École doctorale IAEM Lorraine

Neural-Symbolic Learning for Semantic Parsing

THÈSE

présentée et soutenue publiquement le

pour l'obtention du

Doctorat de l'Université de Lorraine

(Mention Informatique)

par

Chunyang Xiao

Composition du jury

Rapporteurs :	Anette Frank	Professeur, Heidelberg University, Germany
	Mark Steedman	Professeur, University of Edinburgh, UK
Examinateurs :	Jonathan Berant	Professeur Assistant, Tel-Aviv University, Israël
	Miguel Couceiro	Professeur, LORIA, Nancy, France
Invités :	Katja Filippova	Chercheur, Google, Zurich, Switzerland
	Eric Gaussier	Professeur, Laboratoire d'Informatique de Grenoble, France
Directrice de thèse :	Claire Gardent	Directrice de Recherches CNRS, LORIA, Nancy, France
CoDirecteur de thèse :	Marc Dymetman	Chercheur, Naver Labs Europe, Grenoble, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Acknowledgements

I would like to first thank my supervisors, Marc and Claire. Marc is a wonderful person with whom I spent most of my PhD time. I have learned from him how to formulate and start a PhD project as well as how to write and communicate for scientific purposes; but I feel that the most important thing I have learned from him is how to always be as rigorous as possible towards our own research — while enjoy our lives as much as we could. For this purpose, I want to thank Marc also particularly for the so much fun time I have had with him outside 'the scope' of this thesis.

I feel very fortunate to have Claire as my other supervisor who is always there for help. Due to the location constraints, she does frequently the phone meeting with Marc and me and during the 'hard' time of my PhD, she descended directly to Grenoble to discuss ideas based on which I have my first long paper accepted.

I want to thank my colleagues in Grenoble most of whom are now in NaverLabs. Guillaume was the reason I started my thesis in semantic parsing in Grenoble; Matthias is my manager who gives various excellent advice at different stages of my PhD; Tomi is always having lunch with us (PhDs and interns) sharing his rich experience with us; I hear so many exciting crisp ideas from Chris; I really enjoy the debates among Xavier, James, Guillaume, Ariadna. I love listening to computer vision research. NaverLabs is an ideal place for a PhD student and I know that I am extremely fortunate for having worked in such an environment.

I want to thank my PhD thesis reviewers: Mark and Anette for their positive and constructive feedback. I also want to thank my PhD committee: Jonathan, Eric, Katja, Miguel. You have made the defense a memorable day for me.

Finally, I want to thank my parents for always encouraging me to question and challenge myself; I also want to thank them and my wife who have been supportive for my PhD study. Finally I want to thank my daughter born during my PhD (who does not contribute to the thesis but certainly makes my life much more colorful).

Résumé

Notre but dans cette thèse est de construire un système qui réponde à une question en langue naturelle (NL) en représentant sa sémantique comme une forme logique (LF) et ensuite en calculant une réponse en exécutant cette LF sur une base de connaissances. La partie centrale d'un tel système est l'analyseur sémantique qui transforme les questions en formes logiques.

Notre objectif est de construire des analyseurs sémantiques performants en apprenant à partir de paires (NL, LF). Nous proposons de combiner des réseaux neuronaux récurrents (RNN) avec des connaissances préalables symboliques exprimées à travers des grammaires hors-contexte (CFGs) et des automates. En intégrant des CFGs contrôlant la validité des LFs dans les processus d'apprentissage et d'inférence des RNNs, nous garantissons que les formes logiques générées sont bien formées; en intégrant, par le biais d'automates pondérés, des connaissances préalables sur la présence de certaines entités dans la LF, nous améliorons encore la performance de nos modèles. Expérimentalement, nous montrons que notre approche permet d'obtenir de meilleures performances que les analyseurs sémantiques qui n'utilisent pas de réseaux neuronaux, ainsi que les analyseurs à base de RNNs qui ne sont pas informés par de telles connaissances préalables.

Abstract

Our goal in this thesis is to build a system that answers a natural language question (NL) by representing its semantics as a logical form (LF) and then computing the answer by executing the LF over a knowledge base. The core part of such a system is the semantic parser that maps questions to logical forms.

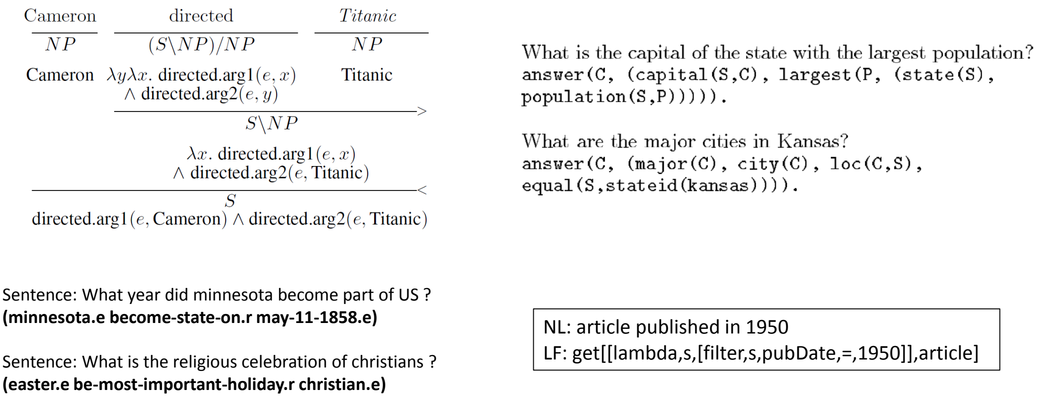
Our focus is how to build high-performance semantic parsers by learning from (NL, LF) pairs. We propose to combine recurrent neural networks (RNNs) with symbolic prior knowledge expressed through context-free grammars (CFGs) and automata. By integrating CFGs over LFs into the RNN training and inference processes, we guarantee that the generated logical forms are well-formed; by integrating, through weighted automata, prior knowledge over the presence of certain entities in the LF, we further enhance the performance of our models. Experimentally, we show that our approach achieves better performance than previous semantic parsers not using neural networks as well as RNNs not informed by such prior knowledge.

Contents

Analyse Sémantique avec Apprentissage Neuro-Symbolique

Le but de l'analyse sémantique est de convertir un texte en langue naturelle (NL) en une représentation sémantique (MR) qui peut être utilisée ou non dans une tâche en aval. Selon le contexte applicatif ou formel, il existe de nombreux types de représentations sémantiques.

La Fig. 1 montre quelques exemples de paires (NL, MR) à partir de différents jeux de données existants. En haut à gauche, nous montrons un exemple tiré de [Reddy *et al.*, 2014] où une phrase de NL est associée à un lambda-terme. Comme l'illustre cet exemple, de telles MR peuvent être obtenues en analysant les phrases NL avec une grammaire catégorielle combinatoire (CCG) [Steedman, 1996]. En haut à droite se trouve un exemple tiré de l'ensemble de données Geoquery [Zelle and Mooney, 1996] où la sémantique d'une question NL est représenté par une requête Prolog qui peut être exécutée à partir d'une base de données logique pour obtenir la réponse. En bas à gauche se trouve un exemple tiré de l'ensemble de données Wikianswers [Fader *et al.*, 2013] où une question NL est associée à un triplet qui contient à la fois la sémantique


Cameron
directed
Titanic
NP
$(S\backslash NP)/NP$
NP
Cameron
$\lambda y \lambda x.$ directed.arg1(e, x) $\wedge$ directed.arg2(e, y)
Titanic
$S\backslash NP$
$\lambda x.$ directed.arg1(e, x) $\wedge$ directed.arg2$(e,$ Titanic$)$
S
directed.arg1$(e,$ Cameron$)$ $\wedge$ directed.arg2$(e,$ Titanic$)$

What is the capital of the state with the largest population?

```
answer(C, (capital(S,C), largest(P, (state(S),
population(S,P))))).
```

What are the major cities in Kansas?

```
answer(C, (major(C), city(C), loc(C,S),
equal(S,stateid(kansas)))).
```

Sentence: What year did minnesota become part of US ?
(minnesota.e become-state-on.r may-11-1858.e)

Sentence: What is the religious celebration of christians ?
(easter.e be-most-important-holiday.r christian.e)

NL: article published in 1950
LF: get[[lambda,s,[filter,s,pubDate,=,1950]],article]

Figure 1: Exemples de paires (NL, MR) avec types de MR.

et la réponse à la question NL. Enfin, en bas à droite se trouve un exemple tiré de l'ensemble de données SPO (Semantic Parsing Overnight) [Wang *et al.*, 2015] où la MR est une requête complexe exécutable sur la base de connaissances associée (KB).

La recherche sur la construction d'analyseurs sémantiques a une longue histoire. Par exemple, LUNAR [Woods *et al.*, 1972], un système qui répond aux questions en langue naturelle sur les roches lunaires a été développé au début des années 70, parmi beaucoup d'autres systèmes similaires dont on trouvera un survol dans [Androutsopoulos, 1995]. Ces systèmes réussissaient à traiter des questions dans leurs domaines limités, mais comme ils étaient construits sur la base de règles spécifiées manuellement, ils étaient difficiles à appliquer à d'autres domaines.

Pour surmonter les limites des systèmes fondés sur des règles, les chercheurs ont commencé à étudier des systèmes d'apprentissage qui peuvent être entrainés sur des exemples, en particulier des paires annotées (NL, LF) [Zelle and Mooney, 1996; Wong and Mooney, 2006; Kate and Mooney, 2006; Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Kwiatkowski *et al.*, 2013]. Les approches proposées utilisent souvent des connaissances préalables (ou hypothèses linguistiques plausibles) pour réduire le nombre de candidats LF puis apprennent un classificateur (e.g. modèle log-linéaire, SVM) pour sélectionner ces candidats. Les systèmes développés ont atteint de bonnes performances sur plusieurs ensembles de données difficiles à l'époque, comme Geoquery [Zelle and Mooney, 1996] et ATIS [Dahl *et al.*, 1994].

Dans cette thèse, nous nous concentrons sur un cas spécifique d'analyse sémantique, appelée "analyse sémantique exécutable" dans la littérature [Liang, 2016], qui correspond à l'analyse de questions NL pour les convertir en requêtes sur une KB. Dans ce contexte, les représentations sémantiques sont des requêtes KB formelles et les résultats sont évalués en exécutant ces requêtes sur le KB cible et en comparant la ou les valeurs renvoyées avec la ou les réponses attendues. La fig. 2 illustre un tel processus. La phrase *"Which university did Obama go to ?"* est transformé en la représentation sémantique $\textsc{Type.University} \sqcap \textsc{Education.BarackObama}$ qui, lorsqu'elle est exécutée sur la KB fournit la réponse *"Occidental College, Columbia University"'* .

Nous considérons deux types distincts de requêtes KB, à savoir (i) les requêtes simples telles que (1a) dont les réponses sont contenues dans une très grande KB tel que Reverb [Fader *et al.*, 2011] et (ii) les requêtes complexes sur les KBs de petites et moyennes tailles [Wang *et al.*, 2015] telles que (1b).

(1) a. *What is the main language in Hong Kong ?*
(cantonese.e, be-major-language-in.r, hong-kong.e)

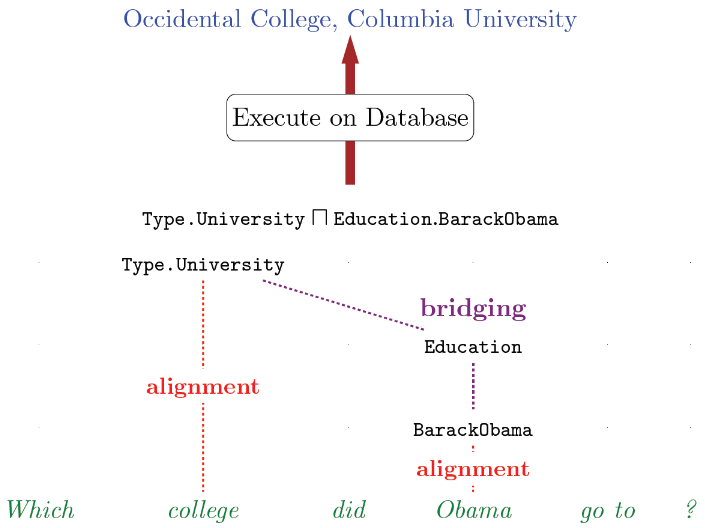


Figure 2: Exemple de transformation entre une question NL et une requête KB.

b. *How many fouls were played by Kobe Bryant in 2004?*
count(R(fouls).(player.KobeBryant ⊓ season.2004))

Dans le premier cas, la requête est structurellement simple et la sémantique est un fait qui peut être représenté par un triplet. La difficulté réside dans la correspondance appropriée entre mots NL et symboles KB. Par contre, dans le second cas, la requête peut être structurellement complexe et comporter un certain degré de compositionnalité.

Pour les requêtes simples, [Bordes *et al.*, 2014b] propose une approche d'apprentissage de la représentation qui apprend à la fois des représentations vectorielles de phrases/triplets KB et une fonction de score entre elles ce qui permet d'obtenir des résultats compétitifs. Nous suivons leur approche mais proposons certaines connaissances préalables que nous pouvons ajouter pour améliorer la performance. Dans le chapitre 4, nous montrons ainsi que la performance peut être améliorée en ajoutant un régularisateur d'orthogonalité qui distingue entre les entités et les relations.

Pour les requêtes plus complexes, à la suite de travaux réalisés par [Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015] démontrant la capacité des réseaux neuronaux récurrents (RNN) en traduction automatique, nous proposons d'adapter les approches basées sur les RNNs à l'analyse sémantique. En d'autres termes, au lieu de faire correspondre une phrase à sa traduction, nous proposons d'utiliser les RNNs pour transformer une requête NL en sa représentation de sémantique (linéarisée). Dans ce contexte, la principale question de recherche abordée dans cette thèse est la suivante:

> ***Question de recherche 1:*** *Dans quelle mesure les modèles fondés sur les RNNs peuvent-ils calculer la correspondance entre une requête de NL et sa MR?*

Nous commençons par appliquer un modèle basé sur les RNNs à des requêtes NL relativement complexes et trouvons qu'il améliore les résultats par rapport à un système d'analyse sémantique plus traditionnel basé sur des traits [features] définis manuellement [Wang *et al.*, 2015].

Toutefois, avec une quantité limitée de données, il est possible d'améliorer davantage la performance des modèles RNN en intégrant des connaissances préalables dans ces modèles. Une question importante lors de l'utilisation des RNNs pour l'analyse sémantique est qu'ils n'incluent pas de notion explicite de syntaxe et ne garantissent donc pas la bonne formation des représentations sémantiques obtenues, ni ne fournissent un support direct pour la compositionnalité. Autrement dit, les RNNs pour l'analyse sémantique soulèvent la question de recherche suivante:

> ***Question de recherche 1.1:*** *Comment un modèle de séquence à séquence peut-il être contraint de respecter la compositionnalité et de garantir la bonne forme syntaxique des représentations sémantiques produites?*

Nous abordons cette question au chapitre 5. Tout d'abord, nous remarquons que la bonne forme de la MR de sortie peut être assurée par une grammaire connue *a priori*. Sur la base de cette grammaire, nous proposons d'utiliser notre modèle basé sur les RNNs pour prédire les *séquences de dérivation* (DS) qui correspondent aux étapes de dérivation par rapport à la grammaire sous-jacente. La prédiction de DS facilite l'intégration des connaissances grammaticales préalables et garantit ainsi que la DS produite est toujours bien formée. Nous montrons empiriquement qu'un modèle basé sur les RNN intégrant ce type de connaissances grammaticales antérieures permet d'obtenir de meilleures performances que les méthodes RNN de base qui n'intègrent pas ces connaissances préalables.

Certains analyseurs sémantiques traditionnels (e.g. [Liang *et al.*, 2011; Berant *et al.*, 2013]) identifient les entités nommées en premier et utilisent les entités identifiées comme connaissances préalables pour ensuite rechercher la MR tout entière. Nous examinons comment nous pouvons intégrer des connaissances préalables similaires, dans notre modèle basé sur les RNNs, qui garantissent la bonne formation des DS produites:

> ***Question de recherche 1.2:*** *Comment traiter les entités nommées*

> *comme des connaissances préalables supplémentaires dans notre modèle basé sur les RNNs?*

Dans le chapitre 6, nous proposons d'aborder cette question en modélisant la probabilité que certaines règles correspondant à des entités nommées soient présentes, à travers l'utilisation d'automates pondérés (WFSA). Ces WFSAs permettent de fournir un biais à notre analyseur sémantique en favorisant ou désavantageant la présence de certaines règles lors de la prédiction. L'un des avantages de la modélisation de ce type de connaissances préalables par les WFSAs est qu'elles peuvent être combinées efficacement avec la WCFG que nous utilisons pour assurer la grammaticalité. Le résultat de la combinaison (qui est encore une WCFG) est utilisé comme 'Background' pour guider les prédictions de la séquence dérivationnelle par le RNN. Nous montrons empiriquement que cette approche RNN plus Background peut atteindre de meilleures performances que notre analyseur sémantique précédent, qui n'avait pas de connaissances préalables sur les entités nommées.

En conclusion, nous proposons d'utiliser les RNNs pour construire des systèmes d'analyse sémantique exécutables, comme l'ont aussi proposé d'autres chercheurs [Dong and Lapata, 2016; Jia and Liang, 2016; Xiao *et al.*, 2016b], et nous observons que bien que ces modèles puissent être bien adaptés à la tâche d'analyse sémantique, leurs performances peuvent être encore améliorées en incorporant des connaissances préalables dans le modèle. Nous proposons ainsi de construire des analyseurs sémantiques "neuro-symboliques" car ces connaissances préalables peuvent souvent être exprimées de manière appropriée sous certaines formes symboliques à travers l'utilisation d'automates et de grammaires.

Feuille de route

Nous divisons notre thèse en deux parties. Dans la première partie, nous présentons l'arrière-plan des outils qui seront utiles pour les analyseurs sémantiques que nous construirons dans les chapitres 5 et 6; cette première partie contient une revue historique des systèmes d'analyse sémantique et une introduction aux aspects des réseaux neuronaux et des automates/grammaires qui seront nécessaires plus tard. Dans la deuxième partie, nous décrivons les contributions principales de la thèse, à savoir la manière dont nous incorporons les connaissances préalables symboliques dans des modèles basés sur les réseaux neuronaux afin d'améliorer leurs performances pour l'analyse sémantique. Dans ce qui suit, nous résumons les différents chapitres.

Partie I Arrière-plan

Le chapitre 1 (Systèmes d'analyse sémantique exécutables) donne un aperçu de ces systèmes. Suivant le développement historique, nous présentations dans ce chapitre des analyseurs sémantiques classiques basés sur des règles (par exemple, [Woods *et al.*, 1972]), puis des analyseurs statistiques qui peuvent apprendre à partir de données constituées de paires annotées (NL, LF) (par exemple, [Wong and Mooney, 2006; Zettlemoyer and Collins, 2005]); puis, nous discutons de quelques directions de recherche récentes sur les analyseurs sémantiques visant à réduire les efforts d'annotation (e.g. [Liang *et al.*, 2011]) et/ou à réduire les efforts d'ingénierie en utlisant des modèles plus puissants (e.g. [Bordes *et al.*, 2014a]).

Le chapitre 2 (Réseaux neuronaux) traite de plusieurs architectures de réseaux neuronaux (i.e. Perceptrons multicouches (MLPs) et RNNs) qui sont largement utilisées actuellement pour l'apprentissage d'analyseurs sémantiques (e.g. [Dong and Lapata, 2016; Xiao *et al.*, 2016b]. Ces réseaux sont importants pour nous car il s'agit de modèles paramétriques expressifs que l'on peut apprendre à partir des données. Nous utilisons largement ces réseaux neuronaux dans les contributions présentées aux chapitres 5 et 6.

Chapitre 3 (Automates et grammaires) propose une brève introduction aux automates et grammaires hors-contexte (CFGs) et à leurs versions pondérées. Nous discutons également de l'algorithme d'intersection entre un automate pondéré et une CFG pondérée. Nous utilisons ces objets symboliques pour exprimer des connaissances préalables pour des tâches d'analyse sémantique. [1]

Partie II Contributions

Le chapitre 4 (Plonglements Orthogonaux pour l'Analyse Sémantique de Requêtes Simples) discute d'une tâche d'analyse sémantique où pour répondre à la question, il faut trouver un triplet correct de la forme (e_1, r, e_2) où e_1, e_2 sont des entités et r est une relation. Comme nous l'avons mentionné plus haut, dans cette tâche, la principale difficulté réside dans l'identification correcte de la transformation entre les mots NL et les symboles KB. Sur la base des méthodes de plongements [embeddings] proposées par [Bordes *et al.*, 2014b], nous montrons que l'intégration des connaissances préalables qui sépare les plongements d'entités et les plongements de relations, améliore les résultats antérieurs.

[1] Les algorithmes d'intersection sont utiles lorsque nous combinons les connaissances préalables de différentes sources.

Le chapitre 5 (Analyse Sémantique Neuronale sous Connaissances Grammaticales Préalables) se concentre sur l'analyse sémantique de requêtes NL plus complexes. Nous proposons d'utiliser une architecture basée sur les RNNs pour apprendre un analyseur sémantique et montrer qu'il est plus performant que les analyseurs sémantiques basés sur des techniques d'apprentissage plus traditionnelles. Nous montrons également que les performances peuvent être encore améliorées en intégrant une CFG *a priori* sur les LFs. En intégrant cette CFG, nous garantissons que les LFs produits par notre système sont grammaticalement correctes.

Le chapitre 6 (Automates en tant que Sources de Connaissances Additionnelles, Modulaires, A Priori) propose une extension par rapport à l'analyseur sémantique décrit au Chapitre 5 où nous ajoutons des connaissances préalables supplémentaires sur certaines entités présentes dans la LF, basées sur la NL observée; ces connaissances préalables sont exprimées à l'aide d'automates qui peuvent être combinés efficacement avec la CFG que nous utilisions précédemment, via l'algorithme d'intersection dont nous avons parlé au Chapitre 3. Nous montrons que l'analyseur sémantique étendu de cette façon améliore les performances.

Dans le chapitre 7, nous tirons les conclusions de ce travail et proposons des perspectives et des indications pour des travaux futurs.

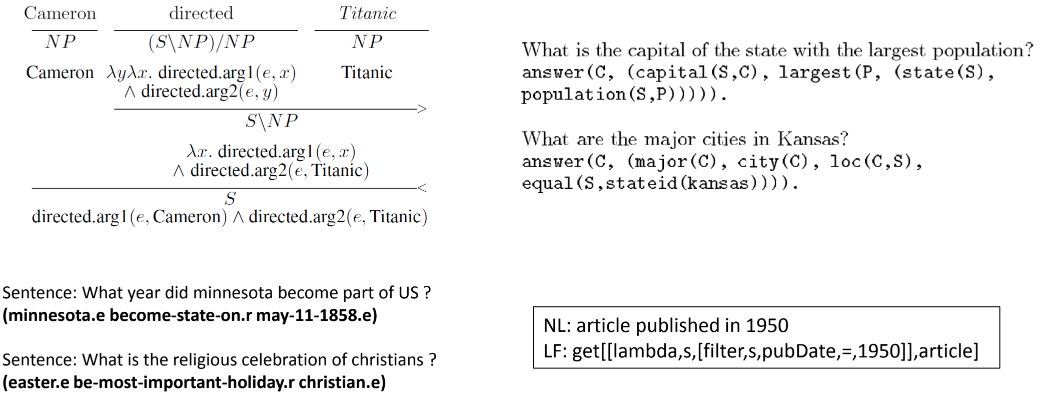
List of Figures

List of Tables

Introduction

The aim of semantic parsing is to convert Natural Language (NL) into a meaning representation (MR) which may or not be used in a downstream task. Depending on the applicative or formal context, there are many different types of meaning representations.

Fig. 1 shows some examples of (NL, MR) pairs from different existing datasets. On the top left is an example taken from [Reddy *et al.*, 2014] where an NL sentence is mapped to a lambda term. As shown in this example, such MRs can be obtained by parsing NL sentences with a combinatorial categorical grammar (CCG) [Steedman, 1996]. On the top right is an example taken from the Geoquery dataset [Zelle and Mooney, 1996] where the meaning of an NL question is represented by a Prolog query which can be executed against a Prolog database to obtain the answer. On the bottom left is an example from the Wikianswers dataset [Fader *et al.*, 2013] where an NL question is associated with a single triple that contains both the meaning and the answer to the NL question. Finally, on the bottom right is an example from the Semantic Parsing Overnight (SPO) dataset [Wang *et al.*, 2015] where the MR is a complex query executable against the associated knowledge base (KB).


Cameron
directed
Titanic
NP
$(S\backslash NP)/NP$
NP
Cameron
$\lambda y \lambda x.$ directed.arg1(e, x) $\wedge$ directed.arg2(e, y)
Titanic
>
$S\backslash NP$
$\lambda x.$ directed.arg1(e, x) $\wedge$ directed.arg2$(e,$ Titanic$)$
<
S
directed.arg1$(e,$ Cameron$)$ $\wedge$ directed.arg2$(e,$ Titanic$)$

What is the capital of the state with the largest population?

```
answer(C, (capital(S,C), largest(P, (state(S),
population(S,P))))).
```

What are the major cities in Kansas?

```
answer(C, (major(C), city(C), loc(C,S),
equal(S,stateid(kansas)))).
```

Sentence: What year did minnesota become part of US ?
(minnesota.e become-state-on.r may-11-1858.e)

Sentence: What is the religious celebration of christians ?
(easter.e be-most-important-holiday.r christian.e)

NL: article published in 1950
LF: get[[lambda,s,[filter,s,pubDate,=,1950]],article]

Figure 1: Examples of (NL, MR) pairs with types of MR.

Research on building semantic parsers has a long history. For example, LUNAR [Woods *et al.*, 1972], a system that answers natural language questions about moon rocks was developped in the early 70's amongst many other similar systems for which a review can be found in [Androutsopoulos, 1995]. Those systems are successful at handling questions within their limited domains, however, as the systems were built on manually specified rules, they were difficult to apply to other domains.

To overcome the limitations of rule-based systems, researchers started to investigate machine learning systems that can learn from examples such as annotated (NL, LF) pairs [Zelle and Mooney, 1996; Wong and Mooney, 2006; Kate and Mooney, 2006; Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Kwiatkowski *et al.*, 2013]. The approaches proposed first use prior knowledge (or linguistic plausible hypotheses) to reduce the number of LF candidates and then learn a classifier (e.g log-linear model, SVM) over those candidates. The systems developped achieved good performance on several challenging datasets at the time such as Geoquery [Zelle and Mooney, 1996] and ATIS [Dahl *et al.*, 1994].

In this thesis, we focus on a specific case of semantic parsing dubbed "executable semantic parsing" in the literature [Liang, 2016], which is the parsing of NL questions into KB queries. In that setting, meaning representations are formal KB queries and results are evaluated by executing these queries on the target KB and comparing the returned value(s) with the expected answer(s). Figure 2 illustrates such a process. The sentence *"Which college did Obama go to?"* is mapped to the meaning representation Type.University ⊓ Education.BarackObama which, when executed on the KB yields the answer *"Occidental College, Columbia University"*.

We consider two distinct types of KB queries namely, (i) simple queries such as (2a) whose answers are contained in a very large KB such as Reverb [Fader *et al.*, 2011] and (ii) complex NL queries on small to medium size KBs [Wang *et al.*, 2015] such as (2b).

(2) a. *What is the main language in Hong Kong ?*
(cantonese.e, be-major-language-in.r, hong-kong.e)

b. *How many fouls were played by Kobe Bryant in 2004?*
count(R(fouls).(player.KobeBryant ⊓ season.2004))

In the first case, the query is structurally simple and the semantics is a fact which can be represented by a single triple. The difficulty resides in appropriately mapping NL words to KB symbols. In the second case, the query can be structurally complex involving a fair amount of compositionality.

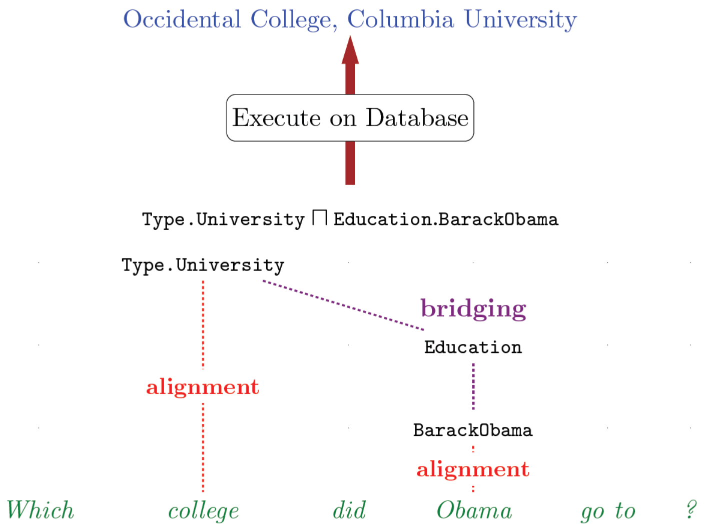


Figure 2: Examples of mapping an NL question to a KB query.

For simple queries, [Bordes *et al.*, 2014b] propose a representation learning approach that learns both vectorial representations of sentences/KB triples and a scoring function between them, which achieves competitive results. We follow their approach and consider the prior knowledge we can add to further improve the performance. In Chapter 4, we show that the performance can be improved by adding an orthogonality regularizer that distinguishes KB entities from relations.

For the more complex queries, following work by [Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015] demonstrating the ability of recurrent neural networks (RNNs) to translate sentences, we propose to adapt RNN-based approaches to semantic parsing. That is, instead of mapping a sentence to its translation, we propose to use RNNs to map an NL query to its (linearized) meaning representation. In this context, the main research question addressed in this thesis is the following:

> ***Research Question 1:*** *To what extent can RNN-based models capture the mapping between an NL query and its MR?*

We start by applying an RNN-based model to fairly complex NL queries and find that it can improve over a more traditional semantic parsing system based on handcrafted features [Wang *et al.*, 2015].

However, with limited amount of data, the performance of RNN models can be further enhanced by incorporating prior knowledge into RNNs. An important issue when using RNNs for semantic parsing is that they do not include any explicit notion of syntax and therefore neither guarantee the well-formedness of the output meaning representations nor provide clear support for compositionality. In other words, RNNs for semantic parsing raise the following research question:

> ***Research Question 1.1:*** *How can a sequence-to-sequence model be constrained to support compositionality and guarantee the syntactic well-formedness of the output meaning representations ?*

We address this issue in Chapter 5. First we note that the well-formedness of the output MR is ensured by a grammar which is known *a priori*. Based on this grammar, we propose to use our RNN-based model to predict *derivation sequences* (DS) that are derivation steps relative to the underlying grammar. Predicting DS eases the task of integrating grammatical prior knowledge thus guarantees that the produced DS is always grammatically correct. We show empirically that an RNN-based model integrating grammatical prior knowledge achieves better performance than various RNN baselines that do not integrate this prior knowledge.

Some traditional semantic parsers (e.g. [Liang *et al.*, 2011; Berant *et al.*, 2013]) often identify named entities first and use identified entities as prior knowledge to then search for the whole MR. We examine how we can additionally integrate similar prior knowledge into our RNN-based model that guarantees the well-formedness of produced DS:

> ***Research Question 1.2:*** *How can named entities be handled as additional prior knowledge by our RNN-based model ?*

In Chapter 6, we propose to address this issue by modeling the probability of certain rules corresponding to named entities being present, using weighted automata (WFSAs). These WFSAs can provide a bias to our semantic parser by favoring/disfavoring the presence of certain rules during DS prediction. An advantage of modelling this kind of prior knowledge by WFSAs is that they can be combined efficiently with a WCFG that we use to ensure grammaticality. The result of the combination (which is still a WCFG) is used as 'Background' to guide RNN DS predictions. We empirically show that our Background RNN approach can achieve better performance than our previous semantic parser that did not have the prior knowledge over named entities.

In conclusion, we propose to use RNNs to build executable semantic parsing systems, along with other researchers [Dong and Lapata, 2016; Jia and Liang, 2016; Xiao *et al.*, 2016b] and see that although these models can be well adapted for the semantic parsing task, their performance can be further enhanced by incorporating prior knowledge into the model. In this thesis, we focus on prior knowledge of various sources that can be combined with RNN models; we propose to build 'neural-symbolic' semantic parsers as we often find that our prior knowledge can be expressed appropriately in some symbolic forms by using automata and grammars.

Roadmap

We divide our thesis into two parts. In the first part, we introduce some background information that will be useful for the semantic parsers that we build in Chapter 5 and 6; the background part contains a historical review of semantic parsing systems and an introduction to aspects of neural networks and of automata/grammars that are needed later. In the second part, we describe the contributions of the thesis concerning how we incorporate symbolic prior knowledge into neural-network based models to enhance their performance for semantic parsing. In what follows, we summarize the different chapters.

Part I Background

Chapter 1 (Executable Semantic Parsing Systems) gives a review of these systems. Following the historical timeline, we discuss in this chapter rule-based semantic parsers (e.g. [Woods *et al.*, 1972]), then statistical ones that can learn from data consisting of annotated (NL, LF) pairs (e.g. [Wong and Mooney, 2006; Zettlemoyer and Collins, 2005]); then, we discuss some current research directions on semantic parsers aiming at reducing annotation efforts (e.g. [Liang *et al.*, 2011]) and/or reducing engineering efforts by using more powerful learning machines (e.g. [Bordes *et al.*, 2014a]).

Chapter 2 (Neural Networks) discusses several neural network architectures (i.e. Multilayer Perceptrons (MLPs) and RNNs) that are widely used for learning semantic parsers these days (e.g. [Dong and Lapata, 2016; Xiao *et al.*, 2016b]). These networks are important for us as they are expressive parametric models that can be learned from data. We largely use these neural networks in the contributions presented in Chapters 5 and 6.

Chapter 3 (Automata and Grammars) gives a brief introduction to automata and context-free grammars (CFGs) and their weighted versions. We also discuss the intersection algorithm between a weighted automaton and a weighted CFG. We use those symbolic objects to express prior knowledge for semantic parsing tasks.[2]

Part II Contributions

Chapter 4 (Orthogonal Embeddings for the Semantic Parsing of Simple Queries) discusses a semantic parsing task where to answer the question, one has to find a correct triple of the form (e_1, r, e_2) where e_1, e_2 are entities and r is a relation. As mentioned above, in that task, the main difficulty resides in correctly identifying the mapping between NL tokens and KB symbols. Based on the embedding methods proposed by [Bordes *et al.*, 2014b], we show that integrating prior knowledge which separates entity embeddings and relation embeddings, improves over previous results. The content of this chapter is reported in [Xiao *et al.*, 2016a].

Chapter 5 (Neural Semantic Parsing under Grammatical Prior Knowledge) focuses on the semantic parsing of more complex NL queries. We propose to use an RNN-based architecture to learn a semantic parser and show that it can perform better than semantic parsers based on more traditional learning machines. We also show that its performance can be further enhanced by integrating the CFG known *a priori* over LFs. By integrating this CFG, we guarantee that the LFs produced by our system are grammatically correct. The content of this chapter is reported in [Xiao *et al.*, 2016b].

