaluate the contribution of each feature. Table 3.6 shows the performance of our system with features added incrementally. These features are compared with the baseline model. The basic feature already achieves 78.09% of F_1 score. And all three WN features contribute to the F_1 score incrementally. After all the WN features are added, our model reaches the best F_1 score of 80.68. Although the addition of the ON Word Sense feature decreases the F_1 score, it still increases the precision.

To investigate the effectiveness of each individual feature, we carried out an ablation study

Table 3.7: Individual Feature Contribution for ON gold trees

Feature	**P**	**R**	**F1**	**Diff**
SVM-Basic	78.09	78.09	78.09	-
WN-Sense	80.23	**79.78**	**80.00**	**+1.81**
WN-Type	78.53	78.09	78.31	+0.22
WN-Hyper	80.00	78.65	79.32	+1.23
Word Sense	**80.59**	76.97	78.74	+0.65

using only one feature at a time. The results in Table 3.7 shows that all the WN and ON word sense features improve the system's performance. This demonstrates that the more fine-grained features, including WN-Sense, WN-Hyper, and ON Word Sense, contribute most to precision, especially the WN-Sense feature.

Table 3.8: Individual Feature Contribution for ON gold trees with dependency node ReVNN-based model

Feature	**P**	**R**	**F1**	**Diff**
Word Embeddings	64.04	64.77	64.41	-
+ POS Embeddings	67.01	**71.59**	**69.23**	+4.82
+ Dep. Label Embeddings	**70.37**	67.05	68.67	-0.56

On the other hand, we also evaluate our preliminary dependency-based ReVNN-based LVC identification on gold standard ON dependency trees. Table 3.8 shows the performance of the identifier with features added incrementally. After the word and POS embeddings are added, the F_1 score of our ReVNN model reaches 69.23. Although these feature embeddings contribute to the F_1 score, the overall performance is far behind the SVM model.

Chapter 4

AMR-Dependency Parse Aligner

4.1 Overview

In order to obtain better alignments between AMR concepts and original word spans, an unsupervised AMR-sentence aligner is designed to improve alignments using heuristics. Since our final goal is to design an AMR parser using dependency parses, an AMR concept to dependency parse node aligner is a natural choice for such alignment. An example alignment is shown in Figure 4.1. Alignment between an AMR concept and a dependency node indicates that the meaning of the sub-graph of the concept and its child concepts correspond to the phrase governed by the head word node. The dependency node also contains additional information, e.g., lemma, syntactic label, named entity (NE), semantic role, etc. For example, the word node "Vinken" on the dependency parse side in Figure 4.1 links to the lexical concept of "Vinken" and, furthermore, links to the "*p2/name*" and "*p/person*" concepts since "Vinken" is the head of the NE "Pierre Vinken" and the head of the whole noun phrase "Pierre Vinken, 61 years old." To incorporate these different features in the aligner, we use the EM algorithm to train different feature probabilities, including rule-based features, lexical forms, relation labels, NE tags, semantic role labels, and global features, etc. Then, EM processing incorporates all the individual probabilities and estimates final alignments.

Our AMR-to-Dependency parse aligner represents one AMR as a list of Concepts $C = \langle c_1, c_2, \ldots, c_{|C|} \rangle$, and the corresponding dependency parse as a list of dependency word nodes $D = \langle d_1, d_2, \ldots, d_{|D|} \rangle$. An alignment function a is designed to produce exactly one alignment to a dependency node d_{c_j} within a single sentence. Alternatively, we can view a as a mapping

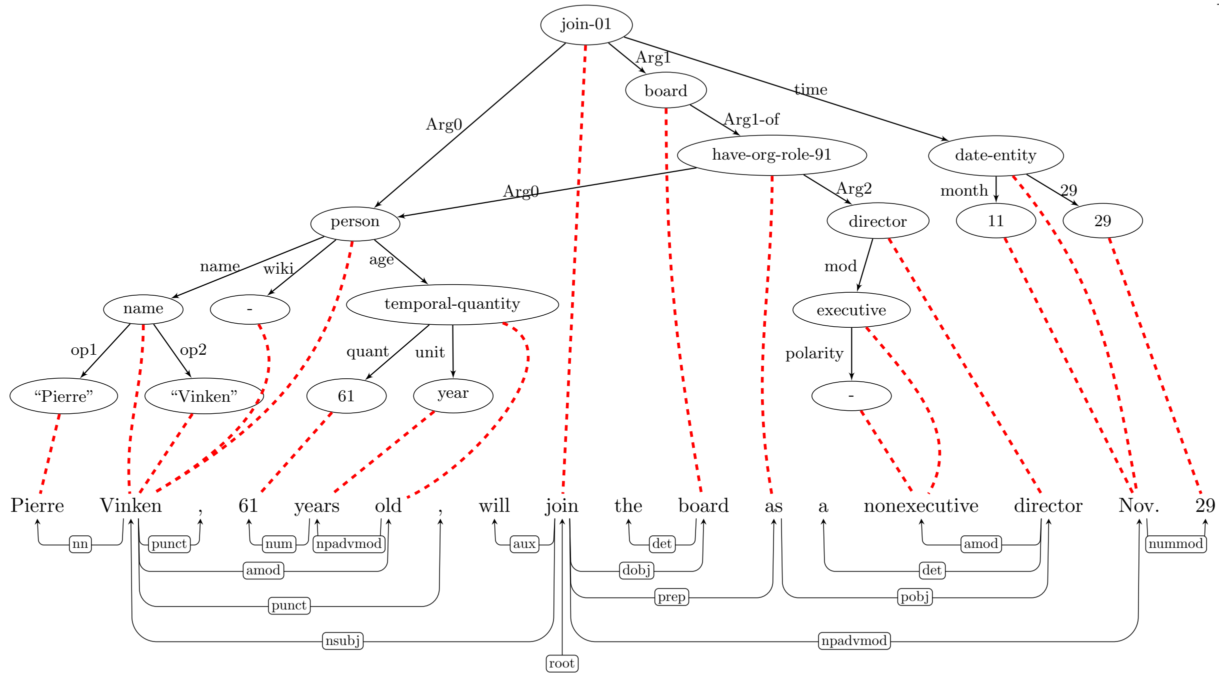


Figure 4.1: The alignment between an AMR (top) and a dependency parse (bottom) for the sentence "Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29." Red, dashed lines link dependency parse nodes and corresponding concepts.

function that accepts one input variable concept c_j and outputs a dependency node d_{c_j} with which c_j is aligned. A is the alignment set that contains all different a_l that cover possible alignments within C and D. Our model adopts an asymmetric alignment direction, where one concept maps to exactly one dependency parse node, and each dependency parse node can be aligned by zero to multiple concepts. We denote dependency node d_{c_j} linked by concept c_j as $d_{c_j} = a(c_j)$. c^p is the parent concept of concept c, while $c^{s_1}, c^{s_2}, ..., c^{s_k}$ are the k child concepts of concept c.

The following sections explain the approaches we take to design our aligner. We first discuss the features that our aligner uses (Section 4.2). Next we propose an EM-based method to integrate all the features into the model (Section 4.3). The design of our decoding tool based on dynamic-programming (DP) is introduced in Section 4.4. The results of this aligner first proposed in Chen (2015); Chen and Palmer (2017) are shown in Section 4.5. The results for AMR parsing with the use of our aligner are reported in Section 4.5.3. Error analysis is found in Section 4.5.4.

Table 4.1: Rules and distribution of basic match types

Match Type	**Portion at Concept**	**Portion at Leaf**
(1) Word	45.2%	73.4%
(2) Word (case insensitive)	-	0.9%
(3) Lemma (case insensitive)	10.8%	0.3%
(4) Partial match with word	6.1%	8.2%
(5) Partial match with lemma	0.2%	0.3%
(6) Numbers	-	3.1%
(7) Ordinal Numbers	-	2.8%
(8) Date	-	4.3%
(9) Others	37.7%	6.5%

4.2 Features

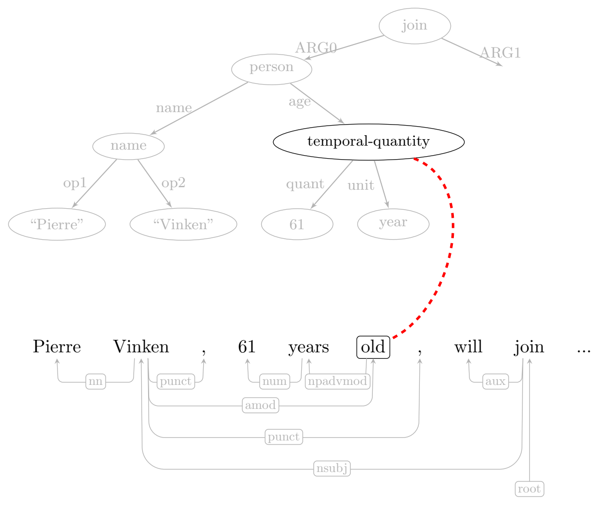
4.2.1 Basic Features

Several AMR concepts use word forms directly. For example, the concept "join-01" in Figure 4.1 aligns to the dependency node "join" naturally. Similarly, the leaf concepts usually align to identical terms in the dependency parse. In Figure 4.1, the name "Pierre Vinken" is aligned to its word forms on the dependency parse leaves. Therefore, we design a straightforward rule-based probability, P_{rule}, which catches the appearance of the surface form. $P_{rule}(c, d_c)$ is defined as the probability that the matching type for a given concept c and dependency node d_c are linked. The different types of rules, e.g., word, lemma, numbers, and data, etc., and their proportional applicabilities to both AMR concepts and leaves are listed in Table 4.1. For example, the rule "Date" type aligns concept "$\underline{11}$" with word node "November" in Figure 4.1, while "Numbers" aligns concept "$\underline{5}$" with word node "five". P_{rule} decide which match type to apply following a greedy matching strategy.

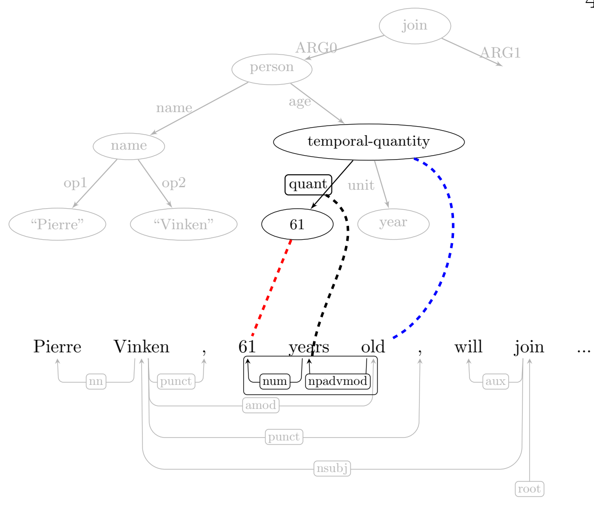
4.2.2 External Features

To capture alignments for concepts which do not match any of the above rules, we design the following four external feature probabilities:

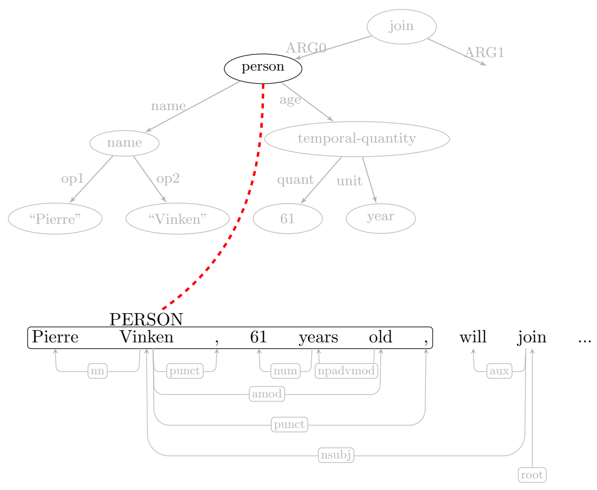
Lemma Probability $P_{Lemma}(c, d_c) = P(c|\,Word(d_c))$



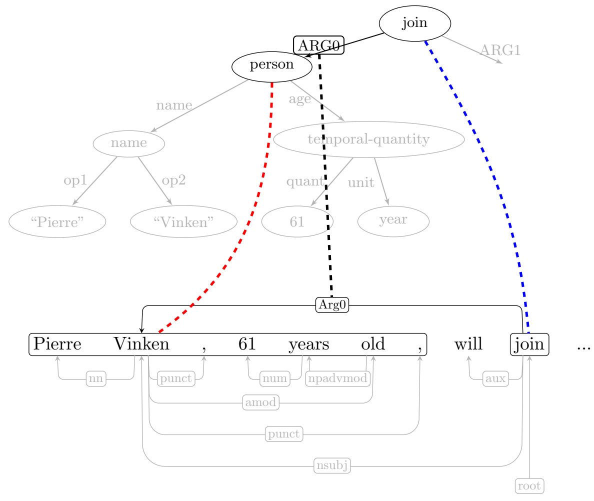
(a) Lemma Feature Probability
$P_{lemma}(c = \textit{temporal-quantity}, d_c = \underline{old})$



(b) Relation Feature Probability
$P_{rel}(c = \textit{61}, d_c = \underline{61}, d_{c^p} = \underline{old})$



(c) Named Entity Probability
$P_{NE}(c = \textit{person}, d_c = \underline{Vinken})$



(d) Semantic Roles Feature example
$P_{SR}(c = \textit{person}, d_c = \underline{Vinken}, d_{c^p} = \underline{join})$

Figure 4.2: Example of features that are used in our aligner. Red dashed lines link AMR concepts (top) and corresponding dependency parse nodes (bottom), while blue dashed lines link the parent AMR concepts and their corresponding dependency parse nodes.

The lemma probability represents the likelihood that concept c aligns to a dependency word d_c. For example, in Figure 4.2a, the concept c ="temporal-quantity" is highly likely to align to the word node d_c = "old" since "old" is usually the head word of a phrase expressing age ("61 years old" here). Also, *have-org-role-91* can align to the word node "director" since "director" appears quite often with *have-org-role-91* (defined as roles in organizations.) Besides, some special leaf concepts, like "*:polarity -*" (negative), and "*:mode expressive*" (used to mark exclamations), also rely on this feature rather than the basic rules.

Relation Probability $P_{rel}(c, d_c, d_{c^p}) = P(AMRLabel(c)|Path(d_c, d_{c^p}))$

Relation probability is the conditional probability of the AMR relation label of c given the parse tree path between d_c and d_{c^p}, where d_c and d_{c^p} represent the dependency nodes that are aligned by c and c^p, respectively. Parse tree path is the concatenation of all dependency tree and direction labels through the tree path between d_c and d_{c^p}. For example, the relation probability of $c = \mathit{61}$, $d_c = \underline{61}$, and $d_{c^p} = \underline{old}$ in Figure 4.2b is $P(quant|\underline{npadvmod \downarrow num \downarrow})$. A parse tree path is a useful feature for extracting relations between any two tree nodes, e.g., Semantic Role Labeling (SRL) (Gildea and Jurafsky, 2002) and relation extraction (Bunescu and Mooney, 2005; Kambhatla, 2004; Xu et al., 2015), so we add the relation probability with path feature to our model.

Named Entity Probability $P_{NE}(c, d_c) = P(c|NamedEntity(d_c))$

The named entity probability is the probability of the concept c conditioned on different NE types (e.g., PERSON, DATE, ORGANIZATION, etc.). $NamedEntity(d)$ indicates the named entity type of the phrase with d as the head word. For example, after NER tagging, the label assigned to "PERSON" is the dependency parse tree node "Vinken". So the NE probability of $P_{NE}(c = person, d_c = \underline{Vinken})$ in Figure 4.2c is $P(person \mid \underline{\text{PERSON}})$. As AMR contains a large amount of named entity information, we assume that a feature based on an external named entity module should improve the alignment accuracy.

Semantic Role Probability $P_{SR}(c, d_c, d_{c^p}) = P(AMRLabel(c)|\ SemanticRoles(d_{c^p}, d_c))$

The semantic role probability is the conditional probability of the AMR relation label of c, given the semantic role d_c if d_{c^p} is a predicate and d_c is d^p's argument. If a predicate-argument structure does not exist between d_{c^p} and d_c, the semantic role probability is omitted. For example, in Figure 4.2d, the semantic role probability of $P_{SR}(c = person, d_c = \underline{Vinken}, d_{c^p} = \underline{join})$ is equal to $P(ARG0|\underline{Arg0})$. Since AMR depends heavily on predicate-argument relations, external predicate-argument information from an external SRL system should enhance the overall alignment.

The above four feature probabilities are learned by the EM algorithm (Section 4.3).

4.2.3 Global Feature

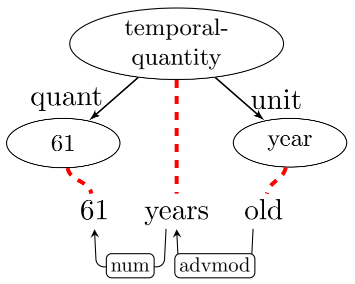


Figure 4.3: An example of incorrect alignment.The overlapping ratio R_{cc} is calculated as follows: let $c = \textit{temporal-quantity}$
$W(c) = \{\underline{61}, \underline{year}\}, W_{child}(c) = \{\underline{61}\} \cup \{\underline{61}, \underline{year}, \underline{old}\} = \{\underline{61}, \underline{year}, \underline{old}\}$
$W_{child}(c) \cap W(c) = \{\underline{61}, \underline{year}\}, penalty(c) = exp(-|\{\underline{old}\}|) = 0.37$
$R_{cc}(c) = (\frac{2}{2}) \times pen(c) = 0.37$

The above basic and external features capture local alignment information. However, to make sure that a concept is aligned to the correct phrase head word representing the same sub-meaning, a global feature is needed to calculate coverage. The design of our concept coverage feature is as follows:

Overlapping Ratio $R_{CC}(c^p)$: The overlapping ratio of the child concept aligned phrases to their parent concept aligned phrases plus the non-covered penalty. The ratio is defined as:

$$R_{CC}(c) = \frac{|W_{child}(c) \cap W(c)|}{|W(c)|} \times penalty(c)$$

$$W(c) = d_c; W_{child}(c) = \bigcup_{c^{s_i} \in child(c)} d_{c^{s_i}}$$

$$penalty(c) = exp(-|W_{child}(c) \setminus (W_{child}(c) \cap W(c))|)$$

where W refers to the set of words that the aligned dependency word node contains. The first term of R_{CC} ensures the child concepts contain the largest possible subspan of the parent concept span. The non-covered *penalty* term is to prevent a child concept from aligning to a word node that contains a larger word span than the child's parent concept. The *penalty* term will increase exponentially if child concepts align to a larger word span. The back slash term "\' refers to set subtraction. Figure 4.3 shows an example of an incorrect alignment where the concept "*temporal-quantity*" aligns to "year" and the concept "*year*" aligns to "old". The overlapping ratio of this alignment is 0.37 since it suffers a penalty. We compare it with the correct alignment in Figure 4.2b, the overlapping ratio of this alignment is 0.67, which is much higher than the incorrect one.

4.3 Training with EM Algorithm

The objective function of our AMR-to-Dependency Parse aligner is implement as follows. Since our long term goal is to design a dependency parse to AMR parser, we define the objective function L_θ as the probability that dependency parses transfer to AMR graphs for the AMR-to-Dependency Parse aligner:

$$\theta = \operatorname{argmax} L_\theta(\text{AMR}|\text{DEP}), \tag{4.1}$$

$$L_\theta(\text{AMR}|\text{DEP}) = \prod_{(C,D,A)\in\mathbb{S}} P(C|D) = \prod_{(C,D,A)\in\mathbb{S}} \sum_{a\in A} P(C,a|D) \tag{4.2}$$

$$P(C,a|D) = \prod_{j=1}^{|C|} P(c_j|d_{c_j} = a(c_j), d_{c_j^p} = a(c_j^p)) \tag{4.3}$$

where $\theta = (P_{lemma}, P_{rel}, P_{NE}, P_{SR})$ is the set of feature probabilities (parameters) we want to estimate, alignment set A is the latent variable we want to observe, and $\mathbb{S}$ is the training samples that contain a set of tuples (C, D, A). In equation 4.3, the probability that dependency tree D translates to AMR C with one alignment combination a is equal to the product of all probabilities that concept c_j in C aligns to dependency node d_{c_j} and c_j^p (the parent concept of c_j) aligns to dependency node $d_{c_j^p}$. The Expectation-Maximization (EM) (Dempster et al., 1977) algorithm is used to estimate the feature probabilities that maximize our objective function.

4.3.1 Expectation-Step

The E-Step estimates all the different alignment probabilities of an input AMR and dependency parse pair by giving the product of feature probabilities. The alignment probability can be calculated using:

$$P(a|C,D) = \prod_{j=1}^{|C|} \frac{P(c_j|d_{c_j} = a(c_j), d_{c_j^p} = a(c_j^p))}{\sum_{l=1}^{|D|} \sum_{i=1}^{|D|} P(c_j|d_i, d_l)} \tag{4.4}$$

$$P(c_j|d_i, d_l)$$
$$= P_{rule}(c_j, d_i) \times P_{lemma}(c_j, d_i) \times P_{rel}(c_j, d_i, d_l) \times P_{NE}(c_j, d_i) \times P_{SR}(c_j, d_i, d_l) \tag{4.5}$$

The alignment probability is equal to the product of all tuple (c, d_c, d_{c^p})'s aligning probabilities. P_{rule} is obtained by a simple calculation from the development set, while P_{lemma}, P_{rel}, P_{NE}, and P_{SR} are initialized uniformly before the first round of the E-step. These feature probabilities will be updated during the M-step.

4.3.2 Maximization-Step

In the M-step, feature probabilities are reestimated by collecting the count of all AMR-dependency parse pairs. The feature probabilities are estimated with

$$cnt_{lemma}(c|Word(d_c); C, D) = \sum_{a \in A} \frac{P(c|d_c, d_{c_p})}{\sum_{i=0}^{|D|} \sum_{l=0}^{|D|} P(c|d_i, d_l)} \tag{4.6}$$

$$cnt_{rel}(AMRLabel(c)|path(d_c, d_{c^p}); C, D) = \sum_{a \in A} \frac{P(c|d_c, d_{c^p})}{\sum_{i=0}^{|D|} \sum_{l=0}^{|D|} P(c|d_i, d_l)} \tag{4.7}$$

$$cnt_{NE}(c|NamedEntity(d); C, D) = \sum_{a \in A} \frac{P(c|d_c, d_{c_p})}{\sum_{i=0}^{|D|} \sum_{l=0}^{|D|} P(c|d_i, d_l)} \tag{4.8}$$

$$cnt_{SR}(AMRLabel(c)|SemanticRole(d_c, d_{c^p}); C, D) = \sum_{a \in A} \frac{P(c|d_c, d_{c^p})}{\sum_{i=0}^{|D|} \sum_{l=0}^{|D|} P(c|d_i, d_l)} \tag{4.9}$$

where $cnt_{lemma}, cnt_{rel}, cnt_{NE},$ and cnt_{SR} are the normalized counts that are collected from the accumulating probability of all possible alignments from the E-step. After we collect all counts for different features, the four feature probabilities, $P_{lemma}, P_{rel}, P_{NE},$ and P_{SR}, are updated with their feature counts.

$$P_{lemma}(c,d) \leftarrow \sum_{C \in AMR, D \in DEP} \frac{cnt_{lemma}(c|Word(d)); C, D)}{\sum_c cnt_{lemma}(c|Word(d); C, D)} \tag{4.10}$$

$$P_{rel}(c,d,d^p) \leftarrow \sum_{C \in AMR, D \in DEP} \frac{cnt_{rel}(AMRLabel(c)|Path(d,d^p); C, D))}{\sum_{amrLabel} cnt_{rel}(amrLabel|Path(d,d^p); C, D)} \tag{4.11}$$

$$P_{NE}(c,d) \leftarrow \sum_{C \in AMR, D \in DEP} \frac{cnt_{NE}(c|NamedEntity(d)); C, D)}{\sum_c cnt_{NE}(c|NamedEntity(d); C, D)} \tag{4.12}$$

$$P_{SR}(c,d,d^p) \leftarrow \sum_{C \in AMR, D \in DEP} \frac{cnt_{SR}(AMRLabel(c)|SemanticRole(d^p,d); C, D))}{\sum_{amrLabel} cnt_{SR}(amrLabel|SemanticRole(d^p,d); C, D)} \tag{4.13}$$

After this, we apply the newer feature probabilities to recalculate alignment probabilities in the E-step again. Iteration of the E- and M-steps continues until convergence or certain criteria are met.

4.4 Decoding

At decoding time, we want to find the most likely alignment a for the given $\langle C, D\rangle$. Applying Equations (4.4) and (4.5), we define the search for alignments as follows:

$$\underset{a}{\operatorname{argmax}} P(a|C,D) = \underset{a}{\operatorname{argmax}} \prod_{j=1}^{|C|} R_{CC}(c_j) * P(c_j|d_{c_j} = a(c_j), d_{c_j^p} = a(c_j^p)) \tag{4.14}$$

This decoding problem finds the alignment a that maximizes the likelihood defined in Equation 4.2. The overlapping ratio (R_{CC}) is introduced as part of the likelihood function to ensure that a parent concept covers a wider word span range than its child concepts. A beam search algorithm is designed to extract the target alignment without exhaustively searching all of the candidate alignments (which has a complexity of $O(|D|^{|C|})$.) The beam search starts from the leaf concepts and then walks through concepts after their child concepts have been traversed. When we go through concept c_j, we need to consider all the following likelihoods: 1) the accumulated likelihood for aligning to any dependency word node d_{c_j} from all the child concepts of c_j, and 2) the product of P_{lemma}, P_{NE}, P_{rel}, P_{SR}, and R_{CC} for c_j. Instead of being used during training, R_{CC} is only applied during decoding time. The probabilities are obtained simply from the product of all the above likelihood. We keep the top-$|b|$ alignment probabilities and their aligned dependency node d_{c_j} for each c_j until we reach the root concept, where $|b|$ is the beam size. Finally we can trace back and find the most likely alignment. The running time for the beam search algorithm is $O(|b| * |C| * |D|^2)$.

4.5 Experiments and Results

4.5.1 Experimental Data

The LDC DEFT Phase 2 AMR Annotation Release 1.0 consists of AMRs with English sentence pairs. To match an AMR with its corresponding dependency parse, we select the sentences which appear in OntoNotes (ON) 5.0[1] . The ON data contains TreeBank, PropBank, and NE annotations. For evaluation purposes, we manually align the AMR concepts and dependency word nodes for the development and test sets. A total of 1,000 sentences with gold alignments is available. Table 4.2 presents the statistics for the experimental data. For the data without gold standard parses, the DEFT AMR training data, we need an automatic dependency parser. We retrained the ClearNLP model with ON 5.0 after first removing the sentences that also appear in the DEFT AMR corpus. We then run it on the training data. The manually aligned data and the source code of the web browser based alignment tool is available on our website: `https://verbs.colorado.edu/~wech5560/amrAlignment` and GitHub: `https://github.com/weitechen/`.

4.5.2 Experimental Results

We run EM for 50 iterations and ensure the EM model converges. Afterwards, we use our decoding algorithm to find the alignments that maximize the likelihood. The test set data is used to evaluate performance.

[1] LDC OntoNotes Release 5.0, Release date: October 16, 2013 https://catalog.ldc.upenn.edu/LDC2013T19

Table 4.2: The data split of the LDC DEFT AMR corpus. *Gold Dep.* refers to the sentences also appearing in OntoNotes 5.0 with gold annotations, while *Auto Dep.* refers to all sentences in DEFT AMR corpus with dependency parses, named entities, and semantic roles generated by ClearNLP. Number of tokens, named entities, and arguments(Arg.) in each data set are also presented

		Sent.	**Token**
Train	Gold Dep.	8,276	176,422
	Auto Dep.	36,521	649,219
Dev.		500	8,695
Test		500	8,786

We first evaluate the performance of our system with external features added incrementally. Table 4.3 indicates the results. By running with the "Gold Dep." data, the only feature that improves significantly over the baseline (rule-based and lexicon features only) is the semantic role feature. The named entity feature actually hurts performance. On the other hand, all the features contribute to the F-Score incrementally for "Auto Dep.". Again, the semantic role feature results in the most positive impact compared to other features with a significant improvement over the baseline.

As we compare the F_1 score on training with "Auto Dep." and "Gold Dep." data set, training with "Auto Dep." outperforms training with "Gold Dep." data in all different feature combinations. We believe there are two reasons for this. First, the "Auto Dep." data contains richer information than the "Gold Dep." data: "Auto Dep." has double the sentence size of "Gold Dep." and proportionally more named entity labels. Second, the automatic dependency parses do not hurt the performance of our aligner very much. We believe that our unsupervised alignment model works better with more data, even without access to gold standard dependency parses.

