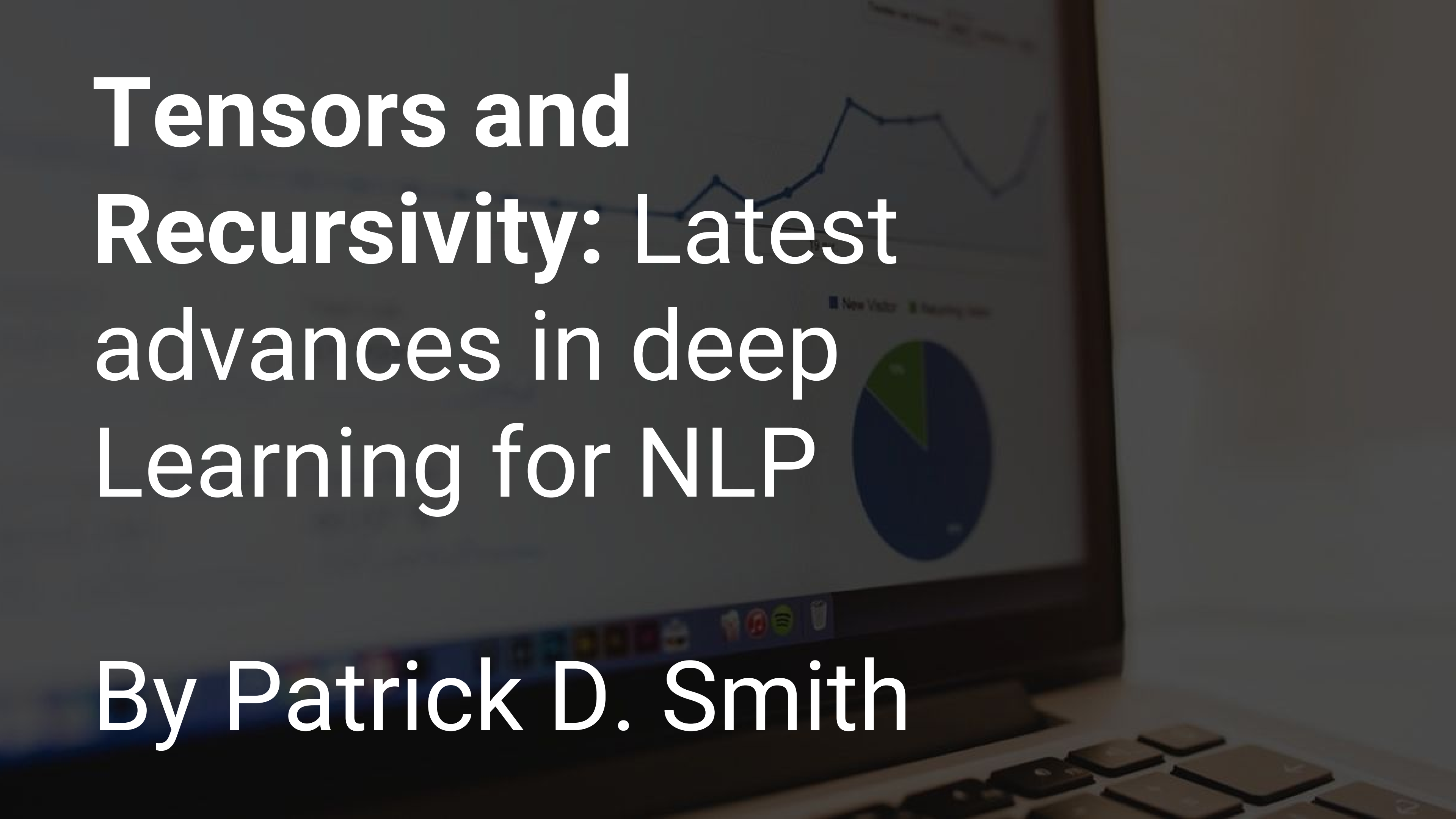


Tensors and Recursivity: Latest advances in deep Learning for NLP

By Patrick D. Smith

Part I: Theory

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

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

Part II: Code

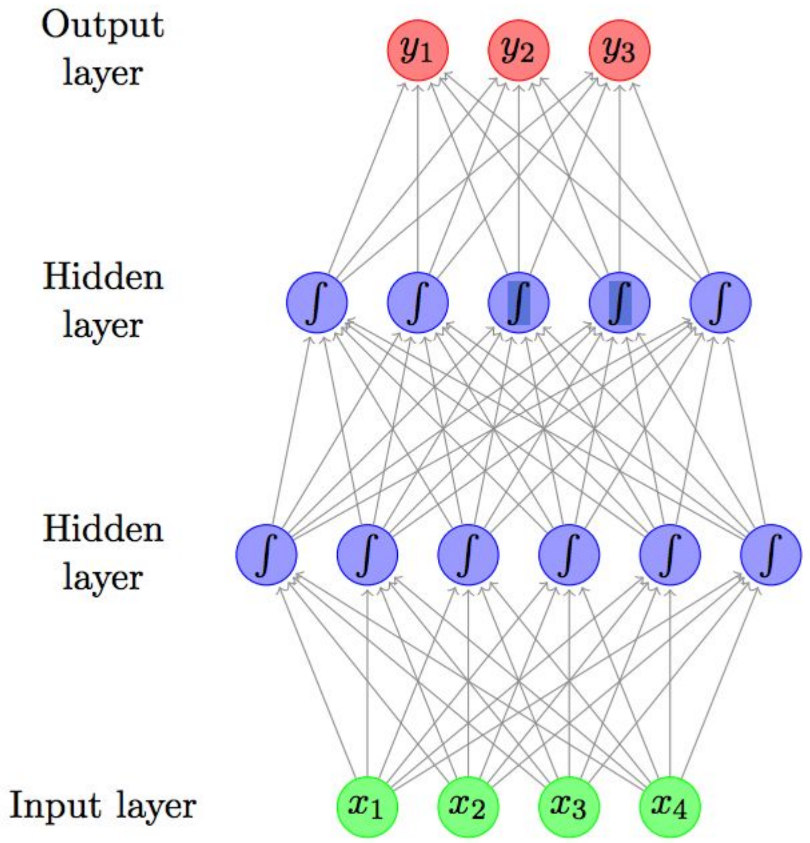
Overview

Outline

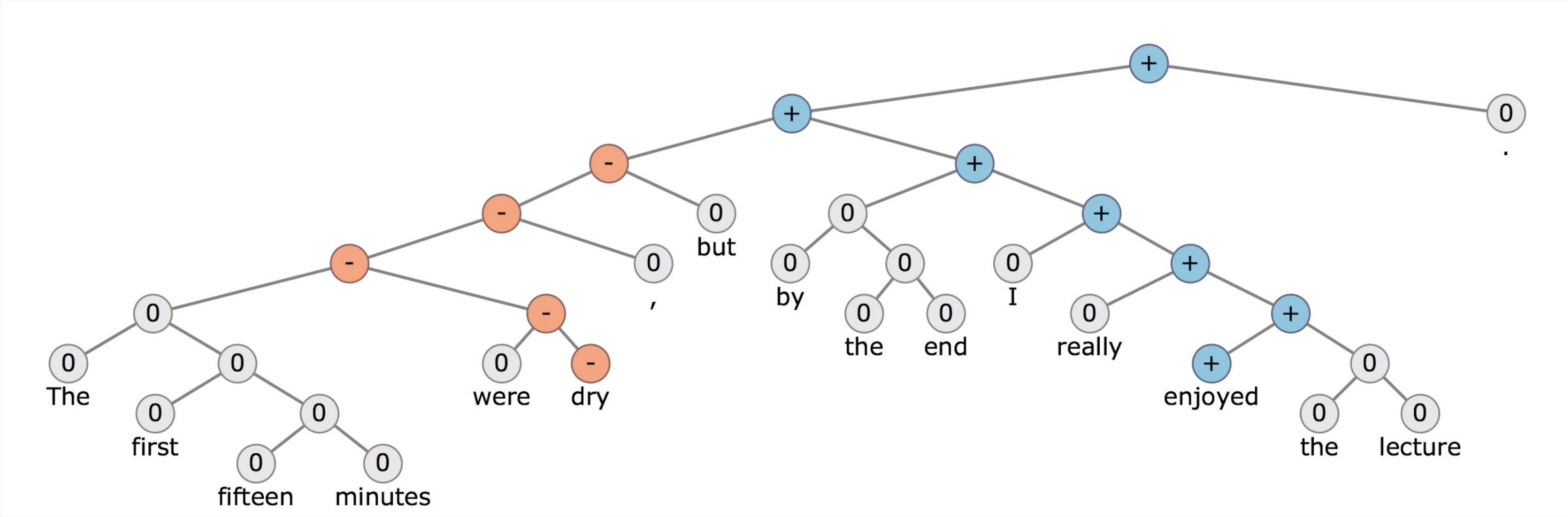
Deep Learning for NLP	● Why use deep learning for natural language processing?
