Parsing with Compositional Vector Grammars

BY RICHARD SOCHER, JOHN BAUER, CHRISTOPHER D. MANNING, ANDREW Y. NG

PRESENT BY YUNCHENG WU

Outline

Introduction

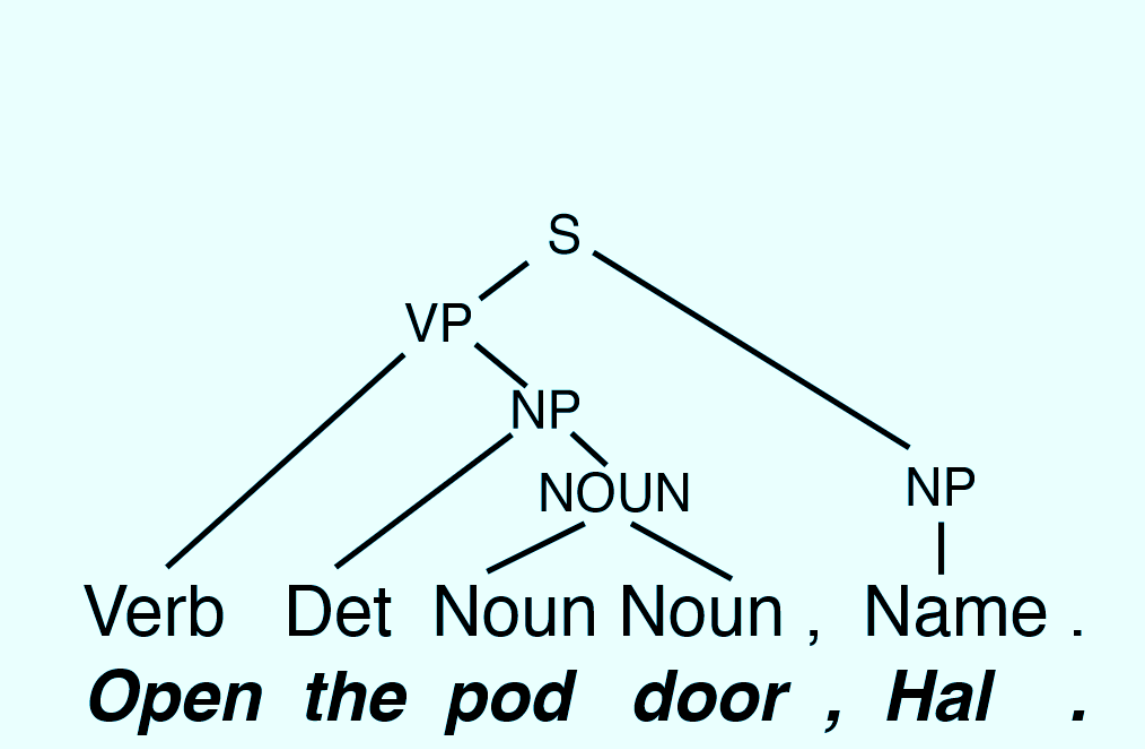
Related Works

Compositional Vector Grammars (this paper)

Experiments

Takeaways

Introduction



Definition

Syntactic parsing is the task that gives syntactic structure for sentences

Motivation

Directly useful applications:

- Grammar Checking

Intermediate stage for subsequent tasks:

- Semantic analysis
- Question answering
- Information extraction

Related Works

Improve discrete syntactic representations

Manual feature engineering (Klein and Manning, 2003a)

Split and merges the syntactic categories to maximize likelihood on the treebank. (Petrov et al. (2006))

Describe each category with a lexical item (head word), which is called lexicalized parsers. (Collins, 2003; Charniak, 2000)

Improve discrete syntactic representations - Problems

Subdividing category can only provide a very limited representation of phrase meaning and semantic similarity

Cannot capture semantic information, such as PP attachment

They ate udon with forks.

vs.

They ate udon with chicken.

