Learning CCG parser

Dor Muhlgay

Outline

- Introduction
- CCG
- The PCCG model
- Learning
- Results
- Conclusion

Introduction

Given a sentence in NL which represents some task, our goal is to recover enough information to complete the task.

- Converting questions to database queries
- Converting robots instructions to path planning

We will achieve this by mapping the sentence to its logical form

list flights to boston

$\lambda x.\text{flight}(x) \wedge \text{to}(x, \text{boston})$

Introduction

We define a learning task - given a data set of sentences and their representation in the logical form, we will learn a function from sentences to their logical form.

f: Sentence → Logical form

list flights to boston

$\lambda x.\text{flight}(x) \wedge \text{to}(x, \text{boston})$

Outline

- Introduction
- CCG
- The PCCG model
- Learning
- Results
- Conclusion

Lambda calculus

The logical form of a sentence is represented by Lambda calculus. Lambda calculus is a formal system in mathematical logic for expressing computation. The lambda–calculus expressions we use are formed from the following items:

- Constants - represents entities in the world by name of predicate.
  - TEXAS, NYC, boston(x)
- Logical connectors: conjunction (∧), disjunction (∨), negation (¬)
- Quantifiers
- Lambda expressions:Lambda expressions represent functions
  - For example λx.borders(x, texas), λx.flight(x) ∧ to(x, boston)

Lambda calculus

Lambda calculus has three basic types:

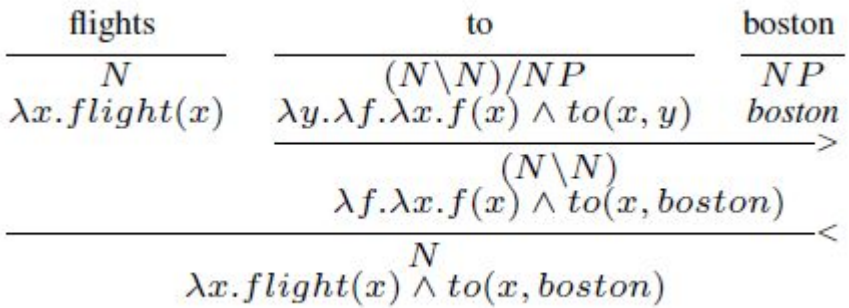
- e: the type of entities
- t: the type of truth values
- r: the type of real numbers.
- Functional types <t1, t2> : The type assigned to functions that map from objects from type t1 to objects from type t2.

In specific domains, we will specify subtype hierarchies for e. For example, in a geography domain we might distinguish different entity subtypes such as cities, states, and rivers

Combinatory Categorial Grammar

CCG is a categorial formalism, which provides transparent interface between syntax and semantics.We will use CCG to construct sentences in logical form from sentences in natural language.

CCG parsing tree example:



CCG categories

The CCG categories are the CCG building blocks. The categories represent the connection between the syntax of the sentence to its semantics.

CCG categories examples:

$$N : \lambda x.flight(x)$$
$$(N\backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x, y)$$
$$NP : boston$$

CCG categories

The expression on the left side of the colon represents the syntactic part.
It may consists:

- Simple part of speech types such as N, NP, or S
- Syntactic combination operator “\” or “/”.

$$N : \lambda x.flight(x)$$
$$(N\backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x, y)$$
$$NP : boston$$

CCG categories

The expression on the right side of the colon represents the semantic part of the category.

- The semantic part is simply a lambda expression
- The semantic part always matches the syntactic part.

$$N : \lambda x.flight(x)$$
$$(N\backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x, y)$$
$$NP : boston$$

CCG lexicon

The CCG lexicon maps between words in the natural language, to the CCG category that represents them:

- Each lexical entry consists a natural word and its corresponding CCG category
- The CCG lexicon is used to map the words of the sentence to the leaves of the CCG parsing tree.
- There are many lexical entries in the lexicon.

CCG lexicon example:

$$\begin{array}{lll} \text{flights} & := & N : \lambda x.flight(x) \\ \text{to} & := & (N \backslash N)/NP : \lambda y.\lambda f.\lambda x.f(x) \wedge to(x,y) \\ \text{boston} & := & NP : boston \end{array}$$

CCG combinators

CCG makes use of a set of CCG combinators, which are used to combine categories to form larger pieces of syntactic and semantic structure.

- There is a small set of CCG combinators
- Each CCG combinator receives 1 or 2 CCG categories, and output one “larger” CCG category.
- The CCG combinators operate jointly on both the syntactic and semantic parts of the categories.

