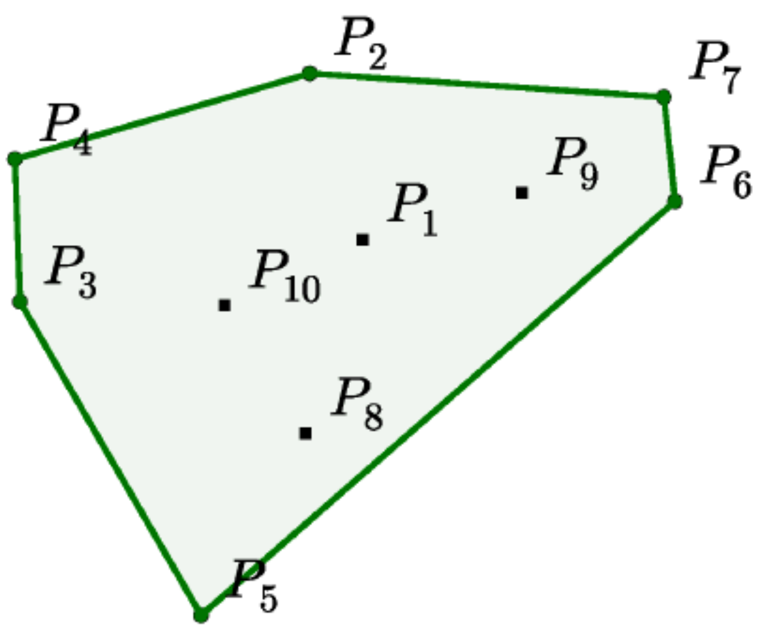
Pointer Networks: Handling variable size output dictionary

Pointer Networks

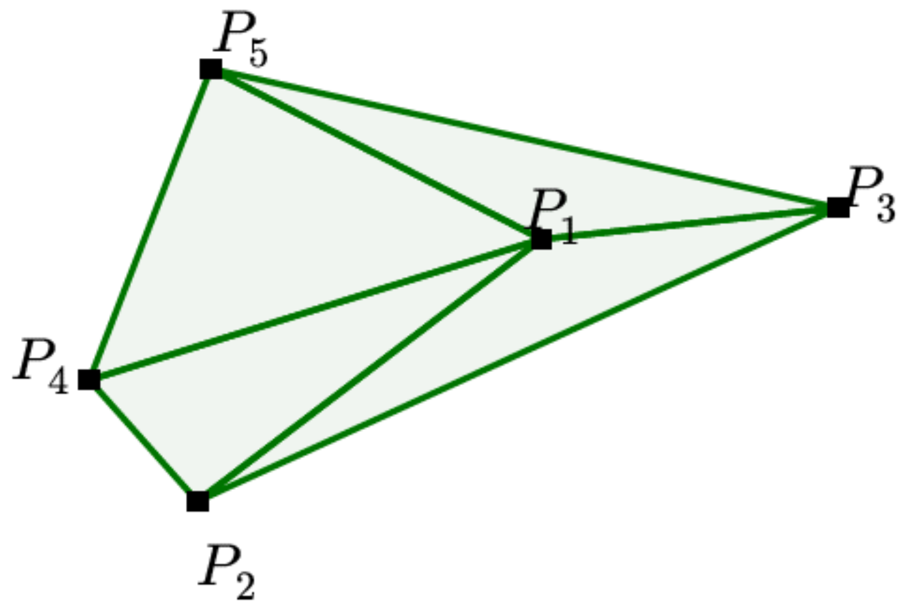
Oriol Vinyals* Google Brain
Meire Fortunato* Department of Mathematics, UC Berkeley
Navdeep Jaitly Google Brain

- Outputs are discrete and correspond to positions in the input. Thus, the output "dictionary" varies per example.
- Q: Can we think of cases where we need such dynamic size dictionary?

Pointer Networks: Handling Variable Size Output Dictionary

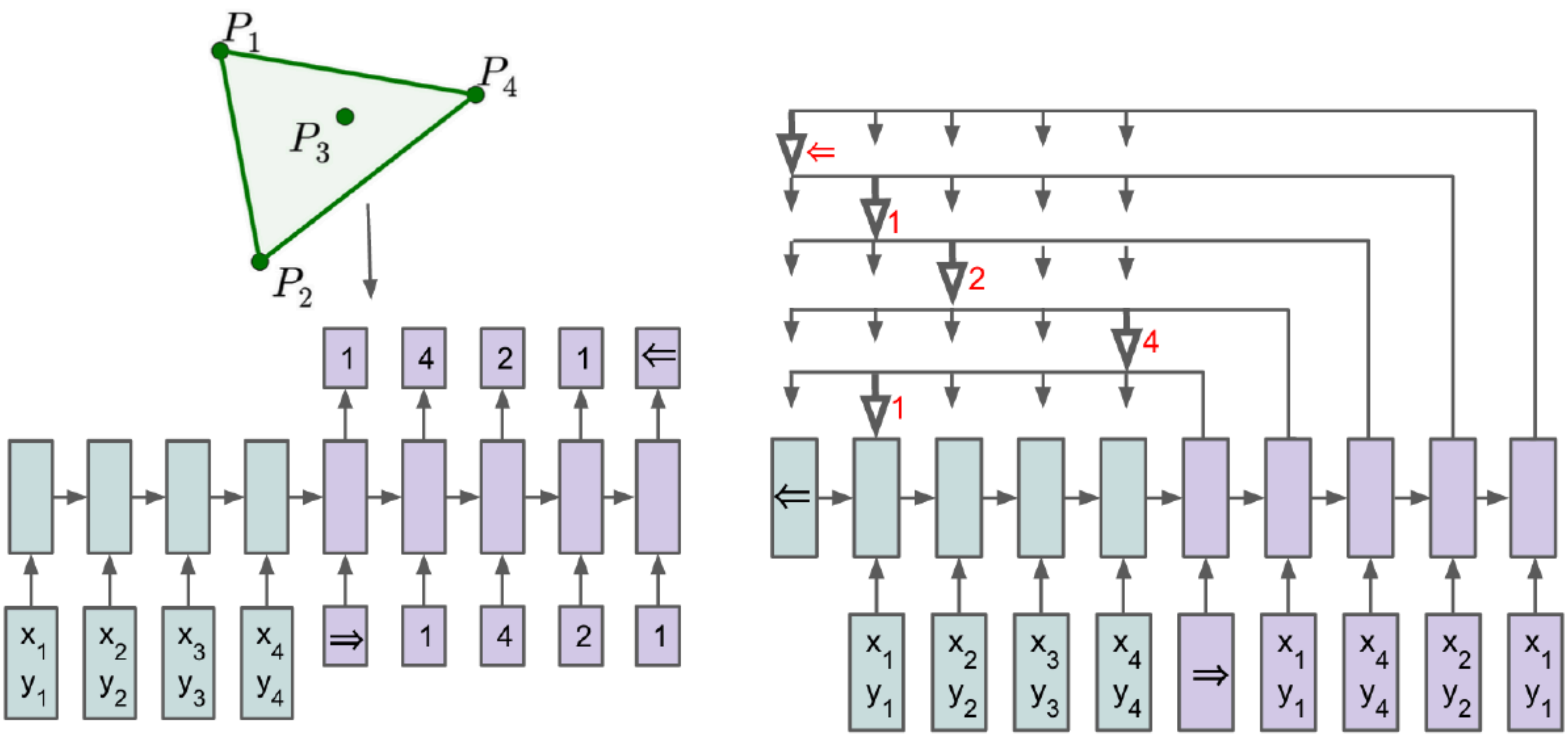


(a) Input $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.



(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, (1, 2, 4), (1, 4, 5), (1, 3, 5), (1, 2, 3), \Leftarrow\}$ representing its Delaunay Triangulation.

Pointer Networks: Handling Variable Size Output Dictionary



(a) Sequence-to-Sequence

(b) Ptr-Net

Pointer Networks: Handling Variable Size Output Dictionary

- Fixed-Size Dictionary

$$
\begin{aligned}
u_j^i &= v^T \tanh(W_1 e_j + W_2 d_i) \quad && j \in (1, \ldots, n) \\
a_j^i &= \mathrm{softmax}(u_j^i) && j \in (1, \ldots, n) \\
d'_i &= \sum_{j=1}^{n} a_j^i e_j
\end{aligned}
$$

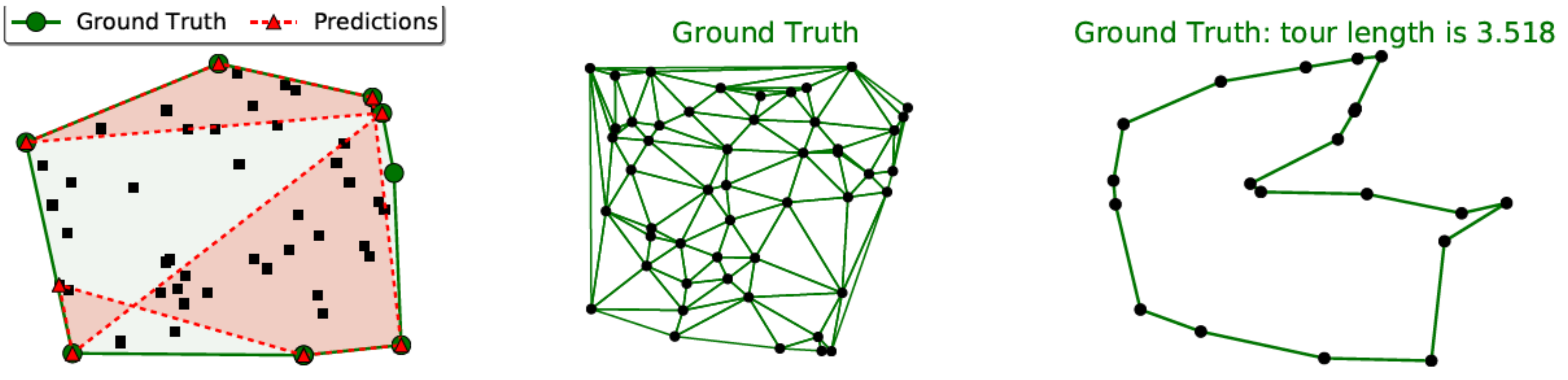
the updated decoder hidden state!, d_i,d'_i are concatenated and feed into a softmax over the fixed size dictionary

- Dynamic Dictionary

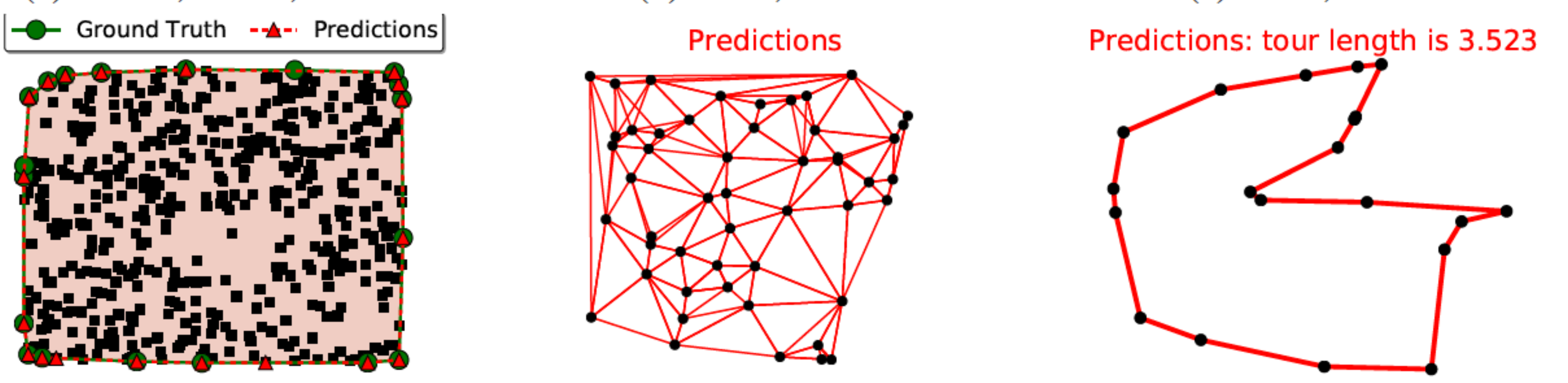
$$
\begin{aligned}
u_j^i &= v^T \tanh(W_1 e_j + W_2 d_i) \quad && j \in (1, \ldots, n) \\
p(C_i | C_1, \ldots, C_{i-1}, \mathcal{P}) &= \mathrm{softmax}(u^i)
\end{aligned}
$$

the decoder hidden state is used to selected the location of the input via interaction with the encoder hidden states e_j

Pointer Networks: Handling Variable Size Output Dictionary



(a) LSTM, m=50, n=50 (b) Truth, n=50 (c) Truth, n=20



(d) Ptr-Net, m=5-50, n=500 (e) Ptr-Net , m=50, n=50 (f) Ptr-Net , m=5-20, n=20

Pointer Networks: Handling Variable Size Output Dictionary

METHOD	TRAINED n	n	ACCURACY	AREA
LSTM [1]	50	50	1.9%	FAIL
+ATTENTION [5]	50	50	38.9%	99.7%
PTR-NET	50	50	72.6%	99.9%
LSTM [1]	5	5	87.7%	99.6%
PTR-NET	5-50	5	92.0%	99.6%
LSTM [1]	10	10	29.9%	FAIL
PTR-NET	5-50	10	87.0%	99.8%
PTR-NET	5-50	50	69.6%	99.9%
PTR-NET	5-50	100	50.3%	99.9%
PTR-NET	5-50	200	22.1%	99.9%
PTR-NET	5-50	500	1.3%	99.2%

Pointer Networks: Handling Variable Size Output Dictionary

Table 2: Tour length of the Ptr-Net and a collection of algorithms on a small scale TSP problem.

n	Optimal	A1	A2	A3	Ptr-Net
5	2.12	2.18	2.12	2.12	2.12
10	2.87	3.07	2.87	2.87	2.88
50 (A1 trained)	N/A	6.46	5.84	5.79	6.42
50 (A3 trained)	N/A	6.46	5.84	5.79	6.09
5 (5-20 trained)	2.12	2.18	2.12	2.12	2.12
10 (5-20 trained)	2.87	3.07	2.87	2.87	2.87
20 (5-20 trained)	3.83	4.24	3.86	3.85	3.88
25 (5-20 trained)	N/A	4.71	4.27	4.24	4.30
30 (5-20 trained)	N/A	5.11	4.63	4.60	4.72
40 (5-20 trained)	N/A	5.82	5.27	5.23	5.91
50 (5-20 trained)	N/A	6.46	5.84	5.79	7.66

Key-variable memory

We use similar indexing mechanism to index location in the key variable memory, during decoding, when we know we need to pick an argument, as opposed to function name. All arguments are stored in such memory.

Carnegie Mellon
School of Computer Science

Language Grounding to Vision and Control

Recursive/tree structured networks

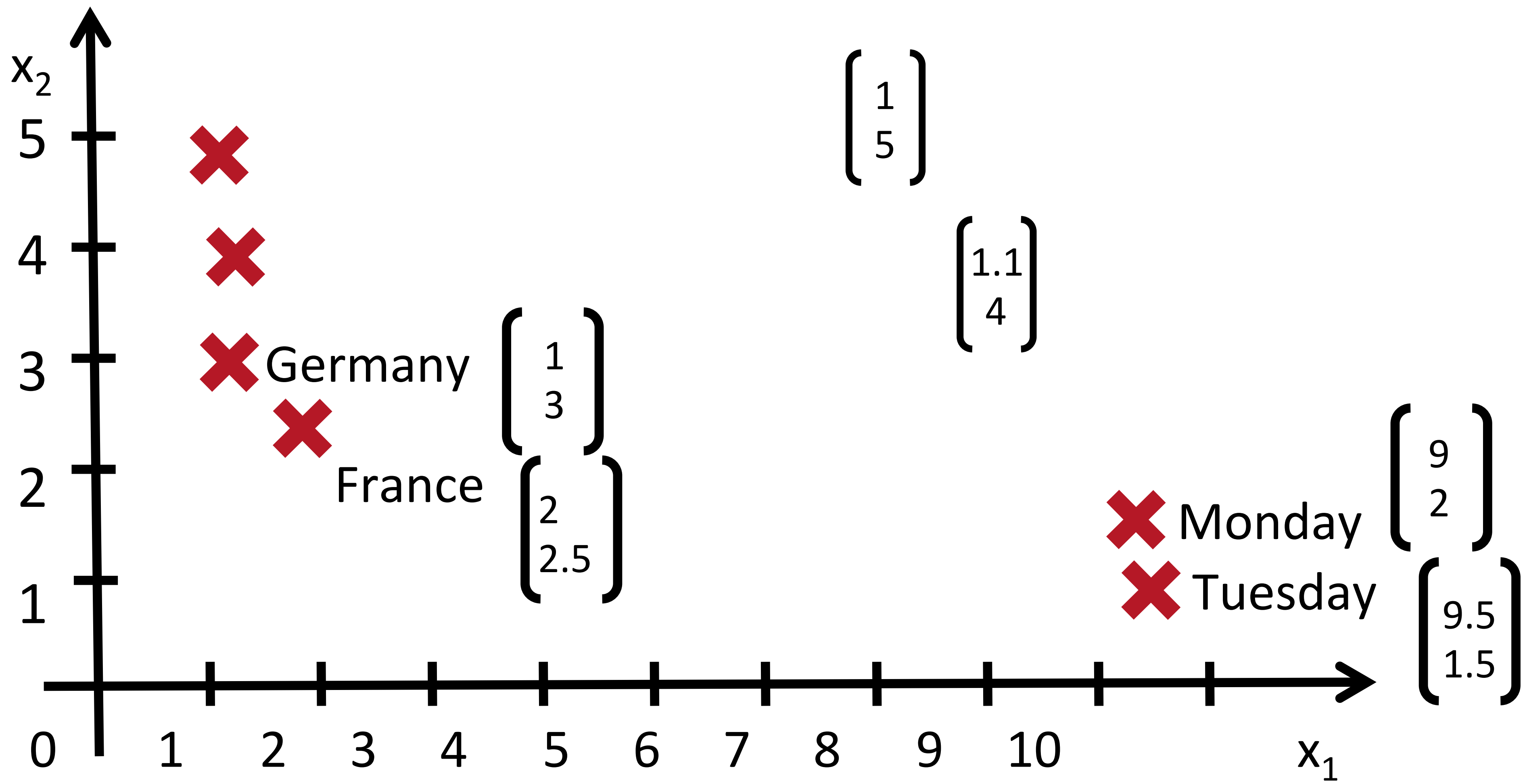
Katerina Fragkiadaki

From Words to Phrases

- We have already discussed word vector representations that "capture the meaning" of word by embedding them into a low-dimensional space where semantic similarity is preserved.

- **But what about longer phrases?** For this lecture, understanding of the meaning of a sentence is representing the phrase as a vector in a structured semantic space, where similar sentences are nearby, and unrelated sentences are far away.

Building on Word Vector Space Models



The country of my birth vs. The place where I was born

How can we represent the meaning of longer phrases? By mapping them into the same vector space as words!

Slide adapted from Manning-Socher

From Words to Phrases

- We have already discussed word vector representations that "capture the meaning" of word by embedding them into a low-dimensional space where semantic similarity is preserved.

- **But what about longer phrases?** For this lecture, understanding of the meaning of a sentence is representing the phrase as a vector in a structured semantic space, where similar sentences are nearby, and unrelated sentences are far away.

- Sentence modeling is at the core of many language comprehension tasks sentiment analysis, paraphrase detection, entailment recognition, summarization, discourse analysis, machine translation, grounded language learning and image retrieval

From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

”A small crowd quietly enters the historical church”.

Slide adapted from Manning-Socher

From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

”A small crowd quietly enters the historical church”.

From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

"A small crowd quietly enters the historical church".

From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

”A small crowd quietly enters the historical church”.

From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

”A small crowd quietly enters the historical church”.

From Words to Phrases

- How can we know when larger units of a sentence are similar in meaning?
  - The snowboarders is leaping over a mogul.
  - A person on a snowboard jumps into the air.

