Tree(t)-Shaped Models

Socher et al, Dyer et al & Andreas et al

By Shinjini Ghosh, Ian Palmer, Lara Rakocevic

Format

5 mins	40 mins	15 + 10 mins	25 mins	10 mins
Introduction	**Papers 1 & 2**	**Breakout Rooms + Break**	**Paper 3**	**Breakout Rooms**
How the three papers tie together	Parsing with CVGs RNNGs	Discussion	Neural Networks for Question Answering	Discussion

What were the shared high level concepts?

Things to think about

- Benefits of continuous representations
- Different methods of integrating compositionality and neural models
- Ways to take advantage of hierarchical structures

Parsing with CVGs (Compositional Vector Grammars)

Socher, Bauer, Manning, Ng (2013)

Presented by Shinjini Ghosh

Motivation

- Syntactic parsing is crucial
- How can we learn to parse and represent phrases as both discrete categories and continuous vectors?

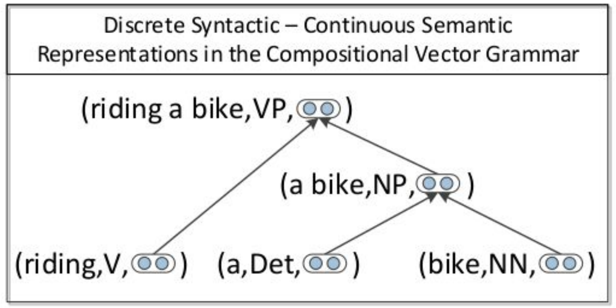

Discrete Syntactic – Continuous Semantic Representations in the Compositional Vector Grammar
(riding a bike,VP,)
(a bike,NP,)
(riding,V,)
(a,Det,)
(bike,NN,)

Figure 1: Example of a CVG tree with (category,vector) representations at each node. The vectors for nonterminals are computed via a new type of recursive neural network which is conditioned on syntactic categories from a PCFG.

Background

- Discrete Representations
  - Manual feature engineering - Klein & Manning 2003
  - Split into subcategories - Petrov et al. 2006
  - Lexicalized parsers - Collins 2003, Charniak 2000
  - Combination - Hall & Klein 2012
- Recursive Deep Learning
  - RNNs with words as one-on vectors - Elman 1991
  - Sequence labeling - Clobert & Weston 2008
  - Parsing based on history - Henderson 2003
  - RNN + re-rank phrases - Costa et al. 2003
  - RNN + re-rank parses - Menchetti et al. 2005

Compositional Vector Grammars

- Model to jointly find syntactic structure and capture compositional semantic information
- Intuition: language is fairly regular, and can be captured by well-designed syntactic patterns…but there are fine-grained semantic factors influencing parsing. E.g., *They ate udon with chicken* vs *They ate udon with forks*
- So, give parser access to both distributional word vectors and compute compositional semantic vector representations for longer phrases

Word Vector Representation

- Occurrence statistics and context - Turney and Pantel, 2010
- Neural LM - embedding in *n*-dimensional feature space - Bengio et al. 2003
  E.g., *king - man + woman = queen* (Mikolov et al. 2013)
- Sentence *S* is an ordered list of (word, vector) pairs

$$x = ((w_1, a_{w_1}), \ldots, (w_m, a_{w_m})).$$

Max-Margin Training Objective for CVGs

- Structured margin loss

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

- Parsing function

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

- Objective function

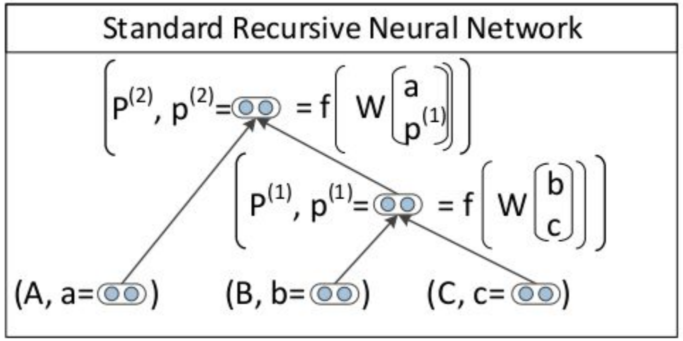
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} r_i(\theta) + \frac{\lambda}{2} \|\theta\|_2^2, \text{ where}$$

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

Scoring Trees with CVGs

- Syntactic categories of children determine what composition function to use for computing the vector of their parents
- For example, an NP should be similar to its N head and not much to its Det
- So, the CVG uses a syntactically-untied (SU-RNN) which has a set of weights, of size the # of sibling category combinations in the PCFG

Scoring Trees with CVGs


Standard Recursive Neural Network
$P^{(2)}, p^{(2)} = \; = f\left(W\begin{bmatrix}a\\p^{(1)}\end{bmatrix}\right)$
$P^{(1)}, p^{(1)} = \; = f\left(W\begin{bmatrix}b\\c\end{bmatrix}\right)$
(A, a=)
(B, b=)
(C, c=)

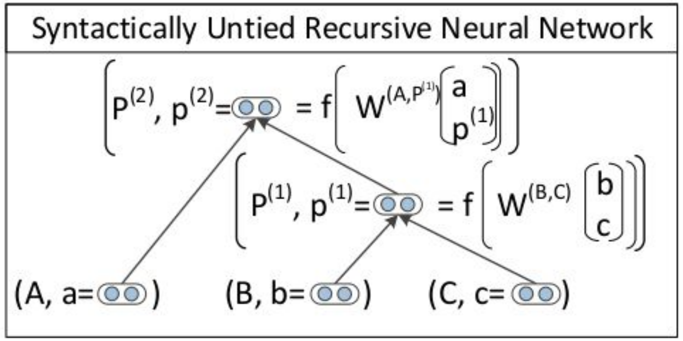

Syntactically Untied Recursive Neural Network
$P^{(2)}, p^{(2)} = \; = f\left(W^{(A,P^{(1)})}\begin{bmatrix}a\\p^{(1)}\end{bmatrix}\right)$
$P^{(1)}, p^{(1)} = \; = f\left(W^{(B,C)}\begin{bmatrix}b\\c\end{bmatrix}\right)$
(A, a=)
(B, b=)
(C, c=)

Figure 2: An example tree with a simple Recursive Neural Network: The same weight matrix is replicated and used to compute all non-terminal node representations. Leaf nodes are n-dimensional vector representations of words.

Figure 3: Example of a syntactically untied RNN in which the function to compute a parent vector depends on the syntactic categories of its children which we assume are given for now.

Parsing with CVGs

- score(CVG) = ∑ score(node)

$$s(\texttt{CVG}(\theta, x, \hat{y})) = \sum_{d \in N(\hat{y})} s\left(p^d\right).$$

- If |*sentence*| = n, |possible binary trees| = Catalan(n)
  ⇒ finding global maximum is exponentially hard
- Compromise: Two-pass algorithm
  - Use base PCFG to run CKY DP through the tree and store top 200 best parses
  - Beam search with full CVG
- Since each SU-RNN matrix multiplication only needs child vectors and not whole tree, this is still fairly fast

Training SU-RNNs

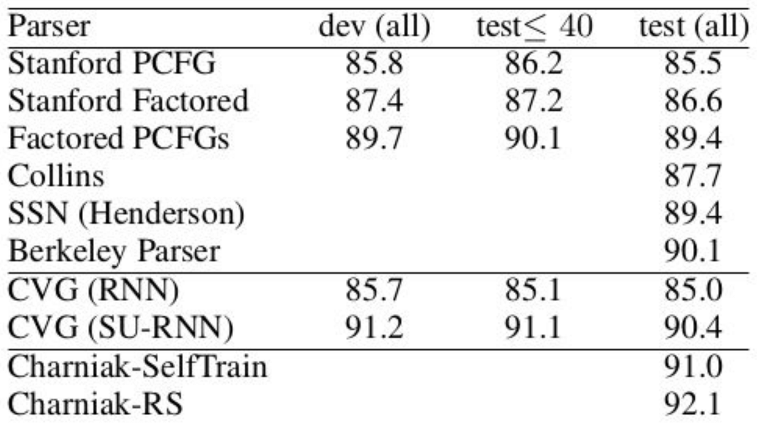
- Two stage training
  - Base PCFG trained and top trees cached
  - SU-RNN trained conditioned on the PCFG

Subgradient Methods and AdaGrad

- Generalize gradient ascent using subgradient method
- Uses diagonal variant of AdaGrad to minimize objective

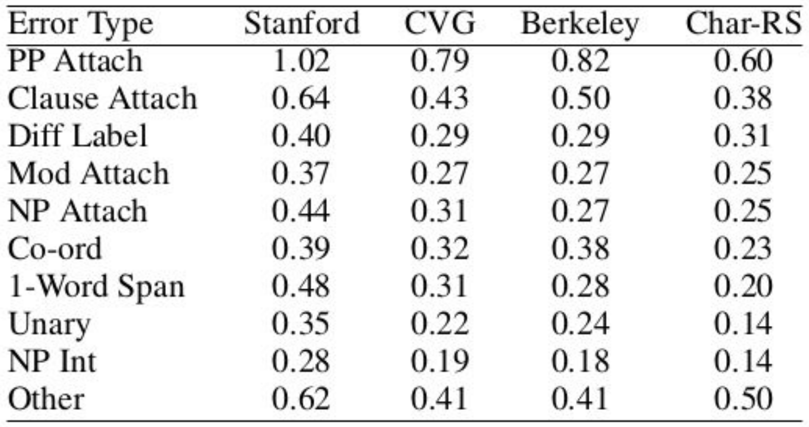
Experimentation

- Cross-validating using first 20 files of WSJ Section 22
- 90.44% accuracy on final test set (WSJ Section 23)



Parser	dev (all)	test≤ 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

Table 1: Comparison of parsers with richer state representations on the WSJ. The last line is the self-trained re-ranked Charniak parser.