CCG combinators - application

The functional application rules:

$$A/B : f \quad B : g \quad \Rightarrow \quad A : f(g) \qquad (>)$$
$$B : g \quad A\backslash B : f \quad \Rightarrow \quad A : f(g) \qquad (<)$$

- The first rule states that a category with syntactic type *A/B* can be combined with a category to the right of syntactic type *B* to create a new category of type A. It also states that the new semantics will be formed by applying the function *f* to the expression *g*.
- The second rule handles arguments to the left - the direction of the application is determined by the direction of the slash.
- The rule is equivalent to function application.

CCG combinators - composition

The functional composition rules:

$$A/B : f \quad B/C : g \quad \Rightarrow \quad A/C : \lambda x.f(g(x)) \quad (> \mathbf{B})$$
$$B\backslash C : g \quad A\backslash B : f \quad \Rightarrow \quad A\backslash C : \lambda x.f(g(x)) \quad (< \mathbf{B})$$

- The notion of functional application can be extended to function combination.
- As before, the direction of the slash implies the direction of the function application

CCG combinators - type shifting

Example of type shifting rules:

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$PP : \lambda x.g(x) \Rightarrow N\backslash N : \lambda f.\lambda x.f(x) \wedge g(x)$$

$$AP : \lambda e.g(e) \Rightarrow S\backslash S : \lambda f.\lambda e.f(e) \wedge g(e)$$

$$AP : \lambda e.g(e) \Rightarrow S/S : \lambda f.\lambda e.f(e) \wedge g(e)$$

- Type shifting rules are unary operation, that receives a category and returns a modified category.
- The rules are used to change a category during parsing.
- The rules helps keeping a compact lexicon

CCG combinators - coordination

The words “and” and “or” are mapped to special categories in the lexicon.

$$\text{and} \vdash C : conj$$
$$\text{or} \vdash C : disj$$

The coordination rule is defined as follows on the syntactic part:

$$X \quad conj \quad X \Rightarrow X$$

The semantic part of the rule is simply connecting the two logical forms with the appropriate logical connector

CCG parsing example

square	blue	or	round	yellow	pillow
ADJ	ADJ	C	ADJ	ADJ	N
$\lambda x.square(x)$	$\lambda x.blue(x)$	$disj$	$\lambda x.round(x)$	$\lambda x.yellow(x)$	$\lambda x.pillow(x)$

Use the lexicon to map the words and phrases in the sentence to CCG categories

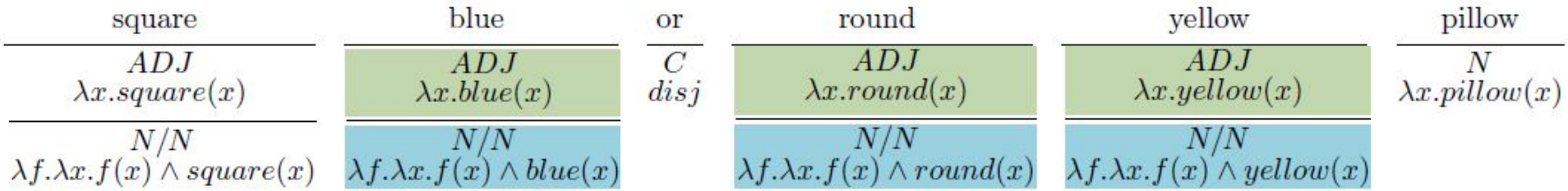
CCG parsing example

square	blue	or	round	yellow	pillow
ADJ $\lambda x.square(x)$	ADJ $\lambda x.blue(x)$	C $disj$	ADJ $\lambda x.round(x)$	ADJ $\lambda x.yellow(x)$	N $\lambda x.pillow(x)$
N/N $\lambda f.\lambda x.f(x) \wedge square(x)$					