- People interpret the meaning of larger text units - entities, descriptive terms, facts, arguments, stories - by **semantic composition** of smaller elements.

"A small crowd quietly enters the historical church".

From Words to Phrases: 4 models

- Bag of words: Ignores word order, simple averaging of word vectors in a sub-phrase. Can't capture differences in meaning as a result of differences in word order, e.g., "*cats climb trees*" and "*trees climb cats*" will have the same representation.

- Sequence (recurrent) models, e.g., LSTMs: The hidden vector of the last word is the representation of the phrase.

- Tree-structured (recursive) models: compose each phrase from its constituent sub-phrases, according to a given syntactic structure over the sentence

- Convolutional neural networks

Q: Does semantic understanding improve with grammatical understanding so that recursive models are justified?

From Words to Phrases: 4 models

- Bag of words: Ignores word order, simple averaging of word vectors in a sub-phrase. Can't capture differences in meaning as a result of differences in word order, e.g., "*cats climb trees"* and "*trees climb cats"* will have the same representation.

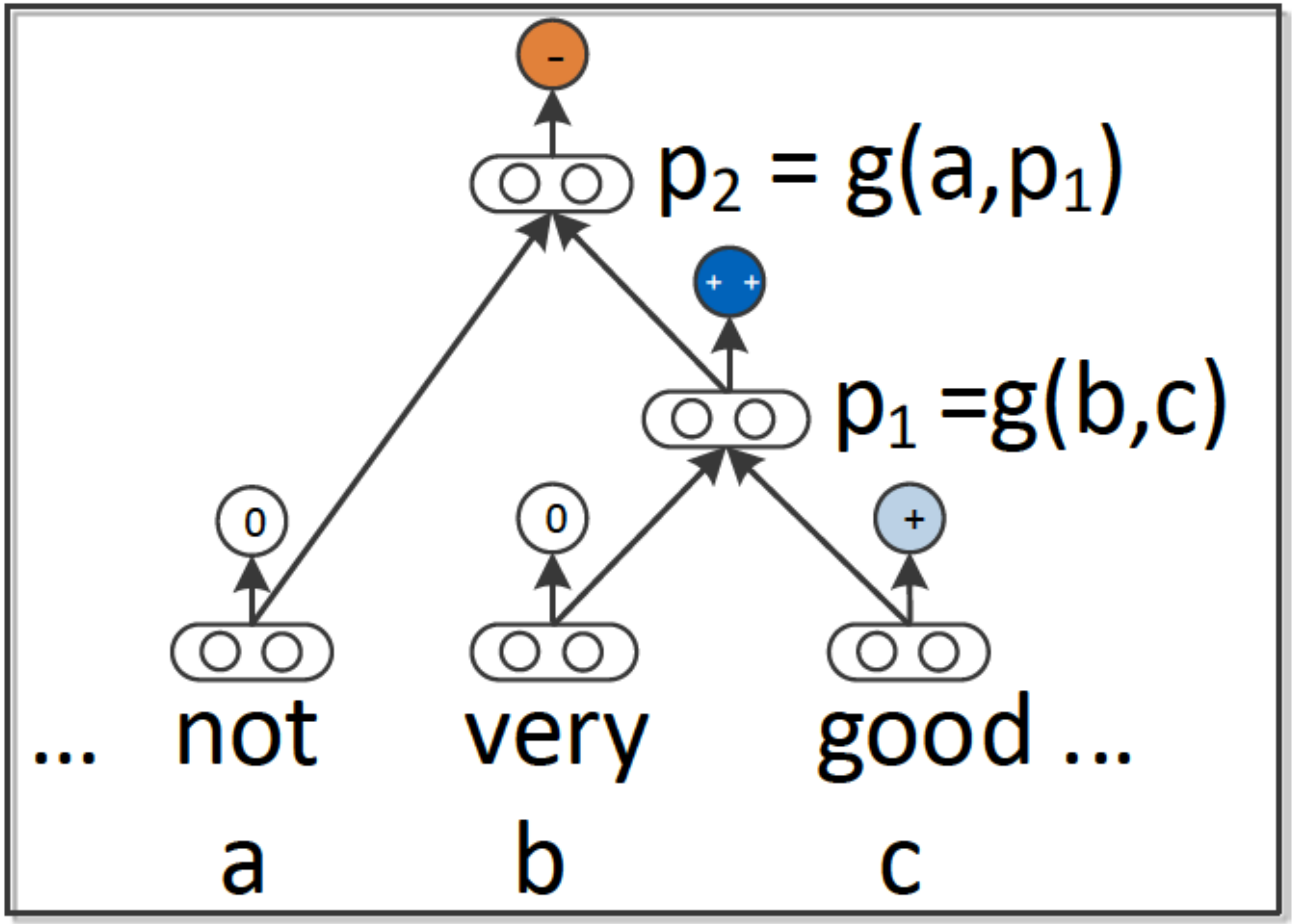
- Sequence models, e.g., LSTMs: The hidden vector of the last word is the representation of the phrase.

- Tree-structured (recursive) models: compose each phrase from its constituent sub-phrases, according to a given syntactic structure over the sentence

- Convolutional neural networks

Q: Does semantic understanding improve with grammatical understanding so that recursive models are justified?

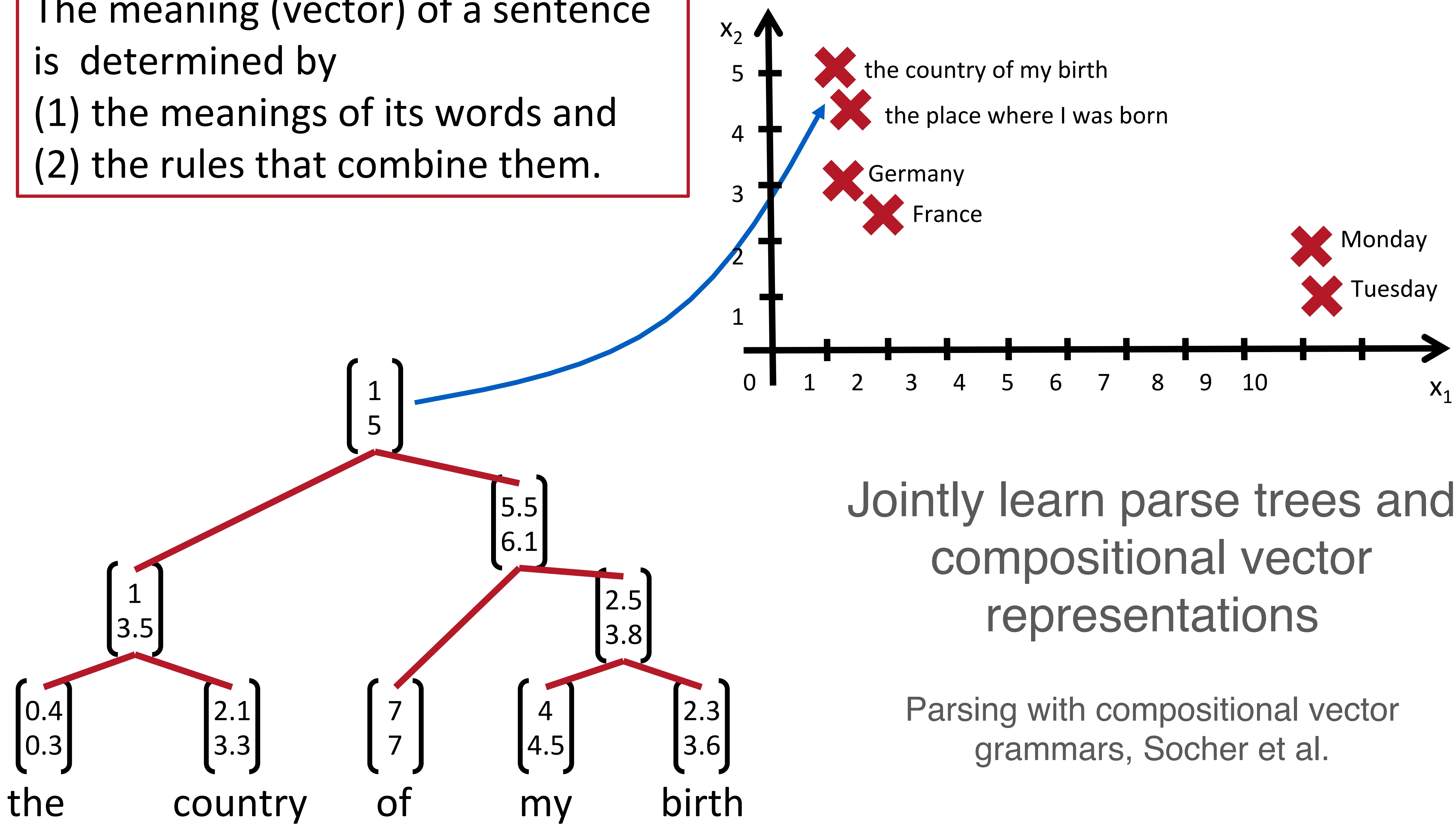
Recursive Neural Networks



Given a tree and vectors for the leaves, compute bottom-up vectors for the intermediate nodes, all the way to the root, via compositional function g.

How should we map phrases into a vector space?

Use principle of compositionality
The meaning (vector) of a sentence is determined by (1) the meanings of its words and (2) the rules that combine them.

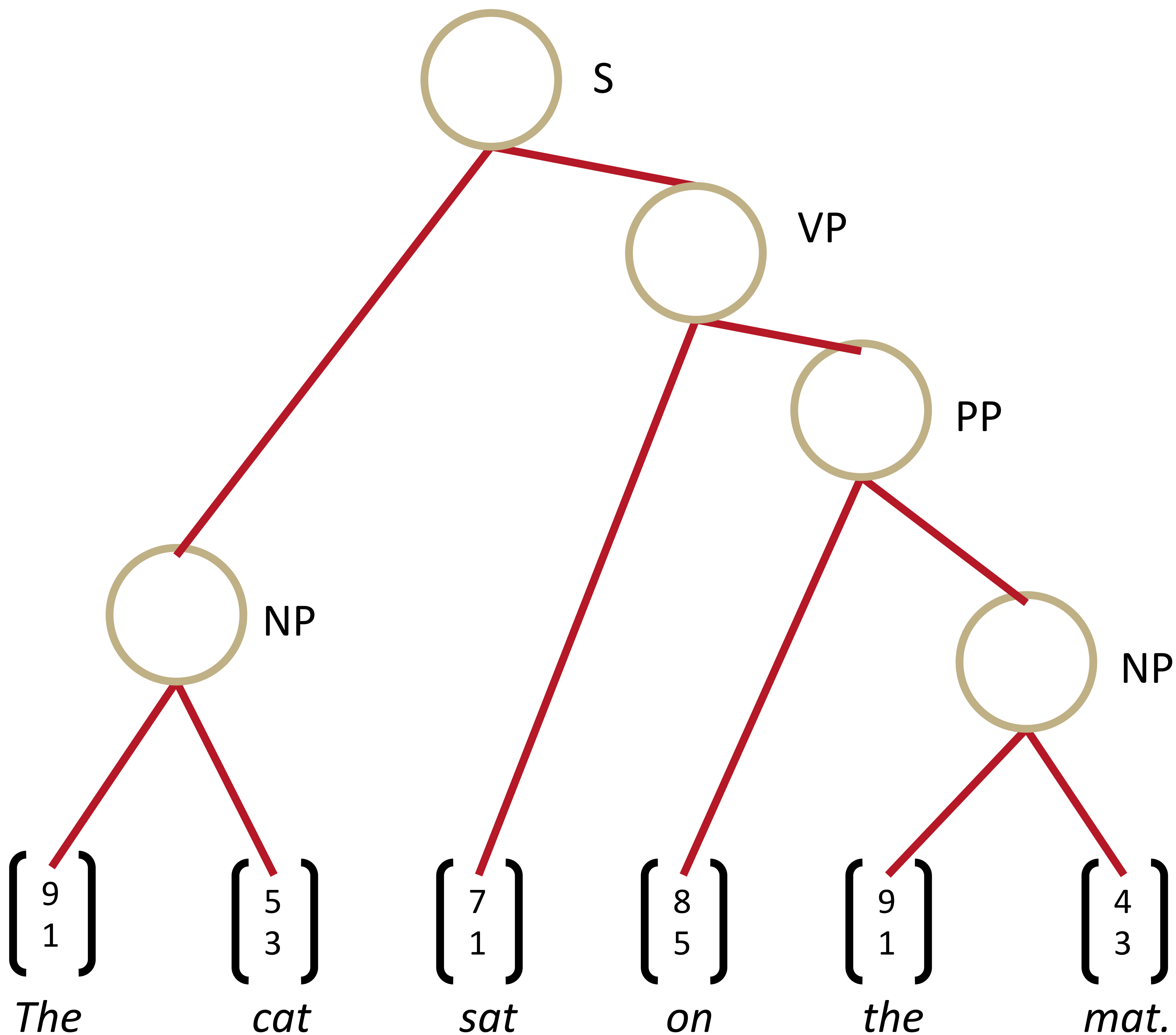


Jointly learn parse trees and compositional vector representations

Parsing with compositional vector grammars, Socher et al.