We then compare our latest aligner (Chen and Palmer, 2017) with three other aligners: JAMR, our basic version of aligner (Chen, 2015), and ISI (Pourdamghani et al., 2014). To make them fit our evaluated data, we design a heuristic method to force every un-aligned concept (e.g., NE and ":polarity -" concepts) to align to a dependency word node according to rule-based and global features (see Section 4.2). The alignments from concept relation to word span (applied in ISI) are discarded in our task. The results of the experiment are shown in Table 4.4. Our latest

Table 4.3: Incremental Feature Contributions for different features: L: lemma; R: relation; N: NE; S: semantic role.

Data	Feature	P	R	F-Score
Gold Dep.	L	84.0	85.0	84.5
	L + S	85.2	86.3	**85.7**
	L + S + R	82.8	83.8	83.3
	L + S + R + N	80.9	81.9	81.4
Auto Dep.	L	84.9	85.4	85.1
	L + S	85.7	87.4	86.5
	L + S + R	85.8	87.7	86.7
	L + S + R + N	86.3	88.0	**87.1**

aligner achieves the best F_1 score, as it should, since it is designed to align AMRs to dependency parses, as was the (Chen, 2015) aligner. Our aligner performs better than Chen (2015) by around 28% of F_1 score. We can conclude that the addition of rule-based feature, global features, and beam-search in decoding time helps the alignment task substantially.

4.5.3 Apply to AMR Parsing

To evaluate how alignment can enhance AMR parsing, we compare the parsing performance of the CAMR parser with different alignments produced by JAMR, ISI, and our aligner. To make the alignments fit the CAMR parser, we convert our alignments to the original JAMR alignment format, word span to AMR concept. We get rid of the ":wiki" tag, which links the named entity to its Wikipedia page, to simplify the parsing task since we consider the Wikification task (Mihalcea and Csomai, 2007) to be inherently different than the AMR parsing task. Smatch v2.0.2 is used to evaluate AMR parsing performance (Cai and Knight, 2012). The evaluation script is obtained from the SemEval 2016 Task 8 website.[2]

A comparison of parsing results is given in Table 4.5. We first train the parser with "Gold Dep." Standard dependency parses and alignments from the different aligners. Results show that our aligner improves by a 2% F_1 score over the two other aligners. Then we train the AMR Parser system with "Auto Dep." data set. The dependency parses attached with semantic roles and

[2] `http://alt.qcri.org/semeval2016/task8/index.php?id=data-and-tools`

Table 4.4: Results of different alignment models

Data	**Aligner**	**P**	**R**	**F-Score**
Gold Dep.	Chen 2015	61.1	53.4	57.0
	JAMR	78.5	62.8	69.8
	ISI	78.6	71.4	74.9
	Ours	**85.2**	**86.3**	**85.7**
Auto Dep.	Chen 2015	62.4	55.5	58.7
	JAMR	80.2	65.9	72.4
	ISI	80.4	74.9	77.6
	Ours	**86.3**	**88.0**	**87.1**

named entities generated by ClearNLP (Choi and Mccallum, 2013) are also provided to CAMR as training data. CAMR uses dependency parsing results from the Stanford dependency parser (Klein and Manning, 2003) by default. Our aligner still achieves slightly better performance than the other two. Modifying the AMR parser to take advantage of parse node-concept alignments could potentially result in greater improvement, since CAMR takes the input alignments as word span to AMR concepts.

4.5.4 Error Analysis

To further understand the advantages and the disadvantages of our model, we go through all incorrect alignments and manually categorize 40% of them into different error types, with their proportion:

Automatic Parsing Errors - 3.8%: ClearNLP has a 92.96% unlabeled attachment score on the Penn English Treebank evaluation set (Marcus et al., 1994), Section 23, for dependency parsing. Therefore, when training our aligner on the "Auto Dep." data set with dependency parses, named entities, and semantic roles generated by ClearNLP, incorrect parses occasionally show up. Since NE and semantic roles are attached to dependency parses, incorrect dependency parses cause additional NE and semantic roles alignment errors, on top of the dependency parse alignment errors.

Long Distance Dependencies - 14.2%: Long sentences with long distance dependencies tend to introduce errors into NLP parsing tasks. Experimental results show that our model runs into

Table 4.5: Using alignments with Brandeis AMR Parser. The *Gold Dep.* corpus uses the gold dependency parses from ON 5.0, while the *Auto Dep.* corpus uses the whole DEFT AMR data (twice the size) with dependency parses generated by ClearNLP.

Data	**Aligner**	**P**	**R**	F_1	**Diff**
Gold Dep.	JAMR	62.2	61.0	61.1	+5.3
	ISI	65.3	63.9	64.5	+1.9
	Our	**68.6**	**64.2**	**66.4**	
Auto Dep.	JAMR	64.2	63.0	63.1	+3.6
	ISI	66.1	65.1	65.6	+1.1
	Our	**68.1**	**64.7**	**66.7**	

trouble when nearby concepts align to dependency nodes which are far from each other. Coreference is an example that is highly likely to align to long distance dependencies, and our model can not deal with it well.

Duplicate Words - 17.4%: When two identical concepts align to different word nodes, our model is confused by duplicate words. In Figure 4.4, there are two "first"s in the sentence. One refers to "first 6 rounds", and the other refers to "first position". However, our model faultily aligns both *ordinal-entity* concepts to the same "first" word node. Our model did not distinguish these two ordinal-entities since the lexicon and named entity tags of the two "first"s are identical.

Meaning Coverage Errors - 40.4%: We define a good alignment as a concept that aligns to the correct phrase head word representing the same sub-meaning. So instead of aligning to a concept's word and lemma, sometimes a concept aligns to its parent node (head word). However, the lexicon features dominate the alignment probability in our EM calculation, causing our model to tend to align a concept with its word form instead of its head word. For example, English light verb constructions (LVCs), e.g., *take a bath*, are thought to consist of a semantically general verb and a noun that denotes an event or state. AMR representation always drops the light verb, selecting the eventive noun as the concept. Our model sometimes aligns this eventive noun concept to its nominal word node, which is incorrect as the light verb in the dependency parse covers the same sub-meaning and should for that reason be aligned instead.

```
(a / and
  :op1 (o / occupy-01
    :ARG0 (p3 / person :name (n / name :op1 "Mingxia" :op2 "Fu"))
    :ARG1 (p4 / position
      :ord (o3 / ordinal-entity :value 1))
  :op2 (o2 / occupy-01
    :ARG0 (p / person :name (n2 / name :op1 "Bin" :op2 "Chi"))
    :ARG1 (p2 / position :ord (o4 / ordinal-entity :value 3))
  :mod (r / respective)
  :time (r3 / round-05 :quant 6 :ARG1 (c / compete-01)
    :ord (o5 / ordinal-entity :value 1)))
```

Figure 4.4: The AMR annotation of the sentence "In the first 6 rounds of competition, Mingxia Fu and Bin Chi are occupying the first and third positions respectively"

Chapter 5

AMR Parser Framework

5.1 Overview

We address the main components of our AMR parser in this chapter. The bottom part of Figure 1.3 provides an overview of our parser. We first propose a frame identifier which recognizes and labels the frame of a candidate predicate in Section 5.3.1. Then we divide our parser into two sub-models: the concept identifier and the relation identifier. This design is different from other transition-based AMR parsers which merge a concept identifier and a relation identifier into a single task. The concept identifier receives a dependency tree node as input and generates a series of candidate concepts. The proposed concept identifier is described in Section 5.3.4. Our relation identifier, a transition-based parser which decides each parsing transition (action) based on the current status is described in Section 5.3.5. We present the decoder and post-processing in Section 5.3.6. Lastly, to show how we inject syntactic knowledge into our model, we introduce a dependency-based ReVNN as an essential feature in all the above sub-models. We begin with a brief description of this feature is in Section 5.2.

5.2 Dependency-Based Recursive Neural Network

A dependency parse tree, a model of syntactic knowledge including head words, dependency labels, and NEs (from an external system), is a prominent linguistic structure in our model. It is obvious that the design of the sub-models of our AMR parser, including frame identification, concept identification, and relation identification, all rely heavily on dependency parse trees. Our models reuse basic features like word forms, lemmas, POS tags, and dependency relations repeatedly. Meanwhile, since the NN-based models we use all acquire vectors as inputs, we need to convert all features into vector representation form, e.g., word embeddings. With these considerations in mind, we design a general dependency node vector representation (embedding) to capture the meaning of each dependency node and all its children. The goal of producing such a dependency node embedding is to use this vector as a helpful feature for our sub-models of an AMR parser which takes dependency trees as input. This feature is expected to benefit other NLP applications.

We define our Dependency-Based Recursive Neural Network (ReVNN) here. Given a dependency tree D, our model generates the vector representation of dependency tree node d as e_d. To generate e_d, we compose two types of feature vectors. The first type is a basic feature vector (e_d^{basic}), which is a concatenation of basic feature vectors from the dependency tree node itself. The basic features include the word embedding (e_d^w), POS embedding (e_d^{POS}), dependency relation embedding (e_d^r), etc. The second type is a feature vector accumulated from the vector representations of its children with the RNN-based model (v_d), which represents information from the child nodes. To facilitate information accumulation, we use two LSTM models, $LSTM^L$ and $LSTM^R$, to record and process each child vector representation ($e_{child_d^i}$), where $child_d^i$ represents the i-th child node of d. The features that we use to compose $e_{child_d^i}$ are identical to the features of e_d^{basic}. Both LSTM models process the child node vectors. The only difference is that $LSTM^L$ processes child nodes before d, while $LSTM^R$ processes child nodes after d. The LSTMs receive $child_d^i$ as inputs according to the word order of $child_d^i$ from far to near. The reason that we design two LSTM models rather than just one is that the behaviors of child nodes before d and after d varies. The outputs of these two LSTM models, v_d^L and v_d^R, are then concatenated into v_d. For the tree node that does not contain any left or right child nodes, i.e., a leaf node, we use an empty vector (v^{empty}) to replace v_d^L or v_d^R.

The two types of feature representations, e_d^{basic} and v_d, are then concatenated and passed to a feedforward layer with a sigmoid activation function. This layer maps the concatenation feature vector to the final dense vector representation, e_d. The idea of the dependency-based ReVNN can

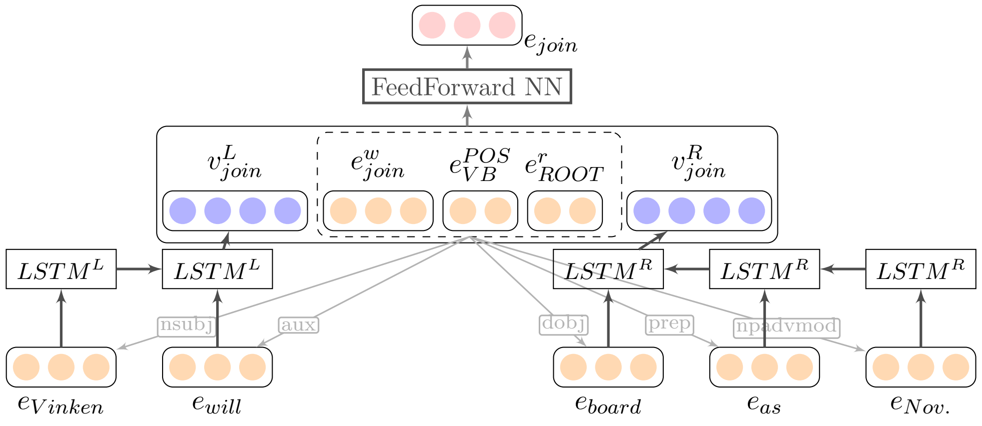


Figure 5.1: Overall framework of Learning Dependency-Based ReVNN

be expressed formally as:

$$
\begin{aligned}
e_d &= FeedForward(e_d^{basic} \oplus v_d) \\
e_d^{basic} &= e_d^w \oplus e_d^{POS} \oplus e_d^r;\ v_d = v_d^L \oplus v_d^R \\
v_d^L &= h^L_{child_d^i};\ v_d^R = h^R_{child_d^j} \\
h^L_{child_d^i} &= LSTM^L(e_{child_d^i}, h_{child_d^{i-1}});\ h^R_{child_d^j} = LSTM^R(e_{child_d^j}, h_{child_d^{j+1}})
\end{aligned} \tag{5.1}
$$

where $FeedForward$ is the feedforward layer with sigmoid activation, h_i^L is the hidden state of $LSTM^L$ after processing e_i, $child_d^i$ is the i-th child node of d which is before the closest word node to d, while $child_d^j$ is the j-th child node of d which is after d and the closest word node to d, and $\oplus$ indicates vector concatenation.

The hierarchy of our model is presented in Figure 5.1. In this figure, colored circles indicate vector values; rounded squares indicate the concatenation of different vectors; the squares are the NN model itself. Our model creates the feature vector of dependency node "join" (e_{join}) for the same dependency parse in Figure 4.1. "Vinken" and "will" are the two direct child nodes that appear before "join", while "board", "as", and "Nov." are the three direct child nodes that appear after "join". Therefore, the vector representations of "Vinken" and "will" are processed by $LSTM^L$, while the vector representation of "Nov.", "as", and "board" are processed by $LSTM^R$, respectively. On top of the figure, the concatenated vector is processed by a feedforward layer, then the feature representation e_{join} is created.

To train the parameters in our model, i.e., the feedforward network and LSTM, we need a loss function to help us update the parameters with stochastic gradient descent during backpropagation. For this reason, the ReVNN model is covered by other parent models, including our frame, concept, and action identifiers, which are all multi-label classifiers. The parent model defines its own loss function $J(\theta)$ for training, where θ refers to all trainable module parameters. In the forward-propagation step, our ReVNN model uses Equation 5.1 to generate the vector representation for all dependency tree nodes. The parent model uses these vector representations as input features. In backpropagation, the parent model returns the gradient descent of the vector representation in respect to its loss function $J(\theta)$. The vector e_d is then updated by the gradient descent term with a learning rate α as

$$e_d \leftarrow e_d - \alpha \frac{\partial}{\partial e_d} J(\theta) \tag{5.2}$$

Applying the chain rule, all parameters in the set $\theta_{ReVNN} = \{\theta_{LSTM^L}, \theta_{LSTM^R}, \theta_{FeedForward}\}$ can be estimated in the backpropagation step as well. The LSTM parameter θ_{LSTM} includes the forget gate parameters $(W^{(f)}, b^{(f)})$, the input gate parameters $(W^{(i)}, b^{(i)})$, and the output gate parameters $(W^{(o)}, b^{(o)})$. Additional details of LSTM implementation with definitions of these parameters can be found in (Hochreiter and Schmidhuber, 1997). Also, the feedforward parameters $\theta_{FeedForward}$ include a weight matrix $W^{(ff)} \in \mathbb{R}^{(|v_d|+|e_d^{basic}|)\times|e_d|}$ and a bias term $b^{(ff)} \in \mathbb{R}^{|e_d|}$. These parameters in θ_{ReVNN} are updated repeatedly with numerous epochs of training.

5.3 Parser

In this section, we will briefly describe the essential parts of our AMR parser. First, we will design a frame identifier to label PropBank frame IDs for verbal and nominal predicates (Section 5.3.1). The next step is to develop the concept identifier, which predicts a list of candidate concepts for each dependency tree node (Section 5.3.4). Once we have these concept candidates with IDs, we will be ready to train our transition-based relation identifier (Section 5.3.5). The transition-based parser predicts the action that indicates the conversion of a dependency tree to its AMR graph. The conversion includes the changing of dependency parse structures and the prediction of the relations between concepts. The transition-based parser predicts the action (transition) according to the status of parsing states. An action extractor which selects the oracle actions in training time (Section 5.3.5.2) is needed to explore gold training data. By giving the oracle actions as inputs, we then train our transition-based parser with several NN components and their corresponding features (Section 5.3.5.3). After our parser converts a dependency tree to a list of AMR root concepts, a series of post-processing steps is taken to form a complete AMR graph (Section 5.3.6).

5.3.1 Frame Identifier

A Frame Identifier (FI) is similar to a traditional word sense disambiguation task. Given a target word w_p and its surrounding words, an FI automatically labels the frame ID of w_p according to its corresponding frame file. For example, given the entire "Pierre Vinken" sentence, an FI marks "join" as the predicate and selects the frame ID "join.01" which expresses the meaning of "attach, connect things together." FI is a preliminary step for our AMR parsing task. As we mentioned in Section 1.2.3, AMR uses OntoNotes predicates as core role concepts. Also, AMR depends heavily on predicate-arguments from PropBank and labels edges as PropBank core arguments. By combining a separated FI and additional parts of concept identification, e.g., named entity recognition and wikification, the AMR parser enhances its overall performance. For this reason, we treat the FI task as a separate sub-system within our parser.

Although light verb constructions potentially play an important role in the semantic parsing task, they have not been considered as an important characteristic in the frame identification task. An LVC detector can conceivably improve the performance of AMR parsing and the SRL task. For concept identification in AMR, the eventive noun in an LVC, rather than the accompanying light verb, is always picked as the real concept.

In this section, we will focus on designing an FI to fit the requirements of the AMR parser.

This identifier is based on our dependency-based ReVNN network combining various linguistic features (Section 5.3.2). In the second part, the LVC detector which we introduced in Chapter 3 is introduced to our pipeline. The resulting predicates with frame ID are then provided to our concept identifier in the next step.

5.3.2 Dependency-Based ReVNN Frame Identifier

Our frame identifier is a classifier which takes our dependency-based ReVNN network as its underlying structure. Here, PropBank is the Frame sembank we select. The first step is to indicate the candidate word that we want to identify. We select verbs and nouns that appear in the PropBank frame files as candidate words. The next step is to decide the frame ID for the given words. Here we use the dependency-based ReVNN network to execute our classifier. The design of our identifier is shown in Figure 5.2. This framework is identical to the ReVNN model introduced in Section 5.2. We are assuming we want to disambiguate the frame ID of word d (*join* in Figure 5.2.) Given the dependency parse tree of the input sentence, k-direct child nodes of d are extracted. We use three basic linguistic features to represent each child node, including word form, part-of-speech, and a relation label. As we follow the same workflow as our dependency tree ReVNN, the vector representation of the candidate tree node d is then generated. In the FI task, the next step is to submit the vector representation of d (e_{join} in Figure 5.2), to another feedforward layer with a hyperbolic tangent (tanh) function as activation, which generates the representation vector, z_d. In the last step, z_d is submitted to a softmax layer to train a multi-class classifier. The softmax classifier outputs the frame ID of word d.

Generally speaking, a softmax layer generates a list of K values where the i-th element represents the probability that the input instance belongs to class-i and K is the number of classes we want to classify. Thus, our FI can be formally described as:

$$P(\bar{y} = fi|d) = \text{softmax}(z_d) = \frac{\exp(z_d^{fi})}{\sum_{j=1}^{|FRAME|} \exp(z_d^j)}$$

$$z_d = \tanh(e_d^T W^{FI} + b^{FI})$$

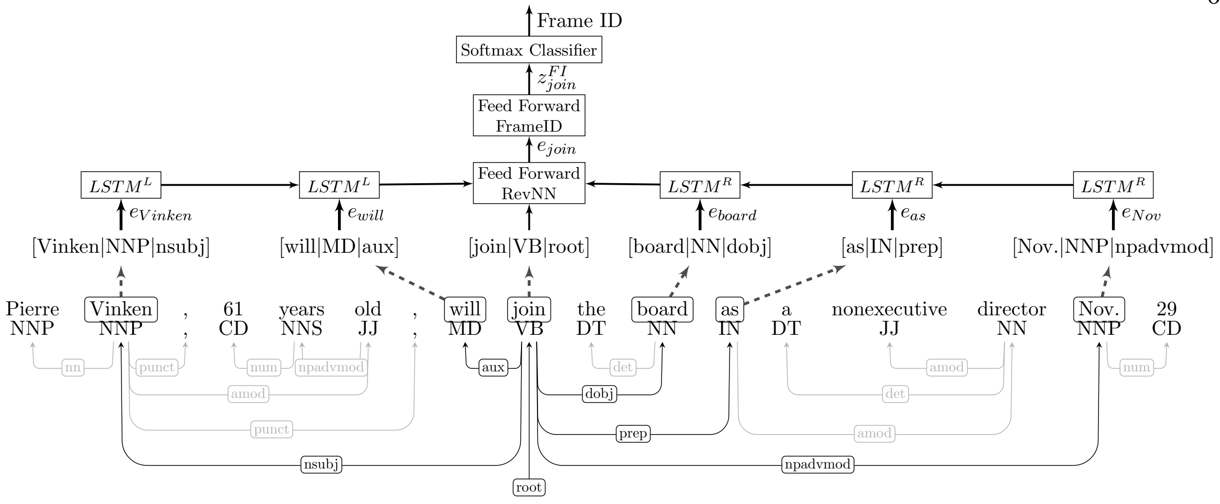


Figure 5.2: Dependency-based ReVNN Frame Identifier Framework. The square blocks indicate NN units, with "[ ]" denoting vector combinations.

where $|FRAME|$ is the total number of frames in PropBank, $W^{FI} \in \mathbb{R}^{dim(e_d)\times|FRAME|}$ is the weighted matrix in $FF_{FrameID}$ layer, b^{FI} is the bias term of $FF_{FrameID}$, $z_d \in \mathbb{R}^{|FRAME|}$ is the output vector of $FF_{FrameID}$ layer. The summation term in the softmax function ensures that the total probabilities sum to 1. The softmax layer uses multinomial logistic regression to calculate resulting probabilities. Given gold standard frame IDs during training time, the loss function of our classifier is defined as:

$$J_{FI}(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{fi=1}^{|FRAME|} 1\{y_{(i)} = fi\} \log \frac{\exp(z^{fi}_{(i)})}{\sum_{j=1}^{|FRAME|}\exp(z^{j}_{(i)})}] \tag{5.3}$$

where $1\{k\}$ is the indicator function which generates a vector whose k-th element is 1 with the rest of the vector being 0. We can then use backpropagation and stochastic gradient descent on J_{FI} to update w. Moreover, by applying Equation 5.2, all parameters in θ_{ReVNN} can be updated via backpropagation as well.

After our FI predicts frame IDs for the candidate predicates, we then compare the resulting frame ID with the output of our LVC detector. It is straightforward to treat all the results from our LVC detector as correct. If a verb predicate is assigned to a frame ID but also to an LVC, this verb is reset as a non-predicate verb. Meanwhile, the eventive noun of the LVC is assigned a frame ID. The comprehensive results of our Frame ID and LVC detector then pass to our concept identifier as inputs to the next step.

We use a dependency tree structure with ReVNN rather than the whole sentence for two reasons. First, a dependency parse tree provides a larger number of related words for the target word d. In a word sense disambiguation task, local context tends to provide stronger word sense evidence than long distance context words (Yarowsky, 1995), which we believe is also applicable to the frame identification task. The second reason is that direct child nodes affect the target word sense more than other dependency word nodes. Direct child nodes are always the head word of the child phrase. Using the direct child node $child_d^i$ and the word d itself as features can filter out irrelevant word node information, e.g. articles. Also, using only direct child nodes reduces the training and running time of our LSTM network. To compare the dependency-based ReVNN FI to the traditional whole sentence FI, we design a window-based FI which accepts a sentence as input and identifies the frame ID of target word d considering the words nearest to d. The design of this window-based model is in Figure 5.3. This model extracts features from the n-near content words of d. The feature representation of each word is its word embeddings and POS embeddings. Then the concatenation of the vector representations of these nearby words plus the target words itself is then passed to the feedforward and softmax layers, like the last two layers of the ReVNN model. An alternative model of the window-based one is to use the left and right LSTMs, like the two LSTMs in ourReVNN model, to replace the concatenation of vectors. We called it the Window-Based+NN model. The design of the window-based model is quite straightforward and easy to train. Only the softmax weight and feedforward matrix need to be updated in the backpropagation step. On the other hand, to train the window+NN-based model is as complicated as the ReVNN model.

5.3.3 Data Pre-processing

AMR is enriched by different types of linguistic information on its graph. To simplify our parser, we use several external systems to pre-process our data and obtain this information. For semantic labels (SR), we use the SRL system in the NLP4j toolkit[1] to mark the predicate-argument

[1] https://github.com/emorynlp/nlp4j

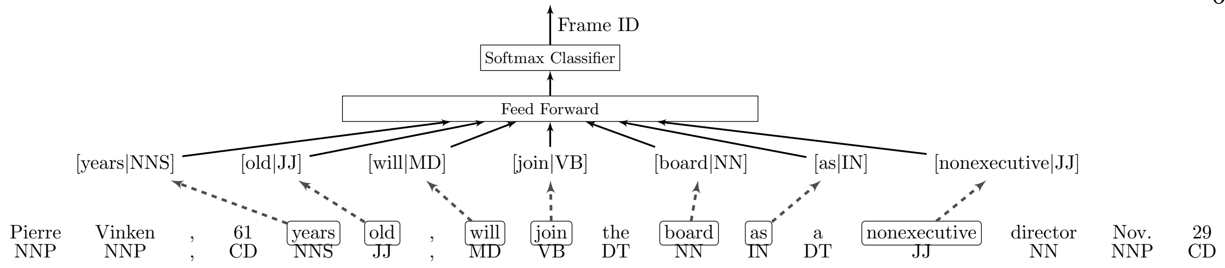


Figure 5.3: Window-based Frame Identifier Framework with window size = 3.

labels on the dependency trees. We re-train the NLP4j SRL system with the unification version of PropBank frame files from ON 5.0. For named entity (NE) tags, we use the named entity recognition (NER) system in the NLP4j toolkit. NLP4j NER uses a fine-grained tag set, including eleven entity name types and seven value types, using the same tag set as in ON 5.0. For wiki-links, we select the Illinois wikifier[2] (Ratinov et al., 2011; Cheng and Roth, 2013) to help us connect entities in sentences to their most relevant Wikipedia page (link). Both systems detect a reasonable range of short phrases (usually noun phrases), which assists the concept identification task. After entities and terms are extracted, named entity tags and wiki-links are attached to head words of the short phrase on a dependency tree. In the case that these extracted phrases do not match the phrase structure of the dependency tree, we heuristically take the lowest common ancestor node as the attachment place. Additional helpful external information includes coreference chains, which aids our post-processing step. We use the coreference resolution system from the Stanford CoreNLP system (Clark and Manning, 2016) to extract the coreference chains.

Another phase of data pre-processing is to use an aligner (Chapter 4) and a frame identifier (Section 5.3.1) on the data set. At the end of this phase, the data set contains dependency trees with attached semantic roles, named entity tags, wiki-links, and alignments between dependency tree nodes, and AMR concepts. An example dependency parse tree with this rich linguistic information is in Figure 5.4.

[2] https://cogcomp.cs.illinois.edu/page/software_view/Wikifier

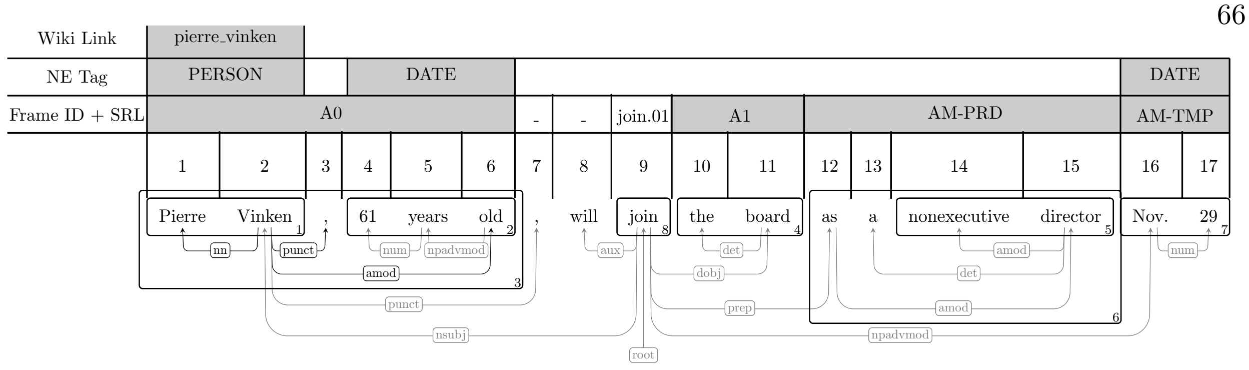


Figure 5.4: Dependency tree with rich linguistic information, include predicate-argument structures, NE tags, and wiki-links.

5.3.4 Concept Identifier

The goal of concept identification (CI) is to extract AMR concepts from an input dependency tree with attached linguistic information. After pre-processing, the dependency parse to AMR concept alignments and the linguistic information are both assembled on the dependency nodes. By extracting the alignments, we can design our concept identifier to receive a dependency parse tree node and return a list of concept candidates. We take the alignments in Figure 4.1 as an example. Our CI generates the "*Person*", "*name*", "*Vinken*", and "-" (the wiki-links) concepts when it receives the dependency tree node "Vinken" as input, while the "*Pierre*" concept will be generated upon receiving the dependency tree node "Pierre." Thus, the concepts that our CI produces correspond to our alignment results. The complete list of the producing concepts from the sample alignments in Figure 4.1 is in Table 5.1. Since the generated concepts are derived from dependency-to-AMR alignments, we can also assign the type of concept generation by reusing the basic feature rules in Table 4.1. In Table 5.1, terms in parentheses represent aligning types.