Chapter 6 (Automata as Additional, Modular, Prior Knowledge Sources) proposes an extension over the semantic parser described in Chapter 5 where we add additional prior knowledge about certain entities being present in the corresponding LF based on the observed NL; the prior knowledge is expressed using automata that can be combined efficiently with the CFG that we used previously, via the intersection algorithm we discussed in Chapter 3. We show that the extended semantic parser improves in performance. The content of this chapter is reported in [Xiao *et al.*, 2017].

In Chapter 7 we draw conclusions and give perspectives and pointers for future work.

[2] Intersection algorithms are useful when we combine the prior knowledge of different sources.

Part I

Background

Chapter 1

Executable Semantic Parsing Systems

Contents

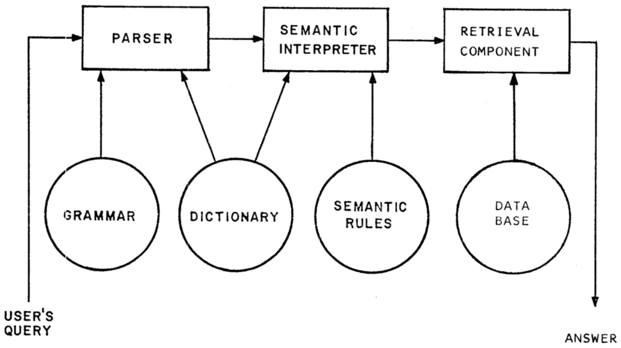


Figure 1.1: LUNAR question answering system with its main components.

1.1 Early Developments

The history of executable semantic parsing systems is rich and a survey of the early systems can be found in [Androutsopoulos, 1995]. In this section, we briefly review one specific, probably the best known, early executable semantic parsing system, namely LUNAR [Woods *et al.*, 1972]. LUNAR is a prototype which allows English language access to a large database of lunar sample information. The system is rule-based and is one of the earliest systems designed to handle natural language queries formulated by real users. LUNAR will also serve as a motivating example for us in this section to discuss two important developments around executable semantic parsing systems, namely powerful grammar formalisms that can take into account syntax and semantics at the same time and executable semantic parsing systems that can be learned from data.

1.1.1 LUNAR

To answer a natural language query (NL) formulated by a user, LUNAR proposes a design consisting of three components. The first component (PARSER) syntactically analyses the input query NL and assigns it a parse tree. The second component (SEMANTIC INTERPRETER) uses this parse tree to produce the LF which is the semantic interpretation of the input utterance.[3] The third component (RETRIEVAL COMPONENT) directly executes the LF output from the SEMANTIC INTERPRETER, similar to the process where an SQL query (or other query lan-

[3] While the SEMANTIC INTERPRETER is dependent on the KB and its specified query language, the PARSER is more domain independent and can hopefully be easily adapted to other executable semantic parsing tasks.

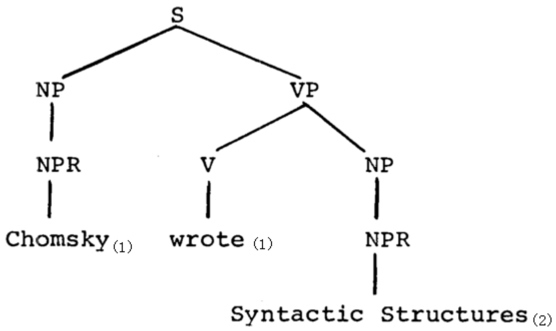
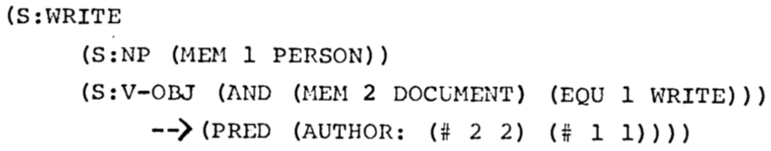


Figure 1.2: Syntactic Tree produced by LUNAR for the sentence “Chomsky wrote Syntactic Structures”.



```
(S:WRITE
     (S:NP (MEM 1 PERSON))
     (S:V-OBJ (AND (MEM 2 DOCUMENT) (EQU 1 WRITE)))
          -->(PRED (AUTHOR: (# 2 2) (# 1 1))))
```

Figure 1.3: Semantic rule pattern matched for “Chomsky wrote Syntactic Structures”.

guages such as SPARQL etc.) is executed over an SQL database, then return the execution results to the user.

Fig. 1.1 shows the major components of LUNAR. As the design of RETRIEVAL COMPONENT is more related to research fields such as database and query language while we focus on semantic parsing in this thesis, in the following, we only describe the process of mapping NL to LF and do not discuss the RETRIEVAL COMPONENT further.

We will use the sentence “Chomsky wrote Syntactic Structures” as an illustration and show how LUNAR deals with this example to parse this sentence into a logical form (LF).

When receiving the input sentence, LUNAR will first call a syntactic parser (PARSER) which analyses the sentence using a phrase structure grammar and produces the phrase structure tree shown in Fig. 1.2. Note that the terminals are numbered according to the convention defined in LUNAR. This parse tree will be passed to the next component to produce the LF.

The SEMANTIC INTERPRETER relies on semantic rules to extract the LF from the parse tree. Semantic rules consist of rules whose left-hand side (LHS) is a template composed of elements in the parse tree involving both terminals and nonterminals and whose right-hand side (RHS) is the semantic interpretation of such templates. For example, consider the rule in Fig. 1.3.

On its LHS, the rule specifies a template consisting of a tree fragment plus additional semantic conditions on the numbered nodes of the fragment. More precisely, the above LHS has two components: the first component is an NP nonterminal and the second component is a V-OBJ nonterminal; the first component needs to in addition satisfy the semantic condition (MEM 1 PERSON) stating that the node numbered 1 should be a person (MEM ... PERSON) and similar conditions exist for the second component. Thus, the LHS indicates that if the sentence has a subject which is a person, a verb "write",[4] and an object of "write" which is a document then the template matches the string and will produce the semantic interpretation specified on the RHS.

The semantic interpretation specified by the RHS of this rule is (PRED (AUTHOR: (# 2 2) (# 1 1)) which is a predicate AUTHOR with two arguments. The RHS indicates that the meaning of the sentence is computed by substituting the interpretations of the node number 1 (# **1** 1) in the first component (# 1 **1**, which is the NP component in our example) and the node number 2 in the second component (# 2 2) into the indicated argument places.

This rule can be used to give a semantic interpretation for our sentence "Chomsky wrote Syntactic Structures". The LHS of our semantic rule matches the parse tree as the sentence verifies both syntactical and semantic conditions. Then the semantic interpretation is constructed by substituting the first argument of predicate AUTHOR by "Syntactic Structures" and the second argument by "Chomsky" and we obtain the semantics PRED (AUTHOR: (Syntactic Structures) (Chomsky)).

Limits of the system Despite the success and achievements of LUNAR, the system has several drawbacks.

First, the system is not robust in several aspects. The system is based on a limited vocabulary (3500 words) and cannot deal with words out of this vocabulary. In particular, the system cannot deal with spelling errors. The system will fail on ungrammatical sentences as the very first component PARSER will fail.

Secondly, the engineering effort required to make such a system work in practice is considerable. In the LUNAR technical report [Woods *et al.*, 1972], the authors describe in detail the grammatical rules (for PARSER) and semantic rules (for SEMANTIC INTERPRETER) used in the system. The writing merely of these rules takes more than 100 pages in the report (grammatical rules in pages 159-214 and se-

[4] In practice, a morphological analyzer is used so that the rule recognizes the verb "write" as well as its variants such as "writes", "wrote", etc.

mantic rules in pages 217-263) with semantic rules not transferable to other domains.

Discussion

- LUNAR calculates the semantics of a sentence in two steps: parse syntactically the sentence (PARSER) and then produce the semantics based on the syntax tree (SEMANTIC INTERPRETER). The SEMANTIC INTERPRETER is domain-specific and only the PARSER can be used to adapt to other domains. One may hope for a general grammar formalism that is able to produce both syntax and semantics that are not domain specific. Several research directions have been raised (which we will discuss in subsection 1.1.2) with progress being made in the later research for building executable semantic parsing systems.
- With the rise of machine learning, one may hope to avoid writing rules and let a machine learn the mappings (under or not under the forms of rules) between NL and LF using training data consisting of pairs (NL,LF). This new paradigm not only opens doors for a significant reduction of engineering effort, but also improves robustness because problems such as spelling errors or ungrammatical sentences will be seen in training and the model will learn to deal with these aspects. In subsection 1.1.3 we review some early attempts trying to incorporate machine learning for building executable semantic parsing systems.

Remarks To reduce engineering effort to build executable semantic parsing systems, researchers in the 70s and the 80s proposed to focus on controlled languages [Hendrix *et al.*, 1978; Warren and Pereira, 1982]. As this is not the focus of the thesis, we refer interested readers to the above references.

1.1.2 Grammar formalisms dealing with both syntax and semantics

Many grammar formalisms such as LFG (Lexical Functional Grammar) [Dalrymple, 2001], CCG (Combinatorial Categorical Grammar) [Steedman, 1996], UCG (Unification Categorical Grammar) [Calder *et al.*, 1988], HPSG (Head-driven Phrase Structure Grammar) [Pollard and Ivan A., 1994], and TAG (Tree Adjoining Grammar) [Joshi and Vijay-Shanker, 2001] have been proposed so that both syntax and semantics can be analyzed simultaneously. We will give examples on LFG and CCG as illustrations.

LFG assumes two syntactic representation levels. Constituent structure (c-structure) encodes the sentence into a phrase structure tree and Functional structure (f-structure)

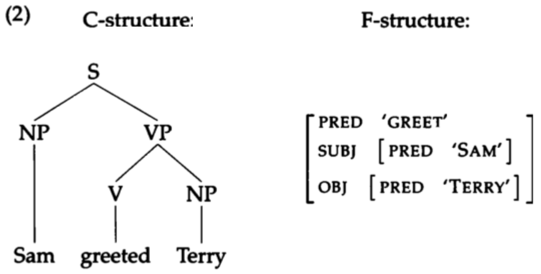


Figure 1.4: An example of the c-structure and the f-structure in LFG for "Sam greeted Terry".

encodes the syntactic predicate argument structure of the sentence. LFG uses the following type of syntactic rules to produce the c-structure and f-structure:

$$\mathbf{S} \rightarrow \mathbf{NP}((\uparrow \mathit{SUBJ}) =\downarrow)\ \mathbf{VP}(\uparrow=\downarrow)$$
$$\mathbf{VP} \rightarrow \mathbf{V}(\uparrow=\downarrow)\ \mathbf{NP}((\uparrow OBJ) =\downarrow)$$

These rules indicate the c-structure as in a CFG. For example, the nonterminal **S** expands to two nonterminals **NP**, **VP**.

The rules also indicate the f-structure by the content in the parenthesis. For example, the rule expanding **S** (noted by the metavariable ↑ on the **NP** node) says that **S** has a SUBJ attribute whose value is the f-structure for the **NP** daughter (noted by the metavariable ↓ on the **NP** node) , and that the **S** node corresponds to an f-structure which is the same as (i.e "unified" with) the f-structure for the **VP** daughter. A similar rule involving c-structure and f-structure also exists for **VP**.

Fig. 1.4 shows an example of the resulting c-structure and f-structure after applying these rules to the sentence "Sam greeted Terry". The c-structure is a typical derivation tree and the f-structure is encoded in key-value pairs.

[Dalrymple, 2001] proposes to use semantic rules to further construct the semantics of the whole sentence based on the f-structure of the sentence. For instance, a rule on predicate *greet* indicating that if the subject means X and the object means Y, then the sentence means $greet(X, Y)$ allows us to construct the semantics $greet(Sam, Terry)$ for the sentence "Sam greeted Terry".

CCG [Steedman, 1996] is a popular grammar formalism inside categorical grammar where elements like verbs are associated with a syntactic "category" which identifies them as functions, and specifies the type and directionality of their arguments and the type of their result. For example, a transitive verb is a function from an (object) NP into a predicate — that is, into a function from (subject) an NP into

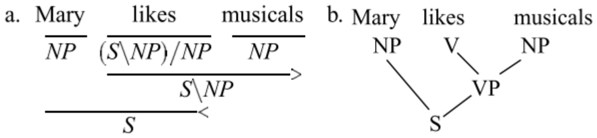


Figure 1.5: Comparison of CCG and CFG for parsing the sentence "Mary likes musicals".

an S:

$$likes := (S \backslash NP)/NP$$

CCG uses the "result leftmost" notation in which a rightward-combining functor over a domain β into a range α is written α/β, while the corresponding leftward combining functor is written $\alpha\backslash\beta$. α and β may themselves be function categories:

- Forward Application (>): $X/Y \ Y \Rightarrow X$
- Backward Application (<): $Y \ X\backslash Y \Rightarrow X$

Fig. 1.5 shows how CCG uses these types of rules to parse the sentence and compares the CCG parsing procedure with classic CFG ones. Note that the major burden of specifying particular grammars is transferred from phrase structure rules to lexical rules.

CCG categories can be regarded as encoding semantic types, making it possible to combine words to form the semantic interpretation of the sentence. For example, we can specify the semantic translation of the verb 'likes' as:

$$likes := (S \backslash NP_{3s})/NP : like'$$

where $like'$ is the semantics[5]. We also specify the Forward/Backward functional application on top of our category application:

- Forward Application (>): $X/Y : f \quad Y : a \ \Rightarrow \ X : fa$
- Backward Application (<): $Y : a \quad X\backslash Y : f \ \Rightarrow \ X : fa$

Applying these rules on the sentence 'Mary likes musicals' yields the semantics $like'musicals'mary'$ as shown in Fig. 1.6.

As we shall see in the later chapters, these grammar formalisms, notably CCG have played important roles in the development of executable semantic parsing systems [Zettlemoyer and Collins, 2005; Reddy *et al.*, 2014]. Using those formalisms,

[5] $3s$ in the CCG rule is a feature standing for third person singular.

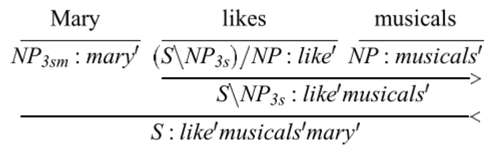


Figure 1.6: Semantics construction for the sentence “Mary likes musicals”.

generic LFs that are close to the input sentence can be first produced, before being further translated into specific query languages to be executed over the knowledge base [Reddy *et al.*, 2014].

1.1.3 Early executable semantic parsing systems learned from Data

[Miller *et al.*, 1996; Schwartz *et al.*, 1996] are amongst the earliest approaches learning executable semantic parsing systems from data. Similar to LUNAR, [Miller *et al.*, 1996; Schwartz *et al.*, 1996] propose an architecture relying on the CFG-like syntactic tree to map NLs to LFs. The differences with LUNAR are two folds: the models [Miller *et al.*, 1996; Schwartz *et al.*, 1996] try to generate the syntactic trees directly labelled with semantic classes thus avoiding the two components design and they are directly learned from data.

To train their model, the authors use the ATIS corpus. ATIS is a dataset that contains air travel information such as flight departure/arrival cities, departure/arrival time, etc. In this corpus, each NL is paired with an LF written in frames. For example, the sentence in Fig. 1.7 will be paired with the LF *(Frame: TOLOC.TIME Slots: (FROMLOC.CITY: BOSTON, TOLOC.CITY=Atlanta))* where *Frame* denotes the required information and *Slots* list the constraints. A more detailed description of the ATIS dataset can be found in 1.2.1 inside this thesis.

The authors propose to enrich the initial ATIS dataset with syntactic trees whose nodes are also annotated with semantic classes. More precisely, each sentence is associated with a syntactic tree whose nodes are labelled with both a syntactic and sometimes a semantic class.[6] Fig. 1.7 illustrates this annotation. For example, the word ‘when’ has the semantics ‘time’ and has the syntactic structure ‘wh-head’; the vp node just below the /wh-question root is associated with the semantics ‘arrival’, etc.

Note that at test time, if one succeeds in producing such a semantic/syntactic tree, it may be easier in the following to convert the semantic information in the

[6] Because not every word carries semantics (e.g. ’do’, ’the’ in the sentence of Fig. 1.7), one can note in the figure that every node is associated with a syntactic class but not always with a semantic one.

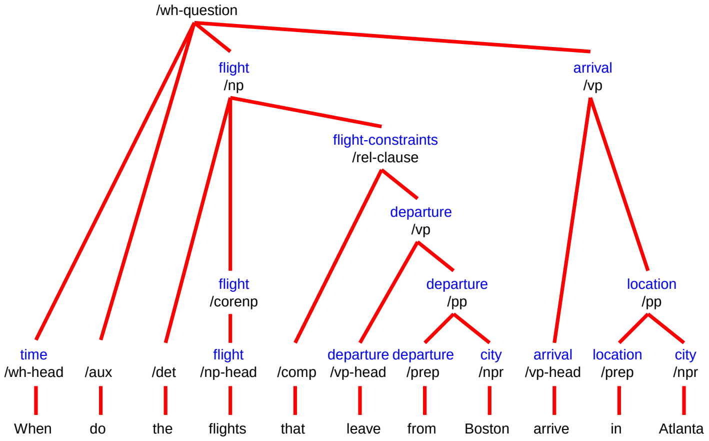


Figure 1.7: Annotations for the sentence "When do the flights that leave from Boston arrive in Atlanta".

tree to LFs written in frames if necessary. Either one can conceive a system using rules: for example, from the tree in Fig. 1.7, it is trivial to extract the *Slots* indicating the departure and arrival cities; the *Frame* can be further deduced as the /wh-question is associated with both 'arrival' and 'time' semantics; or one can use some probabilistic modelling when the semantics in the tree presents ambiguity. In either cases, the produced semantic/syntactic tree is directly useful for producing required LFs: semantic labels identify the basic units of meaning, while syntactic structures help identify relationships between those units. Thus, in the following, we will just focus on the methods proposed by the authors [Miller *et al.*, 1996; Schwartz *et al.*, 1996] to learn to generate the semantic/syntactic trees.

The authors choose probabilistic models to predict the semantic/syntactic tree; let W be a sentence and T be a semantic/syntactic parse tree.[7] One would like to estimate $P(T|W)$. According to Bayes rules:

$$P(T|W) = \frac{P(T)P(W|T)}{P(W)}$$

Since $P(W)$ is constant for any given word string, candidate parses T can be ranked

[7] Here, T denotes a parse tree not containing terminals.

by considering only the product $P(T)P(W|T)$. Both $P(T)$ and $P(W|T)$ are directly estimated from data using markovian assumptions; the probability $P(T)$ is modeled by transition probabilities between nodes in the parse tree, and $P(W|T)$ is modeled by word transition probabilities:

- $P(T)$ takes the form $P(node_n|node_{n-1}, node_{up})$ where $node_{n-1}$ is the node having the semantic/syntactic information of the previous preterminal node[8] and $node_{up}$ is the node having the semantic/syntactic information of the nonterminal at one level above. For example, $P(location/pp)$ in Fig. 1.7 is conditioned on $arrival/vp - head$ (i.e $node_{n-1}$) and $arrival/vp$ (i.e $node_{up}$) according to the model.

- $P(W|T)$ is modelled by word transition probability $P(word_n|word_{n-1}, preterminal)$ where $word_{n-1}$ is the previous word and *preterminal* is the syntactic/semantic node directly associated with the word to be generated. For example, the probability of word 'Boston' in Fig. 1.7 is conditioned on 'from' (i.e $word_{n-1}$) and *city/npr* (i.e *preterminal*) according to the model.

These probabilities can be estimated directly from counts in the data followed by smoothing techniques to take into account unseen words or nodes.[9]

The authors [Schwartz *et al.*, 1996] train their model on 4500 utterances of the ATIS dataset and test it on the ATIS DEC94 dataset [Dahl *et al.*, 1994]. The system achieves an accuracy of 0.86 on the test data. For the test utterances of Class A (the sentences whose meanings do not depend on the context), the system achieves an accuracy of 0.91.[10]

Discussion Compared to a manually written rule based system such as LUNAR, these works on executable semantic parsing show promising directions to *learn* a semantic parser from annotated data. However, a major issue with the above approaches is that they require a large amount of effort for actually annotating intermediate results (i.e. semantic/syntactic trees). As we shall see, approaches proposed later try to reduce this annotation effort by either treating the parse trees as hidden variables or by estimating a model directly mapping an NL to the corresponding LF.

[8] We refer to preterminal nodes the nodes just above the words, which are sometimes called tags also in the literature.

[9] We refer readers interested in smoothing techniques to the paper [Miller *et al.*, 1996].

[10] Note that these scores are produced by evaluating the accuracy over LFs written in frames; the system evaluated thus contains the parsing model described so far and a model converting the semantics in the tree to LFs written in frames that we did not describe.

1.2 Machine Learning Approaches

In this section, we will discuss some semantic parsing approaches [Kate *et al.*, 2005a; Kate and Mooney, 2006; Wong and Mooney, 2006; Wong and Mooney, 2007; Zettlemoyer and Collins, 2007; Zettlemoyer and Collins, 2005] having both machine learning and grammar aspects. We choose to discuss these approaches in this section for several reasons. First, they are natural improvements over the early semantic parsing systems learned from data that we have discussed in subsection 1.1.3 as they eliminate the needs for intermediate annotations. Secondly, those systems were state of the art systems tested on existing public datasets where the results can be directly compared. Finally, the drawbacks that those systems reveal (e.g the rule application is too strict to take into account many phenomena in natural language questions) and the questions that we have on those systems (e.g can those systems extend to much larger domains?) motivate further research on semantic parsing systems which we will describe in section 1.3.[11]

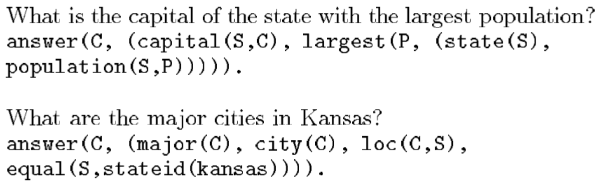
Caveat. There is a lot of work around building semantic parsing systems that we will not discuss here. For example, we will not discuss inductive logic programming (ILP) approaches [Zelle and Mooney, 1996; Tang and Mooney, 2001] as the proposed systems learn rules to build deterministic semantic parsers thus do not reflect the statistical machine learning aspects we focus on in this thesis[12]. We will also not discuss [Papineni *et al.*, 1997] which proposes to use a log-linear model to rank all logical forms (LFs) given an NL input. While the idea of simply ranking all LFs is very attractive as we do not need to rely on linguistic assumptions about the NL, unfortunately, the proposed approach can not generalize to situations where the number of LFs explode.

1.2.1 Datasets

Datasets play an important role in machine learning approaches. For semantic parsing, given a dataset consisting of (NL, LF) pairs, the systems fit a model (parametrized or non-parametrized) on this training data with the objective that the learned model will perform well on test data by predicting a correct LF for an unseen NL.

[11] Another reason is that the works we choose to review in this section are between 2000-2010, so the section order also fits the timeline.

[12] [Kate *et al.*, 2005a] also builds a deterministic semantic parser, however their proposed method is based on SCFG and lays the ground for papers that later on incorporate statistical machine learning [Kate and Mooney, 2006; Wong and Mooney, 2006; Wong and Mooney, 2007].



What is the capital of the state with the largest population?

```
answer(C, (capital(S,C), largest(P, (state(S),
population(S,P))))).
```

What are the major cities in Kansas?

```
answer(C, (major(C), city(C), loc(C,S),
equal(S,stateid(kansas)))).
```

Figure 1.8: Examples of (NL,LF) in the Geoquery dataset.

As said in the previous section, datasets can help to alleviate engineering efforts to write rules. One can instead try to make the system learn those rules (or other representations) so that the system generalizes well on test data. Another advantage of having datasets is that they are natural benchmarks for comparing different systems. We can indeed compare two systems learning from the same training data on their performance on the test data to see which system learns better on the dataset. As the comparison may generalize to other datasets, it can indicate promising approaches.

As we are interested in executable semantic parsing, a dataset must contain two things here. First, (NL,LF) pairs based on which the systems can perform learning. Second, a knowledge base (KB) so that one can execute the LF on the KB. We will review two such datasets in this section.

Geoquery [Zelle and Mooney, 1996] is a dataset about the US geography. In its original version, the dataset contains 250 (NL, LF) pairs. Later on [Tang and Mooney, 2001] extended the dataset to include 880 (NL, LF) pairs with 600 training pairs and 240 test pairs. We refer by Geoquery to this larger extension which is a popular benchmark for many executable semantic parsing systems whose performances can be directly compared [Tang and Mooney, 2001; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Zettlemoyer and Collins, 2007]. The KB accompanying this dataset contains about 800 Prolog facts providing basic information about the U.S states, including: population, area, capital city, neighboring states, major cities etc.

Fig. 1.8 shows some examples (NL, LF) pairs from this dataset. The LFs are written in Prolog programming languages where strings starting with capital letters (S,C,P in the two examples) are logical variables. Certain semantic parsing systems [Kate and Mooney, 2006; Wong and Mooney, 2006] cannot deal with LFs with logical variables, so to circumvent the problems, some authors propose to rewrite those LFs into a variable free formalism called FunQL [Kate *et al.*, 2005b] and report the performance of their semantic parsers based on those LFs.

```
Show me flights from Boston to New York.
Frame: FLIGHT
Slots: FROMLOC. CITY = Boston
       TOLOC.CITY = NewYork
```

Figure 1.9: Examples of (NL,LF) in the ATIS dataset.

ATIS [Dahl *et al.*, 1994] is a dataset which contains air travel information such as flight departure city, destination city, departure time, flying time, etc. The initial ATIS dataset [Hirschman *et al.*, 1993] contains flight information about 11 cities, 9 airports and 765 flights. Later on, the database was largely expanded to contain flight information about 46 cities, 52 airports and 23457 flights; new annotations were acquired accompanying the dataset [Dahl *et al.*, 1994].

Inside the ATIS dataset, the logical forms (LFs) are rather 'flat' and do not have much structure. In consequence, while some researchers try to predict the LFs [Zettlemoyer and Collins, 2007], many other researchers in the community [He and Young, 2005; Raymond and Riccardi, 2007; Mesnil *et al.*, 2015] learn to map NLs into LFs that are written in frames that have slots to be filled and propose to learn the mapping as a sequence tagging problem. See Fig. 1.9 for an example of (NL, LF) inside the dataset where LFs are written in frames. Inside each frame, the *Frame* indicates the required information (i.e. the flights in the example) and *Slots* list all the constraints concerning the required information.

For studying executable semantic parsing systems, the most popular benchmark is the ATIS dataset of Class A which contains only the sentences that can be understood (can be translated into LFs) without looking at the context. In the following, we will refer by ATIS dataset to this subset which contains 4978 (NL, LF) pairs in its training set. At least two test sets are available called Nov93 and DEC94 containing 448 and 445 sentences respectively. The authors choose to report their system performance on one test set [Zettlemoyer and Collins, 2007] or both test sets [Papineni *et al.*, 1997; He and Young, 2005; Raymond and Riccardi, 2007].

Contrary to the evaluations in Geoquery, researchers studying the ATIS dataset compare their systems not by evaluating the precision and recall based on LFs (or frames in ATIS) but on the precision and the recall for each slot in the test data. The evaluation based on slots is possible and justified inside the ATIS dataset but obviously cannot be used for evaluating more complex LFs which cannot be decomposed into slots.

Robocup [Kuhlmann *et al.*, 2004] is a dataset about robotic soccer where commands to soccer robots from team coach written in natural language are paired with

NL: *"If the ball is in our goal area then our player* 1 *should intercept it."*

CLang:
```
((bpos (goal-area our))
    (do our {1} intercept))
```

Figure 1.10: Examples of (NL,LF) in the Robocup dataset.

their formal language representation written in Clang [Chen *et al.*, 2003]. Fig. 1.10 shows an example of a natural language command (noted with **NL**) paired with its corresponding Clang LF.

The Robocup dataset is a collection of total 300 such pairs. As the dataset is rather small, authors using the dataset [Kate *et al.*, 2005a; Kate and Mooney, 2006; Wong and Mooney, 2006; Wong and Mooney, 2007] report their system performance on average accuracy during cross validations. This dataset will not be used as benchmark inside the thesis. Nevertheless, we introduce it here as several examples illustrating semantic parsing systems are taken from the dataset.

1.2.2 Semantic Parsing systems with linguistic hypothesis about NL

In this subsection, we review SCFG (Synchronous CFG) [Kate *et al.*, 2005a; Kate and Mooney, 2006; Wong and Mooney, 2006; Wong and Mooney, 2007] and CCG based approaches [Zettlemoyer and Collins, 2007; Zettlemoyer and Collins, 2005] to semantic parsing. Like approaches described in subsection 1.1.3, these works also suppose that the input utterance (NL) can be parsed by a grammar. However, one difference is that grammars used here seem to be more adapted to semantic parsing tasks as the produced LFs directly take the same form as the target LFs either to be evaluated or executed.[13]

For example, [Wong and Mooney, 2006] suppose that a SCFG (synchronous CFG) generates both NLs and LFs; the same authors later extend SCFG to λ-SCFG [Wong and Mooney, 2007] so that LFs with binding variables can be handled; [Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007] suppose that NLs can be parsed by a certain CCG.

Another important difference is that instead of requiring the parse trees to be annotated like [Miller *et al.*, 1996; Schwartz *et al.*, 1996], the parse trees here are regarded as hidden; having hidden parse trees makes the learning problem harder

[13] This is in contrast to the approach discussed in 1.1.3 where the produced semantics has to be further analyzed.

$$\begin{aligned}
\textsc{Rule} &\rightarrow \langle \textit{if}\,\textsc{Condition}_{\boxed{1}}\,,\ \textsc{Directive}_{\boxed{2}}\,.\ ,\\
&\quad (\textsc{Condition}_{\boxed{1}}\ \textsc{Directive}_{\boxed{2}})\,\rangle\\
\textsc{Condition} &\rightarrow \langle \textsc{Team}_{\boxed{1}}\ \textit{player}\ \textsc{Unum}_{\boxed{2}}\ \textit{has the ball}\ ,\\
&\quad (\texttt{bowner}\ \textsc{Team}_{\boxed{1}}\ \{\textsc{Unum}_{\boxed{2}}\})\,\rangle\\
\textsc{Team} &\rightarrow \langle \textit{our}\ ,\ \texttt{our}\rangle\\
\textsc{Unum} &\rightarrow \langle \textit{4}\ ,\ \texttt{4}\rangle
\end{aligned}$$

Figure 1.11: An SCFG example which can generate synchronously the NL part 'if our player 4 has the ball' and the LF part *bowner our 4*.

but allows to significantly reduce annotation efforts. As a consequence, these methods can be directly applied to datasets such as Geoquery and ATIS without further annotations.

1.2.2.1 SCFG based approaches

[Kate *et al.*, 2005a] propose to learn an SCFG to map NLs into LFs. Analogous to an ordinary CFG, each SCFG rule consists of a single non-terminal on the left-hand side (LHS). The right-hand side (RHS) of an SCFG rule is a pair of strings (instead of a single string in CFG) (α, β), where the non-terminals in β are a permutation of the non-terminals in α. In our context, α is a string for the NL part and β for the LF part. Fig. 1.11 shows an SCFG example.

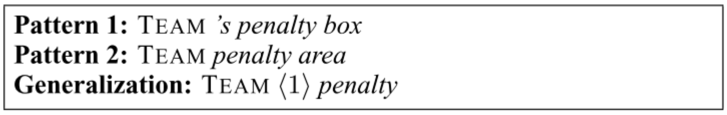
Consider the sentence (3a) from the RoboCup dataset [Chen *et al.*, 2003] whose corresponding logical form (LF) is (3b).

(3) a. If our player 4 has the ball, then our player 6 should stay in the left side of our half.
    b. *((bowner our {4})(do our {6} (pos (left (half our))))).*

The SCFG shown in Fig. 1.11 generates both the NL and the LF (if our player 4 has the ball, *bowner our 4*) of the conditional part of this example.

In the following, it is assumed that the LFs can always be parsed by a certain CFG which is known *a priori*. As this condition holds true for almost all computer languages, it should hold true also for the LFs as the LFs need to be executed thus are actually some computer languages. Because this CFG is known *a priori*, 'half' of the SCFG is known *a priori*. To illustrate, suppose that the CFG known for LFs include the LF part of SCFG shown in Fig. 1.11;[14] it is as if we have an SCFG grammar including rules such as

[14] We will refer to this CFG simply as CFG known *a priori* in the follwing of this subsubsection.



Pattern 1: TEAM *'s penalty box*
Pattern 2: TEAM *penalty area*
Generalization: TEAM ⟨1⟩ *penalty*

Figure 1.12: Output rule generalizing on two NL strings with the same LF production (penalty-area $TEAM$).

(4) a. $TEAM \rightarrow (?, our)$
b. $CONDITION \rightarrow (?, bowner\ TEAM\ \{UNUM\})$

where ? are the placeholders for the NL part. and one needs only to search for the NL part in each SCFG rule.

[Kate *et al.*, 2005a] propose to learn the SCFG rules (i.e fill in the NL part from 'half specified' rules (4a) (4b) etc.) by using a bottom-up rule induction process which infers general rules from maximally specific ones. At initialization, for each training pair (NL, LF), rules are created where the NL part contains the whole sentence. For example, given the (NL, LF) pair (3a, 3b) and the CFG known *a priori* for LFs, one would create the SCFG rule shown in (5) by filling the NL part of the rule (4a) with the whole sentence:

(5) $TEAM \rightarrow (\text{If our player 4 ... our half}, our)$.

This rule however is too specific to be applied for test sentences, so one needs to generalize those rules. Ideally, we want to induce from the above, very specific, SCFG rule, a rule like $TEAM \rightarrow (\text{our}, our)$ which is much more general. [Kate *et al.*, 2005a] propose a procedure to generalize rules based on pair of rules with the same LF production. At each iteration, they sample two rules with the same LF production out of all the rules constructed so far;[15] when receiving the two rules, they first compute the longest common subsequence (with gaps) of the two NL strings containing also non terminals and regard the subsequence as generalization candidate. An example of this common subsequence computation is given in Fig. 1.12 where two rules Pattern 1 and Pattern 2 (we only show the NL part of the rule indicated by 'Pattern' as the LF production is the same) are merged into one more general rule candidate. If the rule is appropriate[16], they add the rule to be a rule in the final SCFG.

The learned SCFG may have ambiguity, the authors propose to resolve these ambiguities by adding more context to the NL parts of each involved SCFG rule,

[15] The rules initially are like (3a) before generalization.

[16] Authors define a score of appropriateness for each rule which is the product of its accuracy and coverage.

making the learned SCFG deterministic.

The above SCFG learning approach is tested on Geoquery showing an accuracy of 0.88 and a recall of 0.53.[17]

There are two problems with this approach that learns a deterministic SCFG. First, the SCFG is not very robust. For example, the learned rules can not deal with misspelled words never seen in training (the low recall of the system somehow indicates the non robustness of the system). Secondly, the learned SCFG is made deterministic at the price of making rules rather specific. One may want to have more general rules and learn a stochastic SCFG. If more than one derivation tree matches the sentence, one can learn a ranker to discriminate these derivation trees. We will now see some approaches which are also based on learning an SCFG for semantic parsing, but propose to address the above issues.

[Kate and Mooney, 2006] propose to deal with the robustness problems by using kernel-SVMs.