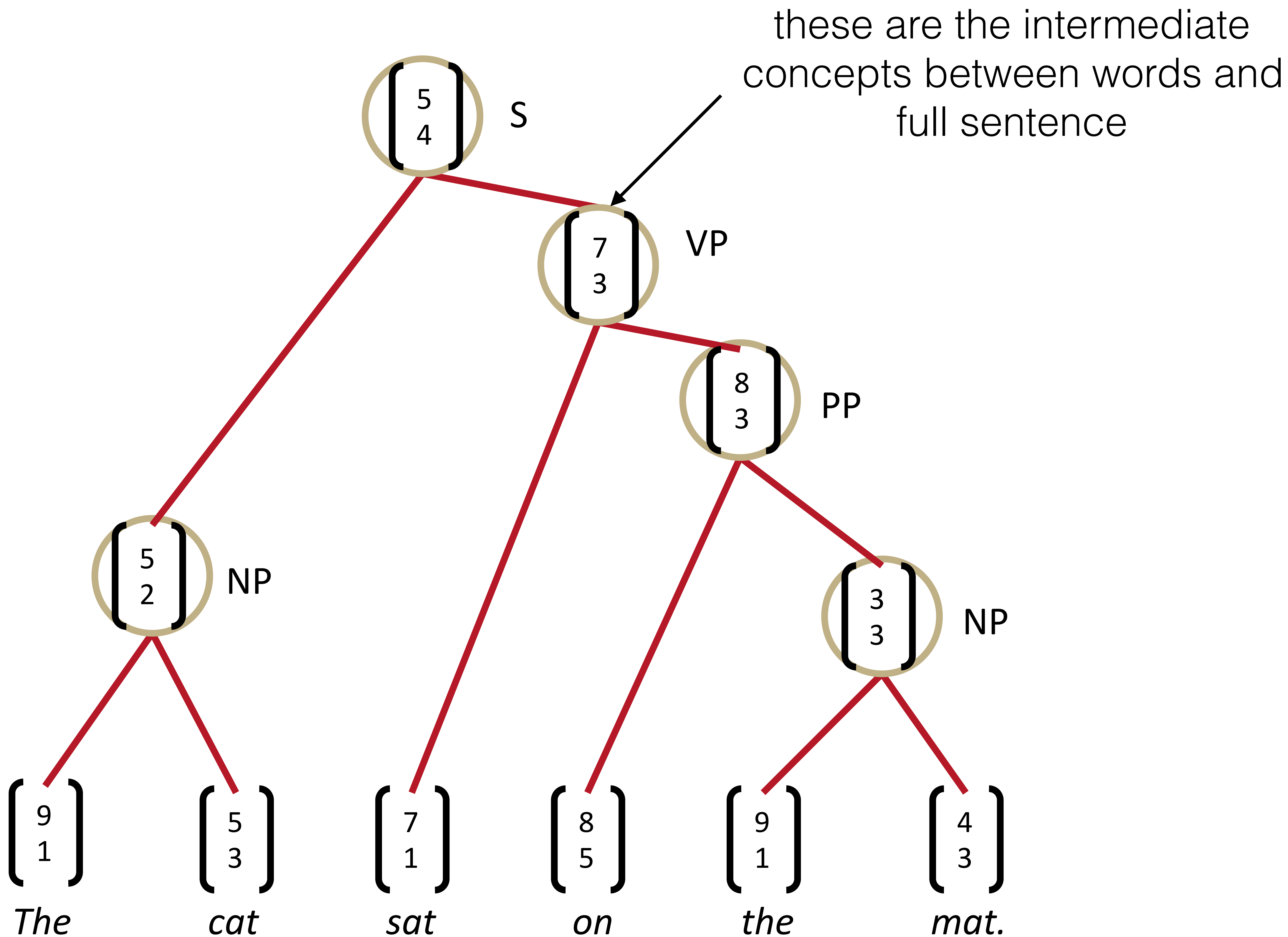
Slide adapted from Manning-Socher

Constituency Sentence Parsing



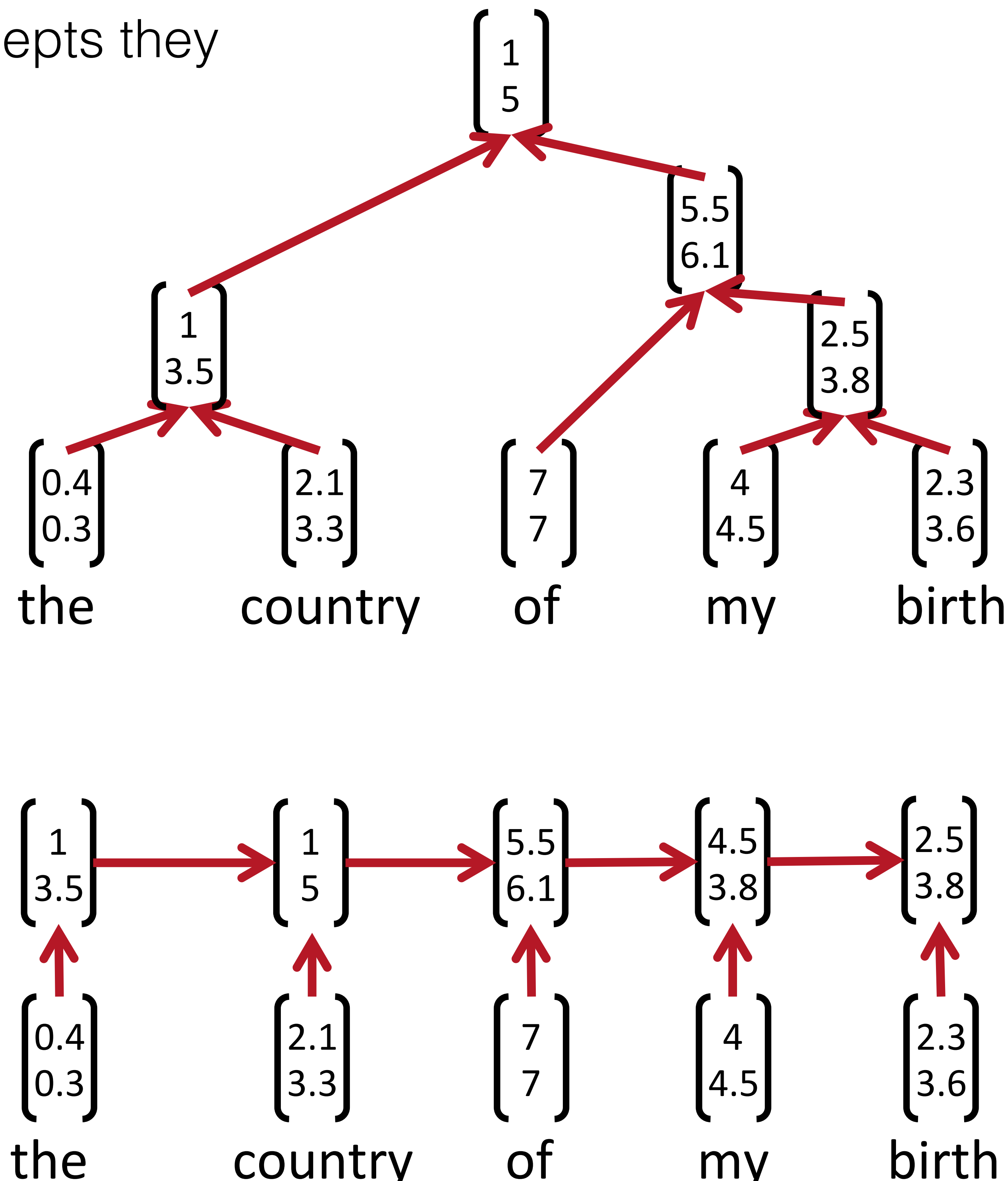
Slide adapted from Manning-Socher

Learn Structure and Representation



Recursive vs. Recurrent Neural Networks

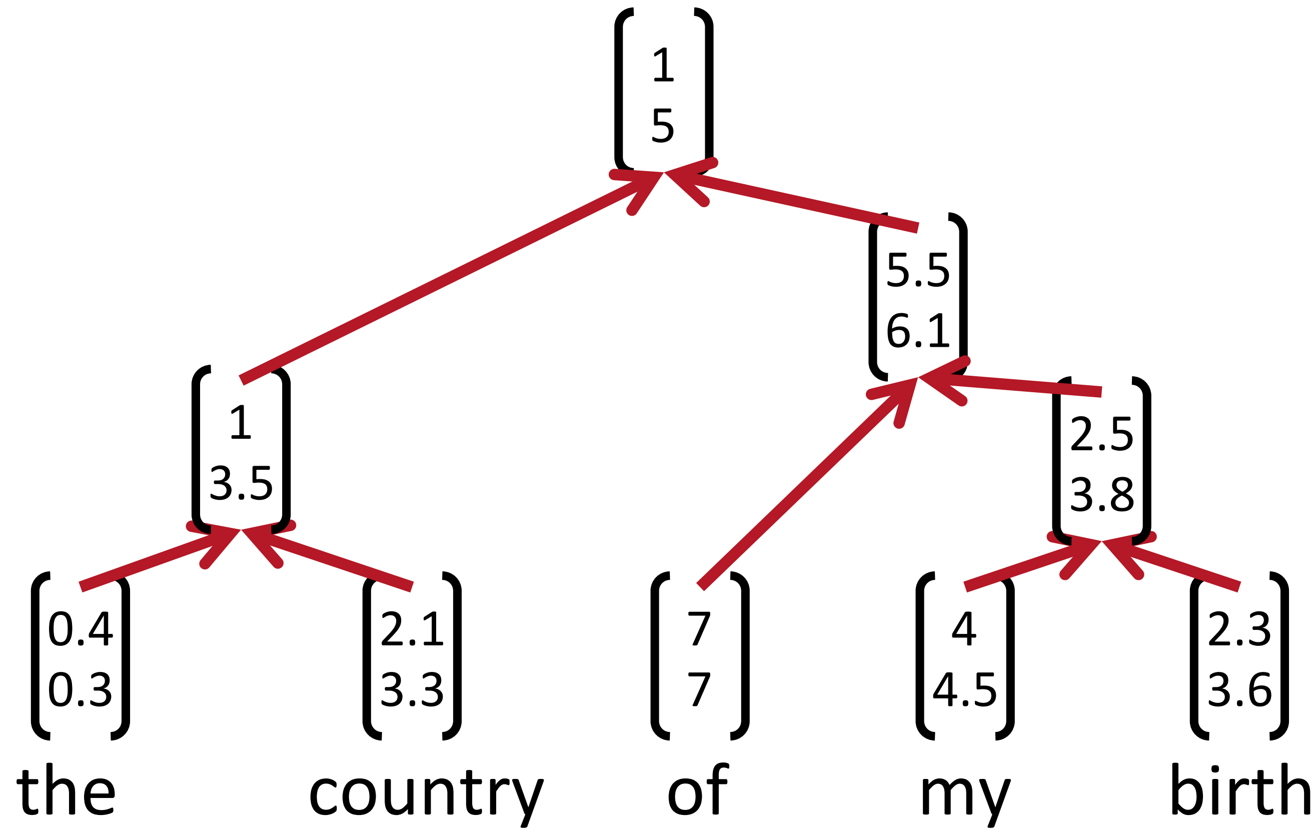
Q: what is the difference in the intermediate concepts they build?



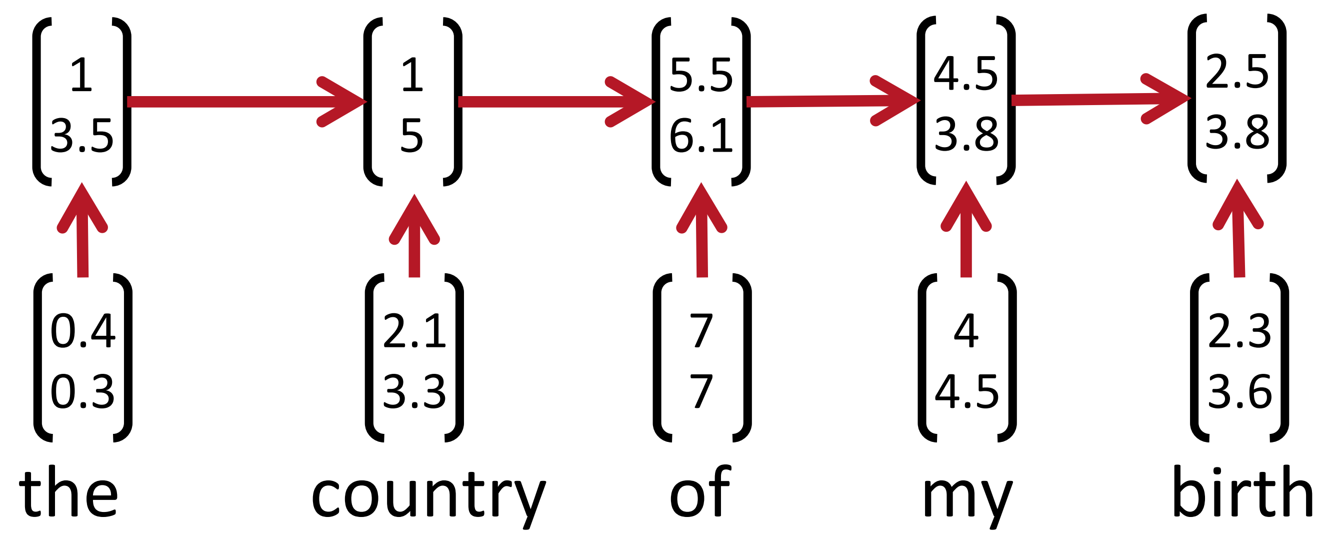
Slide adapted from Manning-Socher

Recursive vs. Recurrent Neural Networks

Recursive neural nets require a parser to get tree structure.

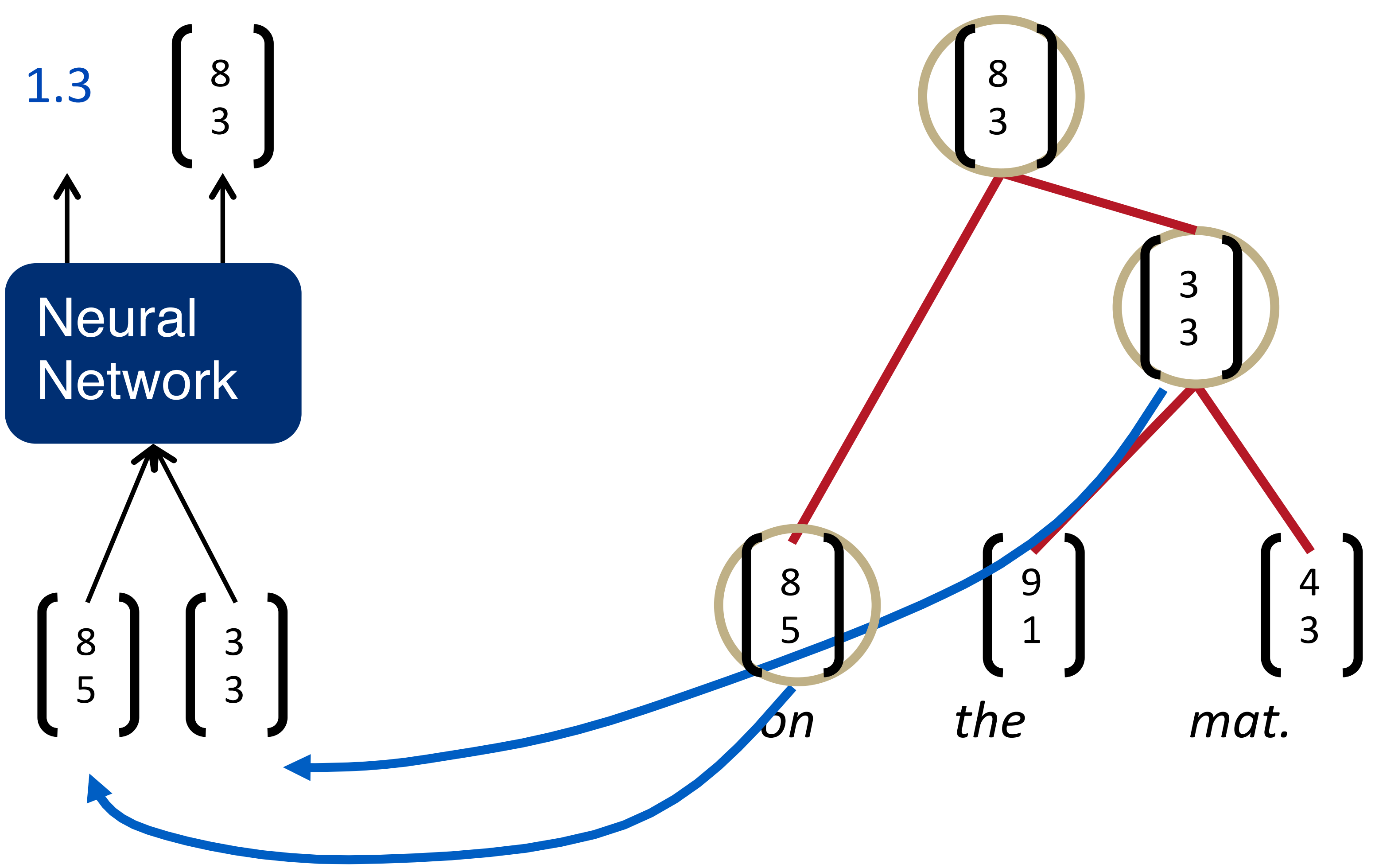


Recurrent neural nets cannot capture phrases without prefix context and often capture too much of last words in final vector. However, they do not need a parser, and they are much preferred in current literature at least.



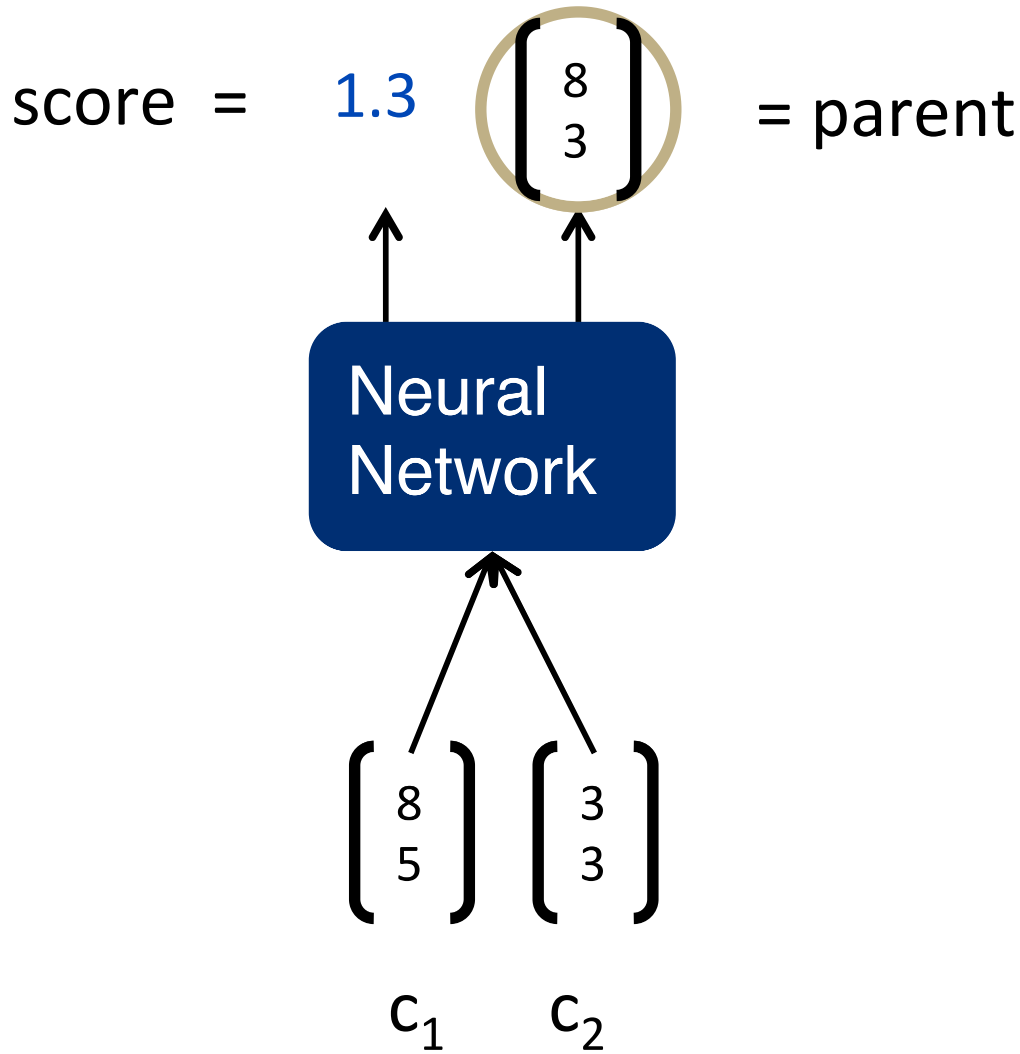
Recursive Neural Networks for Structure Prediction

- Inputs: Two candidate children's representations
- Outputs:
  1. The semantic representation if the two nodes are merged.
  2. Score of how plausible the new node would be.



Slide adapted from Manning-Socher

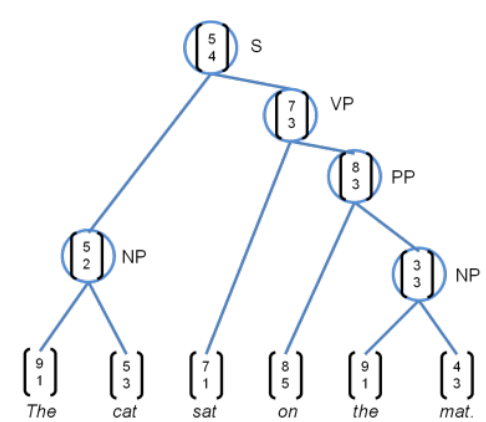
Recursive Neural Network (Version 1)



score $= U^{\mathsf{T}} p$

$$\boxed{\text{parent p} \quad p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right),}$$

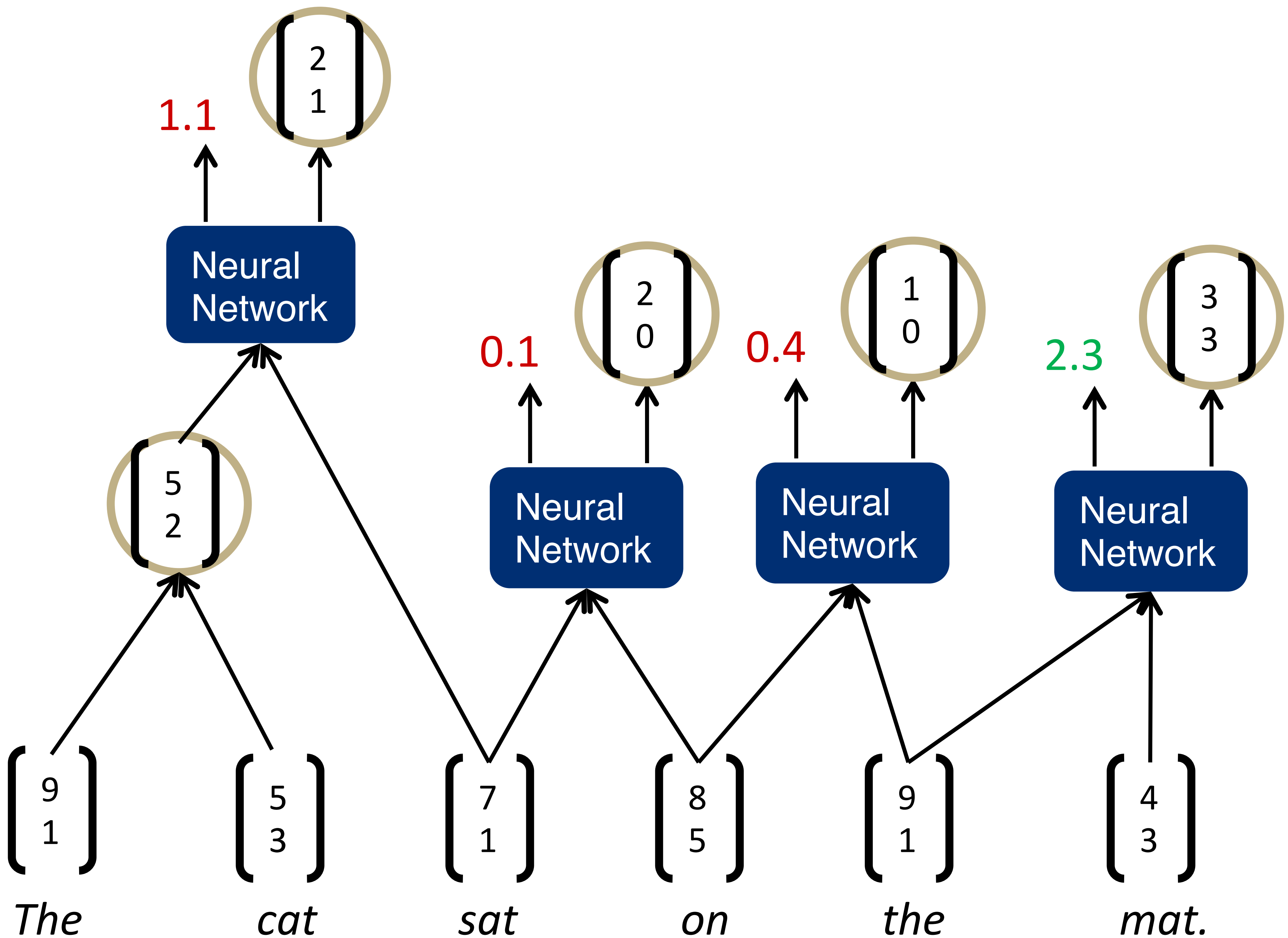
Same *W* parameters at all nodes of the tree