Observation of these generation rules leads us to conclude these constraints of our CI:

- A dependency tree node only produces one or zero concepts by the basic rules or frame ID.
- A dependency tree node can generate any number of concepts by the "Others" and "Wiki-links" rule types.

Table 5.1: Concepts extracted from aligned dependency trees. The table shows the sample alignment from Figure 4.1. Basic aligning types are followed by concepts.

	Words	Concepts		Words	Concepts
(1)	Pierre	"*Pierre*" (Word)	(9)	join	*j / join-01* (Frame ID)
(2)	Vinken	"*Vinken*" (Word), *p2 / name* (Others),	(10)	the	-
		"-" (Wiki-link), *p / person* (Others)	(11)	board	*b / board* (Lemma)
(3)	,	-	(12)	as	*h / have-org-role-91* (Others)
(4)	61	*61* (Numbers)	(13)	a	-
(5)	years	*y / year* (Lemma)	(14)	nonexecutive	"-" (Others),
(6)	old	*t / temporal-quantity* (Others)			*e / executive* (Partial Lemma)
(7)	,	-	(15)	director	*d2 / director* (Lemma)
(8)	will	-	(16)	Nov.	*11* (Date)
			(17)	29	*29* (Date)

- A dependency tree node is marked as "Remove" if it does not match any rules above.

According to the distribution in Table 4.1, more than 60% of concept alignments belong to the basic match type. Meanwhile, as we calculate the number of concept types in the *Gold Dep.* dataset (Table 4.2), there are more than 8,000 different concepts that belong to the "Other" type. This unbalanced data distribution increases the difficulty of developing our classifier. Instead of designing a multi-label classifier on top of our model, we divide our identifier into a dual-classifier model.

Figure 5.5 shows the design of our CI. The underlying part is similar to our FI, which uses the dependency-based ReVNN as its essential model. One minor difference is that we add more linguistic features, e.g., frame IDs, NEs, and wiki-links, to generate the dependency-based vector representation, e_d. Both e_d and $e_{child_d^i}$ expand their feature representations from the basic feature vector (e_d^{basic}). Then the model follows the same workflow as the vector representation e_d generated by the feedforward layer (FF_{ReVNN}). The next step is to use the same e_d to train two domain-specific feedforward layers: a basic feedforward (FF_{basic}) layer as well as an additional feedforward (FF_{other}) layer. The output of these two feedforward layers are z^{Basic} and z^{Other}. Since the goals of the "basic" and "other" classifiers are different, the logistic regression layers are also different. The "basic" classifier uses a softmax layer which outputs exactly one of the basic concept generation rules. In figure 5.5, we assign a "non-exist" label for an instance that does not generate any

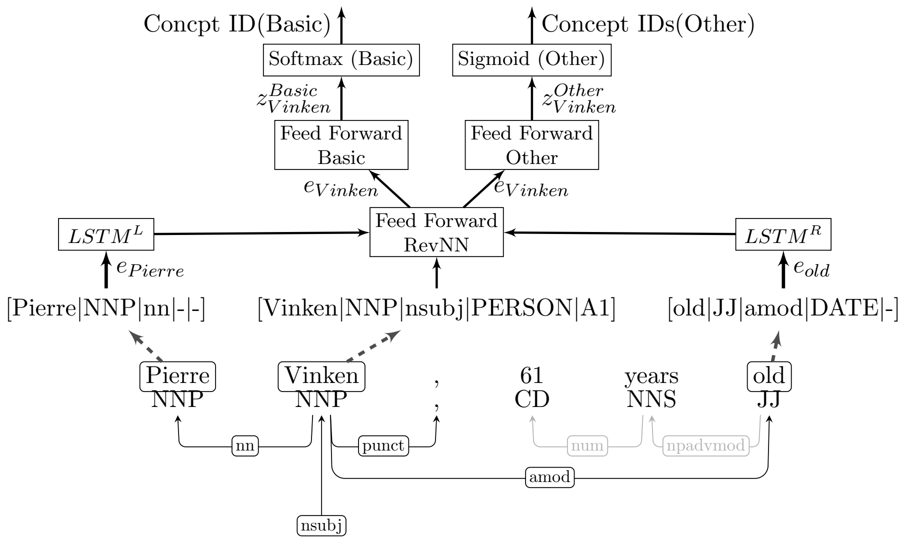


Figure 5.5: Dependency-based ReVNN Concept Identifier Framework. Square blocks indicate the NN units, while "[ ]" denotes vector combinations.

basic type concept. The "other" classifier uses a sigmoid layer which outputs several different "other" concept labels. The sigmoid function is typically designed for binary logistic regression. This type of classifier which outputs multiple possible labels performs as well as training n binary classifiers. Concept labels with probabilities larger than 0.5 are regarded as candidate concepts. The formulation of the softmax and sigmoid layers are listed below:

$$P_{basic}(\bar{y} = bid|d) = softmax(z_d^{basic}) = \frac{\exp(z_d^{bid})}{\sum_{j=1}^{|BID|} \exp(z_d^j)}$$

$$z_d^{basic} = \tanh(e_d^T W^{Basic} + b^{Basic})$$

$$P_{other}(\bar{y} = oid|d) = sigmoid(z_d^{other}) = \frac{1}{1 + \exp(-z_d^{oid})}$$

$$z_d^{other} = \tanh(e_d^T W^{Other} + b^{Other})$$

where $|BID|$ is the total number of basic types, $|OID|$ is the total number of concepts in the other type, $W^{Basic} \in \mathbb{R}^{dim(e_d)\times|BID|}$ and $W^{Other} \in \mathbb{R}^{dim(e_d)\times|OID|}$ are the weighted matrices in FF_{Basic} and FF_{Other}, b^{Basic} and b^{Other} are the bias terms, $z_d^{basic} \in \mathbb{R}^{|BID|}$ and $z_d^{other} \in \mathbb{R}^{|OID|}$ are the output vectors of FF_{Basic} and FF_{Other}. With the alignment between dependency parse node and

AMR concepts, the loss functions of our CI are defined as:

$$J_{Basic}(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{bid=1}^{|BID|} 1\{y_{(i)} = bid\} \log \frac{\exp(z_{(i)}^{bid})}{\sum_{j=1}^{|BID|}\exp(z_{(i)}^{j})}] \tag{5.4}$$

$$J_{Other}(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{oid=1}^{|OID|} 1\{y_{(i)} = oid\} \log \frac{1}{1+\exp(-z_{d}^{oid})}] \tag{5.5}$$

By traversing all dependency tree nodes and applying all alignments to the P_{basic} and P_{other} functions, we obtain the predicted "basic" and "other" labels for all dependency tree nodes in the forward step. Moreover, by applying the two loss functions above, all parameters, including W^{Basic}, W^{Other}, b^{Basic}, b^{Other}, and e_d, will be updated during backpropagation.

The final step for our CI is to gather discrete concepts and attributes to form the concept in a meta-unit. For example, the name concept is a better unit than treating all subcomponents individually. The elements to form the name concept are quite solid, which usually contain ":opN" only (e.g., *name :op1 "Pierre" :op2 "Vinken"* in Table 5.1.) By observing the AMR graphs in the DEFT test and development sets, we categorize the concepts into different types. Descriptions and definitions of these concept types are in Table 5.2. All concepts belong to exactly one concept type. Gathering the concept segments to the concept type makes the relation identification in the next step more efficient. First, gathering concepts avoids identifying leaf concepts in our relation parser. By employing manual rules, the resulting concepts are well-formed and almost in the correct form. In addition, the concept type is coarse-grained information which is a useful feature for our relation identifier. Because the behavior of a single concept type is stable, our relation identifier benefits from this coarse-grained type feature.

After we apply the manual rules above, the resulting concepts with labels are presented. All dependency parse tree nodes with their identified concepts are passed to the relation identifier. The resulting concepts and concept types from Figure 4.1 and Table 5.1 are in Table 5.3.

Table 5.2: Illustrates meta-entity types with their definitions, main components, relation label to parent, and sample concept types. The first part is the basic concept category, while the second part is the attribute category.

Name	**Main Elements**	**Relation Label**	**Sample**
Multiple Sentences	:sntN	ROOT	(m / **multi-sentence** :snt1 (...) :snt2 (...))
And	:opN	-	(a / **and** :op1 (...) :op2 (...))
Quantity	:quant, :unit	-	(**temporal-quantity** :quant 5 :unit (w / week))
Date Entity	:month, :day, :weekday, etc.	:time	(d / **date-entity** :month 1 :day 29)
Name	:opN	:name	:name (n / **name** :op1 “Pierre” :op2 “Vinken”)
Predicate	Core and Non-Core Roles	-	(j / **join-01** :ARG0 (...) :ARG1 (...) :time (...)) (h / **have-org-role-91** :ARG0 (...) :ARG1 (...))
Basic	Non-Core Roles	-	(d2 / **director** :mod (...))
Wiki	-	:wiki	:wiki “United_Kingdom”
Mode	imperative, expressive, interrogative,	:mode	:mode imperative
Negative	-	:polarity	:polarity -
Attribute	-	-	:value 5

5.3.5 Transition-Based Parser

5.3.5.1 Inverse Relation

To maintain AMRs as rooted acyclic graphs, the “inverse” tag is used. To use an “inverse” tag on an AMR relation, an ”-of” string is appended to the end of the relation. For example, an AMR concept *(p / person :ARG0-of l / lead-02)* expresses the same meaning as *(l / lead-02 :ARG0 p / person)*. The “inverse” tag is implemented for three reasons. First, the root concept of an AMR graph serves as a representation of overall focus. The “inverse” tag can help AMR to “pop” the concept being focused on up to the root node. Second, it avoids forming cycles in the graph. Third, it reduces the number of re-entrancies. Two tags have solely an inverse form: “part-of” and “consist-of”.

When designing an AMR parser, the inverse relation is observed to increase data sparsity. The number of relation types almost doubles. The statistics of the number of core and non-core arguments in DEFT AMR Corpus are shown in Table 5.4. According to the number, 19.6% of the core arguments and 5.1% of the non-core arguments are in the inverse form. Since the inverse relation represents the same meaning as the original relation tag, we can simply “reverse” an inverse

Table 5.3: Resulting concepts of our concept identifier from Figure 4.1 and Table 5.1. Concept types are listed as well

#	Concept	Type	#	Concept	Type
c_0	-	Wiki	c_6	(d2 / director)	Basic
c_1	(p2 / name :op1 “Pierre” :op2 “Vinken”)	Name	c_7	(h / have-org-role-91)	Predicate
c_2	(t / temporal-quantity :quant 61 :unit (y / year))	Quantity	c_8	(b / board)	Basic
c_3	(p / person)	Basic	c_9	(d / date-entity :month 11 :day 29)	Date Entity
c_4	-	Negative	c_{10}	(j / join-01)	Predicate
c_5	(e / executive)	Basic			

Table 5.4: Distribution of AMR Core and Non-Core argument relation types and inverse forms in LDC DEFT AMR corpus test set

Core Arguments	# of tags	# of inverse tags	Non-Core Arguments	# of tags	# of inverse tags
:ARG0	82,458	17,439	:location	7,722	423
:ARG1	127,836	28,647	:manner	4,324	374
:ARG2	37,694	3,161	:quant	7,195	289
:ARG3	3,643	338	:time	12,918	285
:ARG4	1,199	54	:instrument	416	193
Other Core Arguments	38	7	:concession	787	149

relation before parsing. The reversed AMR graph of Figure 1.2 is in Figure 5.6. By following the reversed strategy, the “*:ARG1-of* relation is then reversed to “*:ARG1* relation (the label in the round square). The advantage of reversing an inverse relation is that this reduces the data variety of the relation tags, which is important when identifying the relation between two entities. However, the drawbacks of applying the reverse strategy on an inverse relation are also apparent. First, it will generate additional re-entrancies. For example, in Figure 5.6, the concept “*board*” is referred to twice: by its original parent concept “*join-01*” and also by the additional parent concept “*have-org-role-91*”. Another drawback is, the reverse strategy may generate multiple roots, which conflicts with the single-rooted graph constraint. For example, in Figure 5.6, “*have-org-role-91*” is an additional root concept besides the original concept “*join-01*”.

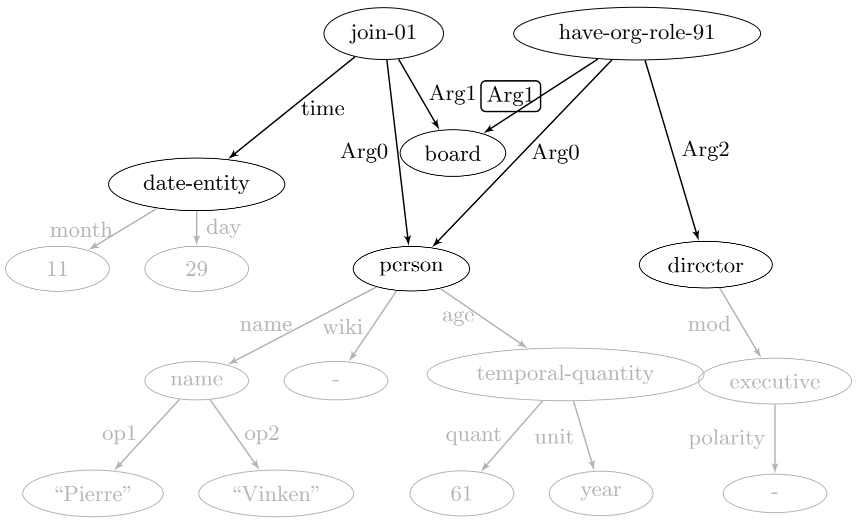


Figure 5.6: The AMR annotation of sentence "Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29." with reversed relations.

5.3.5.2 Action Extractor

Relation identification of AMRs is defined as follows: given a dependency tree D with a list of dependency nodes $[d_1, ..., d_m]$, a list of concept candidates $[c_1, ..., c_n]$, and the alignments A, the transition-based parser aims to 1) convert the dependency tree structure to an AMR graph, 2) identify the relations between a pair of concepts, and 3) assign a relation label to this concept pair. We define our transition-based AMR parser as a tuple $\Gamma = (S, \Lambda, s_0, s_{term})$, where

- S: a set of parsing states. All parsing states are represented as tuples $(\lambda_{store}, \lambda_{in}, \lambda_{post}, \lambda_{comp}, c_a, R)$, where $\lambda_{store}, \lambda_{in}$, λ_{post}, and λ_{comp} are stacks of concepts which store non-processing concepts, in-processing concepts, post-processing concepts, and concepts that have completed processing, respectively. c_a is the concept that our parser currently focuses on. R is a set of labeled arcs storing identified relations and labels.

- Λ: a set of parsing actions (transitions), each of which is a function $S_i \rightarrow S_j$. After applying the transition action on the current parsing state, the contents in the stacks or the labeled arcs in S change.

- s_0: initial state. Here we set the initial state as $(\lambda_{store}, \lambda_{in}, \lambda_{post}, \lambda_{comp}, c_a, R) = ([c_1, c_2, ...c_n],$

$[], [], [], \phi, \emptyset)$. λ_{store} initiates with concept candidates arranged in post-order traversal.

- s_{term}: terminal state. We set s_{term} as $(\lambda_{store}, \lambda_{in}, \lambda_{post}, \lambda_{comp}, c_a, R) = ([], [], [], [c_{root1}, c_{root2}, ..., c_{rootN}], [], \phi, R_{final})$, where c_{root} consists of the root concepts of the output AMR graph.

There are seven different actions in our algorithm. The descriptions of the actions are shown in Table 5.5.

- FORWARD-MOVE: Move the top concept c_j in λ_{in} to λ_{post} if there is no relation between c_a and c_j.
- SHIFT: Move the top concept in λ_{store} which is aligned with the current dependency node to the focusing concept c_a. Also, a set of concepts in λ_{post} which are relative to c_a are moved to λ_{in}.
- LINK-REL $\xrightarrow{rel}$: Assign arc with label *rel* between c_a and c_j (the top concept at λ_{in}).
- LINK-ARG $\xrightarrow{arg}$: Assign arc with semantic label *arg* between c_a and c_j (the top concept at λ_{in}).
- SWAP: Switch c_a with the top concept in λ_{in}.
- NEXT-NODE: Move c_a to λ_{post}. This happens when λ_{in} is empty.
- MOVE-COMPLETE: Move c_a to λ_{comp}. This happens if c_a is a root concept and all its child concepts have been linked.

All these actions are performed with some preconditions. For example, LINK-REL, LINK-ARG, and SWAP can only be applied when c_a is assigned and λ_{in} is not empty. All preconditions are listed in Table 5.5.

The SHIFT action moves the top concept in λ_{store} to the concept c_a in focus. Meanwhile, the concepts in λ_{post} that are related to c_a are moved to λ_{in}, which aims to discover the relation or argument links later. Thus, there are two fundamental strategies we need to design carefully. The

Table 5.5: Actions in our Transition-based Relation Identification. For each row, the first line indicates an action and the second line indicates preconditions of that action.

FORWARD-MOVE	$(\lambda_{store}, [\lambda_{in}\|c_j], [\lambda_{post}], \lambda_{comp}, c_i, R) \Rightarrow (\lambda_{store}, [\lambda_{in}], [\lambda_{post}\|c_j], \lambda_{comp}, c_i, R)$ $oracle(c_i, c_j) = \varnothing \wedge c_i \neq \phi$
SHIFT	$([\lambda_{store}\|c_i], \lambda_{in}, [\lambda_{post}\|c_{j0}, c_{j1}, ..., c_{jn}], \lambda_{comp}, \phi, R) \Rightarrow ([\lambda_{store}], [c_{j0}, c_{j1}, ..., c_{jn}], [\lambda_{post}], \lambda_{comp}, c_i, R)$ $\lambda_{in} = [] \wedge \forall c_{jk} hasDepRelation(c_i, c_{jk}) \wedge previousAction =$ NEXT-NODE
LINK-REL $\xrightarrow{rel}$	$(\lambda_{store}, [\lambda_{in}\|c_j], \lambda_{post}, \lambda_{comp}, c_i, R) \Rightarrow (\lambda_{store}, [\lambda_{in}], \lambda_{post}, \lambda_{comp}, c_i, R \cup \{c_i \xrightarrow{rel} c_j)$ $\exists oracle(c_i, c_j) = \{c_i \xrightarrow{rel} c_j\}$
LINK-ARG $\xrightarrow{arg}$	$(\lambda_{store}, [\lambda_{in}\|c_j], \lambda_{post}, \lambda_{comp}, c_i, R) \Rightarrow (\lambda_{store}, [\lambda_{in}], \lambda_{post}, \lambda_{comp}, c_i, R \cup \{c_i \xrightarrow{arg} c_j)$ $\exists oracle(c_i, c_j) = \{c_i \xrightarrow{arg} c_j\}$
SWAP	$(\lambda_{store}, [\lambda_{in}\|c_j], \lambda_{post}, \lambda_{comp}, c_i, R) \Rightarrow (\lambda_{store}, [\lambda_{in}\|c_i], \lambda_{post}, \lambda_{comp}, c_j, R)$ $isDescendantConcept(c_j, c_i)$
NEXT-NODE	$(\lambda_{store}, \lambda_{in}, [\lambda_{post}], \lambda_{comp}, c_i, R) \Rightarrow (\lambda_{store}, \lambda_{in}, [\lambda_{post}\|c_i], \lambda_{comp}, \phi, R)$ $\lambda_{in} = []$
MOVE-COMPLETE	$(\lambda_{store}, \lambda_{in}, \lambda_{post}, [\lambda_{comp}], c_i, R) \Rightarrow (\lambda_{store}, \lambda_{in}, \lambda_{post}, [\lambda_{comp}\|c_i], \phi, R)$ $\lambda_{in} = [] \wedge isAllDescendantConceptLinked(c_i)$

first strategy involves the order in which we move concepts in λ_{store} to the focusing concept c_a. The second strategy involves which concepts in λ_{post} should be pushed to λ_{in} for further processing. Non-optimal strategies might cause non-desirable actions during parsing, for example causing extra SWAP actions, which are difficult to detect. As the structure of a dependency parse and its AMR graph correspond to a high degree, we can rely on the dependency parse tree as the basic input structure. Therefore, the order of popping the concept from λ_{store} is identical to the post-order traversal of the dependency tree nodes and popping its aligned concepts. For the dependency tree nodes that are aligned to multiple concepts, we prefer to pop a concept located near the leaf earlier than a concept at a higher level. As such, we define the popping order as follows:

$$\begin{aligned}&\text{Attributes (e.g., :polarity negative, number, Wiki links)} \Rightarrow \text{Name Concept} = \text{Quantity Concept}\\ &\Rightarrow \text{Basic Concept} \Rightarrow \text{And Concept} \Rightarrow \text{Multiple Sentences Concept} \qquad (5.6)\end{aligned}$$

When popping from a dependency tree node with multiple concepts, our parser extracts the concept belonging to the top category first. Next, when our parser performs another SHIFT action, the category belonging to the bottom category will be popped. The concepts that are moved from λ_{post} to λ_{in} are the target child concepts that likely contain links with concept c_a in focus. We want the concepts in λ_{in} to cover all the concepts that are highly likely to link to c_a. Conversely, if we place too many concepts in λ_{in}, our parser needs to perform extra FORWARD-MOVE steps to skip concepts without any relations to c_a. Thus the following rules are designed to decide which concepts need to be popped when the focusing concept c_a aligns to dependency node d:

(1) Pop the concepts in λ_{in} that are aligned to the descendant dependency nodes of d

(2) Pop the concepts in λ_{in} that are aligned to the sibling dependency nodes of d. This only happens when d is the rightmost child nodes of its parent node.

Table 5.6 illustrates parsing states generated by our algorithm. There are two things to point out in this parsing state sample. First, we only create the major links in our transition parser. Reentrancies are skipped here as these usually link two concepts whose aligned dependency tree nodes are not nearby. Capturing the reentrancy in the parser will increase the number of transitions dramatically. We figure out the reentrancies as a post-processing step. Second, two concepts (c_7 and c_{10}) exist in λ_{comp} when our parsing processing terminates. Since there are no direct relations between these root concepts and they do not consist of a disjoint subset, our parser can terminate the parsing and move all root concepts to the next step, which is to reconstruct and form the AMR graph.

Our transition-based parser is somewhat more complicated than a basic shift-reduce parsing algorithm (Nivre, 2008). A basic shift-reduce algorithm contains a buffer to store word tokens and a stack to store elements that are being processed. Our parser increases the number of stacks to four. We also extend the set of transitions from the basic shift-reduce algorithm by adding SWAP and MOVE-COMPLETE. Therefore, our parser has a greater ability to create directed acyclic graphs that are more complex than a non-projective tree. Clearly, time complexity becomes an issue as we improve the power of our transition-based parser. To observe the increased of actions when we parse longer sentences, we count the number of different oracle actions applied when parsing all AMR graphs in the DEFT test set. A stacked bar graph describing the average number of actions for different sentence lengths is given in Figure 5.7. According to the figure, the number of actions increases linearly as sentence length increases. In the same figure, we list the distribution

Table 5.6: All relation parsing states achieved by applying our transition-based parsing algorithm to the dependency tree in Figure 4.1 and concept candidates in Table 5.3.

	λ_{store}	λ_{in}	λ_{post}	λ_{comp}	Dep. Tree Node	c_a	R	Oracle Action
1	$[c_0, ..., c_{10}]$	[]	[]	[]	-	-	$\{\emptyset\}$	Next-Node
2	$[c_0, ..., c_{10}]$	[]	[]	[]	Pierre(1)	-		Delete
3	$[c_0, ..., c_{10}]$	[]	[]	[]	-	-		Next-Node
4	$[c_0, ..., c_{10}]$	[]	[]	[]	,(3)	-		Delete
5	$[c_0, ..., c_{10}]$	[]	[]	[]	-	-		Next-Node
6	$[c_0, ..., c_{10}]$	[]	[]	[]	61(4)	-		Delete
7	$[c_0, ..., c_{10}]$	[]	[]	[]	-	-		Next-Node
8	$[c_0, ..., c_{10}]$	[]	[]	[]	years(5)	-		Delete
9	$[c_0, ..., c_{10}]$	[]	[]	[]	-	-		Next-Node
10	$[c_0, ..., c_{10}]$	[]	[]	[]	old	-		Shift
11	$[c_0, c_1, c_3, ..., c_{10}]$	[]	[]	[]	-	c_2		Next-Node
12	$[c_0, c_1, c_3, ..., c_{10}]$	[]	$[c_2]$	[]	,(7)	-		Delete
13	$[c_0, c_1, c_3, ..., c_{10}]$	[]	$[c_2]$	[]	-	-		Next-Node
14	$[c_0, c_1, c_3, ..., c_{10}]$	[]	$[c_2]$	[]	Vinken(2)	-		Shift
15	$[c_1, c_3, ..., c_{10}]$	$[c_2]$	[]	[]	-	c_0		Forward-Move
16	$[c_1, c_3, ..., c_{10}]$	[]	$[c_2]$	[]	-	c_0		Next-Node
17	$[c_1, c_3, ..., c_{10}]$	[]	$[c_2, c_0]$	[]	Vinken(2)	-		Shift
18	$[c_3, ..., c_{10}]$	$[c_0, c_2]$	[]	[]	-	c_1		Forward-Move
19	$[c_3, ..., c_{10}]$	$[c_0]$	$[c_2]$	[]	-	c_1		Forward-Move
20	$[c_3, ..., c_{10}]$	[]	$[c_2, c_0]$	[]	-	c_1		Next-Node
21	$[c_3, ..., c_{10}]$	[]	$[c_2, c_0, c_1]$	[]	Vinken(2)	-		Shift
22	$[c_4, ..., c_{10}]$	$[c_1, c_0, c_2]$	[]	[]	-	c_3		Link-Rel $\xrightarrow{:age}$
23	$[c_4, ..., c_{10}]$	$[c_1, c_0]$	[]	[]	-	c_3	$R \cup \{c_3 \xrightarrow{:age} c_2\}$	Link-Rel $\xrightarrow{:wiki}$
24	$[c_4, ..., c_{10}]$	$[c_1]$	[]	[]	-	c_3	$R \cup \{c_3 \xrightarrow{:wiki} c_0\}$	Link-Rel $\xrightarrow{:name}$
25	$[c_4, ..., c_{10}]$	[]	[]	[]	-	c_3	$R \cup \{c_3 \xrightarrow{:name} c_1\}$	Next-Node
26	$[c_4, ..., c_{10}]$	[]	$[c_3]$	[]	will(8)	-		Delete
27	$[c_4, ..., c_{10}]$	[]	$[c_3]$	[]	-	-		Next-Node
28	$[c_4, ..., c_{10}]$	[]	$[c_3]$	[]	the(10)	-		Delete
29	$[c_4, ..., c_{10}]$	[]	$[c_3]$	[]	-	-		Next-Node
30	$[c_4, ..., c_{10}]$	[]	$[c_3]$	[]	board(11)	-		Shift
31	$[c_4, ..., c_7, c_9, c_{10}]$	[]	$[c_3]$	[]	-	c_8		Next-Node
32	$[c_4, ..., c_7, c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	a(13)	-		Delete
33	$[c_4, ..., c_7, c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	-	-		Next-Node
34	$[c_4, ..., c_7, c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	nonexecutive(14)	-		Shift
35	$[c_5, ..., c_7, c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	-	c_4		Next-Node
36	$[c_5, ..., c_7, c_9, c_{10}]$	[]	$[c_3, c_8, c_4]$	[]	nonexecutive(14)	-		Shift
37	$[c_6, c_7, c_9, c_{10}]$	$[c_4]$	$[c_3, c_8]$	[]	-	c_5		Link-Rel $\xrightarrow{:polarity}$
38	$[c_6, c_7, c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	-	c_5	$R \cup \{c_5 \xrightarrow{:polarity} c_4\}$	Next-Node
39	$[c_6, c_7, c_9, c_{10}]$	[]	$[c_3, c_8, c_5]$	[]	director(15)	-		Shift
40	$[c_7, c_9, c_{10}]$	$[c_5]$	$[c_3, c_8]$	[]	-	c_6		Link-Rel $\xrightarrow{:mod}$
41	$[c_7, c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	-	c_6	$R \cup \{c_6 \xrightarrow{:mod} c_5\}$	Next-Node
42	$[c_7, c_9, c_{10}]$	[]	$[c_3, c_8, c_6]$	[]	as(12)	-		Shift
43	$[c_9, c_{10}]$	$[c_8, c_6]$	$[c_3]$	[]	-	c_7		Forward-Move
44	$[c_9, c_{10}]$	$[c_6]$	$[c_3, c_8]$	[]	-	c_7		Link-Arg $\xrightarrow{:ARG2}$
45	$[c_9, c_{10}]$	[]	$[c_3, c_8]$	[]	-	c_7	$R \cup \{c_7 \xrightarrow{:ARG2} c_6$	Move-Complete
46	$[c_9, c_{10}]$	[]	$[c_3, c_8]$	$[c_7]$	-	-		Next-Node
47	$[c_9, c_{10}]$	[]	$[c_3, c_8]$	$[c_7]$	29(17)	-		Delete
48	$[c_9, c_{10}]$	[]	$[c_3, c_8]$	$[c_7]$	-	-		Next-Node
49	$[c_9, c_{10}]$	[]	$[c_3, c_8]$	$[c_7]$	Nov.(16)	-		Shift
50	$[c_{10}]$	[]	$[c_3, c_8]$	$[c_7]$	-	c_9		Next-Node
51	$[c_{10}]$	[]	$[c_3, c_8, c_9]$	$[c_7]$	.(18)	-		Delete
52	$[c_{10}]$	[]	$[c_3, c_8, c_9]$	$[c_7]$	-	-		Next-Node
53	$[c_{10}]$	[]	$[c_3, c_8, c_9]$	$[c_7]$	join(9)	-		Shift
54	[]	$[c_9, c_8, c_3]$	[]	$[c_7]$	-	c_{10}		Link-Arg $\xrightarrow{:ARG0}$
55	[]	$[c_9, c_8]$	[]	$[c_7]$	-	c_{10}	$R \cup \{c_{10} \xrightarrow{:ARG0} c_3\}$	Link-Arg $\xrightarrow{:ARG1}$
56	[]	$[c_9]$	[]	$[c_7]$	-	c_{10}	$R \cup \{c_{10} \xrightarrow{:ARG1} c_8\}$	Link-Rel $\xrightarrow{:time}$
57	[]	[]	[]	$[c_7]$	-	c_{10}	$R \cup \{c_{10} \xrightarrow{:time} c_9\}$	Move-Complete
58	[]	[]	[]	$[c_7, c_{10}]$	-	-		Next-Node
59	[]	[]	[]	$[c_7, c_{10}]$	-	-		-

of different actions to generate AMR graphs from the test set. The actions that appear most likely are Forward-Move and Next-Node while Swap actions appear to be the least likely. However, the Swap action is essential since some AMR graphs can only be generated by applying Swap at the correct time. The lack of training instances with Swap actions causes our parser difficulty in decoding the correct AMR graph at running time.

