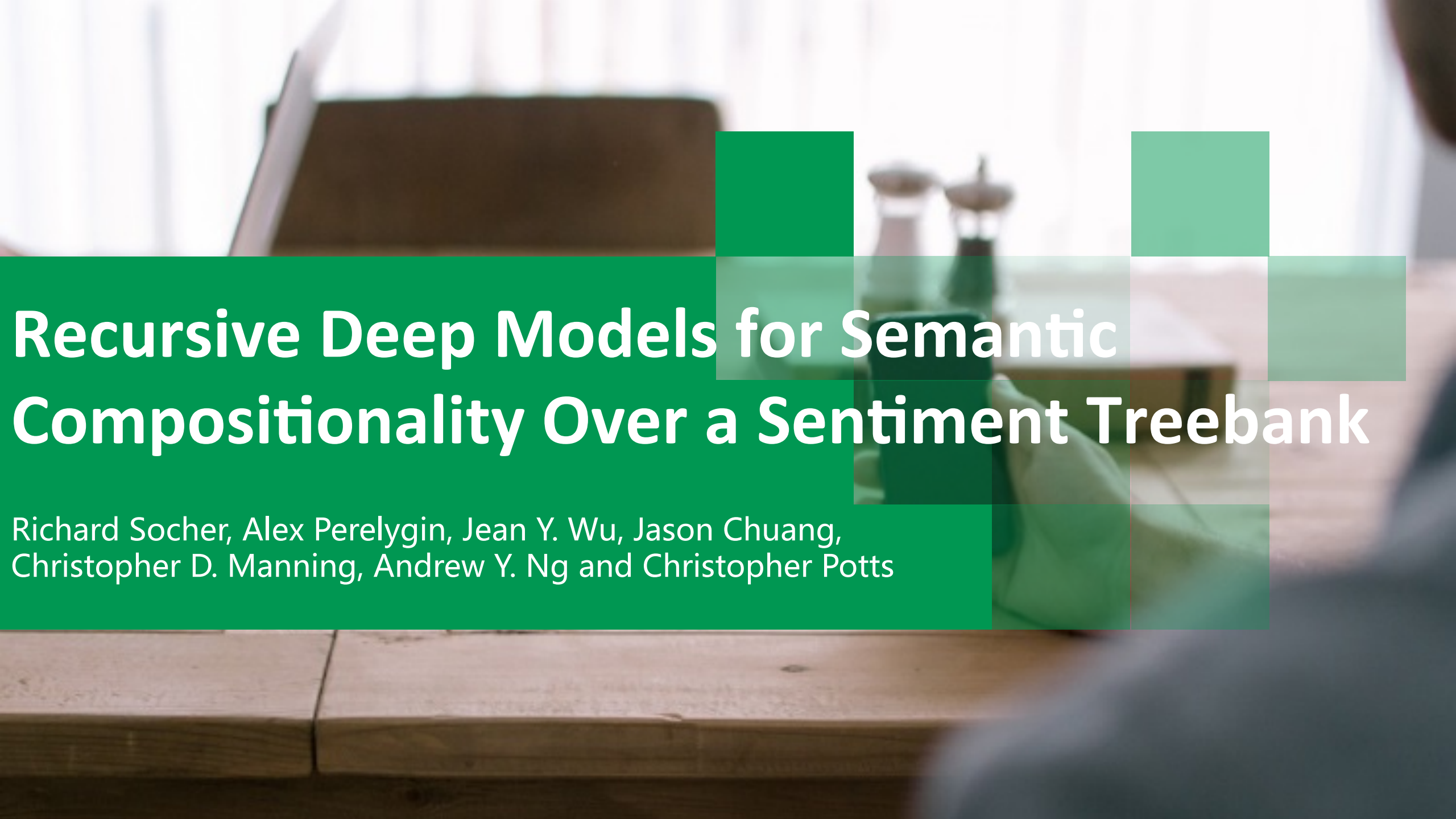


# Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang,
Christopher D. Manning, Andrew Y. Ng and Christopher Potts

# CONTENTS

# 1 Introduction

# What is sentiment analysis?

- This movie is really disappointing

- This is the greatest comedy ever filmed

- Full of zany characters and richly applied satire, and some great plot twists

- It was pathetic. The worst part about it was the boxing scenes.

# Why sentiment analysis?

- *Movie*:  is this review positive or negative?
- *Products*: what do people think about the new iPhone?
- *Public sentiment*: how is consumer confidence? Is despair increasing?
- *Politics*: what do people think about this candidate or issue?
- *Prediction*: predict election outcomes or market trends from sentiment

2

# Related Research

# Related Research

- Only work well in long documents.
- Rely on a few words with strong sentiment like ‘awesome’
- Ignore word order

For single sentence :

- sentiment accuracies even for binary positive/negative classification has not exceeded 80% for several years.
- the more difficult multiclass case including a neutral class, accuracy is often below 60%

Past Algorithms:

- Rely on a few words with strong sentiment
- Ignore word order

# What we want

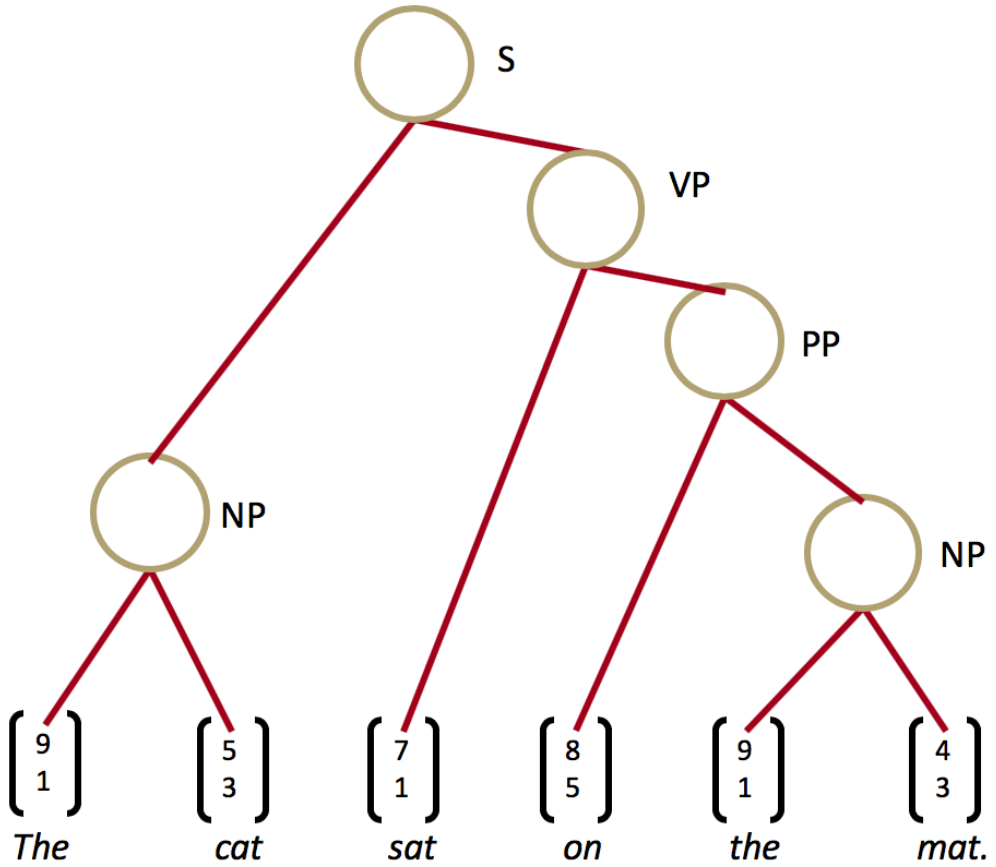
Vectors representing
phrases and sentences
that do not ignore word order
and capture semantics for NLP tasks

# How can we map phrases and Sentences into a vector space

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

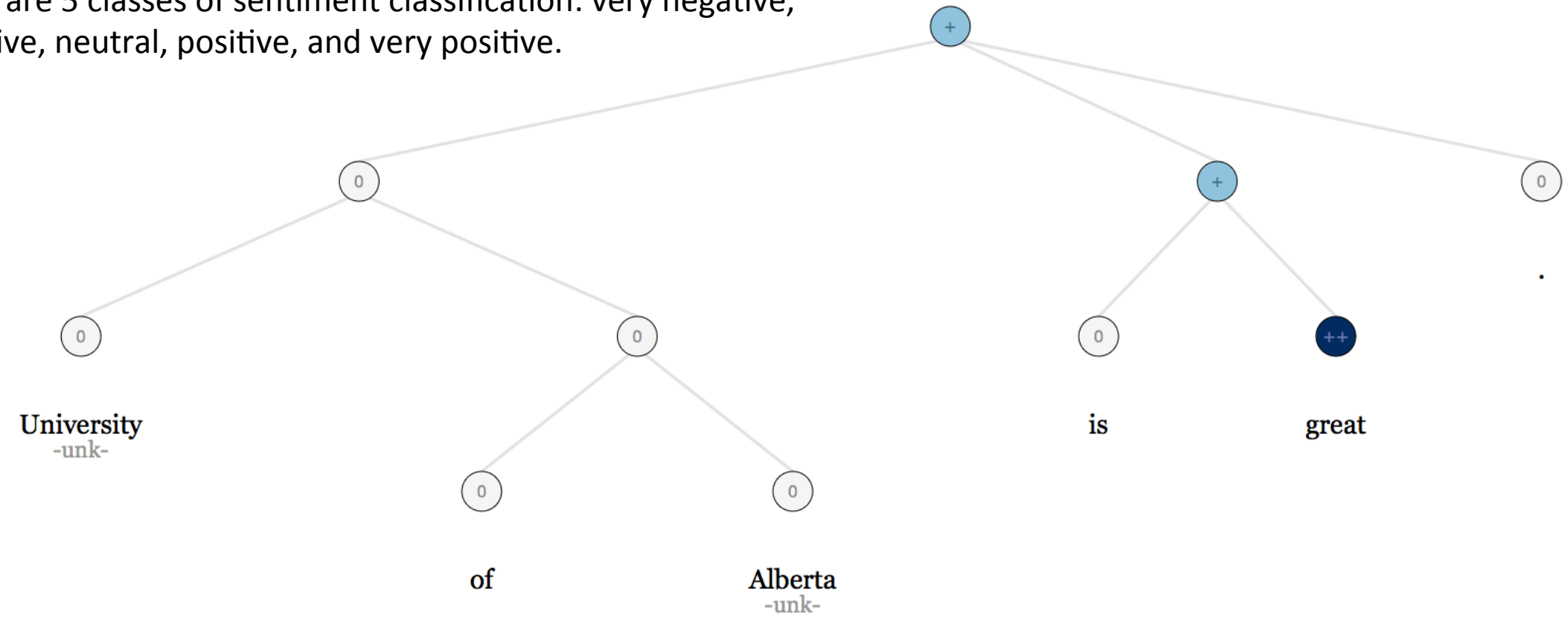
(1) Each word is represented as a d-dimensional vector.

(2) Use parse tree to get the structure of the sentence.



# Demo

There are 5 classes of sentiment classification: very negative, negative, neutral, positive, and very positive.