Slide adapted from Manning-Socher

Parsing a Sentence

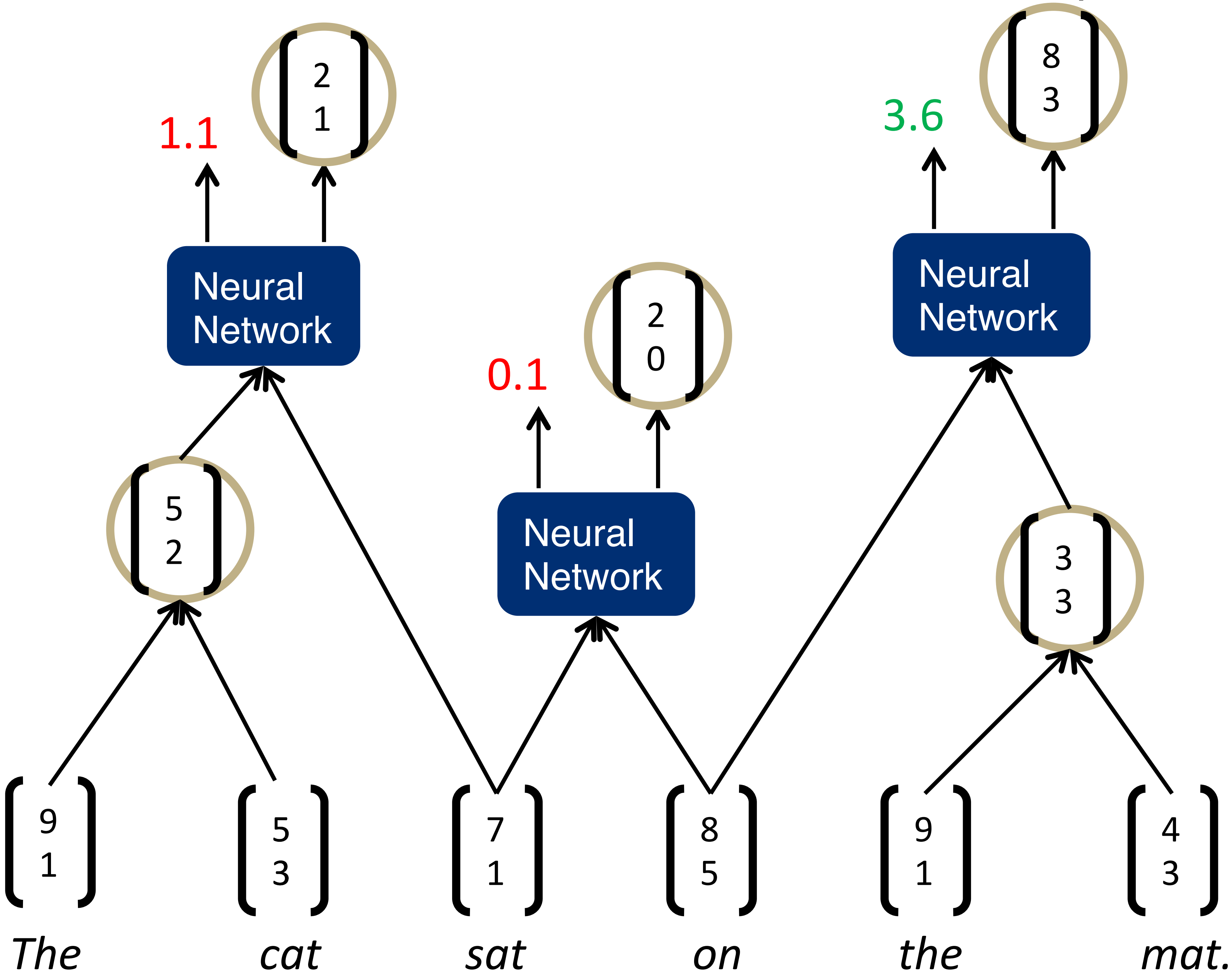
Bottom-up beam search



Slide adapted from Manning-Socher

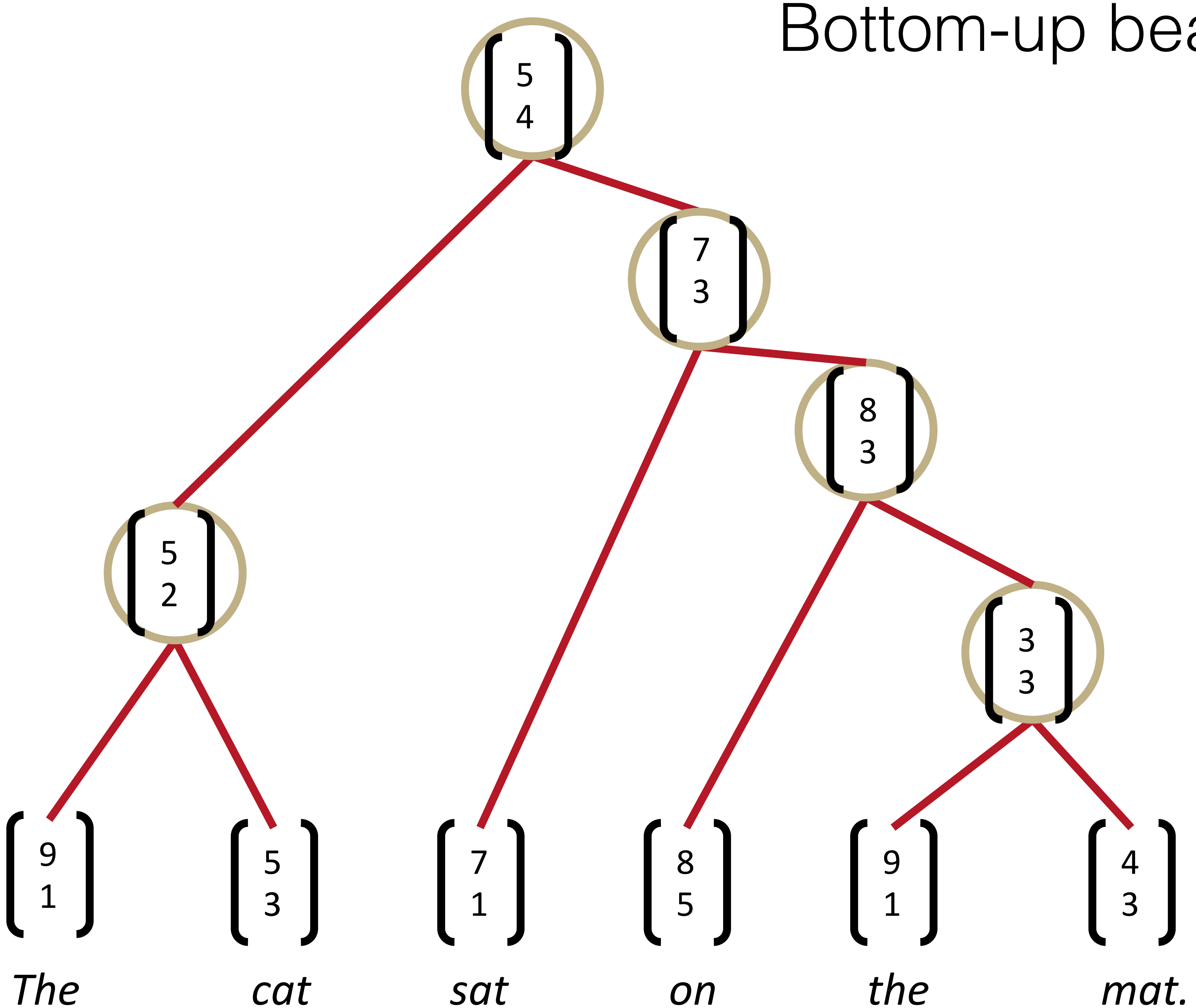
Parsing a Sentence

Bottom-up beam search



Parsing a Sentence

Bottom-up beam search



Slide adapted from Manning-Socher

Cost function

- The score of a tree is computed by the sum of the parsing decision scores at each node:

$$s(x, y) = \sum_{n \in nodes(y)} s_n$$

- *x* is sentence; *y* is parse tree

Cost function

- Max-margin objective:

$$J = \sum_i s(x_i, y_i) - \max_{y \in A(x_i)} \left(s(x_i, y) + \Delta(y, y_i) \right)$$

parse trees resulting from beam search

- The loss *Δ(y, yi)* penalized all incorrect decisions

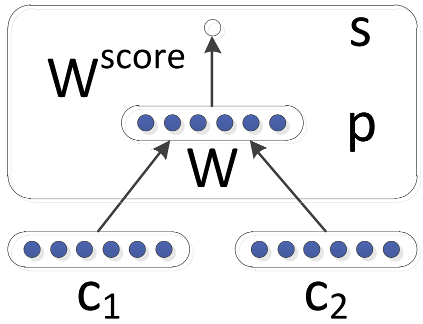
Backpropagation Through Structure

- We update parameters, and sample new trees for every example periodically.0
- In practice, first we compute the top best trees from a PCFG (probabilistic context free grammar), and then we use those trees to learn the parameters of the recursive net, using backdrop through structure (similar to backdrop through time).
- This means the trees for each example are not updated during parameter learning
- It is like a cascade

RecursiveNN Version 1: Discussion

Single weight matrix RecursiveNN could capture some phenomena, but not adequate for more complex, higher order composition and parsing long sentences.

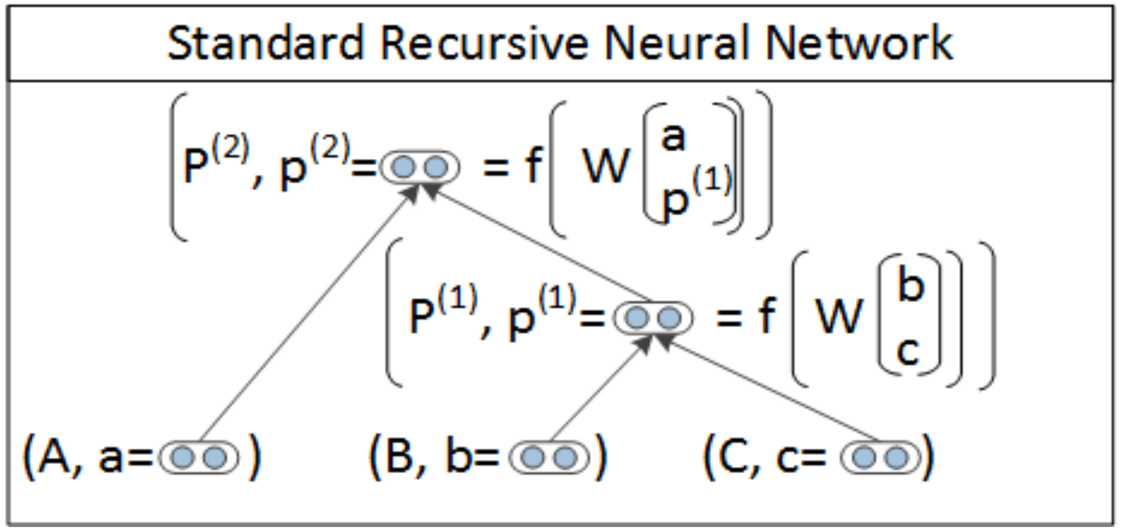
- There is no real interaction between the input words.
- The composition function is the same for all syntactic categories, punctuation, etc.

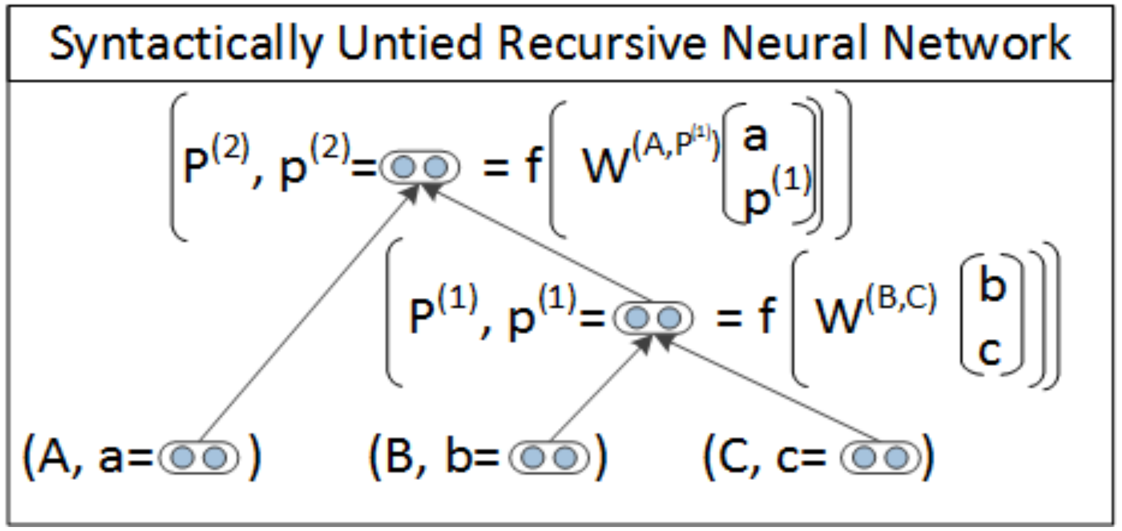


Version 2: Syntactically-Untied RNN

- We use the discrete syntactic categories of the children to choose the composition matrix.
- A TreeRNN can do better with different composition matrix for different syntactic environments.
- This gives better results

A,B,C are part of speech tags





Slide adapted from Manning-Socher

Version 2: Syntactically-Untied RNN

- Problem: Speed. Every candidate score in beam search needs a matrix-vector product.
- Solution: Compute score only for a subset of trees coming from a simpler, faster model (PCFG)
  - Prunes very unlikely candidates for speed
  - Provides coarse syntactic categories of the children for each beam candidate.
- Compositional Vector Grammar = PCFG + TreeRNN

Version 2: Syntactically-Untied RNN

- Scores at each note computed by combination of PCFG and SU-RNN:

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

- Interpretation: Factoring discrete and continuous parsing in one model:

$$\begin{aligned} &P((P_1, p_1) \rightarrow (B, b)(C, c)) \\ &= P(p_1 \rightarrow b \;\; c | P_1 \rightarrow B \;\; C) P(P_1 \rightarrow B \;\; C) \end{aligned}$$

Slide adapted from Manning-Socher

Experiments

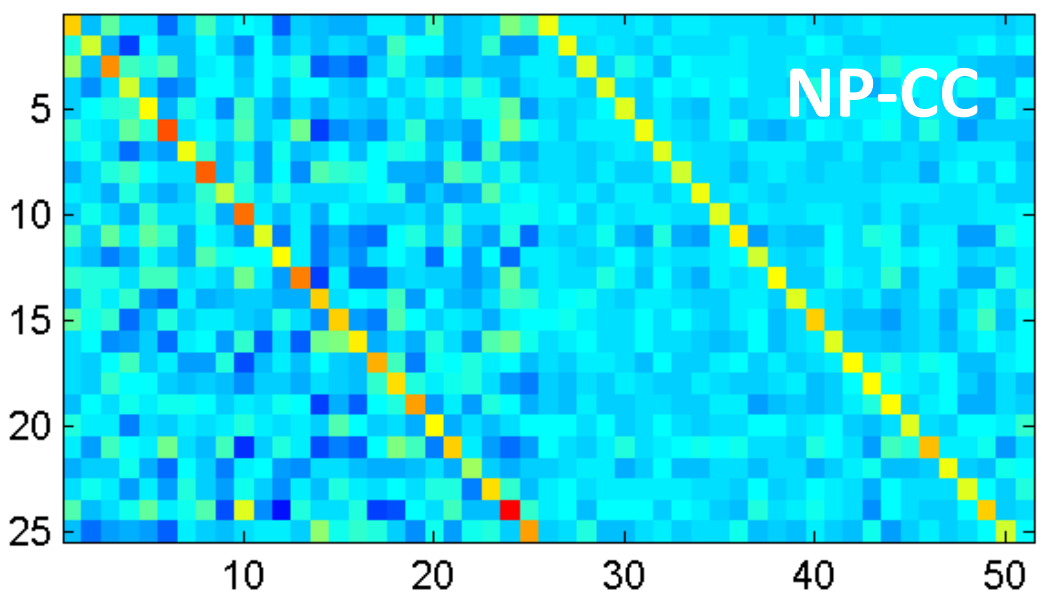
- Standard WSJ split, labeled F1
- Based on simple PCFG with fewer states
- Fast pruning of search space, few matrix-vector products
- 3.8% higher F1, 20% faster than Stanford factored parser

Parser	Test, All Sentences
Stanford PCFG, (Klein and Manning, 2003a)	85.5
Stanford Factored (Klein and Manning, 2003b)	86.6
Factored PCFGs (Hall and Klein, 2012)	89.4
Collins (Collins, 1997)	87.7
SSN (Henderson, 2004)	89.4
Berkeley Parser (Petrov and Klein, 2007)	90.1
CVG (RNN) (Socher et al., ACL 2013)	85.0
CVG (SU-RNN) (Socher et al., ACL 2013)	90.4
Charniak - Self Trained (McClosky et al. 2006)	91.0
Charniak - Self Trained-ReRanked (McClosky et al. 2006)	92.1

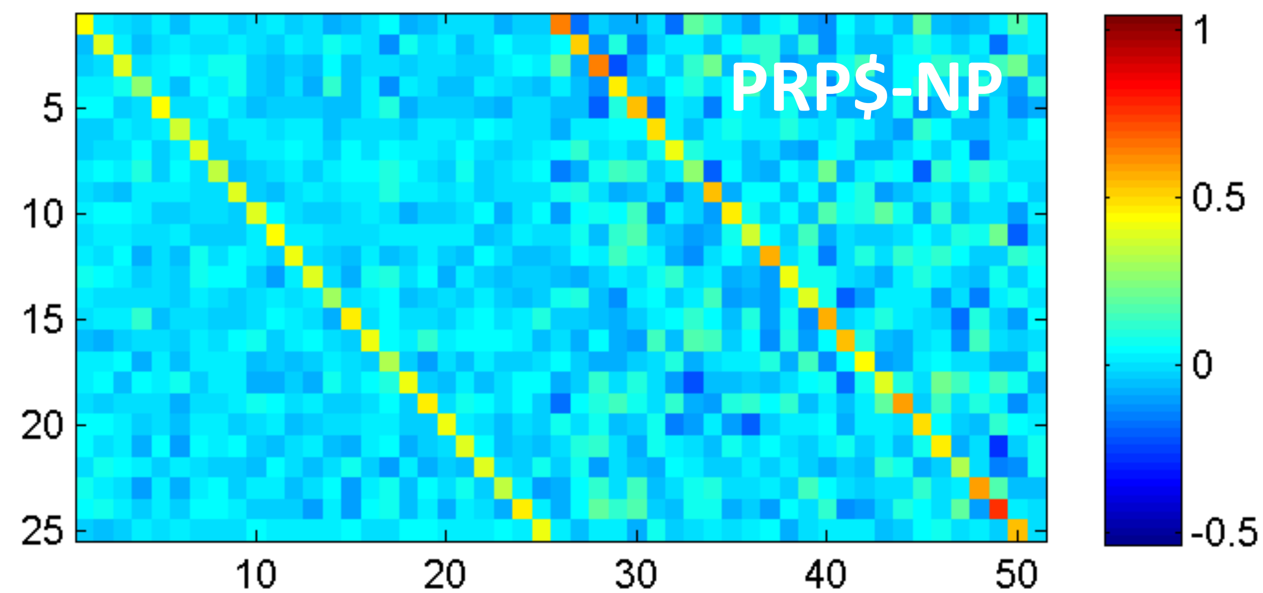
SU-RNN/CVG

- Learns soft notion of head words
- Initialization: $W^{(\cdot\cdot)} = 0.5[I_{n\times n} I_{n\times n} 0_{n\times 1}] + \epsilon$

CC: coordinating conjunction, e.g., ``and”



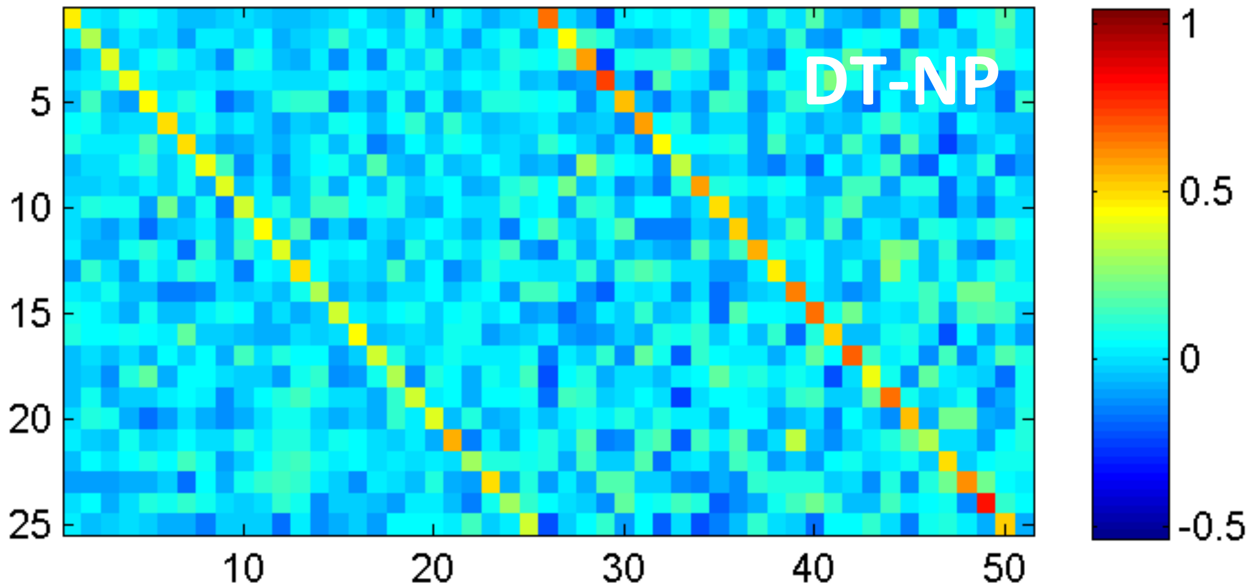
PRP$: possessive pronoun, e.g.,``my”, ``his”