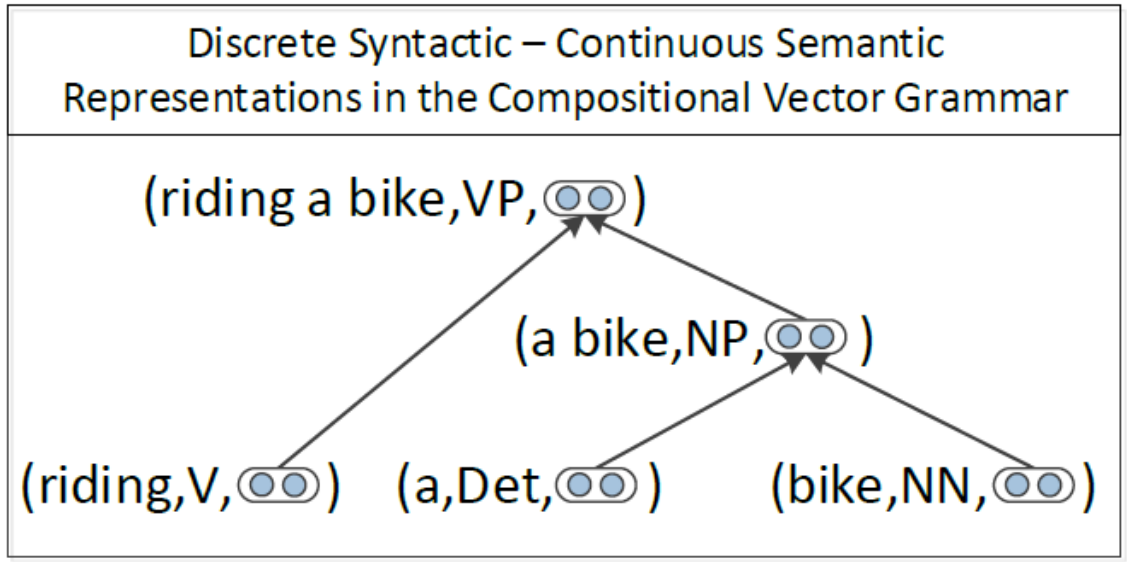
Deep learning and Recursive deep learning

Use neural network on sequence labeling and learning appropriate features. (Collobert and Weston, 2008)

Use neural networks on large scale parsing by estimating the probabilities of parsing decisions based on parsing history. (Henderson, 2003)

Use recursive neural network to re-rank possible phrase attachments in an incremental parser (Costa et al., 2003).

Compositional Vector Grammars (CVG)



Overview

CVG builds on top of a standard PCFG parser

CVG combine syntactic and semantic information in the form of distributional word vectors

In general, CVG merges idea from generative models and discriminative models

Word vector representations

Representation of words

- Learn **distributional word vectors** on neural language models

Representation of sentence:

- A sentence is an ordered list of m words
- For the **i^{th} word in the sentence**, the **i^{th} column of the matrix** stores the corresponding **word vector**
- Use a binary vector to retrieve all word vectors and **get an ordered list of (word, vector) pairs**

Max-Margin training objective for CVGs - Goal

Learn a function g which is parameterized by θ. $g_\theta(X) = Y$, where X is a set of given sentences and Y is a set of all possible labeled binary parse trees.

Max-Margin training objective for CVGs – Structured Margin Loss

During training, give the model a sentence and a correct parse tree y. The model returns a proposed parse tree $\hat{y}$

Measure discrepancy between trees by counting the number of nodes with incorrect span or label in the proposed parse tree

$\Delta(y_i, \hat{y}) = \sum_{d \in N(\hat{y})} \kappa 1\{d \notin N(y_i)\}$

- $\kappa = 0.1$ for all experiments

Max-Margin training objective for CVGs – Altogether

For a given set of training instances, we search for the function g_θ, with the smallest expected loss on a new sentence

$$g_\theta(x) = \arg\max_{\hat{y} \in Y(x)} s\big(CVG(\theta, x, \hat{y})\big)$$

- Y: a set of all possible labeled binary parse trees.
- s(): scoring function, more details later
- CVG(): find the parse tree

Max-Margin training objective for CVGs – Altogether

Highest scoring tree will be the correct tree $g(x_i) = y_i$

- Its score is larger up to a margin to other possible trees
- $s\big(CVG(\theta, x_i, y_i)\big) \geq s\big(CVG(\theta, x_i, \hat{y})\big) + \Delta(y_i, \hat{y})$