Apply the type shifting rule to change the adjectives types

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

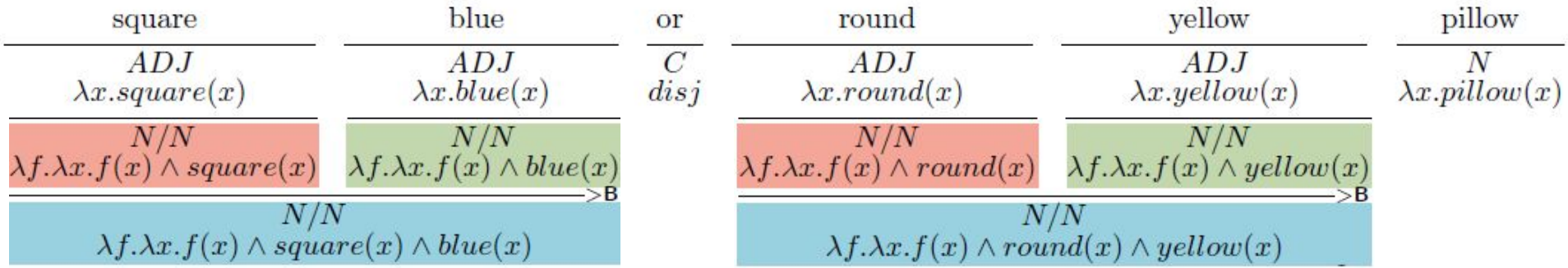
CCG parsing example



Apply the type shifting rule to change the adjectives types

$$ADJ : \lambda x.g(x) \Rightarrow N/N : \lambda f.\lambda x.f(x) \wedge g(x)$$

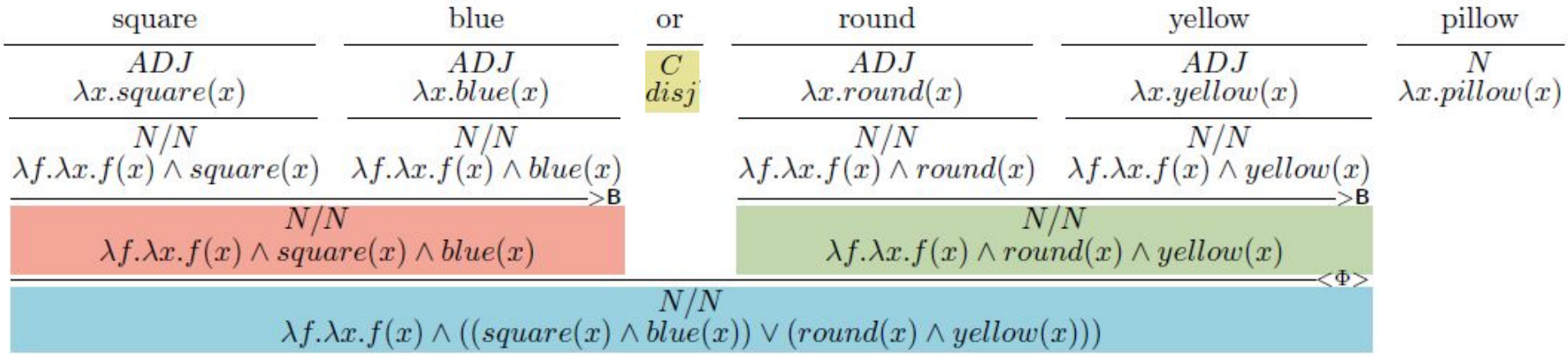
CCG parsing example



Apply the function composition rule:

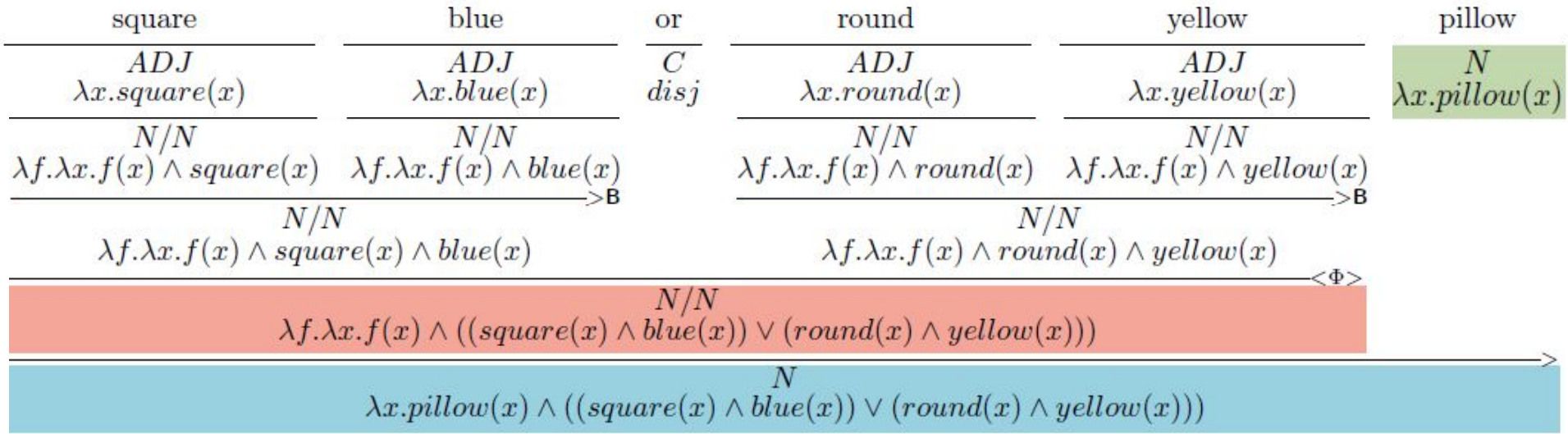
$$A/B : f \quad B/C : g \Rightarrow A/C : \lambda x.f(g(x)) \quad (> B)$$