Recursive Deep Learning	● The basis for RNTN's - Recursive Neural Networks
Putting it All Together: RNTN's	● Recurrent Neural Tensor Networks and their applications
Implementing RNTN's	● Coding RNTN's in Python

Deep Learning Architecture Overview

1. Input Layer
2. Hidden Layer
3. Output Layer



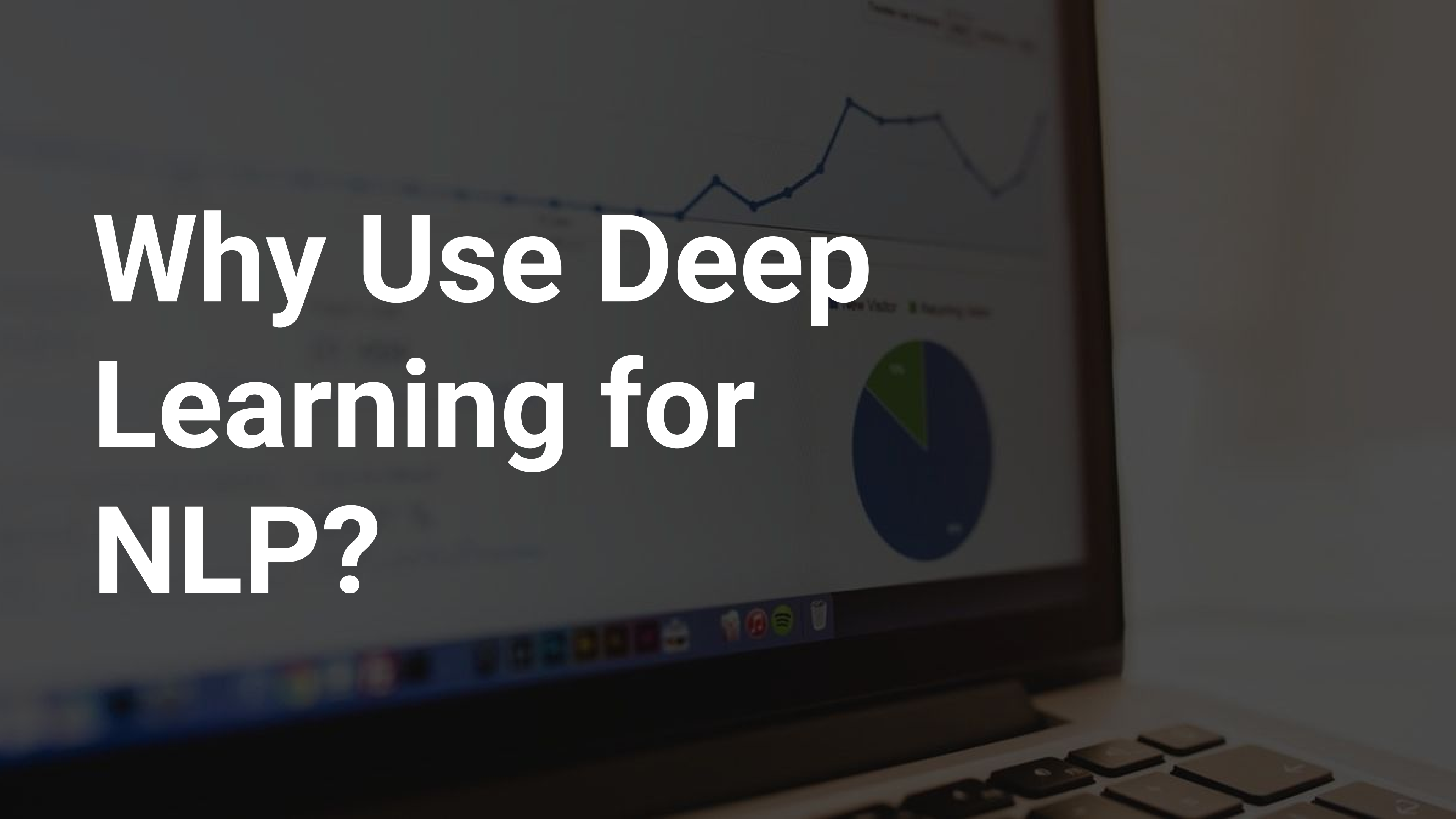
Deep Learning For Natural Language Processing