Max-Margin training objective for CVGs – Training objective

For entire dataset:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} r_i(\theta) + \frac{\lambda}{2}\left|\left|\theta\right|\right|_2^2$$

For each training data object:

$$r_i(\theta) = \max_{\hat{y} \in Y(x_i)} \left(s\left(CVG(x_i, \hat{y})\right) + \Delta(y_i, \hat{y}) \right)$$

To minimize the function

- The score of the correct tree y_i is increased
- The highest scoring incorrect tree $\hat{y}$ is decreased

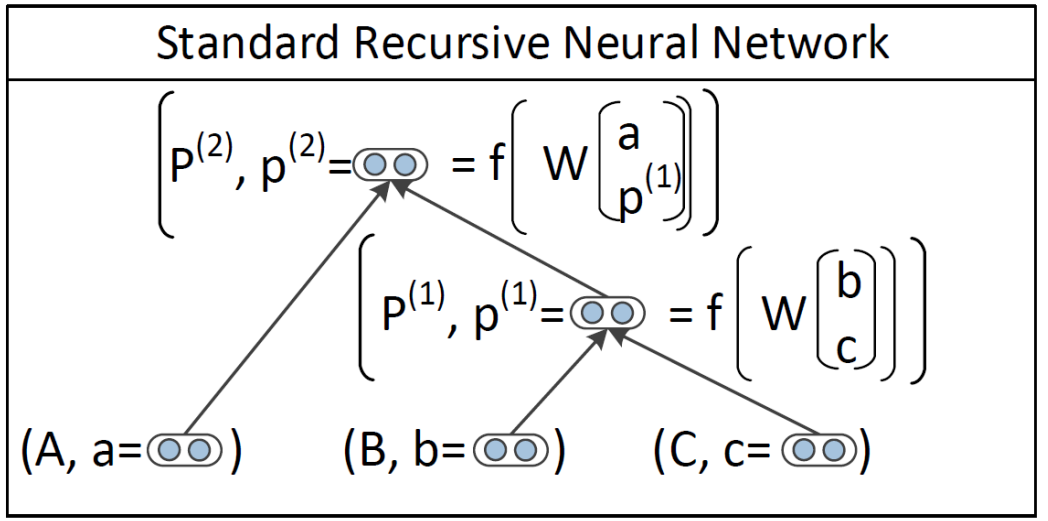
Scoring trees with CVGs – Standard RNN

Ignores all POS tags and syntactic categories

Each non-terminal node even with different categories is associated with the same neural network

Compute activations for each node from the bottom up

- Concatenate children vectors
- Multiple the vector by the parameter weights of RNN
- Apply an element-wise nonlinearity function f=tanh to output the vector



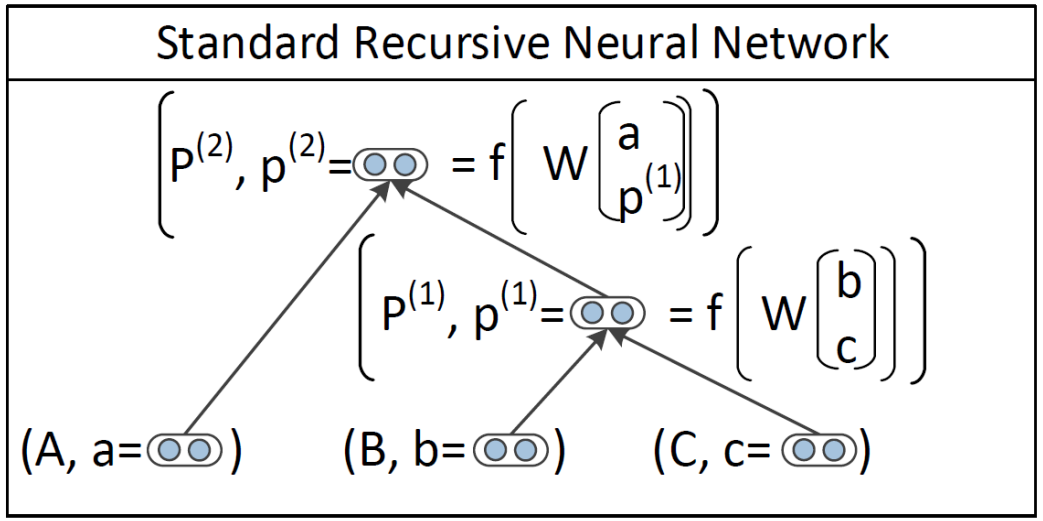
Scoring trees with CVGs – Standard RNN

Scoring the syntactic constituency of a parent

- $s\left(p^{(i)}\right) = v^T p^{(i)}$
- V is a vector of parameters that need to be trained
- Score is used to find the highest scoring tree