Recall that a CFG for LFs is known *a priori*; for each CFG rule producing an LF string (i.e $TEAM \rightarrow our$), they propose to learn a string kernel SVM [Lodhi *et al.*, 2002] predicting if a certain rule should be used for a given sentence. More precisely, given a CFG rule noted π; at first iteration, all the sentences where π is used are collected as positive examples and all the other sentences are used as negative examples to train the kernel SVM. For example, given (NL, LF) pair shown in (3a,3b), the authors use the kernel SVM to learn that the rule ($TEAM \rightarrow our$) should be predicted as the sentence is a positive example for this rule while the rule ($REGION \rightarrow$ penalty-area $TEAM$) should not be predicted (sentence being a negative example for this rule).

Once these classifiers are learned, they are used to parse the NL string where an additional constraint that substrings covered by the children of a node are not allowed to overlap is taken into account. For a given sentence, several parses are often possible, so a probability score for each rule application covering a certain NL substring is needed. Denote the NL substring L, authors propose to use the trained SVM to determine the probability score. For example, for the rule ($TRAVERSE \rightarrow traverse$) covering four consecutive words 'which river runs through' in Fig. 1.13, authors use the trained SVM to predict its probability score based on this four word input. The probability is estimated based on the SVM margin with the input L.

With those probability scores, a given sentence is parsed producing the derivation

[17] The results shown here are a tree based version of the presented method where the NL part also includes syntactic information (e.g. several terminals grouped forming an NP nonterminal).

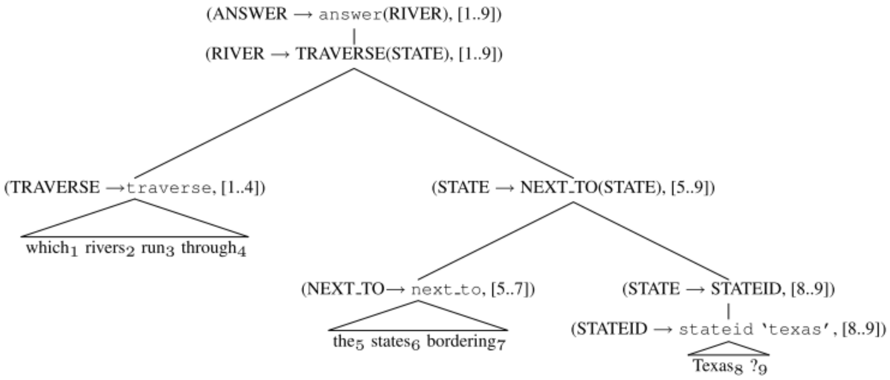

(ANSWER → answer(RIVER), [1..9])
(RIVER → TRAVERSE(STATE), [1..9])
(TRAVERSE →traverse, [1..4])
which1 rivers2 run3 through4
(STATE → NEXT_TO(STATE), [5..9])
(NEXT_TO→ next_to, [5..7])
the5 states6 bordering7
(STATE → STATEID, [8..9])
(STATEID → stateid 'texas', [8..9])
Texas8 ?9

Figure 1.13: A derivation tree that correctly parses the NL.

with the maximum score.[18] Fig. 1.13 shows an example where these classifiers help to find the correct derivation tree. When the produced LF is found to be correct, each applied rule with its covered substring is added as positive training examples for the SVMs. Sometimes, the learned classifiers can make mistakes and return the wrong derivation tree; in those cases, some wrongly predicted rules together with their associated substrings are added as negative examples. The kernel SVMs will then be retrained with the 'enriched' training set and the procedure iterates several times.

The authors test their system on Geoquery and achieve a precision of 0.93 and a recall of 0.72. The proposed method is shown to be particularly robust to noise such as spelling errors.

Inspired by phrase based machine translation [Chiang, 2005], [Wong and Mooney, 2006] propose to use word alignment techniques to learn overgenerating SCFG rules and then rank possible derivations.

First, the authors propose to use the target side non-ambiguous CFG to parse each LF and produce the corresponding derivation sequence (DS) which is the sequence of rules used to produce the LF. A word alignment model then is learned between the NL and the DS. The authors argue that learning such an alignment model results more useful information compared to trying to align between NLs and LFs as not every token in LFs (e.g. the parentheses in the LF) carries specific meanings.[19] An example of such alignments is shown in Fig. 1.14.

[18] Note that to calculate the exact maximum, one has to consider every derivation as the classifier assigns a nonzero score to all the rules, so approximate beam search based algorithms are derived to calculate the derivation with the maximum score.

[19] Authors also argue that learning alignment between LFs and DSs is better because tokens in

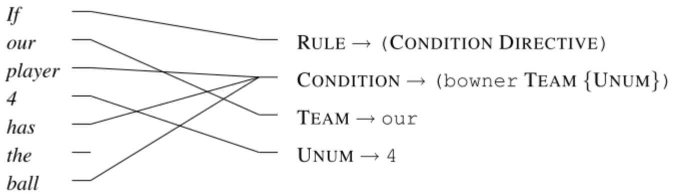

If
our
player
4
has
the
ball
RULE → (CONDITION DIRECTIVE)
CONDITION → (bowner TEAM {UNUM})
TEAM → our
UNUM → 4

Figure 1.14: An example of alignment between words and CFG rules in [Wong and Mooney, 2006].

Then, with the help of these alignments an SCFG is constructed in a bottom-up manner. Similar to [Kate *et al.*, 2005a], the system tries to construct an SCFG rule by adding an NL part for each LF production rule resulting rules like $TEAM \rightarrow$ our, our where first 'our' is an NL expression. Concretely, the system starts with LF production rules whose RHS is all terminals (e.g. $TEAM \rightarrow our$ and $UNUM \rightarrow 4$). For each of these productions, a rule $X \rightarrow (\alpha, \beta)$ is constructed[20] where α consists of the words to which the production rule is linked where the 'linking' is found by learned alignments.

For example, consider the alignments shown in Fig. 1.14. The rule expanding the nonterminal $UNUM$ is linked to '4' while the rule expanding the nonterminal $CONDITION$ is linked to 'player', 'has' and 'ball'. According to the rule construction procedure just sketched, $UNUM \rightarrow 4, 4$ can be extracted as '4' is the only word linked to the rule expanding $UNUM$.

Then the system considers the production rules whose RHS contain also non-terminals. In this case, a constructed pattern (the NL part of an SCFG rule) consists of words to which the production rule as well as the non-terminals inside the rule are linked. For example, one constructed rule would be: [21]

$$CONDITION \rightarrow < TEAM_1\ player\ UNUM_2\ has\ (1)\ ball,\ (bowner\ TEAM_1\ UNUM_2) >$$

as all the NL words appearing in this rule link to either the rule expanding *CONDITION* or to the nonterminals on its RHS shown in Fig. 1.14.

Once the SCFG is constructed, the next task is to learn a ranker to rank all the derivation sequences that can generate the NL. The authors use a log-linear model:

$$P_\lambda(d|e) = \frac{1}{Z_\lambda} exp \sum_i \lambda_i f_i(d, e),$$

LFs may exhibit polysemy; we don't think this point is very relevant; in any case, polysemy is unavoidable for the NLs.

[20] Note that similar to [Kate *et al.*, 2005a] X and β is given by the LF production rule.

[21] The (1) denotes a word gap of size 1, due to the unaligned word *the*.

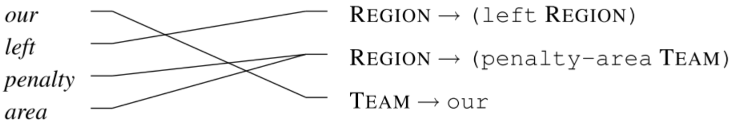


Figure 1.15: An example of alignment between words and CFG rules in [Wong and Mooney, 2006] where no rule can be extracted for $REGION \rightarrow$ penalty-area $TEAM$.

where e is the NL input, d is the derivation sequence (DS), f_i are manually defined features involving d, e or both and λ are parameters to be learnt.

The whole system with the learned SCFG and the ranker achieves a precision of 0.87 and a recall of 0.73 on Geoquery dataset. Note that the system adopts the SCFG formalism thus cannot predict LFs with variables, so instead of learning to predict the Prolog queries in Geoquery, the system learns to predict the FunQL queries which are semantically equivalent LFs without variables.

According to the SCFG construction procedure described above, the model [Wong and Mooney, 2006] is only effective when words linked to a specific LF production rule stay close to each other in the sentence. As a failure example, consider the sentence 'our left penalty area' and its corresponding LF *(left (penalty-area our))* with learned alignment shown in Fig. 1.15. In this case, no SCFG rule can be constructed to generate the LF *(penalty-area TEAM)* as we impose that all the words from the NL part in a rule should link to the production rule or its RHS nonterminal while the word 'left' is not linked to this rule; the fact that 'our' and 'penalty area' is separated by 'left' making the rule production impossible.

Discussion We can compare the work [Wong and Mooney, 2006] with previous work [Macherey *et al.*, 2001] which has also proposed to use alignment models to learn a semantic parsing system. One major difference and a contribution of [Wong and Mooney, 2006] is that authors propose to use CFG over the LFs as prior knowledge; without using this CFG as prior knowledge, the LFs that a system produces can be syntactically ill-formed. Another difference is that the model of [Wong and Mooney, 2006] uses SCFG from hierarchical phrase based machine translation [Chiang, 2005] while the model in [Macherey *et al.*, 2001] is not based on hierarchical machine translation but simpler alignment models.

Techniques similar to [Wong and Mooney, 2006] can be used to handle LFs with variables as shown in [Wong and Mooney, 2007]. One needs to extend SCFG to λ-SCFG in which each rule has the form:

$$A \rightarrow < \alpha, \lambda x_1 \dots \lambda x_k . \beta >$$

Where α is an NL sentence and β is a string of terminals, non-terminals, and variables. The variable-binding operator λ binds occurrences of the logical variables $x_1, \ldots, x_k$ in β. Learning a λ-SCFG can be performed in a bottom up way, similar to the learning of a SCFG; the ranker can also be constructed similarly.

Being able to handle variables can ease the λ-SCFG learning. If the LF (a, b) is equivalent to (b, a), one can choose to represent the LF such that the LF is maximally isomorphic[22] to the input utterance (NL) thus ease the learning of λ-SCFG. As a result, when authors [Wong and Mooney, 2007] test their system on the Geoquery dataset with LFs written in Prolog (and rearrange the LFs to be maximally isomorphic to the NLs), they found a large improvement over the system [Wong and Mooney, 2006] and achieve a precision of 0.92 and a recall of 0.87.

1.2.2.2 CCG based approaches

[Zettlemoyer and Collins, 2005] propose to use PCCG (probabilistic CCG), an extension of CCG [Steedman, 1996][23] to learn an executable semantic parser. Like [Wong and Mooney, 2006], the system both parses sentences using CCG rules also called lexicons and ranks them using a log-linear model; the difference is that in [Zettlemoyer and Collins, 2005] the lexicons and the feature parameters are learnt jointly.

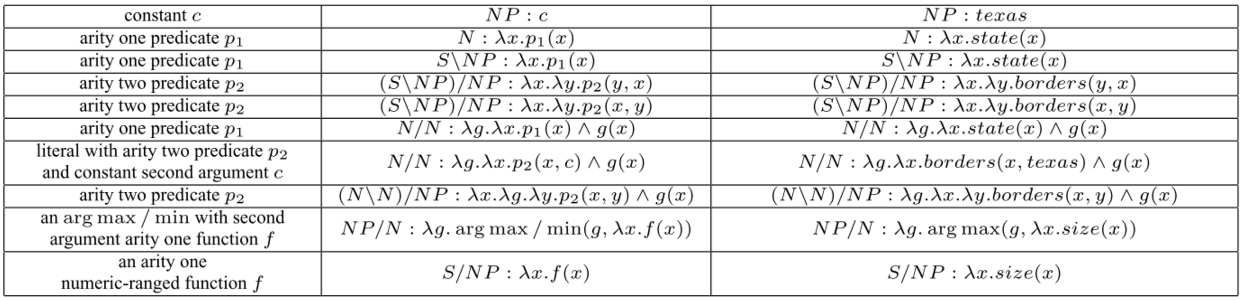
A CCG parser parses a sentence with CCG rules. For example, consider the following subset of CCG rules. Each rule associates a lexical item with a syntactic category and a semantics written in lambda calculus. For instance, the first rule associates the lexical item Utah with the syntactic category NP and the semantics *utah*.

- Utah := *NP*:*utah*
- Idaho := *NP*:*idaho*
- borders := *(S\NP)/NP*: $\lambda x.\lambda y.borders(y, x)$

With these 3 rules, one can parse the sentence “Utah borders Idaho” (with forward and backward applications) and produce the correct LF $borders(utah, idaho)$. The central question is then how to successfully learn those rules (also called lexicons) together with their feature parameters based uniquely on (NL, LF) pairs when parsing steps are not available in the considered datasets.

[22] Isomorphic in the sense that the structure of the LF being similar to that of the NL.

[23] Inside this thesis, we briefly discussed CCG in subsection 1.1.2.



constant c	$NP : c$	$NP : texas$
arity one predicate p_1	$N : \lambda x.p_1(x)$	$N : \lambda x.state(x)$
arity one predicate p_1	$S\backslash NP : \lambda x.p_1(x)$	$S\backslash NP : \lambda x.state(x)$
arity two predicate p_2	$(S\backslash NP)/NP : \lambda x.\lambda y.p_2(y, x)$	$(S\backslash NP)/NP : \lambda x.\lambda y.borders(y, x)$
arity two predicate p_2	$(S\backslash NP)/NP : \lambda x.\lambda y.p_2(x, y)$	$(S\backslash NP)/NP : \lambda x.\lambda y.borders(x, y)$
arity one predicate p_1	$N/N : \lambda g.\lambda x.p_1(x) \wedge g(x)$	$N/N : \lambda g.\lambda x.state(x) \wedge g(x)$
literal with arity two predicate p_2 and constant second argument c	$N/N : \lambda g.\lambda x.p_2(x, c) \wedge g(x)$	$N/N : \lambda g.\lambda x.borders(x, texas) \wedge g(x)$
arity two predicate p_2	$(N\backslash N)/NP : \lambda x.\lambda g.\lambda y.p_2(x, y) \wedge g(x)$	$(N\backslash N)/NP : \lambda g.\lambda x.\lambda y.borders(x, y) \wedge g(x)$
an arg max / min with second argument arity one function f	$NP/N : \lambda g.\arg\max / \min(g, \lambda x.f(x))$	$NP/N : \lambda g.\arg\max(g, \lambda x.size(x))$
an arity one numeric-ranged function f	$S/NP : \lambda x.f(x)$	$S/NP : \lambda x.size(x)$

Figure 1.16: All the CCG rules used in [Zettlemoyer and Collins, 2005].

[Zettlemoyer and Collins, 2005] propose an iterative procedure to learn the lexicon together with their feature parameters. First, they limit the forms of rules that the system will learn. In practice, all the learned rules will be in one of the forms shown in Fig. 1.16. Given an (NL, LF) pair, the system creates temporarily all the rules possibly useful to map the NL to the LF via a function called *GENLEX*, mathematically:

$$GENLEX(NL, LF) = \{x := y | x \in W(NL), y \in C(LF)\}$$

In the above formula, $W(NL)$ is the set of all subsequences of words in the NL and $C(LF)$ returns the rules involved in the LF that are in one of the forms shown in Fig. 1.16. For example, consider the sentence “Utah borders Idaho” paired with the LF $borders(utah, idaho)$. In this example, C(LF) will contain three elements: $utah$, $idaho$ and $\lambda x.\lambda y.borders(y, x)$; W(NL) will contain all the substrings such as ‘utah’, ‘borders’, ‘utah borders’, etc. Thus, rules such as Utah := NP:$utah$ and borders := NP:$utah$ will all be considered as candidates initially. Note that the GENLEX function over-generates lexicons to map an NL into the corresponding LF, which guarantees that the system always finds a parse leading to the correct form.

$$P(LF, DS|NL, \theta) = \frac{e^{f(LF,DS,NL)\cdot\theta}}{\sum_{LF,DS} e^{f(LF,DS,NL)\cdot\theta}}$$

To distinguish between several possible parses, a PCCG is actually used. As shown in the equation above, the PCCG defines a probability distribution over (LF,DS) (DS is a derivation sequence) using a log-linear model; in the equation, $f(LF, DS, NL)$ are manually defined features for a rule capturing interactions between the associated syntax/semantics. In the model [Zettlemoyer and Collins, 2005], only weights for each lexicon are used as features.

Note that the objective is to maximize the probability of the correct LF which can come from different DS's with $P(LF|NL,\theta) = \sum_{DS \rightsquigarrow LF} P(LF, DS|NL, \theta)$;[24] given a set of rules, parameters in the log-linear model (i.e the weight of each lexicon) can be updated using gradient descent so that the likelihood of the correct LF $P(LF)$ is maximized.[25]

To output a compact set of rules[26] [Zettlemoyer and Collins, 2005], [Zettlemoyer and Collins, 2005] propose to iterate between lexicon generation steps (using *GENLEX*) and parameter estimation steps. More precisely, after the initialization step that set weights for generated lexicons and initial lexicons Λ_0, the system iterates between the following two steps at iteration t where a lexicon set Λ_t is initialized to be empty before the iteration:

- For each sentence, the system calls the *GENLEX* procedure overgenerating lexicons and parses the sentence with their currently associated weights; then, only the rules that are used for the parse with the highest probability score are added to Λ_t. The system parses the whole corpus this way and then set $\Lambda_t = \Lambda_t \cup \Lambda_0$.
- The feature parameters involving Λ_t are optimized by using gradient descent.

After a fixed number of iterations, the learning procedure stops and output the lexicons used at the last iteration Λ_t together with their feature weights; this more compact lexicon is used at test time. The system was tested on Geoquery and achieves a precision of 0.96 and a recall of 0.79.

Discussion Compared with the performance of [Wong and Mooney, 2007] on the Geoquery dataset, the system of [Zettlemoyer and Collins, 2005] has a higher precision and lower recall. We think that the lower recall is due to the rigidity of the learned CCG grammar as it cannot parse ungrammatical sentences unseen at training time. For example, consider a CCG grammar with two rules in the ATIS domain:

- flights := N:$\lambda x.flight(x)$
- one way := *(N/N)*:$\lambda f.\lambda x.f(x) \wedge one\_way(x)$

[24] We use $DS \rightsquigarrow LF$ denote all the DS that produce the concerned LF.

[25] Even though the number of DS's is huge, the gradients can be calculated exactly in this case by using dynamic programming. We refer interested readers to the original paper [Zettlemoyer and Collins, 2005] on this aspect. This is notably contrary to some models we discuss in 1.3.1.1 where gradients can no more be calculated exactly (there is no efficient way).

[26] In particular, we want the system to avoid outputting rules such as Idaho := *NP*:*utah*.

$$\underbrace{list}_{null}\ \underbrace{twa}_{airline\_code}\ \underbrace{flights}_{null}\underbrace{from}_{null}\underbrace{washington}_{fromloc.city}\underbrace{to}_{null}\underbrace{philadelphia}_{toloc.city}$$

Figure 1.17: ATIS annotated with concept sequences.

The grammar can successfully parse the sentence “one way flights” and give the correct corresponding parse $\lambda x.flight(x) \wedge one\_way(x)$. However, the system will fail for the sentence “’flights one way” (while one can argue that the sentence should be understood by the system) thus lowering the recall of the system.

[Zettlemoyer and Collins, 2007] propose a remedy to the above problem by allowing more forms of rules in the system so that a larger number of sentences can be parsed by the considered CCG. The additional rules involve application and composition rules, type-raising rules and crossed composition rules. We discuss here the added functional application rules:

- A\B:f B:g $\Rightarrow$ A:$f(g)$
- B:g A/B:f $\Rightarrow$ A:$f(g)$

These application rules are variants of the original application rules, where the slash direction on the principal categories (A\B or A/B) is reversed. With the second rule, one can correctly parse the sentence “’flights one way” with the result being the same as “one way flights”.

Having additional rules lead to significantly more parses for any sentence x. However, the authors found that the log-linear ranker[27] is still able to learn well to choose a correct parse among all the candidate parses. Authors test their system on the ATIS NOV93 dataset and achieves a precision of 0.95 with a recall of 0.97.

Remark As the semantics in the ATIS dataset is written in frames thus pretty ‘flat’, people have proposed to annotate each sentence in the dataset with a sequence of associated concepts as shown in Fig. 1.17. Under these conditions, one can apply sequence tagging techniques using models such as Conditional Random Fields (CRFs) [Lafferty *et al.*, 2001] and Recurrent Neural Networks (RNNs) [Rumelhart *et al.*, 1988]. Note however that annotations will be very difficult if not impossible when the logical forms (LFs) are not that ‘flat’ and have more structures. For this reason, approaches based on sequence tagging that we describe here cannot be applied for general semantic parsing tasks and we will not discuss them in detail inside the thesis.

[27] Compared to [Zettlemoyer and Collins, 2005], the ranker incorporates a richer set of features.

CRFs are discriminative undirected graphical models which find their applications in sequence tagging problems such as POS tagging and entity recognition. At each prediction step, the model makes a prediction based on manually defined features that take into account the current word, the context of the current word and the previous prediction (tag)[28]; the model assigns a global score to each target sequence and learns its parameters to rank the correct target sequence highest amongst all the sequences, using a log-linear model. Mathematically, let y be the target sequence and x the input sequence; the model (first-order CRF) is defined by a set of features f_k with their associated weights λ_k; $Z(x)$ is the normalization term summing over all the sequences y:

$$p(y|x) = \frac{exp(\sum_t \sum_{k=1}^{K} \lambda_k f(y_{t-1}, y_t, x))}{Z(x)}$$

[Raymond and Riccardi, 2007] apply CRFs on the ATIS dataset where the features are mainly defined to link the words with corresponding concepts. For example, one can define features such as if the word 'washington', it links to either the concept fromloc.city or the concept toloc.city; the ambiguity can be further resolved by applying more specific features, e.g. if the word 'washington' is preceded by 'from' then the tag is 'fromloc.city'.

The authors test their approach on the ATIS dataset (Nov93+Dec94) and achieve a F1 score of 0.96.

[Mesnil *et al.*, 2015] propose to use Recurrent Neural Networks (RNNs) for the semantic parsing problems seen as sequence tagging problems. Compared to CRF models, the main advantage of using such deep learning models is to avoid feature engineering as deep learning models integrate both feature design and classification into the learning procedure. Some discussions on RNNs can be found in section 3.2, 3.3 inside the thesis.

The considered neural network architecture runs an RNN through the input sequence x, calculates a hidden vector at each step (word) of x based on the word context and already predicted tags; then in the simplest case, the model learns to predict the tag at the current step based on this hidden vector through cross-entropy minimization.

More precisely, the network first maps each 1-hot encoded word to a continuous

[28] In [Lafferty *et al.*, 2001], CRF is introduced so that the current tag depends only on the previous tag. However, the model can still be trained reasonably efficiently with the current tag depending on the n previous tags when n is small [Cuong *et al.*, 2014].

	Precision	Recall	F1
[Tang and Mooney, 2001]	0.90	0.79	0.84
[Zettlemoyer and Collins, 2005]	0.96	0.79	0.87
[Wong and Mooney, 2007]	0.92	0.87	0.89
[Zettlemoyer and Collins, 2007]	0.92	0.86	0.89

Table 1.1: Results on the Geoquery dataset for systems learning to predict a Prolog query.

word embedding which is a real-valued vector of certain dimensions (typically several hundreds which is much less than the total number of words). Then an RNN is run through the word embeddings producing a hidden vector at each time step. Many variants of RNNs exist, the authors propose to use the following formulation which is a hybrid of Elman type [Elman, 1990] (i.e. the $h(t-1)$ term which is the previous hidden vector in the equation below) and Jordan type [Jordan, 1986] (i.e. the C_d term which is a context window of size d centered on the t-th word in the input sequence x and the $y(t-1)$ term which is the previous predicted tag, in the equation below):

$$h(t) = f(UC_d + U'P(y(t-1)) + U * h(t-1))$$

The class label at each time t is then calculated based on $h(t)$ using softmax.

As said, one can train this RNN step by step (trying to make the predicted tag at each step correct). However, this standard RNN training (also called teacher forcing) can create the label bias problem [Bottou, 1991],[29] a problem that CRF models can solve. So the authors propose to build a CRF model on top of the RNN model where hidden states are regarded as CRF features.

The authors test their model (word embeddings+RNN+CRF) and observe improvements over CRF baselines on the ATIS dataset achieving an F1 score of 0.96.

We want to finally mention that the literature seeing ATIS as a sequence tagging problem is abundant. Just to mention a few of them, [Kuhn and De Mori, 1995] propose to use classification trees to classify slots and [He and Young, 2005] propose to use hidden a vector space (HVS) model.

Remark Tables 1.1 and 1.2 summarize the performance of the different systems on the Geoquery dataset discussed in this section and Table 1.3 summarizes the

[29] The label bias problem appears due to the asymmetry we create between training and test: at training time, the model always has access to the correct previous tags, which is not satisfied at test time.

	Precision	Recall	F1
[Wong and Mooney, 2006]	0.87	0.73	0.79
[Kate and Mooney, 2006]	0.93	0.72	0.81

Table 1.2: Results on the Geoquery dataset for systems learn to predict a FunQL query.

	F1
[He and Young, 2005]	0.90
[Zettlemoyer and Collins, 2007]	0.96
[Raymond and Riccardi, 2007]	0.96
[Mesnil *et al.*, 2015]	0.96

Table 1.3: Results on ATIS dataset, note that the results of [Zettlemoyer and Collins, 2007] is tested only on the Nov93 dataset while other systems are tested both on the Nov93 and the Dec94 dataset.

performance of systems on ATIS. We have two remarks based on these tables.

- Systems based on CCG [Zettlemoyer and Collins, 2007] can perform well on both datasets. From the tables, we can see that they are amongst the best performing systems.[30] This suggests that when we can safely suppose that most of the sentences can be parsed by a certain grammar, the use of this linguistic prior knowledge (CCG in this case) can efficiently reduce the search space thus largely ease the learning procedure [Zettlemoyer and Collins, 2005]. More than that, it seems that this prior knowledge is so useful that even a relaxed form can largely benefit the task [Zettlemoyer and Collins, 2007].

- All the best performing systems use some discriminative models. One particularly interesting class of discriminative models are deep learning models as they avoid heavy feature engineering for a given task. However, at this stage, we have seen the application of deep learning models (RNNs, in particular) only to semantic parsing seen as a sequence tagging task [Mesnil *et al.*, 2015] while as we have said, this form of semantic parsing is very limited compared to the task we focus on where one needs to generate a structured LF (which is not necessarily a linear chain and which can be of any length).

[30] One should note that the CCG system is only tested on the NOV93 dataset on ATIS while most of other systems are tested on both the NOV93 and the DEC94 datasets.

1.3 New Challenges

In the previous section, we have seen some successful semantic parsing systems learned based on a training corpus that consists of (NL, LF) pairs. Despite of the success, there are two limits of those systems that more recent systems try to address:

- One may note that all the previous systems assume having access to annotated LFs, which requires a lot of annotation efforts: annotating LFs often require expertise for the language in which LFs are expressed. We will review in this section different proposals raised by researchers in recent years to train semantic parsers while reducing the annotation efforts. One important direction is to train the system based on (natural language question, answers) pairs [Liang *et al.*, 2011; Berant *et al.*, 2013; Berant and Liang, 2014; Pasupat and Liang, 2015] that we will note as (NL, A) in the following; this reduces the annotation efforts notably because annotating an answer does not require as much expertise as annotating a LF. Another idea of annotating LFs with less efforts is to turn the LF annotation problem to a simpler one such as creating paraphrases [Wang *et al.*, 2015]. Finally, noting that text information is abundant and easy to access nowadays, we will review proposals to leverage these texts to learn semantic parsers [Reddy *et al.*, 2014].

- Previous semantic parsing systems require also much engineering efforts. For example, to build the semantic parsing system [Zettlemoyer and Collins, 2005] based on PCCG, the authors need to specify the set of necessary CCG rules and also the features for the log-linear ranker. Also, although CCG-based semantic parser induction is language- and domain- independent as shown in [Kwiatkowski *et al.*, 2011], sometimes minor changes in CCG rule specification are necessary to well adapt to a particular domain.[31] Thus, researchers have proposed machine learning models that learn semantic parsers with less engineering efforts [Bordes *et al.*, 2014a; Yih *et al.*, 2015] (e.g. by using end to end deep learning models). We will review those works as well in this section.

1.3.1 Reduce annotation efforts

1.3.1.1 Learning with QA pairs

In recent years, learning a semantic parser based on (NL, A) pairs has boosted the research for semantic parsing and a review of learning with QA pairs can be found

[31] For example, [Zettlemoyer and Collins, 2007] has to extend the CCG rules that they use previously [Zettlemoyer and Collins, 2005] when building a semantic parser for another domain.

in [Liang, 2016]. The main problem in learning with QA pairs is that LFs are no more available thus have to be treated as hidden variables. Under this condition, suppose that we want to build a semantic parser using/extending previous approaches such as [Wong and Mooney, 2007; Zettlemoyer and Collins, 2005] that first induce a grammar that parses the natural language question while generating LF candidates, then use a ranker that learns to choose the best LF amongst the candidates,[32] learning with QA pairs implies two complications:

- For the ranker, instead of learning a log-linear ranker maximizing the likelihood of the annotated LF $P(LF|NL,\theta) = \frac{e^{f(LF,NL)\cdot\theta}}{\sum_{LF} e^{f(LF,NL)\cdot\theta}}$, one needs to learn to maximize the probability of getting the correct answer $P(A|NL,\theta)$ by marginalizing over LFs, thus giving the objective function:

  $$P(A|NL,\theta) = \sum_{LF \rightsquigarrow A} \frac{e^{f(LF,NL)\cdot\theta}}{Z}$$

  where we use $LF \rightsquigarrow A$ to denote the LFs whose executions give the answer A; $Z = \sum_{LF} e^{f(LF,NL)\cdot\theta}$ is the normalizing factor summing over all possible LFs. Note that as we sum on the numerator over all the LFs reaching the answer which is a set of LFs that don't factorize over the nodes inside a parse tree (e.g CFG parse tree, CCG parse tree etc.), there is no way to estimate the quantity $P(A|NL,\theta)$ or its gradient exactly besides first enumerating all the LFs and execute each one of them.[33] Thus, to optimize the objective function $P(A|NL,\theta)$, beam search (to search for LFs reaching the correct answer) with approximate gradients (gradients calculated based on the beams) are generally used.

- For the grammar induction, first note that all the learning approaches we describe in the previous section (e.g. [Zettlemoyer and Collins, 2005; Wong and Mooney, 2007]) need the LFs to be given in the dataset: [Zettlemoyer and Collins, 2005] first decompose the LFs into elementary forms that are known *a priori* (see Fig.1.16 for an example of the elementary forms) before trying to learn the NL expressions associated with these elementary forms; to learn an SCFG that can translate the NL into corresponding LFs, [Wong and Mooney, 2007] first parse the LFs using a CFG over LFs known *a priori* before trying

[32] As we shall see in this chapter, all the works we review in this subsection adopts this architecture using a grammar and a ranker but with important difference on the role the grammar plays.

[33] This is contrary to the model [Zettlemoyer and Collins, 2005] where the probability of the correct LF $P(LF|NL,\theta)$ and its gradient can be estimated exactly in an effecient way (by using dynamic programming).

to deduce the NL expressions that trigger the SCFG rules.

As the direct application of previous approaches [Zettlemoyer and Collins, 2005; Wong and Mooney, 2007] are not possible, one may try to devise a two step approach to build a semantic parser where the first step tries to find all the correct LFs. However, as the number of LFs can be huge even for relatively small datasets due to the compositionality, the solution is not applicable in general cases.

To our best knowledge, the first work proposing to learn a semantic parsing system with QA pairs (notably addressing the grammar induction difficulty) is [Liang *et al.*, 2011]. We will first give the general idea behind the approach before entering into more details.

As the LFs are not available in the dataset, the system we want to build has to explore the space of LFs randomly at the beginning.[34] During the random exploration, the system may find LFs that give the correct answer for some NLs in the dataset; the semantic parser can then learn a model over these (NL,LF) pairs so that at the next iteration, the system can explore the LF space focusing on promising LFs by using the model it has learned. The system iterates this way each time switching between learning a semantic parser and exploring LFs using it, forming a bootstrapping approach; the system stops when it finds enough proportion of correct answers for the NL questions in the dataset, suggesting a good performance of the semantic parser. This bootstrapping approach is examined under a theoretical point of view in [Daumé and Marcu, 2005] where the authors name it LaSO (Learning as Search Optimization).

To implement the approach [Liang *et al.*, 2011] learning a semantic parser by bootstrapping, one first needs a flexible grammar so that the system can explore randomly the LF space at first place. Fig. 1.18 shows an example of parsing a sentence with a flexible grammar. While the rules of type 'lexicon' can be generated by using some prior knowledge, the rules of type 'join' and 'intersection' are generated randomly at the beginning before being generated by the semantic parser's guidance. Fig. 1.18 also illustrates another flexibility of the grammar as the sentence is not fully analyzed by the grammar and some tokens in the sentence (e.g. 'was' and '?') are skipped.

[34] With some exceptions where prior knowledge can be used. For example, to answer the question 'States that border Utah', one may encode the prior knowledge into the semantic parser that if the NL contains 'utah' or 'Utah', the LFs have to contain *utah*; however, this form of prior knowledge is generally limited to named entities.

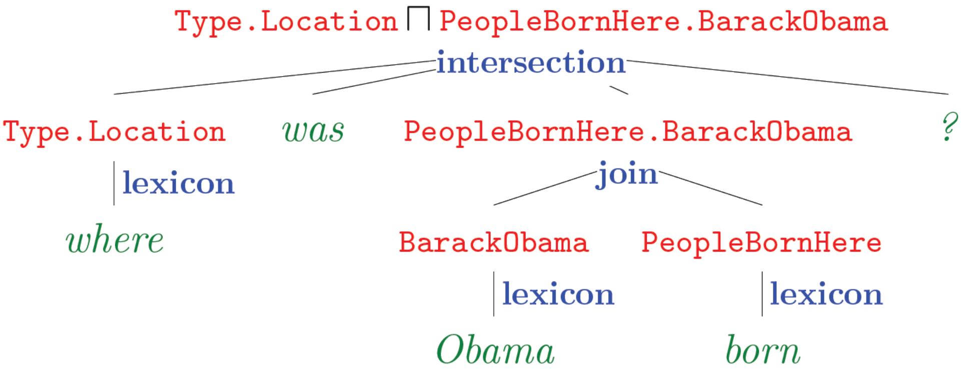


Figure 1.18: An example of a derivation tree for the sentence ‘Where was Obama born’ using a flexible grammar; the derivation steps are each labeled with the type of rule (in blue) and the logical form (in red).

One can see that the grammar we are discussing here is very different from grammars such as CCG which do not generate rules on the fly and which do not skip tokens.[35].

A rich feature set is also needed for the semantic parser to implement the bootstrapping learning approach. When a correct LF is found,[36] the feature parameters of the corresponding rule application inside the LF will be updated (very often using gradient based optimization algorithms) to make sure that the correct rule will be applied in the future on this example; when the features associated with the rule application is rich enough, we can hope that some features will generalize over NLs allowing to find more correct LFs at the next iterations.