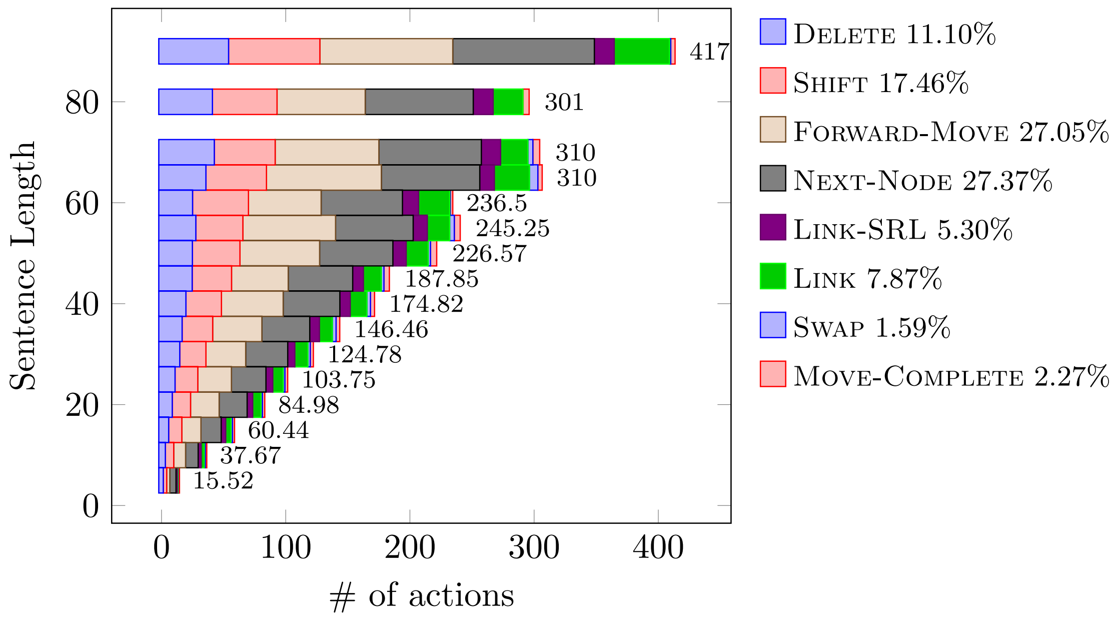


Figure 5.7: Distribution of different action types vs. length of sentence. Numbers by the legends are the distribution of different actions our parser needs to apply when parsing all sentences in the DEFT test set.

5.3.5.3 Relation Identification Model

With the parsing states and oracle transitions from our action extractor, we can train a classifier that decides the correct action under different states. Traditionally, a transition-based parser decides the action and the arc labels simultaneously. For example, in Nivre (2008)'s dependency parser, the classifier selects the correct action from $|T| = 2{*}N_l$ + two options, where N_l is the number of different dependency labels. However, the feature usages and weight parameters for training an action classifier are different from training an arc label classifier. Instead of using one model to decide both actions and arc labels, we design three separate output feedforward layers, z_{action}, z_{arg}, and z_{rel} for our parser. The z_{action} layer decides which action our parser should take, while z_{arg} and z_{rel} decide the arguments and relation tags, respectively. Figure 5.8 illustrates the hierarchy of our parser. The underlying layer outputs, h_{state}, which contains the feature representations of the states, are shared by the three output feedforward layers.

For the underlying feature output layer (h_{state}), two sub-vectors are composed. The first part

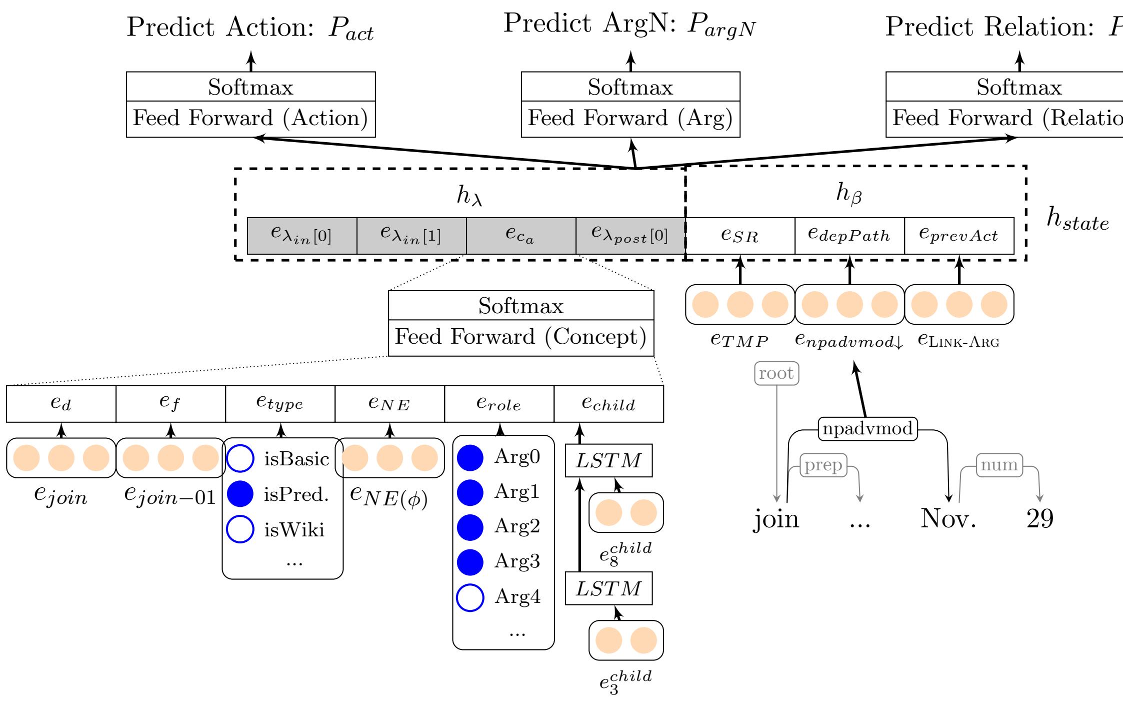


Figure 5.8: Overall Relation Identification Model. This figure demonstrates the state of the 55-th action in Table 5.3, where $e_a = e_{10}$ (join-01) and the top concept in $\lambda_{in} = e_9$ (d / date-entity). Small circles represent an element in a vector. Blue circles represent elements set to 1, while hollow circles represent elements set to 0.

is h_λ, which represents the concepts in λ_{in}, λ_{post}, and c_a. The second part is h_β, which contains the features between c_a and the top concept in λ_{in}, e.g., predicate-argument relation e_{SR} and dependency path $e_{depPath}$, and the features of the parser status, e.g., the previous action $e_{prevAct}$ that the parser took. For the first sub-vector h_λ, we extract the concept vector representations of the top two concepts in λ_{in}, the top concepts in λ_{post}, and c_a. These four vectors are concatenated to form h_λ. When the stack is empty, a random vector is generated to represent an empty concept. For the second vector h_β, the vector representation of the semantic role, dependency path, and previous action are also generated and concatenated. h_λ and h_β then form the underlying feature output layer h_{state}.

Next, we shift our attention to deciding how to map a concept to its vector representation which composes the h_λ vector. Since each concept aligns to one dependency node, we can obtain lexical and dependency relations of concept c from its aligning dependency node d plus other linguistic information attached to d, e.g., the named entity and semantic roles that c contains. The vector representation is the combination of six different categories of features:

- **Dependency-Based ReVNN** (e_d): The dependency-based ReVNN which we described in Section 5.3.2 represents the meaning of the dependency node based on the node itself and all its child nodes. This feature vector is a combination of various linguistic features attached to a dependency tree node, including word embedding, POS embedding, and dependency relation embedding.

- **Alignment Type and Frame ID Embedding** (e_f): Alignment types to which the Concept c - Dependency tree node d pair belongs are defined in Section 4.2.1. Additionally, if c belongs to a frame, we use its frame ID as the representation instead. As such, we create random embeddings to express each alignment type and frame ID.

- **Concept Type** (e_{type}): We define eleven concept types in Table 5.2 resulting in e_{type} being a one-hot vector with eleven elements. The i-th element is set to 1 if c belongs to the i-th concept type. Otherwise, all other elements are set to 0.

- **Named Entity** (e_{NE}): the named entity tag vector representation.

- **Frame Role Sets** (e_{role}): The core argument usages for a frame are defined in its frame file. For example, Table 1.1 defines the core argument usages for the different frames of "free." When c belongs to a frame ID, we use a one-hot vector to express the core argument usages. For example, if the target frame has "Arg1" and "Arg2" as its core arguments, we then mark the 2-nd and 3-rd elements in e_{role} as 1, and set the rest of the elements to 0.

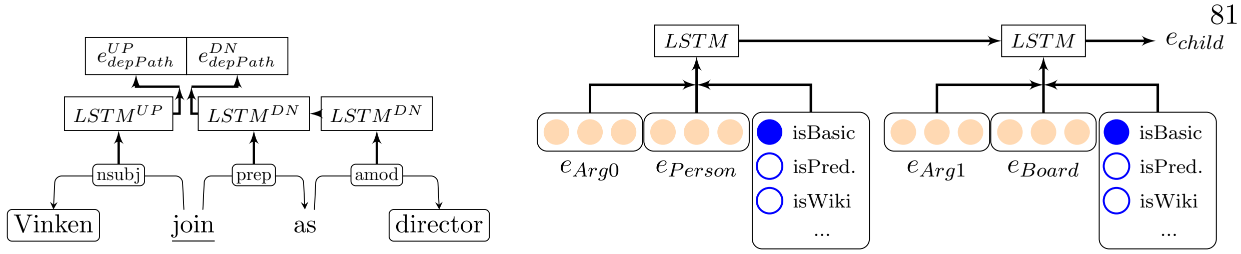
- **Child Concepts** (e_{child}): When deciding the argument labels and relation labels between c and its child concept c_{child}, it is important to consider the relation labels that were previously assigned to other child concepts. A single relation label usually appears once under the same concept. Furthermore, the properties of the child concept are also important features for relation identification. For this, we need a mechanism to record all child relation labels plus child concept properties. An LSTM network is an excellent fit for such purposes. The child concept LSTM ($LSTM_{child}$) receives the embedding of the relation labels and child concept properties, i.e., aligning type, frame ID embedding, and concept type, from the previous linking child concepts. The output of $LSTM_{child}$ then becomes e_{child}. In Figure 5.9b, the child concept model generates the e_{child} for c_a with two previous linking child concepts, ":ARG0 Person" and ":ARG1 board".

The concatenation vector of the above six features is then passed to a feedforward layer, $z_{Concept}$, which generates the concept vector representations.

Besides the vector representation of a concept, the features between concepts also need to be addressed. h_β captures the relation features between the focusing concept c_a and the top concept in λ_{in} (denoted as $\lambda_{in}[0]$ or c_{child}.) We assume that c_a aligns to dependency tree node d_a, while c_{child} aligns to node d_{child}. If either c_a or c_{child} is empty, we simply fill in a randomly generated vector to represent the empty concept. Otherwise, the following three feature vectors comprise h_β:

- **Semantic Role** (e_{SR}): We have previously discussed the high degree to which AMR relation labels rely on semantic roles. Thus we decode all semantic roles in PropBank to their vector representations as one feature. Once there is a semantic role existing between d_a and d_{child}, we map this role embedding to e_{SR}.

- **Dependency Path** ($e_{depPath}$): We have also previously discussed how dependency path features are useful for identifying the relation between entities on the dependency tree. In our parser, dependency path features are used to capture the syntactic relation between d_a and d_{child}. Here we implement the shortest dependency path: the long short-term memory model (SDP-LSTM) proposed by Xu et al. (2015). Figure 5.9a illustrates the framework of our dependency path feature with the LSTM model. The first step is to find the common ancestor node, $d_{ancestor}$, of d_a and d_{child}. In Figure 5.9a, the common ancestor node for "Vinken" and "director" is "join". Then we extract two separate dependency tree paths: the first from d_a to $d_{ancestor}$ ("Vinken" to "join"), the other from d_{child} to



(a) Dependency path features with LSTM (b) Child concept features with LSTM

Figure 5.9: The figure on the left illustrates the dependency path relation between "Vinken" and "director". The figure on the right illustrates the generation of child concepts at the 59-th action in Table 5.5

$d_{ancestor}$ ("director" to "join".) Unlike the original paper, here we only use dependency labels as input feature vectors. The two output vectors ($e^{UP}_{depPath}$ and $e^{DN}_{depPath}$) from the LSTM models then compose the $e_{depPath}$ vector.

- **Previous Action** ($e_{prevAct}$): The previous action is another important feature in a transition-based parser. Here we decode all seven action types in our parser to action embeddings. Then the previous action vector is extracted to form the $e_{prevAct}$ vector.

As our parser model is comprised of different sub-models and feature vectors, the number of model parameters that we need to estimate is enormous. All model parameters and embeddings are updated with the backpropagation algorithm during training. However, both forward and backward propagation of the dependency-based ReVNN model is time-consuming. The e_d vector is induced from all its descendant nodes. When our model obtains the e_d vector of a root node, our ReVNN model needs to traverse and calculate the vector representations of all tree nodes with the tree LSTM model. Also, a single dependency node man need to induce its ReVNN vector repeatedly if our parser obtains the ReVNN vector of its ancestor nodes from different training batches. These issues slow our parsing model down. To balance the power of ReVNN and the running speed, we adopt the following two strategies:

- When the ReVNN model wants to apply forward and backward propagation to dependency node d_a, it only traverses d_a's direct child. The grandchild nodes are not affected.

- The ReVNN model computes forward and backward propagation only once in each epoch of training. The model induces all the vector representations at the start of each epoch, then caches these values for further use in the same epoch. In backpropagation, the ReVNN model accumulates all the gradients in respect to the same ReVNN parameter in the same epoch. Then each parameter is updated only once in each epoch as well.

To train our model, with the training sample $\mathbb{S} = \{(C_i, D_i, A_i)\}_{i=1}^{m}$ that contains a set of AMR(C_i)-dependency parse tree(D_i) pairs with alignments(A_i), we need to extract the oracle transition set $\{a_{(j)}\}_{j=1}^{|\Lambda_i|}$ from their corresponding states $s_i^{(j)}$, where $a_{(j)} = oracle(s_i^{(j)})$, and $s_i^{(j)} = \{\lambda_{store}^{(j)}\lambda_{in}^{(j)}, \lambda_{post}^{(j)}, \lambda_{comp}^{(j)}, c_a^{(j)}, R^{(j)}, a_{(j-1)}\}_i$. $|\Lambda_i|$ is the number of oracle transitions we extracted from training instance i. Also, when $a_{(j)}$ is Link-Arg or Link-Rel, we need to extract the argN ($argN_{(j)}$) or relation ($rel_{(j)}$) labels by giving their corresponding states $s_i^{(j)}$. $h_{state-i}^{(j)}$ is the underlying feature output under current state $s_i^{(j)}$. The formal definition of our three parsers is defined as:

$$P_{act}(\bar{y} = act|s_i^{(j)}) = \frac{\exp(\tanh((h_{state-i}^{(j)}W_{act} + b_{act})^{(act)}))}{\sum_{k=1}^{|ACT|}\exp(\tanh((h_{state-i}^{(j)}W_{act} + b_{act})^{(k)}))}$$

$$P_{argN}(\bar{y} = argN|s_i^{(j)}) = \frac{\exp(\tanh((h_{state-i}^{(j)}W_{argN} + b_{argN})^{(argN)}))}{\sum_{k=1}^{|ARGN|}\exp(\tanh((h_{state-i}^{(j)}W_{argN} + b_{argN})^{(k)}))}$$

$$P_{rel}(\bar{y} = rel|s_i^{(j)}) = \frac{\exp(\tanh((h_{state-i}^{(j)}W_{rel} + b_{rel})^{(rel)}))}{\sum_{k=1}^{|REL|}\exp(\tanh((h_{state-i}^{(j)}W_{rel} + b_{rel})^{(k)}))}$$

The final training objective is to minimize the cross-entropy loss, plus a l_2-regularization term:

$$\begin{aligned} L(\theta) = -\frac{1}{\sum_{i=1}^{m}|\Lambda_i|}[\sum_{i=1}^{m}[\sum_{j=1}^{|\Lambda_i|}(\sum_{act=1}^{|ACT|} 1\{a_{(j)} = act\} log P_{act}(\bar{y} = act|s_i^{(j)})) \\ + 1\{a_{(j)} = \textsc{Link-Arg}\}\sum_{argN=1}^{|ARGN|} 1\{argN_{(j)} = argN\} log P_{argN}((\bar{y} = argN|s_i^{(j)})) \\ + 1\{a_{(j)} = \textsc{Link-Rel}\}\sum_{rel=1}^{|REL|} 1\{rel_{(j)} = rel\} log P_{rel}(\bar{y} = rel|s_i^{(j)})]] + \frac{\lambda}{2}\|\theta\|^2 \end{aligned}$$

where θ is the set of all parameters $\{\theta_{ReVNN}, \theta_{childConcept}, \theta_{depPath}, E_f, E_{type}, E_{NE}, E_{role}, E_{SR}, W_{act}, b_{act}, W_{argN}, b_{argN}, W_{rel}, b_{rel}\}$. E stands for the embeddings we create for different words, POS, named entity, etc. The W and b parameters are for the feedforward models. θ_{ReVNN}, $\theta_{childConcept}$, $\theta_{depPath}$ are another set of parameters containing the parameters for ReVNN, child concept, and dependency path models.

5.3.6 Decoding and Post Processing

At decoding time, our parser takes two steps to generate the AMR graph by giving the dependency tree and a list of candidate concepts. The first step is to run the transition-based parser from its initial states until it finalizes with a list of standalone root concepts. Our parser inferences the transition actions from the parser parameters that we trained. The second step is the post-processing part, which attaches the coreference mentions to the existing graphs, and links the root concepts to form a complete AMR graph.

We start from the transition parsing. The input dependency parse tree is D, the input candidate concept is a list of $[c_1, c_2, ..., c_n]$, and the model parameter is θ. We pass these inputs to our parsing data structures to compose the initial state s_0. Then we repeatedly apply the current state $s^{(j)}$ and the trained parameters θ to the action classifier, P_{act}. Here we greedily pick the action with the maximum probability as the predicted one.

$$\bar{a_{(j)}} = \operatorname*{argmax}_{a \in ACT} P_{act}(a|s^{(j)})$$

If the predicted action $\bar{a_j}$ is Link-Arg or Link-Rel, our parser needs to infer the argN classifier P_{argN} or relation classifier P_{rel} to predict the link labels between c_a and the top node of λ_{in}.

$$\bar{argN}_{(j)} = \operatorname*{argmax}_{argN \in ARGN} P_{argN}(argN|s^{(j)}) \ \textit{if} \ \bar{a_{(j)}} = \textsc{Link-Arg}$$

$$\bar{rel}_{(j)} = \operatorname*{argmax}_{rel \in REL} P_{rel}(rel|s^{(j)}) \ \textit{if} \ \bar{a_{(j)}} = \textsc{Link-Rel}$$

After the parser obtains the predicted action $\bar{a_{(j)}}$ (and predicted argN label $ar\bar{gN}_{(j)}$ or rel label $re\bar{l}_{(j)}$ if applicable), we will apply $(\bar{a_{(j)}}, ar\bar{gN}_{(j)}, re\bar{l}_{(j)})$ to update the parsing states. Our parser follows the update rules in Table 5.5. To improve running speed, we implement several cache mechanisms for the sub-models. For example, the ReVNN vectors are inferred at the beginning of the decoding time by traversing all dependency tree nodes. Also, our parser initializes the empty child concept vector e_{child} for each concept candidate. Once our parser predicts that c_a should link to c_{child} with label l, we update the empty child concept with the vector that is generated by $LSTM_{child}$ giving (c_{child}, l). By repeatedly predicting the actions and updating the transition states, our parser continues generating a list of root concepts in λ_{comp} until it reaches the terminal state s_{term}.

When our parser terminates with a list of root concepts $[c_{root1}, c_{root2}, ..., c_{rootN}]$, we move to the post-processing step. First, we attach the mentions of coreference chains to the AMR graph. Here we use an external coreference resolution system to extract the coreference chains within the sentence. Say the coreference resolution system finds a coreference chain $[m_1, m_2, ..., m_n]$. Our parser ignores cross-sentence coreference chains. Then we apply the following heuristic strategies to attach coreference mentions to the graph. A sample coreference chain attachment is in Figure 5.10.

(1) Links the mentions to their corresponding dependency tree nodes. Dependency nodes containing coreference mentions are represented as d_{m_i}. In Figure 5.10, { I, I, my } are coreference mentions in the dependency tree.

(2) Discovers the head concept c_h that aligns to the mentioned dependency nodes. If

- exactly one mentioned dependency node d_h is aligned to one concept, then this concept is selected as the head concept;
- multiple mentioned dependency nodes are aligned to different concepts, then the concept that aligns to the first mention is selected as the head concept;
- no mentioned dependency node is aligned to any concept, then the process is discarded.

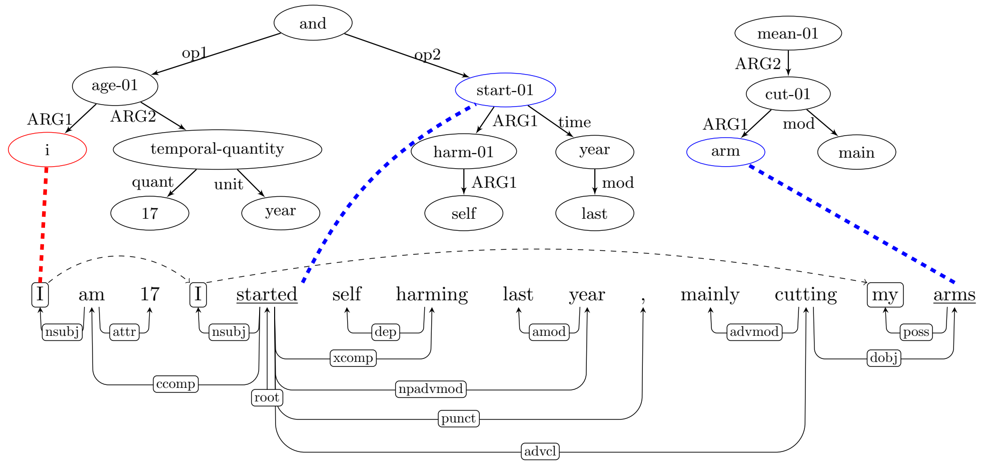


Figure 5.10: Sample of coreference mention attachment. The red line refers to the link between a coreference head to its corresponding concepts. The blue lines refer to the links between the parents of coreference mentions and their corresponding concepts

The head concept in Figure 5.10 is “i” in a red circle. The red line represents the link between c_h and d_h.

(3) For the mentioned dependency node other than d_h, we discover its parent concept, c_p, that covers the parent dependency node of this mention. This follows the same strategy as finding the head concept. The “start-01” and “cut” concepts are the parent concepts that cover the two mentions “I” and “my.”

(4) Attaches the reentrancy link between c_p and c_h with correct relation labels. In figure 5.10, we attach the “i” concept under “start-01” with “:ARG0” label, and attach the “i” concept under “arm” concept with “:part-of” label.

We will discuss the approach to find the appropriate relation labels in a subsequent section. The approach is identical to the way we identify the relation label between root concepts.

The last step for decoding is to link the root concepts to form a complete AMR graph. The goal is to discover the possible parent concepts for the root concepts one by one. Thus we want to start the search from the root concept that should be attached to the lower level of the graph. The last root concept in λ_{comp} naturally becomes the root of the whole AMR graph. To order the root concepts, we follow Equation 5.6 which is first designed for popping aligned concepts in CI. The type of concept that is at the beginning of the list is also to be processed earlier than other concept types. In case two root concepts own the same popping priority, we just follow the same order in λ_{comp}. Next, we extract the candidate concepts by which the root concept, c_{rootN}, should be linked. The candidates are acquired first from the dependency tree. We assume that c_{rootN} aligns to dependency node d_{rootN}. So the concepts that are aligned to the parent and grandparents of d_{rootN} are placed in the candidate list. Also, the existing root concepts in λ_{comp} are also placed on the candidate list. By giving the c_{root}, the candidate list C_{rootN}, and the model parameter θ, we greedily pick the relation label with the highest probability combinations with the following equation:

$$argN\bar{R}el_{root} = \underset{\substack{c \in C_{root} \\ a \in \{\textsc{Link-Arg}, \textsc{Link-Rel}\} \\ argN \in ARGN \\ rel \in REL}}{\operatorname{argmax}} \begin{cases} P_{act}(\textsc{Link-Arg}|s) * P_{argN}(argN|s) & \text{if } a = \textsc{Link-Arg} \\ P_{act}(\textsc{Link-Arg}|s') * P_{argN}(argN|s') & \text{if } a = \textsc{Link-Arg} \\ P_{act}(\textsc{Link-Rel}|s) * P_{rel}(rel|s) & \text{if } a = \textsc{Link-Rel} \\ P_{act}(\textsc{Link-Rel}|s') * P_{rel}(rel|s') & \text{if } a = \textsc{Link-Rel} \end{cases}$$

$$\text{where } s = ([], [c], [], [], c_{root}, \textsc{Forward-Move}, \emptyset)), s' = ([], [c_{root}], [], [], c, \textsc{Forward-Move}, \emptyset))$$

In these equations, we generate two "fake" states, s and s', to calculate the probability combinations for linking c_{rootN} and its concept in the candidate list. In s state, the concept in the candidate list is placed at the focusing concept, and c_{rootN} is placed at the top concept on λ_{in}. On the other hand, in s' state, c_{root} is placed at the focusing concept, and the concept in the candidate list is placed at the top on λ_{in}. Then the above equations search for the (action, child concept, argN/rel label) tuple with the highest probability. We also use these equations to decide the label between coreference mentions. The only difference is the mention has decided its parent concept already. Once we predict the relation label, we just follow the best tuples to link the concepts. If the best tuple comes from the s' state, we use the "inverse tag" on the relation label. Then we remove c_{rootN} from λ_{comp} and move to the next root concept. Post-processing is complete until the last root concept is in λ_{comp}.

Chapter 6

Experiments

In this chapter, we present the settings for and results of our experiments. To begin with, the design of our four experiments and the settings of the model's hyperparameters are presented in Section 6.1. After this, we address the evaluation of our sub-models: the Frame Identifier (Section 6.2), Concept Identifier (Section 6.3), and Relation Identifier (Section 6.4).