What is an RNTN?

A Recursive Neural Tensor Network (RNTN) is a new form of network that can be used to understand hierarchical data.

- RNTN's are effectively a new type of network layer based on the idea that there are multiple multiplicative effects when two word vectors are combined.
  - To date, these networks have outperformed all previous forms of networks in sentiment analysis tasks.
- In a nutshell, an RNTN is a new form of composition function that is recursively applied across hierarchically structured data, such as a parse tree



Why Use Deep Learning for NLP?

Understanding the problem

Words as “One-Hot” Vectors

The majority of traditional, rule based natural language processing procedures represent words was “One-Hot” encoded vectors

Lack of Lexical Semantics

A lot value in NLP comes from understanding a word in relation to its neighbors and their syntactical relationship

Problems with Bag of Words

Bag of Words models, including TF-IDF models, cannot distinguish certain contexts

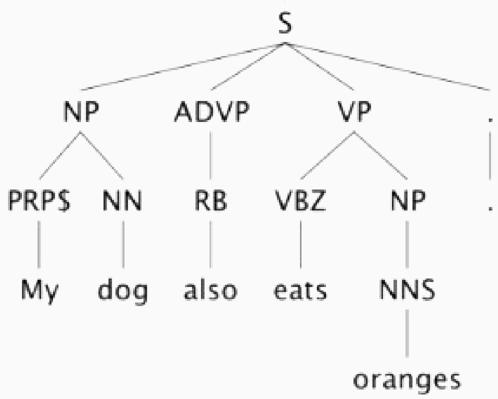
Problems with One Hot Encoding

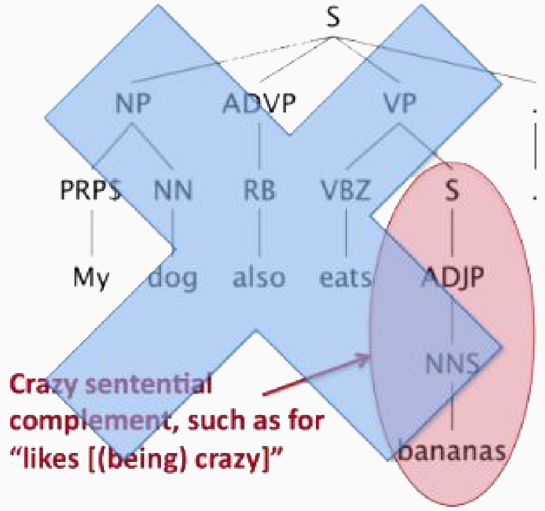
- Two different words will have no interaction between them
- “One Hot” will compute enormously long vectors for large corpi

House [0 1 0 0 0 0 0 0 0 0 0 0 0 0]
Abode [0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Lack of Lexical Semantics

Traditional models largely focus on syntactic level representations instead of semantic level representations




S
NP
ADVP
VP
.
PRP$
NN
RB
VBZ
S
.
My
dog
also
eats
ADJP
NNS
bananas
Crazy sentential complement, such as for "likes [(being) crazy]"

Socher and Manning, 2013. 6

Problems with Bag of Words Models

- Sentiment analysis can be easy for longer corpi
- However, for dataset of single sentence movie reviews (Pang and Lee, 2005) accuracy never reached above 80% for >7 years

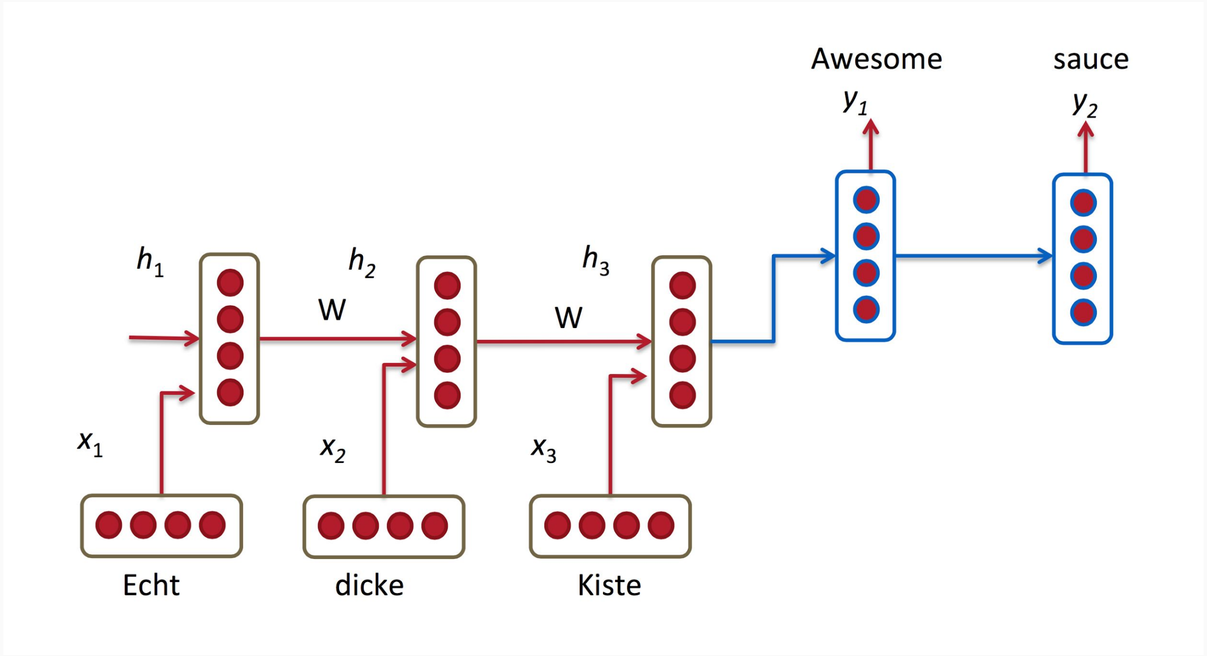
+ white blood cells destroying an infection
- an infection destroying white blood cells