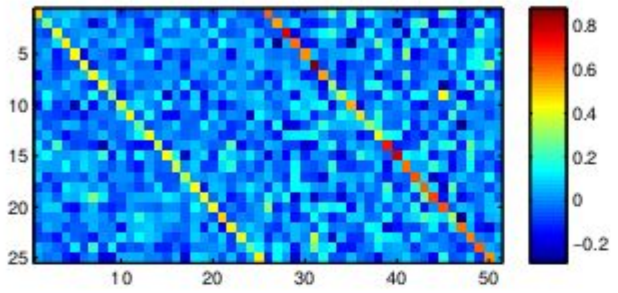


Error Type	Stanford	CVG	Berkeley	Char-RS
PP Attach	1.02	0.79	0.82	0.60
Clause Attach	0.64	0.43	0.50	0.38
Diff Label	0.40	0.29	0.29	0.31
Mod Attach	0.37	0.27	0.27	0.25
NP Attach	0.44	0.31	0.27	0.25
Co-ord	0.39	0.32	0.38	0.23
1-Word Span	0.48	0.31	0.28	0.20
Unary	0.35	0.22	0.24	0.14
NP Int	0.28	0.19	0.18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

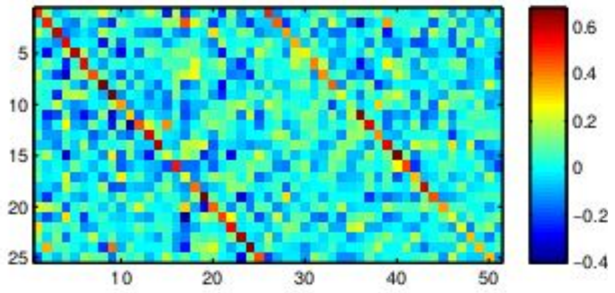
Table 2: Detailed comparison of different parsers.

Model Analysis: Composition Matrices

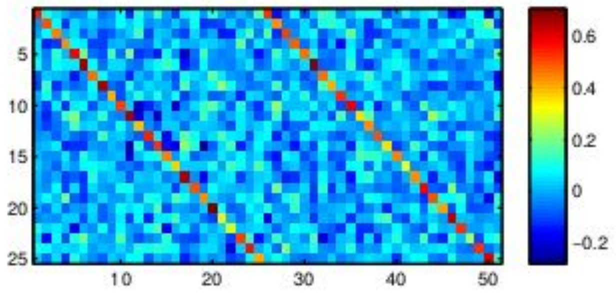
- Model learns a soft vectorized notion of head words:
  - Head words are given larger weights and importance when computing the parent vector
  - For the matrices combining siblings with categories VP:PP, VP:NP and VP:PRT, the weights in the part of the matrix which is multiplied with the VP child vector dominates
  - Similarly NPs dominate DTs



DT-NP



VP-NP



ADJP-NP

Model Analysis: Semantic Transfer for PP Attachments

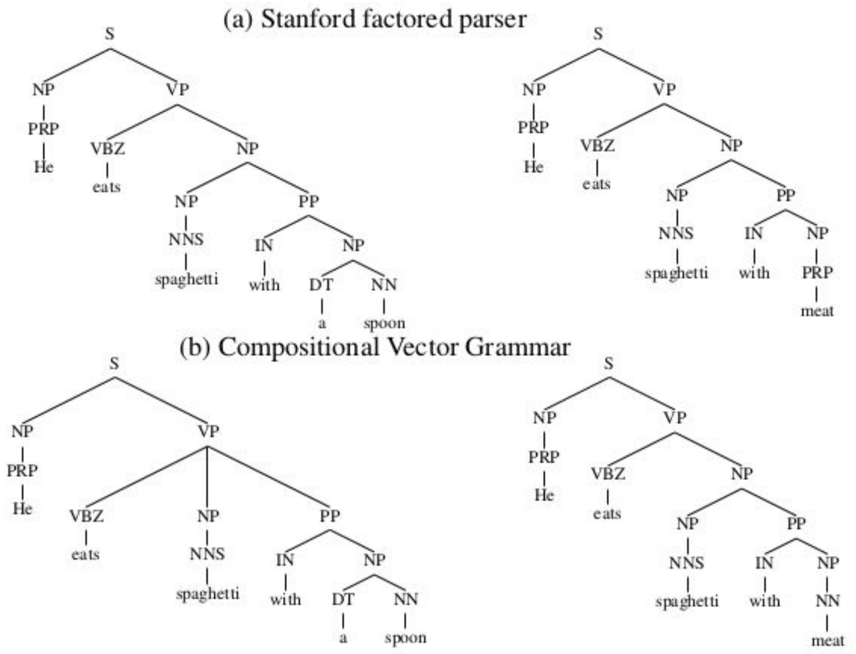

(a) Stanford factored parser
S NP VP PRP VBZ NP He eats NP PP NNS IN NP spaghetti with DT NN a spoon
S NP VP PRP VBZ NP He eats NP PP NNS IN NP spaghetti with PRP meat
(b) Compositional Vector Grammar
S NP VP PRP VBZ NP PP He eats NNS IN NP spaghetti with DT NN a spoon
S NP VP PRP VBZ NP He eats NP PP NNS IN NP spaghetti with NN meat

Figure 4: Test sentences of semantic transfer for PP attachments. The CVG was able to transfer semantic word knowledge from two related training sentences. In contrast, the Stanford parser could not distinguish the PP attachments based on the word semantics.

Conclusion

- Paper introduces CVGs
- Parsing model that combines speed of small state PCFGs with semantic richness of neural word representations and compositional phrase vectors
- Learns compositional vectors using new syntactically untied RNN
- Linguistically more plausible since it chooses composition function for parent node based on children
- 90.44% F1 on full WSJ test set
- 20% faster than previous Stanford parser

Discussion

- Is basing composition functions just on children nodes enough? (Garden path sentences? Embedding? Recursive nesting?)
- Is this really incorporating both syntax and semantics *at once*? Or merely a two-pass algorithm?
- Other ways to combine syntax and semantics?

Recurrent Neural Net Grammars

Dyer et al

Why not just Sequential RNNs?

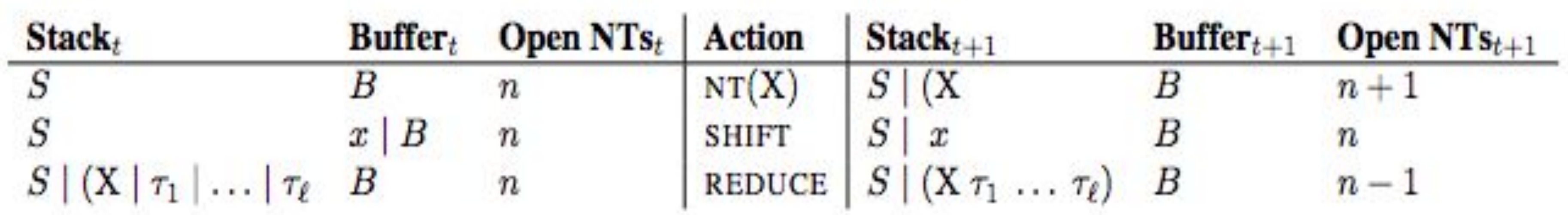
“Relationships among words are largely organized in terms of latent nested structure rather than sequential surface order”

Definition of RNNG

A triple consisting of:

- N: finite set of nonterminal symbols
- Σ: finite set of terminal symbols st $N \cap \Sigma = \varnothing$
- Θ: collection of neural net parameters