6.1 Experimental Settings

We design several experiments for the evaluation of our proposed models. For the Frame Identifier (FI) and Concept Identifier (CI) models, we compare our ReVNN-based identifier with two baseline models: the Window-Based and the Window+NN-Based model with various window sizes. The FI task is trained and evaluated on the OntoNotes 5.0 corpus rather than the DEFT AMR data because ON contains explicit frame IDs for target predicate words. In comparison, the CI task is trained and evaluated on the DEFT AMR data. We use the AMR-to-Dependency Parse aligner to generate the alignments between AMRs and dependency parses for the DEFT AMR data. Then the alignment pairs of AMR concepts and dependency nodes are converted to produce the training set. We evaluate our CI models by using the test set with manual alignments. The concepts extracted from the gold alignments in the test set are the gold standard concepts. We also evaluate our CI with LVC information. For our relation identifier, we design two experiments to evaluate the performance of our transition-based parser. First, we evaluate its performance based on action predictions. We also investigate the effectiveness of each feature. Second, we evaluate its performance based on the Smatch score and compare it with other AMR parsers.

To introduce linguistic features into our models, first we convert the features into vector representations (embeddings). The first sub-table in Table 6.1 lists the embeddings we used and their dimensions. The word embeddings are acquired from the Gigaword dataset[1] , OntoNotes, and DEFT AMR data. The Gigaword dataset contains an enormous number of documents from various domains: around ten million documents and more than four billion English words. We use Google Word2Vec[2] to train the word embeddings from these datasets. Further, the other embeddings are initialized randomly with real numbers between -1 and 1. All the embedding values are updated during training except pre-trained word embeddings.

During training and inference, several hidden layers output the intermediate vectors. For example, the output vector of the ReVNN sub-model (e_d) is reused and shared several times by our models. We list the dimensions of these hidden layers at the bottom left of Table 6.1. In all our models for FI and CI tasks, including Window-based, Window+NN-based, and ReVNN-based, a feedforward layer is placed before the final softmax or sigmoid layers of our classifiers. We set the output dimension of each feedforward layers to 500 (Top right in Table 6.1.)

We implement all these models in PyTorch[3] , a popular deep learning framework which supports dynamic neural network functions. In PyTorch, several primary and advanced neural networks, e.g., feedforward, LSTM, attention mechanism, etc., are provided. Moreover, the essential NN components, e.g., optimizer, automatic backpropagation, and loss functions, are ready to use. With PyTorch as an NN framework, it is easy to combine and bridge different networks and components into one model. Further, PyTorch's dynamic computation graphing aids the implementation of our ReVNN model since the number of dependency child nodes varies.

The hyperparameters for training our models are listed at the bottom right of Table 6.1. To train our models, we use Mini-batch stochastic gradient descent with mini-batch size 100. To speed-up mini-batch learning, we use the RMSprop (Tieleman and Hinton, 2012) algorithm to

[1] https://catalog.ldc.upenn.edu/ldc2012t21
[2] https://code.google.com/archive/p/word2vec/
[3] http://pytorch.org/

Table 6.1: The settings of the hyperparameters in our models of different tasks

Embeddings Dimensions

Embeddings	Variable	Dim.
Word	e^w	200
POS	e^{POS}	10
Dependency Relation Label	e^r	30
Aligning Type and Frame ID	e_f	20
Concept Type	e_{type}	11
Named Entity	e_{NE}	20
Frame Role Set	e_{role}	8
Semantic Role	e_{SR}	20
Concept Relation Label	e_{con-r}	20
Action	$e_{prevAct}$	10

Output Dimensions of Final Feed Forward Layers

Model	Sub-Model	Variable	Dim.
FI	Window		500
	Window+NN		500
	ReVNN	W^{FI}	500
CI - Basic	Window		500
	Window+NN		500
	ReVNN	W^{Basic}	500
CI - Other	Window		500
	Window+NN		500
	ReVNN	W^{Other}	500

Output Dimensions of Hidden Layers

Hidden Layer	Variable	Dim.
Concept	$e_{concept}$	500
ReVNN	e_d	300
Child Concept	e_{child}	300
Path	e_{path}	40

Other Hyperparameters

Name	Value
Mini-Batch Size	100
Learning Rate	10^{-4}
Smoothing Constant	0.99
Weight Decay	0.0

adjust the learning rate value α based on the number of epochs and training loss values.

Several features are adopted in our models. Table 6.2 illustrates feature usages between our models. In the FI task, all models use words, POS tags, and dependency relations as features, while in the CI tasks all models use this same feature set plus NEs, Frame IDs, and semantic roles as external features. The relation identifier uses all the feature embeddings and hidden layers listed in Table 6.1.

6.2 Frame Identifier

For the FI task, we evaluate our target models with the ON data set. First, we split 95% of the predicates with their frame IDs off as the training set and use the remaining 5% as the test set. Then we only pick the frames that contain multiple frame IDs as the target words. We remove the frames with single frame IDs. We compare our ReVNN-based model with three baseline models:

Table 6.2: Feature usages of all our models

Models	Word	POS	Dependency Relation	Named Entity	Frame ID	Semantic Role	ReVNN	Dependency Path
FI - Window	✓	✓	✓					
FI - Window+NN	✓	✓	✓					
FI - ClearNLP	✓	✓	✓					
FI - ReVNN	✓	✓	✓				✓	
CI - Window	✓	✓	✓	✓	✓	✓		
CI - Window+NN	✓	✓	✓	✓	✓	✓		
CI - ReVNN	✓	✓	✓	✓	✓	✓	✓	
Relation Identifier	✓	✓	✓	✓	✓	✓	✓	✓

the Window-based model and Window+NN-based model, which we describe in Section 5.3.2; the ClearNLP which use a linear model. The Window-based model takes the n nearest words of the target predicate as input feature words. These n nearest words then map to their feature vector representations and are concatenated to form the input vector before the softmax layer classifier. For the Window+NN-based model, one minor difference is that it uses two LSTMs to form the input features for the n nearby words before and after the target predicate. We also add a linear model based on Support Vector Machine. We use ClearNLP as the linear model, which identifies the frmae ID in its semantic role labeling model.

Table 6.3 lists the accuracy of different FI models. Our ReVNN model surpasses Window-based and Window+NN-based baseline models by 3.5% - 8.5%. Among the baseline models, the additional LSTM networks raise the accuracy around 3% from the Window-based model. Increasing the window size boosts accuracy by a small amount. And our ReVNN model also surpasses the ClearNLP model by 2.0%.

One of our objectives is to discover the differences in accuracy between frames with different occurrence frequencies. We first calculate the occurrence of all frames in ON 5.0. Observation shows that the distribution of frame frequency is extremely unbalanced. Around 38% of the frame occurrences belong to the top 10 frames. More than 71% of the occurrences belong to the top 250 frames. The low-frequency frame resembles a classic long tail distribution since 29% of the occurrences belong to the remaining 5,291 frames. A histogram diagram of frequency distributions excluding the top 10 frames is shown on the right of Figure 6.1.

Table 6.3: Accuracy of the Frame Identifier between comparison models. Accuracy among different frame ID frequencies also listed.

Model	Window Size	High Freq.	Med Freq.	Low Freq.	Total
Window	3	88.1	87.7	70.6	85.1
Window+NN	3	91.6	93.3	77.4	89.8
Window	5	88.2	87.8	72.5	85.5
Window+NN	5	91.8	93.6	77.9	90.1
ClearNLP		94.6	94.2	80.0	91.6
ReVNN		**95.7**	**95.2**	**83.5**	**93.6**

	Frame	Appear in ON	P	R	F_1
1	be.01	44,950	99.3	90.5	94.7
2	be.03	33,662	86.9	99.5	92.8
3	have.01	16,633	98.8	100.0	99.4
4	say.01	14,492	91.8	100.0	95.7
5	do.01	8,801	97.9	100.0	98.9
6	have.03	6,562	91.5	99.6	95.4
7	be.02	3,841	98.3	76.5	86.0
8	do.02	3,697	86.9	97.3	91.8
9	know.01	3,047	93.2	85.1	89.0
10	think.01	2,921	100.0	100.0	100.0

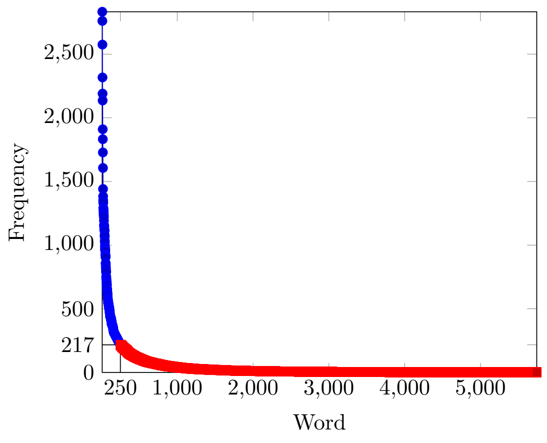


Figure 6.1: On the left, high frequency frames in ON 5.0 are listed with experimental results of identifying high frequency frames with our ReVNN model. On the right, the histogram shows the frequency distribution of frames belonging to medium- and low-frequency groups. Blue dots represent the medium-frequency group, while red dots represent the low-frequency group.

We divide frames into high-, medium-, and low-frequency groups. The frames in the high-frequency group and their frequencies are on the left of Figure 6.1. F_1 scores of all high-frequency frames in our ReVNN model are also presented. The top 10 to top 250 frames belong to the medium-frequency group. The gap frequency is 217. The remaining frames belong to the low-frequency group.

Table 6.4 presents the contributions of different features individually. POS helps to increase

Table 6.4: Accuracy of the Frame Identifier based on feature usages.

Model	Window Size	Word	+POS	+ Dependency Relation
Window	3	80.8	83.3	85.1
Window+NN	5	86.7	88.6	89.8
Window	3	82.5	83.09	85.5
Window+NN	5	87.7	88.6	90.1
ClearNLP		89.2	90.8	91.6
ReVNN		91.3	92.7	93.6

overall accuracy by around 1.5%, while dependency relations help to increase accuracy by about 1.2%. We can conclude that these lexical and syntactic features both aid the FI task.

6.3 Concept Identifier

In this section, we present the experimental results of our Concept Identifiers over the DEFT AMR data. To train our models, we use the training set from the DEFT data, which contains 39,260 gold AMR graphs. We apply our data pre-processing step (Section 5.3.3) to our training and test data. We then run our dependency parse to AMR aligner on the training data. The resulting dependency parse to AMR alignments are the inputs of our CI models. At test time, we evaluate on the test data set which contains 500 dependency parse - AMR pairs with gold standard alignments, described in Section 4.5.1.

When given a dependency tree node d, the set of concepts that align to d represents the gold standard concepts that we want to extract from our CI model. We design two classifiers, “Basic” and “Other,” which generates zero to many concept candidates when giving the target dependency node. When no concept is generated, i.e., the “Basic” classifier outputs a “non-exist” label. When all the output probabilities of the “Other” classifier are lower than 0.5, we use a special tag “Delete” as the output concept of the target dependency node. Otherwise, we compare the resulting concepts to the gold standard concepts which align to the target dependency node and calculate the F_1 score. The baseline models are still the Window-based and Window+NN-based models with different window sizes. All these underlying models generate the vector representation

Table 6.5: F_1 score of Concept Identifier based on feature usages.

Model	Window Size	Word +Frame ID +Semantic Role	+POS	+Dependency Relation	+NE
Window	3	73.8	75.4	77.1	78.3
Window+NN	5	77.4	78.8	80.2	81.1
Window	3	73.5	75.1	76.8	77.6
Window+NN	5	77.2	78.8	79.5	80.2
ReVNN		78.2	81.3	81.8	82.2

of the target dependency node and then pass it to the two classifier layers, FF_{Basic} and FF_{Other}.

Table 5.3 presents the F_1 scores of all CI models. Our ReVNN-based model surpasses the best Window+NN-based model by 1.1% F_1 score. An interesting finding is that the increase in window size in the baseline models does not improve the F_1 score for the CI task. Instead, the F_1 score drops 0.7% - 0.9%. We categorize the resulting concepts based on the category types we describe in Table 5.2. We evaluate the F_1 score based on the corresponding concept types. The results of the concept types we are interested in are listed in Table 5.3 as well. The F_1 score of the predicate concept is relatively high, reaching 91.2% with the ReVNN model. However, the generation of the date entity concept does not turn out well, reaching only 70.0% with the ReVNN model. On the other hand, for the attributes, the Wiki type works well. However, the mode attribute type is extremely challenging to predict. The best F_1 score only approaches 16.4%.

Table 6.5 lists the contributions of individual features. Our model uses words, frame IDs and semantic roles as features at first. Then, the rest of the feature set is added one by one. The adding of features behaves similarly to the adding of features in the FI task. All features contribute positively to the F_1 score. Although the improvements from adding the POS and the dependency relation features are minor in the ReVNN model, they still aid performance in the baseline model. The named entity feature improves the F1 score in all models as well.

To evaluate the effectiveness of LVC in the CI task, we also evaluate our ReVNN with and without LVC information from our LVC detector (Chapter 3). Results are also shown in Figure 6.2. We discover that by adding LVC information to our ReVNN-based CI, the overall F_1 score increases

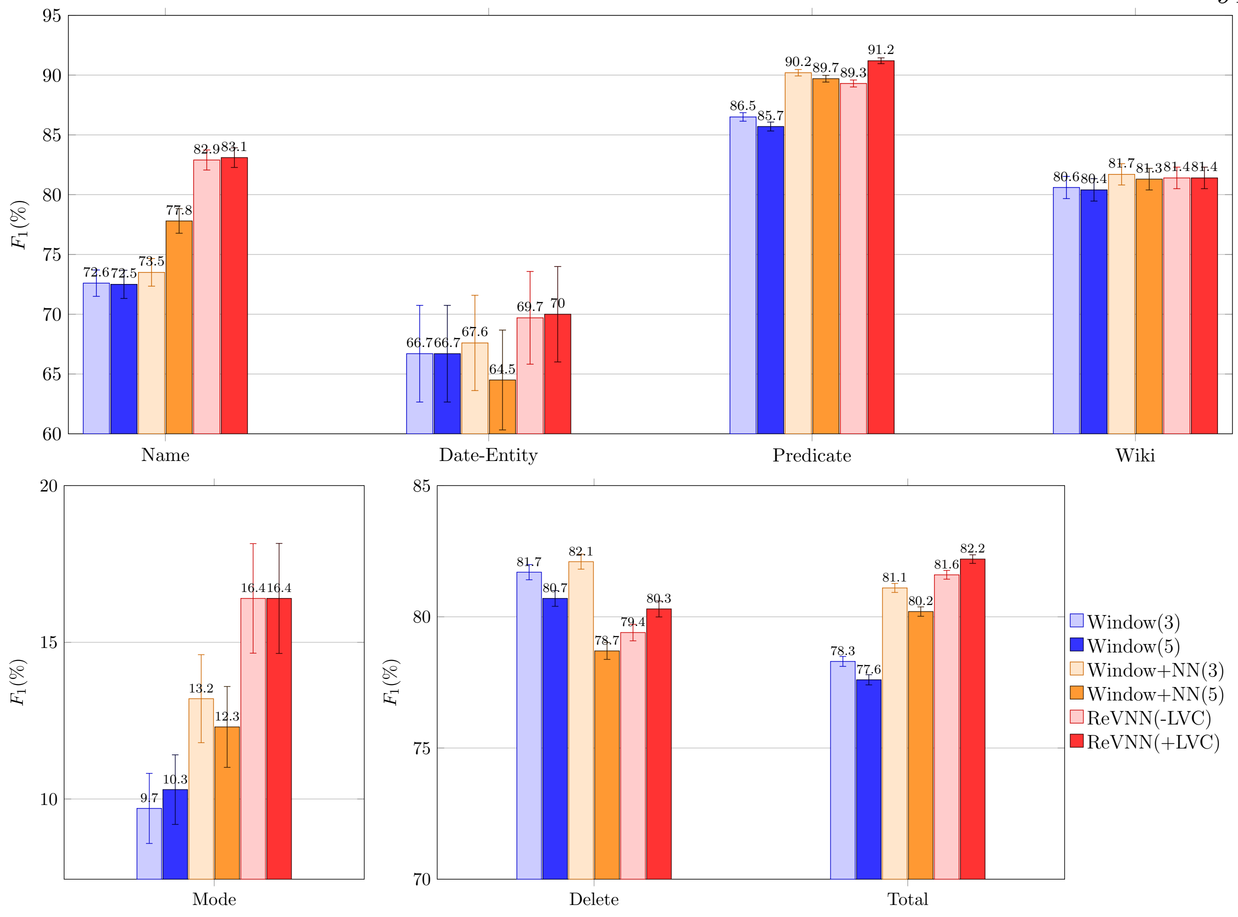


Figure 6.2: The F_1 score of the Concept Identifier between comparison models, and the F_1 score of our ReVNN Concept Identifier with and without LVC information. The score among different concept types is also listed.

0.6%. Also, the individual concept types also benefit from the LVCs, especially the predicate type concept. Both the prediction of the predicate type concept and the "Delete" label raise their F_1 scores, 1.9%, and 0.9%, respectively. The improvement of other concept types is insignificant. When we only compare the sentences that contain LVCs, the improvement to predicate identification is around 5.0%.

6.4 Transition-Based Parser

In this section, we evaluate our transition-based parser from two perspectives. First, we evaluate the transition-based parser based on the performance of the transition, argN, and relation classifier. Second, we evaluate our parser based on Smatch score, an evaluation metric for parsing AMR graphs.

6.4.1 Action Identifier

We start the evaluation of the transition-based parser based on the performance of three classifiers: transition, argN, and relation. Our ReVNN-based model generates the underlying layer h_{state} which contains the feature representations of the current parsing state. h_{state} is then passed to the three output feedforward layers: z_{action}, z_{arg}, and z_{rel}. We use the training set of DEFT AMR data to train our parser. The test set is composed of DEFT AMR test data with manual alignments between AMR concepts and dependency nodes. The experimental data is submitted to the oracle action extractor (Section 5.3.5.2) to receive a list of oracle actions, argNs, and relation labels with their corresponding parsing states. These labels and states are the inputs when training our parser. In addition to these regular inputs, we also add some artificial inputs to deal with the input order of the child concept module. Although the input order of the linking of child concepts is nevertheless important during training, our child concept module still generates different child concept hidden vectors (e_{child}) by giving different child concept orders. Thus, we add the artificial training instances, which randomly changes the input orders of the real child concepts and keeps the other states. The addition of these artificial training instances increases the flexibility of our parsing model.

Table 6.6 lists the accuracies of these three classifiers and the F_1 scores of each action label in different epochs. These values are tested on the model trained with all features described in Section 5.3.5.3. The first thing to point out is that more epochs of training do not imply better performance when training our model. The action and argN classifiers reach their best accuracies at the 60th epoch with 79.3% and 78.1%, respectively, while the relation classifier reaches its best 69.5% at the 80th epoch. F_1 scores of each action labels also show that they reach the best scores at the 60th epoch. Accuracies drop a little when we train our model with more epochs. Second, the relation classifier is more difficult to train since the number of relation labels is far more than

Table 6.6: The left part of the table shows F_1 scores of each action labels. This model trains with the whole feature set. The accuracies of the three classifiers between different epochs are also listed on the right of the table.

Epoch	Forward-Move	Link-Arg	Link-Rel	Swap	Move-Complete	Action Avg.	ArgN Avg.	Rel. Avg
	(in F_1 score)					(in accuracy)		
30	83.4	78.0	**78.6**	53.5	64.5	79.2	75.6	62.6
60	**83.6**	**78.7**	78.1	**54.1**	64.5	**79.3**	**78.1**	68.0
80	83.4	77.5	77.8	53.4	**64.6**	78.9	77.1	**69.5**
100	82.9	76.4	77.9	52.3	63.9	78.5	77.4	68.6

the action and argN labels. Also, predictions for Swap and Move-Complete actions are more difficult than the predictions for other action labels. For Swap, the classifier needs to compare the structures of the parent concept with the child concept and decide whether to switch them or not. Swap happens only in a few situations. For Move-Complete, the action classifier needs to know that the target concept has already linked all child concepts as well as recognize itself as a root concept. Move-Complete also appears rarely in our oracle action lists. On the other hand, the Forward-Move, Link-Arg, and Link-Rel are somewhat easy to predict. Their F_1 scores are higher than 77%.

The incremental contributions of each feature are presented in Table 6.7. Results are tested on the model trained at the 60-th epoch. Each feature listed improves the accuracy of the three classifiers. Results show that the fundamental lexical and semantic features, i.e., word, POS, and dependency relation, provide minor but stable improvements. Furthermore, the semantic roles and dependency path boost the accuracy from 1.0% to 2.5%, even though the underlying parser used other linguistic features. All three classifiers reach their best accuracy when adding all features. Figure 6.3 provides the comprehensive view of the accuracies of the three classifiers with different feature combinations at different epochs. The graphs are consistent with the view that additional epochs of training after the 60-th epoch do not obtain better performance on all three classifiers. Moreover, the adding of any features improves the accuracy by certain amounts.

To further evaluate the argN and relation classifiers, we present the confusion matrices of

Table 6.7: Incremental feature contribution of accuracy on Action, ArgN, and Relation tasks at the 60th epoch.

Task	Word	+POS	+Dependency Relation	+NE	+Semantic Role	+Dependency Path
Action	73.9	74.2	76.4	76.5	77.7	**79.3**
ArgN	70.2	71.8	74.4	75.7	77.1	**78.1**
Relation	59.2	59.6	62.9	64.1	66.5	**68.0**

these two classifiers in Table 6.8. For the confusion matrix of the argN labels, "ARG1" is the top majority core label with more than half of the core argument occurrences. The identification of "ARG2" labels is suboptimal. One third of "ARG2" gold labels are recognized as "ARG1" by our argN classifier. For the confusion matrix of the top 11 relation labels, ":mod" is used most frequently. We note that the classifiers do not disambiguate ":mod" from ":quant", ":degree", and ":location" well. Also, ":manner" and ":quant" are difficult to disambiguate.

6.4.2 AMR Parser

The final experiment we design is to apply the whole parsing pipeline, including FI, CI, oracle action extractor, decoding, and post-processing, to generate AMR graphs from sentences in the test set. The DEFT AMR 2016 dataset is our experimental data which contains various genres of corpora and AMRs, including newswires, TV broadcast scripts, web blogs, machine translation

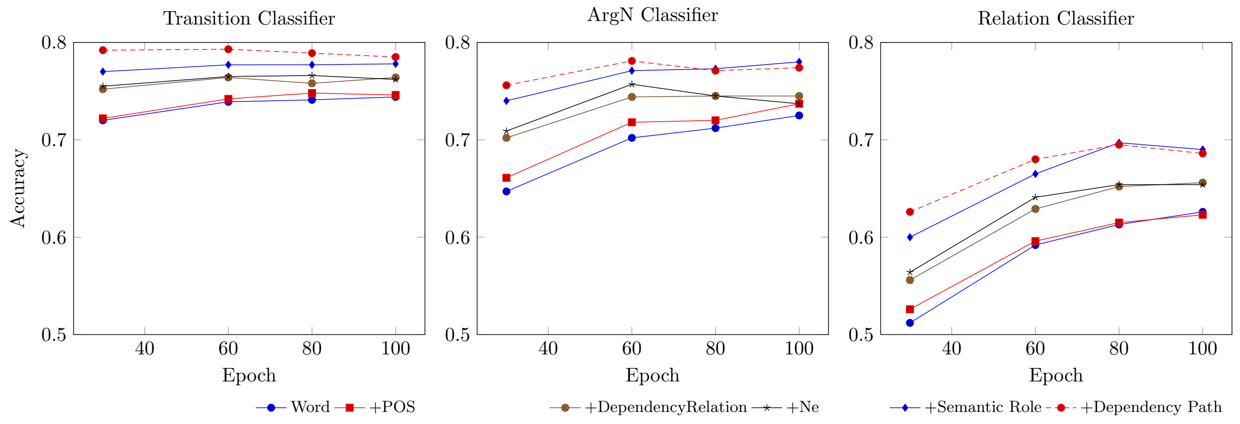


Figure 6.3: Accuracies of Transition, ArgN, and Relation classifier using incremental feature addition. The X-axis represents the number of epochs in training

Table 6.8: Confusion matrices of ArgN labels (left) and Rel labels in the top 11 majority (right)

		Predict ArgN				
		ARG0	ARG1	ARG2	ARG3	ARG4
Gold ArgN	ARG0	504	134	57	4	0
	ARG1	81	1422	93	9	0
	ARG2	39	193	367	3	0
	ARG3	6	11	13	19	0
	ARG4	0	7	8	0	4

		Predict Rel										
		:mod	:name	:time	:op	:quant	:degree	:location	:domain	:unit	:condition	:manner
Gold Rel	:mod	494	0	22	12	15	27	1	8	3	2	3
	:name	0	302	0	0	2	0	0	0	0	0	0
	:time	16	0	209	0	3	2	6	0	1	7	1
	:op	10	1	2	173	15	0	0	5	0	0	1
	:quant	28	2	1	7	95	10	1	2	3	0	23
	:degree	29	0	1	0	4	96	0	2	1	0	1
	:location	22	0	8	10	3	2	65	0	0	3	0
	:domain	10	0	4	3	0	0	0	60	0	0	0
	:unit	2	0	0	1	0	0	0	2	63	0	0
	:condition	4	0	8	3	0	0	0	0	0	26	2
	:manner	17	0	6	1	0	1	0	0	0	3	20

Table 6.9: Data split by domain from the LDC DEFT AMR corpus and the SemEval Blind Evaluation Data.

Dataset	Training	Dev	Test	Blind Evaluation
BOLT Discussion Forum MT	1,061	133	133	0
Broadcast Conversation	214	0	0	0
Weblog and WSJ	0	100	100	0
BOLT Discussion Forum English	6,455	210	229	0
DEFT Discussion Forum English	19,558	0	0	257
Guidelines AMRs	819	0	0	0
2009 Open MT	204	0	0	0
Proxy reports	6,603	826	823	0
Weblog	866	0	0	232
Xinhua MT	741	99	86	18
Agence France-Presse news	0	0	0	23
Associated Press news	0	0	0	52
New York Times news	0	0	0	471
Totals	36,521	1,368	1,371	1,053

evaluation text, etc. We use the same training and test set as previous CI and action identifier experiments. In addition to the test set used in previous experiments, a blind evaluation set from the SemEval 2016 task is provided as additional test data. Comparing this blind evaluation set with the original test set, the blind one contains more sentences from different genres. Half of the sentences are from discussion forums, and the other half from newswire. The domain distribution of the blind dataset is different from the distribution of the training dataset. The data split by domain of experimental data is listed in Table 6.9.

Table 6.10: Incremental feature contribution to Smatch scores at the 60th epoch on our transition-based parser

Epoch	Word	+POS	+Dependency Relation	+NE	+Semantic Role	+Dependency Path
30	61.3	62.1	64.1	65.3	67.6	69.2
60	63.4	**64.3**	65.6	66.1	**68.6**	**69.6**
80	**63.9**	64.0	65.5	**66.8**	68.5	**69.6**
100	63.8	63.4	**65.9**	66.0	**68.6**	69.0

Table 6.10 presents the incremental feature contributions of Smatch scores at different epochs. These scores are evaluated on the DEFT test set. Results show that every feature contributes a certain amount to Smatch scores. Generally speaking, semantic role features provide the most Smatch score improvements among all features. Syntactic features, i.e., dependency relations and dependency paths, also provide substantial contributions to the system. As we change our attention to the number of epochs, we see that after the 60th epoch, the improvement of the Smatch score becomes insignificant. After the 80th epoch, Smatch scores start to drop. The variety of Smatch scores based on the feature combinations and based on the epochs are consistent with the variety of action identification accuracy described in the previous section. The scores in Table 6.10 are also illustrated on the left of Figure 6.4 as the ling graph.

We are also interested in the relation between Smatch score and the length of test sentence. In right Figure 6.4, we draw a line graph to illustrate the association between average Smatch scores and sentence lengths. Like other parsing tasks, e.g., dependency parsing and relation extraction, performance drops dramatically with longer sentences. Average Smatch scores for sentences shorter than 20 words are above the average score. Otherwise, average Smatch scores for sentences longer than 20 words are lower than average. Sentences longer than 80 words reach the lowest Smatch score at 0.605. Discovering the relations between two concepts with long distance dependencies is always difficult, especially as our parser is based on the dependency parse hierarchy. Automatic dependency parsers still have trouble dealing with long distance relations. The same difficulty affects our AMR parser as well.

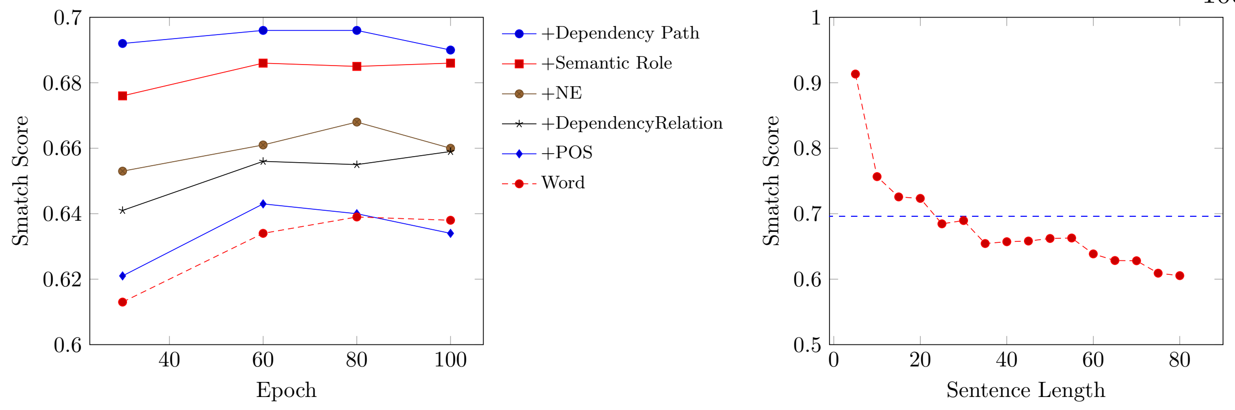


Figure 6.4: Smatch scores of our parser with different feature combinations (left) and with different sentence lengths (right). In the right figure, the model is trained with all available features at the 60th epoch. The blue dotted line represents the average Smatch score (69.6).