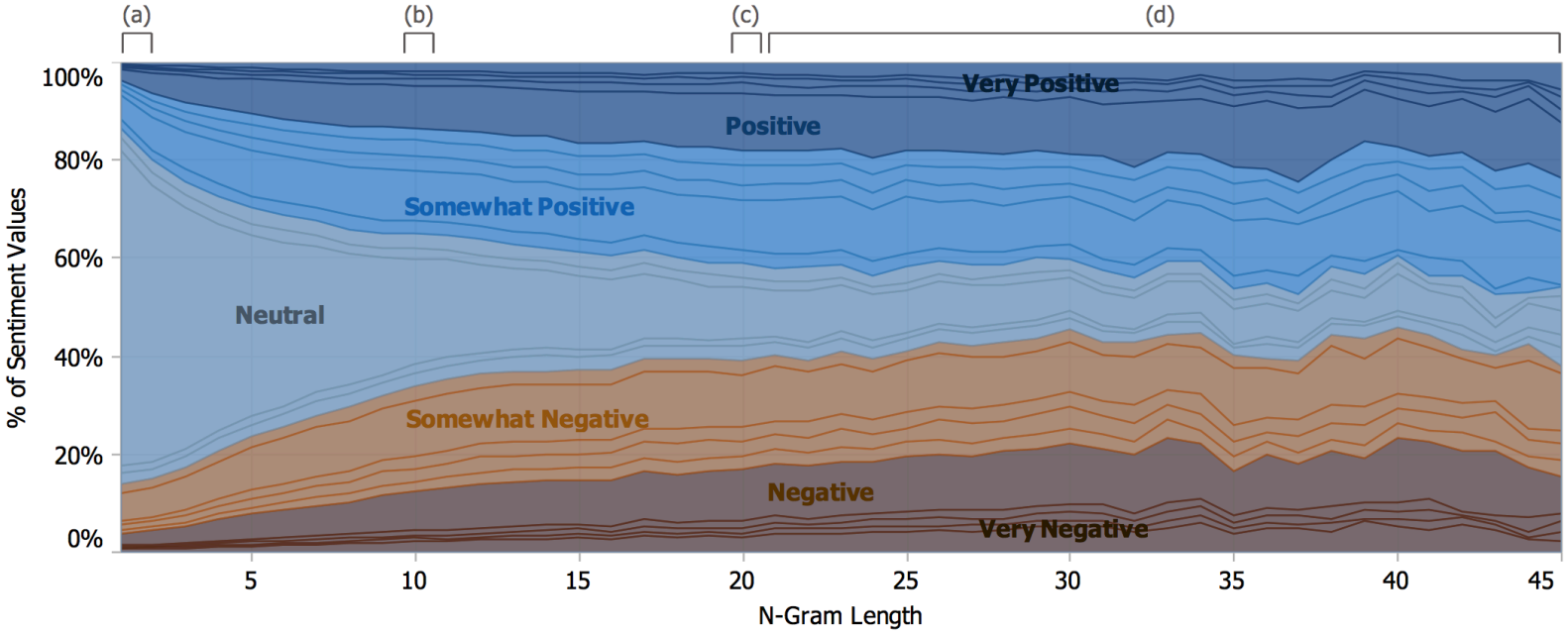

http://nlp.stanford.edu:8080/sentiment/rntnDemo.html

3

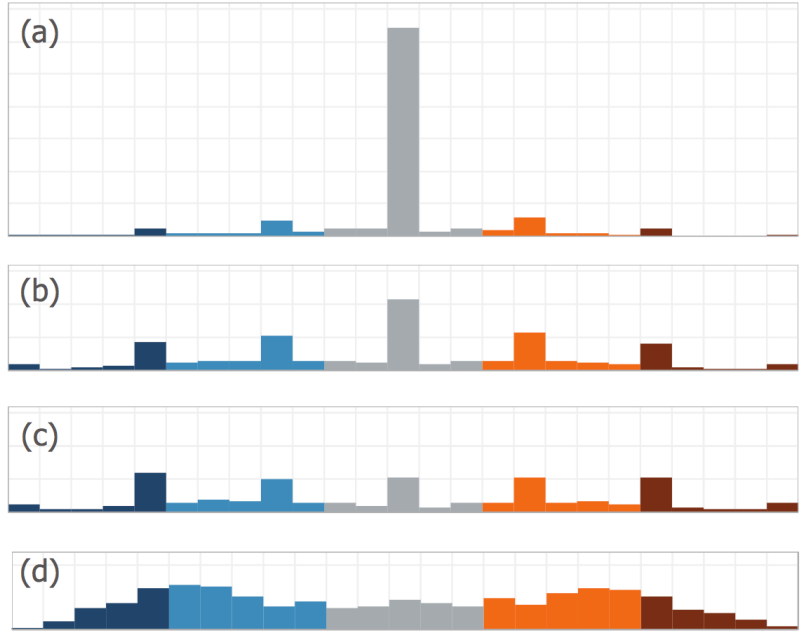
# Sentiment Treebank

# Sentiment Treebank

Fine grained sentiment labels for 215,154 phrases in the parse trees of 11,855 sentences