For example, suppose that the system finds a correct LF (notably the ‘join’ and ‘intersection’ operations) for the given sentence in Fig. 1.18 through exploration. If the only feature associated with the ‘join’ rule application is its frequency in the dataset (similar to [Zettlemoyer and Collins, 2005]), the model will just try to apply ‘join’ rules more often at the next iterations during bootstrapping. However, better generalization may be achieved if the ‘join’ rule is also associated with the NL feature such as ‘PERSON born’;[37] in this case, the semantic parser will favor to apply the ‘join’ rule at the next iterations when it sees the pattern ‘PERSON born’ after parameter updates.

[35] Note that constructing a semantic parser with CCG is indeed possible, but as the CCG will not produce semantics tailored to the domain of interest, some ‘semantic matching’ module needs to be implemented [Reddy *et al.*, 2014].

[36] The ‘correct’ here means that the LF gives the correct answer under execution.

[37] Suppose that the system can identify a ‘Obama’ in the sentence as a person.

By putting together a flexible grammar and a rich feature set, [Liang *et al.*, 2011] test the bootstrapping approach to learn a semantic parser based on QA pairs.[38] On the Geoquery dataset, the semantic parser can only find 29% of the correct LFs at first by 'pure exploration', but training over these examples are enough to increase this ratio to 65% and then 95% after the next few iterations. Some patterns are discovered during bootstrapping through learning with the rich feature set; the system discovers for example that when a word 'in' is seen in the NL, the relation 'loc' (i.e. location) will be triggered with high probability.

The system overall achieves an accuracy of 91.1%, a new state of art on Geoquery while only trained on QA pairs (previous systems are trained on (NL, LF) pairs.).

Many recent works propose improvements of the above bootstrapping approach to learn a semantic parser with QA pairs.

[Berant *et al.*, 2013] propose two ways to better exploit prior knowledge to limit the LF space that the system searches. The first way consists of constructing a larger lexicon set corresponding to the rules applied at the bottom level in the derivation tree (e.g. the 'lexicon' rules in Fig. 1.18) that map NL expressions to LFs; the second way consists of using type constraints of the LFs to control generation of rules other than lexicons (e.g. the 'join' and 'intersect' rules in Fig. 1.18)).

Their system is tested on an RDF knowledge base Freebase [Bollacker *et al.*, 2008] which contains about 2.5 billion triples in the form (e_1, r, e_2); in the triple, e_1, e_2 are entities forming the subject (e_1) and the object (e_2) respectively and r is a relation forming the predicate of the triple (e.g. ('barack_obama','birthplace','honolulu')). A dataset called **Webquestions** containing QA pairs is constructed for which the questions can be answered by either a Freebase entity or a value or a list of Freebase entities. The questions are collected via Google suggest API and turkers are asked to answer the questions using only Freebase. A total number of 5810 QA pairs is collected, 35% is reserved for test; for the remaining 65%, a 80%-20% split is performed for training and validation.

Lexicons over entities are often constructed by string matching. However, such methods do not work well with relations as the relation mention often differs from its NL expressions. For example, in texts, we find 'Barack Obama was born in Honolulu' to express that 'Barack Obama's birthplace is Honolulu'. To construct

[38] Another main contribution of [Liang *et al.*, 2011] is to introduce a new semantic representation $\lambda - DCS$; we do not discuss further over this aspect inside the thesis as we do not focus on how the semantics is coded but rather how learning can be performed for given semantics.

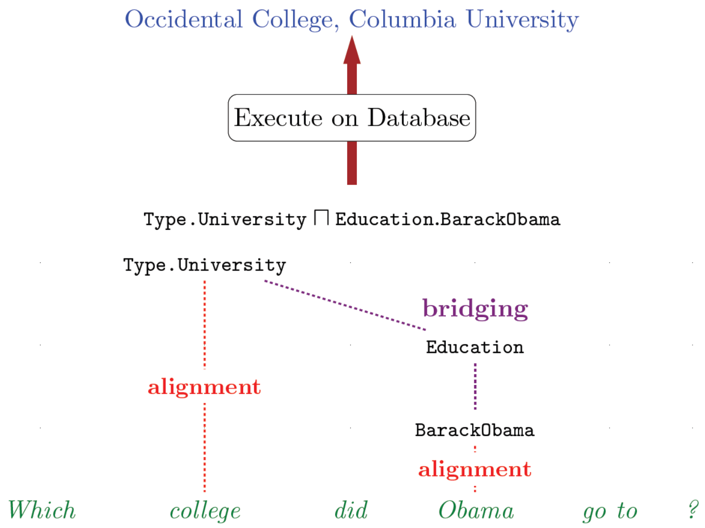


Figure 1.19: An example where the binary predicate 'Education' is generated on the fly while obeying type constraints (i.e 'bridging').

lexicons for relations, authors propose to use triples (e.g. 'barack_obama', 'born in', 'Honolulu') extracted from free texts (texts that one can find on the web) and align them with the knowledge base using the heuristics [Cai and Yates, 2013] which construct an alignment between a phrase and a relation if they co-occur with many same entities. For instance, in the above example, 'born in' and 'birthplace' co-occur with ('barack_obama', 'honolulu'); if they co-occur with many other such pairs, an alignment (a lexicon on relation) will be constructed.

The construction of LFs obeys type constraints for which authors propose to exploit as prior knowledge. Similar to [Liang *et al.*, 2011], in [Berant *et al.*, 2013], some LFs have to be generated on the fly; however, to limit the LF search space, [Berant *et al.*, 2013] note that one can generate LFs on the fly while taking into account type constraints (an operation called 'bridging'). Fig. 1.19 illustrates the 'bridging' operation. Once rules of type 'lexicons' are generated (which generate the LFs Type.Univesity and BarackObama), other rules such as 'Education' need to be generated on the fly. However, the predicates generated at this place which link Type.University and BarackObama must be binary predicates which operate on (Type.University, Type.Person) as type constraints; those constraints narrow down the search space of predicates.

The whole system (with predicate lexicons and bridging) is tested on Webquestions achieving 35.7% in accuracy.

[Berant and Liang, 2014] propose to enrich the feature set in the semantic parser by exploiting existing resources such as paraphrase corpus and pretrained word vectors.

For each LF query, they propose to use prespecified templates to generate a question in natural language called 'canonical form'. Using those templates, they are able to generate the question *Who directed Top Gun ?* from the LF query (? Directed.TopGun).[39]

Once the canonical forms are generated, alignment features can be estimated between canonical forms and the NL using an alignment model pretrained on a paraphrase corpus; besides, the similarity can also be estimated using pretrained word vectors [Mikolov *et al.*, 2013a] where both the representations of canonical forms and NLs are taken as the average of content word vectors (nouns, verbs and adjectives). The alignment features can detect different phrases expressing the same notion; for example, it can make 'genres' close to 'kinds' or 'types'. On the other hand, vectorial representation can identify correct paraphrases when it is hard to directly associate phrases from NL to canonical forms; for example, it can make the NL 'Where is made Kia car?' close to the canonical form 'What city is Kia motors a headquarters of?'.

Using those additional features, the semantic parser improves its ranker over LF candidates. When the system is tested on Webquestions dataset, it achieves an accuracy of 39.9%.

1.3.1.2 Smart Annotations

While annotation efforts can be reduced by replacing (NL, LF) pairs with (NL, A) pairs, annotation of LFs can itself be reduced as noted by [Wang *et al.*, 2015]. Instead of asking experts to annotate LFs for a given NL, the annotation process proposed by [Wang *et al.*, 2015] begins with LFs and is illustrated in Fig. 1.20. First, similar to [Berant and Liang, 2014], a grammar generates LFs paired with a textual form called canonical forms. For example, in Fig. 1.20, the grammar generates the LF **R(date).(student.Alice∩university.Brown)** paired with the canonical form **start date of student alice whose university is brown university**. The generated canonical forms do not need to be 'natural' English but have the property to reflect transparently the semantics of the sentence. Once the canonical forms are generated, they are given to turkers to be paraphrased giving paraphrases such as **When did alice start attending brown university**; pairs of (NL, LF) can then

[39] We refer interested readers to the original paper on the exact forms of the templates.

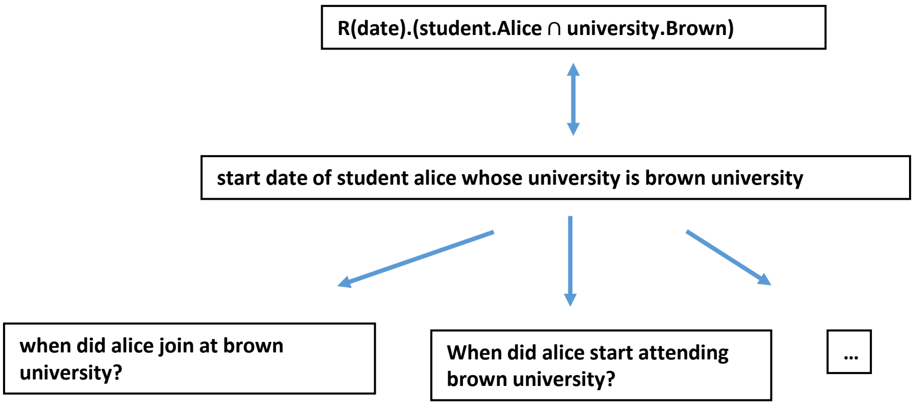

R(date).(student.Alice ∩ university.Brown)
start date of student alice whose university is brown university
when did alice join at brown university?
When did alice start attending brown university?
...

Figure 1.20: The annotation process proposed by [Wang *et al.*, 2015].

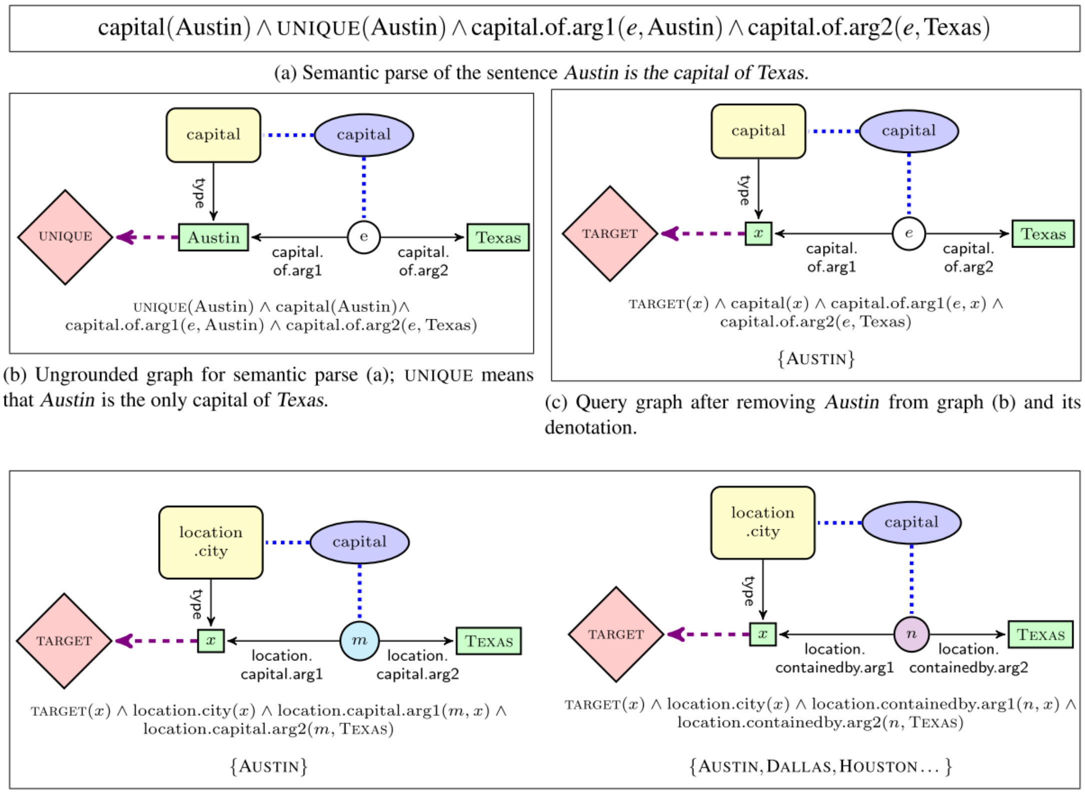
be collected where NLs are the paraphrases. This process allows to reduce annotation efforts as creating paraphrases is arguably much easier than annotating LFs.

1.3.1.3 Using Free Texts

[Reddy *et al.*, 2014] propose an approach to build a semantic parser without QA pairs (thus requiring no annotation effort). The key insight is to represent natural language (NL) as semantic graphs whose topology shares commonalities with the interested knowledge base (KB) such as Freebase; then the problem is reduced to learn to match graphs (between semantic graphs and the KB query graph). We will first explain how to represent NL into semantic graphs before going into graph matching algorithms. Fig. 1.21 illustrates the steps to convert a sentence ('Austin is the capital of Texas.') first into a semantic graph then into a Freebase grounded query graph.

To represent a sentence into a semantic graph, authors propose to first obtain the semantic parse of the sentence by parsing the sentence using a CCG. Fig. 1.22 illustrates the parsing process constructing both the semantics and the syntactics for the sentence 'Cameron directed Titanic'. At the beginning, all words are associated with both a syntactic and a semantic category (the whole representation called a lexicon); for example, the lexicon 'Cameron' constitutes the word associated with the syntactic category NP and the semantic category Cameron. Those lexicons are combined using forward and backward application till producing the syntactic category S and the semantics directed.arg1(e,Cameron)$\wedge$directed.arg2(e,Titanic).[40]

[40] Inside the thesis, we discuss in more detail CCGs being able to produce simultaneously syntax


capital(Austin) ∧ UNIQUE(Austin) ∧ capital.of.arg1(e, Austin) ∧ capital.of.arg2(e, Texas)
capital
capital
type
UNIQUE
Austin
e
Texas
capital.
of.arg1
capital.
of.arg2
UNIQUE(Austin) ∧ capital(Austin)∧
capital.of.arg1(e, Austin) ∧ capital.of.arg2(e, Texas)
capital
capital
type
TARGET
x
e
Texas
capital.
of.arg1
capital.
of.arg2
TARGET(x) ∧ capital(x) ∧ capital.of.arg1(e, x) ∧
capital.of.arg2(e, Texas)
{AUSTIN}
location
.city
capital
type
TARGET
x
m
TEXAS
location.
capital.arg1
location.
capital.arg2
TARGET(x) ∧ location.city(x) ∧ location.capital.arg1(m, x) ∧
location.capital.arg2(m, TEXAS)
{AUSTIN}
location
.city
capital
type
TARGET
x
n
TEXAS
location.
containedby.arg1
location.
containedby.arg2
TARGET(x) ∧ location.city(x) ∧ location.containedby.arg1(n, x) ∧
location.containedby.arg2(n, TEXAS)
{AUSTIN, DALLAS, HOUSTON ... }

(a) Semantic parse of the sentence *Austin is the capital of Texas*.

(b) Ungrounded graph for semantic parse (a); UNIQUE means that *Austin* is the only capital of *Texas*.

(c) Query graph after removing *Austin* from graph (b) and its denotation.

Figure 1.21: Steps involved in converting a natural language sentence to a Freebase grounded query graph [Reddy *et al.*, 2014].

One may note that the semantics in this example differs in forms with the semantics that we have seen in subsection 1.1.2 produced also by a CCG; this is because the semantic representation chosen here[41] is more related to how the interested KB (Freebase) represents the semantics.

Once the semantic parse is produced (i.e corresponding to the step Fig. 1.21(a)), the authors propose to specify rules to transform it into a graph representation based on each part of the semantic parse. For example, in Fig. 1.21(b), the semantics capital(Austin) is transformed into the entity Austin (indicated by the green rectangular box) having the property of being unique and having the type capital; the entity Austin is linked to another entity Texas in Fig. 1.21(b) through a binary relation whose mention in the sentence is capital, which is the graphical representation of the semantics capital.of.arg1(e,Austin)∧capital.of.arg2(e,Texas). Several semantic

and semantics in subsection 1.1.2.

[41] We refer interested readers to Appendix of [Reddy *et al.*, 2014] to see how authors define semantic patterns based on the CCG syntactic category information.

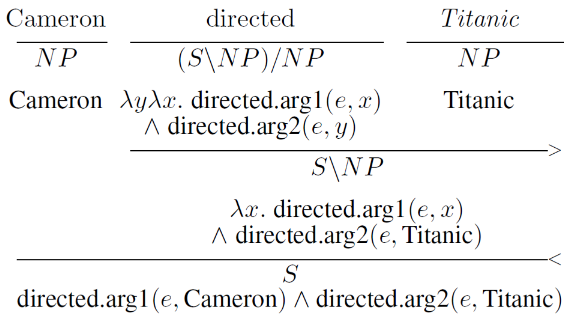

Cameron
directed
Titanic
NP
(S\NP)/NP
NP
Cameron
λyλx. directed.arg1(e, x) ∧ directed.arg2(e, y)
Titanic
>
S\NP
λx. directed.arg1(e, x) ∧ directed.arg2(e, Titanic)
<
S
directed.arg1(e, Cameron) ∧ directed.arg2(e, Titanic)

Figure 1.22: CCG derivation containing both syntactic and semantic construction.

graphs may be produced from one semantic parse due to ambiguity.[42]

At test time, a natural language query can be transformed into a graph representation (e.g. semantic graph Fig. 1.21(c)) using the above proposed techniques; however, the graph needs to be further transformed to make the graph representing a KB query so that it can be executed on the KB. The authors propose to use *denotations* as weak supervision to learn this transformation.

To learn the transformation, the authors propose to first *overgenerate* the possible KB graphs corresponding to the semantic graph. To be more precise, entity nodes in the semantic graph are mapped to KB entities through entity recognition[43], type nodes are mapped to all the possible KB types consistent with the entity; for example, the type 'capital' in Fig. 1.21(b) is mapped to types such as location.city, location.capital_city, book.book_subject, broadcast.genre etc; similarly an edge between two entities is mapped to all the edges linking the two entities in the KB. Obviously, only one (or a few) KB graphs will be faithful representation(s) of the sentence.

To actually get a faithful KB representation, the authors propose to use *denotations* as criteria. First, an entity is removed from the sentence graph where the entity becomes the 'target' (Fig. 1.21(c)); the same operation is performed also for each KB graph obtained by transforming the semantic graph. Then we consider a KB graph representation correct if its denotation (i.e. the result of the execution) is the same as the removed entity. Fig. 1.21(d) illustrates the procedure of selecting a correct KB graph representation. The two KB graphs in Fig. 1.21(d) differ from their

[42] The authors give one specific kind of ambiguity where a number can either be directly linked to an entity or be produced by some count operations.

[43] The web corpus Clueweb09 which the authors use already contain this mapping using entity recognition.

relation transformation (i.e the left graph transforms capital.of into location.capital while the right graph transforms the same relation into location.containedby), the left one is correct while the right one is not as the denotation entity Austin must be unique according to the semantic graph Fig. 1.21(b).

One can learn this graph transformation when semantic graphs are paired with their correct (in the sense of denotation) KB graphs. The authors propose to use a linear scoring function for the transformation where the KB graph with the maximum score is returned at test time:

$$(\hat{g}, \hat{u}) = argmax_{g,u}\phi(g, u, NL, KB) \cdot \theta$$

In the equation, u is the semantic graph and g is the transformed KB graph; $\phi(g, u, NL, KB)$ are prespecified features and θ are the parameters to be learned. Features characterizing the transformation include alignment between types (e.g. $\phi_{type}(capital, location.city)$ in Fig. 1.21) and edges (e.g. directed, film.directed_by), lexical similarity which counts the number of stems shared by the semantic graph and the KB graph (i.e. directed and film.directed_by has one stem overlap), etc. The parameters θ over features are learned using the averaged structured perceptron algorithm [Collins, 2002].

The whole model is tested on the a subset of Webquestions dataset involving questions on business, film and people; the system achieves an precision of 41.9%, a recall of 37.0% thus a F1 score of 39.3%. The system performs better than the system of [Berant and Liang, 2014] on this subset.

1.3.2 Reduce engineering efforts

Most semantic parsing systems we have discussed till this point requires significant engineering efforts. In [Berant *et al.*, 2013; Berant and Liang, 2014], feature engineering is required to specify the set of features for the log-linear ranker. In [Reddy *et al.*, 2014], the authors need to specify rules to transform the semantics of the sentence (obtained from a CCG parser) into a semantic graph; also, features have to be manually specified to learn the matching function between the semantic graphs and the KB graphs.

We will describe in this subsection several works trying to reduce the engineering efforts [Bordes *et al.*, 2014a; Yih *et al.*, 2015]. The idea is to use models that can learn the features and the classifier at the same time thus avoid having to specify features manually.

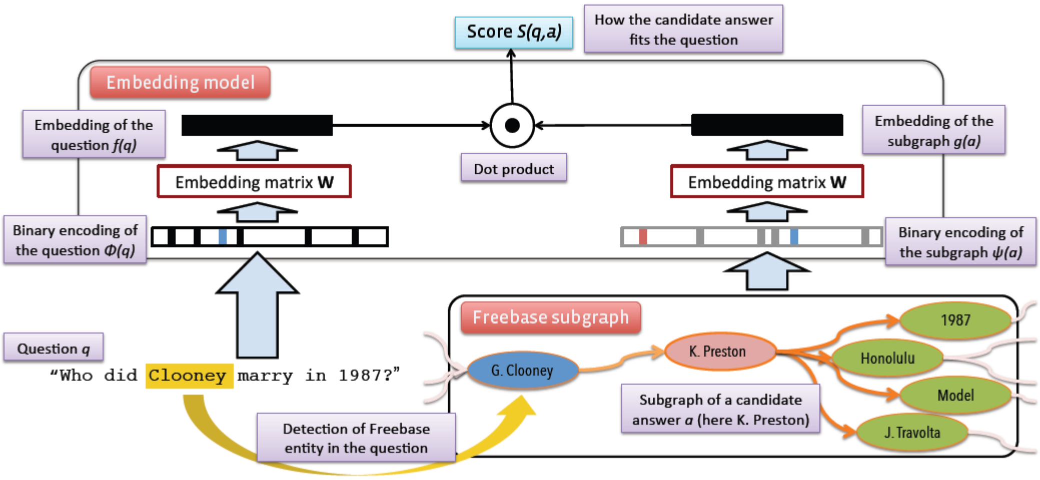

Score $S(q,a)$
How the candidate answer fits the question
Embedding model
Embedding of the question $f(q)$
Dot product
Embedding of the subgraph $g(a)$
Embedding matrix W
Embedding matrix W
Binary encoding of the question $\Phi(q)$
Binary encoding of the subgraph $\psi(a)$
Question q
"Who did Clooney marry in 1987?"
Detection of Freebase entity in the question
Freebase subgraph
G. Clooney
K. Preston
Subgraph of a candidate answer a (here K. Preston)
1987
Honolulu
Model
J. Travolta

Figure 1.23: Illustration of the subgraph embedding model scoring a candidate answer [Bordes *et al.*, 2014a].

1.3.2.1 Matching answer subgraphs

[Bordes *et al.*, 2014a] propose an approach to answer natural language questions with few handcrafted features. The main idea is to represent both the semantics of the question (q) and the answer (a) into real-valued vectors (also called embeddings) of the same dimension and then learn a scoring function (i.e the dot product of the two vectors) between them. As we shall see that one can use the question or the answer directly to map them to the embedding space, the approach avoids manually specifying features (for the question or for the answer).

Fig. 1.23 gives a detailed illustration of the approach to answer the factoid questions like 'Who did Clooney marry in 1987?' in Webquestions[44] using Freebase as the KB. First, an entity is extracted from the question using string matching between the words in the question and KB entity mentions[45]. Using this entity, a subgraph representing the answer (Freebase subgraph in Fig. 1.23) is calculated which involves 1) a path from the extracted entity to the candidate answer[46] 2) other entities directly linked (via 1 predicate) to the candidate answer.

Then embeddings of both the question and the answer subgraph are calculated. Let N_W denote the total number of words and N_S the total number of entities and relations; with $N = N_W + N_S$, W is a matrix of $R^{k \times N}$ and each column of W

[44] Recall that the answers for the questions in Webquestions are either an entity or a value or a list of entities.

[45] When several entities can be extracted, the entity with most connections in the KB is chosen.

[46] To limit the search space at prediction time, the path involves only one or two predicates (1 predicate married_to in Fig. 1.23)

corresponds to the embedding of either a word or a KB element (i.e. a relation or an entity) of dimension k where we force the norm L_2 of each embedding to be no more than 1. The question is first mapped into a sparse vector $\phi(q) \in N^N$ indicating the number of times each word appearing in the question; then the embedding of the question $f(q)$ is calculated as $W\phi(q)$. Similarly, the subgraph representing the answer is mapped into another sparse vector $\psi(a) \in N^N$ indicating the number of times each KB element (an entity or a relation) appearing in the subgraph and the embedding of the answer subgraph $g(a)$ is calculated as $W\psi(a)$. The scoring function between the question and the answer subgraph $S(q, a)$ is defined as the dot product between the two embeddings: $S(q, a) = f(q) \cdot g(a)$.

To learn the model (the embedding matrix W), the authors propose to minimize a margin based loss. Let $D = \{(q_i, a_i), i = 1, 2, \dots, |D|\}$ be the training set of questions (q_i) paired with their answer subgraph (a_i). The loss to minimize is:

$$\sum_{i=1}^{|D|} \sum_{\bar{a} \in \bar{A}(a_i)} max\{0, m - S(q_i, a_i) + S(q_i, \bar{a})\}$$

where m is the margin (fixed to 0.1). $\bar{a}$ is an incorrect answer subgraph sampled from $\bar{A}(a_i)$ the set of incorrect subgraphs for a_i. The $\bar{a}$ is sampled either by changing the candidate path (i.e. the path from the extracted entity in the question to the candidate entity in a_i) to point to another entity or directly changing the entity answer to another entity randomly generated. Optimization is accomplished via stochastic gradient descent. At test time, beam search is used to generate answer subgraphs that are ranked using the embedding model; the answer subgraph with the maximum score is returned.

Learning W ($N \times k$ parameters to learn) requires a lot of data and Webquestions alone is not enough. The authors propose to both use the KB and other resources (e.g Clueweb09) to generate question answer pairs using templates. For example, Freebase contains triples in the form of (entity1, type1.type2.predicate, entity2); given the triple (barack_obama, person.country.nationality, united_states) and the template 'what is the *predicate* of the *type1 entity1* ?', one can generate a question answer pair ('what is the nationality of barack obama ?', united states).

The system is tested on Webquestions achieving a F1 score 39.2 (compared to 39.9 in [Berant and Liang, 2014]).

Remark The main problem with the approach [Bordes *et al.*, 2014a] is that the subgraph representing the answer does not necessarily match with the semantics

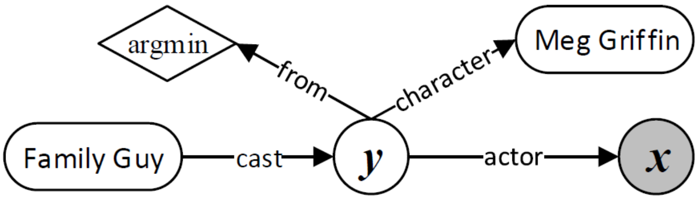


Figure 1.24: The Freebase graph representation for the question 'Who first voiced Meg on Family Guy?'.

of the question; notably, some semantics in the question will be missing from the subgraph. Consider the question 'Who first voiced Meg on Family Guy?' whose Freebase graph representation is shown in Fig. 1.24. The answer subgraph according to [Bordes *et al.*, 2014a] will only contain an instantiation of the 'core chain' $Family\ Guy \xrightarrow{cast} y \xrightarrow{actor} x$ and the entities directly linked to x where x is instantiated with a certain entity; the entity *Meg Griffin* and the constraint *argmin* indicated by 'Meg' and 'first' respectively in the sentence will be omitted in the subgraph representation (if the entity *Meg Griffin* is not connected with the instantiated entity x). The mismatch between the semantics actually expressed and the KB subgraph gives an unwanted bias to the system that can harm the performance.

1.3.2.2 Matching sentence semantic subgraphs

In [Yih *et al.*, 2015], contrary to [Bordes *et al.*, 2014a], the KB graph represents reliably the semantics of the sentence (e.g. the Freebase graph representation of the sentence 'Who first voiced Meg on Family Guy?' is the one shown in Fig. 1.24), which is similar to [Reddy *et al.*, 2014]. However, instead of constructing the graph from the semantics produced by a CCG parser like [Reddy *et al.*, 2014], the authors propose to directly look for the KB subgraph which maps best to the sentence, in a spirit similar to [Bordes *et al.*, 2014a].

The KB subgraph searching procedure in [Yih *et al.*, 2015] consists of 3 stages as shown in Fig. 1.25. First, an entity is identified in the natural language question using an entity recognizer designed for short and noisy text [Yang and Chang, 2016]. The 10 top ranked entities are kept for later search stages together with their entity linking score. On the left of Fig. 1.25 shows the KB graph expanding from an empty graph to a graph with one entity node during the first stage.

The second stage identifies the 'core chain' which is a chain graph starting from

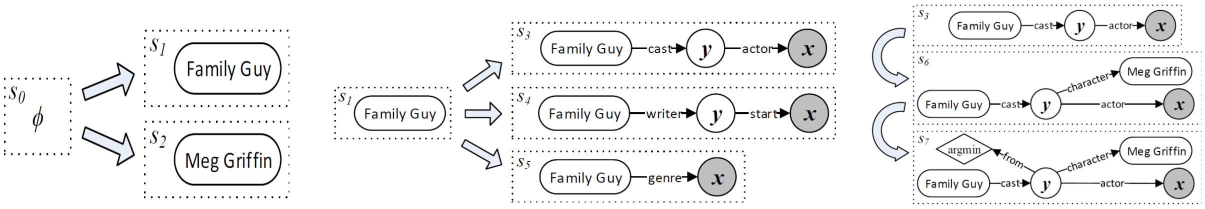


Figure 1.25: The subgraph search procedure for mapping a natural language question into a KB subgraph in [Yih *et al.*, 2015].

the identified entity node to the entity node being queried. To limit the search space, like [Bordes *et al.*, 2014a], the 'core chain' is limited to contain no more than two predicates. The middle of Fig. 1.25 shows the KB graph expanding from the entity node 'Family Guy' to graphs containing a core chain. Note that as the entity node is given from the previous stage, only legitimate predicates are searched in this stage, similar to the subgraph construction procedure in [Bordes *et al.*, 2014a] or $\lambda - DCS$ tree construction procedure in [Berant *et al.*, 2013].

To select the best 'core chains' (i.e identifying the best sequences of predicates), the authors propose to use a CNN architecture to map both the natural language question and the 'core chain' to an embedding space before calculating a matching score between the two embeddings. The CNN architecture is shown in Fig. 1.26 and applies to both the questions and the 'core chains'. First, words are represented as a sparse vector of letter-trigrams ($x_t \rightarrow f_t$). Then convolutions are used to project the words (represented by vectors of letter-trigrams) to a local feature vector ($f_t \rightarrow h_t$), followed by max pooling operation extracting the most salient local features to form a global feature vector v. v is passed to an MLP (multilayer perceptron) to finally obtain the semantic feature y for either the question or the sequence of predicates.

Once the core chain is constructed, some constraints need to be added so that the subgraph reflects the whole semantics of the sentence and executes to a correct answer. The authors propose to add constraints using simple rules. For example, the argmin/argmax constraints are triggered if the sentence contains words such as 'first', 'latest'.

As several candidates are kept at each stage, many subgraphs are produced after the three stages; a log-linear ranker with features is used to choose the best KB subgraph. To train the ranker and the CNN identifying the core chain, the authors propose to first find the correct KB subgraphs for each question (which is found to be feasible over Webquesions) and then use these subgraphs as supervised signals for training. The whole system with three stages of search and the ranker is tested

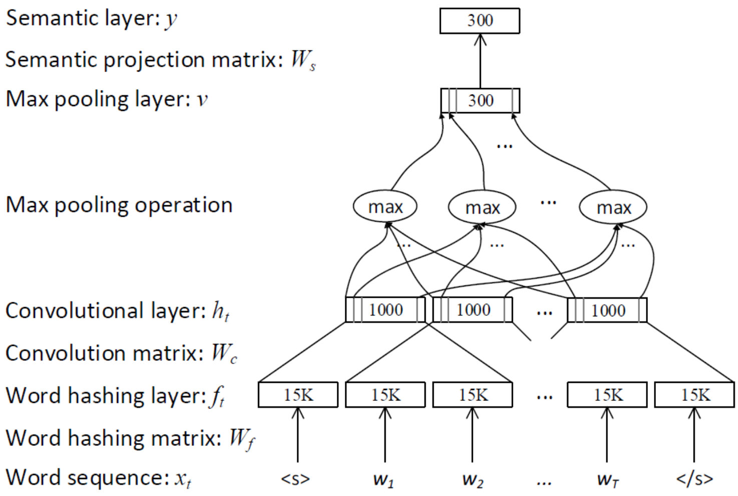


Figure 1.26: The convolutional neural networks (CNN) used to score the match between the question and the core chain (a sequence of predicates).

on Webquestions and achieves a F1 score 52.5, the best of all the systems we have reviewed so far.

Remark Both [Bordes *et al.*, 2014a] and [Yih *et al.*, 2015] learn embeddings of sentences and KB subgraphs for their QA systems with different choices on architectures. [Bordes *et al.*, 2014a] learn the embedding of the sentence (or the subgraph) based on the bags of words representation; [Yih *et al.*, 2015] learn the embedding based on letter-trigrams representation. The later one is more robust to spelling errors. The architecture of [Yih *et al.*, 2015] is also more deep with convolution and max-pooling layers, possibly capturing more appropriate features to learn the scoring function between the question and the KB subgraph.

In the later chapters (chapter 5,6), we will discuss our works amongst others on constructing semantic parsers using Recurrent Neural Networks (RNNs). We (and probably other authors) are inspired by the work [Sutskever *et al.*, 2014] in the field of machine translation where the authors propose to use RNNs to map a sentence in the source language to a sentence in the target language. Some descriptions and use cases of RNNs are discussed inside this thesis in Chapter 3 about neural networks.

Chapter 2

Automata and Grammars

Contents

2.1 Automata and Context-Free Grammars

In this section, we introduce the notions of finite state automata (FSA), context free grammars (CFG), and their weighted versions (WFSA,WCFG). These core concepts will be used in later chapters. The operations over these objects (notably intersections) together with the associated algorithms will be described in the next section.

Definition 2.1. A weighted finite state automaton (WFSA) is represented formally by a 6-tuple $(Q, \Sigma, \delta, i, f, \nu)$, where:
Q is a finite set of states,
Σ is a finite set of symbols, called the alphabet of the automaton,
δ is the transition function, $\delta : Q \times \Sigma \to Q$,
i is the start state ($i \in Q$), the state of the automaton before any input has been processed,
f is a set of states of Q ($f \subset Q$) called final states,
ν is the weight function, $\nu : \delta \to K$ where K is a *semiring.*

Semirings used in NLP applications include boolean semirings, log semirings and tropical semirings. See Fig. 2.1 for some of their properties. We refer interested readers to [Mohri, 2009] for detailed descriptions. WFSAs over boolean semirings

SEMIRING	SET	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+ \cup \{+\infty\}$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Figure 2.1: Some examples of semirings.

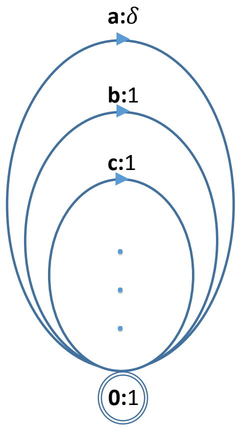


Figure 2.2: A WFSA example with weight $\delta \ll 1$.

have equivalent representations as non weighted FSAs. In the thesis, we mostly use WFSAs with probability semirings.