Parser Transitions



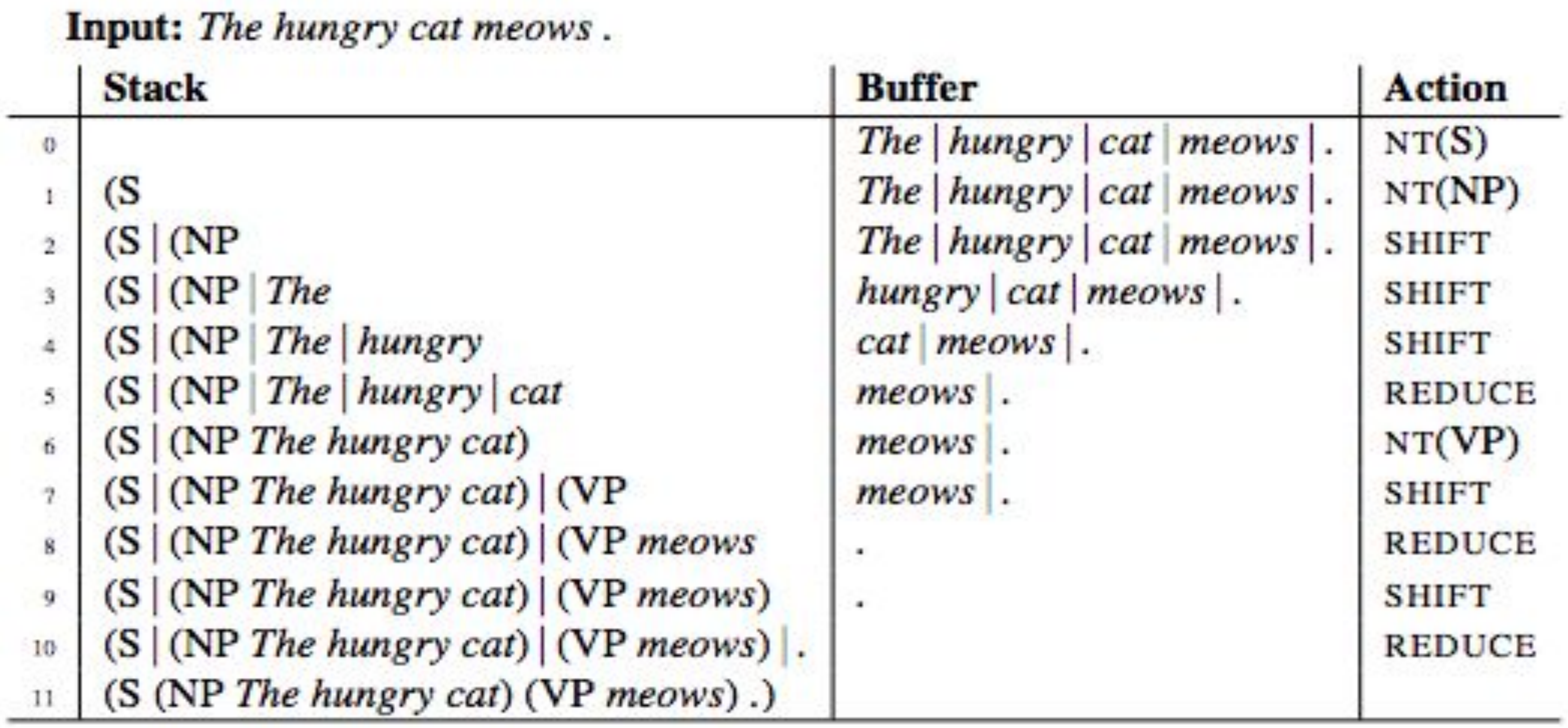
$\textbf{Stack}_t$	$\textbf{Buffer}_t$	$\textbf{Open NTs}_t$	**Action**	$\textbf{Stack}_{t+1}$	$\textbf{Buffer}_{t+1}$	$\textbf{Open NTs}_{t+1}$
S	B	n	NT(X)	$S \mid (\mathrm{X}$	B	$n+1$
S	$x \mid B$	n	SHIFT	$S \mid x$	B	n
$S \mid (\mathrm{X} \mid \tau_1 \mid \ldots \mid \tau_\ell$	B	n	REDUCE	$S \mid (\mathrm{X}\ \tau_1\ \ldots\ \tau_\ell)$	B	$n-1$

NT(X) = introduces an “open nonterminal” X onto the top of the stack.

SHIFT = removes the terminal symbol x from the front of the input buffer, and pushes it onto the top of the stack

REDUCE = repeatedly pops completed subtrees or terminal symbols from the stack until an open NT is encountered, then pops NT and uses as label of a new constituent with popped subtrees as children

Example of Top-Down Parsing in action



Input: *The hungry cat meows .*

	Stack	Buffer	Action
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

Constraints on Parsing

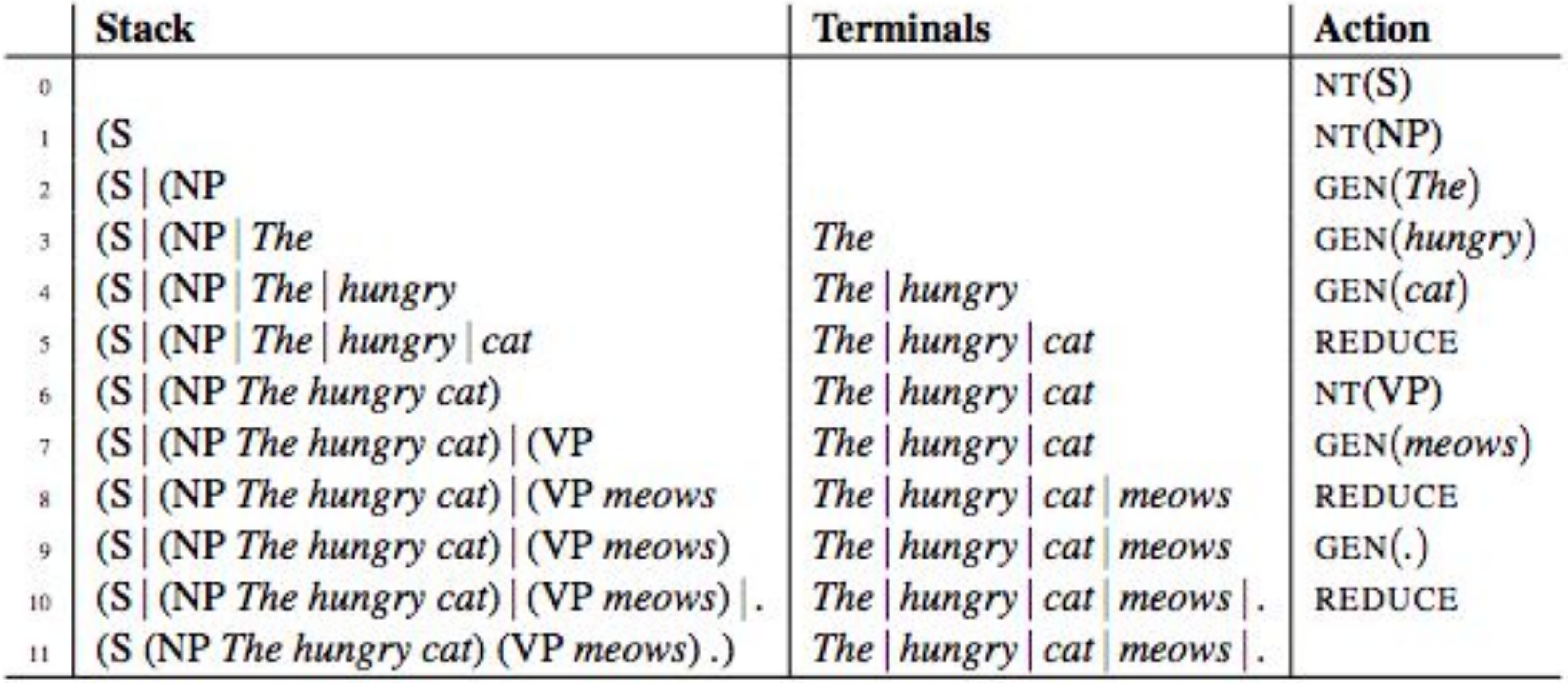
- The NT(X) operation can only be applied if B is not empty and $n < 100$. 4
- The SHIFT operation can only be applied if B is not empty and $n \geq 1$.
- The REDUCE operation can only be applied if the top of the stack is not an open nonterminal symbol.
- The REDUCE operation can only be applied if $n \geq 2$ or if the buffer is empty

Generator Transitions

Start with parser transitions and add in the following changes:

1. there is no input buffer of unprocessed words, rather there is an output buffer (T)
2. instead of a SHIFT operation there are GEN(x) operations which generate terminal symbol $x \in \Sigma$ and add it to the top of the stack and the output buffer

Example of Generation Sequence



	Stack	Terminals	Action
0			NT(S)
1	(S		NT(NP)
2	(S | (NP		GEN(*The*)
3	(S | (NP | *The*	*The*	GEN(*hungry*)
4	(S | (NP | *The* | *hungry*	*The* | *hungry*	GEN(*cat*)
5	(S | (NP | *The* | *hungry* | *cat*	*The* | *hungry* | *cat*	REDUCE
6	(S | (NP *The hungry cat*)	*The* | *hungry* | *cat*	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*The* | *hungry* | *cat*	GEN(*meows*)
8	(S | (NP *The hungry cat*) | (VP *meows*	*The* | *hungry* | *cat* | *meows*	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	*The* | *hungry* | *cat* | *meows*	GEN(.)
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	*The* | *hungry* | *cat* | *meows* | .	REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)	*The* | *hungry* | *cat* | *meows* | .	