Socher and Manning, 2013.

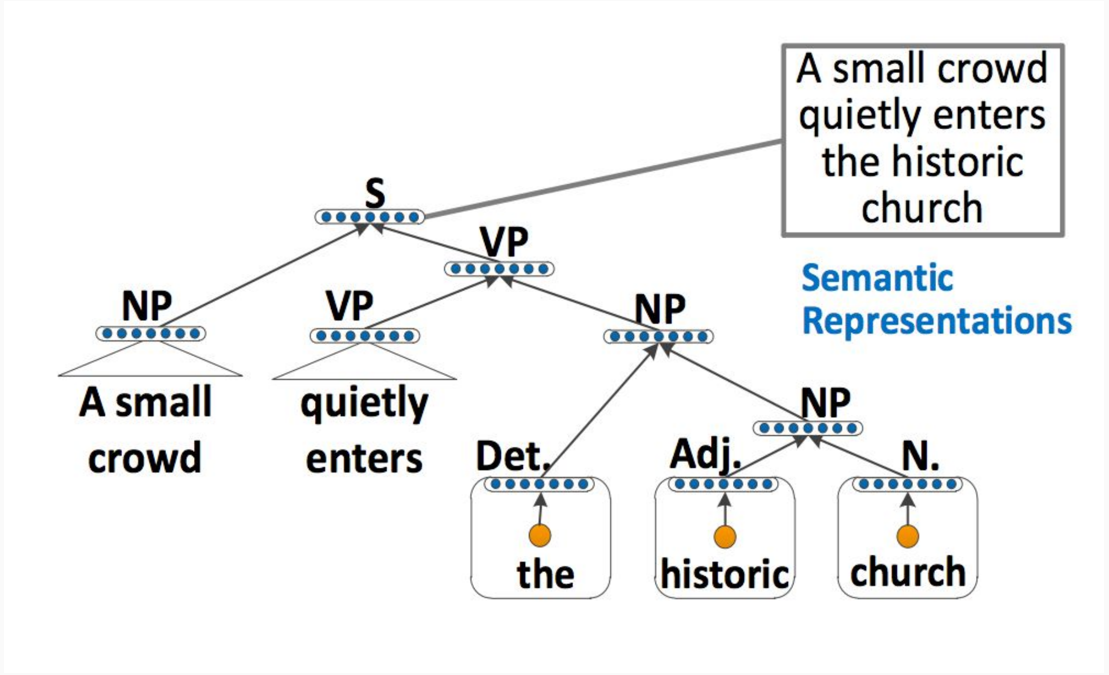
Understanding the Problem

- Many works in sentiment analysis have focused on single categories such as negative or positive, or ratings scales
- Performance of sentiment analysis models have leveled off due to the lack of understanding of negation and, in effect, the structure of natural language
  - For instance: The bag-of-words vectors for “I love this car”, “I don’t love this car”, and “Don’t you love this car” are similar.
- Natural language is best represented through a ***parse tree***
  - ***This is the basis of deep learning for NLP and the use of Recursive Neural Networks***