Fig. 2.2 shows a WFSA example. The automaton has only one state that is both initial (indicated by 0) and final (indicated by the double circle). The weights belong to the probability semiring and are indicated after ':' on both transition arcs and final states. This automaton generates sequences such as a, bc, etc. and associates a weight with every sequence..

As most of the weights are 1, many sequences will be of weight 1 (weights are combined through multiplication in the probability semiring, Fig. 2.1) with the exception of those sequences containing the symbol a. If a sequence contains $k \geq 1$ instances of a, it will receive a weight of δ^k which is much smaller than 1. So this automaton concisely expresses the belief that a sequence is unlikely to contain a instances. In this thesis, we use automata to express beliefs that we have over LFs in semantic parsing.

Definition 2.2. A weighted context free grammar (WCFG) is defined by the 5-tuple $G = (V, \Sigma, R, S, \mu)$ where:

V is a finite set; each element $v \in V$ is called a *nonterminal* (sometimes also called syntactic category).

Σ is a finite set of *terminals*, disjoint from V, which make up the actual content of a sentence. The set of terminals is the alphabet of the language defined by the grammar G.

R is a finite relation, $R \subseteq V \times (V \cup \Sigma)^*$, where the asterisk represents the Kleene star operation. The members of R are called the *rules* or *productions* of the grammar.

S is the start symbol, used to represent the whole sentence. It must be an element of V.

μ is the weight function, $\mu : R \rightarrow K$ where K is a semiring.

Non-weighted CFGs have equivalent representations as WCFGs over boolean semirings. In the thesis, we use WCFGs over both boolean semirings and probability semirings.

A WCFG provides a simple and mathematically precise mechanism for describing the methods by which sentences are built from smaller components, capturing the "compositional structure" of sentences. As an example, consider the following WCFG containing four rules defining arithmetic expressions with weights specified after ':':

(1) $\mathbf{E} \rightarrow \mathbf{E} + \mathbf{E} : 0.8$

(2) $\mathbf{E} \rightarrow \mathbf{E} * \mathbf{E} : 0.9$

(3) $\mathbf{E} \rightarrow (\ \mathbf{E}\ ) : 1$

(4) $\mathbf{E} \rightarrow id : 1$

Here, The only nonterminal is **E** (which is also the start symbol), the terminals are $+$, $*$, (,) and *id*. Using this grammar, we can obtain complex arithmetic expresssions such as $(id + id) * id$.

We can consider a *derivation tree* (DT) that transforms the start symbol into this sentence. For each derivation tree (DT), the root is a rule identifier associated with the start symbol, all internal nodes are labelled also with rule identifiers[47] and all leaves are labelled with terminals. For example, Fig. 2.3 shows the derivation tree that generates the arithmetic expression $(id + id) * id$.

One can also associate a derivation sequence (DS) with the sentence which is a sequence of rules to transform the start symbol into the sentence. It can be easily seen

[47] In Fig. 2.3 rule identifiers are indicated by indexed nonterminals.

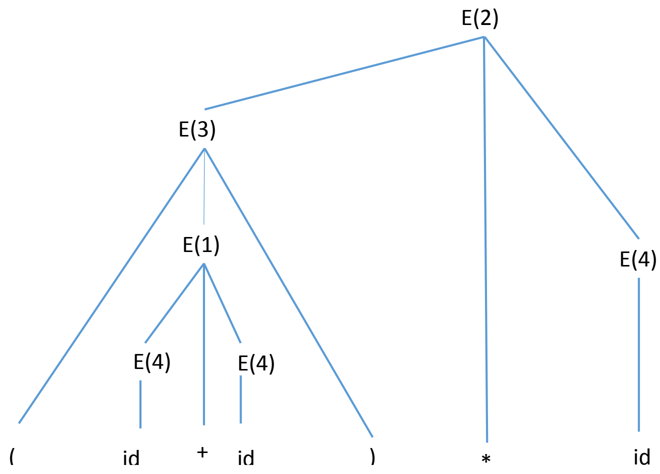


Figure 2.3: The derivation tree producing the string (id+id)*id.

that if we construct a sequence of rules using the leftmost traversal of a derivation tree, we obtain a DS. For example, the leftmost traversal of the tree in Fig. 2.3 corresponds to the derivation sequence: (2),(3),(1),(4),(4),(4). The weight associated to this DS/DT is the multiplication of the weights for each rule: 0.9*1*0.8*1*1*1=0.72.

One can note that this simple WCFG always guarantees a balanced parenthesis as the application of rule (3) implies that parentheses always come in pairs, something that one cannot guarantee using a WFSA. In fact, WCFGs are strictly more powerful than WFSAs and a proof for non weighted version (with CFG and FSA) can be found in [Hopcroft *et al.*, 2006].

2.2 Intersection between a WCFG and WFSA(s)

We will describe an intersection algorithm between WFSA(s) and WCFG in this section. Two elements motivate us for introducing the algorithm. Firstly, such algorithms are 'historically' important for many computational linguistic tasks (i.e. syntactic parsing, hierarchical machine translation [Chiang, 2005]). Secondly, the symbolic models (i.e. based on WFSA(s) and WCFG) possess important 'modularity properties' which is a consequence of having efficient intersection algorithms and those properties are typically missing for recently more popular deep learning models.

Historical importance Intersection algorithms are closely linked to the problem of syntactic parsing as noted by [Billot and Lang, 1989; Nederhof and Satta, 2003]. One can formally see the problem of syntactic parsing as intersection between a

CFG and a 'sentence FSA'.[48] Furthermore, to model the preference amongst all the possible parses according to the grammar, one needs to weight the FSA and the CFG thus formally intersects a WFSA with a WCFG.

Intersection algorithms also play an important role in hierarchical machine translation [Chiang, 2005]. Under these translation models, a weighted SCFG (synchronous CFG) is used to translate a source sentence into a target sentence. At decoding time, given a source sentence, all possible target sentences according to the SCFG form a WCFG. To choose the target sentence among all these possibilities, one typically takes into account a language model by intersecting the WCFG with the language model. As language models are often WFSAs (i.e. n-grams models), the intersection operation is formally intersecting a WCFG and a WFSA as noted by [Aziz *et al.*, 2014].

Modularity properties The use of intersection algorithms permits a modular design. For example, suppose one needs to produce a model producing sentences taking into account several constraints (i.e word coming from a dictionary, sentence being grammatically correct, etc.). Instead of building one single model at from scratch, one can build much 'smaller models' each taking into account one type of constraints, which is arguably much easier; however, to solve the original task, one has the problem of combining these 'smaller models' through intersection, so this modular design is feasible as long as efficient intersection algorithms exist between those 'smaller models'.

One should remark that it is non trivial to ensure the existence of an efficient intersection algorithm between models and this property is indeed missing for many learning devices. For example, one could try to implement the 'smaller models' above using neural networks instead of symbolic models. However, there are no obvious ways to combine (intersect) the neural networks to form a new model taking into account all the constraints.[49]

2.2.1 Intersecting a WFSA with a WCFG

Intersection between an FSA and a CFG is a CFG. The proof [Bar-Hillel *et al.*, 1961] relies on the construction of a CFG which represents the intersection. The

[48] The sentence FSA is a linear automaton having the initial state as the start of the sentence; at each word it transits from one state to the next one till the end of the sentence, corresponding to the final state.

[49] To combine neural networks, some researchers have proposed to approximate them using finite state machines (WFSAs) [Deoras *et al.*, 2011].

construction can be easily extended to the weighted cases [Nederhof and Satta, 2003] (i.e a WFSA and a WCFG) that we will describe here.

Given inputs a WCFG $G = (V, \Sigma, R, S, \mu)$ and a WFSA $M = (Q, \Sigma, \delta, i, f, \nu)$,[50] the intersection between G and M results in a WCFG $G_\cap = (V_\cap, \Sigma, R_\cap, S_\cap, \mu_\cap)$ where $V_\cap = Q \times (\Sigma \cup V) \times Q$, $S_\cap = (i, S, f)$ and $R_\cap$ consists of the set of rules obtained as follows:

- For each rule $\pi = (A \to X_1 \dots X_m) \in R$,[51] $m \geq 0$ and each sequence of states $r_0 \dots r_m \in Q$, add the rule $\pi_\cap = ((r_0, A, r_m) \to (r_0, X_1, r_1) \dots (r_{m-1}, X_m, r_m))$ to $R_\cap$; for $m = 0$,[52] let $R_\cap$ contain the rule $\pi_\cap = ((r_0, A, r_0) \to \epsilon)$ for all $r_0 \in Q$. In both cases, let $\mu_\cap(\pi_\cap) = \mu(\pi)$.
- For each transition $\tau = (r \xrightarrow{a} t) \in \delta$, let the rule $\pi_\cap = (r, a, t) \to a$ be in $R_\cap$ and let $\mu_\cap(\pi_\cap) = \nu(\tau)$.

One can check that the resulting grammar $G_\cap$ represents all the sentences accepted by both G and M. Furthermore, the weight for a sentence in $G_\cap$ is the multiplication of its weights in G and M. Formal proof of correctness on this weighted intersection algorithm can be found in [Nederhof and Satta, 2003].

However, the above intersection algorithm is too inefficient to be useful in practice. For an automata M having n states and any rule $\pi = (A \to X_1 \dots X_m) \in R$, the intersection algorithm described above constructs a $G_\cap$ producing n^{m+1} rules out of the original rule π.[53]

The resulting $G_\cap$ contains many rules that are non-generating, meaning that no terminal string can be derived from the nonterminal on the LHS of these rules. One can use a bottom-up procedure to avoid generating these rules making the intersection algorithm less inefficient and more useful in practice. The bottom-up procedure that we will describe is an adaption of that proposed by [Nederhof and Satta, 2008] and is related to the syntactic parsing algorithm by [Graham *et al.*, 1980].

Before entering into the intersection algorithm, we introduce the dot notation: given a production $A \to \alpha\beta \in R$, the notation $A \to \alpha \cdot \beta$ represents a condition

[50] Note that without loss of generality, we suppose that G and M share the same set of terminals Σ.

[51] Note that throughout the section, the X_i involved in the rule can be either terminals or nonterminals.

[52] The cases $m = 0$ correspond to rules producing the empty string noted by $A \to \epsilon$.

[53] To see the number of rules, one notes that the new rules are constructed by 'inserting' states between two consecutive X_i, X_{i+1} or at the beginning or at the end of the rule ($m + 1$ possibilities); as this holds true for each state insertion, the number is n^{m+1}.

in which α has already been handled[54] and β is expected. As we apply a dynamic programming procedure (very similar to syntactic parsing) for the intersection, the dot notation helps to record the current state for the rules (i.e. rules in the items I and the agenda A in Fig. 2.4); in particular, it helps to distinguish two states involving the same rule but that are not 'handled' till the same position inside the rule.

Fig. 2.4 describes the bottom up procedure for the intersection. For a state transition in the automata such as $s \xrightarrow{a:w'} t \in \delta$, the w' denotes the weight associated with the transition; similarly, the w in $A \xrightarrow{w} \alpha \in R$ denotes the weight of the rule. Thus, compared to the bottom up procedure in [Nederhof and Satta, 2008], we incorporate the calculation of weights into our algorithm. Furthermore, our algorithm explicitly outputs all the rules of the resulting WCFG (i.e $R_\cap$).

We will first give a high-level description of the algorithm in Fig. 2.4. In this description, we will not make any distinction between elements in I and elements in A; the distinction is only necessary at a later stage of the description.

Note that as in our previous algorithm, all the rules in the new grammar $R_\cap$ are generated by 'indexing' rules in the original grammar R.[55] The difference is that now the algorithm will choose indexes that correspond to generating rules. To do this, at the intial state, we have all the rules with the dot at the beginning of each right-hand side. Then the algorithm advances by producing new dotted rules (i.e by advancing the dots) at each iteration from the current dotted rules and the nonterminals that are known to be generating (i.e N in Fig. 2.4); the algorithm stops when no more dotted rules can be further deduced from all the current dotted rules and the generating nonterminals. When the algorithm stops, $R_\cap$ will consist of the set of all the rules with the dot situated at the end (or we can collect them during each iteration as in our algorithm).

Now, let's consider how to produce new dotted rules from current dotted rules and generating nonterminals. There are two ways:

- Given a dotted rule $(r, A \xrightarrow{w} \alpha \cdot X\gamma, s)$ and knowing that (s, X, t) is a generating nonterminal, one can produce a new dotted rule by advancing the dot: $(r, A \xrightarrow{w} \alpha(s, X, t) \cdot \gamma, t)$. This is shown in lines 22-25 in Fig. 2.4.

- Given a dotted rule where the dot is at the end of the rule $(r, A \xrightarrow{w} \alpha\cdot, s)$, one can add the nonterminal (r, A, s) into the set of generating nonterminals N by

[54] 'Handled' in the sense that all the nonterminals involved in α will be generating.

[55] Exceptions are the rules expanding the nonterminals in the form (s, a, t) where a is a terminal. These rules are generated by examing the transitions in the automata; as they are all generating by construction, we simply add them in our algorithm in Fig. 2.4 (line 10-12).

```
main():
        $\boldsymbol{N} = \emptyset$ {set of all generating terminals for $N_\cap$}
        $\boldsymbol{I} = \emptyset$ {multiset of passive items}
        $\boldsymbol{A} = \emptyset$ {multiset of active items to be processed}
        $\mathbf{R}_\cap = \emptyset$ {set of rules for the new grammar}

        for all $s \in Q$ :
                for all $(A \overset{w}{\rightarrow} \alpha) \in R$ :
                        $\boldsymbol{A} = \boldsymbol{A} \cup \{(s, A \overset{w}{\rightarrow} \cdot \alpha, s)\}$
        for all $(s \xrightarrow{a:w'} t) \in \delta$ :
                add_symbol$(s, a, t)$
                add_rule_terminal$(s \xrightarrow{a:w'} t)$
        while $\boldsymbol{A} \neq \emptyset$ :
                choose a rule $\{(s, A \overset{w}{\rightarrow} \alpha \cdot \beta, t)\}$ from $\boldsymbol{A}$
                $\boldsymbol{A} = \boldsymbol{A} - \{(s, A \overset{w}{\rightarrow} \alpha \cdot \beta, t)\}$
                add_item$(s, A \overset{w}{\rightarrow} \alpha \cdot \beta, t)$

add_item$(r, A \overset{w}{\rightarrow} \alpha \cdot \beta, s)$ :
        if $\beta = \epsilon$ :
                add_symbol$(r, A, s)$
                add_rule_nonterminal$(r, A \overset{w}{\rightarrow} \alpha, s)$
        else if $\beta = X\gamma$ :          # X is either a terminal or a nonterminal
                $\boldsymbol{I} = \boldsymbol{I} \cup \{(r, A \overset{w}{\rightarrow} \alpha \cdot \beta, s)\}$
                for all $(s, X, t) \in \boldsymbol{N}$ :
                        $\boldsymbol{A} = \boldsymbol{A} \cup \{(r, A \overset{w}{\rightarrow} \alpha\ (s, X, t) \cdot \gamma, t)\}$

add_symbol$(s, X, t)$ :
        if $(s, X, t) \notin \boldsymbol{N}$ :
                $\boldsymbol{N} = \boldsymbol{N} \cup \{(s, X, t)\}$
                for all $(r, A \overset{w}{\rightarrow} \alpha \cdot X\beta, s) \in \boldsymbol{I}$ :
                        $\boldsymbol{A} = \boldsymbol{A} \cup \{(r, A \overset{w}{\rightarrow} \alpha\ (s, X, t) \cdot \beta, t)\}$

add_rule_terminal$(s \xrightarrow{a:w'} t)$ :
        $\mathbf{R}_\cap = \mathbf{R}_\cap \cup \{((s, a, t) \overset{w'}{\rightarrow} a)\}$

add_rule_nonterminal$(r, A \overset{w}{\rightarrow} \alpha, s)$ :
        $\mathbf{R}_\cap = \mathbf{R}_\cap \cup \{((r, A, s) \overset{w}{\rightarrow} \alpha)\}$
```

Figure 2.4: The bottom up procedure of the intersection algorithm. Inputs are WCFG $G = (V, \Sigma, R, S, \mu)$ and WFSA $M = (Q, \Sigma, \delta, i, f, \nu)$.

making $N = N \cup \{(r, A, s)\}$. Having new generating nonterminals (lines 19-21) will further help to produce new dotted rules by advancing the dot.

To summarize this high-level description, to calculate the intersection between a WFSA M and a WCFG G, one uses two ways described above to generate exhaustively all the new dotted rules from the original rules dotted at the beginning and the initial generating nonterminals (lines 10-12); after the exhaustive generation, the rules of the new grammar consist of the dotted rules with the dot at the end.

Now to actually implement this idea, one needs to specify an order of generating new dotted rules through a loop (lines 13-16).[56] The dotted rules are now distinguished by either belonging to the items I or to the agenda A. I are the *passive* dotted rules as at the beginning of each iteration, no more dotted rules can be further deduced from $\{N, I\}$ — they are all in $\{N, I, A\}$; on the contrary, A are the *active* dotted rules that may generate new dotted rules with current $\{N, I\}$.

With this distinction, at each iteration, we pop out an element from A and obtain either a dotted rule or a generating nonterminal and add it to the corresponding groups (i.e N or I); then new dotted rules are produced from the updated $\{N, I\}$ and are added into A guaranteeing that *all the dotted rules that can be deduced from the current* $\{N, I\}$ *are always in* $\{N, I, A\}$ *after each iteration*; this invariance in particular guarantees that all the dotted rules will have been produced when the loop is finished[57] (i.e. when no more elements can be popped from A).

The main loop of generating dotted rules is shown in lines 13-16 in Fig. 2.4; after popping an element from A, it calls the procedure *add_ item* which first either adds a nonterminal in N (i.e $\beta = \epsilon$) or adds a dotted rule in I (i.e. $\beta = X\gamma$). In both cases, new dotted rules will be generated from the updated $\{N, I\}$ and will be added into A. During the loop, a rule is added into $R_\cap$ (line 21) only when the nonterminal on its LHS is known to be generating.

The way that our algorithm handles weights is very similar to [Nederhof and Satta, 2003], as we have described at the beginning of this subsection. For rules that are generated by indexing the rules in the original grammar (lines 36-37), their weights remain unchanged; for rules that are added by inspecting the transition in the automata (lines 33-34), their weights are the weights for the associated transition.

[56] Specifying the generation order can avoid in particular performing the same operations several times.

[57] Dotted rules $(r, A \xrightarrow{w} \alpha \cdot \beta, t)$ are created from the rules in the original grammar $(A \xrightarrow{w} \alpha\beta)$ by choosing r, t and the insertion place of the dot. As the number of choices for r, t and the insertion place are all finite, the number of all possible dotted rules is finite, which guarantees that the loop finishes.

Remark that the bottom up procedure described above eliminates non-generating nonterminals but does not eliminate 'non reachable' nonterminals. A nonterminal is said to be 'reachable' if a string involving that nonterminal can be derived from the start symbol (i.e S in $G = (V, \Sigma, R, S, \mu)$). A top down procedure can be applied after the bottom up procedure [Nederhof and Satta, 2008] to effectively eliminate non reachable nonterminals in the resulting intersected grammar. In practice, we use an intersection algorithm adapted from the Earley parsing algorithm [Dyer, 2010] which combines bottom up and top down elements.

A final remark concerns intersecting WFSA(s) and a WCFG. Till now, we have focused on the intersection algorithm between one WFSA and a WCFG; as the result of this intersection is still a WCFG, one can repeat the process to intersect several WFSAs and a WCFG. However, as a much faster algorithm exists for intersecting WFSAs which gives another WFSA,[58] it may be preferable to first intersect the WFSAs and then intersect the resulting WFSA with the WCFG.

[58] The library Openfst http://www.openfst.org implements such algorithms.

Chapter 3

Neural Networks

Contents

3.1 Multilayer Perceptrons

3.1.1 The Multilayer Perceptron model

A Multilayer Perceptron (MLP) is a neural network that can map an input vector x to an output vector y through a cascade of non-linear functions f_i giving the following equation between the input and the output:

$$y = f_n(f_{n-1} \dots f_1(x)).$$

Given an input vector $z \in R^{m_1}$, all the functions f_i in the MLP take a particular form: $f_i(z) = g_i(W_i(z))$.[59] $W_i \in R^{m_2 \times m_1}$ is a matrix transforming the m_1 dimensional real-valued vector z into another m_2 dimensional real-valued vector and g_i is a non-linear differentiable function applied to the m_2 dimensional real-valued output vector. Some popular choices of g_i for the intermediate layers $(f_1, f_2, \dots, f_{n-1})$ are element-wise sigmoid/tanh/relu.[60] Note that in neural network terminology, we refer to the function f_i as *the ith layer* and say that the ith layer outputs a vector of dimension m_2; we refer to the number of layers in the neural network as its *depth*.

When the MLPs are used for multiclass classification, the g_n of the last layer is generally a softmax function: $g_n(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ where z is the real-valued output vector of the last layer (z_j denotes the value of the jth coordinate in z) and K is its dimension.

MLPs possess some mathematical properties that may be related to their good performance in different machine learning tasks. One of the properties is being 'universal approximators', given by the *universal approximation theorem*. The theorem states that an MLP can approximate any continuous function on a compact set of R^n given enough neurons (i.e vector size of hidden layers). [Cybenko, 1989] proved a first version of the theorem for sigmoid activation functions; the particular choice of sigmoids was generalized later to a wider range of activation functions [Hornik, 1991].[61]

The theorem shows that the MLPs are expressive models. However, it does not address the question of finding parameters such that the MLP approximates well a given continuous function.

3.1.2 Learning through BackPropagation

Now that we have specified and parametrized our model as MLPs, we discuss the learning procedure to choose parameters. In MLPs, the parameters to learn are the weight matrices W_i for each layer. We suppose that we are in a supervised setting

[59] More precisely, $f_i(z) = g_i(W_i(z) + b_i)$ where b_i is referred as the bias term; as one can always extend W_n and x to integrate the bias, we omit it through the chapter for clarity.

[60] The relu operator is not differentiable at 0; however, all the presentations in this chapter can be easily adapted to the cases where the operator is differentiable almost everywhere instead of being differentiable. For clarity, we only discuss about differentiable cases.

[61] As both proofs only need to assume one single hidden layer for the MLPs to provide universal approximators, no advantages are shown here for having multiple hidden layers. We do not know of any formal proofs but there are many empirical results and intuitions [Bengio, 2009] related to the advantage of using 'deep' neural networks.

where an x in the training data is annotated with its correct one-hot vector y'[62] for the true label; we use softmax as the non-linear transformation for the last layer so that our neural network outputs a probability distribution over all the labels.

Before starting the learning procedure, we need to specify our *loss function.* A loss function for us is a non-negative function $f(y, y')$ which reflects our belief about good models; y' is the one-hot vector for the true label and y is our MLP output. As our MLP predicts a probability distribution y on each input data x, the typical loss that we use is the cross-entropy loss: $f(y, y') = -\langle y', log\ y\rangle = -\sum_i y'_i\ log\ y_i$ where index i iterates through the coordinates of y and y'; the cross entropy loss is minimized when y equals to y' and intuitively the loss is smaller when y is closer to y'. During our training procedure (i.e learning of parameters), we will tune our parameters to minimize the loss over the training data in the hope that the model may perform well on unseen test data.

To minimize the loss function, first note that if it is differentiable on the MLP output y,[63] then it is also differentiable on all the parameters of the neural network (weight matrices W_i of each layer) as the output vectors of each layer are differentiable over their corresponding weight matrices and composition over differentiable functions is differentiable.

As the MLP parameters are all differentiable for the loss function, a natural idea to minimize the loss function is to apply gradient descent algorithm (or its stochastic version) iteratively [Bottou, 2010].

As all the gradients are well defined, one can simply calculate each of them independently at every iteration. However, we shall see that many calculations are redundant and BackPropagation (BP) is an especially efficient algorithm to calculate the gradients of all the parameters of the model by exploiting this redundancy. We will use a slightly abusive notation $\frac{dv}{du}$ to denote derivatives, gradients or jacobians depending on the dimensions of v and u. For example, $\frac{d(f(y,y'))}{dW_i}$ is a gradient vector containing all the partial derivatives of the form $\frac{\partial(f(y,y'))}{\partial W_i[r,c]}$ where $W_i[r, c]$ is the element in row r and column c of W_i; $\frac{dy}{dW_i}$ is a jacobian matrix whose column j is the gradient of coordinate y_j over W_i: $\frac{dy_j}{dW_i}$.

To illustrate BP, suppose that we have an MLP of depth n. Let's note the intermediate output vector after the input x to be y_1, the output vector after y_1 to be y_2, ..., and $y_n = y$ (i.e. we note $f_1(x) = y_1, f_2(f_1(x)) = y_2$, etc.) Then, one can

[62] The one-hot vector y' for the true label is a vector filled with a 1 for the true label and 0s for the other labels.

[63] This is the case for the cross-entropy loss, the log loss and many other losses used in machine learning.

expand the gradients of W_i according to the chain rule: $\frac{d(f(y,y'))}{d(W_i)} = \frac{d(f(y,y'))}{dy_i}\frac{dy_i}{d(W_i)}$. The first term $\frac{d(f(y,y'))}{dy_i}$ can be further expanded using the chain rule across outputs of different layers in the neural network: $\frac{d(f(y,y'))}{dy_i} = \frac{d(f(y,y'))}{dy}\frac{dy}{dy_{n-1}} \dots \frac{dy_{i+1}}{dy_i}$.

One useful remark is that $\frac{d(f(y,y'))}{dy_i}$ also equals $\frac{d(f(y,y'))}{dy_{i+1}}\frac{dy_{i+1}}{dy_i}$ and the quantity $\frac{d(f(y,y'))}{dy_{i+1}}$ is needed to calculate the gradient on W_{i+1} as $\frac{d(f(y,y'))}{d(W_{i+1})} = \frac{d(f(y,y'))}{dy_{i+1}}\frac{dy_{i+1}}{d(W_{i+1})}$. Thus, to save computation time to calculate the gradients on layer i, one can use the quantity $\frac{d(f(y,y'))}{dy_{i+1}}$ calculated at the layer $i+1$.

This idea gives the BP algorithm where one starts by calculating the gradient on the output (y_n) of last layer, then descends layer by layer to calculate the gradients on the output vectors (y_i) at each layer, each time using the calculation performed by the previous step; once the gradient over an output vector (y_i) is calculated, it can be used to calculate the gradient over the parameters (W_i) of that layer.

Remark One can also think of algorithms exploiting the calculation redundancy by using ‘forward propagation’. The fact that backpropagation is used when training MLPs for label predictions is because the dimension of $f(y, y')$ (which is 1) is much smaller than the dimension of $W_i \in R^{m_2 \times m_1}$ (which is $m_2 \times m_1$). In this case, it can be shown that backpropagation is more efficient than forward propagation [Baydin *et al.*, 2015].

3.2 Recurrent Neural Networks

3.2.1 Recurrent Neural Network model

To introduce recurrent neural networks (RNNs), let’s consider the task of language modeling. A language model assigns a probability distribution P over all sequences of words (i.e sentences) $w_1, \dots, w_m$.

To estimate the distribution P, a natural way to tackle the problem is to use the Markov assumption. More precisely, for all words w_i in the sentence $w_1, \dots, w_m$, we suppose that $P(w_i|w_1, \dots, w_{i-1}) = P(w_i|w_{i-n}, \dots, w_{i-1})$; in other words, the probability of the current word depends only on the n previous words which we call the ‘n-gram prefix’, an assumption that underlies the traditional n-gram models.

However, natural languages often exhibit long range dependencies over the words, which motivates the construction of a model being able to handle a long prefix (i.e a ‘large n-gram prefix’); traditional n-gram models cannot handle large n well due to data sparsity: larger n implies less data to estimate the conditional probabilities. For example, suppose that we have a language of vocabulary size $|V|$. For traditional n-gram models which estimate each $P(w_i|w_{i-n}, \ldots, w_{i-1})$ by counting occurrences in the dataset, if we suppose a model with prefix length 0, we just need to estimate the probability of occurrence for each word, so there are $|V|$ parameters to estimate; if we suppose a model with prefix length n, there are $|V|^n$ variations for the prefix; as each prefix can be followed by $|V|$ possible words, there are $|V|^{n+1}$ parameters to estimate which is too difficult for a large n as we have only a corpus of limited length.

RNNs are models that can in theory handle a prefix of any length without suffering such data sparsity problems; the main idea is to use *parameter sharing* to pass both long range and short range information. In particular, RNNs do not use additional parameters for handling long prefixes inside the model; the number of parameters to estimate for an RNN no more depends on its specified prefix length.

Parameter sharing implies that when an RNN traverses a prefix, at each word (or each step) it produces another state (hidden vector) using the same function (i.e f in the following equation) applying to the current word and the current state, giving the following recurrence over states:

$$h^t = f(h^{t-1}, x^t, \theta),$$

where h^t is the state (hidden vector) at time t; x^t is the input (input word w_t in our language modeling example) received at time t and θ denotes the parameters of our RNN.

To further specify the model, the following equation provides a simple parametrization for the function f:

$$h^t = tanh(Wh^{t-1} + Ux^t).$$

The above parametrization uses an input-to-hidden weight matrix U and a hidden-to-hidden weight matrix W.

To make the RNN capable of predicting a sequence of discrete words, one can add a hidden-to-output weight matrix V and then use the softmax function to predict a probability distribution over all the vocabulary $\hat{y}^t = softmax(Vh^t)$[64]. Fig. 3.1 illustrates this RNN architecture.

The recurrence in RNNs gives the model the theoretical power of being universal in the sense that any function computable by a Turing machine can be computed by such a recurrent neural network of a finite size. The proof [Siegelmann and Sontag, 1991] relies on using a theoretical RNN to simulate an unbounded stack

[64] Note the difference between $\hat{y}^t$ and y^t: $\hat{y}^t$ is the output of the RNN while y^t in Fig. 3.1 is the target value at t.

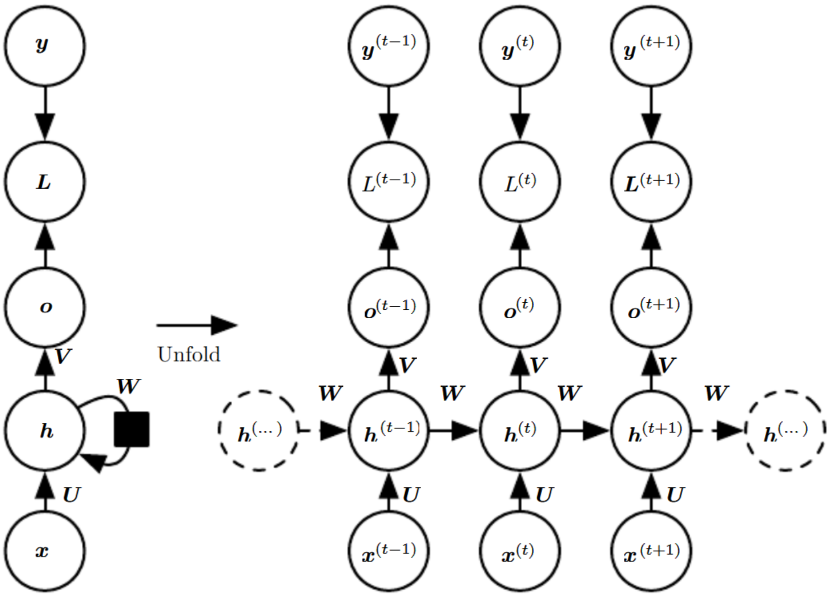


Figure 3.1: RNN computational graph that maps an input sequence x to a corresponding sequence of output o values. L stands for the loss computation, which computes $\hat{y} = softmax(o)$ before comparing this with target y. The RNN has input-to-hidden connections parametrized by a weight matrix U, hidden-to-hidden recurrent connections parametrized by a weight matrix W, and hidden-to-output connections parametrized by a weight matrix V. On the left we draw the RNN with recurrent connections that we unfold on the right. Figure taken from [Goodfellow *et al.*, 2016].

by representing its activations and weights with rational numbers of unbounded precision. In practice, implementations use floating-point representations of limited precision and therefore are not Turing-complete.

Language modeling application We can now describe how to use a trained RNN to generate texts with a probability score. For the input at each step i, we let $x_i = \tilde{y}_{i-1}$[65] in this case where $\tilde{y}_{i-1}$ is sampled from the distribution $\hat{y}^{i-1}$ calculated by the RNN. At time t, the model calculates a hidden state h_t based on the prefix $\tilde{y}_1, \ldots, \tilde{y}_{t-1}$ it has generated itself;[66] the predicted probability distribution for the current word at step t can then be calculated using the hidden state h_t: $\hat{y}^t = softmax(Vh^t)$ according to our model. One can sample from this distribution to choose a word $\tilde{y}^t$ and continue generating texts using the same mechanism. When a sentence is finished generating, the associated score is the product of the sampling probabilities $\tilde{y}^i$ for all i in the sentence.

[65] The beginning of the text x_1 can also be handled by the same RNN but need some special treatment [Goodfellow *et al.*, 2016].

[66] More precisely, the prefix is sampled from the probability distribution that the RNN predicts at each time step.

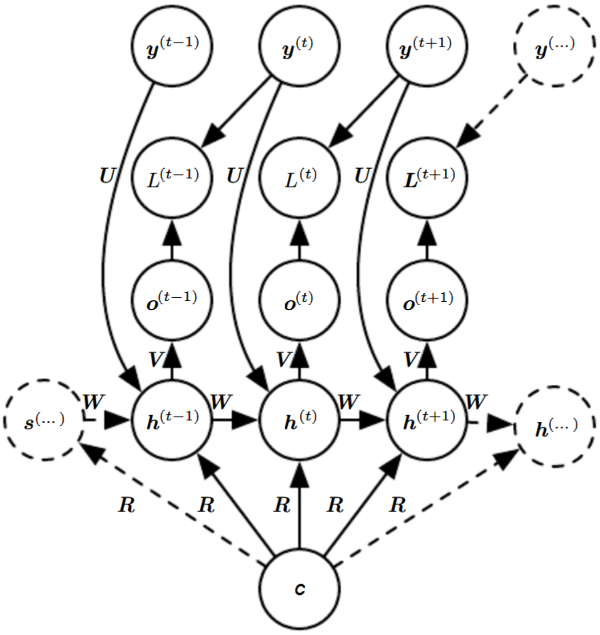


Figure 3.2: An RNN that generates a distribution over sequences Y conditioned on a fixed-length vector input c. Figure taken from [Goodfellow *et al.*, 2016].

Other NLP tasks as conditional language modeling Many NLP tasks can be seen as generating text based on some conditions. For example, machine translation can be seen as generating a text in the target language based on a sentence in the source language; natural language generation can be seen as generating a target text based on some meaning representation. We describe in this paragraph how to use one particular RNN architecture to generate a text based on some condition.