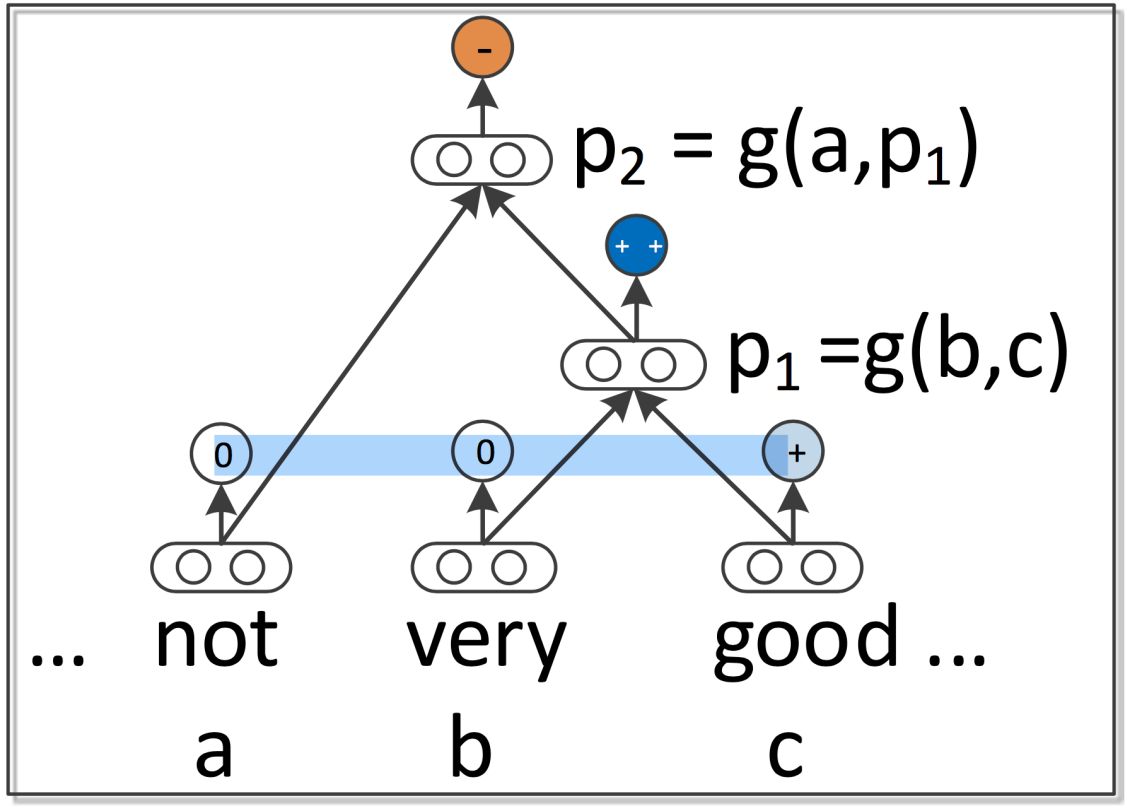

Distributions of sentiment values for (a) unigrams, (b) 10-grams, (c) 20-grams, and (d) full sentences.




Many shorter n-grams are neutral
Longer phrases are well distributed

# 4 Recursive Neural Models

# First Model: Standard Recursive Neural Network




Compute parent vectors in a bottom up fashion

Each word is represented as a d-dimensional vector.

First, it is determined which parent already has all its children computed. In the tree example, P1 has its two children's vectors since both are words. RNNs use the following equations to compute the parent vectors P1:

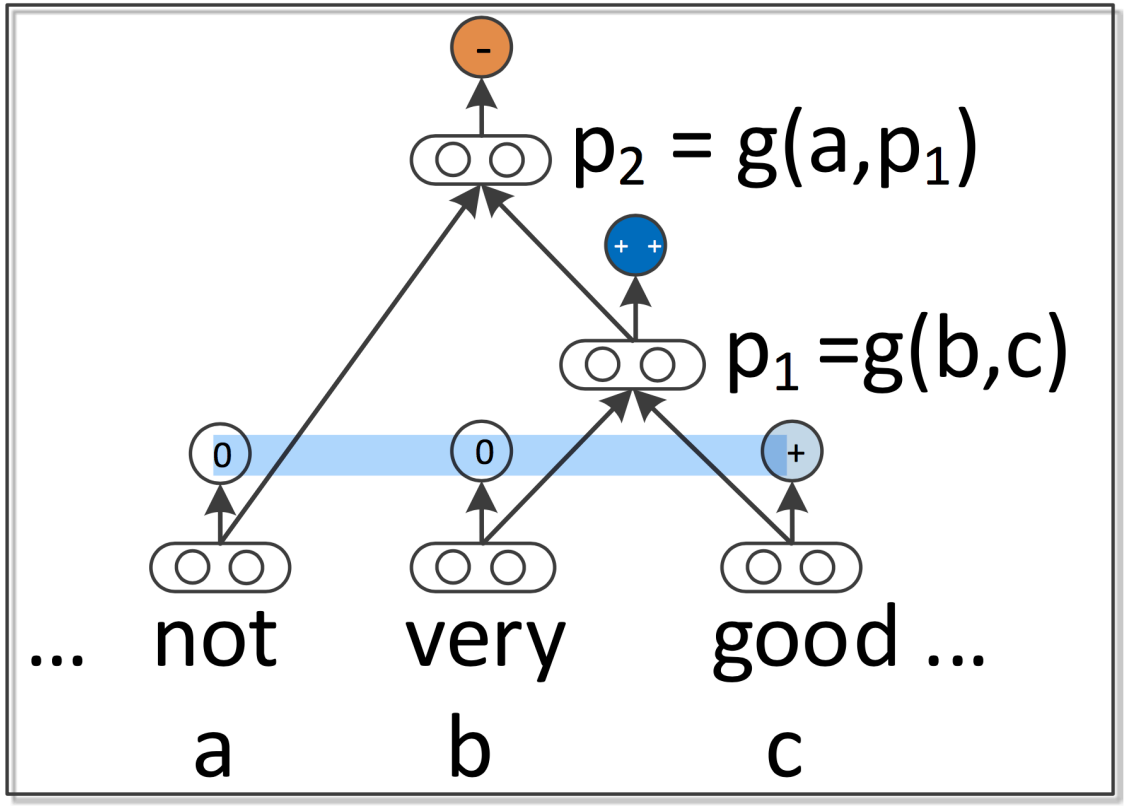
$$p_1 = f\left(W\begin{bmatrix} b \\ c \end{bmatrix}\right)$$

f = tanh is a standard element-wise nonlinearity,
$W \in R(d \times 2d)$ is the main parameter to learn.