Capturing the Recursive Compositionality of Natural Language



Capturing the Recursive Compositionality of Natural Language



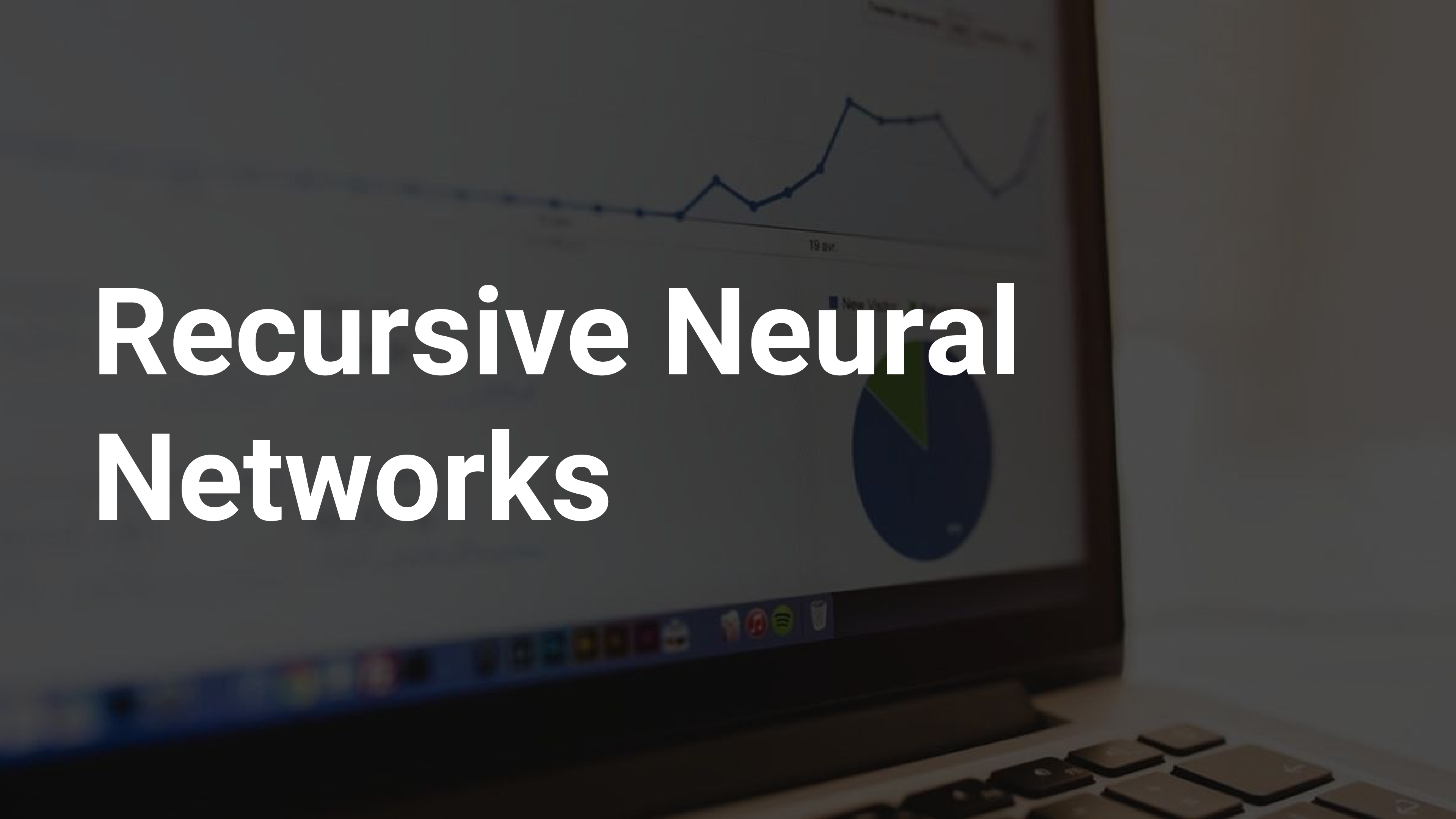
Understanding the Solution

#1. Learning Representations	● Distributed representations can help us minimize error
#2. The need for distributed representations	● Utilizing vectors of real numbers to represent words
#3. Unsupervised Feature + Weight Learning	● Most data in the world is unlabeled, therefore we need to utilize unsupervised methods
#4. Learning multiple levels of Representation	● Build multiple levels of sophistic linguistic understanding

The Solution - Recursive Networks

Recursive Neural Networks Address Three Issues:

1. Instead of using a bag-of-words representation, RNN-based models exploit hierarchical structure and use compositional semantics to understand sentiment.
2. RNN’s can be trained both on unlabeled domain data and on supervised sentiment data and do not require any language-specific sentiment lexica, parser, etc.
3. Rather than limiting sentiment to a positive/negative scale, RNN’s predict a multidimensional distribution over several complex sentiments

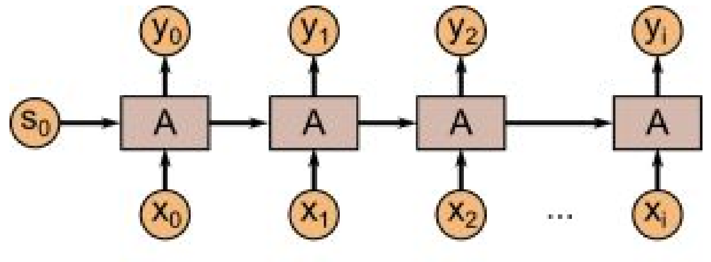


Recursive Neural Networks

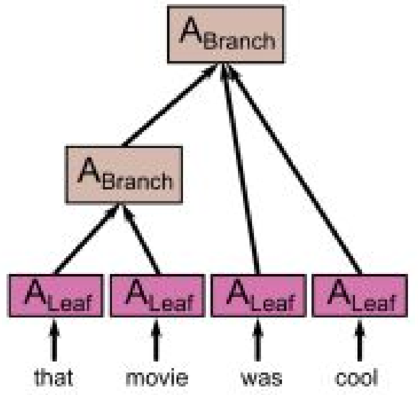
Recursive Deep Learning

Overview	● Why use deep learning for natural language processing?
Recursive Networks for Sentence Parsing	● Neural Tensor Theory, its origins, and applications
Structure	● Recurrent Neural Tensor Networks and their applications
Non-RNTN Recursive Learning	● Recursive Autoencoders & Matrix-Vector Recursive NN’s

Recurrent vs. Recursive Neural Networks



Recurrent Neural Net



Recursive Neural Net

Recursive Neural Networks

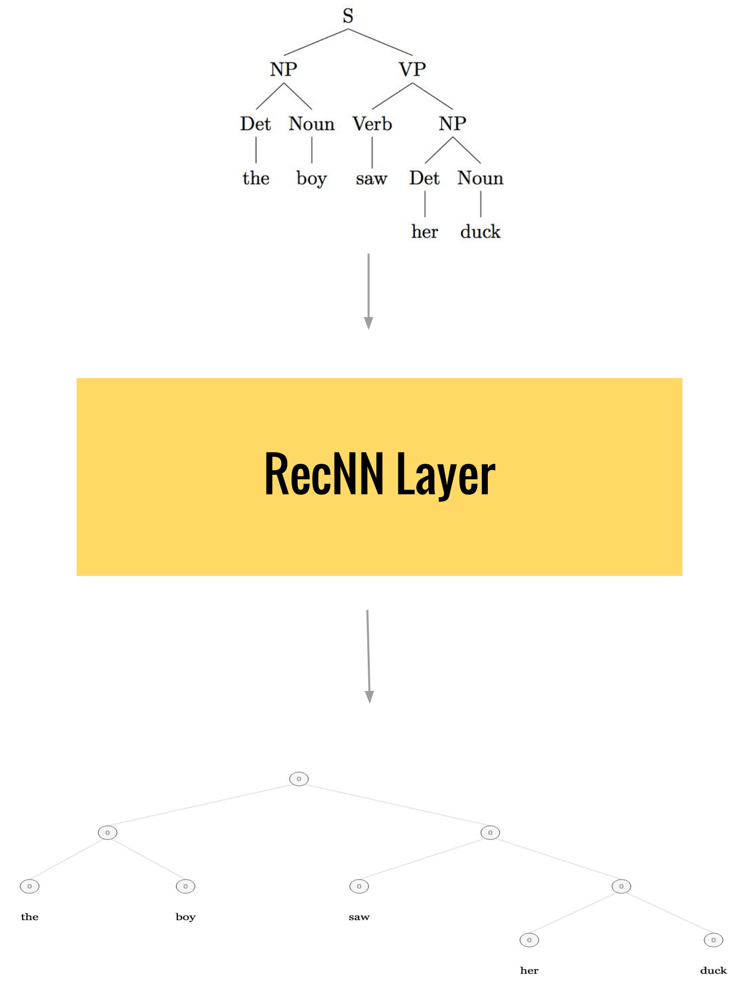
- A recursive neural network is a recurrent neural network where the unfolded network given some finite input is expressed as a (usually: binary) tree, instead of a "flat" chain (as in the recurrent network).
- Recursive Neural Networks are exceptionally useful for learning structured information
- Recursive Neural Networks are both:
    - Architecturally Complex
    - Computationally Expensive