Table 6.11: Comparison of different AMR parsers in SemEval test and blind evaluation dataset. The systems marked with * are transition-based parsers.

System	Test Set	Blind Evaluation Set
CAMR*	66.5	62.0
RIGA *	67.2	62.0
JAMR (2016)	67.0	56.0
CAMR(2017)*	68.1	63.6
CU-NLP	70.7	65.1
Our parser	69.6	64.2

One last step is to compare our parser to different AMR parsing systems. We list comparable systems and their results in Table 6.11. The CAMR system (Wang et al., 2015a) is a transition-based parser which uses dependency parses as inputs. They proposed an updated version as CAMR (2017) (Wang and Xue, 2017), which improves their concept identifier with a bi-directional LSTM and character-based encoding. RIGA (Barzdins and Gosko, 2016) is another transition-based system which inherits from the CAMR system. CAMR and RIGA are the two winners in the SemEval 2016 AMR parsing task. On the other hand, JAMR (2016) is the revision of the very first JAMR AMR parser (Flanigan et al., 2014), which uses the MSCG algorithm for parsing. Foland and Martin (2016; 2017) proposed their latest version of the CU-NLP system, which uses a bidirectional LSTM and an improved concept identifier to develop their parsing system. CU-NLP is currently the state-of-the-art system. Results show that our system reaches the best Smatch score among all the transition-based systems in both test and blind evaluation datasets. Our system is 1.0% lower in Smatch score than the CU-NLP system. The Smatch score of parsing sentences from blind evaluation data drops 5.0% in average.

6.4.3 Error Analysis

To further understand the advantages and the disadvantages of our AMR parser, we randomly select 50 sentences from the DEFT AMR test set and manually compare our AMR graphs with the gold standard. The list of the selected sentences with the gold standard AMRs and the AMRs produced by our parser is presented in Appendix B. The AMR graphs on the left are the gold standard AMRs, while the graphs on the right are the parser AMRs. We categorize the errors into three different types:

- **Semantic Error**: Semantic error refers to producing incorrect concepts that represent different semantic meanings, i.e., missing nodes, inaccurate label, etc. For example, the AMR with ID DF-199-194215-653_0484.15 shows that our parser picks "threaten-01" as an *ARG1* of "cave-01", while in the gold standard AMR, "you" is the correct *ARG1* of "threaten-01". The reason for this problem might be semantic role AM-CAU (Agent/-Causer) on the phrase "to his threats". Thus our parser links "cave-01" to "threaten-01" with an *ARG1* label, missing the implicit "you" argument. Another example of semantic error is connecting two concepts with an incorrect label. For example, in the same graph, our parser misses the LVC for "right-05", when our parser recognizes "right" as a basic attribute "right". Around 42% of the errors belong to the semantic error category.

- **Structural Error**: Structural errors involve the graph structure, including incorrect placement in DAG, missing arguments, and missing frames. For example, in the AMR ID DF-200-192400-625_6304.5, "more" should modify "(p/point)". However, our parser puts it under "(c /contrast-01)". Also, in the AMR ID DF-200-192400-625_6304.3, our parser misses the *ARG0* of “have-org-role-91”. Structural errors most likely occur because our transition-based parser predicts the wrong action during decoding. Our concept identifier generates the correct concept candidates. However, our parser does not put them in the correct position on the graph. Around 41% of the errors belong to the structural error category.

- Frame Error: Frame error means the graph contains duplicated core arguments or incorrect frame assignments. For example, in the AMR ID DF-199-194215-653_0484.17, there are two “*ARG0*”s under concept “say-01”. Also, in AMR ID DF-200-192400-625_6304.24, our parser mislabels two frames with concept “normalize-02” and “get-03”. This type of error is due to our frame identifier and concept identifier producing correct concept candidates. Around 27% of the errors belong to the frame error category.

Besides, there are some errors that come from the wrong dependency parse trees, semantic roles, coreference chains, or wiki-links. However, generally speaking, most of the errors are due to our action identification which produces incorrect structures. Our frame identifier and concept identifier predicting wrong concepts are also another primary reason, and the insertion of unnecessary “condition” concepts.

Chapter 7

Conclusion

7.1 Contributions

This dissertation focuses on a pipeline of parsing AMR graphs from raw English sentences. Among all fundamental and essential modules, we discover that basic linguistic information, including lexical and syntactic features, still play important roles in the semantic parsing task. For the LVC detector, new rules and heuristics based on syntactic relations are introduced to select LVC candidates. A set of features, including lexical features, word senses, and WordNet features, are presented to aid the LVC detector. The SVM-based LVC detector allows us to identify LVCs more reliably and with more accuracy than comparable models. Our model gives a significant improvement in F_1 scores in the detection of LVCs compared to other models. LVCs are also shown to support our AMR parsing task. For the dependency parse to AMR aligner, a list of basic matching types is defined, covering approximately 60% of the aligning pairs between dependency nodes and AMR concepts. Our EM-based algorithm, which estimates individual feature probabilities, improves the performance of aligning dependency nodes to AMR concepts. The more accurately aligned pairs help improve AMR parsing Smatch scores. For our AMR parser, several sub-models are designed to build our parser. The Rev-Neural Network as our underlying structure is introduced and shown to generate syntactic-rich and more robust vector representations of dependency parse nodes than window-based models. For the Frame Identifying task, our ReVNN-based Frame Identifier shows significant improvements on all frequency groups of frames against the Window-based and Window+NN-based models. The addition of individual features contributes consistently to the accuracy of frame identification. For the Concept Identification task, our ReVNN-based Concept Identifier shows improvements in accuracy in the identification of different concept types. The addition of individual features also improves the accuracy of concept identification. Moreover, by adding LVC information to the model, the CI model demonstrates a small but stable boost in accuracy. For action identification, our transition-based parser extracts oracle actions in a linear relationship to input sentence length. For the AMR parsing task, our ReVNN transition-based parser shows improvement of accuracy on action and argument identification. The addition of individual features also contributes to action identification. For the parsing, our ReVNN-based AMR parser reaches the highest Smatch score compared to other transition-based AMR parsers.

To the best of our knowledge, this is the first time that these AMR parsing components have been applied to the ReVNN-based model. Our results for identifying frames, concepts, and actions show that the ReVNN model gives helpful feedback for many NLP tasks, e.g., word sense disambiguation and relation identification, tasks which depend heavily on syntactic structures and long-distance word relations. Additionally, the ReVNN model is a great "container" which can integrate different types of linguistic knowledge and features into a single structure. The three identifiers in our AMR parser show that the addition of individual features steadily increases the performances of our classifiers, even when the characteristics of these features are diverse. Particularly, semantic roles and dependency paths are the two main features added last to our feature set, yet contributing the most in our transition-based parser. The later a feature is added to a feature set, in general, the less it contributes to accuracy. The increase of accuracy when adding semantic role and dependency path features to our ReVNN model shows that the ReVNN model is still benefited by very different features for different kinds of tasks. Above all, our ReVNN-based parser can be extended to implement other transition-based semantic parsing tasks, e.g., first-order logical form and Universal Conceptual Cognitive Annotation (UCCA) (Hershcovich et al., 2017), which can be parsed and derived from dependency parse structures. These semantic parsing tasks also need to adopt different linguistic information as input features, the advantage of the ReVNN model.

7.2 Future Work

There is still much room for improvement. First, no oracle optimal transition algorithm is performed in our oracle action extractor. Consequently, the action extractor might generate a sequence of actions with extra actions to form the final AMR graph, or even make the final AMR graph unreachable. We intend to apply a dynamic oracle (Goldberg and Nivre, 2012), which provides transitions that lead to the best reachable graph from the given state, to our transition-based parser. Improvement to the oracle action extractor will increase parsing performance. Second, we plan to integrate a sequence-to-sequence model into our transition-based parser. Sequence-to-sequence models have achieved great success in machine translation. The encoder receives entire source sentences and generates intermediate vectors which represent sentence meaning and parsing states. The decoder receives intermediate vectors and source words as inputs and outputs corresponding translated words. An attention mechanism is introduced to link the source word and corresponding translated word during parsing. As such, the attention mechanism is an alternative design for word alignment. Inspired by this idea, we can apply a sequence-to-sequence model with attention to our AMR parser framework. Such encoders usually use multi-layers of recurrent neural networks, i.e., LSTMs or gated recurrent units, to generate intermediate vectors from input words. We plan to change this underlying RNN layer to our ReVNN-model which we believe can better represent sentence meaning. Furthermore, our dependency parse to AMR aligner can assist the attention mechanism. Previous attempts to design the sequence-to-sequence based AMR parser (Peng et al., 2017; Barzdins and Gosko, 2016; Konstas et al., 2017) ran into the problem of insufficient training data. Because of this, the attention mechanism cannot perform as expected. By integrating our aligner with attention, we believe the resulting sequence-to-sequence model can improve parsing results.

We also plan to further observe the intermediate vectors on each dependency tree node generated by the ReVNN model. The ReVNN model integrates linguistic features to dependency tree nodes, which can also represent the phrase that it covers. After experimenting with different models and different feature combinations, 3 of the best and most distinctive can always be combined into an ensemble approach to improve performance. Resulting vectors are potentially better units for the representation of the meaning of corresponding phrases. One direction is to apply phrase vector representations to a paraphrase identification task since the resulting vectors of paraphrase pairs are ideally similar to each other. Another direction is to provide distinct vector representations based on word senses. Current word embedding approaches treat individual words with different word senses as the same vector representation. The ReVNN model generates different vector representations for distinct word senses based on dependency structures. We believe such vector representations will support many NLP applications. Last but not least, we can apply our AMR parses and the ReVNN model to other NLP applications, e.g., textual entailment and natural language inference. These NLP applications require rich semantic representations of the sentences. Thus we can apply our ReVNN model for semantic parsing. Instead of using dependency parse trees as the input graph, we can use AMRs that our model produces as the input structures of the ReVNN model.

Bibliography

Artzi, Y., Lee, K., and Zettlemoyer, L. (2015). Broad-coverage ccg semantic parsing with amr. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1699–1710, Lisbon, Portugal. Association for Computational Linguistics.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The berkeley framenet project. In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1, ACL '98, pages 86–90, Stroudsburg, PA, USA. Association for Computational Linguistics.

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.

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. CoRR, abs/1604.01278.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. Comput. Linguist., 19(2):263–311.

Bunescu, R. C. and Mooney, R. J. (2005). A shortest path dependency kernel for relation extraction. In Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, HLT '05, pages 724–731, Stroudsburg, PA, USA. Association for Computational Linguistics.

Butt, M. (2003). The Light Verb Jungle. Harvard Working Papers in Linguistics, 9: 1–49.

Cai, S. and Knight, K. (2012). Smatch: an evaluation metric for semantic feature structures. submitted.

Carreras, X. and Màrquez, L. (2005). Introduction to the conll-2005 shared task: Semantic role labeling. In Proceedings of the Ninth Conference on Computational Natural Language Learning, CONLL '05, pages 152–164, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chen, D. and Manning, C. (2014). A fast and accurate dependency parser using neural networks. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Chen, W.-T. (2015). Learning to map dependency parses to abstract meaning representations. In Proceedings of the ACL-IJCNLP 2015 Student Research Workshop, pages 41–46, Beijing, China. Association for Computational Linguistics.

Chen, W.-T., Bonial, C., and Palmer, M. (2015). English light verb construction identification using lexical knowledge.

Chen, W.-T. and Palmer, M. (2017). Unsupervised amr-dependency parse alignment. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 558–567. Association for Computational Linguistics.

Cheng, X. and Roth, D. (2013). Relational inference for wikification. In EMNLP.

Choi, J. D. and Mccallum, A. (2013). Transition-based dependency parsing with selectional branching. In In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics.

Choi, J. D. and Palmer, M. (2011). Transition-based semantic role labeling using predicate argument clustering. In Proceedings of the ACL 2011 Workshop on Relational Models of Semantics, RELMS '11, pages 37–45, Stroudsburg, PA, USA. Association for Computational Linguistics.

Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. Comput. Linguist., 16(1):22–29.

Clark, K. and Manning, C. D. (2016). Improving coreference resolution by learning entity-level distributed representations. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 643–653, Berlin, Germany. Association for Computational Linguistics.

Clear, J. H. (1993). The digital word. chapter The British National Corpus, pages 163–187. MIT Press, Cambridge, MA, USA.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th International Conference on Machine Learning, ICML '08, pages 160–167, New York, NY, USA. ACM.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B, 39(1):1–38.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1370–1380, Baltimore, Maryland. Association for Computational Linguistics.

Dligach, D. and Palmer, M. (2011). Good seed makes a good crop: Accelerating active learning using language modeling. In Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, OR.

Do, Q. N. T., Bethard, S., and Moens, M. (2017). Improving implicit semantic role labeling by predicting semantic frame arguments. CoRR, abs/1704.02709.

Duffield, C., Hwang, J., Brown, S., Viewig, S., Davis, J., and Palmer, M. (2007). Criteria for the manual grouping of verb senses. In Proceedings of the Linguistic Annotation Workshop, Prague.

Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. Comput. Linguist., 19(1):61–74.

Elman, J. L. (1990). Finding structure in time. COGNITIVE SCIENCE, 14(2):179–211.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research, 9:1871–1874.

FitzGerald, N., Täckström, O., Ganchev, K., and Das, D. (2015). Semantic role labeling with neural network factors. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP '15).

Flanigan, J., Thomson, S., Carbonell, J., Dyer, C., and Smith, N. A. (2014). A discriminative graph-based parser for the abstract meaning representation. In Proc. of ACL, Baltimore, Maryland. Association for Computational Linguistics.

Geoffrion, A. (1974). Lagrangian relaxation for integer programming. mathprogstud, 2:82–114.

Gerber, M. and Chai, J. Y. (2012). Semantic role labeling of implicit arguments for nominal predicates. Computational Linguistics, 38(4):755–798.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. Comput. Linguist., 28(3):245–288.

Goldberg, Y. and Nivre, J. (2012). A dynamic oracle for arc-eager dependency parsing. In COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India, pages 959–976.

Gurrutxaga, A. and Alegria, I. (2012). Measuring the compositionality of nv expressions in basque by means of distributional similarity techniques. In LREC, pages 2389–2394.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task, CoNLL '09, pages 1–18, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hashimoto, K., Miwa, M., Tsuruoka, Y., and Chikayama, T. (2013). Simple customization of recursive neural networks for semantic relation classification. In EMNLP, pages 1372–1376. ACL.

Henderson, J., Merlo, P., Musillo, G., and Titov, I. (2008). A latent variable model of synchronous parsing for syntactic and semantic dependencies. In Proceedings of the Twelfth Conference on Computational Natural Language Learning, CoNLL '08, pages 178–182, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hermann, K. M., Das, D., Weston, J., and Ganchev, K. (2014). Semantic frame identification with distributed word representations. In Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics.

Hershcovich, D., Abend, O., and Rappoport, A. (2017). A transition-based directed acyclic graph parser for ucca. In Proc. of ACL, pages 1127–1138.

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

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). OntoNotes: the 90% solution. In Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers, NAACL-Short '06, page 5760, Stroudsburg, PA, USA. Association for Computational Linguistics.

Johansson, R. and Nugues, P. (2008). Dependency-based semantic role labeling of propbank. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '08, pages 69–78, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kambhatla, N. (2004). Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions, ACLdemo '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1, ACL '03, pages 423–430, Stroudsburg, PA, USA. Association for Computational Linguistics.

Klyueva, N., Doucet, A., and Straka, M. (2017). Neural networks for multi-word expression detection. In Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Workshop on Multiword Expressions, page 6, Valencia, Spain.

Konstas, I., Iyer, S., Yatskar, M., Choi, Y., and Zettlemoyer, L. (2017). Neural AMR: sequence-to-sequence models for parsing and generation. CoRR, abs/1704.08381.

Lee, Y. K. and Ng, H. T. (2002). An empirical evaluation of knowledge sources and learning algorithms for word sense disambiguation. In Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10, EMNLP '02, pages 41–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

Levy, O. and Goldberg, Y. (2014). Dependency-based word embeddings. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 2: Short Papers, pages 302–308.

Luong, M., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2015). Multi-task sequence to sequence learning. CoRR, abs/1511.06114.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1994). Building a large annotated corpus of english: the penn treebank. Computational Linguistics, 19.

Matthiessen, C. and Bateman, J. (1991). Text generation and systemic-functional linguistics: experiences from English and Japanese. Communication in artificial intelligence. Pinter.

May, J. (2016). Semeval-2016 task 8: Meaning representation parsing. In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016), pages 1063–1073, San Diego, California. Association for Computational Linguistics.

McCullagh, P. and Nelder, J. A. (1989). Generalized linear models (Second edition). London: Chapman & Hall.

Melamud, O., Levy, O., and Dagan, I. (2015). A simple word embedding model for lexical substitution. In Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing, pages 1–7, Denver, Colorado. Association for Computational Linguistics.

Mihalcea, R. and Csomai, A. (2007). Wikify!: Linking documents to encyclopedic knowledge. In Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM '07, pages 233–242, New York, NY, USA. ACM.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. CoRR, abs/1310.4546.

Miller, G. A. (1995). Wordnet: a lexical database for english. Communications of the ACM, 38(11):39–41.

Misra, D. and Artzi, Y. (2016). Neural shift-reduce ccg semantic parsing. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1775–1786. Association for Computational Linguistics.

Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. Comput. Linguist., 34(4):513–553.

Palmer, M., Guildea, D., and Kingsbury, P. (2005). The Proposition Bank: An annotated corpus of semantic roles. Computational Linguistics, 31(1):71–105.

Peng, X., Wang, C., Gildea, D., and Xue, N. (2017). Addressing the data sparsity issue in neural AMR parsing. CoRR, abs/1702.05053.

Pourdamghani, N., Gao, Y., Hermjakob, U., and Knight, K. (2014). Aligning english strings with abstract meaning representation graphs. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 425–429. Association for Computational Linguistics.

Pradhan, S., Hacioglu, K., Ward, W., Martin, J. H., and Jurafsky, D. (2005). Semantic role chunking combining complementary syntactic views. In Proceedings of the Ninth Conference on Computational Natural Language Learning, CONLL '05, pages 217–220, Stroudsburg, PA, USA. Association for Computational Linguistics.

Pradhan, S., Hovy, E., Marcus, M., M.Palmer, Ramshaw, L., and Weschedel, R. (2007). Ontonotes: a unified relational semantic representation. In Proceedings of the First IEEE International Conference on Semantic Computing(ICSC-07), Irvine, CA.

Pradhan, S. S., Ward, W., and Martin, J. H. (2008). Towards robust semantic role labeling. Comput. Linguist., 34(2):289–310.

Pust, M., Hermjakob, U., Knight, K., Marcu, D., and May, J. (2015). Using syntax-based machine translation to parse english into abstract meaning representation. CoRR, abs/1504.06665.

Ratinov, L., Roth, D., Downey, D., and Anderson, M. (2011). Local and global algorithms for disambiguation to wikipedia. In ACL.

Savary, A., Ramisch, C., Cordeiro, S., Sangati, F., Vincze, V., QasemiZadeh, B., Candito, M., Cap, F., Giouli, V., Stoyanova, I., and Doucet, A. (2017). The parseme shared task on automatic identification of verbal multiword expressions. In Proceedings of the 13th Workshop on Multiword Expressions (MWE 2017), pages 31–47, Valencia, Spain. Association for Computational Linguistics.

Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic Compositionality Through Recursive Matrix-Vector Spaces. In Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP).

Socher, R., Manning, C. D., and Ng, A. Y. (2010). Learning continuous phrase representations and syntactic parsing with recursive neural networks. Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop, pages 1–9.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 1631–1642, Stroudsburg, PA. Association for Computational Linguistics.

Steedman, M. (1996). Surface structure and interpretation. Linguistic inquiry monographs. MIT Press, Cambridge (Mass.), London.

Steedman, M. (2000). The Syntactic Process. MIT Press, Cambridge, MA, USA.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. CoRR, abs/1503.00075.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Titov, I., Henderson, J., Merlo, P., and Musillo, G. (2009). Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In Proceedings of the 21st International Jont Conference on Artifical Intelligence, IJCAI'09, pages 1562–1567, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Tu, Y. and Roth, D. (2011). Learning english light verb constructions: Contextual or statistical. In Proceedings of the Workshop on Multiword Expressions: from Parsing and Generation to the Real World, pages 31–39. Association for Computational Linguistics.

Van de Cruys, T. and Moirón, B. V. (2007). Semantics-based multiword expression extraction. In Proceedings of the Workshop on A Broader Perspective on Multiword Expressions, pages 25–32. Association for Computational Linguistics.

Vincze, V., Nagy T., I., and Zsibrita, J. (2013). Learning to detect english and hungarian light verb constructions. ACM Trans. Speech Lang. Process., 10(2):6:1–6:25.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1990). Readings in speech recognition. chapter Phoneme Recognition Using Time-delay Neural Networks, pages 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Wang, C., Nianwen, X., and Sameer, P. (2015a). A transition-based algorithm for amr parsing. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics.

Wang, C. and Xue, N. (2017). Getting the most out of amr parsing. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 1268–1279, Copenhagen, Denmark. Association for Computational Linguistics.

Wang, C., Xue, N., and Pradhan, S. (2015b). 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 (Volume 2: Short Papers), pages 857–862, Beijing, China. Association for Computational Linguistics.

Weischedel, R., Palmer, M., Marcus, M., Hovy, E., Pradhan, S., Ramshaw, L., Xue, N., Taylor, A., Kaufman, J., Franchini, M., El-Bachouti, M., Belvin, R., and Houston, A. (2013). OntoNotes release 5.0.

Werling, K., Angeli, G., and Manning, C. D. (2015). Robust subgraph generation improves abstract meaning representation parsing. CoRR, abs/1506.03139.

Weston, J., Bengio, S., and Usunier, N. (2011). Wsabie: Scaling up to large vocabulary image annotation. In Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI.

Xu, Y., Mou, L., Li, G., Chen, Y., Peng, H., and Jin, Z. (2015). Classifying relations via long short term memory networks along shortest dependency path. CoRR, abs/1508.03720.

Xue, N. and Palmer, M. (2004). Calibrating features for semantic role labeling. In In Proceedings of EMNLP 2004, pages 88–94.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics, ACL '95, pages 189–196, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhao, H., Chen, W., and Kit, C. (2009). Semantic dependency parsing of nombank and propbank: An efficient integrated approach via a large-scale feature selection. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1, EMNLP '09, pages 30–39, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhou, J. and Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 1127–1137, Beijing, China. Association for Computational Linguistics.

Appendix A

Transitions of CAMR Parser

CAMR parser is a transition-based system. They define their parser as a state triple (α, β, G). α is a buffer that stores indices of the nodes which have not been processed. β is another buffer $[\beta_0, \beta_1, ..., \beta_j]$ and each element β_i in β indicates the edge (α_0, β_i) which has not been processed. The span graph G stores the partial parses for the input sentence.

When the parsing procedure starts, α is initialized with a post-order traversal of the input dependency tree with top most element α_0. β is initialized with node α_0s children or set to null if α_0 is a leaf node. G is initialized with all the nodes and edges in dependency tree. Initially, all the nodes of G have a span length of one and all the labels for nodes and edges are set to null. As the parsing procedure goes on, the parser will process all the nodes and their outgoing edges in dependency tree in a bottom-up left-right manner, and at each state certain action will be applied to the current node or edge. The parsing process will terminate when both α and β are empty.

There are 8 types of actions in the parser, which is summarized in Table A.1. The action set could be divided into two categories based on conditions of buffer β. When β is not empty, parsing decisions are made based on the edge (α_0, β_0); otherwise, only the current node α_0 is examined. Also, to simultaneously make decisions on the assignment of concept/relation label, extra parameter lr or lc with some actions. $\gamma : V \to L_V$ is the concept labeling function for nodes, and $\delta : A \to L_A$ is the relation labeling function for arcs. So $\delta[(\alpha_0, \beta_0) \to l_r]$ means assigning relation label l_r to arc (α_0, β_0). All the actions update buffer α, β and apply some transformation $G \Rightarrow G'$ to the partial graph.