Constraints on Generation

- The GEN(x) operation can only be applied if $n \geq 1$.
- The REDUCE operation can only be applied if the top of the stack is not an open nonterminal symbol and $n \geq 1$.

Transition Sequences from Trees

- Any parse tree can be converted to a sequence of transitions via a depth-first, left-to-right traversal of a parse tree.
- Since there is a unique depth-first, left to-right traversal of a tree, there is exactly one transition sequence of each tree

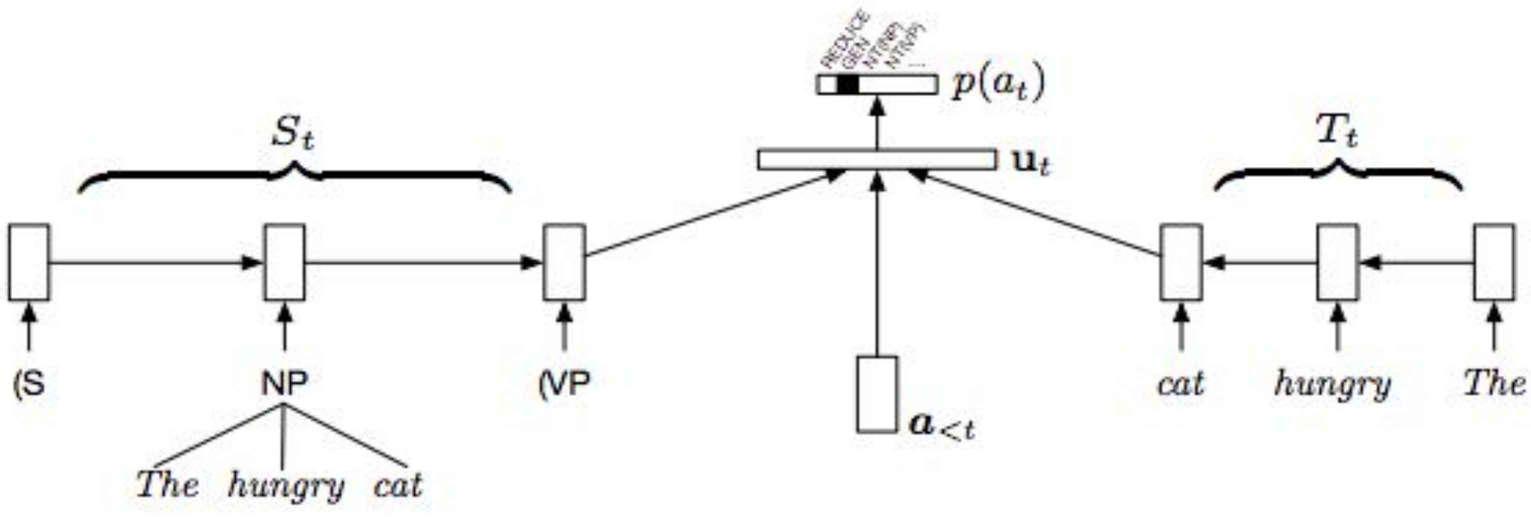
Generative Model

$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} p(a_t \mid \boldsymbol{a}_{<t})$$

$$= \prod_{t=1}^{|\boldsymbol{a}(\boldsymbol{x},\boldsymbol{y})|} \frac{\exp \mathbf{r}_{a_t}^\top \mathbf{u}_t + b_{a_t}}{\sum_{a' \in \mathcal{A}_G(T_t, S_t, n_t)} \exp \mathbf{r}_{a'}^\top \mathbf{u}_t + b_{a'}}$$

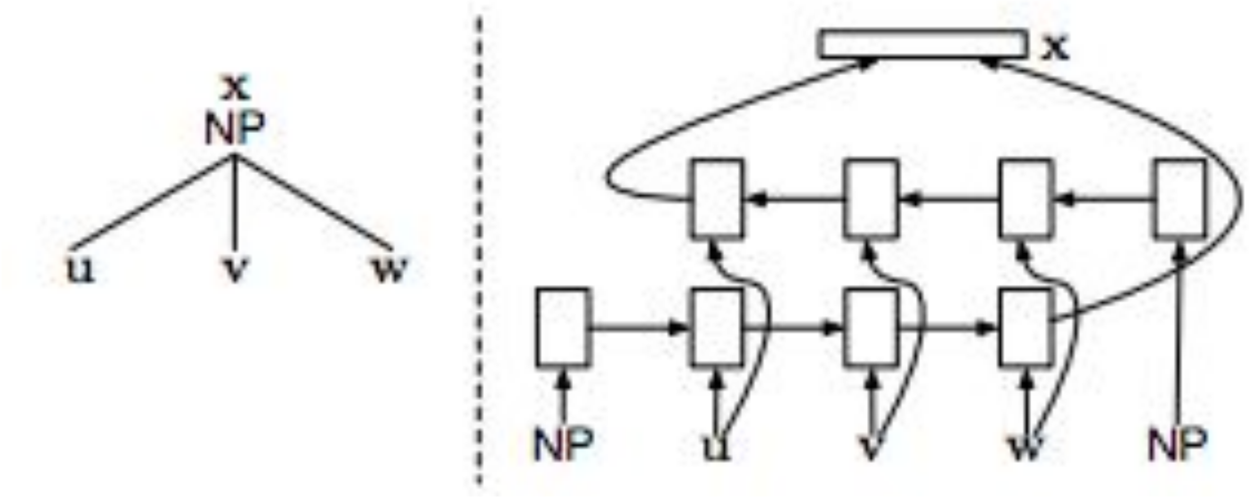
Where $\mathbf{u}_t = \tanh(\mathbf{W}[\mathbf{o}_t; \mathbf{s}_t; \mathbf{h}_t] + \mathbf{c})$

Generative Model - Neural Architecture



Neural architecture for defining a distribution over a_t given representations of the stack, output buffer and history of actions.

Syntactic Composition Function



Composition function based on bidirectional LSTMS

Evaluating Generative Model

To evaluate as LM:

- Compute marginal probability

To evaluate as parser:

- Find MAP parse tree ➔ tree y that maximizes joint distribution defined by generative model

Inference via Importance Sampling

Uses conditional proposal distribution $q(x \mid y)$ with following properties:

1. $p(x, y) > 0 \Rightarrow q(y \mid x) > 0$
2. samples $y \sim q(y \mid x)$ can be obtained efficiently
3. the conditional probabilities $q(y \mid x)$ of these samples are known

Discriminative parser fulfills these properties, so this is used as the proposal distribution.

Deriving Estimator ...

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} q(\boldsymbol{y} \mid \boldsymbol{x}) w(\boldsymbol{x}, \boldsymbol{y}) \\ &= \mathbb{E}_{q(\boldsymbol{y} \mid \boldsymbol{x})} w(\boldsymbol{x}, \boldsymbol{y}). \end{aligned}$$

Where “importance weights” w(x,y) = p(x,y) / q(y I x)

… then replace the expectation with it’s Monte Carlo estimate

$$\boldsymbol{y}^{(i)} \sim q(\boldsymbol{y} \mid \boldsymbol{x}) \quad \text{for } i \in \{1, 2, \ldots, N\}$$

$$\mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})} w(\boldsymbol{x}, \boldsymbol{y}) \overset{\text{MC}}{\approx} \frac{1}{N} \sum_{i=1}^{N} w(\boldsymbol{x}, \boldsymbol{y}^{(i)})$$

Experimental Setup

Discriminative Model:

- Hidden dimensions of 128, 2 Layer LSTMs

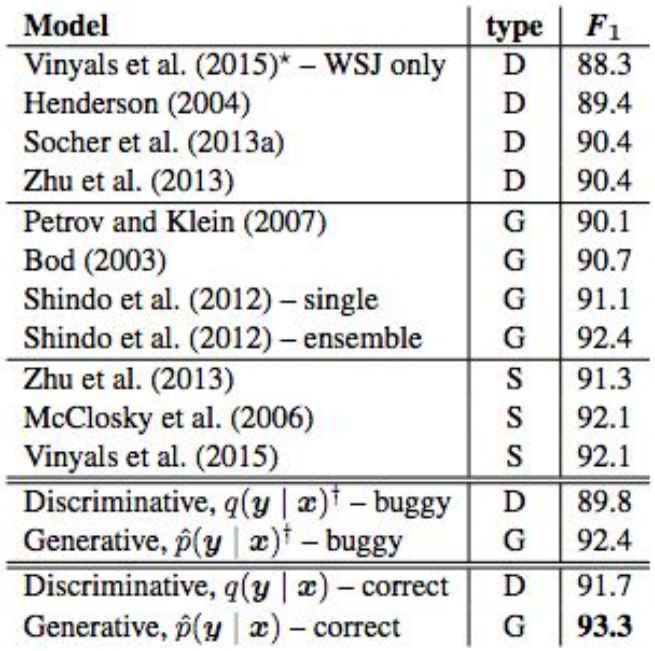
Generative Model:

- Hidden dimensions of 256, 2 Layer LSTMs

Both

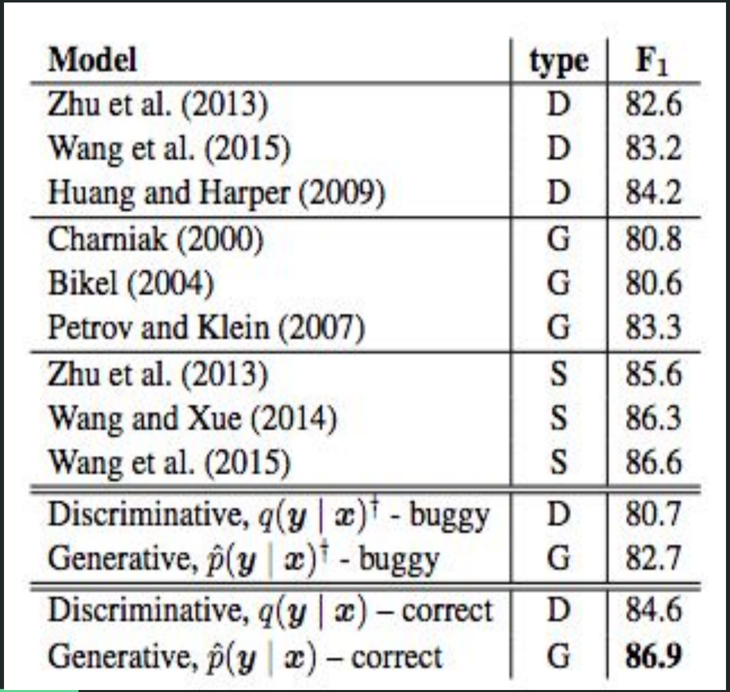
- Dropout rate to maximize validation set likelihood
- For training, SGD with learning rate of 0.1

English Parsing Results



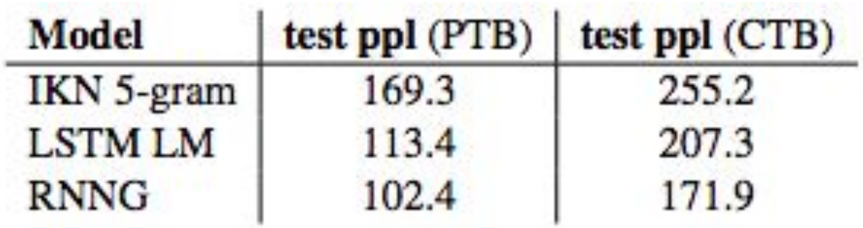
Model	type	F_1
Vinyals et al. (2015)* – WSJ only	D	88.3
Henderson (2004)	D	89.4
Socher et al. (2013a)	D	90.4
Zhu et al. (2013)	D	90.4
Petrov and Klein (2007)	G	90.1
Bod (2003)	G	90.7
Shindo et al. (2012) – single	G	91.1
Shindo et al. (2012) – ensemble	G	92.4
Zhu et al. (2013)	S	91.3
McClosky et al. (2006)	S	92.1
Vinyals et al. (2015)	S	92.1
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ – buggy	D	89.8
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ – buggy	G	92.4
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	91.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**93.3**

Chinese Parsing Results



Model	type	F_1
Zhu et al. (2013)	D	82.6
Wang et al. (2015)	D	83.2
Huang and Harper (2009)	D	84.2
Charniak (2000)	G	80.8
Bikel (2004)	G	80.6
Petrov and Klein (2007)	G	83.3
Zhu et al. (2013)	S	85.6
Wang and Xue (2014)	S	86.3
Wang et al. (2015)	S	86.6
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ - buggy	D	80.7
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})^\dagger$ - buggy	G	82.7
Discriminative, $q(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	D	84.6
Generative, $\hat{p}(\boldsymbol{y} \mid \boldsymbol{x})$ – correct	G	**86.9**

Language Model Results



Model	**test ppl** (PTB)	**test ppl** (CTB)
IKN 5-gram	169.3	255.2
LSTM LM	113.4	207.3
RNNG	102.4	171.9

Takeaways

1. Effective at both language modeling and parsing
2. Generative model obtains :

   a. Best known parsing results using a single supervised generative model and

   b. Better perplexities in LM than state-of-the-art sequential LSTM LMs
3. Parsing with generative model better than with discriminative model

Discussion

- Why does the discriminative model perform worse than the generative model?
- Ways to extend this, outlook for future uses?
- What structural difference in English vs Chinese grammar that might be contributing to a higher accuracy in parsing?

Learning to Compose Neural Networks for Question Answering

Andreas et. al. 2016

Presented by Ian Palmer

Motivation: We want to interact with machines via natural language (Q&A)

Database QA

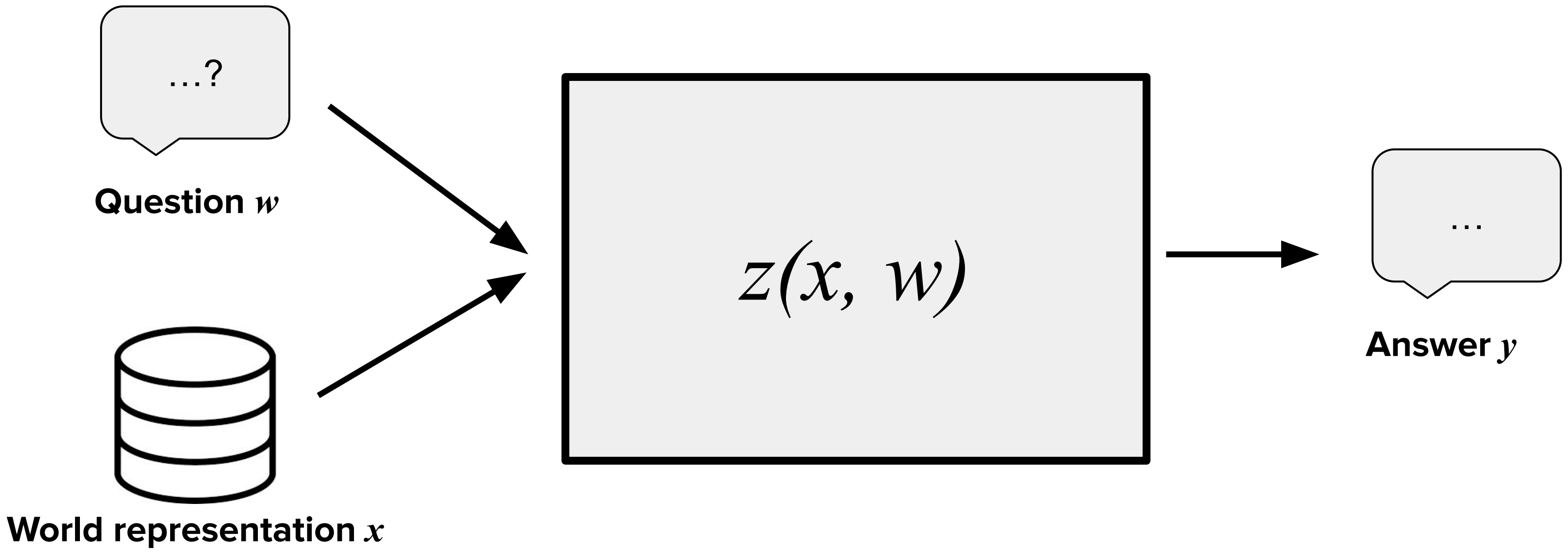
- Logical forms
  - Train on logical form examples[1] or QA pairs[2]
- Neural models
  - Shared embedding space[3]
  - Attention-based models[4]

Visual QA

- RNN-based approaches[5]
- Attention-based models[6]

1. Wong and Mooney 2007; 2. Kwiakowski et. al. 2010; 3. Bordes et. al. 2014; 4. Hermann et. al. 2015;
5. Ren et. al. 2015; 6. Yang et. al. 2015

...but all of these approaches have one thing in common

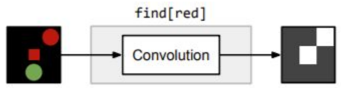


Can we allow z to vary?