CCG parsing example



Apply the coordination rule to combine the categories adjacent to “or”

CCG parsing example



Use the function application rule to apply the adjective on the noun:

$$A/B : f \quad B : g \Rightarrow A : f(g) \quad (>)$$

Extended CCG

Zettlemoyer and Collins extended CCG and added the following combinators:

- A set of functional combinators that allow the parser to significantly relax constraints on word order.
- A set of type-raising rules which allow the parser to cope with missing function words.

Outline

- Introduction
- CCG
- The PCCG Model
- Learning
- Results
- Conclusion

CCG ambiguity

A sentence might have more than one valid parse tree and logical form.
Ambiguity can be caused by:
One source of ambiguity is lexical items having more than one entry in the lexicon.

- Multiple lexical items are mapped to same word/phrase. For example, New York might have entries NP : new york city and NP : new york state.
- A single logical form might be derived from multiple parse trees (spurious ambiguity).

PCCG

A CCG receives a sentence S, and outputs a pair (L,T), where L is the logical form and T is the parse tree. In CCG there might be multiple pairs (L,T) that matches a sentence S.

We resolve the ambiguity issue by generalizing CCG to probabilistic CCG (PCCG):

- A PCCG defines a conditional distribution $P(L,T|S)$ over possible (L,T) pairs for a given sentence S.
- PCCGs deal with ambiguity by ranking alternative parses for a sentence in order of probability.

PCCG - Feature Vector

We assume a function f mapping (L, T, S) triples to feature vectors in $\mathbb{R}^d$.

- f is defined by d individual features, so that $f(L, T, S) = \langle f1(L, T, S), \ldots, fd(L, T, S) \rangle$.
- Each feature fj is typically the count of some sub-structure within (L, T, S). The features are restricted to be sum of local features, i.e features that depend only on a specific node in the parse tree.
  - The count of a specific lexical entry in the subtree
  - The number of times a predicate is conjoined with itself:

$$\lambda x.flight(x) \wedge from(x, new\_york) \wedge from(x, boston)$$

PCCG - Log linear model

We chose to represent the probability $P(L,T|S)$ in the log-linear model:

$$P(L,T|S;\bar{\theta}) = \frac{e^{\bar{f}(L,T,S)\cdot\bar{\theta}}}{\sum_{(L,T)} e^{\bar{f}(L,T,S)\cdot\bar{\theta}}}$$

The model is parameterized by the vector $\bar{\theta} \in \mathbb{R}^d$.

PCCG - Parsing

In order to Parse a sentence in the PCCG model, we would like to find

$$y^*(x) = \arg \max_{y \in \text{GEN}(x;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x, y)$$

Given the parameters vector and a lexicon, we can use a variant of the CKY algorithm with beam search to find some approximation of y^*:

- We keep a parse chart C, where each entry $C[i,j]$ represents the sub-sentence from word i to j.
- Each entry $C[i,j]$ has a set of possible parse-tree root for the sub sentence they represents and their score. The root holds a full CCG category.
- We prune the roots sets to keep only the N-highest scoring roots in each entry.
- After filling the chart, we look for the highest scoring root in $C[1,n]$, and backtrack to recover the entire tree.