Our architecture supposes first that the condition is modeled by a real-valued vector c. One can then integrate c into the recurrence model:

$$h^t = f(h^{t-1}, x^t, c, \theta).$$

Fig. 3.2 illustrates the architecture; the interaction between the input vector c and each hidden unit vector h_t is parametrized by a newly introduced weight matrix R that was absent from our previous model. The product Rc is added as an additional input to the hidden units at every time step allowing the RNN to take into account the condition information. The following equation shows a parametrization of the model:

$$h^t = tanh(Wh^{t-1} + Ux^t + Rc).$$

3.2.2 Training for RNNs

We now discuss the training (learning of parameters) for RNNs. For a guiding example, we will use the RNN architecture shown in Fig. 3.1 to address the language modeling task; other 'conditional language modeling' tasks can be trained using similar algorithms. Given a sentence $w_1, \ldots, w_m$, we have according to the probability chain rule:

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}),$$

which decomposes the probability over sequences into a product of probabilities over words (or equivalently, sum of probabilities in log). Using this decomposition, instead of minimizing the loss over the whole sequence, one can minimize the loss over the prediction of each word. More precisely, for each step i, our model takes as input the prefix $y^1, \ldots, y^{i-1}$ given in the training corpus and tries to predict the target word y^i.[67]

The prediction of each word is then a multiclass classification task that we have discussed in the section about MLPs; similar to MLPs, the training over RNN parameters also uses the idea of BackPropagation (BP) with some modifications. To calculate the gradient of parameters for the loss, one can unfold n times the RNN the way we show on the right of Fig. 3.1.[68] After the unfolding procedure, the RNN looks exactly like an MLP of n+1 layers with the difference that 1) each layer receives an extra input word (i.e. w_i's)[69] 2) RNN parameters (e.g the hidden layer transition function parameters W) are shared across all layers. To tackle the problem of shared parameters, one needs to apply a generalized BP (see section 6.5.6 in [Goodfellow *et al.*, 2016]) to the MLP unfolded from the RNN .

The BP algorithm for training RNNs is commonly called BPTT for BackPropagation Through Time as the algorithm involves unfolding the RNN by several previous steps and calculating the gradients over those steps.

[67] When minimizing loss over words, the model always gets a 'correct' prefix $y^1, \ldots, y^{i-1}$ during training which is no more the case at test time, which creates unwanted asymmetry between training and test; this is a general problem for 'teacher forcing' training techniques and we refer interested readers on this topic to section 10.2.1 in [Goodfellow *et al.*, 2016].

[68] n can be regarded as a hyperparameter of the model specifying the number of steps back in RNN that one wants to apply Backpropagation.

[69] One can note that the extra input does not complicate the gradient calculation.

3.3 Recurrent Neural Network Variants

3.3.1 Vanishing/exploding gradient problems

The RNNs are designed to be expressive models that are able to model long range dependencies. However, a problem of "exploding or vanishing gradients" [Hochreiter, 1991] appears when using BPTT to train RNNs. More precisely, when a gradient backpropagates to the nth previous layer through unfolding and when n is large enough, the norm of the gradient can be either too big (exploding) or too small (vanishing); the exploding gradients will harm the model performance by actually ignoring previous training steps and the vanishing gradients imply that the model does not take into account long range dependencies during training. In this subsection, we will briefly describe the problem.

If we ignore the $tanh$ activation function in the equation $h^t = tanh(Wh^{t-1}+Ux^t)$, the recurrence between the hidden states resembles matrix multiplication ($h^t = Wh^{t-1}$) with some 'input perturbations'. Suppose that the matrix W admits an eigendecomposition, so $W = Q\Sigma Q^{-1}$ where Σ is a diagonal matrix filled with eigenvalues and Q is invertible. Then running the RNN n steps gives:

$$h_n = Q\Sigma^n Q^{-1} h_0.$$

For the eigenvalues larger than 1, the power of n will make those values explode; in these circumstances, any change of h_0 in the direction given by the associated eigenvector will be much amplified in h_n, giving an *exploding gradient*. On the other hand, for eigenvalues smaller than 1, any change of h_0 in the corresponding eigenvector direction will only cause a small change on h_n, giving a *vanishing gradient*.

Now let's examine the effect of adding a non-linear function such as $tanh$ after each matrix application W. The gradient of $tanh$ is bounded by 1 and goes to zero at infinity. In fact, the gradient goes to zero quickly and the norm is less than 0.5 in the interval $[-\infty, 1]$ and the interval $[1, \infty]$. Thus, adding this non-linearity will decrease (sometimes significantly) the gradient norm compared to the simplified case discussed above. As a consequence, the gradient backpropagated over a long range is now more likely to vanish than to explode after adding the non-linearity application.

3.3.2 Leaky units

There are several ways to make changes over the simple RNN model in Fig. 3.1 to take into account long range dependencies (by making sure that the gradient is able to backpropagate through a long range). Section 10.9 in [Goodfellow *et al.*, 2016] gives a review of methods proposed in the literature. We here focus on one particular solution: *Leaky units*; introducing leaky units helps us gain some insights on more complex RNN design such as LSTM (Long Short Term Memory) [Hochreiter and Schmidhuber, 1997] that we will discuss later in this section.

Suppose that apart from the recurrence on hidden states, we have another state c called "memory state" (or leaky unit) which also obeys some recurrence:

$$c^t = \alpha c^{t-1} + (1-\alpha)v^t,$$

where the parameter α is between 0 and fixed prior to training, and v is some extra input. In the above equation, when α is near one, the current memory state will be influenced by another memory state from many previous steps back, and we say that the memory state remembers information from the past for a long time; on the other hand, when α is near 0, the information from the past is rapidly discarded.

[Mozer, 1992] were the first to the best of our knowledge to propose the above memory state and integrate it into an RNN where one replaces v by the hidden state of an RNN:

$$\begin{aligned} c^t &= \alpha c^{t-1} + (1-\alpha)h^t \\ h^t &= tanh(Wh^{t-1} + Ux^t). \end{aligned}$$

At test time, c can used for predictions of sequences. One notices that c^t is modeled as a compromise between c^{t-1} and h: on the one hand, when α is near 1, information can be kept through many steps in the memory state c, which is useful when the current prediction depends on the long range information; on the other hand, when α is near 0, more 'recent' information in h is used for predictions. [Mozer, 1992] shows that with well chosen α, the proposed architecture is able to better capture long range information compared to an RNN without memory states.

3.3.3 Some remarks on LSTMs

Leaky units allow an RNN to accumulate information (such as evidence for a particular feature) over a long range. However, once that information has been used, it might be also useful for the RNN to *forget* that information. For example, if a sequence is made of independent subsequences, then we may want to accumulate evidence inside each subsequence but forget the evidence at the end of each subsequence; if we choose to implement this mechanism with an RNN with leaky units, it means that we want to make α be near 1 inside each subsequence, but almost drop to near 0 when the RNN begins processing another subsequence. LSTMs [Hochreiter and Schmidhuber, 1997] are RNNs that incorporate the idea of using memory cells that adapt the coefficient α according to the context. Below are the equations that describe an LSTM:

$$\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1}) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1}) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1}) \\
c_t &= f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + U_c h_{t-1}) \\
h_t &= o_t \circ tanh(c_t)
\end{aligned}$$

In the equations above, σ is the sigmoid function which is applied to its input in an elementwise way; $\circ$ stands for elementwise multiplication; as in other sections of the chapter, we omit the bias term in the equations for clarity. $x_1, \ldots, x_n$ is the input sequence; the LSTM outputs a sequence of hidden vectors $h_1, \ldots, h_n$. $W_f, W_i, W_o, W_c, U_f, U_i, U_o, U_c$ are parameter matrices to be learned.

The f_t, i_t, o_t, varying between 0 and 1, are called forget gate, input gate and output gate respectively. These gates control the information flowing into the memory state (c_t) and output state (h_t). For example, the update on the memory state c_t is written as $c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + U_c h_{t-1})$ where the update is influenced by the forget gate f_t and the input gate i_t. When f_t is near 1, most of the information in the previous memory state c_{t-1} will be copied to c_t; when f_t is near 0, the information in c_{t-1} will be mostly forgot. Similarly, when i_t is near 1, the recent information (captured by $tanh(W_c x_t + U_c h_{t-1})$) will be mostly copied contrary to the case where the information is mostly discarded with i_t near 0.

One should also notice the similarity between the recurrence in c_t and the leaky units. The forget gate f_t plays the role of α in the leaky units to control the flow of long range information. As $f_t = \sigma(W_f x_t + U_f h_{t-1})$ which is between 0 and 1 and depends on the input x_t, the LSTM has the possibility to learn how to choose different values of f_t depending on the input at different steps t.

The LSTMs have been shown to learn long-term dependencies more easily than the simple recurrent architectures and have been very successful in many NLP applications such as machine translation [Sutskever *et al.*, 2014], natural language generation [Wen *et al.*, 2015] and syntactic parsing [Vinyals *et al.*, 2014], to name just a few. We will use LSTMs as an important component for our semantic parsing systems in the later chapters.

Part II

Contributions

Chapter 4

Orthogonal Embeddings for the Semantic Parsing of Simple Queries

Contents

4.1 Introduction

In this chapter, we consider a 'simple' semantic parsing task by focusing on the Wikianswers [Fader *et al.*, 2013] dataset where the answers to the user query can be found in the Reverb [Fader *et al.*, 2011] knowledge base (KB). In the Wikianswers dataset, the underlying semantics is very simple: just one single triple (e_1, r, e_2) involving two entities (e_1, e_2) and one relation (r). However, the task remains challenging due to the large variety of lexicalizations for the same semantics.

We follow the approach of Bordes et al. [2014b] which learns the embeddings of words and KB elements. They model the semantics of natural language sentences and KB triples as the sum of the embeddings of the associated words and KB elements respectively. Then a scoring function involving the sentence embedding and the triple embedding is learned so that for a given natural language sentence the correct triple achieves a high score. We are particularly interested in this approach as it is robust to syntax errors that impact the performance of more traditional semantic parsers. For example, Reddy et al. [2014] propose to use a CCG parser [Clark and Curran, 2007] to construct their model and achieve good results; however, for the WebQuestions dataset [Berant *et al.*, 2013], the authors observe that 25% of the system failures are due to the fact that the sentence cannot be parsed by the CCG parser. The situation is worse for more simple grammars, Berant et al. [2013] manually write a grammar to parse the questions in the WebQuestions dataset, however, this simple grammar seems to be able to parse no more than half of the questions.[70]

In our work, we push the results of Bordes et .al [2014b] further when relying only on the KB to create training data[71]. Our contribution is to introduce a new orthogonality regularizer which distinguishes entities from relations. We also investigate the tradeoff captured by the orthogonality constraints. With a toy example in section 4.4, we show that if the predictions of entities and relations are independent of each other given the input question, orthogonal embeddings we propose allow to better learn to answer of these questions.

The orthogonality constraint in the context of question answering is new, although it has been successfully used in other contexts such as topic modelling [Yao *et al.*, 2014]. Like [Bordes *et al.*, 2014b], we use almost no linguistic features such as POS tagging, parsing, etc.

[70] In their experiments, authors report that if they let the grammar generate the 200 most probable parses given a sentence (beam size), only in 40% cases one of the parse may correspond to a parse whose execution gives the correct answer. (oracle accuracy)

[71] Bordes et .al [2014b] use predefined templates to create (q, a) pairs from KB; we adopt the same procedure in this work.

4.2 The ReVerb Question Answering Task

The ReVerb question answering task was first introduced in [Fader *et al.*, 2013] as follows. Given a large RDF KB and a natural language (NL) question whose answer is given by a triple contained in that KB, the task is to find a correct triple. For example, a correct answer to the NL question "What is the main language in Hong Kong ?" would be the KB triple *(cantonese.e, be-major-language-in.r, hong-kong.e)*. RDF triples are assertions of the form (e_1, r, e_2) where r is a binary relation from some vocabulary R and e_1, e_2 are entities from a vocabulary E.

The KB used is ReVerb[72], a publicly available set of 15 million extractions [Fader *et al.*, 2011] defined over a vocabulary of approximately 600K relations and 3M entities. The test set used for evaluation consists of 698 questions extracted from the website Wikianswers, many of which involve paraphrases.

4.3 Embedding model

Word embeddings are generally learned [Deerwester *et al.*, 1990; Mikolov *et al.*, 2013b; Lebret and Collobert, 2015; Faruqui *et al.*, 2014] such that words with similar context will naturally share similar embeddings as measured for instance by cosine similarity. The context are often words that co-occur in some fixed window with the word in question.

The word embeddings learned in [Bordes *et al.*, 2014b] encode a different context (arguably more suited for QA tasks) which are all the triple-answers co-occurred with the word as we shall see in subsection 4.3.1. While those embeddings are probably better for QA tasks, we find that they do not reflect all the prior knowledge we have for the task. Notably the embeddings for elements in answer triples do not encode the information whether it is an entity or a relation. We propose in subsection 4.3.4 to incorporate those prior knowledge using a novel regularizer — orthogonality regularizer which favors entity embeddings orthogonal to relation embeddings.

4.3.1 Scoring function

The model learns the embedding of each word and KB element by trying to score the correct answers highest.[73] Mathematically, let q be the query in natural language, and a be the answer-triple to align. Denote the total number of words as N_w and

[72] `http://reverb.cs.washington.edu`

[73] The model and training/inference procedures are similar to [Bordes *et al.*, 2014a] that we have discussed in Chapter 1 (1.3.2.1).

```
1. Sample a positive training pair $(q_i, a_i)$ from D.
2. Create a corrupted triple $a'_i$
3. If $S(q_i, a_i) - S(q_i, a'_i) < 0.1$ :
     make a stochastic gradient ascent on $S(q_i, a_i) - S(q_i, a'_i) - \lambda|E.R|$
4. Normalize the embedding vector
```

Algorithm 1: Training with orthogonality regularizer

the number of KB elements as N_{kb}. Then denote by $\phi(q) \in \{0,1\}^{N_w}$ the 1-hot representation indicating the presence or absence of words in the query. Similarly we denote the sparse representation on the KB side as $\psi(a)$. Let $M \in R^{d \times N_w}$ be the embedding matrix for words and $K \in R^{d \times N_{kb}}$ be the embedding matrix for the elements in the KB. d is the low dimension chosen by the user.

The embedding of the sentence is then calculated as $M\,\phi(q)$ and similarly the embedding of the answer-triple as $K\,\psi(a)$. We can score the matching of these embeddings:

$$S(q, a) = (M\,\phi(q))^\top (K\,\psi(a))$$

which is the dot product between the embedding of the sentence and the embedding of the triple. The model is introduced in [Bordes *et al.*, 2014b] and we use the same scoring function. Note that the model actually sums up each word embedding to form the embedding of the sentence.[74]

4.3.2 Inference

The inference procedure is straightforward. Given a question q and a set of possible answer triples noted $A(q)$, the model predicts the answer by returning the triple with the highest score:

$$a' = argmax_{a \in A(q)} S(q, a)$$

4.3.3 Training

Originally in [Bordes *et al.*, 2014b], given a question to be answered, training is performed by imposing a margin-constraint between the correct answer and negative ones. More precisely, note a' a negative answer to the question q (the correct answer to q being a). Then for each question answer pair, the system tries to maximize the following function by performing a gradient ascent step:

[74] We can see here that each word will encode information from triple answers if we use a gradient-like algorithm to maximize the scoring function. In fact, the gradient received by each word for a certain (q, a) pair will be exactly $K\,\psi(a)$.

$$min(\epsilon,\ S(q,a) - S(q,a'))$$

with ϵ the margin set to 0.1. In addition, the norms of columns in M and K are constrained to be inferior to 1. The training is done in a stochastic way by randomly selecting a question answer pair at each step. For each gradient step, the step size is calculated using Adagrad [Duchi *et al.*, 2011]. For a given (q,a) pair, the negative example is created by randomly replacing each element of (e_1, r, e_2) by another one in the KB with probability 2/3.

4.3.4 Enforcing Orthogonal Embeddings

In this chapter, we are especially interested in the additional assumptions we can make on the model in order to cope with data sparsity. Indeed, when the number of training data supporting the computation of embeddings is small, embedding models are brittle and can lead to disappointing results. We noticed that one important assumption that is not discussed in the basic approach is that the embedding space is the same for relations and entities. That approach has a tendency to learn similar embeddings for entities and relations, even if they have different meanings. Intuitively, we would like to balance that tendency by a "prior knowledge" preference towards choosing embeddings of entities and relations which are orthogonal to each other.

To justify this assumption, consider a simple case where the underlying semantics is (e,r) as in the sentence "John eats". We will use the same letters e and r to indicate both an entity or relation words and their corresponding embeddings. In [Bordes *et al.*, 2014b], the embedding of the sentence semantics is then calculated as $e + r$ for this very simple case.

Now suppose that $\forall e' \neq e,\ ||e - e'||_2 \geq \epsilon$ (i.e John is different from Mary with margin ϵ) and that the same kind of constraints also holds for relations. However, even when these constraints are satisfied, it is not guaranteed that $||e + r - e' - r'||_2 \geq \epsilon$, which means that the model may still get confused on the whole semantics even if each part is clear.

One obvious and linguistically plausible solution is to say that entities and relations lie in orthogonal spaces. Indeed, if relations and entities are orthogonal $(\forall r, e\ (r\ \perp\ e))$, then if two entities e, e' and two relations r, r' are distinct (i.e., $||e - e'||_2^2 \geq \epsilon$ and $||r - r'||_2^2 \geq \epsilon$), it follows that $||e + r - e' - r'||_2^2 = ||e - e'||_2^2 + |||r - r'||_2^2 \geq 2\epsilon$ by Pythagorean theorem. That is, two sentences involving distinct entities and/or relations will have different representations in the embedding space.

In real problems, however, posing a hard orthogonality constraint largely reduces the model's expressive power[75], so we decide to add it as a regularizer. More concretely, let the correct triple be (e_1, r, e_2) and the negative one be (e_1', r', e_2'). Consider that we are in a case not satisfying the margin constraint, then we will try to maximize the following regularized function $S(q,a) - S(q,a') - \lambda|E.R|$ with a gradient step. The regularizer $|E.R| = |e_1.r| + |e_2.r| + |e_1'.r'| + |e_2'.r'|$ is minimized when entities and relations live in orthogonal space. Algorithm 1 shows the whole training procedure; the regularization parameter λ is chosen via an automatically constructed development set for which we randomly selected 1/2000 of all the triples in the KB and generate associated questions. We discard these triples from training and choose the λ value based on the score on the development set. The λ value is by this means set to 0.01 with λ in $\{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$. Once the λ value is chosen, we retrain the whole system.

4.4 Experiments

4.4.1 Toy example

In this section, we illustrate the benefits of orthogonality via a toy example. We construct a KB containing 50 entities (E) and 50 relations (R) then generate all their cross products obtaining 2500 fact pairs. In consequence the entities and relations are independent.

For every $e_i \in E$, we suppose that there is a single word lexicalizing the entity noted "e_i". Similarly, we note the lexicalization of r_j "r_j". We separate these 2500 pairs into training (2450) and test (50). Notice that similar to Wikianswers, this toy dataset involves KB entities and relations whose type is known *a priori*.

The training corpus is built using one simple generation rule : $(e_i, r_j) \rightarrow$ "e_i r_j". Negative examples are created by replacing with probability 1/2 both entity and relation with another one. We embed all the words and KB symbols in a space of 20 dimensions. We compare the model [Bordes *et al.*, 2014b] with the model where we enforce E and R (and also their lexicalizations "E" and "R") to be orthogonal. This means that words or KB symbols in fact live in an embedding space of dimension 10.

At test time, for a given sentence "e_i r_j", a set of (e, r) pairs is ranked and we compute the proportion of cases where the first ranked pair is correct. Table 4.1 shows the results for both systems on two configurations: a configuration (Accuracy(1))

[75] Especially, if the embeddings are orthogonal between entities and relations, the knowledge of a given entity can not help to infer the relation and vice versa.

Model	Accuracy (1)	Accuracy (2)
Embedding	76%	54%
Ortho_Embedding	90%	68%

Table 4.1: Experimental results on toy example.

Method	Prec	Recall	F1	MAP
Embedding	0.60	0.60	0.60	0.34
Ortho_Embedding	0.63	0.63	0.63	0.36

Table 4.2: Performance for re-ranking question answer pairs of test set for different systems on Wikianswers

where the number of pairs to be ranked is 1250 and another (Accuracy(2)) with 2500 pairs.[76] In both cases, imposing the orthogonality constraint improves performance by a large margin.

4.4.2 Wikianswers

Wikianswers contains a set of possible triples for each question and we re-rank these triples to report our system's performance. This is the "re-ranking" setting used in [Bordes *et al.*, 2014b].

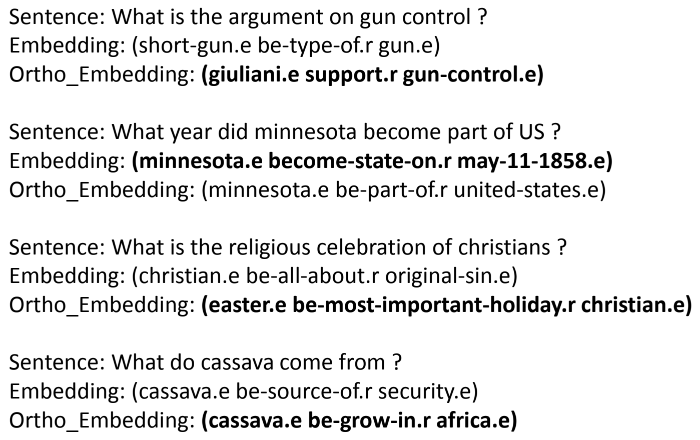
Table 4.2 shows that our technique improves the performance also on the larger, non-synthetic, Wikianswers dataset provided by Fader [2013] over the Bordes [2014b]'s method.[77] In addition, Fig. 4.1 shows some examples where the two systems differ and where the orthogonality regularized embeddings seem to better support the identification of similar relations. For instance, "is the argument on" is mapped to *support.r* rather than *be-type-of.r* and "is the religious celebration of" to *be-most-important-holiday.r* rather then *be-all-about.r.*

4.5 Related Work

Fader et al. [2013] present one of the first approaches for dealing with open domain question answering. The idea is to use certain templates (e.g. 'what is the r of e ?') where r is a relation and e is an entity lexicalized that will be by their KB mentions. A semantic parsing system can be constructed by mapping questions to these templates through exact string matching (e.g., 'what is the population of China'

[76] In Accuracy(1) configuration, we make sure that the correct answer is included.

[77] The scores are taken from [Bordes *et al.*, 2014b] for which we have reimplemented and confirmed the results.



Sentence: What is the argument on gun control ?
Embedding: (short-gun.e be-type-of.r gun.e)
Ortho_Embedding: **(giuliani.e support.r gun-control.e)**

Sentence: What year did minnesota become part of US ?
Embedding: **(minnesota.e become-state-on.r may-11-1858.e)**
Ortho_Embedding: (minnesota.e be-part-of.r united-states.e)

Sentence: What is the religious celebration of christians ?
Embedding: (christian.e be-all-about.r original-sin.e)
Ortho_Embedding: **(easter.e be-most-important-holiday.r christian.e)**

Sentence: What do cassava come from ?
Embedding: (cassava.e be-source-of.r security.e)
Ortho_Embedding: **(cassava.e be-grow-in.r africa.e)**

Figure 4.1: Some examples for which our system differs from [Bordes *et al.*, 2014b]. Gold standard answer triples are marked in bold.

matches the above template lexicalized by the relation ‘population’ and the entity ‘china’). However, the system will fail against minor changes of the above sentence (e.g. ‘what is the number of habitants in China’) making it of high precision but of low recall.

To improve the recall, the authors propose to use a paraphrase corpus. The sentences that match exactly to lexicalized templates in the paraphrase corpus are treated as annotated together with their paraphrases. Then an alignment is learned between sentences matching the templates and their paraphrases to enlarge the initial lexicons that consist of KB mentions. Finally, using this lexicon, multiple KB queries can be derived from a NL question. These queries are then ranked using a log-linear scoring function. The proposed method quadruples the recall with only 8% loss in precision compared to the above exact string matching approach using templates.

Bordes et al. [2014b] introduce a linguistically leaner IR-based approach which identifies the KB triple most similar to the input NL question by learning triple embeddings and question embeddings. We notice that Bordes’ [2014b] system performs relatively well (MAP score 0.34) on the Wikianswers dataset even without using the paraphrase corpus. This suggests that the embedding method successfully captures the similarity between NL questions and KB queries. Our work continues this direction by further separating relations with entities.

The idea of distinguishing entities and relations in question answering can also be found in [Yih *et al.*, 2014]. However, their work supposes that we can first cut the sentence into “entity part” and “relation part”, then calculate the matching score of each part. Our model does not need this cut and simply enforces the entity embeddings and relation embeddings (on the KB side) to be different.

More generally, our model proposes to incorporate KB information into embedding models. A popular approach along this line is ‘retrofitting’ word vectors [Faruqui *et al.*, 2014] so that words with same labels stay close in the embedding space. However, we doubt if the ‘retrofitting’ approach is useful in our case because the approach will tend to make all entities staying close together while we need to distinguish them for our QA task. Our approach does not favor all entity embeddings to stay closer instead.

Orthogonality or near orthogonality is a property which is desired in many embedding techniques. In random indexing [Sahlgren, 2005], a near orthogonality is ensured amongst the embeddings of different contexts used to learn the word vectors. In [Zanzotto and Dell’Arciprete, 2012], to approximate tree kernels in a distributed way, different subtree feature embeddings are also constructed to be near orthogonal.

This chapter gives yet another motivation for orthogonal embeddings for the special case where the semantics of a sentence is modeled as the sum of its associated word embeddings. As we have shown, in this case orthogonal word embeddings help to model their independence.

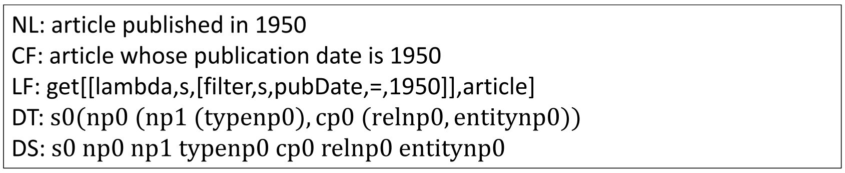
4.6 Conclusion

This chapter introduces an embedding model for question answering with orthogonality regularizer. We show that orthogonality helps to capture the differences between entities and relations and that it helps to improve performance on the Wikianswers dataset.

Chapter 5

Neural Semantic Parsing under Grammatical Prior Knowledge

Contents



NL: article published in 1950
CF: article whose publication date is 1950
LF: get[[lambda,s,[filter,s,pubDate,=,1950]],article]
DT: s0(np0 (np1 (typenp0), cp0 (relnp0, entitynp0))
DS: s0 np0 np1 typenp0 cp0 relnp0 entitynp0

Figure 5.1: Example of natural language utterance (NL) from the SPO dataset and associated representations considered in this work. CF: canonical form, LF: logical form, DT: derivation tree, DS: derivation sequence.

5.1 Introduction

In this chapter, we focus on a semantic parsing task where the NL question may be semantically complex, leading to a logical form query with a fair amount of compositionality.

Given the recently shown effectiveness of RNNs (Recurrent Neural Networks), in particular Long Short Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997], for performing sequence prediction in NLP applications such as machine translation [Sutskever *et al.*, 2014] and natural language generation [Wen *et al.*, 2015], we try to exploit similar techniques for our task. However we observe that, contrary to those applications which try to predict intrinsically *sequential* objects (texts), our task involves producing a *structured* object, namely a logical form that is tree-like by nature and also has to respect certain *a priori* constraints in order to be interpretable against the knowledge base.

In our case, building on the work "Building a Semantic Parser Overnight" [Wang *et al.*, 2015], which we will refer to as SPO, the LFs are generated by a grammar which is known *a priori*, and it is this grammar that makes explicit the structural constraints that have to be satisfied by the LFs. The SPO grammar, along with generating logical forms, generates so-called "canonical forms" (CF), which are direct textual realizations of the LF that, although they are not "natural" English, transparently convey the meaning of the LF (see Fig. 5.1 for an example).

Based on this grammar, we explore three different ways of representing the LF structure through a sequence of items. The first one (LF Prediction, or LFP), and simplest, consists in just linearizing the LF tree into a sequence of individual tokens; the second one (CFP) represents the LF through its associated CF, which is itself a sequence of words; and finally the third one (DSP) represents the LF through a derivation sequence (DS), namely the sequence of grammar rules that were chosen to produce this LF.

We then predict the LF via LSTM-based models that take as input the NL question and map it into one of the three sequentializations. In the three cases, the LSTM predictor cannot on its own ensure the grammaticality of the predicted sequence, so that some sequences do not lead to well-formed LFs. However, in the DSP case (in contrast to LFP and CFP), it is easy to integrate inside the LSTM predictor local constraints which guarantee that only grammatical sequences will be produced.

In summary, the contribution of this chapter is twofold. Firstly, we propose to use sequence prediction for semantic parsing. Our experimental results show some significant improvements over previous systems. Secondly, we propose to predict derivation sequences taking into account grammatical constraints and we show that the model performs better than sequence prediction models not exploiting this knowledge. These results are obtained without employing any reranking or linguistic features such as POS tags, edit distance, paraphrase features, etc., which makes the proposed methodology even more promising.

5.2 Background on SPO

The SPO paper [Wang *et al.*, 2015] proposes an approach for quickly developing semantic parsers for new knowledge bases and domains when no training data initially exists. In this approach, a small underlying grammar is used to generate canonical forms and pair them with logical forms. Crowdsourcing is then used to paraphrase each of these canonical forms into several natural utterances. The crowdsourcing thus creates a dataset (SPO dataset in the sequel) consisting of (NL, CF, LF) tuples where NL is a natural language question with CF and LF the canonical and the logical form associated with this question. In Chapter 1 (1.3.1.2), we have discussed this annotation approach which is arguably easier than annotating LFs based on NLs.

SPO learns a semantic parser on this dataset by firstly learning a log-linear similarity model based on a number of features (word matches, ppdb matches, matches between semantic types and POSs, etc.) between NLs and the corresponding (CF, LF) pairs. At decoding time, SPO parses a natural language utterance NL by searching among the derivations of the grammar for one for which the projected (CF, LF) is most similar to the NL based on this log-linear model. The search is based on a so-called "floating parser" [Pasupat and Liang, 2015], a modification of a standard chart-parser, which is able to guide the search based on the similarity features.

In contrast, our approach does not search among the derivations for the one that maximizes a match with the NL, but instead directly tries to predict a decision sequence that can be mapped to the LF.

The SPO system together with its dataset were released to the public[78] and we exploit this release in this chapter.

5.3 Neural Approach Integrating Grammatical Constraints

5.3.1 Grammars and Derivations

```
s0:  s(S) → np(S).
np0: np(get[CP,NP]) → np(NP), cp(CP).
np1: np(NP) → typenp(NP).
cp0: cp([lambda,s,[filter,s,RELNP,=,ENTNP]]) →
        [whose], relnp(RELNP), [is], entitynp(ENTNP).
...
typenp0:   typenp(article) → [article].
relnp0:    relnp(pubDate) → [publication, date]
entitynp0: entitynp(1950) → [1950].
...
```

Figure 5.2: Some general rules (top) and domain-specific rules (bottom) in DCG format.

The core grammatical resource released by SPO is a generic grammar connecting logical forms with canonical form realizations. They also provide seven domain-specific lexica that can be used in combination with the generic grammar to obtain domain-specific grammars which generate (LF, CF) pairs in each domain, in such a way that LFs can then be used to query the corresponding knowledge base. While SPO also released a set of Java-based parsers and generators for these grammars, for our own purposes we found it convenient to translate the grammars into the formalism of Definite Clause Grammars [Pereira and Warren, 1980], a classical unification-based extension of CFGs, which — through a standard Prolog interpreter such as Swipl[79] — provide direct support for jointly generating textual realizations and logical forms and also for parsing text into logical forms; we found this translation process to be rather straightforward and we were able to cover all of the SPO grammars.

Figure 5.2 lists a few DCG rules, general rules first, then lexical rules, for the SPO "publications" domain. Nonterminals are indicated in bold, terminals in italics. We provide each rule with a unique identifier (e.g. s0, np0, ...), which is obtained by

[78] https://github.com/percyliang/sempre

[79] http://www.swi-prolog.org/

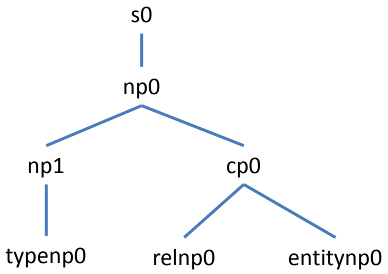


Figure 5.3: A derivation tree. Its leftmost derivation sequence is [s0, np0, np1, typenp0, cp0, relnp0, entitynp0].

typenp0	*article* article
relnp0	*publication date* pubDate
entitynp0	*1950* 1950
cp0	*whose publication date is 1950* [lambda,s,[filter,s,pubDate,=,1950]
np1	*article* article
np0	*article whose publication date is 1950* get[[lambda,s,[filter,s,pubDate,=,1950]],article]
s0	*article whose publication date is 1950* get[[lambda,s,[filter,s,pubDate,=,1950]],article]

Figure 5.4: Projection of the derivation tree nodes into (i) a canonical form and (ii) a logical form.

concatenating the name of its head nonterminal with a position number relative to the rules that may expand this nonterminal; we can then consider that the nonterminal (e.g. **np**) is the "type" of all its expanding rules (e.g. np0, np1, ...).

According to standard DCG notation, uppercase items `S, NP, CP, RELNP, ENTNP` denote unification variables that become instantiated during processing. In our case unificaion variables range over logical forms and each nonterminal has a single argument denoting a partially instantiated associated logical form. For instance, in the cp0 rule, **`relnp`** is associated with the logical form `RELNP`, **`entitynp`** with the logical form `ENTNP`, and the LHS nonterminal **cp** is then associated with the logical form `[lambda, s, [filter, s, RELNP, =, ENTNP]]`.[80]

In Figure 5.3, we display a *derivation tree* DT (or simply *derivation*) relative to this grammar, where each node is labelled with a rule identifier. This tree projects on the one hand onto the canonical form *article whose publication date is 1950*, on the

[80] This logical form is written here in DCG list notation; in the more "Lispian" format used by SPO, it would be written `(lambda s (filter s RELNP = ENTNP))`.

other hand onto the logical form `get[[lambda,s,[filter,s, pubDate,=,1950]],article]`.

Figure 5.4 shows how these projections are obtained by bottom-up composition. For instance, the textual projection of node cp0 is obtained from the textual representations of nodes relnp0 and entitynp0, according to the RHS of the rule cp0, while its logical form projection is obtained by instantiation of the variables `RELNP` and `ENTNP` respectively to the LFs associated with relnp0 and entitynp0.

Relative to these projections, one may note a fundamental difference between derivation trees DT and their projections CF and LF: while the well-formedness of DT can simply be assessed locally by checking that each node expansion is valid according to the grammar, there is in principle no such easy, local, checking possible for the canonical or the logical form; in fact, in order to check the validity of a proposed CF (resp. LF), one needs to *search* for *some* DT that projects onto this CF (resp LF). The first process, of course, is known as "parsing", the second process as "generation". While parsing has polynomial complexity for grammars with a context-free backbone such as the ones considered here, deciding whether a logical form is well-formed or not could in principle be undecidable for certain forms of LF composition.[81]

To be able to leverage sequence prediction models, we can associate with each derivation tree DT its leftmost *derivation sequence* DS, which corresponds to a preorder traversal of the tree. For the tree of Figure 5.3, this sequence is [s0, np0, np1, typenp0, cp0, relnp0, entitynp0]. When the grammar is known (in fact, as soon as the CFG core of the grammar is known), two properties of the DS hold (we omit the easy algorithms underlying these properties; they involve using a prefix of the DS for constructing a partial derivation tree in a top-down fashion):

1. knowing the DS uniquely identifies the derivation tree.
2. knowing a prefix of the DS (for instance [s0, np0, np1, typenp0]) completely determines the *type* of the next item (here, this type is **cp**).