Disadvantage:

- One composition function cannot fully capture all compositions



Scoring trees with CVGs – Standard RNN Alternatives

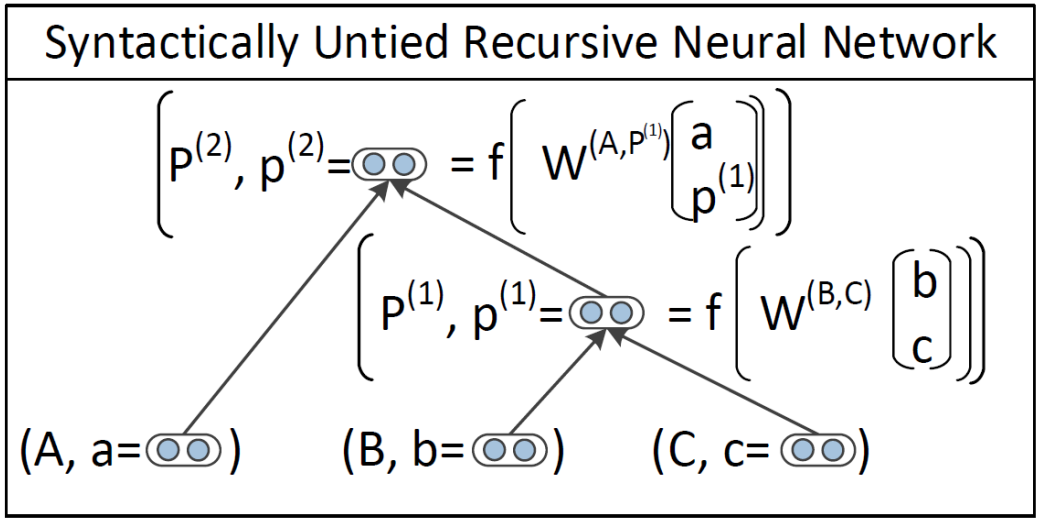
Two-layered RNN:

- More expressive
- Hard to train because it is very deep
- Vanishing gradient problem
- Number of model parameters explodes
- Composition functions do not capture the syntactic commonalities between similar POS tags or syntactic categories

Scoring trees with CVGs – SU-RNN

CVG: Combine discrete syntactic rule probabilities and continuous vector compositions

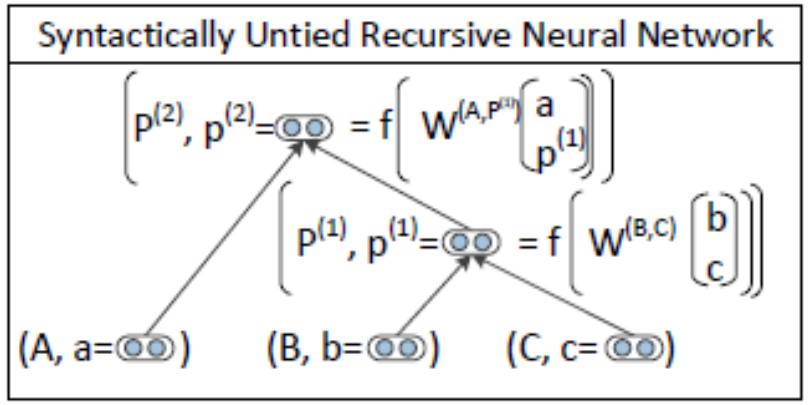
Syntactic categories of the children determine what composition function to use to compute their parents