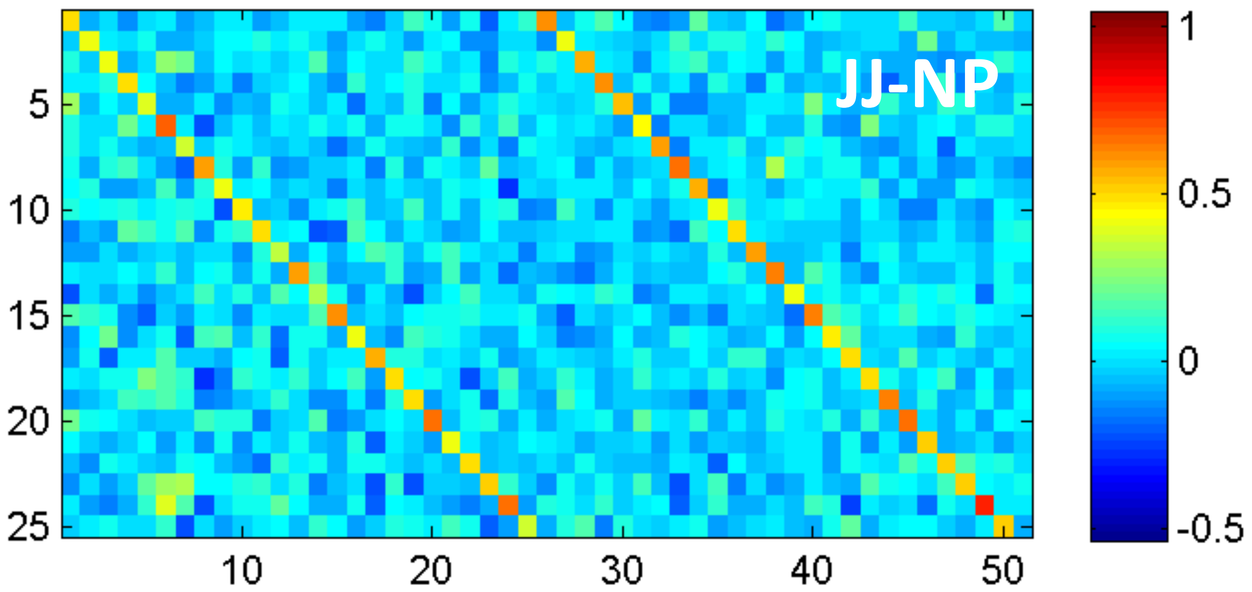


Learning relative weighting is the best you can do with such linear interactions, W1c1+W2c2

Part of speech tags: https://www.winwaed.com/blog/2011/11/08/part-of-speech-tags/

SU-RNN/CVG





Part of speech tags: https://www.winwaed.com/blog/2011/11/08/part-of-speech-tags/

Phrase similarity in Resulting Vector Representation

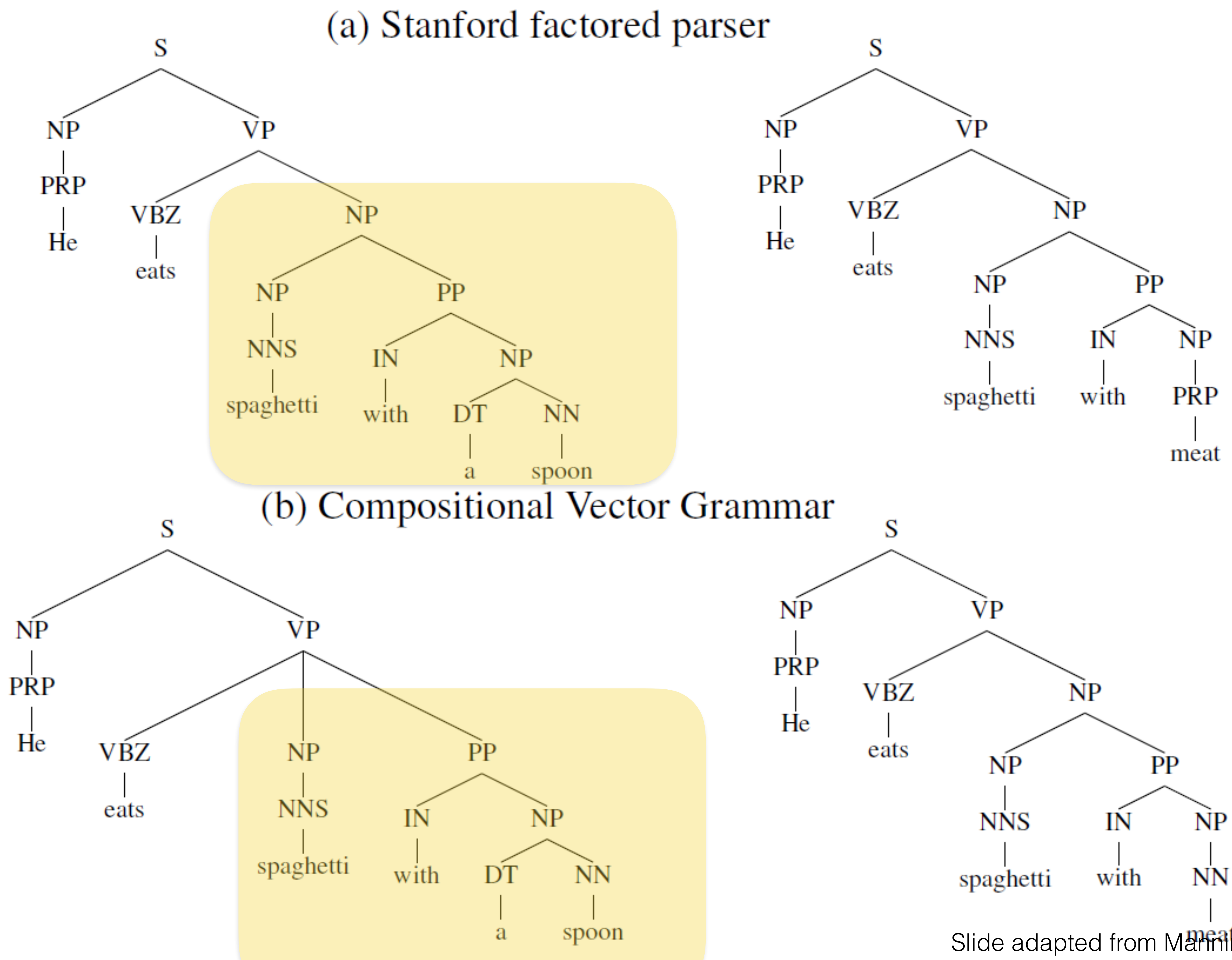
- All the figures are adjusted for seasonal variations
  - All the numbers are adjusted for seasonal fluctuations
  - All the figures are adjusted to remove usual seasonal patterns
- Knight-Ridder wouldn't comment on the offer
  - Harsco declined to say what country placed the order
  - Coastal wouldn't disclose the terms
- Sales grew almost 7% to $UNK m. from $UNK m.
  - Sales rose more than 7% to $94.9 m. from $88.3 m.
  - Sales surged 40% to UNK b. yen from UNK b.

Slide adapted from Manning-Socher

SU-RNN Analysis

- Can transfer semantic information from single related example
- Train sentences:
  - He eats spaghetti with a fork.
  - She eats spaghetti with pork.
- Test sentences:
  - He eats spaghetti with a spoon.
  - He eats spaghetti with meat.

SU-RNN Analysis

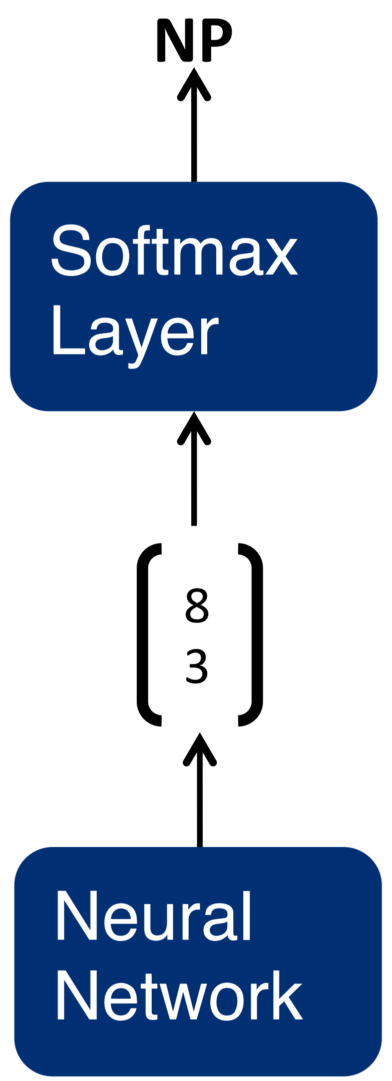


Slide adapted from Manning-Socher

Labeling

- We can use each node's representation as features for a *softmax* classifier:

$$p(c|p) \quad = \quad softmax(Sp)$$



- Training similar to model in part 1 with standard cross-entropy error + scores of composition

Slide adapted from Manning-Socher

Version 3: Recursive Matrix-Vector Spaces

Before:

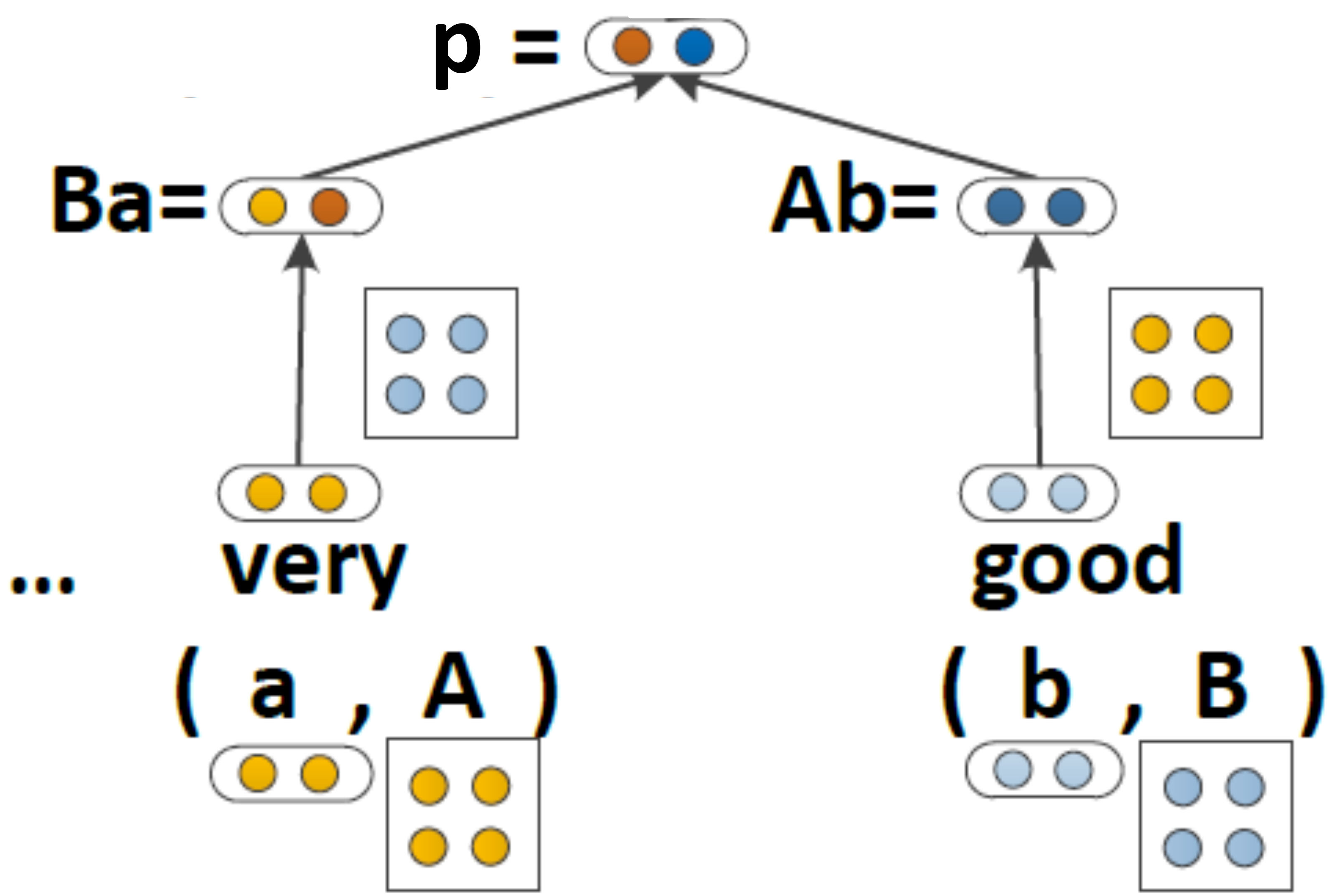
$$p = \tanh\left(W\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

- We just saw one way to make the composition function more powerful was by untying the weights W.
- But what if words act mostly as an operator, e.g. "very" in *very good, thus i do not want to take a weighted sum of the word vectors, i instead want to amplify ``good" 's vector.*

Version 3: Matrix-Vector RNNs

$$p = f\left(W\begin{bmatrix} a \\ b \end{bmatrix}\right)$$

$$\boxed{p = f\left(W\begin{bmatrix} Ba \\ Ab \end{bmatrix}\right)}$$



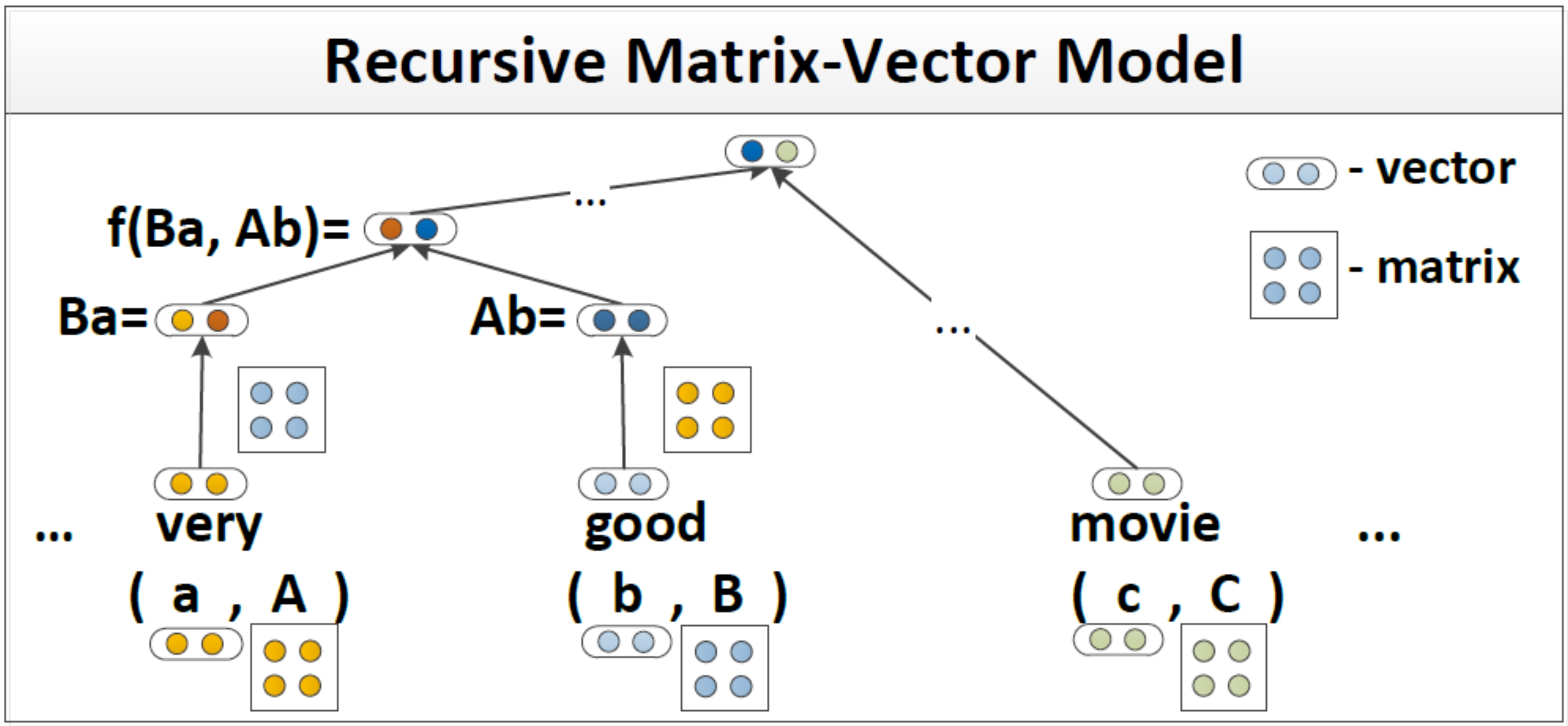
Slide adapted from Manning-Socher

Matrix-Vector RNNs

Each word is represented by both a matrix and a vector

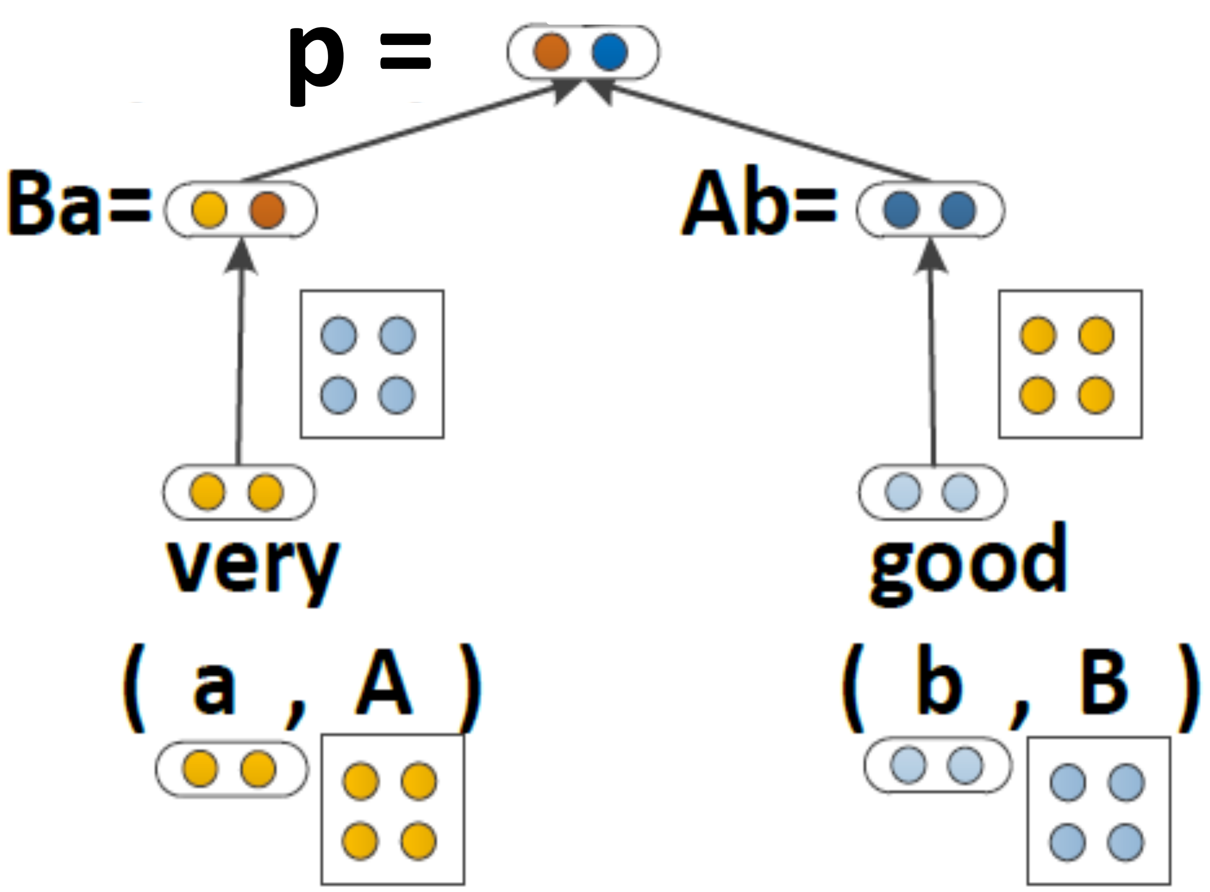
$$p = \tanh\left(W\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

$$\boxed{p = \tanh\left(W\begin{pmatrix} C_2 c_1 \\ C_1 c_2 \end{pmatrix} + b\right)}$$