Outline

- Introduction
- CCG
- The PCCG Model
- Learning
- Results
- Conclusion

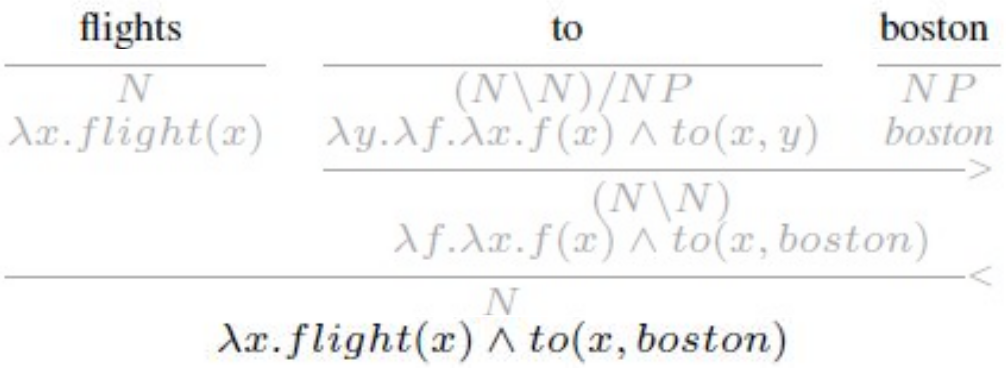
Learning

Our goal is to learn a CCG parser from our supervised data set.

- In particular, we would like to learn the parameter vector $\bar{\theta}$ and the CCG lexicon. The combinators rules are predefined.
- How does the supervised data looks like?

Learning - supervised data

A fully supervised data set includes the sentence, parse tree and logical form.



However, usually the data set doesn't contain the parse tree, and we will treat it as a hidden variable.

Learning

We will introduce an algorithm that learns both the lexicon and parameter vector $\bar{\theta}$. The algorithm iterates over the examples, and perform the following steps for each example:

- **Step 1- Lexicon update:** Add new lexical entries according to the example.
- **Step 2- Parameters update:** Re-estimate the parameters according to the new lexicon. Can be done in the following ways:
  - Gradient ascent (Zettlemoyer and Collins 2005)
  - Hidden variable Structured perceptron (Zettlemoyer and Collins 2007)

Gradient Ascent

We will define the log-likelihood function:

$$O(\bar{\theta}) = \sum_{i=1}^{n} \log P(L_i|S_i;\bar{\theta})$$
$$= \sum_{i=1}^{n} \log \left(\sum_{T} P(L_i, T|S_i;\bar{\theta}) \right)$$

Where $P(L,T|S)$ is represented by the log-linear model:

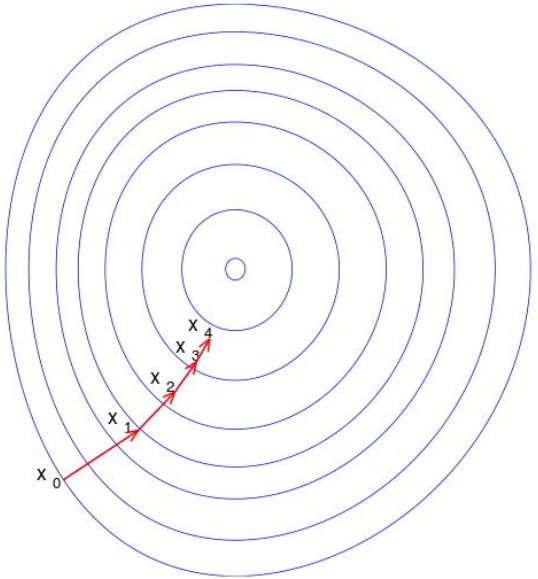
$$P(L,T|S;\bar{\theta}) = \frac{e^{\bar{f}(L,T,S)\cdot\bar{\theta}}}{\sum_{(L,T)} e^{\bar{f}(L,T,S)\cdot\bar{\theta}}}$$

Our goal is to find $\bar{\theta}$ that maximizes the log-likelihood function.

Gradient Ascent

Gradient ascent is an iterative optimization algorithm to find a local maximum.

- At each iteration the algorithm takes steps proportional to the the gradient of the function at the current point.
- The gradient points to in the direction of the greatest rate of increase of the function. At each step we increase the function until we reach a local maxima.
- The size of each step is defined by the step-rate parameter. The steps should be large enough so the algorithm will converge quickly, but small enough so we won't overshoot (skip the local maxima).