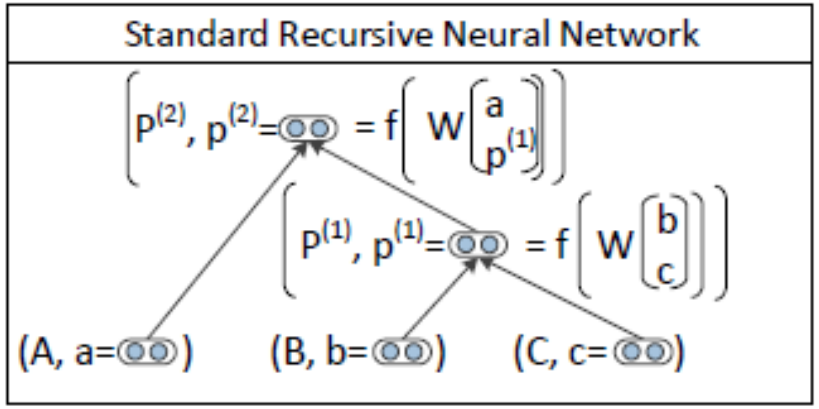


Scoring trees with CVGs – SU-RNN

A dedicated composition function for each rule can well capture common composition processes.

- Example:
  - NP should be similar to its head noun and little influenced by a determiner.
  - In an adjective modification both words considerably determine the meaning of the phrase.





Scoring trees with CVGs – SU-RNN vs. RNN

- Weight matrix
  - For each category combinations, CVG is parameterized by a weight matrix W
  - Standard RNN parameterized by a single weight matrix W

- Scoring
  - SU-RNN Scoring:

$$s(p^{(1)}) = (v^{(B,C)})^T p^{(1)} + \log P(P_1 \rightarrow B\ \ C)$$

  - Standard RNN Scoring:

$$s(p^{(i)}) = v^T p^{(i)}$$

Parsing with CVGs - Approach

Two bottom-up passes through the parsing chart

First pass

- Only the base PCFG to run CKY dynamic programming through the tree
- Keep top 200 best parses

Second pass

- Search in 200 best parse trees with full CVG model and select the best tree

Training SU-RNNs – General idea

First stage

- Train base PCFG
- Cache top trees

Second stage

- Train SU-RNN on cached top trees

Training SU-RNNs – Details

Objective function

$$r_i(\theta) = \max_{\hat{y} \in Y(x_i)} \left(s\left(CVG(x_i, \hat{y})\right) + \Delta(y_i, \hat{y}) \right)$$

Minimize the objective by

- Increasing the scores of the correct tree's constituents
- Decreasing the scores of the highest scoring incorrect tree

Derivatives are computed via backpropagation

Specific for SU-RNN

- Each node has a category. Derivatives at each node only add to overall derivative of the specific matrix at that node

Experiments

Setup

Dataset: Penn Treebank WSJ

Hyperparameters

- PCFG modification
  - Decrease the state splitting of PCFG grammar
  - Ignore all category splits
  - 948 transformation matrices and scoring vectors
- Regularization $\lambda = 10^{-4}$
- AdaGrad's learning rate $\alpha = 0.1$
- Vector dimensions = 25
  - F1 score is higher than higher dimensions
  - Computation complexity is better than higher dimensions

Parser	dev (all)	test$\leq$ 40	test (all)
Stanford PCFG	85.8	86.2	85.5
Stanford Factored	87.4	87.2	86.6
Factored PCFGs	89.7	90.1	89.4
Collins			87.7
SSN (Henderson)			89.4
Berkeley Parser			90.1
CVG (RNN)	85.7	85.1	85.0
CVG (SU-RNN)	91.2	91.1	90.4
Charniak-SelfTrain			91.0
Charniak-RS			92.1

Result

Accuracy

- Dev set F1: 91.2%
- Final test set F1: 90.4%

Speed

- 1320s CVG vs. 1600s currently published Stanford factored parser.