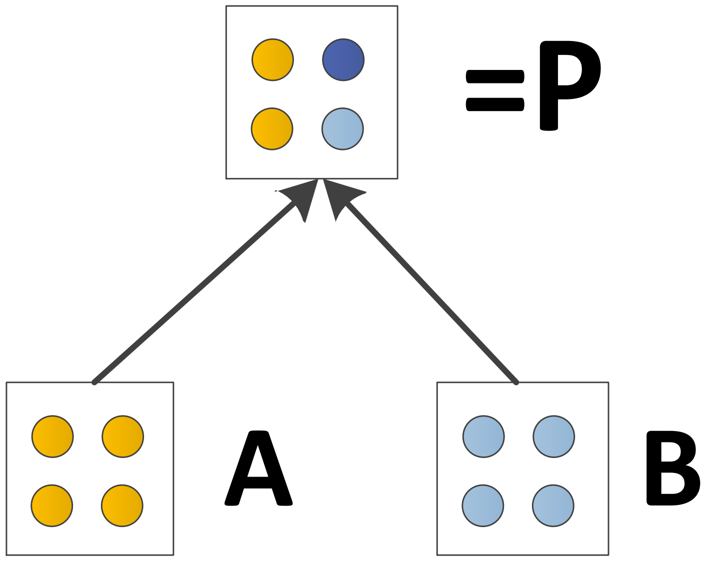
Matrix-Vector RNNs

$$p = f\left(W\begin{bmatrix} Ba \\ Ab \end{bmatrix}\right)$$



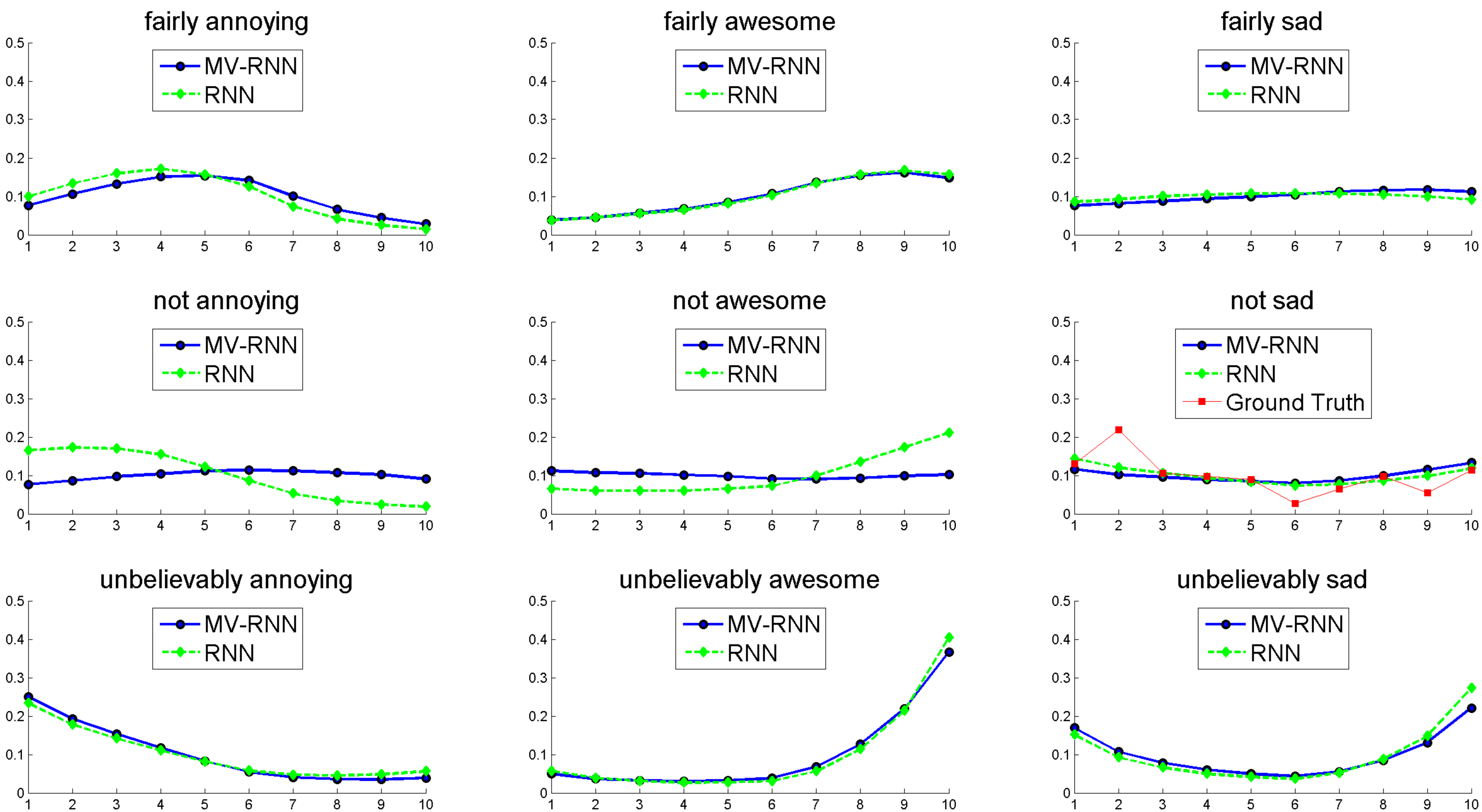
$$\boxed{P = g(A, B) = W_M\begin{bmatrix} A \\ B \end{bmatrix}}$$

$$W_M \in \mathbb{R}^{n\times 2n}$$



Predicting Sentiment Distributions

Good example for non-linearity in language



Slide adapted from Manning-Socher

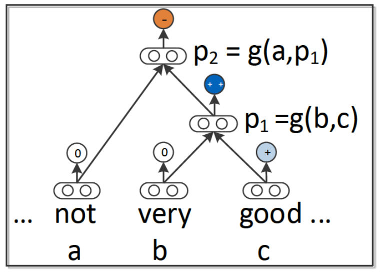
Classification of Semantic Relationships

Classifier	Features	F1
SVM	POS, stemming, syntactic patterns	60.1
MaxEnt	POS, WordNet, morphological features, noun compound system, thesauri, Google n-grams	77.6
SVM	POS, WordNet, prefixes, morphological features, dependency parse features, Levin classes, PropBank, FrameNet, NomLex-Plus, Google n-grams, paraphrases, TextRunner	82.2
RNN	–	74.8
MV-RNN	–	79.1
MV-RNN	**POS, WordNet, NER**	**82.4**

Problems with MV-RNNs

- Parameters of the model grow quadratically with the size of the vocabulary (due to matrices)
- Can we find a more economical way to have multiplicative interactions in recursive networks?
- Recursive tensor networks

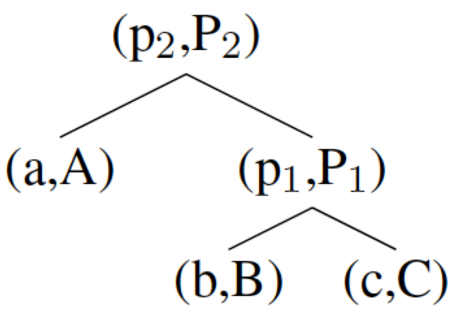
Compositional Function



- standard linear function + non-linearity, captures additive interactions:

$$p_1 = f\left(W\begin{bmatrix} b \\ c \end{bmatrix}\right), p_2 = f\left(W\begin{bmatrix} a \\ p_1 \end{bmatrix}\right)$$

- matrix/vector compositions (Socher 2011): represent each word and phrase by both a vector and a matrix. The number of parameters grows with vocabulary.



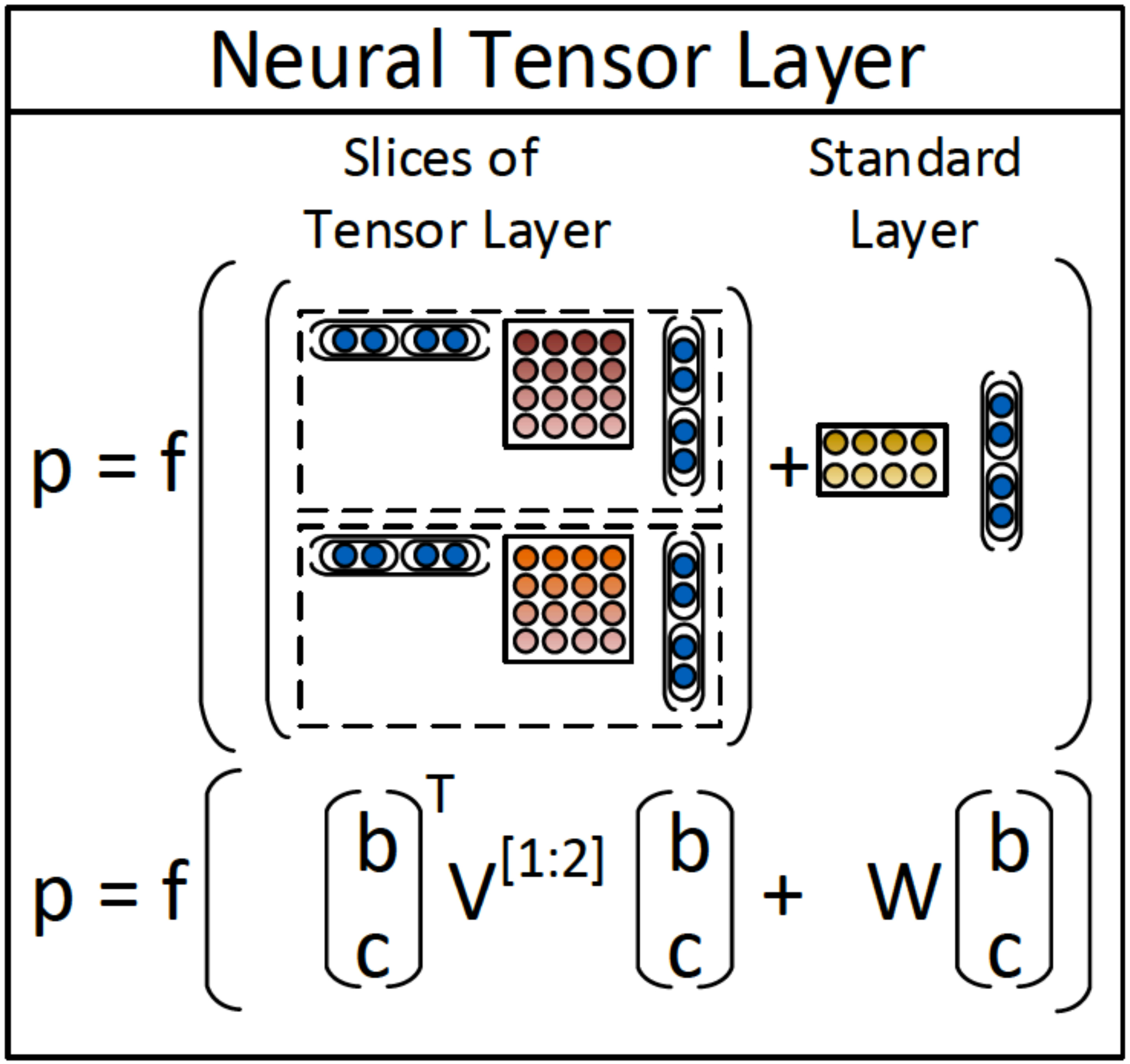
$$p_1 = f\left(W\begin{bmatrix} Cb \\ Bc \end{bmatrix}\right), P_1 = f\left(W_M\begin{bmatrix} B \\ C \end{bmatrix}\right)$$

- Recursive neural tensor networks. Parameters are both the word vectors as well as then composition tensor V, shared across all node compositions. **Q:** what is the dimensionality of V ?

$$h = \begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]}\begin{bmatrix} b \\ c \end{bmatrix}; h_i = \begin{bmatrix} b \\ c \end{bmatrix}^T V^{[i]}\begin{bmatrix} b \\ c \end{bmatrix}$$

Slide adapted from Manning-Socher

Version 4: Recursive Neural Tensor Networks



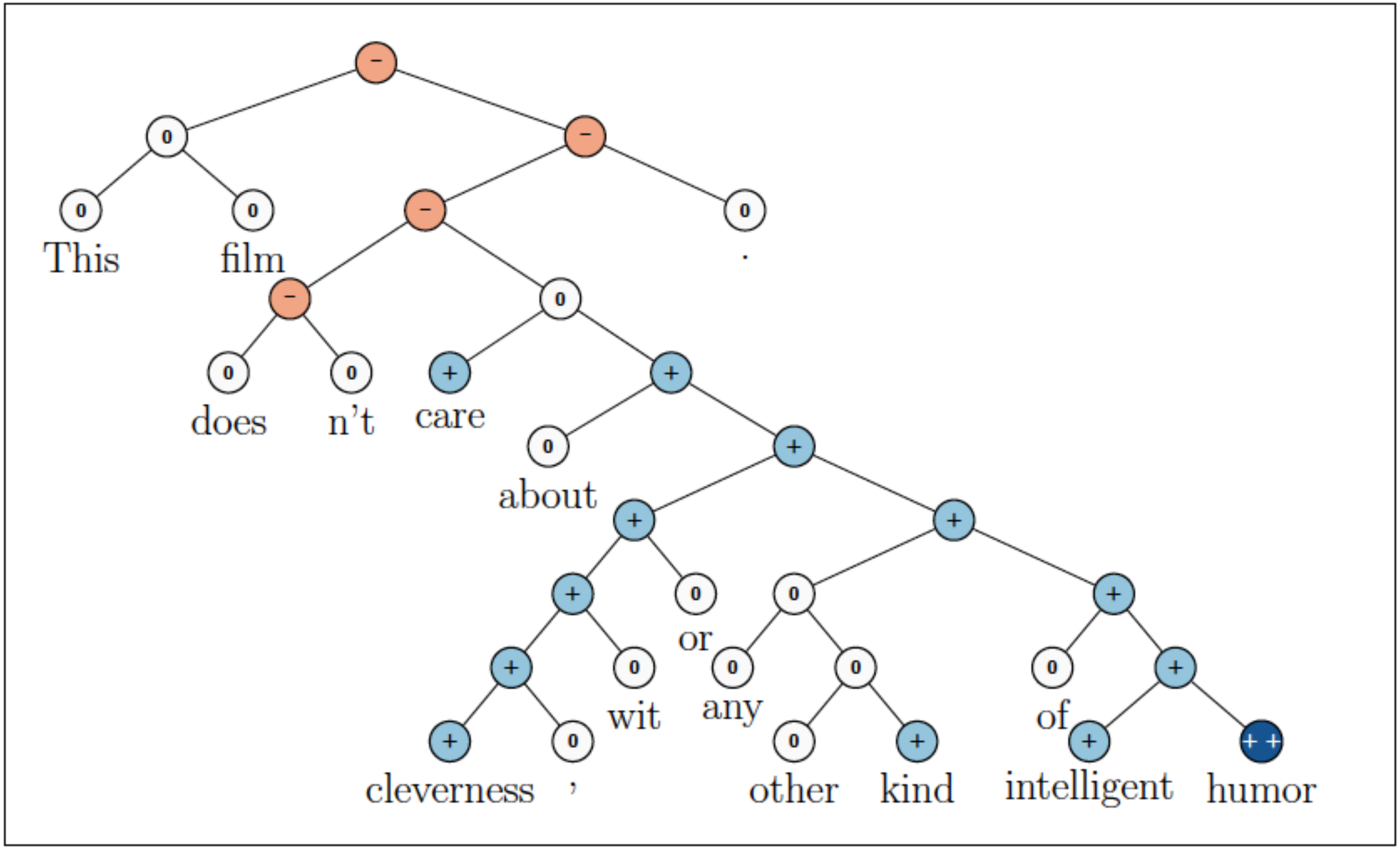
Slide adapted from Manning-Socher

Training

- We train the parameters of the model so that we minimize classification error at the root node of a sentence (e.g., sentiment prediction, does this sentence feel positive or negative?) or, at many intermediate nodes if such annotations are available:

$$E(\theta) = \sum_i \sum_j t_j^i \log y_j^i + \lambda \|\theta\|^2$$

Evaluation



Plus + and minus - indicate sentiment prediction in the different places of the sentence

Evaluation

- Using a dataset with fine grain sentiment labels for all (intermediate) phrases

Model	Fine-grained		Positive/Negative	
	All	Root	All	Root
NB	67.2	41.0	82.6	81.8
SVM	64.3	40.7	84.6	79.4
BiNB	71.0	41.9	82.7	83.1
VecAvg	73.3	32.7	85.1	80.1
RNN	79.0	43.2	86.1	82.4
MV-RNN	78.7	44.4	86.8	82.9
RNTN	**80.7**	**45.7**	**87.6**	**85.4**

Table 1: Accuracy for fine grained (5-class) and binary predictions at the sentence level (root) and for all nodes.

Evaluation

- Correctly capturing compositionality of meaning is important for sentiment analysis due to negations that reverse the sentiment, e.g., "I didn't like a single minute of this film", "the movie was not terrible" etc.

Model	Accuracy	
	Negated Positive	Negated Negative
biNB	19.0	27.3
RNN	33.3	45.5
MV-RNN	52.4	54.6
RNTN	**71.4**	**81.8**

Table 2: Accuracy of negation detection. Negated positive is measured as correct sentiment inversions. Negated negative is measured as increases in positive activations.