The first property implies that if we are able to predict DS, we are also able to predict DT, and therefore also LF and CF. The second property implies that the sequential prediction of DS is strongly constrained by *a priori* knowledge of the underlying grammar: instead of having to select the next item among all the possible rules in the grammar, we only have to select among those rules that are headed by a

[81] The term 'projection' is borrowed from the notion of *bimorphism* in formal language theory [Shieber, 2014] and refers in particular to the fact that the overall logical form is constructed by bottom-up composition of logical forms associated with lower nodes in the derivation tree. In our DCG grammars, this composition actually involves more complex operations (such as "beta-reduction") than the simple copyings illustrated in the small excerpt of Fig. 5.2.

specific nonterminal. Under a simple condition on the grammar (namely that there are no “unproductive” rules, rules that can never produce an output[82]), following such constrained selection for the next rule guarantees that the derivation sequence will always lead to a valid derivation tree.

At this point, a theoretical observation should be made: *there is no finite-state mechanism on the sequence of rule-names that can control whether the next rule-name is valid or not.*[83] The relevance of that observation for us is that the RNNs that we use are basically finite-state devices (with a huge number of states, but still finite-state), and therefore we do not expect them in principle to be able to always produce valid derivation sequences *unless* they can exploit the underlying grammar for constraining the next choice.

5.3.2 Sequence prediction models

In all these models, we start from a natural language utterance NL and we predict a sequence of target items, according to a common sequence prediction architecture that will be described in section 5.3.3.

5.3.2.1 Predicting logical form (LFP model)

The most direct approach is to directly predict a linearization of the logical form from the NL, the input question. While an LF such as that of Figure 5.1 is really a structured object respecting certain implicit constraints (balanced parentheses, consistency of the variables bound by lambda expressions, and more generally, conformity with the underlying grammar), the linearization treats it simply as a sequence of tokens: `get [ [ lambda s [ filter s pubDate = 1950 ] ] article ]`. At training time, the LFP model only sees such sequences, and at test time, the next token in the target sequence is then predicted without taking into account any structural constraints. The training regime is the standard one attempting to minimize the cross-entropy of the model relative to the logical forms in the training set.

5.3.2.2 Predicting derivation sequence (DSP-X models)

Rather than predicting LF directly, we can choose to predict a derivation sequence DS, that is, a sequence of rule-names, and then project it onto LF. We consider three variants of this model.

[82] The general grammar ensures a good coverage of possible logical and canonical forms. However, when this general grammar is used in particular domains, some rules are not relevant any more (i.e. become "unproductive"), but these can be easily eliminated at compile time.

[83] This is easy to see by considering a CFG generating the non finite-state language $a^n b^n$.

DSP This basic derivation sequence prediction model is trained on pairs (NL, DS) with the standard training regime. At test time, it is possible for this model to predict ill-formed sequences, which do not correspond to grammatical derivation trees, and therefore do not project onto any logical form.

DSP-C This is a Constrained variant of DSP where we use the underlying grammar to constrain the next rule-name. We train this model exactly as the previous one, but at test time, when sampling the next rule-name inside the RNN, we reject any rule that is not a possible continuation.

DSP-CL This last model is also constrained, but uses a different training regime, with Constrained Loss. In the standard learning regime (used for the two previous models), the incremental loss when predicting the next item y_t of the sequence is computed as $-\log p(y_t)$, where $p(y_t)$ is the probability of y_t according to the RNN model, normalized (through the computation of a softmax) over *all* the potential values of y_t (namely, here, all the rules in the grammar). By contrast, in the CL learning regime, the incremental loss is computed as $-\log p'(y_t)$, where $p'(y_t)$ is normalized *only over the values of y_t that are possible continuations* once the grammar-induced constraints are taken into account, ignoring whatever weights the RNN predictor may (wrongly) believe should be put on impossible continuations. In other words, the DSP-CL model incorporates the prior knowledge about well-formed derivation sequences that we have thanks to the grammar. It computes the actual cross-entropy loss according to the underlying generative process of the model that is used once the constraints are taken into account.

5.3.2.3 Predicting canonical form (CFP model)

The last possibility we explore is to predict the sequence of words in the canonical form CF, and then use our grammar to parse this CF into its corresponding LF, which we then execute against the knowledge base.[84]

Table 5.1 provides length and vocabulary-size statistics for the LFP, DSP and CFP tasks.

[84] Although the general intention of SPO is to unambiguously reflect the logical form through the canonical form (which is the basis on which Turkers provide their paraphrases), we do encounter some cases where, although the CF is well-formed and therefore parsable by the grammar, several parses are actually possible, some of which do not correspond to queries for which the KB can return an answer. In these cases, we return the first parse whose logical form does return an answer. Such situations could be eliminated by refining the SPO grammar to a moderate extent, but we did not pursue this.

Target sequence	DS	CF	LF
Length	10.5	11.8	47.0
Vocabulary Size	106.0	55.8	59.9

Table 5.1: Characteristics of different target sequences.

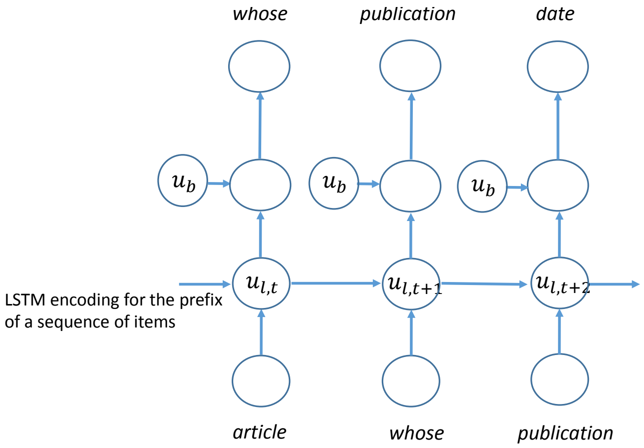


Figure 5.5: Our neural network model which is shared between all the systems where we illustrate its use for CFP. An MLP encodes the sentence in unigrams and bigrams and produces u_b. An LSTM encodes the prefix of the predicted sequence generating $u_{l,t}$ for each step t. The two representations are then fed into a final MLP to predict the next choice of the target sequence.

We see that, typically, for the different domains, DS is a shorter sequence than LF or CF, but its vocabulary size (i.e. number of rules) is larger than that of LF or CF. However DS is unique in allowing us to easily validate grammatical constraints. We also note that the CF is less lengthy than the LF, which uses a number of non "word-like" symbols such as parentheses, lambda variables, and the like.

5.3.3 Sequence prediction architecture

5.3.3.1 Neural network model

The goal of our neural network is to estimate the conditional probability $p(y_1, \ldots, y_{T'}|x_1, \ldots, x_T)$ where $(x_1, \ldots, x_T)$ is a natural language question and $(y_1, \ldots, y_{T'})$ is a target sequence (linearized LF, CF or derivation sequence). In all three cases, we use the same neural network model, which we explain in this subsection.

Suppose that the content of the NL is captured in a real-valued vector u_b, while the prefix of the target sequence up to time t is captured in another real-valued vector $u_{l,t}$. Now, the probability of the target sequence given the input question can be estimated as:

$$\begin{aligned} p(y_1, \dots y_{T'}|x_1, \dots, x_T) &= \prod_{t=1}^{T'} p(y_t|u_b, y_1, \dots y_{t-1}) \\ &= \prod_{t=1}^{T'} p(y_t|u_b, u_{l,t-1}) \end{aligned}$$

In all our systems, the u_b capturing the content of the NL is calculated from the concatenation of a vector u_1 reading the sentence based on unigrams and another vector u_2 reading the sentence based on bigrams. Mathematically, $u_1 = \tanh(W_1 v_1)$ where v_1 is the 1-hot unigram encoding of the NL and $u_2 = \tanh(W_2 v_2)$ where v_2 is its 1-hot bigram encoding. Then $u_b = \tanh(Wu)$, where u is the concatenation of u_1 and u_2. W_1, W_2 and W are among the parameters to be learnt. For regularization purposes, a dropout procedure [Srivastava *et al.*, 2014] is applied to u_1 and u_2.

The prefix of the target sequence up to time t is modelled with the vector $u_{l,t}$ generated by the latest hidden state of an LSTM [Hochreiter and Schmidhuber, 1997]; LSTM is appropriate here in order to capture the long distance dependencies inside the target sequence. The vector $u_{l,t}$ is then concatenated with u_b (forming u_{bl} in the equation below) before passing through a two-layer MLP (Multi-Layer Perceptron) for the final prediction:

$$p(y_{t+1}|u_{l,t}, u_b) = \mathrm{softmax}(W'_2 \tanh(W'_1 u_{bl}))$$

Using deep structures such as this MLP for RNN prediction has been shown to be beneficial in previous work [Pascanu *et al.*, 2013].

The overall network architecture is summarized in Figure 5.5. We train the whole network to minimize the cross entropy between the predicted sequence of items and the reference sequence.

This network architecture can easily support other representations for the input sentence than unigrams and bigrams, as long as they are real-valued vectors of fixed length. We can just concatenate them with u_1 and u_2 and generate u_b as previously. In fact, in initial experiments, we did concatenate an additional representation which reads the sentence through an LSTM, but the performance was not improved.

5.3.3.2 Decoding the target sequence

We implemented a uniform-cost search algorithm [Russell and Norvig, 2003] to decode the best decision sequence as the sequence with the highest probability. The algorithm finishes in a reasonable time for two reasons: 1) as indicated by Table 5.1, the vocabulary size of each domain is relatively small, and 2) we found that our model predicts relatively peaked distributions. Of course, it would also be easy to use a beam-search procedure, for situations where these conditions would not hold.

5.4 Experiments

5.4.1 Setup

We conduct our experiments on the SPO dataset. To test the overall performance of a semantic parser, the SPO dataset contains seven domains focusing on different linguistic phenomena such as multi-arity relations, sublexical compositionality, etc. The utterances in each domain are annotated both with logical forms (LFs) and canonical forms (CFs). The number of such utterances vary from 800 to 4000 depending on the domain. The size of training data is indeed small but as the target vocabulary is always in the domain, thus very small as well, it is actually possible to learn a reasonable semantic parser.

In the SPO dataset, the natural utterances were split randomly into 80%-20% for training and test, and we use the same sets. We perform an additional 80%-20% random split on the SPO training data and keep the 20% as development set to choose certain hyperparameters of our model. Once the hyperparameters are chosen, we retrain on the whole training data before testing.

For LFP experiments, we directly tokenize the LF, as explained earlier, and for CFP experiments we directly use the CF. For DSP experiments (DSP, DSP-C, DSP-CL) where our training data consist of (NL, DS) pairs, the derivation sequences are obtained by parsing each canonical form using the DCG grammar of section 5.3.

We compare our different systems to SPO. While we only use unigram and bigram features on the NL, SPO uses a number of features of different kinds: linguistic features on NL such as POS tags, lexical features computing the similarity between words in NL and words in CF, semantic features on types and denotations, and also features based on PPDB [Ganitkevitch *et al.*, 2013].

At test time, like SPO, we evaluate our system on the proportion of questions for which the system is able to find the correct answer in the knowledge base.

5.4.2 Implementation details

We choose the embedding vectors u_1 for unigrams and u_2 for bigrams to have 50 dimensions. The vector u_b representing the sentence content has 200 dimensions. The

	Basketball	Social	Publication	Blocks	Calendar	Housing	Restaurants	Avg
SPO	46.3	48.2	59.0	41.9	74.4	54.0	75.9	57.1
LFP	73.1	70.2	72.0	55.4	71.4	61.9	76.5	68.6
CFP	80.3	79.5	70.2	54.1	73.2	**63.5**	71.1	70.3
DSP	71.6	67.5	64.0	53.9	64.3	55.0	76.8	64.7
DSP-C	80.5	**80.0**	**75.8**	**55.6**	**75.0**	61.9	**80.1**	**72.7**
DSP-CL	**80.6**	77.6	70.2	53.1	**75.0**	59.3	74.4	70.0

Table 5.2: Test results over different domains on SPO dataset. The numbers reported correspond to the proportion of cases in which the predicted LF is interpretable against the KB *and* returns the correct answer. LFP = Logical Form Prediction, CFP = Canonical Form Prediction, DSP = Derivation Sequence Prediction, DSP-C = Derivation Sequence constrained using grammatical knowledge, DSP-CL = Derivation Sequence using a loss function constrained by grammatical knowledge.

word embedding layer has 100 dimensions, which is also the case of the hidden layer of the LSTM $u_{l,t}$. Thus u_{bl} which is the concatenation of u_b and $u_{l,t}$ has 300 dimensions and we fix the next layer to u_{bl} to have 100 dimensions. The model is implemented in Keras[85] on top of Theano [Bergstra *et al.*, 2010]. For all the experiments, we train our models using rmsprop [Tieleman and Hinton., 2012] as the backpropagation algorithm[86]. We use our development set to select the number of training epochs, the dropout factor over unigrams representation and the dropout factor over bigrams representation, by employing a grid search over these hyperparameters: epochs in {20, 40, 60}, unigrams dropout in {0.05, 0.1} and bigrams dropout in {0.1, 0.2, 0.3}.

5.4.3 Experimental results

5.4.3.1 Results on test data

Table 5.2 shows the test results of SPO and of our different systems over the seven domains.

It can be seen that all of our sequence-based systems are performing better than SPO by a large margin on these tests. When averaging over the seven domains, our 'worst' system DSP scores at 64.7% compared to SPO at 57.1%.

We note that these positive results hold despite the fact that DSP has the handicap that it may generate ungrammatical sequences relative to the underlying grammar, which do not lead to interpretable LFs. The LFP and CFP models, with higher performance than DSP, also may generate ungrammatical sequences.

[85] https://github.com/fchollet/keras

[86] All the hyperparameters of rmsprop as well as options for initializing the neural network are left at their default values in Keras.

	Basketball	Publication	Housing
LFP	6.6	3.7	1.6
CFP	1.8	1.9	2.2
DSP	9.5	11.8	5.8
DSP-C(L)	0.0	0.0	0.0

Table 5.3: Grammatical error rate of different systems on test.

The best results overall are obtained by the DSP-C system, which does take into account the grammatical constraints. This model performs not only considerably better than its DSP baseline (72.7% over 64.7%), but also better than the models LFP and CFP. Somewhat contrary to our expectations, the DSP-CL model, which exploits constraints not only during decoding, but also during training, performs somewhat worse than the DSP-C, which only exploits them during decoding.

We note that, for all the sequence based models, we strictly base our results on the performance of the *first* sequence predicted by the model. It would probably be possible to improve them further by reranking n-best sequence lists using a set of features similar to those used by SPO.

5.4.4 Analysis of results

5.4.4.1 Grammatical errors

We just observed that CFP and LFP perform well on test data although the sequences generated are not guaranteed to be grammatical. We analysed the percentage of grammatical errors made by these models and also by DSP for three domains, which we report in Table 5.3.[87]

The table shows that LFP and especially CFP make few grammatical errors while DSP makes them more frequently. For DSP-C and DSP-CL, the error rate is always 0 since by construction, the derivations must be well-formed. Note that as DSP is not constrained by prior knowledge about the grammar, the grammatical error rate can be high – even higher than CFP or LFP because DSP typically has to choose among more symbols, see Table 5.1.

5.4.4.2 Difference between DSP-C and DSP-CL

We observed that the DSP-CL model performs somewhat worse than DSP-C in our experiments. While we were a bit surprised by that behavior, given that the DSP-

[87] Our DCG permits to compute this error rate directly for canonical forms and derivation sequences. For logical forms, we made an estimation by executing them against the knowledge base and eliminating the cases where the errors are not due to the ungrammaticality of the logical form.

CL has strong theoretical motivations, let us note that the two models are quite different. To stress the difference, suppose that, for a certain prediction step, only two rules are considered as possible by the grammar, among the many rules of the grammar. Suppose that the LSTM gives probabilities 0.004 and 0.006 respectively to these two rules, the rest of the mass being on the ungrammatical rules. While the DSP-C model associates respective losses of $-\log 0.004, -\log 0.006$ with the two rules, the DSP-CL model normalizes the probabilites first, resulting in smaller losses $-\log 0.4, -\log 0.6$.

As we choose the best complete sequence during decoding, it means that DSP-C will be more likely to prefer to follow a different path in such a case, in order not to incur a loss of at least $-\log 0.006$. Intuitively, this means that DSP-C will prefer paths where the LSTM *on its own* gives small probability to ungrammatical choices, a property not shared by DSP-CL. However, a more complete understanding of the difference will need more investigation.

5.5 Related Work and Discussion

In recent work on developing semantic parsers for open-domain and domain-specific question answering, various methods have been proposed to handle the mismatch between natural language questions and knowledge base representations including, graph matching, paraphrasing and embeddings techniques.

Reddy et al. [2014] exploits a weak supervision signal to learn a mapping between the logical form associated by a CCG based semantic parser with the input question and the appropriate logical form in Freebase [Bollacker *et al.*, 2008].

Paraphrase-based approaches [Berant and Liang, 2014] generate variants of the input question using a simple hand-written grammar and then rank these using a paraphrase model. That is, in their setting, the logical form assigned to the input question is that of the generated sentence which is most similar to the input question.

Finally, Bordes et al. [2014b; 2014a] learn a similarity function between a natural language question and the knowledge base formula encoding its answer.

We depart from these approaches in that we learn a direct mapping between natural language questions and their corresponding logical form or equivalently, their corresponding derivation and canonical form. This simple, very direct approach to semantic parsing eschews the need for complex feature engineering and large external resources required by such paraphrase-based approaches as [Berant and Liang, 2014]. It is conceptually simpler than the two steps, graph matching approach proposed by Reddy et al. [2014]. And it can capture much more complex semantic representations than Bordes et al. [2014b; 2014a]'s embeddings based method.[88]

At a more abstract level, our approach differs from previous work in that it exploits the fact that logical forms are structured objects whose shape is determined by an underlying grammar. Using the power of RNN as sequence predictors, we learn to predict, from more or less explicit representations of this underlying grammar, equivalent but different representations of a sentence content namely, its canonical form, its logical form and its derivation sequence. The observation that one can incorporate the grammatical prior knowledge into a semantic parsing system is made by [Kate *et al.*, 2005a; Kate and Mooney, 2006; Wong and Mooney, 2006; Wong and Mooney, 2007]. Our contribution is to incorporate this type of prior knowledge into RNN-based models.

We observe that the best results are obtained by using the derivation sequence, when also exploiting the underlying grammatical constraints. However the results obtained by predicting directly the linearization of the logical form or canonical form are not far behind; we show that often, the predicted linearizations actually satisfy the underlying grammar. This observation can be related to the results obtained by Vinyals et al. [2014], who use an RNN-based model to map a sentence to the linearization of its parse tree,[89] and find that in most cases, the predicted sequence produces well-balanced parentheses. It would be interesting to see if our observation would be maintained for more complex LFs than the ones we tested on, where it might be more difficult for the RNN to predict not only the parentheses, but also the dependencies between several lambda variables inside the overall structure of the LF.

This chapter is closely related to [Dong and Lapata, 2016] which also proposes an RNN-based semantic parser. Fig. 5.6 illustrates the neural network architecture of [Dong and Lapata, 2016]; after encoding the sentence into a real-valued vector, to predict the LF *lambda $0 e (and (>(de- parture time $0) 1600:ti) (from $0 dallas:ci)*, the model first uses an LSTM to predict *lambda $0 e <n> </s>* (LSTM predictions at the top of Fig. 5.6) where *<n>* is a nonterminal to be expanded and *</s>* stands for the end. Then the LSTM is re-used at the next layer to expand the nonterminals in the previous layer (e.g. *<n>* expanded to *and <n> <n> </s>*). The model iterates until there is no more nonterminal to expand. There are two main differences

[88] In [Bordes *et al.*, 2014b; Bordes *et al.*, 2014a], the logical forms denoting the question answers involve only few RDF triples consisting of a subject, a property and an object i.e., a binary relation and its arguments.

[89] Note a crucial difference with our approach. While in their case the underlying ("syntactic") grammar is only partially and implicitly represented by a set of parse annotations, in our case the explicit ("semantic") grammar is known *a priori* and can be exploited as such.

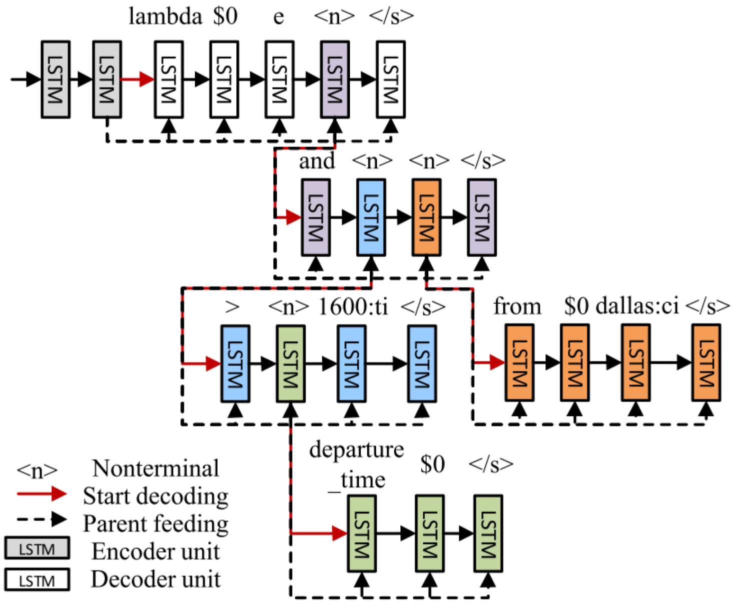


Figure 5.6: Neural network architecture of [Dong and Lapata, 2016] for predicting LFs.

compared to our approach for the decoding strategy:

- The model learns to predict LFs while our DSP-C(L) models learn to predict derivation sequences. Using their way of predicting LFs can ensure well-balanced parentheses as each layer predicts an LF with well-balanced parentheses; however, their model may fail to capture other grammatical constraints thus the produced LF is not guaranteed to be grammatically correct. Our DSP-C(L) models on the other hand always produce grammatically correct LFs.

- When predicting a certain LF token, the model [Dong and Lapata, 2016] is always conditioned on the hidden state of the nonterminal that the model is expanding (e.g. in Fig. 5.6, every prediction on the second layer is conditioned on the hidden state of $<n>$ in the first layer), a mechanism that the authors called 'parent-feeding'. Explicitly conditioning on the parent node representation can help the performance, however, this mechanism is missing in our proposed model. Implementing a 'parent-feeding' mechanism for our DSP-C(L) models doesn't pose theoretical difficulties. For example, consider the DS prediction in Fig. 5.3, an example of 'parent-feeding' for the DSP-C(L) will consist in condition also on np0 when trying to predict cp0.

[Yin and Neubig, 2017] study the code generation problem (i.e generating machine readable codes based on a natural language description) and take the best of both our model and the model of [Dong and Lapata, 2016]. Similar to our work, their neural code generator tries to predict the AST (abstract syntactic tree) to fully capture all the grammatical constraints; similar to the work of [Dong and Lapata, 2016], their neural network is equipped with the 'parent-feeding' mechanism and the attention mechanism [Bahdanau *et al.*, 2015]. They have shown in their work that both are important to achieve state-of-the-art result for code generation.

5.6 Conclusion

In this chapter, we propose a sequence-based approach for the task of semantic parsing. We encode the target logical form, a structured object, through three types of sequences: direct linearization of the logical form, canonical form, derivation sequence in an underlying grammar. In all cases, we obtain competitive results with previously reported experiments. The most effective model is one using derivation sequences and taking into account the grammatical constraints.

In order to encode the underlying derivation tree, we chose to use a leftmost derivation sequence. But there are other possible choices that might make the encoding even more easily learnable by the LSTM, and we would like to explore those in future work.

Chapter 6

Automata as Additional, Modular, Prior Knowledge Sources

Contents

6.1 Introduction

In chapter 5, we propose to use RNNs for semantic parsing; notably, we propose to integrate grammatical prior knowledge into RNN predictors by predicting derivation sequences (DS). In this chapter, we exploit other sources of prior knowledge, combine them with the grammatical one before integrating the result into RNN predictors.

To identify sources of prior knowledge for semantic parsing, we are inspired by traditional semantic parsing systems [Kwiatkowski *et al.*, 2013; Berant and Liang, 2014; Reddy *et al.*, 2014; Pasupat and Liang, 2015] which exploit rich prior knowledge in the form of features and grammars. In this chapter, we propose to integrate this type of prior knowledge into RNNs and particularly LSTMs [Hochreiter and Schmidhuber, 1997] based deep learning models [Jia and Liang, 2016; Dong and Lapata, 2016; Xiao *et al.*, 2016b; Neelakantan *et al.*, 2016] for semantic parsing. Deep learning models have shown to be successful for many NLP tasks; however, we notice that those models can be further enhanced by incorporating prior knowledge and this paper builds on two recent attempts to *combine* prior knowledge with neural networks in an NLP context.

In the context of Natural Language Generation (NLG), Goyal et al. [2016] describe an RNN model that generates sentences character-by-character, conditional on a semantic input. They use a form of prior knowledge, which they call a "background", to guide the RNN in producing string of characters which are (i) valid common English words or (ii) "named entities" (e.g. hotel names, addresses, phone numbers, ...) for which evidence can be found in the semantic input.

In the context of Semantic Parsing, we propose in chapter 5 to use an RNN-based model to predict derivation sequences (DS) that are derivation steps relative to an *a priori* given underlying grammar. The grammar is used to incrementally filter out those derivation steps that may lead to non-interpretable LFs, something which is difficult for the RNN to learn on its own.

While the "background" used by [Goyal *et al.*, 2016] is partially based on its actual semantic input, the prior employed by chapter 5 only exploits knowledge about output well-formedness. In both cases (NLG and Semantic Parsing) however, the output depends on the input; in semantic parsing, if the input question contains the string 'Barack Obama', it is highly likely that the LF of that question involves the entity Barack Obama and therefore, that the rule expanding to "barack obama" is present in the output derivation sequence.

This chapter can be seen as an extension of the semantic parsing approach proposed in chapter 5 using ideas from [Goyal *et al.*, 2016], where we use a background prior that combines the grammaticality constraints with certain types of prior beliefs that we can extract from the NL question.

Combining different sources of prior knowledge, which can also be seen as combining different factors in a graphical model, is a hard problem. In general, to compute the exact combination (with even two factors), one does not have other solutions than to go through an exhaustive enumeration of both factors and multiplying each pair of factors. Our proposed solution to this problem is to implement our input-dependent background through weighted finite-state automata (WFSAs), which we then *intersect* with a weighted context free grammar (WCFG) representing valid grammar derivations.

Intersecting a WFSA with a WCFG can be done through a dynamic programming procedure (thus efficient as it avoids exhaustive enumeration) closely related to chart-parsing.[90] The result of this intersection algorithm is a new WCFG, which can be normalized into a PCFG (Probabilistic CFG), which makes explicit the conditional probabilities for the different ways in which a given derivation sequence can be continued.[91]

The obtained PCFG is then used as a background, and when making its next local choice, the RNN has only to learn to "correct" the choices of the PCFG. In the cases where the background is close to the true distribution, the RNN will learn to predict a uniform distribution thus always referring to the background for such predictions.

This is in fact a desirable behaviour as the background may contain prior knowledge that the RNN is not able to learn based on data (e.g. prior knowledge on entities unseen in training) and the best behavior for the model in those cases is to refer to the background.

We test our new Background RNN semantic parser on an extended version of the SPO dataset [Wang *et al.*, 2015], which removes certain problematic aspects of the original dataset (that made the results too optimistic, as explained in section 6.4). By incorporating simple input-dependent prior knowledge via WFSAs, our model not only improves over its RNN baseline but also over the non-RNN system proposed in [Wang *et al.*, 2015] which involves much richer hand-crafted features.

[90] More on intersection algorithms can be found in chapter 2 (2.2) inside the thesis.

[91] While on first sight the whole procedure may appear somewhat involved, it has the crucial advantage that a global constraint, for instance the required appearance of a certain symbol at some unknown *future* point in the DS, has local consequences much earlier in the incremental process that the network is following.

6.2 Background on Grammars and Automata

6.2.1 The approach of Wang et al. (2015), original and new datasets

```
s0:  s(S) → np(S).
np0: np(get[CP,NP]) → np(NP), cp(CP).
np1: np(NP) → typenp(NP).
cp0: cp([lambda,s,[filter,s,RELNP,=,ENTNP]]) →
        [whose], relnp(RELNP), [is], entitynp(ENTNP).
...
typenp0:    typenp(article) → [article].
relnp0:     relnp(pubDate) → [publication, date]
entitynp0: entitynp(1950) → [1950].
...
```

Figure 6.1: Some general rules (top) and domain-specific rules (bottom) of the *Overnight* in DCG format.

[Wang *et al.*, 2015] (which we refer to as "SPO") proposes a novel way to build datasets for training a semantic parser without having to manually annotate natural language sentences with logical forms (LFs). First a grammar (of which we provide an extract in Fig. 6.1, in the format of Definite Clause Grammars [Pereira and Warren, 1980], reproduced from chapter 5) is used to generate LFs paired with conventional surface realizations called "canonical forms" (CFs). For example, the rules shown in Fig. 6.1 support the generation of the LF *get[[lambda,s,[filter,s,pubDate,=,1950]],article]* along with the CF "article whose publication date is 1950".

The CFs are not necessarily natural English but are supposed to be "semantically transparent" so that one can use crowdsourcing to paraphrase those CFs into natural utterances (NLs) e.g., *Articles published in 1950*. The resulting (NL, LF) pairs make up a dataset which can be used for learning semantic parsers.

After collecting all the paraphrases, the authors of SPO construct a dataset divided into training and test sets by performing a random 80%-20% split over all the (NL, LF) pairs. However, given the data collecting process, each LF tends to correspond to several (in general more than 5) paraphrases. In consequence, inside this original dataset, most of the LFs in the test set have already been seen in training, making the task close to a classification process and easier than it should be.

In addition, as pointed out by Jia and Liang [2016], the original dataset contains very few named entities. In this work, we therefore construct a new dataset called *Overnight+* fixing some of the above issues. More details on our proposed dataset can be found in subsection 6.4.1.

6.2.2 The approach of chapter 5

To learn the semantic parser, SPO first trains a log-linear model based on rich prior features dependent jointly on NL and the corresponding (LF, CF) pair. Then it searches for the derivation tree relative to the grammar for which the produced (LF, CF) pair has the highest score [Pasupat and Liang, 2015].

In contrast, in chapter 5 we propose to use RNN-based models to directly map the NL to its corresponding derivation sequence (DS), which are sequentialized representations of derivation trees in the grammar. Predicting DS provides an efficient sequentialization and makes it easy to guarantee the well-formedness of the predicted sequence. Chapter 5 shows that the model "Derivation Sequence Predictor with Constraints Loss" (DSP-CL) achieves good performance on the original *Overnight* dataset.

This chapter can be seen as extending DSP-CL by integrating some input-dependent prior knowledge into the RNN predictor, allowing it to improve its performance on the more challenging *Overnight+* dataset.

6.3 Symbolic Background Neural Model

6.3.1 Background priors on RNNs

[Goyal *et al.*, 2016], in the context of NLG, propose to modify the standard generative procedure of RNNs:

$$p_\theta(x_{t+1}|x_1, \ldots, x_t, C) = rnn_\theta(x_{t+1}|x_1, \ldots, x_t, C),$$

where C is the observed input context (that is, the input of the seq2seq model), $x_1, \ldots, x_t$ the current output prefix, rnn_θ the softmax output of the RNN parametrized by θ, and p_θ the probability distribution from which the next symbol x_{t+1} is sampled, with:

$$p_\theta(x_{t+1}|x_1, \ldots, x_t, C) \propto b(x_{t+1}|x_1, \ldots, x_t, C) \cdot rnn_\theta(x_{t+1}|x_1, \ldots, x_t, C),$$

where the *background* b is an arbitrary non-negative function over $C, x_1, \ldots, x_t, x_{t+1}$, which is used to incorporate prior knowledge about the generative process p_θ.[92] On one extreme, taking b to be uniform corresponds to the situation where no prior knowledge is available, and one is back to a standard RNN, with all the discriminat-

[92] See also [Dymetman and Xiao, 2016], for a more general presentation, of which the background-RNN can be seen as a special case.

ing effort falling on the *rnn* component and relying on whatever (possibly limited) training data is available in each context; on the other extreme, if the true process p is known, one may take $b = p$, and then the *rnn* component $rnn_\theta(x_{t+1}|x_1, \ldots, x_t, C)$ is only required to produce a close-to-uniform distribution over the target vocabulary, independently of $x_1, \ldots, x_t, C$, which is an easy task to learn. In practice, the interesting cases fall between these two extremes, with the background b incorporating some prior knowledge that the *rnn* component can leverage in order to more easily fit the training data. In the NLG application considered by [Goyal *et al.*, 2016], the output of the seq2seq model is a string of characters, and the background — implemented as a WFSA over characters — is used to guide this output: (i) towards the production of valid common English words, and (ii) towards the production of named entities (e.g. hotel names, addresses, ...) for which evidence can be found in the semantic input.

The approach of chapter 5 can be reformulated into such a "Background-RNN" (BRNN) framework. In that work, the underlying grammar G acts as a yes-no filter on the incremental proposals of the RNN, and this filtering process guarantees that the evolving DS prefix always remains valid relative to G. There:

$$p_\theta(x_{t+1}|x_1, \ldots, x_t, C) \propto b(x_{t+1}|x_1, \ldots, x_t) \cdot rnn_\theta(x_{t+1}|x_1, \ldots, x_t, C),$$

where $C = NL$ is the input question, the x_i's are rule-names, and b takes a value in $\{0, 1\}$, with $b(x_{t+1}|x_1, \ldots, x_t) = 1$ indicating that $x_1, \ldots, x_t, x_{t+1}$ is a valid DS prefix relative to G. With this mechanism in place, on the one hand the BRNN cannot produce "ungrammatical" (in the sense of being valid according to the grammar) prefixes, and on the other hand it can exploit this grammatical knowledge in order to ease the learning task for the *rnn* component, which is not responsible for detecting ungrammaticality on its own anymore.

6.3.2 WCFG background

While the (implicit) background of chapter 5 shown in (6.3.1) is a binary function that does not depend on the NL input, but only on hard grammaticality judgments, in this paper, we propose to use the more general formulation (6.3.1). Now b is soft rather than hard, and it does exploit the NL input.