Gradient Ascent

We will use gradient ascent to maximize the log-likelihood function:

Set $\bar{\theta}$ to some initial value
for $k = 0 \ldots N-1$
  for $i = 1 \ldots n$
    $\bar{\theta} = \bar{\theta} + \frac{\alpha_0}{(1+ct)} \frac{\partial \log P(L_i|S_i;\bar{\theta})}{\partial \bar{\theta}}$

The j element in the gradient vector is defined as follows:

$$\frac{\partial O}{\partial \theta_j} = \sum_{i=1}^{n} \sum_{T} f_j(L_i, T, S_i) P(T|S_i, L_i; \bar{\theta}) - \sum_{i=1}^{n} \sum_{L,T} f_j(L, T, S_i) P(L, T|S_i; \bar{\theta})$$

Gradient Ascent

A few notes:

- The jth parameter is updated by taking the difference between the expected score of the jth feature given the sentence and the label, and the expected score of the jth feature when only the sentence is given. Intuitively, it is the difference between what we the feature should be and what we think it should be with our current parameters.
- The step rate is defined by the parameters α_0 and c. The parameter t is the count of iterations - we decrease the step size at each iteration.
- The gradient ascent is a batch method which requires a pass over the over the entire training set, which effects on the algorithm efficiency. We will introduce the perceptron algorithm, which is fully online.

Structured perceptron (Collins 2002)

Data: $\{(x_i, y_i) : i = 1 \ldots n\}$

For $t = 1 \ldots T$: [iterate epochs]

For $i = 1 \ldots n$: [iterate examples]

$y^* \leftarrow \arg\max_y \langle \theta, \Phi(x_i, y) \rangle$ [predict]

If $y^* \neq y_i$: [check]

$\theta \leftarrow \theta + \Phi(x_i, y_i) - \Phi(x_i, y^*)$ [update]

Structured perceptron

The structured perceptron:

- Fully online algorithm, that is the learning is done example by example.
- The parameters update is done in additive fashion - Simple and easy to implement.
- Requires an efficient algorithm for *argmax*. In some applications of the perceptron algorithm, a set of candidates is generated and the highest scoring output is found from the candidates set.
- If there is a linear separator that classifies all the examples correctly, the algorithm guarantees convergence - we will show a formal proof.

Structured perceptron convergence

Basic definitions:

- Let $B(xi)$ be the set of incorrect label for an example xi, i.e $B(xi) = \{z \mid z \mathrel{!}= yi\}$.
- A training set is "separable with margin $\delta > 0$" if there exists some vector U with $||Uj||$ = 1 (|| || - norm 2), such that for every i end every z in $B(xi)$:

$$\mathbf{U} \cdot \Phi(x_i, y_i) - \mathbf{U} \cdot \Phi(x_i, z) \geq \delta$$

- Let R be a constant such that for every i end every z in $B(xi)$:

$$||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R$$

We will prove that if a training set is "separable with margin δ", then

$$Number\ of\ mistakes \leq \frac{R^2}{\delta^2}$$

Structured perceptron convergence

Proof outline:

- Let $\bar{\alpha}^k$ be the parameter vector just before the algorithm made the k mistake.We will assume that the weight vector is initialized to the be the zero vector.
- We will prove the lower bound: $k^2\delta^2 \leq ||\bar{\alpha}^{k+1}||^2$
- We will prove the upper bound: $||\bar{\alpha}^{k+1}||^2 \leq kR^2$
- The claim immediately follows: $k \leq R^2/\delta^2$

Structured perceptron convergence

Lower bound $k^2\delta^2 \leq ||\bar{\alpha}^{k+1}||^2$ proof:

- Let z be the proposed label when the algorithm made k'th mistake.
- According to the update rule: $\bar{\alpha}^{k+1} = \bar{\alpha}^k + \Phi(x_i, y_i) - \Phi(x_i, z)$
- We take the inner product of both sides of the equality with vector U, and apply the "separable with margin" property:

$$\begin{aligned} \mathbf{U} \cdot \bar{\alpha}^{k+1} &= \mathbf{U} \cdot \bar{\alpha}^k + \mathbf{U} \cdot \Phi(x_i, y_i) - \mathbf{U} \cdot \Phi(x_i, z) \\ &\geq \mathbf{U} \cdot \bar{\alpha}^k + \delta \end{aligned}$$

- Because $\mathbf{U} \cdot \bar{\alpha}^1 = 0$ we get $\mathbf{U} \cdot \bar{\alpha}^{k+1} \geq k\delta$.
- On the other hand: $\mathbf{U} \cdot \bar{\alpha}^{k+1} \leq ||\mathbf{U}||\ ||\bar{\alpha}^{k+1}||$, and hence:

$$||\bar{\alpha}^{k+1}|| \geq k\delta \rightarrow k^2\delta^2 \leq ||\bar{\alpha}^{k+1}||^2$$