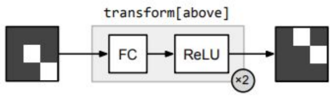
Neural Module Networks (Andreas et. al. 2016)

- Define a network layout predictor P, which maps from strings to network layouts
- P has a toolbox of modules it can construct networks from

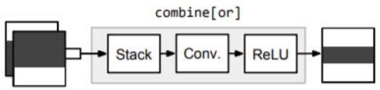
Find
Image → Attention



Transform
Attention → Attention



Combine
Attn x Attn → Attn

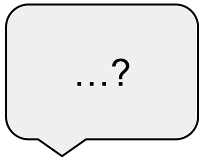


Describe
Image x Attention → Label

Measure
Attention → Label

- Extract model layout from $P(w)$, then compute $p(y \mid w, x; \theta)$

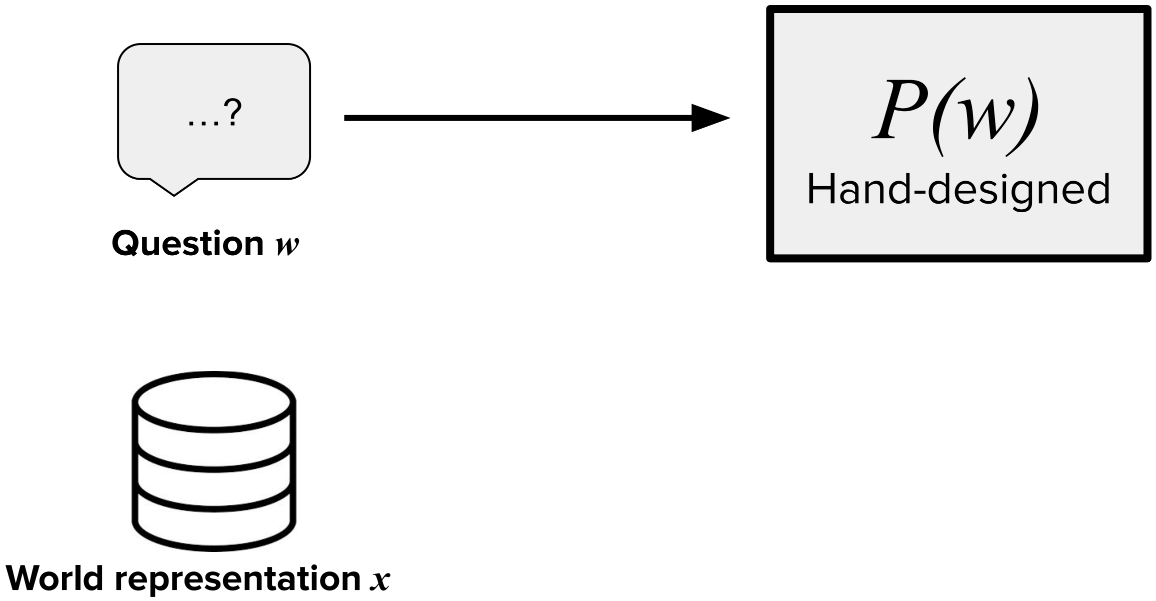
Revised process



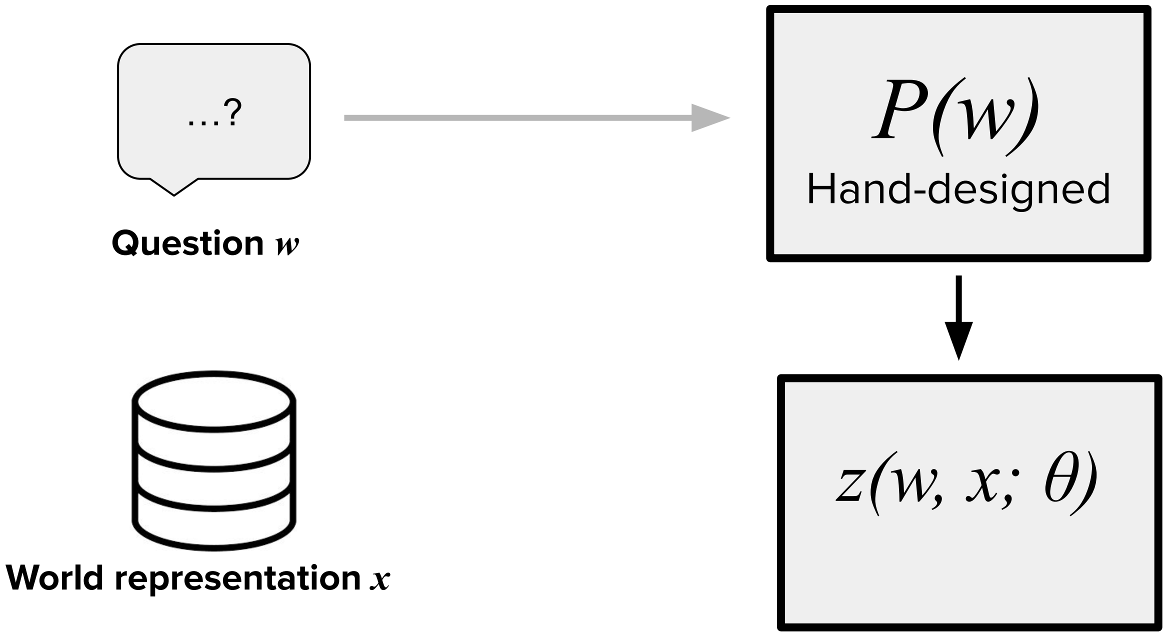
Question *w*

World representation *x*

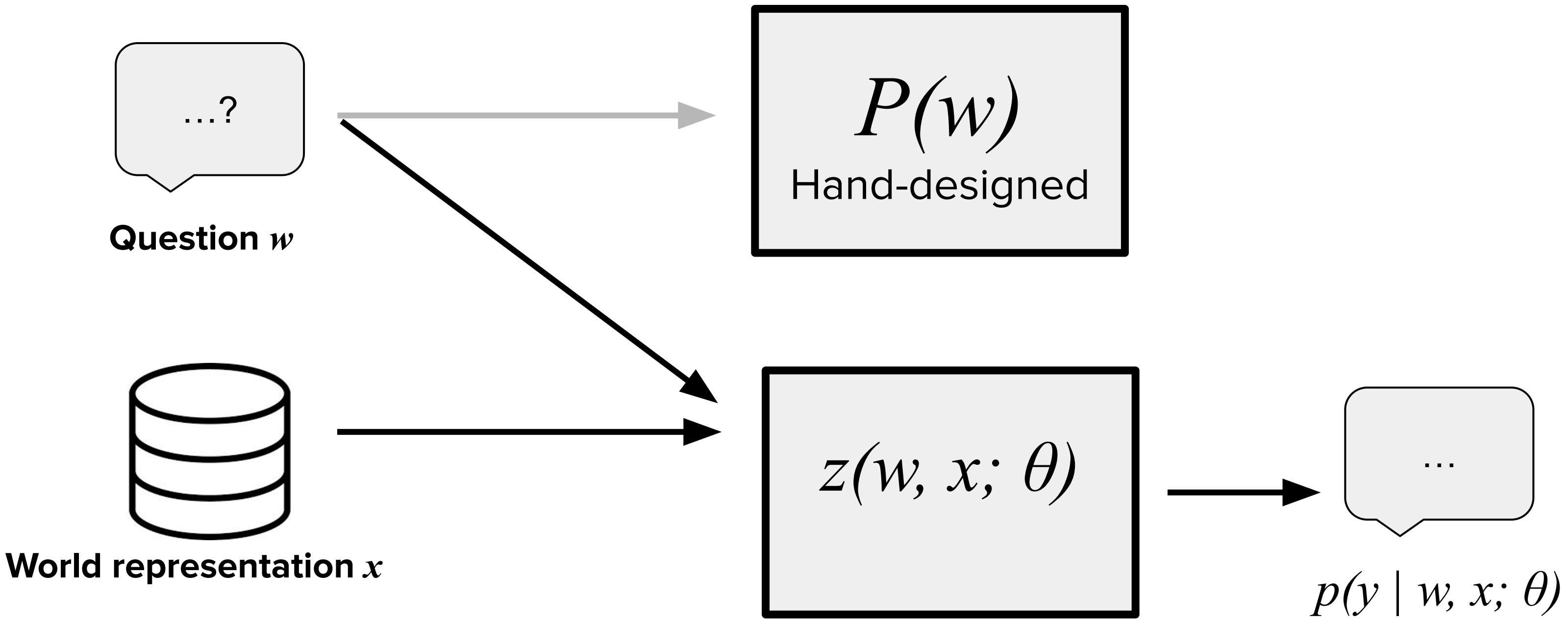
Revised process



Revised process



Revised process



Can we learn P?