Model analysis – Analysis of error type

Largest improved performance over Stanford factored parser: **correct placement of PP phrases.**

Error Type	Stanford	CVG	Berkeley	Char-RS
PP Attach	1.02	0.79	0.82	0.60
Clause Attach	0.64	0.43	0.50	0.38
Diff Label	0.40	0.29	0.29	0.31
Mod Attach	0.37	0.27	0.27	0.25
NP Attach	0.44	0.31	0.27	0.25
Co-ord	0.39	0.32	0.38	0.23
1-Word Span	0.48	0.31	0.28	0.20
Unary	0.35	0.22	0.24	0.14
NP Int	0.28	0.19	0.18	0.14
Other	0.62	0.41	0.41	0.50

Model analysis – Semantic transfer for PP attachments

Training data:

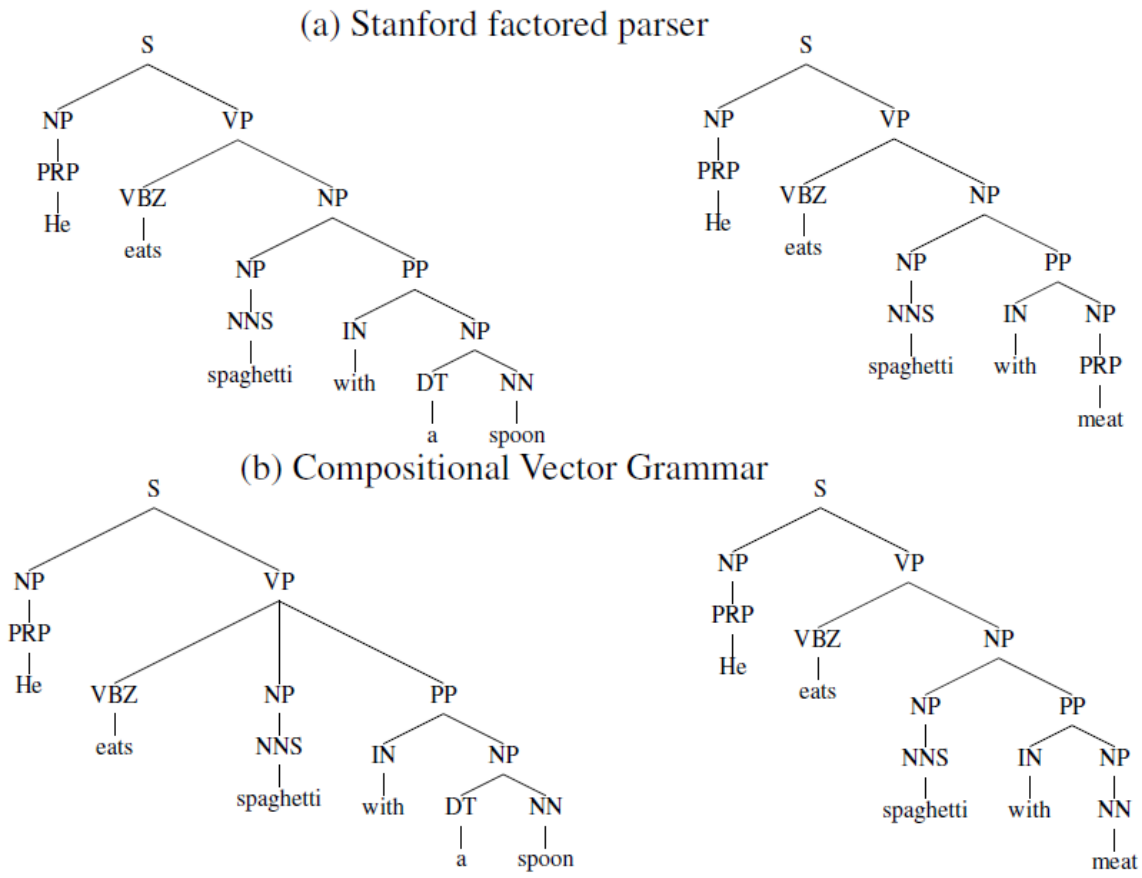
- He eats spaghetti with a fork.
- She eats spaghetti with pork.

Test data:

- He eats spaghetti with a spoon.
- He eats spaghetti with meat.

Parsers

- Stanford parser
- CVG



Model analysis – Semantic transfer for PP attachments

Initial state: both parsers incorrectly attach PP.

After training: CVG is able to correctly parse PP attachments because it can capture semantic information in word vectors.

Takeaways

Takeaways

CVG combines the speed of small-state PCFGs with semantic richness of neural words representations and compositional phrase vectors.

Compositional vectors are learned with SU-RNN.

Model chooses different composition functions for parent node based on syntactic categories of its children.