Structured perceptron convergence

Upper bound $||\bar{\alpha}^{k+1}||^2 \leq kR^2$ proof:

- Let z be the proposed label when the algorithm made k'th mistake.
- According to the update rule: $\bar{\alpha}^{k+1} = \bar{\alpha}^k + \Phi(x_i, y_i) - \Phi(x_i, z)$
- The following holds:

$$\begin{aligned} ||\bar{\alpha}^{k+1}||^2 &= ||\bar{\alpha}^k||^2 + ||\Phi(x_i, y_i) - \Phi(x_i, z)||^2 \\ &\quad +2\bar{\alpha}^k \cdot (\Phi(x_i, y_i) - \Phi(x_i, z)) \\ &\leq ||\bar{\alpha}^k||^2 + R^2 \end{aligned}$$

  The inequality holds because of the assumption $||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R$,
  And because $\bar{\alpha}^k \cdot (\Phi(x_i, y_i) - \Phi(x_i, z)) \leq 0$ (z is the highest scoring label with the the parameters $\bar{\alpha}^k$).
- By induction we get the upper bound: $||\bar{\alpha}^{k+1}||^2 \leq kR^2$

Hidden variable structured perceptron

- Structured perceptron doesn’t work for our learning task as is, since we have latent data - the parse tree.
- Structured perceptron can be extended to support hidden variable.

Hidden variable structured perceptron

Data: $\{(x_i, y_i) : i = 1 \ldots n\}$

For $t = 1 \ldots T$: [iterate epochs]

For $i = 1 \ldots n$: [iterate examples]

$y^*, h^* \leftarrow \arg\max_{y,h} \langle \theta, \Phi(x_i, h, y) \rangle$ [predict]

If $y^* \neq y_i$: [check]

$h' \leftarrow \arg\max_h \langle \theta, \Phi(x_i, h, y_i)$ [predict hidden]

$\theta \leftarrow \theta + \Phi(x_i, h', y_i) - \Phi(x_i, h^*, y^*)$ [update]

Hidden variable structured perceptron

The hidden variable structured perceptron:

- Similar to the gradient ascent algorithm, where the expectation were replaced with argmax.
- Fully online and efficient algorithm.
- No known convergence guarantees, but works well empirically

We will now put all the parts together and introduce an online algorithm for CCG parser (Zettlemoyer and Collins 2007), based on the hidden variable structured perceptron algorithm.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i,z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

The algorithm iterates over the example T times. For each sample, the algorithm follows 3 steps:

- **Step 1 (check correctness):** We try to parse the sentence in the example with our current parameters and lexicon. If the logical form we obtained matches the label, we move to next example.
- **Step 2 (Lexical generation):** We add new lexical entries to our lexicon.
- **Step 3 (Update parameters):** We try to parse the sentence again with the new lexicon. If we fail, we update the parameters so they would fit the new lexicon.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i,z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

Λ - The current lexicon
W - The current parameter vector
F - The feature function
$\text{GEN}(x; \Lambda)$- The set of all the parse trees for the sentence *x*.

- We parse the sentence *xi* according to our current lexicon and parameters.
- The parsing is done by finding the parse tree that maximizes *w*f(x,y)*, and it can be computed efficiently by a CKY-style algorithm.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i, z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

L(y) - Extract the logical form from the parse tree y

- Check if the logical form outputted by the parse tree is correct.
- If the result equals to the example label, the algorithms goes to next example. Else, the algorithm goes to next step - fixing the lexicon.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i,z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

Λ - The current lexicon
λ - The new temporary lexicon
$\text{GENLEX}(x_i, z_i)$ - given a sentence and its logical form, GENLEX generates a set of possible lexical entries.

- The algorithm runs GENLEX to generate a noisy set of possible lexical entries.
- A temporary lexicon is created from our current lexicon and the new lexical entries.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i,z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

$\text{GEN}(x, z; \Lambda)$ - the set of all the parse trees for sentence x, with the lexicon Λ and the logical form z.