Dynamic Neural Module Networks

Comprised of a layout model $p(z \mid x; \theta_l)$ and an execution model $p_z(y \mid w; \theta_e)$

Two major contributions:

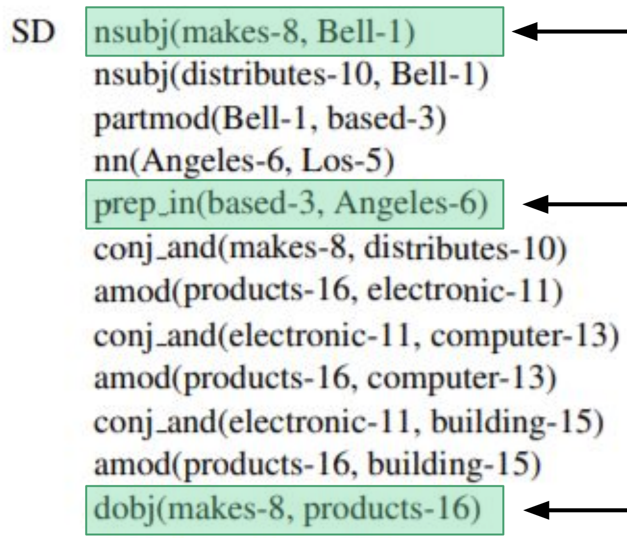
1. Jointly learn network structure predictor with module parameters
2. Extend reasoning to any structured domains

Layout Model Overview

1. Create a dependency parse using the Stanford dependency parser
2. Collect phrases attached to wh-words or copulas
3. Associate these phrases with modules
4. Combine layout fragments using combinatorial modules, then add a top-level labeling module
5. Score layout candidates

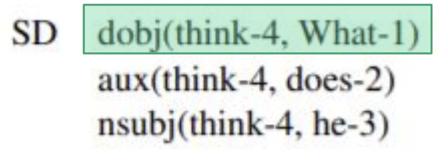
Stanford Dependency (SD) Parse

"Bell, based in Los Angeles, makes and distributes electronic, computer and building products"



```
SD   nsubj(makes-8, Bell-1)                 ←—— Verbs
     nsubj(distributes-10, Bell-1)
     partmod(Bell-1, based-3)
     nn(Angeles-6, Los-5)
     prep_in(based-3, Angeles-6)            ←—— Prepositional
     conj_and(makes-8, distributes-10)           phrases
     amod(products-16, electronic-11)
     conj_and(electronic-11, computer-13)
     amod(products-16, computer-13)
     conj_and(electronic-11, building-15)
     amod(products-16, building-15)
     dobj(makes-8, products-16)             ←—— Nouns
```

"What does he think?"



```
SD   dobj(think-4, What-1)                  ←—— Copulas
     aux(think-4, does-2)
     nsubj(think-4, he-3)
```

de Marneffe & Manning 2008

Module Library

Lookup
→ Attention

Produces an attention map focused at the argument

Find
→ Attention

Computes a distribution over indices in the representation

Relate
Attention → Attention

Directs attention from one part to another

And
Attention* → Attention

Returns intersection of attentions

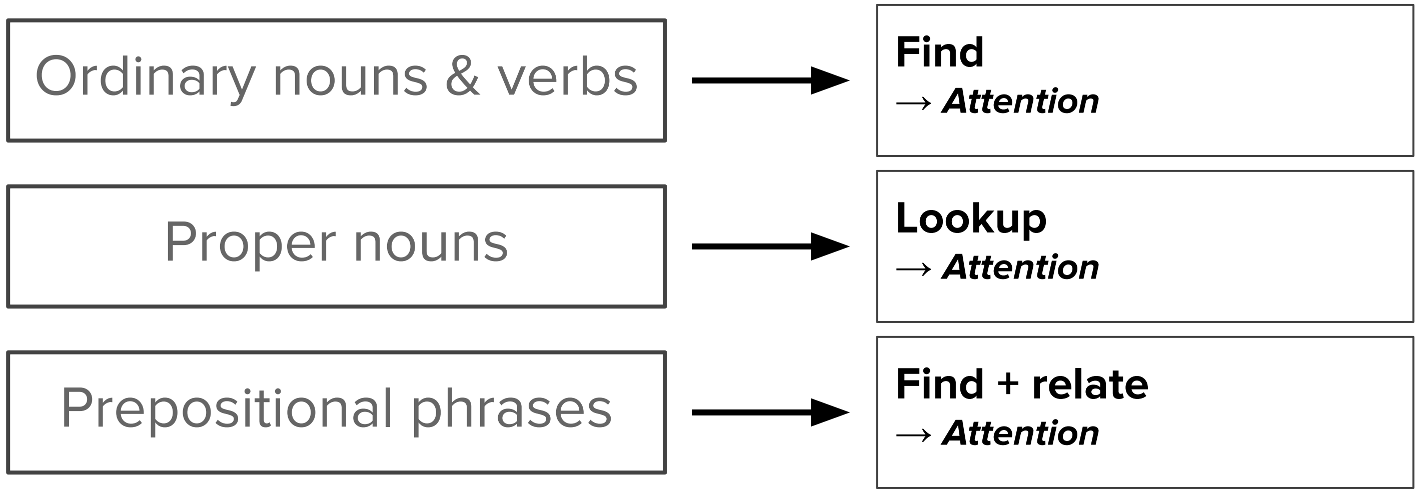
Describe
Attention → Labels

Computes an average of w under attention, then labels it

Exists
Attention → Labels

Inspects attention to produce a label

Module Association



Next, assemble all subsets of these fragments into layout candidates

Scoring Candidates

Obtain scores for each layout, where:

- $h_q(x)$ is a LSTM encoding of the question
- z_n are the layouts, $f(z_n)$ are feature vectors for the layouts

$$s(z_i|x) = a^\top \sigma(Bh_q(x) + Cf(z_i) + d)$$

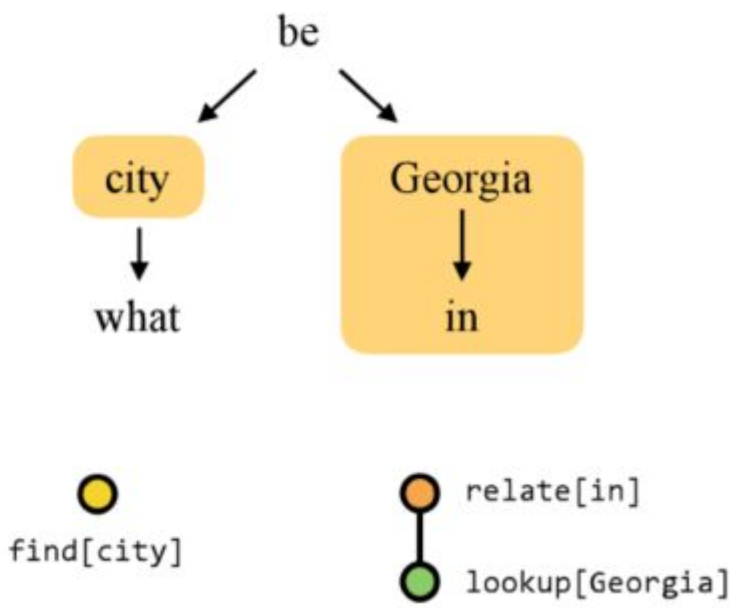
$$p(z_i|x;\theta_\ell) = e^{s(z_i|x)} \Big/ \sum_{j=1}^{n} e^{s(z_j|x)}$$

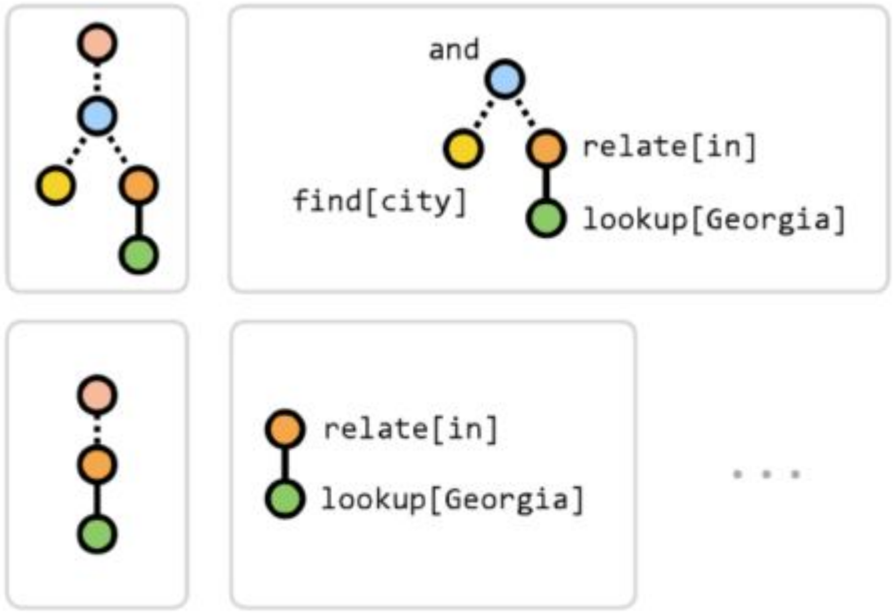
- Lastly, train the layout model with a gradient step:

$$\nabla J(\theta_\ell) = \mathbb{E}[(\nabla \log p(z|x;\theta_\ell)) \cdot \log p(y|z,w;\theta_e)]$$

Example

“What cities are in Georgia?”