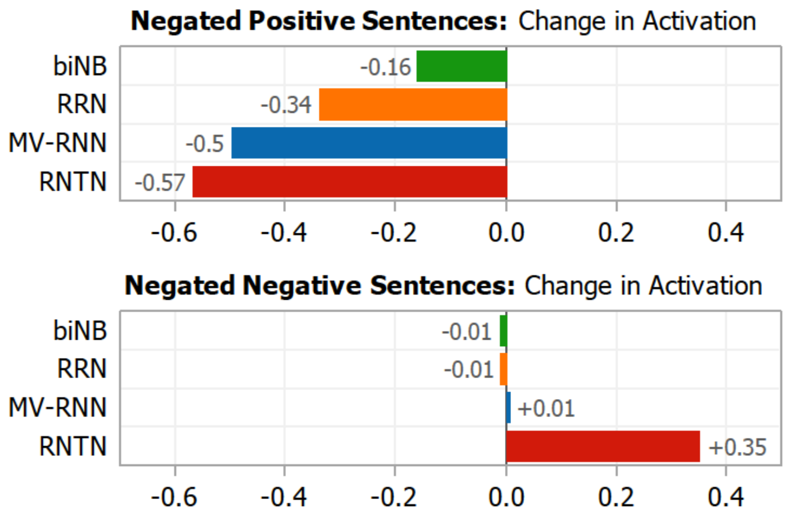
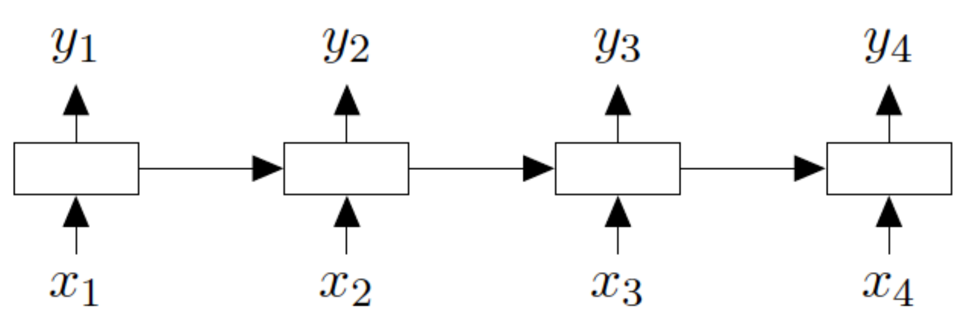
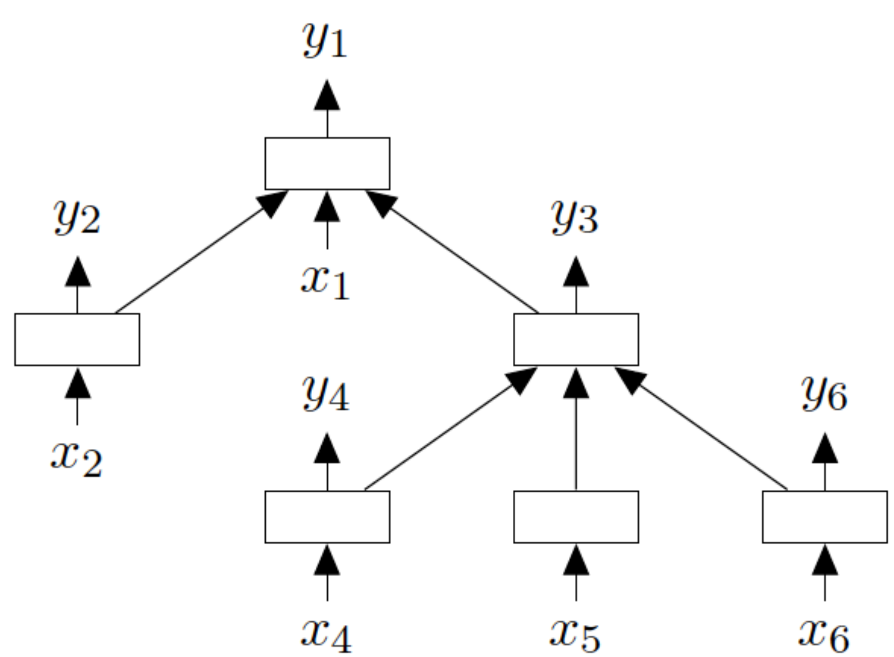


Figure 8: Change in activations for negations. Only the RNTN correctly captures both types. It decreases positive sentiment more when it is negated and learns that negating negative phrases (such as *not terrible*) should increase neutral and positive activations.

Let’s go back to vanilla trees and use LSTMs instead of RNNs



creates intermediate vectors for prefixes

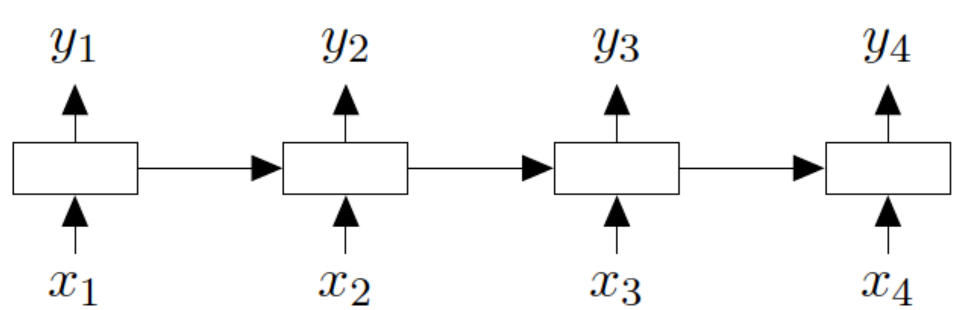


creates intermediate vectors for sub-phrases that are grammatically correct

RNNS VS LSTMS

$$h_t = \tanh\left(Wx_t + Uh_{t-1} + b\right)$$

$$\begin{aligned}
i_t &= \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}\right), \\
f_t &= \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}\right), \\
o_t &= \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}\right), \\
u_t &= \tanh\left(W^{(u)}x_t + U^{(u)}h_{t-1} + b^{(u)}\right), \\
c_t &= i_t \odot u_t + f_t \odot c_{t-1}, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}$$



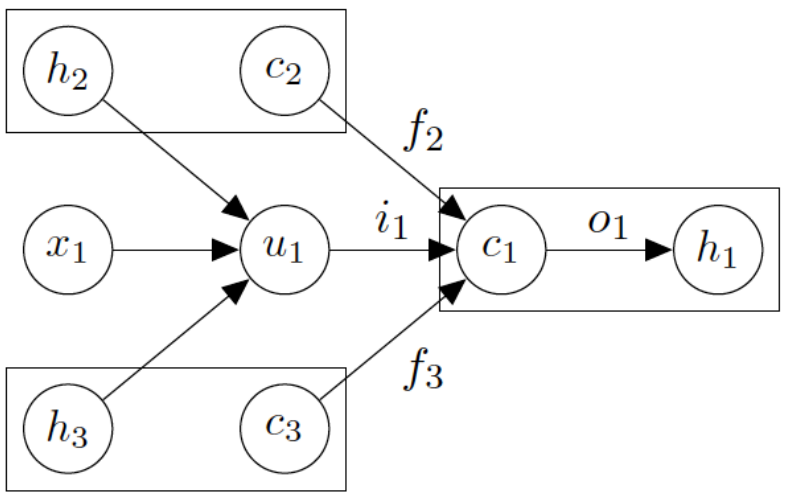
LSTMS vs Tree-LSTMS

What if we use LSTM updates not in a chain but on trees produced by SoA dependency or constituency parsers?

$$
\begin{aligned}
i_t &= \sigma\left(W^{(i)} x_t + U^{(i)} h_{t-1} + b^{(i)}\right), \\
f_t &= \sigma\left(W^{(f)} x_t + U^{(f)} h_{t-1} + b^{(f)}\right), \\
o_t &= \sigma\left(W^{(o)} x_t + U^{(o)} h_{t-1} + b^{(o)}\right), \\
u_t &= \tanh\left(W^{(u)} x_t + U^{(u)} h_{t-1} + b^{(u)}\right), \\
c_t &= i_t \odot u_t + f_t \odot c_{t-1}, \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
$$

We use a different forget gate for every child

$$
\begin{aligned}
\tilde{h}_j &= \sum_{k \in C(j)} h_k, \\
i_j &= \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right), \\
f_{jk} &= \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right), \\
o_j &= \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right), \\
u_j &= \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right), \\
c_j &= i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \\
h_j &= o_j \odot \tanh(c_j),
\end{aligned}
$$



Does children order matter?

child-sum tree LSTMS

$$\tilde{h}_j = \sum_{k \in C(j)} h_k,$$
$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right),$$
$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right),$$
$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right),$$
$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right),$$
$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k,$$
$$h_j = o_j \odot \tanh(c_j),$$

N-ary tree LSTMS

$$i_j = \sigma\left(W^{(i)} x_j + \sum_{\ell=1}^{N} U_\ell^{(i)} h_{j\ell} + b^{(i)}\right), \quad (9)$$
$$f_{jk} = \sigma\left(W^{(f)} x_j + \sum_{\ell=1}^{N} U_{k\ell}^{(f)} h_{j\ell} + b^{(f)}\right), \quad (10)$$
$$o_j = \sigma\left(W^{(o)} x_j + \sum_{\ell=1}^{N} U_\ell^{(o)} h_{j\ell} + b^{(o)}\right), \quad (11)$$
$$u_j = \tanh\left(W^{(u)} x_j + \sum_{\ell=1}^{N} U_\ell^{(u)} h_{j\ell} + b^{(u)}\right), \quad (12)$$
$$c_j = i_j \odot u_j + \sum_{\ell=1}^{N} f_{j\ell} \odot c_{j\ell}, \quad (13)$$
$$h_j = o_j \odot \tanh(c_j), \quad (14)$$

- We use Child-sum tree-LSTMs for dependency trees
- We use N-ary (in particular binary) tree LSTMs on constituency trees

Experiments

- Fine-grain and coarse grain sentiment classification
- Semantic relatedness of sentences

$$\hat{p}_\theta(y \mid \{x\}_j) = \text{softmax}\left(W^{(s)} h_j + b^{(s)}\right),$$
$$\hat{y}_j = \arg\max_y \hat{p}_\theta\left(y \mid \{x\}_j\right).$$

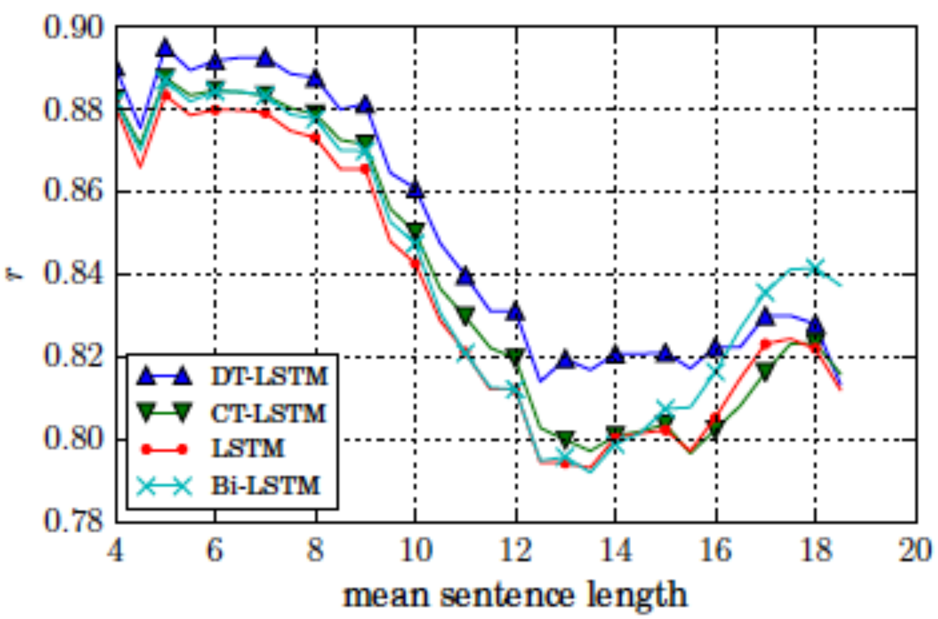
Experiments

- Fine-grain and coarse grain sentiment classification
- Semantic relatedness of sentences

Method	Fine-grained	Binary
RAE (Socher et al., 2013)	43.2	82.4
MV-RNN (Socher et al., 2013)	44.4	82.9
RNTN (Socher et al., 2013)	45.7	85.4
DCNN (Blunsom et al., 2014)	48.5	86.8
Paragraph-Vec (Le and Mikolov, 2014)	48.7	87.8
CNN-non-static (Kim, 2014)	48.0	87.2
CNN-multichannel (Kim, 2014)	47.4	**88.1**
DRNN (Irsoy and Cardie, 2014)	49.8	86.6
LSTM	46.4 (1.1)	84.9 (0.6)
Bidirectional LSTM	49.1 (1.0)	87.5 (0.5)
2-layer LSTM	46.0 (1.3)	86.3 (0.6)
2-layer Bidirectional LSTM	48.5 (1.0)	87.2 (1.0)
Dependency Tree-LSTM	48.4 (0.4)	85.7 (0.4)
Constituency Tree-LSTM		
– randomly initialized vectors	43.9 (0.6)	82.0 (0.5)
– Glove vectors, fixed	49.7 (0.4)	87.5 (0.8)
– Glove vectors, tuned	**51.0** (0.5)	88.0 (0.3)

Experiments

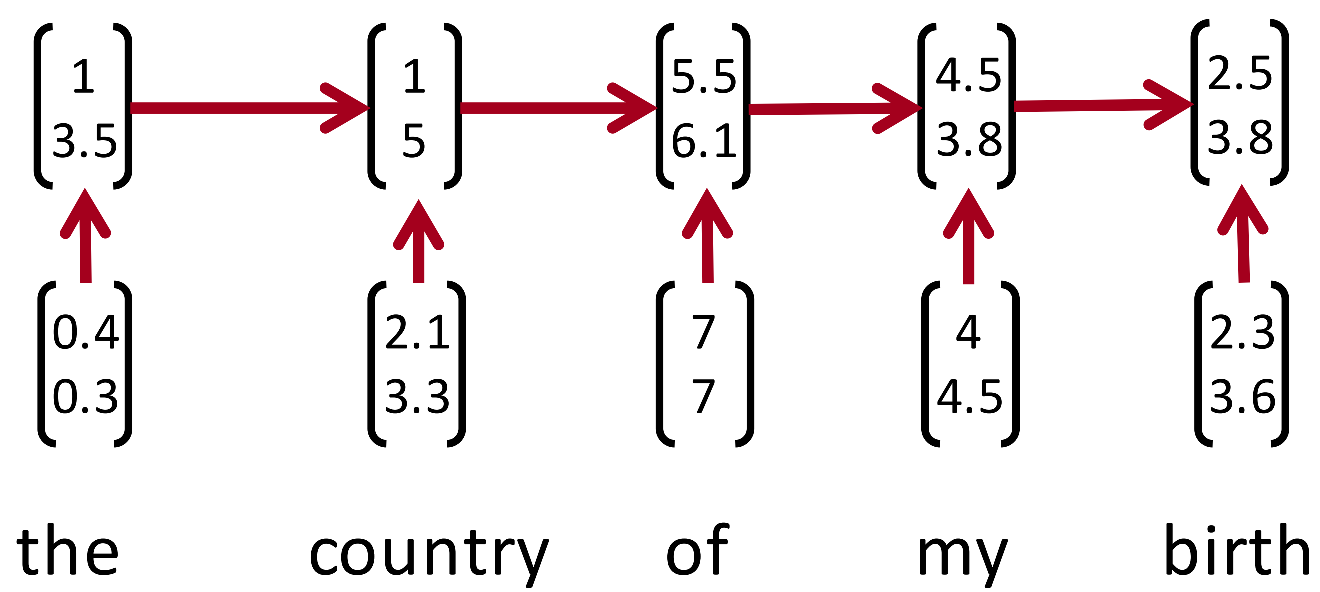
- Fine-grain and coarse grain sentiment classification
- Semantic relatedness of sentences



Method	Pearson's r		Spearman's ρ		MSE	
Illinois-LH (Lai and Hockenmaier, 2014)	0.7993		0.7538		0.3692	
UNAL-NLP (Jimenez et al., 2014)	0.8070		0.7489		0.3550	
Meaning Factory (Bjerva et al., 2014)	0.8268		0.7721		0.3224	
ECNU (Zhao et al., 2014)	0.8414		–		–	
Mean vectors	0.7577	(0.0013)	0.6738	(0.0027)	0.4557	(0.0090)
DT-RNN (Socher et al., 2014)	0.7923	(0.0070)	0.7319	(0.0071)	0.3822	(0.0137)
SDT-RNN (Socher et al., 2014)	0.7900	(0.0042)	0.7304	(0.0076)	0.3848	(0.0074)
LSTM	0.8528	(0.0031)	0.7911	(0.0059)	0.2831	(0.0092)
Bidirectional LSTM	0.8567	(0.0028)	0.7966	(0.0053)	0.2736	(0.0063)
2-layer LSTM	0.8515	(0.0066)	0.7896	(0.0088)	0.2838	(0.0150)
2-layer Bidirectional LSTM	0.8558	(0.0014)	0.7965	(0.0018)	0.2762	(0.0020)
Constituency Tree-LSTM	0.8582	(0.0038)	0.7966	(0.0053)	0.2734	(0.0108)
Dependency Tree-LSTM	**0.8676**	(0.0030)	**0.8083**	(0.0042)	**0.2532**	(0.0052)

From RNNs to CNNs

- Recurrent neural nets cannot capture phrases without prefix context.
- Often capture too much of last words in final vector.

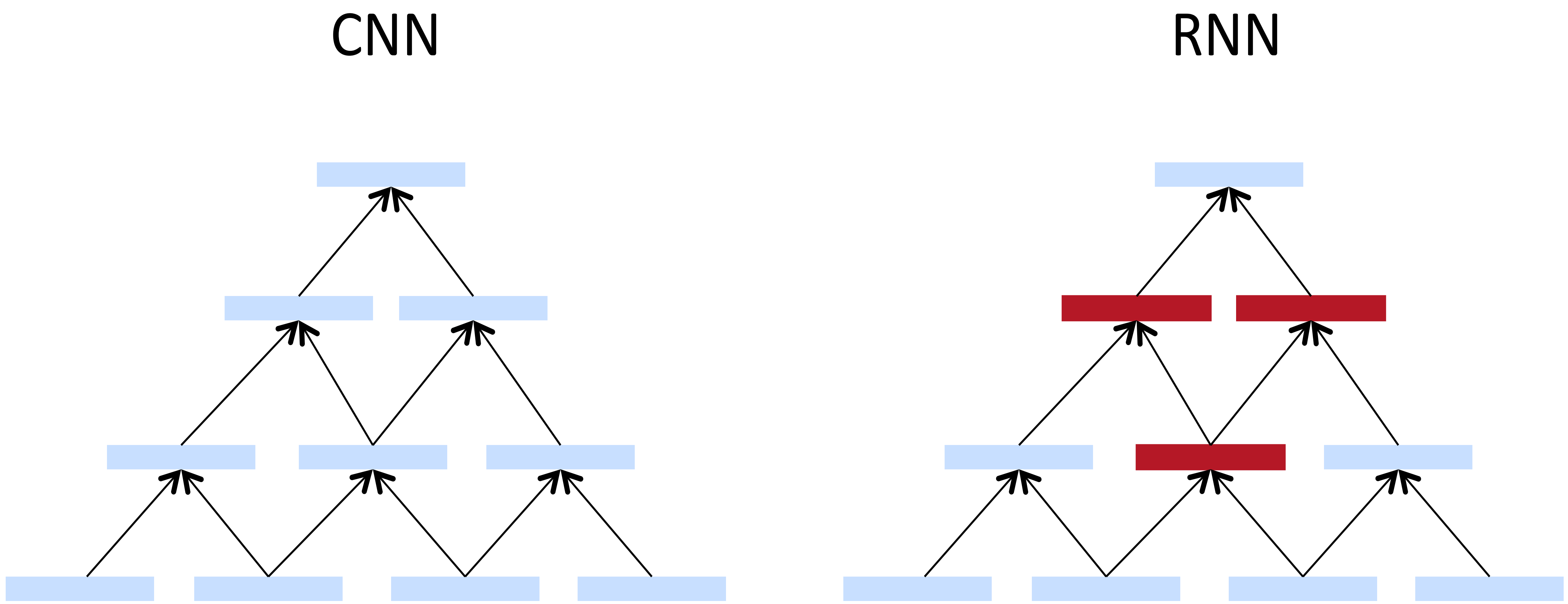


- Softmax is often only at the last step.