Table A.1: Transitions designed in CAMR Parsing system

Action	Current State	⇒	Result State	Assign Labels	Precondition
Next Edge-l_r	$(\alpha_0 \vert \alpha', \beta \vert \beta', G)$	⇒	$(\alpha_0 \vert \alpha', \beta', G')$	$\delta[(\alpha_0, \beta_0) \rightarrow l_r]$	
Swap-l_r	$(\alpha_0 \vert \alpha', \beta \vert \beta', G)$	⇒	$(\alpha_0 \vert \beta_0 \vert \alpha', \beta', G')$	$\delta[(\beta_0, \alpha_0) \rightarrow l_r]$	
Reattach$_k$-l_r	$(\alpha_0 \vert \alpha', \beta \vert \beta', G)$	⇒	$(\alpha_0 \vert \alpha', \beta', G')$	$\delta[(k, \beta_0) \rightarrow l_r]$	β is not empty
Replace Head	$(\alpha_0 \vert \alpha', \beta \vert \beta', G)$	⇒	$(\beta_0 \vert \alpha', \beta = CH(\beta_0, G'), G')$	NONE	
Reentrance$_k$-l_r	$(\alpha_0 \vert \alpha', \beta \vert \beta', G)$	⇒	$(\alpha_0 \vert \alpha', \beta_0 \vert \beta', G')$	$\delta[(k, \beta_0) \rightarrow l_r]$	
Merge	$(\alpha_0 \vert \alpha', \beta \vert \beta', G)$	⇒	$(\tilde{\alpha} \vert \alpha', \beta', G')$	NONE	
Next Edge-l_c	$(\alpha_0 \vert \alpha_1 \vert \alpha', [], G)$	⇒	$(\alpha_1 \vert \alpha', \beta = CH(\alpha_1, G'), G'$	$\gamma[\alpha_0 \rightarrow l_c]$	β is empty
Delete Node	$(\alpha_0 \vert \alpha_1 \vert \alpha', [], G)$	⇒	$(\alpha_1 \vert \alpha', \beta = CH(\alpha_1, G'), G'$	NONE	