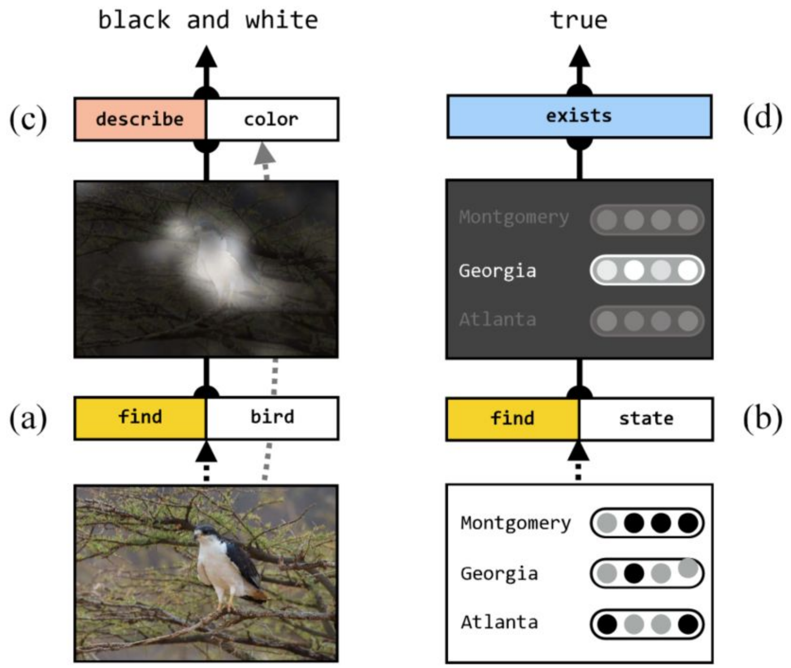




Generating Answers

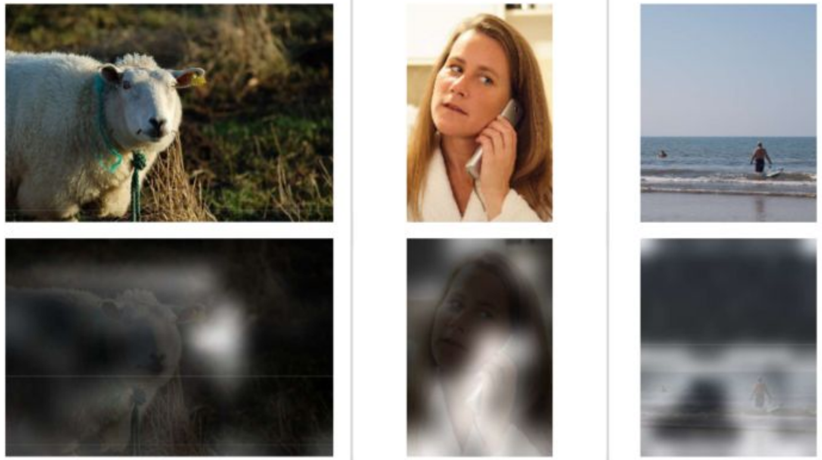
Given a choice of z from the layout model:

- Apply the world w to the network z to get $[z]_w$ (a probability distribution)
- Define $p_z(y \mid w) = ([z]_w)_y$



What color is this bird?   Are there any states?

Evaluation



What is in the sheep's ear?

```
(describe[what]
    (and find[sheep]
        find[ear]))
```

tag

What color is she wearing?

```
(describe[color]
    find[wear])
```

white

What is the man dragging?

```
(describe[what]
    find[man])
```

boat (board)

Image-grounded questions

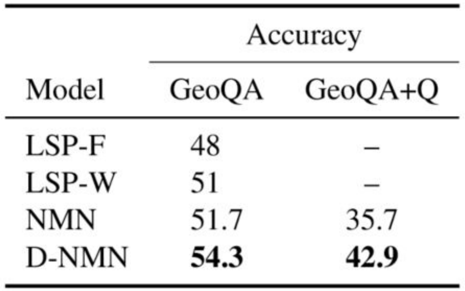
- D-NMN achieved state-of-the-art performance on VQA 2015 challenge
  - Outperforms NMN with hand-designed layout model

	test-dev				test-std
	Yes/No	Number	Other	All	All
Zhou (2015)	76.6	35.0	42.6	55.7	55.9
Noh (2015)	80.7	37.2	41.7	57.2	57.4
Yang (2015)	79.3	36.6	46.1	58.7	58.9
NMN	81.2	38.0	44.0	58.6	58.7
D-NMN	81.1	38.6	45.5	59.4	**59.4**

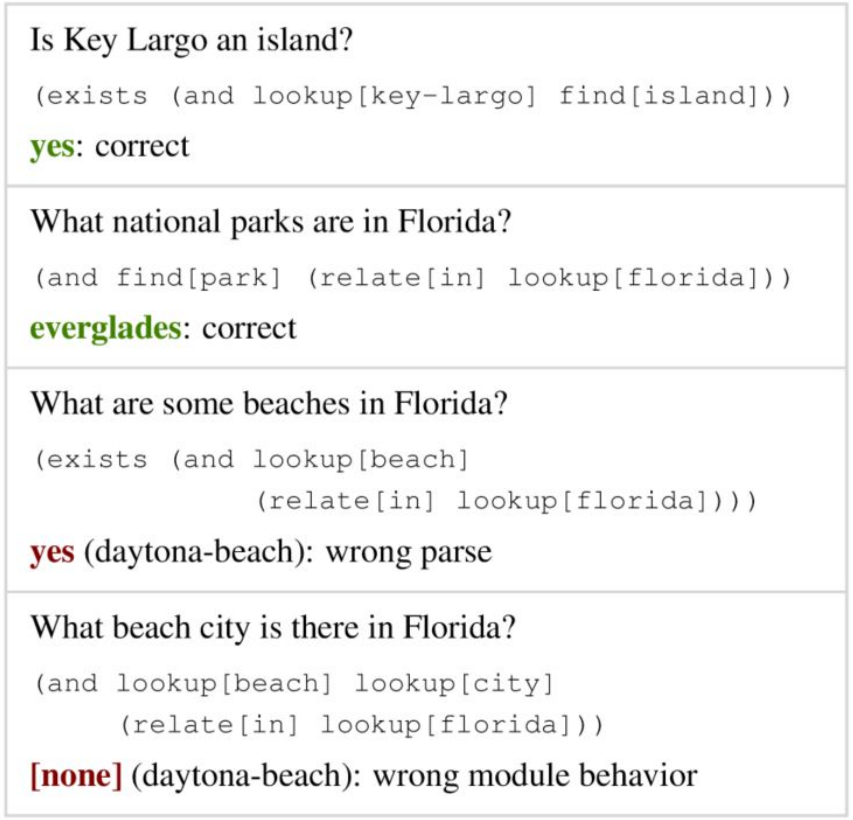
Evaluation

Text-grounded questions

- Introduce GeoQA+Q, which provides more detailed answers to GeoQA questions
- D-NMN performs better than baselines and has a 7% accuracy increase over NMN



	Accuracy	
Model	GeoQA	GeoQA+Q
LSP-F	48	–
LSP-W	51	–
NMN	51.7	35.7
D-NMN	**54.3**	**42.9**



Is Key Largo an island?

```
(exists (and lookup[key-largo] find[island]))
```

yes: correct

What national parks are in Florida?

```
(and find[park] (relate[in] lookup[florida]))
```

everglades: correct

What are some beaches in Florida?

```
(exists (and lookup[beach]
                (relate[in] lookup[florida])))
```

yes (daytona-beach): wrong parse

What beach city is there in Florida?

```
(and lookup[beach] lookup[city]
     (relate[in] lookup[florida]))
```

[none] (daytona-beach): wrong module behavior

Takeaways

Advantage of continuous representations

- Neural representations allow cross-modality and can be learned via gradients

Semantic structure prediction

- Use only the parameters required for the problem
  - Save computation on small problems
  - Use larger networks for harder problems

Discussion

- How could this framework be extended to other domains (e.g. speech, game playing)?
- Is it possible to learn a library of modules from scratch?
- What classes of queries can you/can you not represent?

Key Takeaways + Final Questions

- Benefits of continuous representations
- Integrating compositionality and neural models
- Ways to take advantage of hierarchical structures
- How to reduce/change one model (RNNGs) to another (CVGs)?
- Formally, what’s the relationship btwn the different models?