Recursive NN's - A general structure

We can think of recursive neural networks as taking:

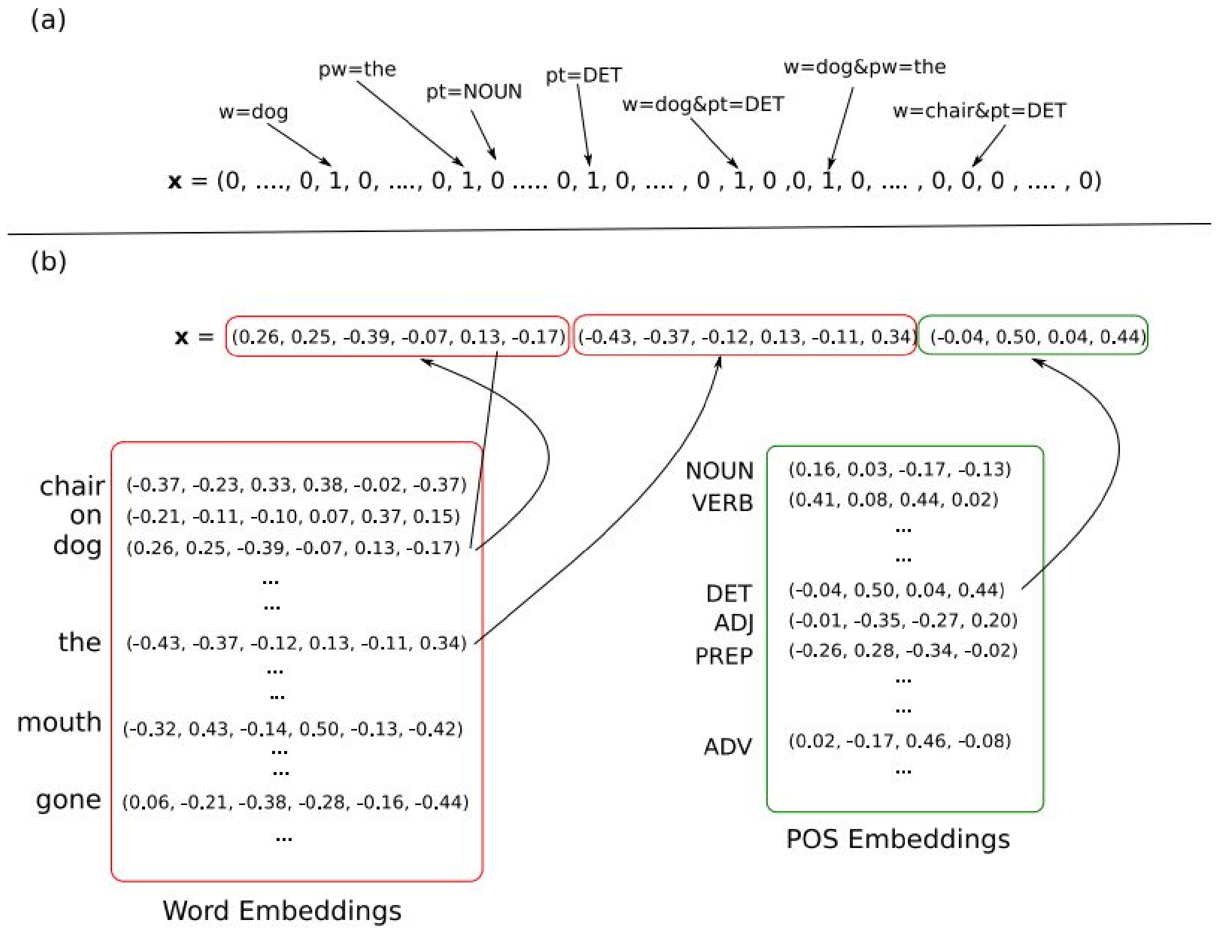
Input = Parse Tree

Output = Sentiment Labels at each tree node



Sparse Representation

Dense Representation


(a)
w=dog
pw=the
pt=NOUN
pt=DET
w=dog&pt=DET
w=dog&pw=the
w=chair&pt=DET
x = (0, ..., 0, 1, 0, ..., 0, 1, 0 ..... 0, 1, 0, ... , 0 , 1, 0 ,0, 1, 0, ... , 0, 0, 0 , ... , 0)
(b)
x = (0.26, 0.25, -0.39, -0.07, 0.13, -0.17) (-0.43, -0.37, -0.12, 0.13, -0.11, 0.34) (-0.04, 0.50, 0.04, 0.44)
chair (-0.37, -0.23, 0.33, 0.38, -0.02, -0.37)
on (-0.21, -0.11, -0.10, 0.07, 0.37, 0.15)
dog (0.26, 0.25, -0.39, -0.07, 0.13, -0.17)
the (-0.43, -0.37, -0.12, 0.13, -0.11, 0.34)
mouth (-0.32, 0.43, -0.14, 0.50, -0.13, -0.42)
gone (0.06, -0.21, -0.38, -0.28, -0.16, -0.44)
Word Embeddings
NOUN (0.16, 0.03, -0.17, -0.13)
VERB (0.41, 0.08, 0.44, 0.02)
DET (-0.04, 0.50, 0.04, 0.44)
ADJ (-0.01, -0.35, -0.27, 0.20)
PREP (-0.26, 0.28, -0.34, -0.02)
ADV (0.02, -0.17, 0.46, -0.08)
POS Embeddings

Solution: Representing Words as Dense Vectors

$$\text{Cat} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Words as Dense Vectors

Each word is represented as a d-dimensional vector randomly sampled from a uniform distribution: $U(-r, r)$, where $r = 0.0001$.