Then, P2 found its two children's vectors are both computed, so now it's time to calculate P2.

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

# First Model: Standard Recursive Neural Network




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

Each parent vector Pi (P1, P2 in this example), is given to the same softmax classifier to compute its label probabilities.

$$y^a = \text{softmax}(W_s a)$$

$W_s \in R^{5\times d}$ is the sentiment classification matrix.

# First Model: Standard Recursive Neural Network

It is still not expressive enough.

what if words act mostly as an operator,
e.g. “very” in *very good*

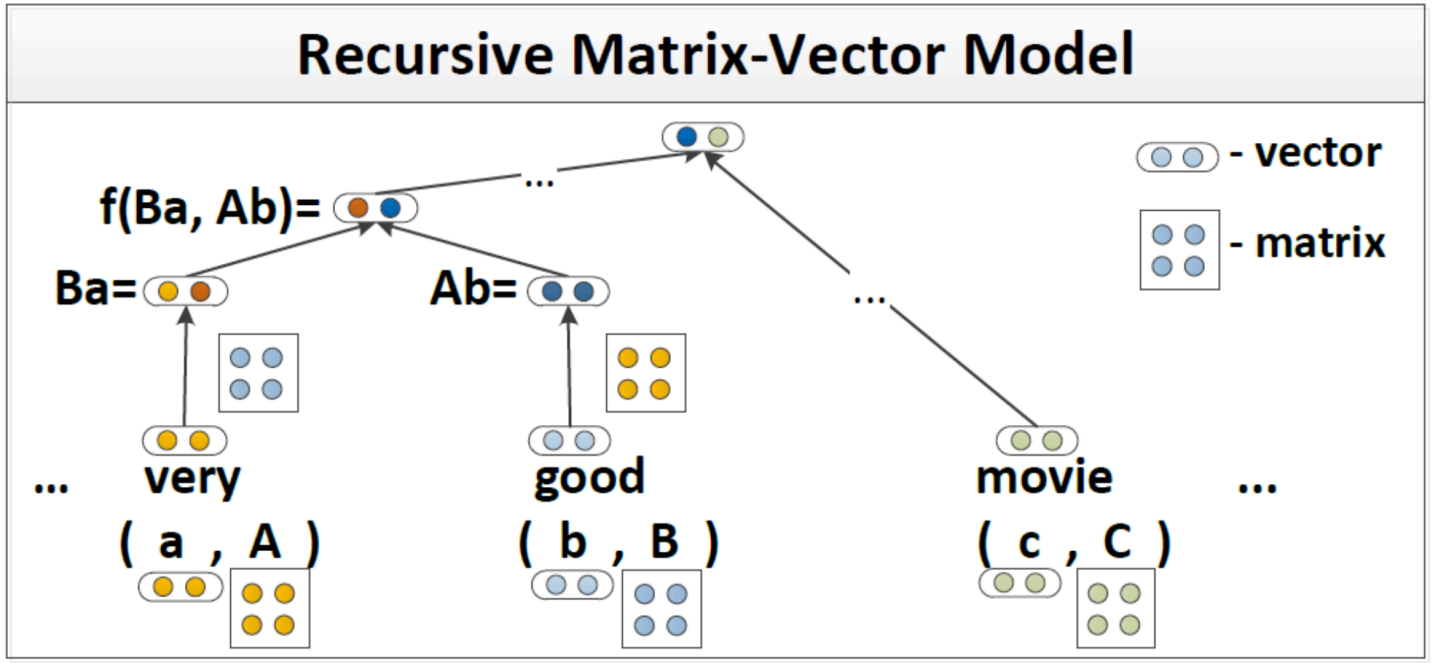
How can we have a vector that emphasizes any other vector in this model?

**We can not!**

**We need to have some form of multiplication of word on another word.**

# Second Model: Matrix-Vector RNNs

**Each word is represented as a d-dimensional vector and a word matrix!**

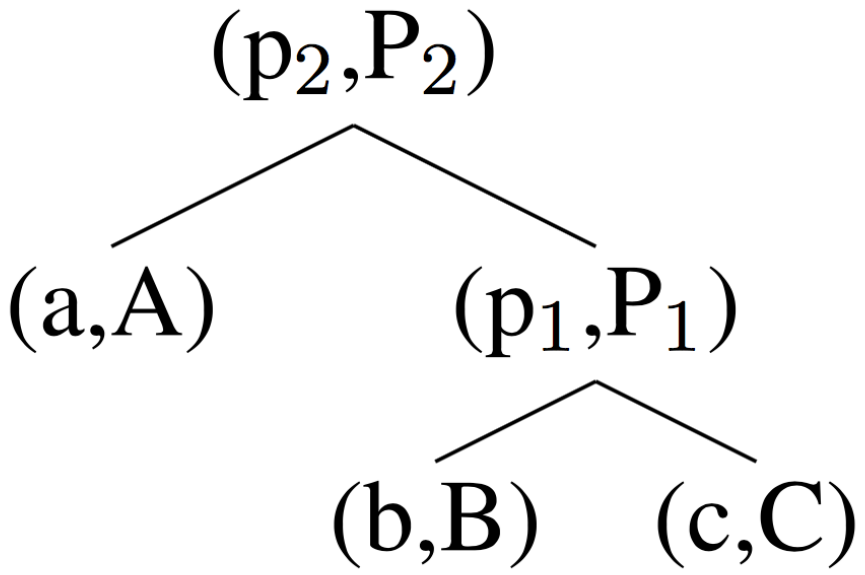



**The word "very" will have a word vector** $V_{very} \in R^d$

**but also a matrix** $V_{very} \in R^{d \times d}$

**It's a way that words "modify" other words!**

# Second Model: Matrix-Vector RNNs




$b \in R^d \quad B \in R^{d \times d}$

The MV-RNN computes the first parent vector and its matrix via two equations:

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

$$W \in R^{d \times 2d}, W_M \in R^{d \times 2d}, p_1 \in R^d, P_1 \in R^{d \times d}$$

The second parent node is computed using the previously computed (vector, matrix) pair ($p_1$, $P_1$) as well as (a, A).