From RNNs to CNNs

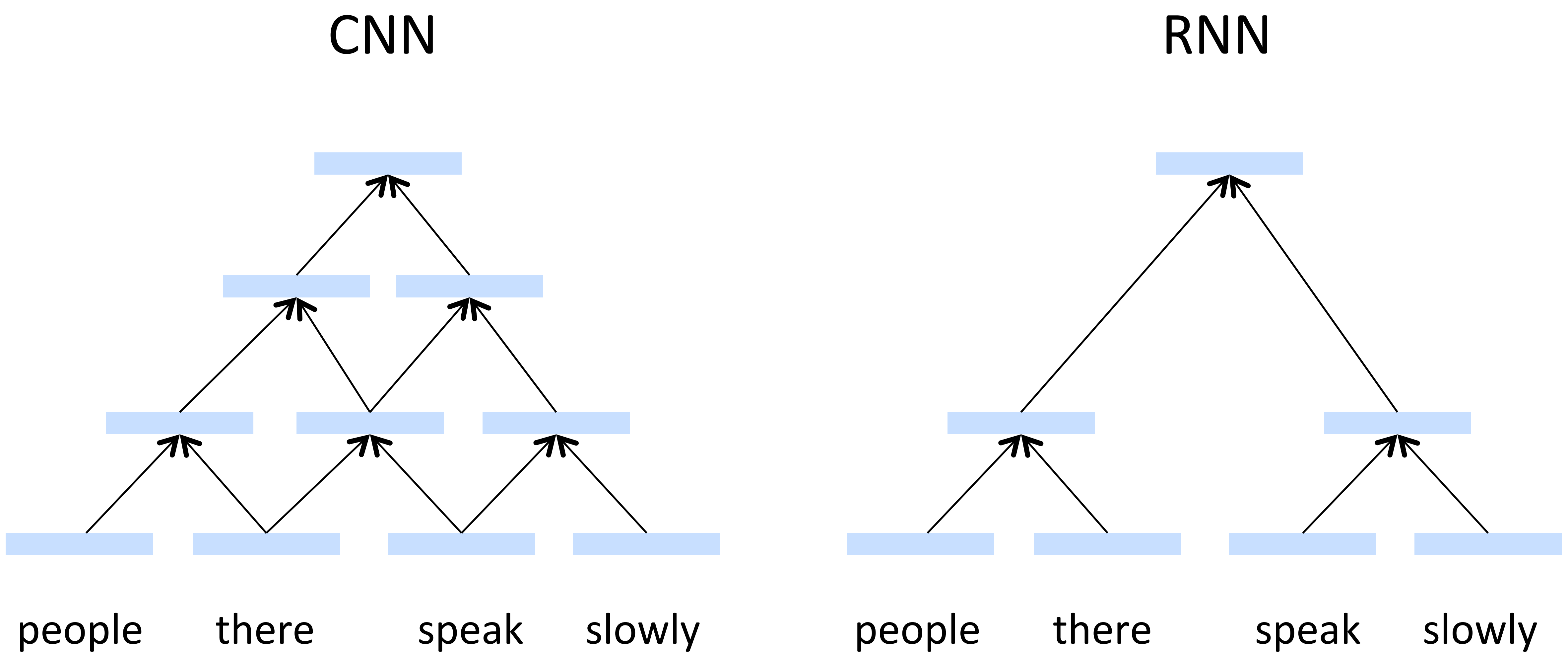
- RNN: Get compositional vectors from grammatical phrases only
- CNN: Compute vectors for every possible phrase
- Example: "the country of my birth" computes vectors for:
  - the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth
- Regardless of whether each is grammatical - many don't make sense
- Don't need parser
- But maybe not very linguistically or cognitively plausible

Relationship between CNN and RNN



Slide adapted from Manning-Socher

Relationship between CNN and RNN



representation for EVERY bigram, trigram etc.

From RNNs to CNNs

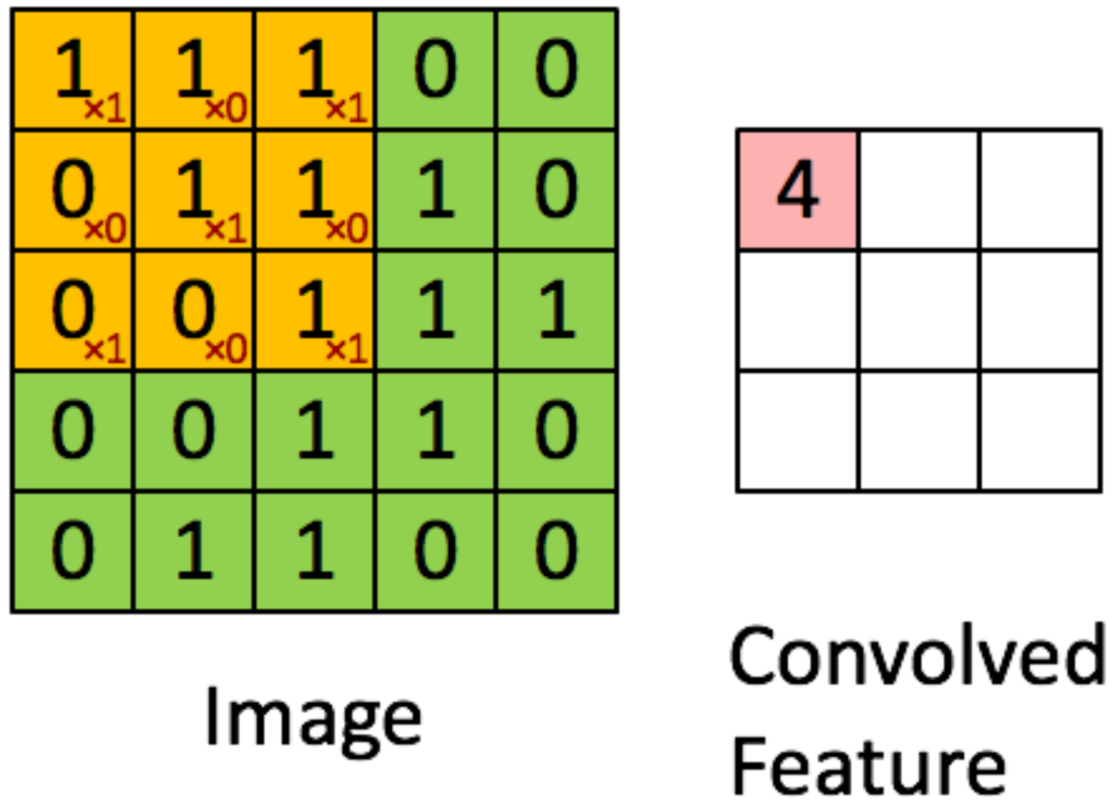
- Main CNN idea: What if we compute vectors for every possible phrase?
- Example: "the country of my birth" computes vectors for:
  - the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth
- Regardless of whether each is grammatical - not very linguistically or cognitively plausible

Convolution

- 1D discrete convolution generally:

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m].$$

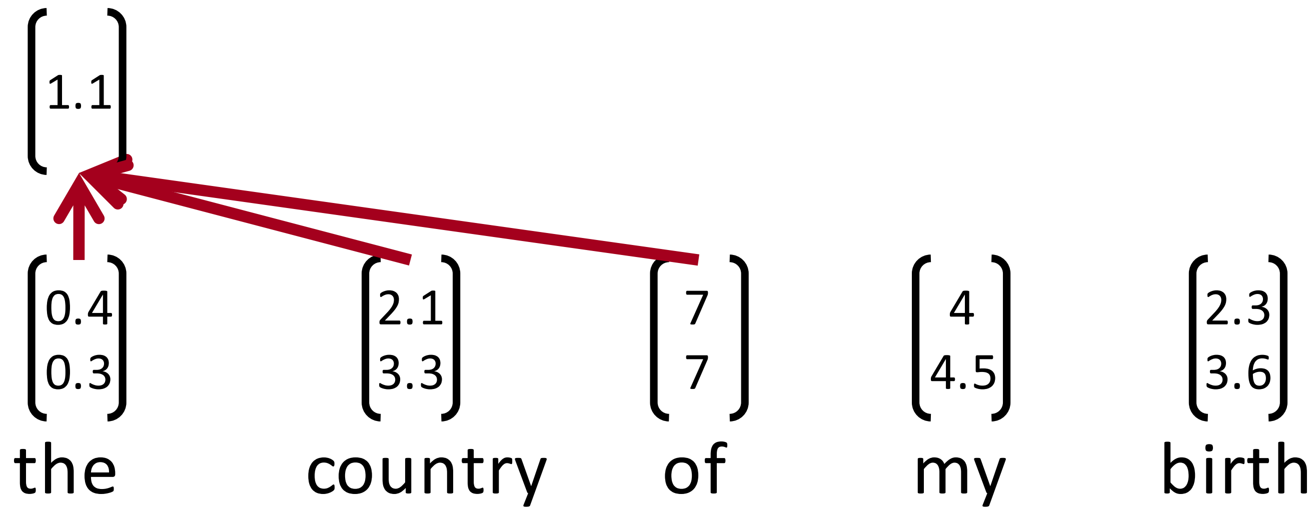
- Convolution is great to extract features from images
- 2D example:
  - Yellow and red numbers show filter weights
  - Green shows input



Slide adapted from Manning-Socher

Single Layer CNN

- A simple variant using one convolutional layer and pooling.
- Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$
- Could be 2 (as before) higher, e.g. 3:

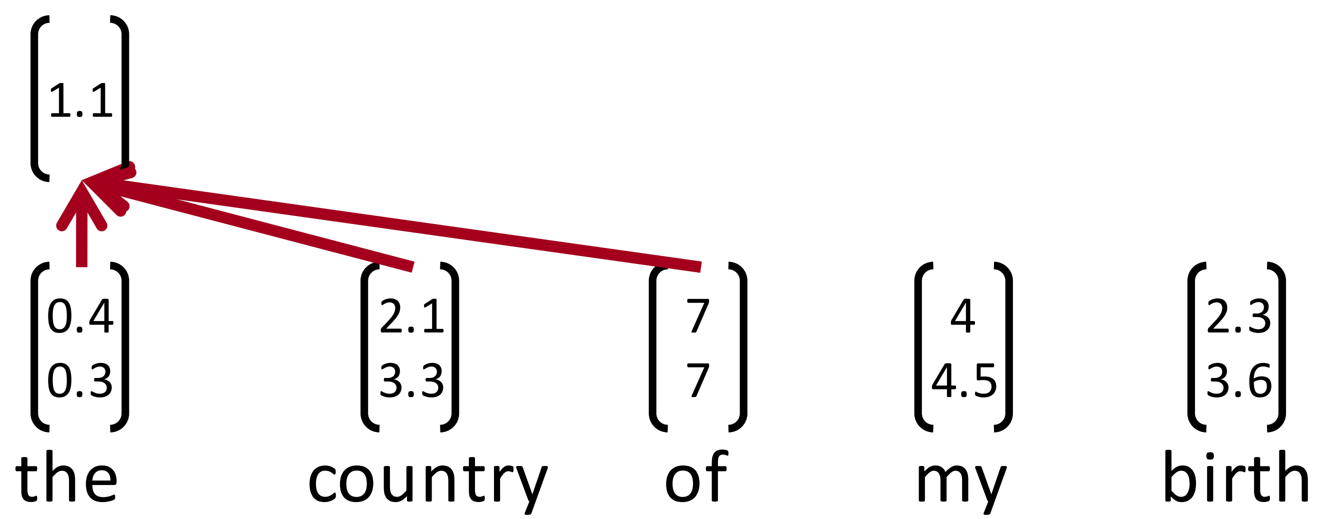


Slide adapted from Manning-Socher

Single Layer CNN

- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$
- Window size h could be 2 (as before) or higher, e.g. 3
- To compute feature for CNN layer:

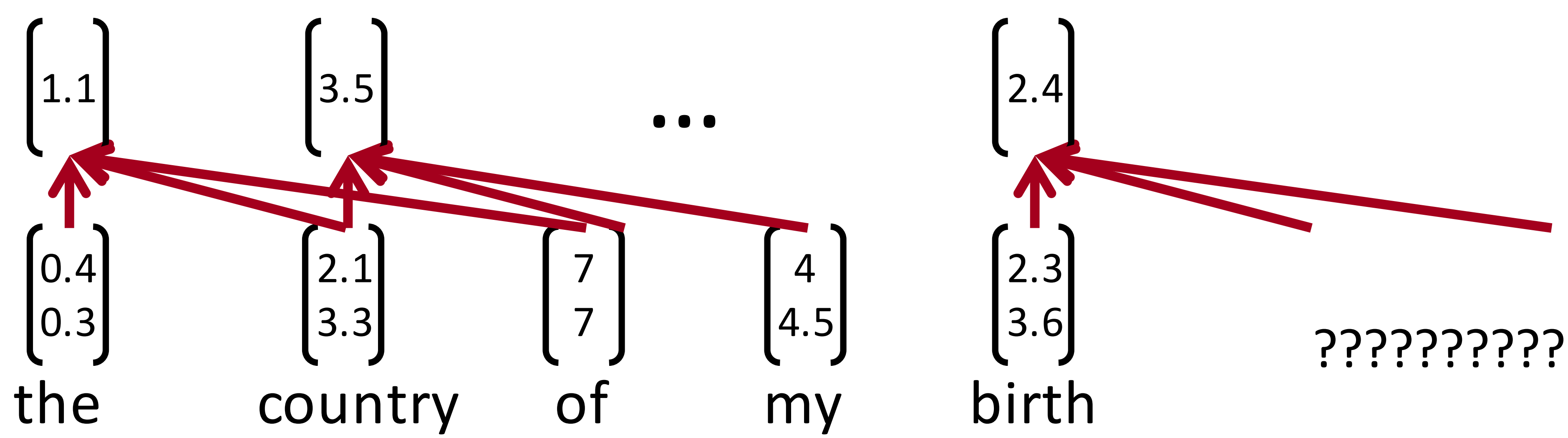
$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



Slide adapted from Manning-Socher

Single Layer CNN

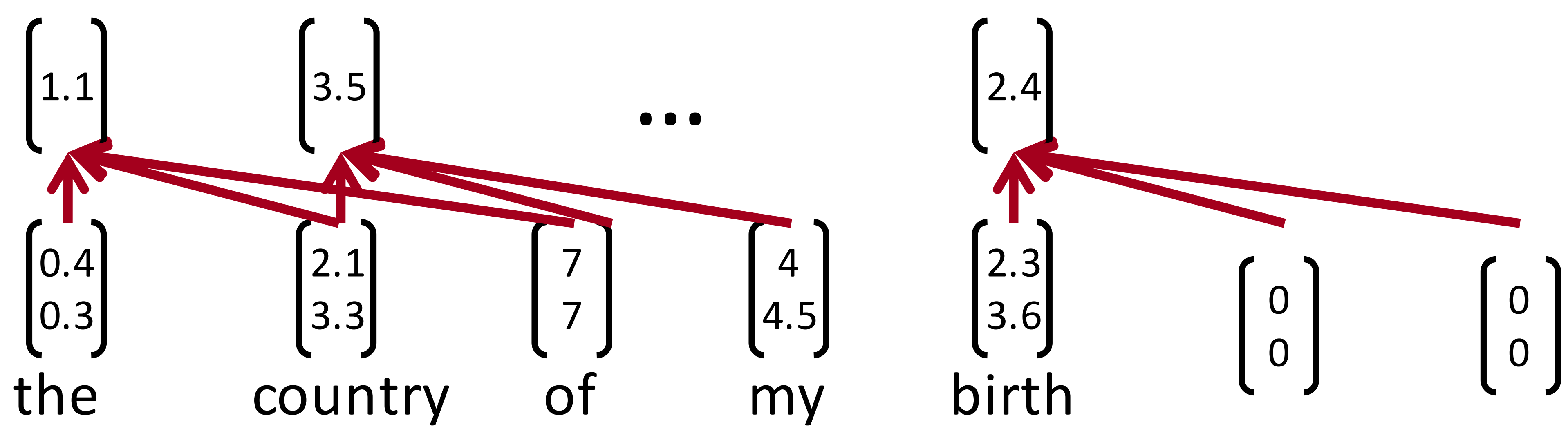
- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length h: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Slide adapted from Manning-Socher

Single Layer CNN

- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length h: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Slide adapted from Manning-Socher

Single Layer CNN: Pooling

- New building block: Pooling
- In particular: max-over-time pooling layer
- Idea: Capture most important activation (maximum over time)
- From feature map $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$
- But we want more features!

Solution: Multiple Filters

- Use multiple filter weights w
- Useful to have different window sizes h
- Because of max pooling, length of c is irrelevant

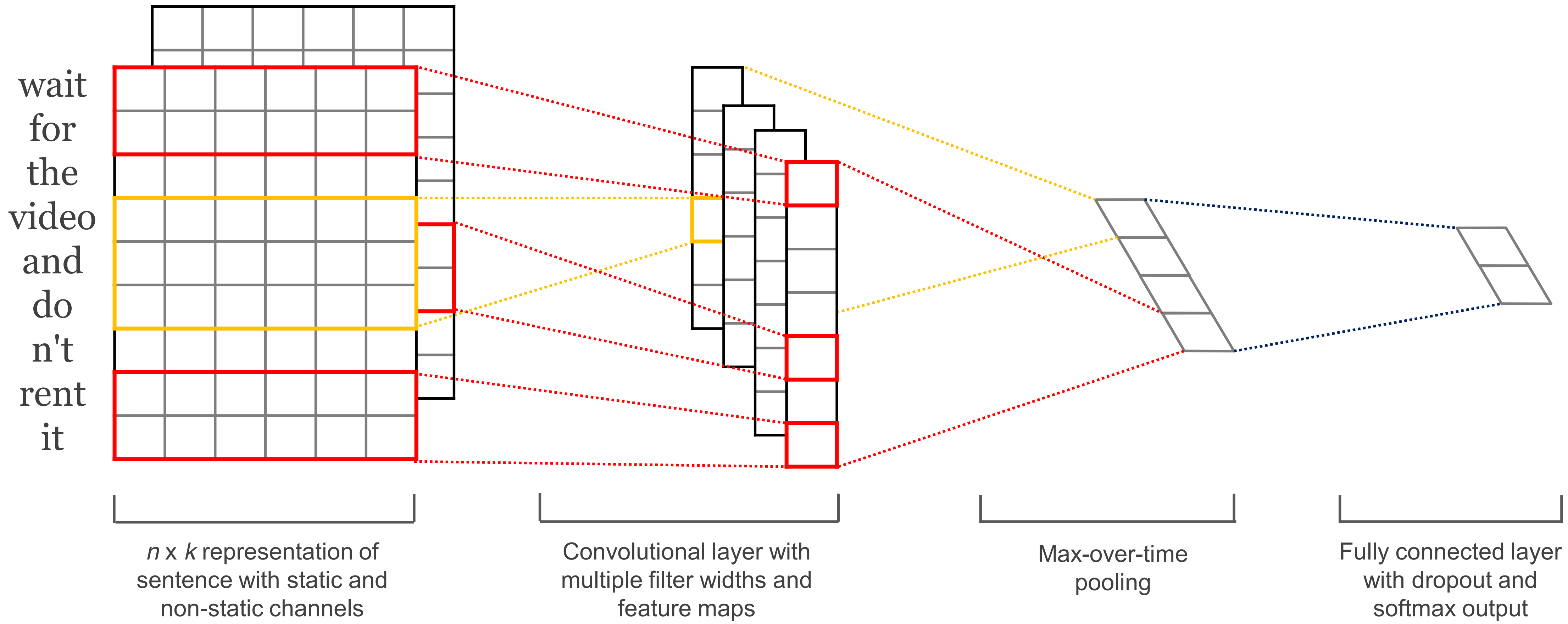
$$\mathbf{c} = \left[c_1, c_2, \ldots, c_{n-h+1}\right] \in \mathbb{R}^{n-h+1}$$

- So we can have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

Classification after one CNN Layer

- First one convolution, followed by one max-pooling
- To obtain final feature vector: $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$
  - Assuming m filters w
- Simple final softmax layer $y = softmax\left(W^{(S)}z + b\right)$

Classification after one CNN Layer



n words (possibly zero padded) and each word vector has k dimensions

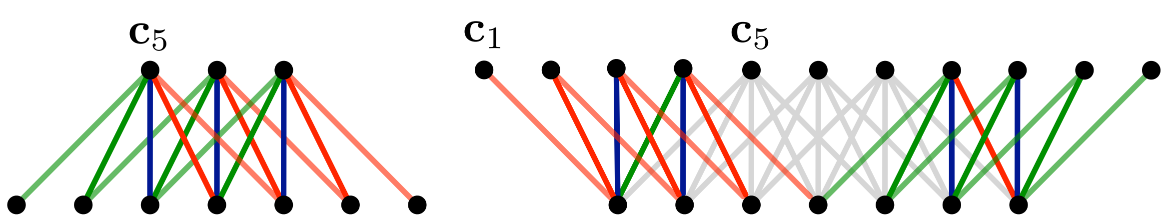
Slide adapted from Manning-Socher

Experiments

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Beyond a single layer: adaptive pooling

- Narrow vs. wide convolution



- Complex pooling schemes (over sequences) and deeper convolutional layers