Appendix B

Sample of AMR Parsing Results

```
::id wb.eng_0003.8  ::smatch 0.8000
::snt Speeding and accidents have surged as
    well :
(s / surge-01
  :ARG1 (a / and
    :op1 (s2 / speed-01)
    :op2 (a2 / accident))
  :mod (a3 / as-well))
```

```
(s / surge-01
  :ARG1 (a / and
    :op1 (a2 / accident)
    :op2 (s2 / speed-01))
  :mod (a3 / as-well))
```

```
::id wb.eng_0003.11
::snt Of course , VDOT has no more money for
    road construction in the Richmond region .
::smatch 0.6531
(h / have-03 :polarity -
  :ARG0 (g / government-organization :wiki "
      Virginia_Department_of_Transportation"
    :name (v / name :op1 "VDOT"))
  :ARG1 (m / money
    :purpose (c2 / construct-01
      :ARG1 (r2 / road
        :location (r3 / region
          :part (c / city :wiki "Richmond,
              _Virginia"
            :name (r / name :op1 "Richmond")))
                 )))
  :mod (o / of-course))
```

```
(o / of-course
  :condition (h / have-02
    :ARG0 (g / government-organization
      :wiki ""
      :name (v / name :op1 "VDOT"))
    :ARG1 (c / construct-01
      :ARG1 (r / road)
      :location (r2 / region
        :mod (c2 / city
          :wiki "Richmond,_Virginia"
          :name (r3 / name :op1 "Richmond")))
      :location (m / money
        :polarity -))))
```

```
::id wb.eng_0003.15
::snt As we predicted , Route 288 is
    generating residential development in
    scarcely populated areas all around its
    exits , overtaxing the local country roads
     .
::smatch 0.6579
(g / generate-01
  :ARG0 (r2 / road :wiki "
      Virginia_State_Route_288"
    :name (r / name :op1 "Route" :op2 288))
  :ARG1 (d / develop-02
    :ARG1 (r3 / residence)
    :location (a / area
      :ARG1-of (p2 / populate-01
        :mod (s / scarce))
      :location (a2 / around
        :op1 (e / exit
          :part-of r2)
        :mod (a3 / all)))
    :ARG0-of (o / overtax-01
      :ARG2 (r4 / road
        :ARG1-of (l / local-02
          :ARG2 (c2 / country)))))
  :ARG1-of (p / predict-01
    :ARG0 (w / we)))
```

```
(p / predict-01
  :ARG0 (w / we)
  :ARG1 (o / overtax-01
    :ARG3 (r / road
      :mod (l / localize-01
        :mod (c / country)))
    :ARG1 (g / generate-01
      :ARG0 (r2 / road
        :wiki "Virginia_State_Route_288"
        :name (r3 / name :op1 "Route" :op2
            288))
      :ARG1 (d / develop-01
        :mod (r4 / residence)
        :location (a / area
          :mod (a2 / around
            :mod (a3 / all)
            :op1 (e / exit
              :poss r2))
          :frequency (p2 / populate-01
            :degree (s / scarce)))))))
```

```
::id wb.eng_0003.21
::snt Here 's another prodiction :
::smatch 0.5000
(t / thing
  :ARG1-of (p / predict-01)
  :mod (a / another))
```

```
(p / predict-01
  :ARG1 (a / another)
  :ARG2 (t / thing))
```

```
::id wb.eng_0003.23
::snt Before the end of the decade , we may
    well see the unfunded liability zoom well
    past the original \$ 400 million it took
    to build the original highway .
::smatch 0.7250
(p / possible-01
  :ARG1 (s / see-01
    :ARG0 (w / we)
    :ARG1 (z / zoom-01
      :ARG1 (l / liability
        :ARG1-of (f / fund-01 :polarity -))
      :ARG4 (p2 / past
        :op1 (m / monetary-quantity :quant
            400000000
          :unit (d / dollar)
          :ARG1-of (t / take-10
            :ARG0 (b2 / build-01
              :ARG1 (h / highway
                :mod (o / original))))
          :mod (o2 / original))
        :mod (w2 / well)))
    :time (b / before
      :op1 (e / end-01
        :ARG1 (d2 / decade))))
  :degree (w3 / well))
```

```
(s / see-01
  :time (b / before
    :op1 (e / end-01
      :ARG1 (d / decade)))
  :ARG0 (w / we)
  :mod (w2 / well)
  :condition-of (p / possible-01)
  :ARG1 (z / zoom-01
    :ARG1 (l / liability
      :mod (f / unfund))
    :ARG2 (t / take-10
      :ARG0 (b2 / build-01
        :ARG1 (h / highway
          :mod (o / original)))
      :ARG1 (p2 / past
        :mod-of e)
      :compared-to (m / monetary-quantity
        :mod (w3 / well)
        :unit (d2 / dollar
          :mod (o2 / original))
        :quant 400000000))))
```

```
::id wb.eng_0003.29
::snt Additional traffic congestion in old
    places ?
::smatch 0.6667
(c / congest-01
  :ARG2 (t / traffic)
  :location (p / place
      :mod (o / old))
  :mode interrogative
  :mod (a2 / additional))
```

```
(c / congest-01
  :mod (a / additional)
  :ARG1 (t / traffic-01)
  :ARG0 (p / place
    :mod (o / old)))
```

```
::id wb.eng_0003.33
::snt You have a city , Richmond , that has
    virtually no traffic .
::smatch 0.6000
(h / have-03
  :ARG0 (c / city :wiki "Richmond,_Virginia" :
      name (r / name :op1 "Richmond"))
  :ARG1 (t / traffic)
  :ARG1-of (h2 / have-polarity-91
    :ARG2 -
    :degree (v / virtual)))
```

```
(h / have-03
  :ARG1 (h2 / have-org-role-91
    :value -
    :mod (t / traffic
      :quant (v / virtual))
    :ARG1 (c / city
      :wiki "Richmond,_Virginia"
      :name (r / name :op1 "Richmond"))))
```

```
::id wb.eng_0003.57
::snt It seems to me that the whole point of
    roads is to facilitate the public interest
     in getting to and from private businesses
     .
::smatch 0.5714
(s / seem-01
  :ARG1 (h / have-purpose-91
    :ARG1 (r / road)
    :ARG2 (f / facilitate-01
      :ARG0 r
      :ARG1 (i3 / interest-01
        :ARG1 (p / public)
        :ARG2 (a / and
          :op1 (g / get-05
            :ARG0 p
            :ARG2 (b / business
              :ARG1-of (p2 / private-03)))
          :op2 (g2 / get-05
            :ARG0 p
            :source b))))
    :mod (w / whole))
  :ARG2 (i / i))
```

```
(h / have-purpose-91
  :ARG2 (f / facilitate-01
    :ARG1 (i / interest-01
      :mod (p / publicize-01)
      :ARG1 (g / get-03
        :ARG2 (a / and
          :op1 (p2 / private-03)
          :op2 (g2 / get-05
             :ARG2 (b / business))))))
  :time (r / road
    :domain (w / whole
      :mod (s / seem-01
        :ARG2 (i2 / i)))))
```

```
::id wb.eng_0003.58
::snt Then , of course there is the sunday
    drive , the trip to grandma 's , and the
    soccer games , but still , the
    overwhelming need is to support business .
::smatch 0.6769
(n / need-01
  :ARG1 (s3 / support-01
     :ARG1 (b / business))
  :ARG0-of (o / overwhelm-01)
  :concession (a / and
    :op1 (d / drive-01
      :time (d2 / date-entity
        :weekday (s / sunday)))
    :op2 (t / trip-03
      :ARG1 (l / location
        :poss (p / person
          :ARG0-of (h / have-rel-role-91
            :ARG2 (g / grandmother)))))
    :op3 (g2 / game
      :mod (s2 / soccer))
    :mod (o2 / of-course)))
```

```
(n / need-01
  :ARG1 (c / course-01)
  :ARG1 (a / and
    :op1 (g / game
      :mod (s / soccer))
    :op2 (t / trip-03
      :ARG1 (h / have-rel-role-91
        :ARG0 (p / person
          :poss-of (l / location))
        :ARG2 (p2 / person
          :op1 (g2 / grandmother)))))
    :op3 (d / drive-01
      :mod (d2 / date-entity
        :weekday (s2 / sunday)))
  :time (o / overwhelm-01)
  :condition-of (s3 / support-01
    :ARG1 (b / business)))
```

```
::id wb.eng_0003.60
::snt If we recognize that roads primaarily
    benefit businesses then we should do two
    things :
::smatch 0.6667
(r2 / recommend-01
  :ARG1 (d / do-02
      :ARG0 w
      :ARG1 (t / thing
      :quant 2))
  :condition (r / recognize-02
       :ARG0 (w / we)
       :ARG1 (b / benefit-01
           :ARG0 (r3 / road)
           :ARG1 (b2 / business)
           :mod (p / primary))))
```

```
(d / do-02
  :ARG1 (r / recognize-01
    :ARG0 (w / we)
    :ARG1 (b / benefit-01
      :ARG0 (r2 / road)
      :mod (p / primary)
      :ARG1 (b2 / business)))
  :ARG2 (r3 / recommend-01
    :ARG1 (t / thing)
    :quant 2))
```

```
::id wb.eng_0003.66
::snt TMT said `` In Virginia , it 's to
    benefit private business plans and not to
    serve the public interest . ''
::smatch 0.6667
(s2 / say-01
  :ARG0 (p2 / person :wiki - :name (n / name :
      op1 "TMT"))
  :ARG1 (a / and
    :op1 (h / have-purpose-91
      :ARG1 (i / it)
      :ARG2 (b / benefit-01
        :ARG1 (p3 / plan
          :mod (b2 / business
            :ARG1-of (p4 / private-03)))))
    :op2 (h2 / have-purpose-91 :polarity -
      :ARG1 i
      :ARG2 (s / serve-01
        :ARG1 (i2 / interest
          :poss (p / public))))
    :location (s3 / state :wiki "Virginia" :
        name (n2 / name :op1 "Virginia"))))
```

```
(s / say-01
  :ARG0 (p / person
    :wiki -
    :name (n / name :op1 "TMT"))
  :ARG1 (i / it)
  :ARG0 (a / and
    :op1 (s2 / serve-01
      :polarity -
      :ARG1 (i2 / interest
        :mod (p2 / public)))
    :op2 (b / benefit-01
        :ARG1 (p3 / plan
          :mod (b2 / business)
          :mod (p4 / private-03)))
    :op3 (s3 / state
      :wiki "Virginia"
      :name (n2 / name :op1 "Virginia"))))
```

```
::id wb.eng_0003.68
::snt I think they are so tightly intertwined
    that they are virtually one and the same .
::smatch 0.7647
(t / think-01
  :ARG0 (i / i)
  :ARG1 (i2 / intertwine-01
    :ARG1 (t2 / they)
    :ARG0-of (c / cause-01
      :ARG1 (s / same-01
        :ARG1 t2
        :mod (v / virtual)))
    :ARG0-of (t3 / tight-05)))
```

```
(t / think-01
  :ARG0 (i / i)
  :time-of (s / same-01
    :mod (v / virtual))
  :time (t2 / tight-05)
  :ARG1 (i2 / intertwine-01
    :ARG1 (t3 / they)
    :op-of (c / cause-01
      :ARG1 t3)))
```

```
::id wb.eng_0003.85
::snt we can actually get some commercial
    development and have the ability to work
    closer to where we live .
::smatch 0.6977
(a / and
  :op1 (p / possible-01
    :ARG1 (d / develop-02
      :ARG3 (w2 / we)
      :mod (c / commerce)))
  :op2 (p2 / possible-01
    :ARG1 (w / work-01
      :ARG0 w2
      :location (c2 / close-10
        :ARG2 (l / live-01
          :ARG0 w2)
        :degree (m / more)))))
```

```
(a / and
  :op1 (w / work-01
    :manner (c / close-01
      :ARG2 (l / live-01
        :ARG0 w2)
      :degree (m / more))
    :ARG1 (p2 / possible-01))
  :op2 (p / possible-01
    :ARG0 (w2 / we)
    :ARG1 (d / develop-02
      :mod (c2 / commerce))))
```

```
::id wb.eng_0003.94
::snt Many people have legitimate preferences
    - horses , hunting , gardening , certain
    types of small business - that makes them
    NOT want to live in an urban core .
::smatch 0.5634
(p / prefer-01
  :ARG0 (p2 / person
    :quant (m / many))
  :ARG1 (a / and
    :op1 (h / horse)
    :op2 (h2 / hunt-01)
    :op3 (g / garden-01)
    :op4 (b / business
      :mod (s / small)
      :mod (t / type
        :mod (c / certain))))
  :ARG1-of (l / legitimate-02)
  :ARG0-of (m2 / make-02
    :ARG1 (w / want-01 :polarity -
      :ARG0 p2
      :ARG1 (l2 / live-01
        :ARG0 p2
        :location (c3 / core
          :mod (u / urban))))))
```

```
(a / and
  :op1 (b / business
    :mod (t / type
      :mod (c / certain)
      :mod (s / small)))
  :op2 (p / person
    :mod (m / many))
  :op3 (w / want-01
    :polarity -
    :ARG1 (l / live-01
      :location (c2 / core
        :mod (u / urban)))
    :mod p)
  :op4 (l2 / legitimate-02)
  :op5 (h / hunt-01)
  :op6 (m2 / make-01
    :ARG2 (g / garden)
    :ARG1 (h2 / horse))
  :op7 (p2 / prefer-01))
```

```
::id wb.eng_0003.98
::snt What 's consistently overlooked in the
    debate ovre 288 is what else that \$ 400
    million could have been spent on .
::smatch 0.5652
(o / overlook-01
  :ARG1 (t / thing
    :mod (e / else)
    :ARG1-of (s2 / spend-01
      :ARG3 (m2 / monetary-quantity :quant
          400000000
        :unit (d2 / dollar))
      :ARG1-of (p / possible-01)))
  :ARG1-of (c / consistent-02)
  :subevent-of (d3 / debate-01
    :ARG1 (r / road :wiki "
        Virginia_State_Route_288" :name (n /
        name :op1 "288"))))
```

```
(t / thing
  :condition (s2 / spend-01
    :ARG1 (m2 / monetary-quantity
        :unit (d2 / dollar)
        :quant 400000000))
  :domain (o / overlook-01
    :manner (c / consistent)
    :location (d3 / debate-01
      :ARG0 (r / road
        :wiki "Virginia_State_Route_288"
        :name (n / name :op1 288)))))
```

```
::id wb.eng_0003.99
::snt How much traffic congestion could have
    been alleviated had the money been applied
     to other pressing road projects or , god
    forbid , even non-road projects .
::smatch 0.6462
(p / possible-01
  :ARG1 (a / alleviate-01
    :ARG1 (c / congest-01
      :ARG2 (t / traffic)
      :quant (m2 / much)))
  :condition (a3 / apply-02
    :ARG1 (m / money)
    :ARG2 (o / or
      :op1 (p2 / project
        :mod (r / road)
        :ARG2-of (p3 / press-01)
        :mod (o2 / other))
      :op2 (p4 / project
        :mod (r2 / road :polarity -)
        :mod (e / even)
        :ARG1-of (f / forbid-01 :mode
            imperative
          :ARG0 (g / god))))))
```

```
(a / alleviate-01
  :ARG1 (c / congest-01
    :mod (m / much)
    :ARG1 (t / traffic))
  :ARG2 (p / press-01
    :ARG2 (o / or
      :op1 (f / forbid-01
        :ARG0 (g / god))
      :op2 (p2 / project
        :degree (e / even)
        :mod (r / road
          :polarity -))
      :op3 (p3 / project
        :mod (o2 / other)
        :mod (r2 / road)))
    :ARG2 (a2 / apply-02
      :ARG1 (m2 / money))))
```

```
::id DF-199-194215-653_0484.6
::snt I want him to be there for his son
    without needing me too.
::smatch 0.6667
(w / want-01
  :ARG0 (i / i)
  :ARG1 (h / he
    :location (t / there
      :beneficiary (p / person
        :ARG0-of (h2 / have-rel-role-91
          :ARG1 h
          :ARG2 (s / son)))))
  :manner (n / need-01 :polarity -
    :ARG0 h
    :ARG1 i
    :mod (t2 / too)))
```

```
(w / want-01
  :ARG0 (i / i)
  :ARG1 (t / there
    :domain (h / he)
    :polarity -
    :ARG0-of (h2 / have-rel-role-91
      :ARG2 (s / son)
      :ARG2 (p / person
        :poss h)))
  :manner (n / need-01
    :mod (t2 / too)
    :ARG1 i))
```

```
::id DF-199-194215-653_0484.10
::snt I do believe every child needs both
    parents weather they are together or not.
::smatch 0.6939
(b / believe-01
  :ARG0 (i / i)
  :ARG1 (n / need-01
    :ARG0 (c / child
      :mod (e / every))
    :ARG1 (p / person
      :mod (b2 / both)
      :ARG0-of (h / have-rel-role-91
        :ARG1 c
        :ARG2 (p2 / parent)))
    :concession (o / or
      :op1 (t / together
        :domain p)
      :op2 (t2 / together :polarity -
        :domain p))))
```

```
(b / believe-01
  :ARG0 (i / i)
  :ARG1 (n / need-01
    :ARG0 (c / child
      :mod (e / every))
    :ARG1 (p / person)
    :mod (t / together)
    :polarity -)
  :purpose (h / have-rel-role-91
    :ARG2 (p2 / parent
      :mod (b2 / both))))
```

```
::id DF-199-194215-653_0484.15
::snt You have a right to live your own life
    and be happy, so don't cave to his threats
    .
::smatch 0.4898
(r / right-05
  :ARG1 (y / you)
  :ARG2 (a / and
    :op1 (l / live-01
      :ARG0 y
      :ARG1 (l2 / life
        :poss y))
    :op2 (h / happy-01
      :ARG1 y))
  :ARG0-of (c / cause-01
    :ARG1 (c2 / cave-01 :polarity -
      :ARG1 y
      :ARG1-of (c3 / cause-01
        :ARG0 (t / threaten-01
          :ARG0 (h3 / he))))))
```

```
(c / cause-01
  :ARG1 (c2 / cave-01
    :condition (r / right
      :mod (y / you)
      :manner (a / and
        :op1 (l / live-01
          :ARG1 (l2 / life)
          :degree (h / happy))))
    :ARG1 (t / threaten-01
      :ARG1 (h2 / he))
    :condition (c3 / cause-01
      :polarity -)))
```

```
::id DF-199-194215-653_0484.17
::snt Others say things like, "I'm going to
    kill myself if you leave me"... get it?
::smatch 0.6316
(m / multi-sentence
  :snt1 (s / say-01
    :ARG0 (p / person
      :mod (o / other))
    :ARG1 (t / thing
      :ARG1-of (r / resemble-01
        :ARG2 (k / kill-01
          :ARG0 (i2 / i)
          :ARG1 i2
          :condition (l / leave-15
            :ARG0 (y2 / you)
            :ARG1 i2)))))
  :snt2 (g / get-through-12 :mode
      interrogative
    :ARG0 (i / it)
    :ARG1 (y / you)))
```

```
(m / multi-sentence
  :snt1 (g / get-01
    :ARG1 (i / it
      :purpose (s / say-01
        :ARG0 (p / person
          :mod (o / other))
        :ARG1 (t / thing)
        :ARG0 (y / you)))
    :manner through)
  :snt2 (r / resemble-01
    :ARG2 (k / kill-01
      :ARG2 (l / leave-15
        :ARG0 (y2 / you)
        :ARG1-of p)
      :ARG1 (i2 / i
        :domain p))))
```

```
::id DF-200-192400-625_6304.3
::snt The first night I was here, all my
    housemates and I went out and it was the
    hardest thing I have ever ever had to do.
::smatch 0.5185
(a / and
  :op1 (g / go-out-17
    :ARG0 (a2 / and
      :op1 (p / person
        :mod (a3 / all)
        :ARG0-of (h2 / have-rel-role-91
          :ARG1 i
          :ARG2 (h4 / housemate)))
      :op2 i)
    :time (d2 / date-entity
      :dayperiod (n / night)
      :ord (o / ordinal-entity :value 1)
      :time-of (b / be-located-at-91
        :ARG1 (i / i)
        :ARG2 (h / here))))
  :op2 (h3 / hard-02
    :ARG1 g
    :degree (m / most)
    :compared-to (t / thing
      :ARG1-of (d / do-02
        :ARG0 i
        :time (e / ever)
        :ARG1-of (o2 / obligate-01)))))
```

```
(a / and
  :op1 (h / hard-02
    :mod (t / thing
      :mod (m / most)))
  :op2 (g / go-04
    :time (n / night
      :ord (o / ordinal-entity
        :value 1))
    :ARG1 (a2 / and
      :op1 (h2 / have-org-role-91
        :ARG2 (h3 / housemate
          :mod (a3 / all)
          :domain i))
      :op2 (p / person)
      :poss i))
  :op3 (b / be-located-at-91
    :ARG1 (i / i)
    :degree (h4 / here))
  :op4 (d / do-02
    :time (e / ever)
    :condition-of i))
```

```
::id DF-200-192400-625_6304.5
::snt Anyways more to the point; Here are the
    facts:
::smatch 0.6667
(m2 / multi-sentence
  :snt2 (f / fact
    :location (h / here))
  :snt1 (c / contrast-01
    :ARG2 (p / point
      :degree (m / more))))
```

```
(m / multi-sentence
  :snt1 (f / fact
    :mod (h / here))
  :snt2 (c / contrast-01
    :ARG2 (p / point)
    :degree (m2 / more)))
```

```
::id DF-200-192400-625_6304.24
::snt If you've always been an anxious person
    you might benefit from speaking to someone
     (my friend got therapy on the NHS for
    anxiety, so its really normal).
::smatch 0.5684
(m / multi-sentence
  :snt1 (p / possible-01
    :ARG1 (b / benefit-01
      :ARG0 (s / speak-01
        :ARG0 y
        :ARG2 (s2 / someone))
      :ARG1 (y / you)
      :condition (p2 / person
        :mod (a2 / anxious)
        :domain y
        :time (a3 / always))))
  :snt2 (g / get-01
    :ARG0 (p3 / person
      :ARG0-of (h / have-rel-role-91
        :ARG1 (i / i)
        :ARG2 (f / friend)))
    :ARG1 (t / therapy)
    :ARG2 (o / organization :wiki "
        National_Health_Service_(England)" :
        name (n / name :op1 "NHS"))
    :ARG0-of (c / cause-01
      :ARG1 (n2 / normal-02
        :ARG1 (i2 / it)
        :degree (r / really)))
    :ARG1-of (c2 / cause-01
      :ARG0 (a4 / anxiety))))
```

```
(m / multi-sentence
  :snt1 (s / speak-01
    :ARG2 (s2 / someone)
    :ARG1 (b / benefit-01
        :ARG0 (p / person
          :mod (a / anxious))
        :ARG1 (y / you)))
  :snt2 (n2 / normalize-01
    :ARG1 (c / cause-01
      :mod (i / it)
      :domain (r / real))
    :condition-of p)
  :snt3 (h / have-rel-role-91
    :ARG0 (i2 / i)
    :ARG2 (f / friend))
  :snt4 (c2 / cause-01
    :ARG0 (a2 / anxiety)
    :location (t / therapy)
    :purpose (g / get-03
      :ARG0 (p2 / person)
      :ARG1 (o / organization
        :wiki "National_Health_Service_(
            England)"
        :quant (n / name :op1 "NHS")))))
```

```
::id DF-200-192400-625_6304.25
::snt And try and get deadlines extended, if
    you're honest I'm sure your uni will allow
     it.
::smatch 0.6071
(m / multi-sentence
  :snt2 (s / sure-02
    :ARG0 (i / i)
    :ARG1 (a / allow-01
      :ARG0 (u / university
        :poss (y2 / you))
      :ARG1 (i2 / it))
    :condition (h / honest-01
      :ARG0 y))
  :snt1 (a2 / and
    :op1 (t / try-01 :mode imperative
      :ARG0 (y / you)
      :ARG1 (g / get-04
        :ARG0 y
        :ARG1 (e / extend-01
          :ARG1 (d / deadline))))))
```

```
(m / multi-sentence
  :snt1 (e / extend-01
    :ARG0 (d / deadline))
  :snt2 (g / get-01)
  :snt3 (a / and
    :op1 (t / try-01
      :ARG0 (y / you)))
  :snt4 (a2 / allow-01
    :ARG0 (u / university
      :poss (y2 / you))
    :ARG1 (i / it)
    :mod (h / honest)
    :ARG1 (i2 / i)))
```

```
::id DF-200-192400-625_6344.5
::snt I can't cope with her outbursts and
    nasty hateful remarks when I'm at home.
::smatch 0.7308
(p / possible-01 :polarity -
  :ARG1 (c / cope-01
    :ARG0 (i / i)
    :ARG1 (a / and
      :op1 (o / outburst
        :poss (s / she))
      :op2 (t / thing
        :ARG1-of (r / remark-01
          :ARG0 s)
        :mod (h / hateful)
        :mod (n / nasty))))
  :time (b / be-located-at-91
    :ARG1 i
    :ARG2 (h2 / home)))
```

```
(p / possible-01
  :ARG0 (i / i)
  :ARG1 (c / cope-01
    :polarity -
    :ARG1 (a / and
      :op1 (s / she)
      :op2 (t / thing
        :domain (r / remark-01
          :mod (n / nasty)
          :ARG1 (h / hateful)))
      :op3 (o / outburst)))
  :time (b / be-located-at-91
    :ARG2 (h2 / home)
    :ARG1 i))
```

```
::id DF-200-192400-625_6344.18
::snt Thanks for reading.
::smatch 0.7692
(t / thank-01
  :ARG1 (y / you)
  :ARG2 (r / read-01
    :ARG0 y))
```

```
(t / thank-01
  :ARG2 (r / read-01)
  :ARG0 (y / you))
```

```
::id DF-200-192400-625_6344.25
::snt I had to deal with verbal abuse from my
    dad for a long 8 years before I came to
    uni and honestly, the only reason why I'm
    here is because it was the only way out.
::smatch 0.6226
(a / and
  :op1 (o / obligate-01
    :ARG2 (d / deal-01
      :ARG0 (i / i)
      :ARG2 (a2 / abuse-03
        :ARG0 (p / person
          :ARG0-of (h3 / have-rel-role-91
            :ARG1 i
            :ARG2 (d2 / dad)))
        :ARG1 i
        :mod (v / verbal))
      :time (b2 / before
        :op1 (c / come-01
          :ARG1 i
          :ARG4 (u / university)))
      :ARG1-of (l / long-03
        :ARG2 (t2 / temporal-quantity :quant 8
          :unit (y2 / year)))))
  :op2 (c2 / cause-01
    :ARG0 (r / reason
      :mod (o4 / only)
      :domain (w / way
        :mod (o2 / only)
        :direction (o3 / out)
        :domain (i2 / it)))
    :ARG1 (b / be-located-at-91
      :ARG1 i
      :ARG2 (h / here))
    :ARG1-of (h2 / honest-01)))
```

```
(a / and
  :op1 (r / reason
    :mod (o / only)
    :quant (h / honest))
  :op2 (o2 / obligate-01
    :ARG0 (i / i)
    :ARG1 (d / deal-01
      :location (a2 / abuse-01
        :mod (v / verbal)
        :ARG1 (p / person
          :ARG1-of (h2 / have-rel-role-91
            :ARG2 (d2 / dad))
          :poss i))
      :time (l / long-03
        :ARG2 (t / temporal-quantity
          :quant 8
          :unit (y / year)))
      :time (c / come-01
        :ARG0 (b / before)
        :ARG4 (u / university)
        :ARG1 i)))
  :op3 (b2 / be-located-at-91
    :ARG2 (h3 / here)
    :ARG1 i)
  :op4 (i2 / it
    :mod (w / way
      :mod (o3 / only)
      :mod (o4 / out)
      :manner t2)))
```

```
::id DF-200-192400-625_6344.28
::snt You could call Childline or something
    similar?
::smatch 0.6486
(p / possible-01 :mode interrogative
  :ARG1 (c / call-02
    :ARG0 (y / you)
    :ARG1 (o / or
      :op1 (o2 / organization :wiki "ChildLine
          "
        :name (n / name :op1 "Childline"))
      :op2 (s / something
        :ARG1-of (r / resemble-01
          :ARG2 o2)))))
```

```
(p / possible-01
  :ARG1 (c / call-02
    :ARG0 (y / you))
  :ARG1 (o / or
    :op1 (s / something
      :mod (r / resemble-01))
    :op2 (o2 / organization
      :wiki "ChildLine"
      :op1 (n / name :op1 "Childline"))))
```

```
::id DF-200-192400-625_6344.31
::snt So you want her sectioned cos she is a
    bit of a pain in the arse and you've
    blamed all your problems on her?
::smatch 0.3704
(c / cause-01
  :ARG0 (a2 / and
    :op1 (p / pain
      :location (a / arse)
      :degree (b / bit)
      :domain s)
    :op2 (b2 / blame-01
      :ARG0 y
      :ARG1 s
      :ARG2 (p2 / problem
        :mod (a3 / all)
        :poss y)))
  :ARG1 (w / want-01 :mode interrogative
    :ARG0 (y / you)
    :ARG1 (s / section-02
      :ARG1 (s2 / she))))
```

```
(w / want-01
  :ARG1 (a / and
    :mode interrogative
    :op1 (y / you)
    :op2 (s / section-01
      :ARG0 (s2 / she))
    :op3 (c / cause-01
      :ARG0 (b / bit
        :mod (p / pain)
        :domain s2))
    :op4 (b2 / blame-01
      :ARG1 (p2 / problem
        :quant (a2 / all))
      :ARG0 s2)))
```

```
::id DF-200-192400-625_6677.1
::snt I think over the last year or 2 i've
    become a much weaker person and in a way
    have dulled down my personality inorder to
     try and make some friends at uni.
::smatch 0.6296
(t4 / think-01
  :ARG0 (i / i)
  :ARG1 (a2 / and
    :op1 (b / become-01
      :ARG1 i
      :ARG2 (p / person
        :ARG1-of (w / weak-02
          :degree (m / more
            :degree (m2 / much))))
      :time (b4 / before
        :op1 (n / now)
        :duration (b2 / between
          :op1 (t / temporal-quantity :quant 1
            :unit (y / year))
          :op2 (t2 / temporal-quantity :quant
              2
            :unit (y2 / year)))))
    :op2 (d / dull-01
      :ARG0 i
      :ARG1 (p2 / personality
        :poss i)
      :mod (d2 / down)
      :purpose (t3 / try-01
        :ARG0 i
        :ARG1 (b3 / befriend-01
          :ARG0 i
          :ARG1 (p3 / person
            :quant (s / some))
          :location (u / university)))
      :manner (w2 / way)
      :time b2)))
```

```
(t / think-01
  :ARG0 (i / i)
  :ARG1 (a / and
    :op1 (p / person
      :mod (w / weak
        :degree (m / more
          :degree (m2 / much))))
    :op2 (d / dull-01
      :ARG1 (w2 / way)
      :direction (d2 / down)
      :ARG0 (p2 / personality
        :poss i)
      :purpose (t2 / try-01
        :ARG1 (b / be.03
          :ARG0 (f / friend)
          :ARG1 (p3 / person
            :mod (s / some)
            :location (u / university)))))
    :op3 (b2 / become-01
      :ARG1 (t3 / temporal-quantity
        :quant 2
        :unit (b3 / before)
        :unit (b4 / between
          :op1 (y / year
            :op1 (t4 / temporal-quantity
              :op1 (y2 / year
                :mod (n / now)
                :quant 1)))))
      :ARG1 i)))
```

```
::id DF-200-192400-625_7046.9
::snt Its not uncommon for people's existing
    feelings about themselves to become more
    intense when they are in new situations
    and feel the pressure to socialise with
    strangers and fit into new roles, i.e.
    being a university student.
::smatch 0.5773
(c / common
  :domain (b / become-01
    :ARG1 (t / thing
      :ARG1-of (f / feel-01
        :ARG0 (p / person)
        :ARG2 p)
      :ARG1-of (e / exist-01))
    :ARG2 (i / intense-02
      :ARG1 t
      :degree (m / more))
    :time (a / and
      :op1 (b2 / be-located-at-91
        :ARG1 p
        :ARG2 (s / situation
          :ARG1-of (n / new-02)))
      :op2 (f2 / feel-01
        :ARG0 p
        :ARG1 (p2 / pressure-01
          :ARG1 p
          :ARG2 (a2 / and
            :op1 (s2 / socialize-01
              :ARG0 p
              :prep-with (s3 / stranger))
            :op2 (f3 / fit-06
              :ARG1 p
              :ARG2 (r / role
                :ARG1-of n
                :example (p3 / person
                  :ARG0-of (s4 / study-01
                    :location (u / university)
                      ))))))))))
```

```
(c / common
  :condition-of (i / intense-02
    :degree (m / more)
    :ARG1 (b / become-01
      :ARG1 (t / thing
        :domain (f / feel-01
          :ARG0 p
          :time-of (e / exist-01
            :ARG2 (p / person)))))
    :time (a / and
      :op1 (s / situation
          :mod (n / new))
      :op2 (f2 / feel-01
        :ARG1 (p2 / pressure-01
          :ARG2 (f3 / fit-06
            :ARG2 (r / role
              :mod (p3 / person)))
          :ARG1 (a2 / and
            :op1 (s2 / socialize-01
              :location (s3 / stranger))))
        :condition (s4 / study-01
          :ARG1 (u / university)
          :ARG2 r))
      :op3 p)))
```

```
::id DF-200-192400-625_7046.14
::snt If i were you id focus on an idol.
    Someone you aspire to (not want to be them
     but someone who your admire and have
    respect for) e.g. someone who came from a
    bad background and ended up successful and
     happy with a family and married etc etc
::smatch 0.5225
(m / multi-sentence
  :snt1 (f / focus-01
    :ARG0 (i / i)
    :ARG2 (i2 / idol)
    :condition (y / you
      :domain i))
  :snt2 (a / aspire-01
    :ARG0 (y2 / you)
    :ARG1 (s / someone
      :example (s2 / someone
        :ARG1-of (c / come-03
          :ARG2 (b / background
            :ARG1-of (b2 / bad-07)))
        :ARG1-of (e / end-up-03
          :ARG2 (a2 / and
            :op1 (s3 / succeed-01
              :ARG0 s2)
            :op2 (h / happy-01
              :ARG1 s2)
            :op3 (h2 / have-03
              :ARG0 s2
              :ARG1 (f2 / family))
            :op4 (m2 / marry-01
              :ARG1 s2)
            :op5 (e2 / et-cetera))))))
  :snt3 (c3 / contrast-01
    :ARG1 (w / want-01 :polarity -
      :ARG0 (y3 / you)
      :ARG1 (t / they
        :domain y3))
    :ARG2 (s4 / someone
      :ARG1-of (a3 / admire-01
        :ARG0 y3)
      :ARG1-of (r / respect-01
        :ARG0 y3))))
```

```
(m / multi-sentence
  :snt1 (r / respect-01)
  :snt2 (s / someone
    :op1 (f / focus-01
      :mod (y / you)
      :ARG0 (i / i)
      :ARG1 (i2 / idol)))
  :snt3 (a / aspire-01
    :ARG0 (y2 / you))
  :snt4 (c / contrast-01
    :ARG0 (y3 / you
      :op1 (s2 / someone))
    :ARG1 (w / want-01
      :polarity -
      :ARG1 (t / they)))
  :snt5 (c2 / come-01
    :ARG1 (s3 / someone)
    :ARG1 (b / background)
      :mod (b2 / bad-07)
      :time-of (e / end-up-03
        :ARG1 (a2 / and
          :op1 (h / happy)
          :op2 (m2 / marry-01)
          :op3 (s4 / succeed-01)
          :op4 (f2 / family)))))
```

```
::id DF-200-192400-625_7200.3
::snt What made them decide toe-walking is a
    sign of autism since it seems so random
    and isn't always indicative?
::smatch 0.6032
(m / make-02
  :ARG0 (a / amr-unknown)
  :ARG1 (d / decide-01
    :ARG0 (t / they)
    :ARG1 (s / signal-07
      :ARG0 (w / walk-01
        :manner (t2 / toe))
      :ARG1 (a2 / autism)))
  :ARG1-of (c / cause-01
    :ARG0 (a3 / and
      :op1 (s2 / seem-01
        :ARG1 (r / random
          :degree (s3 / so)
          :domain w))
      :op2 (i / indicate-01 :polarity -
        :ARG0 w
        :time (a4 / always)))))
```

```
(c / cause-01
  :ARG0 (a / and
    :op1 (r / random
      :degree (s / so))
    :op2 (a2 / always)
    :op3 (s2 / seem-01
      :condition s3)
    :op4 (i / indicate-01
      :polarity -
      :ARG1 (m / make-02
        :ARG1 (d / decide-01
          :ARG0 (t / they)
          :ARG1 (w / walk-01
            :mod (t2 / toe)
            :ARG1 (s3 / sign-02
              :ARG1 (a3 / autism))))))))
```

```
::id DF-200-192400-625_7200.11
::snt My brother's autistic but I haven't
    noticed this with him either, but the
    spectrum is so broad and I myself probably
     have mild asperger's because I have
    problems with communication and don't pick
     up on social reactions/body language.
::smatch 0.6667
(c / contrast-01
  :ARG1 (c2 / contrast-01
    :ARG1 (a / autistic
      :domain (p4 / person
        :ARG0-of (h3 / have-rel-role-91
          :ARG1 (i / i)
          :ARG2 (b / brother))))
    :ARG2 (n / notice-01 :polarity -
      :ARG0 i
      :ARG1 a
      :mod (e / either)))
  :ARG2 (a2 / and
    :op1 (b2 / broad-02
      :ARG1 (s2 / spectrum)
      :degree (s / so))
    :op2 (p / probable
      :domain (h2 / have-03
        :ARG0 i
        :ARG1 (d / disease :wiki "
            Asperger_syndrome" :name (n2 /
            name :op1 "Asperger's")
          :mod (m2 / mild))
        :ARG0-of (c3 / cause-01
          :ARG1 (a5 / and
            :op1 (h / have-03
              :ARG0 i
              :ARG1 (p2 / problem
                :topic (c4 / communicate-01)))
            :op2 (p3 / pick-up-11 :polarity -
              :ARG0 i
              :ARG1 (s4 / slash
                :op1 (r / react-01
                  :mod (s3 / society))
                :op2 (l / language
                  :mod (b3 / body))))))))))
```

```
(c / contrast-01
  :ARG2 (c2 / cause-01
    :ARG2 (b / broad-02
      :degree (s / so)
      :mod (a / and)
      :op-of (s2 / spectrum)))
  :ARG2 (c3 / contrast-01
    :ARG1 (p / person)
    :ARG2 (a2 / autistic)
    :condition (h / have-03
      :ARG0 (a3 / and
        :op1 (d / disease
          :mod (m / mild)
          :name (n / name :op1 "Asperger's"))
        :op2 (p2 / pick-up-11
          :polarity -
          :ARG1 (s3 / slash
            :mod (s4 / society)
            :mod (l / language
              :mod (b2 / body))))
        :op3 (h2 / have-03
          :ARG1 (p3 / problem
            :mod (c4 / communicate-01)))
        :op4 (r / react-01))))
  :time (h3 / have-rel-role-91
    :ARG2 (i / i)
    :ARG2 (b3 / brother))
  :condition-of (n2 / notice-01
    :polarity -
    :mod (e / either)))
```

```
::id DF-200-192400-625_7557.5
::snt they healed, then i cut myself with
    staples i robbed from school and a razor.
::smatch 0.7727
(a / and
  :op1 (h / heal-01
    :ARG1 (t / they))
  :op2 (c / cut-01
    :ARG0 (i / i)
    :ARG1 i
    :ARG3 (a2 / and
      :op1 (s / staple
        :ARG2-of (r / rob-01
          :ARG0 i
          :ARG1 (s2 / school)))
      :op2 (r2 / razor))
    :mod (t2 / then)))
```

```
(a / and
  :op1 (h / heal-01
    :ARG1 (t / they))
  :op2 (c / cut-01
    :ARG0 (i / i)
    :ARG1 (a2 / and
      :op1 (r / rob-01
        :ARG1 (s / school))
      :op2 (r2 / razor)
      :op3 (s2 / staple))))
```

```
::id DF-200-192400-625_7557.24
::snt It might be an idea to talk to your
    school nurse or your GP and tell them what
    's going on.
::smatch 0.6875
(r / recommend-01
  :ARG1 (a / and
    :op1 (t / talk-01
      :ARG0 (y / you)
      :ARG2 (o / or
        :op1 (p / person
          :ARG0-of (h2 / have-org-role-91
            :ARG1 (s / school
              :poss y)
            :ARG2 (n / nurse)))
        :op2 (p2 / person
          :ARG0-of (h / have-rel-role-91
            :ARG1 y
            :ARG2 (p3 / practitioner
              :mod (g / general))))))
    :op2 (t2 / tell-01
      :ARG1 (t3 / thing
        :ARG1-of (g2 / go-on-15))
      :ARG2 o)))
```

```
(r / recommend-01
  :ARG1 (a / and
    :op1 (g / go-02
      :ARG0 (t / thing))
    :op2 (t2 / tell-01
      :ARG1-of o)
    :op3 (t3 / talk-01
      :ARG0 (y / you
        :ARG1-of (h / have-org-role-91
          :ARG1 (s / school)
          :time (n / nurse)
          :ARG0 (p / person))))
    :op4 (h2 / have-rel-role-91
      :ARG2 (p2 / practitioner
        :mod (g2 / general))
      :ARG1 (p3 / person))))
```

```
::id DF-200-192400-625_7557.27
::snt if you keep it bottled up inside it's
    likely to only get worse.
::smatch 0.6667
(l / likely-01
  :ARG1 (w / worsen-01
    :ARG1 (i / it)
    :mod (o / only))
  :condition (k / keep-01
    :ARG0 (y / you)
    :ARG1 (i4 / it
      :ARG1-of (b / bottle-up-02)
      :location (i3 / inside))))
```

```
(b / bottle-up-02
  :mod (i / inside)
  :ARG1 (k / keep-01
    :ARG0 (y / you)
    :ARG1 (i2 / it)
    :condition (w / worsen-01
      :mod (o / only))
    :ARG1 (i3 / it)))
```

```
::id DF-200-192400-625_7557.30
::snt I'm 17 I started self harming last year,
     mainly cutting my arms, but I have
    stopped since the last few months,
::smatch 0.7579
(c / contrast-01
  :ARG1 (a / and
    :op1 (a2 / age-01
      :ARG1 (i / i)
      :ARG2 (t2 / temporal-quantity :quant 17
        :unit (y / year)))
    :op2 (s / start-01
      :ARG0 i
      :ARG1 (h / harm-01
        :ARG1 (s2 / self)
        :ARG1-of (m / mean-01
          :ARG2 (c2 / cut-01
            :ARG1 (a3 / arm
              :part-of i)
            :mod (m3 / main))))
      :time (y2 / year
        :mod (l / last))))
  :ARG2 (s3 / stop-01
    :ARG0 i
    :ARG1 h
    :time (b / before
      :op1 (n / now)
      :quant (f / few
        :op1 (t3 / temporal-quantity :quant 1
          :unit (m4 / month))))))
```

```
c / contrast-01
  :ARG1 (a / and
    :op1 (a2 / age-01
      :ARG1 (i / i)
      :ARG2 (t / temporal-quantity
        :unit (y / year)
        :quant 17))
    :op2 (s / stop-01
      :ARG2 (f / few
        :op1 (t2 / temporal-quantity
          :unit (m / month)
          :quant 1))
      :time (n / now)
      :ARG1 (b / before)
      :ARG0 i)
    :op3 (m2 / mean-01
      :ARG2 (c2 / cut-01
        :mod (m3 / main)
        :ARG1 (a3 / arm
          :poss i))
      :condition (s2 / start-01
        :ARG1 (y2 / year)
        :time (h / harm-01
          :ARG1 (s3 / self)
          :mod (l / last)))
      :ARG1 i)))
```

```
::id DF-200-192400-625_7557.32
::snt I feel so much happier, but seeing the
    scars is both a painful reminder of how I
    felt at that time, how strong my feelings
    were that I would want to do such a thing
    to myself!
::smatch 0.6538
(c / contrast-01
  :ARG1 (f / feel-01
    :ARG0 (i / i)
    :ARG1 (h / happy-01
      :ARG1 i
      :degree (m / more
        :quant (m2 / much
          :degree (s / so)))))
  :ARG2 (r / remind-01
    :ARG0 (s4 / see-01
      :ARG0 i
      :ARG1 (s5 / scar))
    :ARG1 (a / and
      :op1 (t4 / thing
        :ARG1-of (f2 / feel-01
          :ARG0 i
          :time (t / time
            :mod (t2 / that))))
      :op2 (s2 / strong-02
        :ARG1 f2
        :degree (s6 / so)
        :ARG0-of (c2 / cause-01
          :ARG1 (w / want-01
            :ARG0 i
            :ARG1 (d / do-02
              :ARG1 (t3 / thing
                :mod (s3 / such))
              :ARG2 i)))))
    :ARG2 i
    :ARG0-of (p / pain-01
      :ARG1 i)))
```

```
(f / feel-01
  :ARG1 (h / happy
    :domain (i / i)
    :degree (m / more
      :degree (m2 / much
        :degree (s / so))))
  :ARG2 (c / contrast-01
    :ARG1 (s2 / see-01
      :ARG1 (s3 / scar))
    :time (c2 / cause-01
      :ARG0 (a / and
        :op1 (f2 / feel-01
          :time (t / time
            :mod (t2 / that))
          :ARG0 i)
        :op2 (t3 / thing)
        :op3 (s4 / strong-02
          :degree (s5 / so))
        :op4 (w / want-01
          :ARG1 (d / do-02
            :ARG1 (t4 / thing
              :mod (s6 / such))
            :ARG0 i)
          :ARG0 i)
        :poss i)
      :ARG1 (r / remind-01
        :ARG1 (p / pain-01)))))
```

```
::id DF-200-192400-625_7744.4
::snt Am I going crazy or something?
::smatch 0.5882
(o / or :mode interrogative
  :op1 (c / craze-01
    :ARG1 (i / i))
  :op2 (s / something))
```

```
(o / or
  :domain (i / i)
  :op1 (s / something)
  :op2 (c / craze-01))
```

```
::id DF-200-192400-625_7744.5
::snt It happens most often when I'm carrying
    many things, but that is rarely and even
    still daily I'm checking many times. And
    is this OCD to any extent?
::smatch 0.6392
(m / multi-sentence
  :snt1 (a / and
    :op1 (i / it
      :frequency (o / often
        :degree (m2 / most)
        :time-of (c / carry-01
          :ARG0 i
          :ARG1 (t / thing
            :quant (m3 / many))
          :ARG1-of (c3 / contrast-01
            :ARG2 (c4 / carry-01
              :ARG0 i
              :ARG1 t
              :ARG1-of (r / rare-02))))))
    :op2 (c2 / check-01
      :ARG0 i
      :frequency (r2 / rate-entity-91
        :ARG1 (m5 / many)
        :ARG2 (t4 / temporal-quantity :quant 1
          :unit (d / day))
        :mod (s / still)
        :mod (e / even))))
  :snt2 (d2 / disease :mode interrogative :
      wiki "Obsessive compulsive_disorder" :
      name (n / name :op1 "OCD")
    :domain (t3 / this)
    :degree (e2 / extent
      :mod (a2 / any))))
```

```
(m / multi-sentence
  :snt1 (c / contrast-01)
  :snt2 (a / and
    :op1 (i / it)
    :op2 (o / often
      :degree (m2 / most))
    :op3 (c2 / carry-01
      :ARG1 (t / thing
        :mod (m3 / many)))
    :op4 (r / rare-02
      :ARG0 (t2 / temporal-quantity
        :quant 1
        :unit (d / day))
      :degree (e / even)
      :degree (s / still))
    :op5 (c3 / check-01))
  :snt3 (t3 / this
    :mod (m4 / many)
    :manner (e2 / extent
      :mod (a2 / any)
      :domain (d2 / disease
        :wiki "Obsessive compulsive_disorder
            "
        :name (n / name :op1 "OCD")))))
```

```
::id DF-200-192400-625_7744.7
::snt If I don't check, I get very very
    anxious, which does sort of go away after
    15-30 mins, but often the anxiety is so
    much that I can't wait that long.
::smatch 0.6019
(c / contrast-01
  :ARG1 (g / get-03
    :ARG1 (i / i)
    :ARG2 (a / anxious
      :degree (v / very)
      :ARG1-of (g2 / go-01
        :direction (a2 / away)
        :mod (s / sort)
        :time (a3 / after
          :op1 (b / between
            :op1 (t / temporal-quantity :quant
                 15
              :unit (m / minute))
            :op2 (t2 / temporal-quantity :
                quant 30
              :unit (m2 / minute))))))
    :condition (c2 / check-01 :polarity -
      :ARG0 i))
  :ARG2 (a4 / anxiety
    :quant (m3 / much
      :ARG0-of (c3 / cause-01
        :ARG1 (p / possible-01 :polarity -
          :ARG1 (w / wait-01
            :ARG1 i
            :ARG1-of (l / long-03
              :degree (t3 / that))))))
    :time (o2 / often)))
```

```
(p / possible-01
  :degree (t / that)
  :ARG1 o2
  :ARG2-of (a / anxiety
    :mod (o / often)
    :degree (m / much)
    :op1 (l / long-03
      :ARG1 (c / cause-01
        :ARG1 (w / wait-01
          :polarity -
          :ARG1 i)))
    :op2 (g / get-03
      :ARG2 (a2 / anxious
        :domain (i / i)
        :degree (v / very))
      :ARG1 (c2 / contrast-01
        :ARG2 (c3 / check-01
          :polarity -
          :ARG0 i)
        :ARG2 (g2 / go-01
          :mod (s / sort)
          :mod (a3 / away)
          :time (a4 / after
            :location (o2 / or
              :op1 (t2 / temporal-quantity
                :quant 15)
              :op2 (t3 / temporal-quantity
                :quant (m2 / minute
                  :quant (m3 / minute
                    :quant 30))))))))))
```

```
::id DF-200-192400-625_7744.11
::snt They've probably never heard of some
    compulsion like this and I just dont know
    what to do
::smatch 0.8000
(a / and
  :op1 (h / hear-01 :polarity -
    :ARG0 (t / they)
    :ARG1 (c / compulsion
      :mod (s / some)
      :ARG1-of (r / resemble-01
        :ARG2 (t2 / this)))
    :time (e / ever)
    :mod (p / probable))
  :op2 (k / know-01 :polarity -
    :ARG0 (i / i)
    :ARG1 (t3 / thing
      :ARG1-of (d / do-02
        :ARG0 i))
    :mod (j / just)))
```

```
(a / and
  :op1 (k / know-01
    :ARG0 (i / i)
    :mod (j / just)
    :polarity -
    :ARG1 (t / thing)
    :domain-of t2)
  :op2 (h / hear-01
    :ARG0 (t2 / they)
    :mod (p / probable)
    :polarity -
    :time (e / ever)
    :ARG1 (c / compulsion
      :source (r / resemble-01
        :ARG2 (t3 / this)
        :ARG1 (s / some)))))
```

```
::id DF-200-192400-625_7806.1
::snt Usually I'm pretty calm about things and
     like to take things as they come and not
    to worry too much.
::smatch 0.5385
(a / and
  :op1 (c / calm-03
    :ARG1 (i / i)
    :mod (p / pretty)
    :topic (t / thing)
    :mod (u / usual))
  :op2 (l / like-01
    :ARG0 i
    :ARG1 (t2 / take-02
      :ARG0 i
      :ARG1 t
      :prep-as (c2 / come-01
        :ARG1 t)))
  :op3 (w / worry-01 :polarity -
    :ARG1 i
    :quant (m / much
      :degree (t3 / too))))
```

```
(a / and
  :op1 (u / usual)
  :op2 (c / calm-03
    :degree (p / pretty)
    :ARG1 (t / thing))
  :op3 (c2 / come-01
    :ARG0 (w / worry-01
      :manner (m / much
        :degree (t2 / too))))
  :op4 (t3 / take-02
    :ARG1 t)
  :op5 (l / like-01))
```

```
::id DF-200-192400-625_7806.5
::snt I've spoken to a few people about it
    including my parents but they all think
    the same: that I'm going through a random
    phase and that I should hold off taking
    action until it is clear that it is a
    serious problem
::smatch 0.6095
(c / contrast-01
  :ARG1 (s / speak-01
    :ARG0 (i / i)
    :ARG1 (i2 / it)
    :ARG2 (p / person
      :quant (f / few)
      :ARG2-of (i3 / include-01
        :ARG1 (p2 / person
          :ARG0-of (h2 / have-rel-role-91
            :ARG1 i
            :ARG2 (p6 / parent))))))
  :ARG2 (t / think-01
    :ARG0 (p3 / person
      :mod (a / all))
    :ARG1 (a2 / and
      :op1 (g / go-02
        :ARG0 i
        :ARG1 (p4 / phase
          :mod (r2 / random)))
      :op1 (r / recommend-01
        :ARG1 (h / hold-off-08
          :ARG0 i
          :ARG1 (a3 / act-02
            :ARG0 i)
          :time (u / until
            :op1 (c2 / clear-06
              :ARG1 (p5 / problem
                :ARG1-of (s2 / serious-02)))))
        :ARG2 i))
    :ARG1-of (s3 / same-01)))
```

```
(c / contrast-01
  :ARG0 p5
  :ARG2 (t / think-01
    :ARG0 (p / person
      :mod (a / all))
    :ARG1 (s / same-01
      :ARG2 (a2 / and
        :op1 (r / recommend-01
          :ARG1 (a3 / act-02
            :ARG1 (u / until
              :domain (c2 / clear-06)
              :domain (p2 / problem
                :mod (s2 / serious)
                :op-of i3))))
        :op2 (g / go-02
          :ARG2 (p3 / phase
            :mod (r2 / random))
          :time (h / hold-off-08
            :ARG0 i)
          :ARG1 i))))
  :ARG1 (s3 / speak-01
    :ARG0 (i / i)
    :ARG1 (p4 / person
      :mod (f / few)
      :op1 (i2 / include-01
        :ARG2 (p5 / person
          :poss i)
        :ARG2 (p6 / parent)))
    :ARG3 (i3 / it)))
```

```
::id nw.chtb_0318.1
::snt Xinhua News Agency , Tokyo , September 1
    st , by reporter Yiguo Yu
::smatch 0.9474
(b / byline-91
  :ARG0 (p2 / publication :wiki "
      Xinhua_News_Agency"
    :name (n / name :op1 "Xinhua" :op2 "News"
        :op3 "Agency"))
  :ARG1 (p / person :wiki -
    :name (n2 / name :op1 "Yiguo" :op2 "Yu")
    :ARG0-of (r / report-01))
  :location (c2 / city :wiki "Tokyo"
    :name (n3 / name :op1 "Tokyo"))
  :time (d / date-entity :month 9 :day 1))
```

```
(b / byline-91
  :location (c / city
    :wiki "Tokyo"
    :name (n / name :op1 "Tokyo"))
  :time (d / date-entity
    :month 9
    :day 1)
  :ARG1 (p / person
    :wiki -
    :name (n2 / name :op1 "Yiguo" :op2 "Yu"))
  :ARG0 (p2 / publication
    :wiki "Xinhua_News_Agency"
    :name (n3 / name :op1 "Xinhua" :op2 "News"
         :op3 "Agency"))
  :time (r / report-01))
```

```
::id nw.chtb_0318.8
::snt He hoped that all the athletes would "
    fully demonstrate the strength and skill
    that they cultivate daily , as the
    competitors representing Japan , they
    should carry out competition with athletes
     from various countries honestly " .
::smatch 0.6239
(a / and
  :op1 (h / hope-01
    :ARG0 (h2 / he)
    :ARG1 (d / demonstrate-01
      :ARG0 (a2 / athlete
        :mod (a3 / all))
      :ARG1 (a4 / and
        :op1 (s / strong-02
          :ARG1 a2)
        :op2 (s2 / skill)
        :ARG1-of (c2 / cultivate-01
          :ARG0 a2
          :frequency (r3 / rate-entity-91
            :ARG2 (t / temporal-quantity :
                quant 1
              :unit (d2 / day)))))
      :degree (f / full)
      :ARG1-of (c / cause-01
        :ARG0 (r / recommend-01
          :ARG1 (c3 / compete-01
            :ARG0 a2
            :ARG1 (a5 / athlete
              :source (c4 / country
                :mod (v / various)))
            :ARG1-of (h3 / honest-01)
            :prep-as (p / person
              :ARG0-of (c6 / compete-01)
              :ARG0-of (r2 / represent-01
                :ARG1 (c5 / country :wiki "
                    Japan" :name (n / name :
                    op1 "Japan")))))))))
```

```
(a / and
  :op1 (h / hope-01
    :ARG0 (h2 / he))
  :op2 (d / demonstrate-01
    :mod (s / strength)
    :ARG2 (a2 / and
      :op1 (a3 / athlete
        :mod (a4 / all))
      :op2 (s2 / skill)
      :op3 (c / cultivate-01)
      :op4 (r / rate-entity-91
        :ARG2 (t / temporal-quantity
          :quant (d2 / day)
          :quant 1))))
  :op3 (c2 / compete-01
    :ARG1 (r2 / represent-01
      :ARG1 (c3 / country
        :wiki "Japan"
        :name (n / name :op1 "Japan"))))
  :op4 (c4 / cause-01
    :ARG2 (p / person))
  :op5 (c5 / compete-01
    :location (a5 / athlete
      :mod (c6 / country
        :mod (v / various)))
    :time (h3 / honest)))
```

```
::id nw.chtb_0324.2
::snt The 1 - meter diving board preliminaries
     of the Seventh World Swimming
    Championship were held here this morning .
::smatch 0.7143
(h / hold-04
  :ARG0 (p / preliminary
    :subevent-of (g / game :wiki -
      :name (n / name :op1 "Seventh" :op2 "
          World" :op3 "Swimming" :op4 "
          Championship"))
    :topic (b / board
      :mod (d3 / distance-quantity :quant 1
        :unit (m2 / meter))
      :instrument-of (d2 / dive-01)))
  :location (h2 / here)
  :time (d / date-entity :dayperiod (m /
      morning) :mod (t / today)))
```

```
(h / hold-04
  :time (m / morning
    :mod (t / this))
  :location (h2 / here)
  :ARG0 (d / dive-01
    :ARG2 (d2 / distance-quantity
      :quant 1
      :unit (m2 / meter))
    :ARG1 (b / board)
    :location (g / game
      :wiki -
      :name (n / name :op1 "Seventh" :op2 "
          World" :op3 "Swimming" :op4 "
          Championship"))))
```

```
::id nw.chtb_0324.12
::snt He felt that , there were more new
    competitors from our country participating
     in this competition .
::smatch 0.7317
(f / feel-02
  :ARG0 (h / he)
  :ARG1 (p / person
    :quant (m / more)
    :ARG0-of (c / compete-01)
    :ARG1-of (n / new-01)
    :source (c2 / country
      :poss (w / we))
    :ARG0-of (p2 / participate-01
      :ARG1 (c3 / compete-01
        :mod (t / this)))))
```

```
(f / feel-02
  :ARG0 (h / he)
  :ARG1 (p / person
    :ARG0-of (p2 / participate-01
      :ARG1 (c / compete-01
        :mod (t / this)))
    :ARG1-of (c2 / compete-01
      :degree (m / more)
      :mod (n / new)
      :ARG1 (c3 / country))))
```

```
::id nw.chtb_0325.6
::snt Her performance was 303.00 points .
::smatch 0.7143
(p2 / point :quant 303.00
  :domain (p / perform-01
    :ARG0 (s / she)))
```

```
(p / perform-01
  :ARG1 (s / she)
  :domain-of (p2 / point
    :quant 303.00))
```