The vectors are used for classifying each phrase using the same softmax classifier as before.

$$y^a = \mathrm{softmax}(W_s a)$$

# Second Model: Matrix-Vector RNNs

**The number of parameters becomes very large!**

# Third Model: Recursive Neural Tensor Network

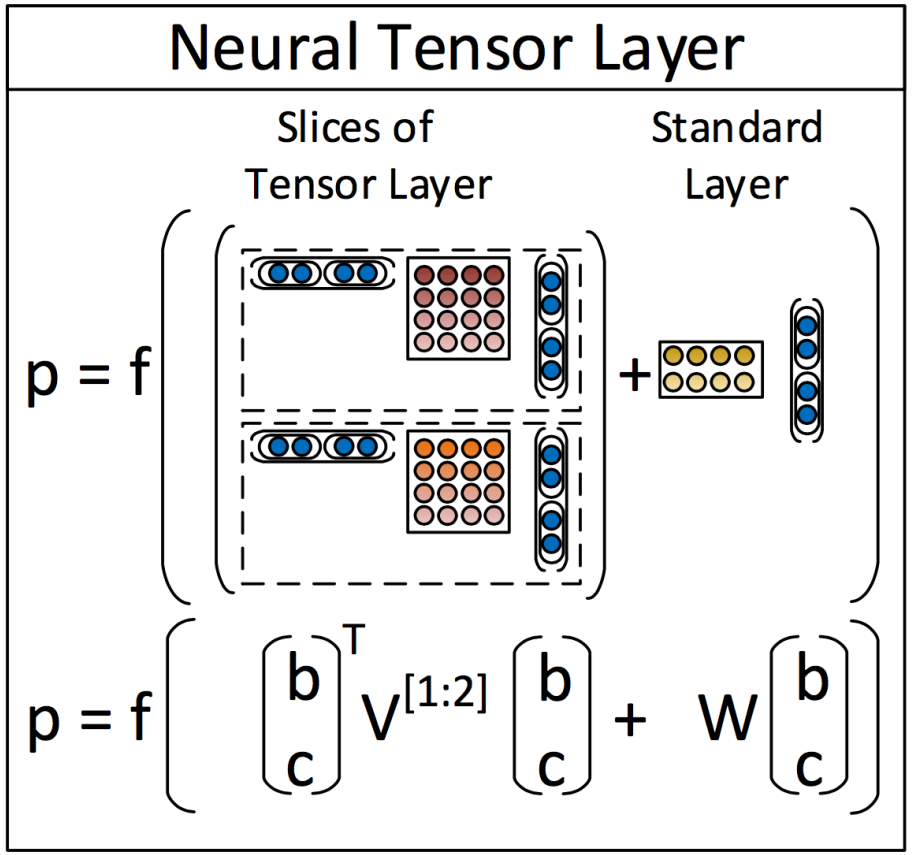
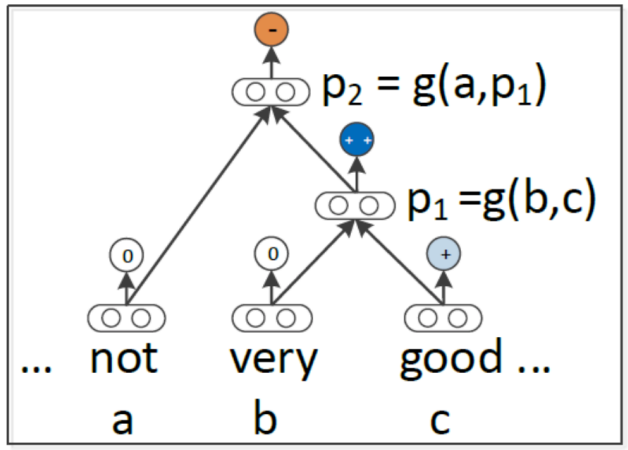



Figure 5: A single layer of the Recursive Neural Tensor Network. Each dashed box represents one of $d$-many slices and can capture a type of influence a child can have on its parent.

Assume d = 2 here




$$V\left[1:d\right] \in R^{2d\times 2d\times d}$$

Each word is represented as a d-dimensional vector.

The RNTN uses this definition for computing p :

$$p_1 = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right)$$

$$p_2 = f\left(\begin{bmatrix} a \\ p_1 \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} a \\ p_1 \end{bmatrix} + W \begin{bmatrix} a \\ p_1 \end{bmatrix}\right)$$