For recursive networks, all of the word vectors are stacked into a word embedding matrix $L \in R\ d \times |V|$ where $|V|$ is the size of the sentence's vocabulary.

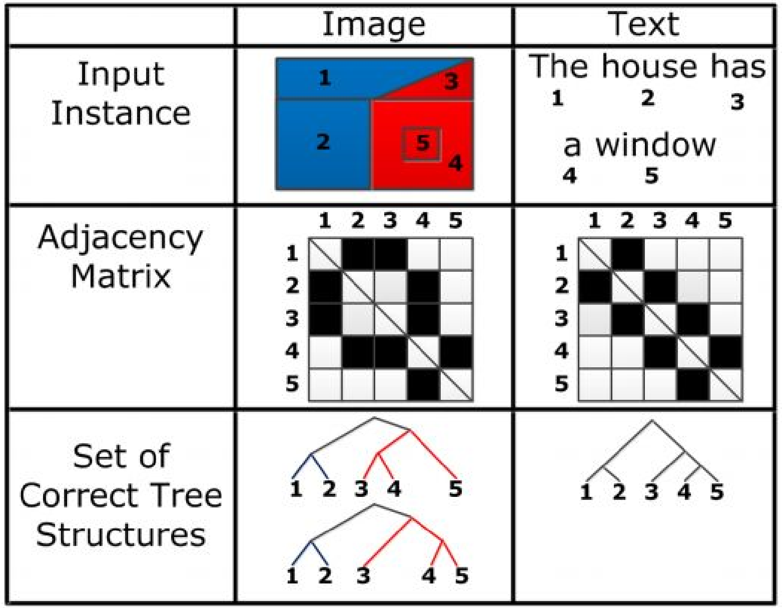
$$\text{Cat} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Words as Dense Vectors

Using semi-supervised pre-training (Word2Vec) we can begin to develop similar word embeddings for similar words.

$$\text{Cat} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

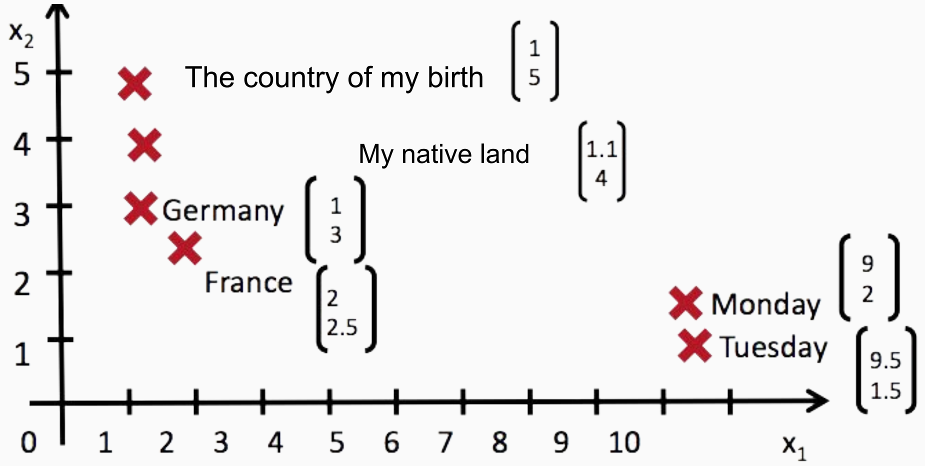
Recursive Neural Network Training Inputs



Socher, pg. 29

Recursive Neural Networks

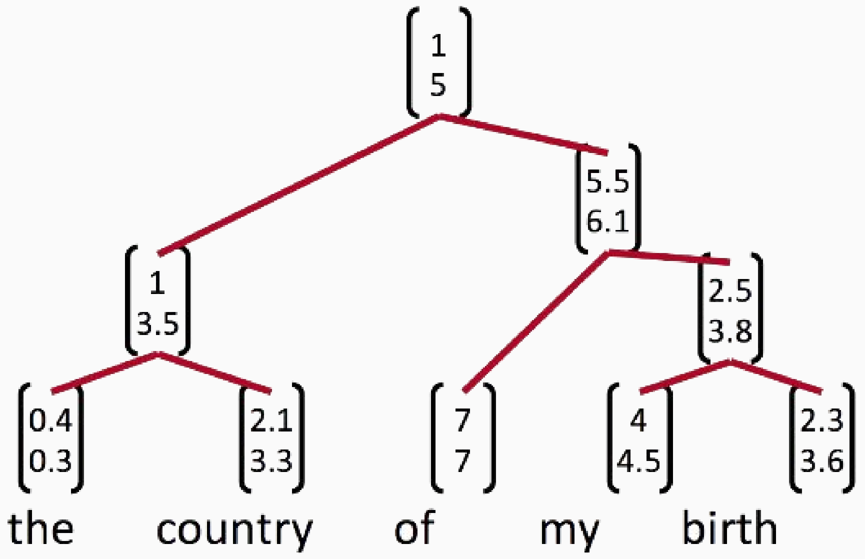
Recursive deep learning allows us to understand the meaning of longer phrases by mapping them to the same vector space



Manning, Socher, et al.

Recursive Neural Networks

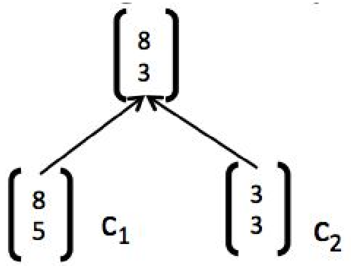
We can extend the concept of embedded vectors to sentences themselves, whereby each combination of word vectors captures some latent meaning.