- The algorithm parse the sentence *xi* according to the temporary lexicon and current parameters, given that the logical form is *zi*.
- The parsing is done by finding the parse tree *y** that maximizes *w*f(x,y)*, from all the parse trees whose logical forms are *zi*.
  - We can use a similar algorithm to the one we used for parsing, where in the pruning step we eliminate trees that aren't consistent with the logical form *zi*.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i,z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

- We obtain the lexical entries of y^*
- We add the new lexical entries to the lexicon.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i, z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

- We parse the sentence *xi* according to the new lexicon and current parameters, and obtains the parse tree *y'*.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i, z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

- If the logical form obtained from the parse tree *y'* doesn't match the label *zi*, we do an additive update of the parameter vector.

Online learning algorithm for CCG

Algorithm:

- For $t = 1 \ldots T, i = 1 \ldots n$:

Step 1: (Check correctness)

- Let $y^* = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y^*) = z_i$, go to the next example.

Step 2: (Lexical generation)

- Set $\lambda = \Lambda \cup \text{GENLEX}(x_i, z_i)$.
- Let $y^* = \arg\max_{y \in \text{GEN}(x_i, z_i;\lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- Define λ_i to be the set of lexical entries in y^*.
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$.

Step 3: (Update parameters)

- Let $y' = \arg\max_{y \in \text{GEN}(x_i;\Lambda)} \mathbf{w} \cdot \mathbf{f}(x_i, y)$.
- If $L(y') \neq z_i$:
  - Set $\mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y^*) - \mathbf{f}(x_i, y')$.

Output: Lexicon Λ together with parameters $\mathbf{w}$.

- After iterating over all the samples T times, the algorithm returns the lexicon Λ and parameter vector w.

GENLEX

GENLEX receives a pair of sentence S and its logical form L, and returns a set of lexical entries. GENLEX generates a large set of lexical entries, some of them may be illegal. Those items are eliminated from the lexicon in a later stage of the algorithm.

The set returned by GENLEX is defined as follows:

$$\text{GENLEX}(S, L) = \{x := y \mid x \in W(S), y \in C(L)\}$$

Where $W(S)$ is the set of all sub-sentences in S, and $C(L)$ is a set of categories derived from L.

GENLEX - categories generation

The categories generation function $C(L)$ s defined through a set of rules that examine L and produce categories based on its structure:

- Each rule consists of a trigger that identifies some substructure within the logical form L.
- For each sub-structure in L that matches the trigger, a category is created and added to C(L).

GENLEX - categories generation

An example of the *C(L)* rules:

Rules		Categories produced from logical form
Input Trigger	Output Category	$\arg\max(\lambda x.state(x) \wedge borders(x, texas), \lambda x.size(x))$
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	$S/NP : \lambda x.f(x)$	$S/NP : \lambda x.size(x)$

Outline

- Introduction
- CCG
- The PCCG Model
- Learning
- Results
- Conclusion

Results

Experiments setup:

- The training set and data set were extracted from ATIS and Geo880 domains.
- Parameters setting:
  - The number of passes over the training set
  - The initial weight of the initial lexical entries' features
  - The initial weight of the learned lexical entries' features

Results

Results compercement with He and Young 2006 on ATIS data set:

	Precision	Recall	F1
Single-Pass Parsing	96.76	86.89	91.56
Two-Pass Parsing	95.11	96.71	95.9
He and Young (2006)	–	–	90.3

Results compercement with Zettlemoyer and Collins 2005 on Geo880 data set:

	Precision	Recall	F1
Single-Pass Parsing	95.49	83.2	88.93
Two-Pass Parsing	91.63	86.07	88.76
ZC05	96.25	79.29	86.95

Results

Results of the ATIS development set for the full algorithm and restricted versions of it without the GCC extensions:

	Precision	Recall	F1
Full Online Method	87.26	74.44	80.35
Without control features	70.33	42.45	52.95
Without relaxed word order	82.81	63.98	72.19
Without word insertion	77.31	56.94	65.58

Outline

- Introduction
- CCG
- The PCCG Model
- Learning
- Results
- Conclusion

Conclusion

- ✓ We introduced the concepts of lambda calculus and combinatory categorial grammar (CCG), which allows us to map sentences to their logical form.
- ✓ We presented two algorithms for parameter estimation in a log-linear parsing model: **gradient ascent** and **hidden variable structured perceptron.**
- ✓ We showed an online algorithm for learning CCG, which combines structured perceptron with lexical learning.
- ✓ We presented some experiments results that shows the system achieves significant accuracy improvements in both the ATIS and Geo880 domains, in compercement with previous works.