# Third Model: Recursive Neural Tensor Network

previous RNN $$p_1 = f\left(W \begin{bmatrix} b \\ c \end{bmatrix}\right)$$

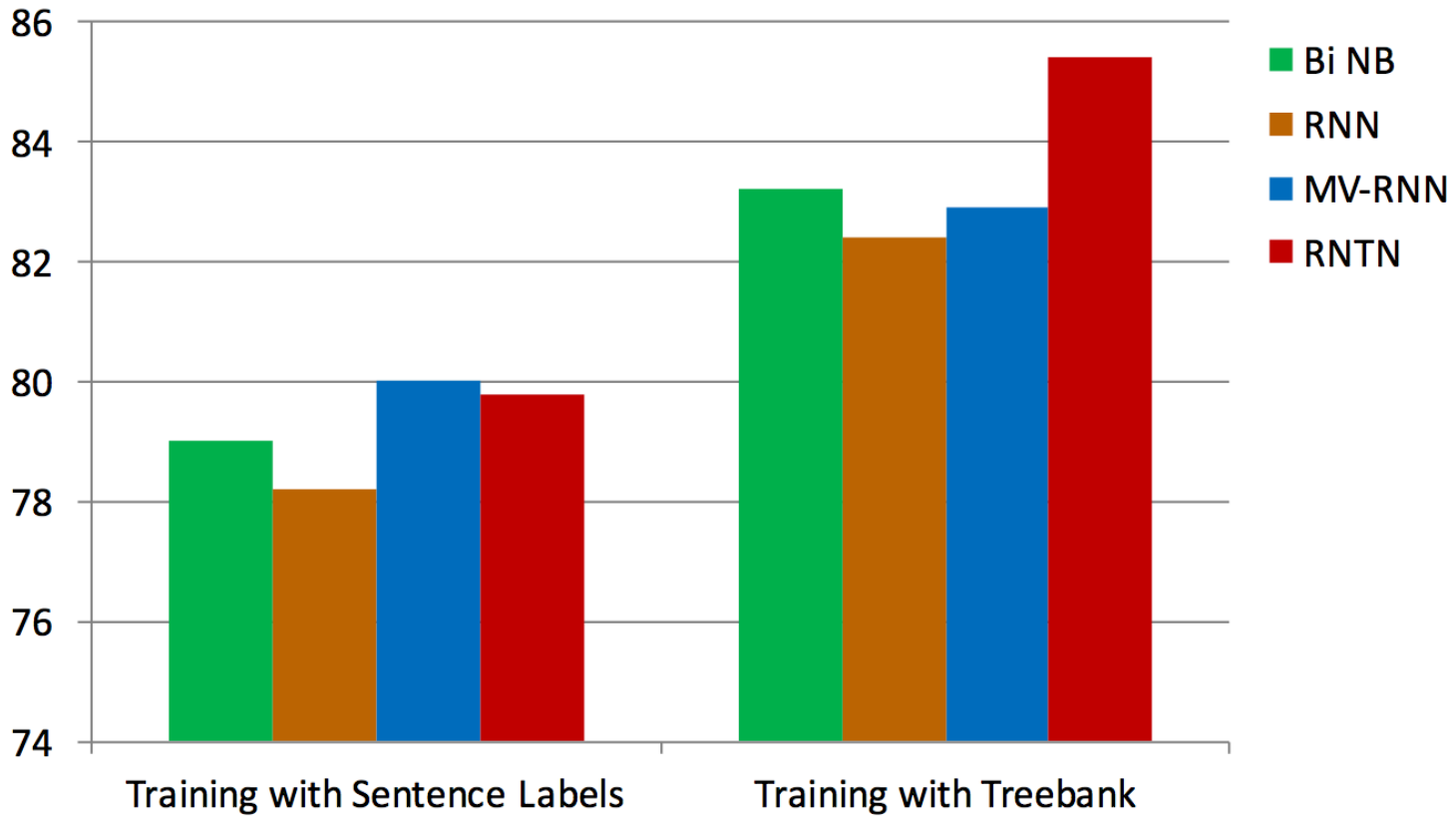
RNTN $$p_1 = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right)$$

previous RNN model is a special case of the RNTN when V is set to 0

# 5 Experiments

# Positive/Negative Results on Treebank

Classifying Sentences: Accuracy improves to 85.4



# Fine grained and binary results

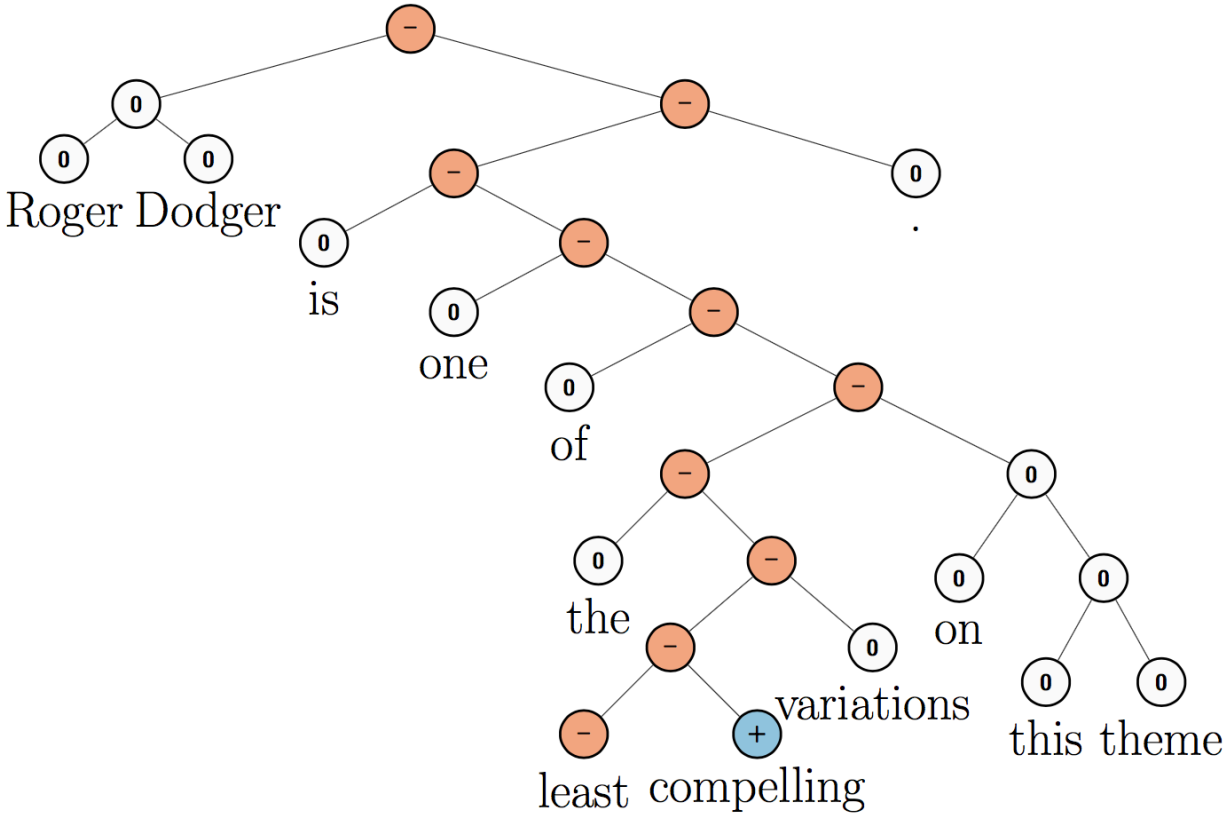
| Model | Fine-grained | | Positive/Negative | |
|---|---|---|---|---|
| | All | Root | All | Root |
| NB | 67.2 | 41.0 | 82.6 | 81.8 |
| SVM | 64.3 | 40.7 | 84.6 | 79.4 |
| BiNB | 71.0 | 41.9 | 82.7 | 83.1 |
| VecAvg | 73.3 | 32.7 | 85.1 | 80.1 |
| RNN | 79.0 | 43.2 | 86.1 | 82.4 |
| MV-RNN | 78.7 | 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.