More specifically, $b(x_{t+1}|x_1, \ldots, x_t, NL)$ is obtained in the following way. First, we use the original grammar G together with the input NL to determine a WCFG (weighted context-free grammar) GW_{NL} over derivation sequences of the original CFG (that is, the terminals of GW_{NL} are rule-names of the original G), as will

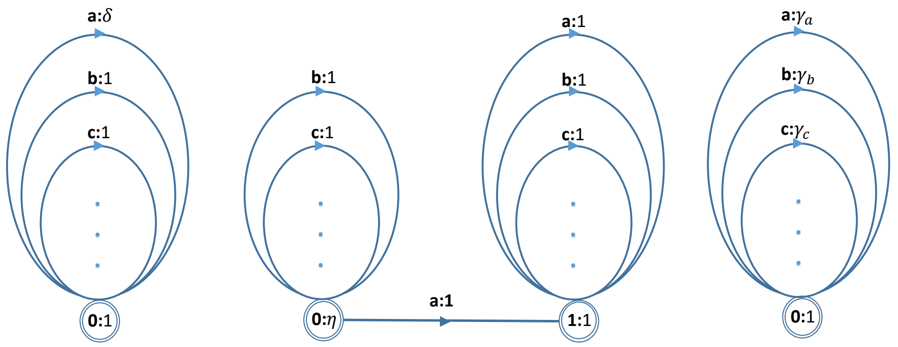


Figure 6.2: Three WFSA's for handling different types of prior information. Edge labels are written in the form *symbol : weight*. The initial state is **0**. Final states are indicated by a double circle and their exit weight is also indicated.

be explained below. Second, we compute $b(x_{t+1}|x_1, \ldots, x_t, NL)$ as the conditional probability relative to GW_{NL} of producing x_{t+1} in the context of the prefix $x_1, \ldots, x_t$.

Apart from our use of a much richer background than chapter 5, our overall training approach remains similar. Our training set consists in pairs of the form (NL, DS); the rnn_θ component of the BRNN (6.3.1) is an LSTM-based network in which the input encoding is a vector based on the unigrams and bigrams present in NL, and where the DS output is a sequence of rule-names from G; the output layer of this network is then combined additively with $\log b$ before a softmax is applied, resulting in the probability distribution $p_\theta(x_{t+1}|x_1, \ldots, x_t, NL)$; finally the incremental cross-entropy loss of the network $-\log p_\theta(\bar{x}_{t+1}|x_1, \ldots, x_t, NL)$ is back-propagated through the network (where $\bar{x}_{t+1}$ is the observation in the training data). Implementation details are provided in section 6.4.

6.3.2.1 Constructing the WCFG background, WFSA factors

As in the case of chapter 5, we still want our background to ensure grammaticality of the evolving derivation sequences, but in addition we wish it to reflect certain tendencies of these sequences that may depend on the NL input. By stating these tendencies through a real-weighted, rather than binary, background $b(x_{t+1}|x_1, \ldots, x_t, NL)$, we make it possible for the rnn component to bypass the background preferences in the presence of a training observation x_{t+1} that does not agree with them, through giving a high enough value to $rnn_\theta(x_{t+1}|x_1, \ldots, x_t, NL)$.

Our approach is then the following. We start by constructing a simple WCFG GW_0 that (1) enumerates exactly the set of all valid derivation sequences relative to the original G, and (2) gives equal weight $1/n_{NT}$ to each of the possible n_{NT}

expansions of each of its non-terminals NT. Thus GW_0 is actually a *Probabilistic* CFG (PCFG), that is, a WCFG that has the property that the sum of weights of the possible rules expanding a given nonterminal is 1. Thus GW_0 basically ensures that the strings of symbols it produces are valid DS's relative to G, but is otherwise non-committal concerning different ways of extending each prefix.

The second step consists in constructing, possibly based on the input NL, a small number of WFSA's (weighted FSA's) each of which represents a certain aspect of the prior knowledge we have about the likely output sequences. These automata will be considered as "factors" (in the sense of probabilistic graphical models) that will be intersected (in other words, multiplied) with GW_0, resulting in a final WCFG GW_{NL} which will then combine the different aspects and be used as our background.[93] In Fig. 6.2, we illustrate three possible such automata; here the output vocabulary consists of the symbols $a, b, c, \ldots$ (in our specific case, they will actually be DS symbols, but here we keep the description generic).

Let us first describe the automaton on the left. Here each symbol appears on an edge with weight 1, with the exception of the edge associated with a, which carries a weight $\delta \ll 1$; this automaton thus gives a "neutral" weight 1 to any symbol sequence that does not contain a, but a much smaller weight δ^k to one that contains $k \geq 1$ instances of a. Once intersected with GW_0, this automaton can be used to express the belief that given a certain input NL, a is unlikely to appear in the output.

The automaton in the middle expresses the opposite belief. Here the exit weight associated with the final (also initial) state $\mathbf{0}$ is $\eta \ll 1$. This automaton gives a weight 1 to any sequence that contains a, but a weight η to sequences that do not. Once intersected with GW_0, this automaton expresses the belief that given the input NL, a is likely to appear in the output.

The automaton on the right is a simple illustration of the kind of prior beliefs that could be expressed on output sequences, independently of the input. Here γ_x denotes the unigram probability of the output symbol x. In the context of semantic parsing, such automata on the output could be used to express certain forms of regularities on expected logical forms, such as, like here, unigram probabilities that are not handled by the grammar GW_0 (which is concerned only by well-formedness constraints), or more generally, observations about certain patterns that are likely or unlikely to occur in the logical forms (e.g. the unlikeliness of mixing basketball players with scientific authors), insofar as such regularities can be reasonably expressed in finite-

[93] Formally, intersecting a WCFG GW with a WFSA A consists in applying a Dynamic Programming algorithm that constructs a new WCFG $GW' = GW \cap A$. If the weight of a certain sequence $x_1, \ldots, x_n$ is ω_{GW} relative to GW (resp. ω_A relative to A), then its weight relative to GW' is $\omega_{GW} \cdot \omega_A$.

state terms.

Why automata? In order to be effective, the background b has to be able to provide the *rnn* component with useful information on the *next incremental step*, conditional on the already generated prefix. In addition, we would like the background to capitalize on different sources of information.

In connection with these desiderata, WCFG and WFSAs have the following remarkable properties: (1) the intersection of several WFSAs is a WFSA which can be efficiently computed, (2) the intersection of a WCFG with a WFSA is a WCFG which can be efficiently computed, (3) given a prefix, the conditional probability of the next symbol relative to a WCFG (resp. a WFSA) can be efficiently computed; here "efficiently computed" means through Dynamic Programming and in polynomial time [Nederhof and Satta, 2003]. These properties are conspicuously absent from most other generative devices. For instance it is far from obvious how to intersect two different RNNs to compute the conditional probability of the next symbol, given a common prefix: while a certain symbol may have a large probability relative to *both* RNNs, the later (global) consequences of choosing this next symbol may be largely incompatible between the two RNNs; in other words, the *local* combined conditional probability cannot be computed solely on the basis of the product of the two *local* conditional probabilities.

Implementation principles. The fact that one can intersect a WCFG with a WFSA to produce another WCFG is a generalization of the classical result [Bar-Hillel *et al.*, 1961] concerning the non-weighted case. The implementation we use is based on the Earley-inspired intersection algorithm of [Dyer, 2010], obtaining a certain WCFG, which we normalize into probabilistic form [Nederhof and Satta, 2003], finally obtaining a PCFG GW_{NL}. In order to compute the background $b(x_{t+1}|x_1, \ldots, x_t, NL)$ we then need to compute the conditional probability relative to GW_{NL} of producing the symbol x_{t+1} given the prefix $x_1, \ldots, x_t$. There are some special-purpose algorithms for doing that efficiently, for instance [Stolcke, 1994], but in this work we use again (unoptimally) the generic Earley intersection algorithm, taking advantage of the fact that the probability mass relative to GW_{NL} of the set of sequences starting with the prefix $x_1, \ldots, x_t, x_{t+1}$ can be obtained by intersecting GW_{NL} with the automaton generating the language of all sequences starting with this prefix.

6.4 Experiments

6.4.1 Setup

The original *Overnight* dataset is a valuable data resource for studying semantic parsing as the dataset contains various domains focusing on different linguistic phenomena; the utterances in each domain are annotated both with logical forms (LFs) and canonical forms (CFs). However, as Jia and Liang [2016] point out, this dataset has two main drawbacks: 1) it contains too few entities compared to real datasets, 2) most of the LFs in test are already seen during training. In consequence, the results achieved on this dataset by different systems [Wang *et al.*, 2015; Jia and Liang, 2016] are probably too optimistic.

To remedy these issues, we release an extended *Overnight+* dataset.[94] First, we group all the data and propose a new split. This split makes a 80%-20% random split on all the LFs and keeps the 20% LFs (together with their corresponding utterances) as test and the remaining 80% as training. Thus LFs seen in test are guaranteed to not be seen during training. For each domain, we also add new named entities into the knowledge base and create a new development set and test set containing those new named entities.[95] Depending on the domain, the number of annotated utterances vary from 800 to 4000 and we eliminate some erroneous annotations from the training set. All the reported experiments are conducted on *Overnight+*.

6.4.2 Implementations

For our BRNN, the background b is composed of a WCFG factor (GW_0 in subsection 6.3.2) and depending on the input, zero to several WFSA factors favoring the presence of certain entities. In the current implementation, we only employ automata that have the same topology as the automaton shown in the middle of Fig. 6.2 where the output vocabulary consists in rule names (e.g. s0, np1, ...) and where the weight η is chosen in [0, 0.0001, 0.01] based on the results obtained on the development set.

Currently, we detect only named entities and dates by using mostly exact string matching (e.g. if we detect 'alice' in the input, we construct an automaton to favor its presence in the LF), as well as a small amount of paraphrasing for dates (e.g we detect both 'jan 2' (as given by CFs) and 'january 2' as January 2nd). We use a library developed by Wilker Aziz[96] for performing the intersection between WFSA(s) and

[94] https://github.com/chunyangx/overnight_more

[95] We use a high-precision heuristic substituting the named entities in the input string with new ones under certain conditions.

[96] https://github.com/wilkeraziz/pcfg-sampling

a WCFG. The intersection algorithm results in a new WCFG, from which the background is computed through prefix-conditional probabilities as explained in section 6.3.

We adopt the same neural network architecture that we have described in section 5.3. We represent the NL semantics by a vector u_b calculated from the concatenation of a vector u_1 encoding the sentence at the level of unigrams and another vector u_2 at the level of bigrams. Dropout [Srivastava *et al.*, 2014] is applied to u_1 (0.1) and u_2 (0.3). We model the DS up to time t with the vector u_t generated by an LSTM [Hochreiter and Schmidhuber, 1997]; we concatenate u_t and u_b and pass the concatenated vector to a two-layer MLP for the final prediction. At test time, we use a uniform-cost search algorithm [Russell and Norvig, 2003] to produce the DS with the highest probability. All the models are trained for 30 epochs.

6.4.3 Experimental Results

Table 6.1 shows the results of different systems. The best average accuracy is achieved by our proposed system BDSP-CL. The system largely improves (48.8% over 34.5% in accuracy) over its RNN baseline DSP-CL which does not have input-dependent WFSA factors. Our system also improves largely over SPO (no-lex) i.e., SPO without "alignment features" (this system still has a rich feature set including string matching, entity recognition, POS tagging, denotation, etc).

In average, BDSP-CL also performs better than the system noted SPO* with the full set of features, but to a more moderate extent. However the results of this SPO* may be too optimistic: the so-called "alignment features" of SPO were obtained from a provided alignment file based on the original *Overnight* training dataset and not on the correct *Overnight+*, because we did not have access to easy means of recomputing this alignment file. The implication is that those features were calculated in a situation where most of the test LFs were already seen in training as explained in subsection 6.4.1, possibly unfairly helping SPO* on the *Overnight+* test set.

6.4.4 Result Analysis

Examples We look into predictions of BDSP-CL, DSP-CL, SPO (no-lex) trying to understand the pros and the cons of our proposed model. Table 6.2 shows some typical cases. Because our current implementation with automata is limited to taking into account prior knowledge on only named entities and dates, our BDSP-CL can miss some important indications compared to SPO (no-lex). For example, for the

New split	Basketball	Social	Publication	Calendar	Housing	Restaurants	Blocks	Avg
SPO (no-lex)	42.2	3.1	**33.1**	**38.8**	31.0	**65.4**	32.1	35.1
SPO*	*47.4*	*40.4*	*43.4*	*56.6*	*30.7*	*67.8*	*37.0*	*46.2*
DSP-CL	51.0	49.7	15.2	22.5	28.7	58.7	15.9	34.5
BDSP-CL	**63.0**	**57.3**	25.5	36.4	**60.7**	64.3	**34.7**	**48.8**

Table 6.1: Test results over all the domains on *Overnight+*. The numbers reported correspond to the proportion of cases in which the predicted LF is interpretable against the KB and returns the correct answer. DSP-CL is the model introduced in chapter 5 that guarantees the grammaticality of the produced DS. BDSP-CL is our model integrating various factors (e.g WCFG, WFSA) into the background. SPO (no-lex) is a feature-based system [Wang *et al.*, 2015] where we desactivate alignment features. SPO* is the full feature-based system but with unrealistic alignment features (explained in subsection 6.4.3) and thus should be seen as an upper bound of full SPO performance.

sentence	BDSP-CL	DSP-CL	SPO(no-lex)
‘what locations are the fewest meetings held’	‘meetings that has the least number of locations’	‘location that is location of more than 2 meeting’	*‘location that is location of the least number of meeting’*
‘which men are 180cm tall’	*‘person whose gender is male whose height is 180cm’*	‘person whose height is 180cm’	‘person whose height is at least 180cm’
‘what position is shaq oneal’	*‘position of player shaq oneal’*	‘position of player kobe bryant’	*‘position of player shaq oneal’*

Table 6.2: Some prediction examples of BDSP-CL, DSP-CL and SPO (no-lex). For readability, instead of showing the predicted LF, we show the equivalent CF. Correct predictions are noted in italics.

sentence ‘what locations are the fewest meetings held’, our model predicts a set of meetings while SPO (no-lex) detects through its features that the question asks about locations; our model seems better at discovering regularities in the data for which SPO (no-lex) does not have predefined features. For example, for the sentence ‘which men are 180cm tall’, our model successfully detects that the question asks about males.

Our model consistently performs better than DSP-CL. The example ‘what position is shaq oneal’ in Table 6.2 illustrates the difference. In this example, both BDSP-CL and SPO (no-lex) correctly predict the LF; however, DSP-CL fails because it cannot predict the entity ‘shaq oneal’ as the entity is never seen in training.

Background Effect If our background is in average closer to the true distribution compared to the uniform distribution, we hypothesize that the RNN will learn to predict a more uniform distribution compared to an RNN without background as

	DSP-CL	BDSP-CL
Avg. KL-divergence	3.13	1.95

Table 6.3: Average KL-divergence to the uniform distribution when models predict rules corresponding to named entities.

explained in subsection 6.3.1. To test this hypothesis, we randomly sample 100 distributions in housing domain when the RNN needs to predict a rule corresponding to a named entity. We calculate the average KL-divergence from these distribution to the uniform distribution and report the results in Table 6.3. The results seem to confirm our hypothesis: the KL-divergence is much smaller for BDSP-CL where a background takes into account the presence of certain named entities depending on the input.

6.5 Related Work

This chapter makes important extensions over chapter 5. While chapter 5 incorporates grammatical constraints into RNN models, we incorporate additional prior knowledge about input dependency. We propose to take into account the well-formedness of LFs by a WCFG and depending on the input, take into account the presence of certain entities inside LFs by WFSA(s). We choose to use WFSA modeling our input-dependent prior knowledge as the algorithm of intersection can efficiently combine WCFG and WFSA(s) to form the background priors guiding the RNN.

Taking into account prior knowledge about named entities is common in more traditional, symbolic semantic parsing systems [Liang *et al.*, 2011; Berant *et al.*, 2013; Kwiatkowski *et al.*, 2013; Reddy *et al.*, 2014]. For example, in [Berant *et al.*, 2013], named entities are first identified before they are used to construct the whole derivation tree. We propose to incorporate those knowledge into an RNN-based model. This is arguably more principled than Jia and Liang [2016]'s approach that incorporates such knowledge into an RNN using data augmentation.

The intersection algorithm used to compute the background allows local weight changes to propagate through the grammar tree thereby influencing the weight of each node inside the tree. This is related to the recent reinforcement learning research for semantic parsing [Liang *et al.*, 2016; Mou *et al.*, 2016] where rewards are propagated over different action steps.

More generally, this chapter is another instance of incorporating prior knowledge into deep learning models. We do this using symbolic objects such as grammar and automata. Salakhutdinov et al. [2013] model prior knowledge over the structure of the problem by combining hierarchical bayesian models and deep models while Hu et al. [2016] handle prior knowledge that can be expressed by first-order logic rules and uses these rules as a teacher network to transfer knowledge into deep models.

6.6 Conclusion

We propose to incorporate a symbolic background prior into RNN-based models to learn a semantic parser taking into account prior knowledge about LF well-formedness and about the likelihood of certain entities being present based on the input. We use a variant of a classical dynamic programming intersection algorithm to efficiently combine these factors and show that our Background-RNN yields promising results on *Overnight+*. In the future, we plan to explore the use of WFSA(s) with different topologies to model more prior knowledge.

Chapter 7

Conclusion and Perspectives

7.1 Summary

The main contribution of this thesis is the investigation of novel, neural approaches to executable semantic parsing. First, I showed that a standard sequence-to-sequence model could successfully be used to map natural language questions to logical forms (LFs), which are knowledge base queries for us. The approach was trained and tested on an existing dataset covering 7 domains and whose queries were compositionally non trivial. I then proposed different ways of integrating prior knowledge in neural approaches to semantic parsing with a focus on symbolic priors (grammars and automata). These contributions can be summarized as follows.

Orthogonal Embeddings for Simple queries. In the case where LFs are structurally simple (e.g. in the form of a triple (e_1, r, e_2) in the Wikianswers dataset [Fader *et al.*, 2011]), we follow the work of [Bordes *et al.*, 2014b] that learns sentence/LF embeddings and a scoring function between them. We remark that their model does not distinguish between entity (e_1, e_2) emeddings and relation (r) embeddings and propose an orthogonality regularizer to capitalize this prior knowledge (Chapter 4). We show empirically that adding this prior knowledge improves performance.

Integrating Grammatical Knowledge. In the case where LFs are more complex involving a fair amount of compositionality as in the Semantic Parsing Overnight (SPO) dataset [Wang *et al.*, 2015], we first notice that we can use RNNs to learn an end-to-end model mapping NL questions to KB queries. While for neural machine translation [Sutskever *et al.*, 2014], input and output are sentences, for semantic parsing, the input is a natural language sentence and the output a linearized logical formula. Based on the grammar developed by [Wang *et al.*, 2015] to generate the SPO dataset, we compare (Chapter 5) different ways of linearizing logical formulae: a direct linearization of the LF, a linearization of the associated canonical realization (produced for each LF by the grammar), and a sequence consisting of derivation steps relative to the underlying grammar. We further propose two ways of integrating grammatical constraints on the derivation sequence inside the RNN- based sequential predictor. We show empirically that our RNN-based semantic parser that integrates grammatical prior knowledge performs better than both its RNN baselines and a more traditional semantic parser.

Modelling Input Dependency. If the input question contains the string 'Barack Obama', it is highly likely that the LF of that question involves the entity Barack Obama and therefore, that the rule expanding to "barack obama" is present in the output derivation sequence. Remarking that our produced LFs have to be executable thus grounded to a particular knowledge base (KB), the contribution presented in Chapter 6 consists in integrating additional prior knowledge about input dependency into our previous RNN-based semantic parser integrating grammatical constraints. Technically, we first construct weighted automata (WFSAs) biasing the likelihood of certain rules corresponding to named entities being present inside the target derivation sequence. We model this prior knowledge using WFSAs because these can be combined efficiently with the grammatical prior knowledge we use previously in the form of a weighted context free grammar (WCFG). This combination is then treated as 'Background' for the RNN and we show that our extended 'Background RNN' semantic parser can further improve the performance.

7.2 Perspectives

There are two main directions for further research which it would be interesting to explore.

Application to Other Semantic Parsing Tasks. Executable semantic parsing aims at predicting a target language *grounded* in a KB. This shares similarityies with tasks such as instructing robots (i.e. mapping natural language utterances to robot commands) [Artzi and Zettlemoyer, 2013] and code generation (i.e mapping natural language text to a piece of general-pupose code) [Quirk *et al.*, 2015; Yin and Neubig, 2017] where the target languages are also *grounded.*

In Chapter 5, we show that having a grammar that constrains LF derivations is effective. Such grammars exist not only for knowledge base queries as we have considered in the thesis, but also for many other *grounded* languages such as robot commands and general-purpose codes. In the future, we would like to test the effectiveness of integrating grammatical prior knowledge on those tasks. An example of this line of research is [Yin and Neubig, 2017] who shows that integrating such grammars into an RNN-based model is effective for code generation.

Other forms of Prior Knowledge. While we have shown some successful semantic parsing examples of incorporating prior knowledge into expressive learning machines, we believe that there are more forms of prior knowledge to consider in the future. In particular, the grammar or automata that we use express explicit prior knowledge / constraints over our considered target sequences while many forms of prior knowledge is implicit (e.g. answer to a LF query, a natural language feedback to a predicted LF). Implicit prior knowledge is arguably more common and abundant and some works start exploiting this form of prior knowledge for neural networks [Liang *et al.*, 2016; Mou *et al.*, 2016; Guu *et al.*, 2017]. It would be interesting to explore to what extent this type of knowledge could be exploited to guide neural semantic parsing.

Bibliography

[Androutsopoulos, 1995] L. Androutsopoulos. Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering*, 1:29–81, 1995.

[Artzi and Zettlemoyer, 2013] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1):49–62, 2013.

[Aziz *et al.*, 2014] Wilker Aziz, Marc Dymetman, and Lucia Specia. Exact decoding for phrase-based statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1237–1249, 2014.

[Bahdanau *et al.*, 2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

[Bar-Hillel *et al.*, 1961] Yehoshua Bar-Hillel, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunicationsforschung*, 14:143–172, 1961.

[Baydin *et al.*, 2015] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015.

[Bengio, 2009] Yoshua Bengio. Learning deep architectures for ai. *Foundation and Trends in Machine Learning*, 2(1):1–127, January 2009.

[Berant and Liang, 2014] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Annual Meeting for the Association for Computational Linguistics (ACL)*, 2014.

[Berant *et al.*, 2013] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[Bergstra *et al.*, 2010] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, jun 2010. Oral.

[Billot and Lang, 1989] Sylvie Billot and Bernard Lang. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting on Association for Computational Linguistics*, ACL '89, pages 143–151. Association for Computational Linguistics, 1989.

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.

[Bordes *et al.*, 2014a] Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, 2014.

[Bordes *et al.*, 2014b] Antoine Bordes, Jason Weston, and Nicolas Usunier. Open question answering with weakly supervised embedding models. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD)*, 2014.

[Bottou, 1991] Léon Bottou. *Une Approche théorique de l'Apprentissage Connexionniste: Applications à la Reconnaissance de la Parole.* PhD thesis, Université de Paris XI, Orsay, France, 1991.

[Bottou, 2010] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *in COMPSTAT*, 2010.

[Cai and Yates, 2013] Qingqing Cai and Alexander Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *In Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

[Calder *et al.*, 1988] Jonathan Calder, Ewan Klein, and Henk Zeevat. Unification categorial grammar: A concise, extendable grammar for natural language processing. In *Proceedings of the 12th conference on Computational linguistics-Volume 1*, pages 83–86. Association for Computational Linguistics, 1988.

[Chen *et al.*, 2003] Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrick Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *Users Manual: RoboCup Soccer Server — for Soccer Server Version 7.07 and Later.* The RoboCup Federation, February 2003.

[Chiang, 2005] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA, 2005.

[Clark and Curran, 2007] Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552, 2007.

[Collins, 2002] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA, 2002.

[Cuong *et al.*, 2014] Nguyen Viet Cuong, Nan Ye, Wee Sun Lee, and Hai Leong Chieu. Conditional random field with high-order dependencies for sequence labeling and segmentation. *J. Mach. Learn. Res.*, 15(1):981–1009, January 2014.

[Cybenko, 1989] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[Dahl *et al.*, 1994] Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 43–48, 1994.

[Dalrymple, 2001] Mary Dalrymple. *Lexical-Functional Grammar (Syntax and Semantics, Volume 34) (Syntax and Semantics).* MIT Press, August 2001.

[Daumé and Marcu, 2005] Hal Daumé, III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 169–176, New York, NY, USA, 2005. ACM.

[Deerwester *et al.*, 1990] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407, 1990.

[Deoras *et al.*, 2011] Anoop Deoras, Tomas Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur. Variational approximation of long-span language models for lvcsr. In *ICASSP*, pages 5532–5535. IEEE, 2011.

[Dong and Lapata, 2016] Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[Duchi *et al.*, 2011] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[Dyer, 2010] Christopher Dyer. *A Formal Model of Ambiguity and its Applications in Machine Translation.* PhD thesis, University of Maryland, 2010.

[Dymetman and Xiao, 2016] Marc Dymetman and Chunyang Xiao. Log-linear rnns: Towards recurrent neural networks with flexible prior knowledge. *CoRR*, abs/1607.02467, 2016.

[Elman, 1990] Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.

[Fader *et al.*, 2011] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference of Empirical Methods in Natural Language Processing (EMNLP '11)*, Edinburgh, Scotland, UK, July 27-31 2011.

[Fader *et al.*, 2013] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1608–1618, 2013.

[Faruqui *et al.*, 2014] Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard H. Hovy, and Noah A. Smith. Retrofitting word vectors to semantic lexicons. *CoRR*, abs/1411.4166, 2014.

[Ganitkevitch *et al.*, 2013] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. Ppdb: The paraphrase database. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 758–764, 2013.

[Goodfellow *et al.*, 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Goyal *et al.*, 2016] Raghav Goyal, Marc Dymetman, and Éric Gaussier. Natural language generation through character-based rnns with finite-state prior knowledge. In *COLING*, pages 1083–1092, 2016.

[Graham *et al.*, 1980] Susan L. Graham, , and Michael Harrison Walter L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462, July 1980.

[Guu *et al.*, 2017] K. Guu, P. Pasupat, E. Z. Liu, and P. Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Association for Computational Linguistics (ACL)*, 2017.

[He and Young, 2005] Yulan He and Steve Young. Semantic processing using the hidden vector state model. *Computer Speech and Language*, 19(1):85–106, January 2005.

[Hendrix *et al.*, 1978] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, June 1978.

[Hirschman *et al.*, 1993] L. Hirschman, M. Bates, D. Dahl, W. Fisher, J. Garofolo, D. Pallett, K. Hunicke-Smith, P. Price, A. Rudnicky, and E. Tzoukermann. Multi-site data collection and evaluation in spoken language understanding. In *Proceedings of the Workshop on Human Language Technology*, HLT '93, pages 19–24, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics.

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[Hochreiter, 1991] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[Hopcroft *et al.*, 2006] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[Hornik, 1991] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Network*, 4(2):251–257, March 1991.

[Hu *et al.*, 2016] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *ACL (1)*, 2016.

[Jia and Liang, 2016] Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[Jordan, 1986] Michael I. Jordan. Serial order: A parallel, distributed processing approach. Technical report, Institute for Cognitive Science, University of California, San Diego, 1986.

[Joshi and Vijay-Shanker, 2001] Aravind K Joshi and K Vijay-Shanker. Compositional semantics with lexicalized tree-adjoining grammar (ltag): How much underspecification is necessary? In *Computing Meaning*, pages 147–163. Springer, 2001.

[Kate and Mooney, 2006] Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *ACL 2006: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 913–920, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[Kate *et al.*, 2005a] Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. Learning to transform natural to formal languages. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 1062–1068, 2005.

[Kate *et al.*, 2005b] Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. Learning to transform natural to formal languages. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, pages 1062–1068, 2005.

[Kuhlmann *et al.*, 2004] Gregory Kuhlmann, Peter Stone, Raymond J. Mooney, and Jude W. Shavlik. Guiding a reinforcement learner with natural language advice:

Initial results in robocup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, July 2004.

[Kuhn and De Mori, 1995] Roland Kuhn and Renato De Mori. The application of semantic classification trees to natural language understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:449–460, 1995.

[Kwiatkowski *et al.*, 2011] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical generalization in ccg grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1512–1523, Stroudsburg, PA, USA, 2011.

[Kwiatkowski *et al.*, 2013] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Empirical Methods in Natural Language Processing, (EMNLP)*, pages 1545–1556, 2013.

[Lafferty *et al.*, 2001] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, 2001.

[Lebret and Collobert, 2015] Rémi Lebret and Ronan Collobert. Rehabilitation of count-based models for word vector representations. In *Computational Linguistics and Intelligent Text Processing - 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part I*, pages 417–429, 2015.

[Liang *et al.*, 2011] Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 590–599, 2011.

[Liang *et al.*, 2016] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *CoRR*, abs/1611.00020, 2016.

[Liang, 2016] Percy Liang. Learning executable semantic parsers for natural language understanding. *Commun. ACM*, 59(9):68–76, 2016.

[Lodhi *et al.*, 2002] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002.

[Macherey *et al.*, 2001] Klaus Macherey, Franz Josef Och, and Hermann Ney. Natural language understanding using statistical machine translation. In *In European Conf. on Speech Communication and Technology*, pages 2205–2208, 2001.

[Mesnil *et al.*, 2015] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, and Geoffrey Zweig. Using recurrent neural networks for slot filling in spoken language understanding. *Trans. Audio, Speech and Lang. Proc.*, 23(3):530–539, March 2015.

[Mikolov *et al.*, 2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[Mikolov *et al.*, 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[Miller *et al.*, 1996] Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 55–61. Association for Computational Linguistics, 1996.

[Mohri, 2009] Mehryar Mohri. *Weighted Automata Algorithms*, pages 213–254. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Mou *et al.*, 2016] Lili Mou, Zhengdong Lu, Hang Li, and Zhi Jin. Coupling distributed and symbolic execution for natural language queries. *CoRR*, abs/1612.02741, 2016.

[Mozer, 1992] Michael C. Mozer. The induction of multiscale temporal structure. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems, 4*, pages 275–282. Morgan Kaufmann, 1992.

[Nederhof and Satta, 2003] Mark-Jan Nederhof and Giorgio Satta. Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, 2003.

[Nederhof and Satta, 2008] Mark-Jan Nederhof and Giorgio Satta. Probabilistic parsing. In *New Developments in Formal Languages and Applications*, pages 229–258. 2008.

[Neelakantan *et al.*, 2016] Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. *CoRR*, abs/1611.08945, 2016.

[Papineni *et al.*, 1997] Kishore Papineni, Salim Roukos, and Todd Ward. Feature-based language understanding. In *Fifth European Conference on Speech Communication and Technology, EUROSPEECH 1997, Rhodes, Greece, September 22-25, 1997*, 1997.

[Pascanu *et al.*, 2013] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013.

[Pasupat and Liang, 2015] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *ACL(1)*, 2015.

[Pereira and Warren, 1980] Fernando C.N. Pereira and David H.D. Warren. Definite clause grammars for language analysis a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231 – 278, 1980.

[Pollard and Ivan A., 1994] Carl Pollard and Sag Ivan A. *Head-driven phrase structure grammar*. University of Chicago Press, 1994.

[Quirk *et al.*, 2015] Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 878–888, Beijing, China, July 2015.

[Raymond and Riccardi, 2007] Christian Raymond and Giuseppe Riccardi. Generative and discriminative algorithms for spoken language understanding. In *InterSpeech*, pages 1605–1608, Antwerp, Belgium, August 2007.

[Reddy *et al.*, 2014] Siva Reddy, Mirella Lapata, and Mark Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

[Rumelhart *et al.*, 1988] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. In *Neurocomputing: Foundations of Research*, chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2 edition, 2003.

[Sahlgren, 2005] Magnus Sahlgren. An introduction to random indexing. In *In Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering*, 2005.

[Salakhutdinov *et al.*, 2013] Ruslan Salakhutdinov, Joshua B. Tenenbaum, and Antonio Torralba. Learning with hierarchical-deep models. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 35(8):1958–1971, 2013.

[Schwartz *et al.*, 1996] Richard Schwartz, Scott Miller, David Stallard, and John Makhoul. Language understanding using hidden understanding models. In *In Proc. ICSLP-96*, pages 997–1000, 1996.

[Shieber, 2014] Stuart M. Shieber. Bimorphisms and synchronous grammars. *J. Language Modelling*, 2(1):51–104, 2014.

[Siegelmann and Sontag, 1991] Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4:77–80, 1991.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15(1):1929–1958, 2014.

[Steedman, 1996] Mark Steedman. *Surface structure and interpretation.* Linguistic inquiry monographs, 30. MIT Press, 1996.

[Stolcke, 1994] Andreas Stolcke. An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities. *Computational Linguistics*, 21(2):165–201, 1994.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112. 2014.

[Tang and Mooney, 2001] Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, Freiburg, Germany, 2001.

[Tieleman and Hinton., 2012] T. Tieleman and G. E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. 2012.

[Vinyals *et al.*, 2014] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. *Neural Information Processing Systems (NIPS)*, 2014.

[Wang *et al.*, 2015] Yushi Wang, Jonathan Berant, and Percy Liang. Building a semantic parser overnight. In *Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL Volume 1: Long Papers*, pages 1332–1342, 2015.

[Warren and Pereira, 1982] David H. D. Warren and Fernando C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *Comput. Linguist.*, 8(3-4):110–122, July 1982.

[Wen *et al.*, 2015] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-hao Su, David Vandyke, and Steve J. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[Wong and Mooney, 2006] Yuk Wah Wong and Raymond J. Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics Annual Meeting (HLT-NAACL-06)*, pages 439–446, New York City, NY, 2006.

[Wong and Mooney, 2007] Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[Woods *et al.*, 1972] W. Woods, R. Kaplan, and B. Nash-Webber. The lunar sciences natural language information system: Final report. Technical report, Bolt, Beranek and Newman, Inc., Cambridge, MA, 1972.

[Xiao *et al.*, 2016a] Chunyang Xiao, Guillaume Bouchard, Marc Dymetman, and Claire Gardent. Orthogonality regularizer for question answering. In *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics, *SEM@ACL 2016, Berlin, Germany, 11-12 August 2016*, 2016.

[Xiao *et al.*, 2016b] Chunyang Xiao, Marc Dymetman, and Claire Gardent. Sequence-based structured prediction for semantic parsing. In *ACL(1)*, 2016.

[Xiao *et al.*, 2017] Chunyang Xiao, Marc Dymetman, and Claire Gardent. Symbolic priors for rnn-based semantic parsing. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4186–4192, 2017.

[Yang and Chang, 2016] Yi Yang and Ming-Wei Chang. S-MART: novel tree-based structured learning algorithms applied to tweet entity linking. *CoRR*, abs/1609.08075, 2016.

[Yao *et al.*, 2014] Enpeng Yao, Guoqing Zheng, Ou Jin, Shenghua Bao, Kailong Chen, Zhong Su, and Yong Yu. Probabilistic text modeling with orthogonalized topics. In *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*, pages 907–910, 2014.

[Yih *et al.*, 2014] Wen-tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. 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 643–648, 2014.

[Yih *et al.*, 2015] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2015*, 2015.

[Yin and Neubig, 2017] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *CoRR*, abs/1704.01696, 2017.

[Zanzotto and Dell'Arciprete, 2012] Fabio Massimo Zanzotto and Lorenzo Dell'Arciprete. Distributed tree kernels. In *International Conference on Machine Learning (ICML)*, 2012.

[Zelle and Mooney, 1996] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, pages 1050–1055, Portland, OR, August 1996. AAAI Press/MIT Press.

[Zettlemoyer and Collins, 2005] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pages 658–666, 2005.

[Zettlemoyer and Collins, 2007] Luke S. Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-2007*, pages 678–687, 2007.