# Negating Positive Sentences

Some sentences contains positive sentences and their negation.

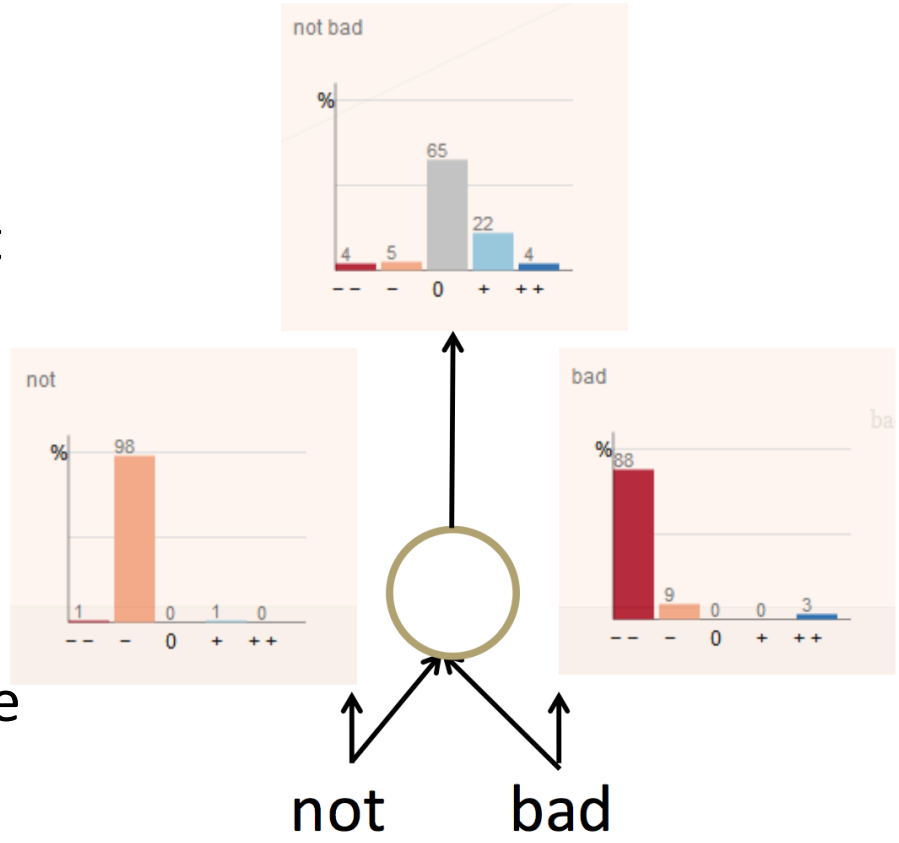
The negation changes the overall sentiment of a sentence from positive to negative.

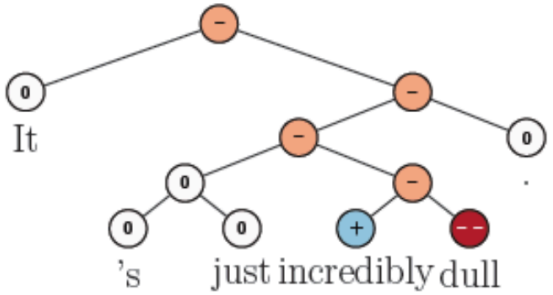


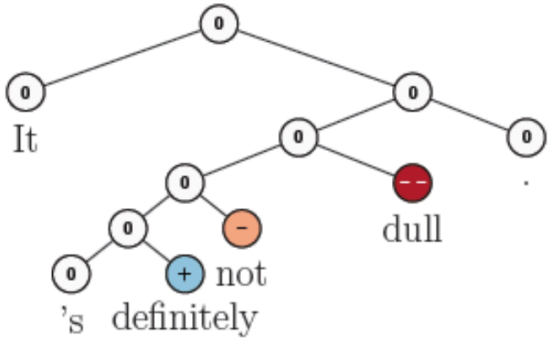
# Negating Negative Sentences.

When negative sentences are negated, the overall sentiment should become less negative, but not necessarily positive.

For instance, ‘The movie was terrible’ is negative but the ‘The movie was not terrible’ says only that it was less bad than a terrible one, not that it was good.



# Accuracy of negation detection

| 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** |


THANK YOU!


Questions