Manning, Socher, et al.

Forward Propagation

- Each Parent in the tree is computed from two children
- The recursivity in the tree comes from the application of the same parameters when calculating each parent node
- Weights are shared across the entire network in a weight matrix W ∈ Rn×2n, where n is the length of your word embedding



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

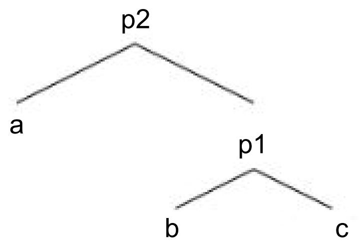
$$p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

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

Manning, Socher, et al. 110, Chen et al

Forward Propagation

The parent vectors must be of the same dimensionality to be recursively compatible and be used as input to the next composition



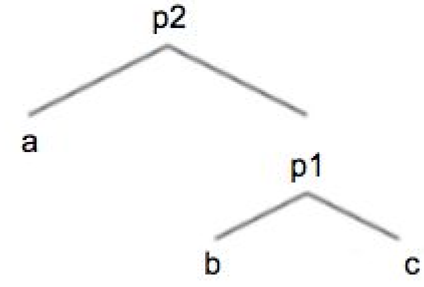
$$p_1 = f\left(W\begin{bmatrix} b \\ c \end{bmatrix}\right), p_2 = f\left(W\begin{bmatrix} a \\ p_1 \end{bmatrix}\right),$$

Manning, Socher, et al. 110, Chen et al

Forward Propagation

Each parent vector *Pi* is given to the same softmax classifier to determine its label probabilities. (IE: Negative of Positive)

Softmax will give us a nonnegative real number from a discrete probability distribution over k possible outcomes



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

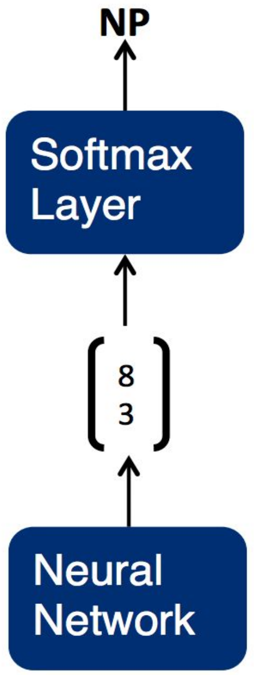
Manning, Socher, et al. 110, Chen et al

Forward Propagation

Here, Ws = sentiment classification matrix

Architecture with Softmax Classifier Included

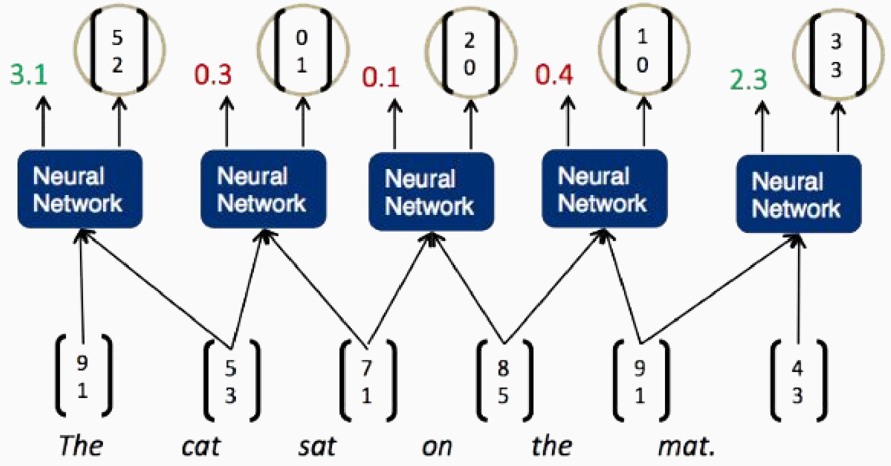
$$y^a = \mathrm{softmax}(W_s a),$$



Manning, Socher, et al. 110, Chen et al

Sentence Parsing with Recursive Neural Networks

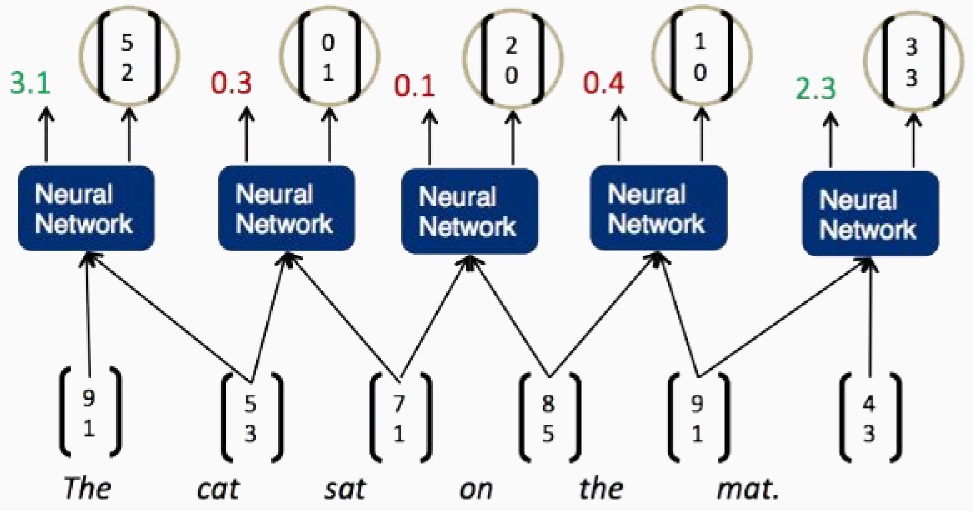
Recursive deep learning allows us to understand the meaning of longer phrases by mapping them to the same vector space



Manning, Socher, et al. 108 - 110

Sentence Parsing with Recursive Neural Networks

Each node must have a label, so we must be able to calculate these labels for any inner node, terminal node, or word vector.



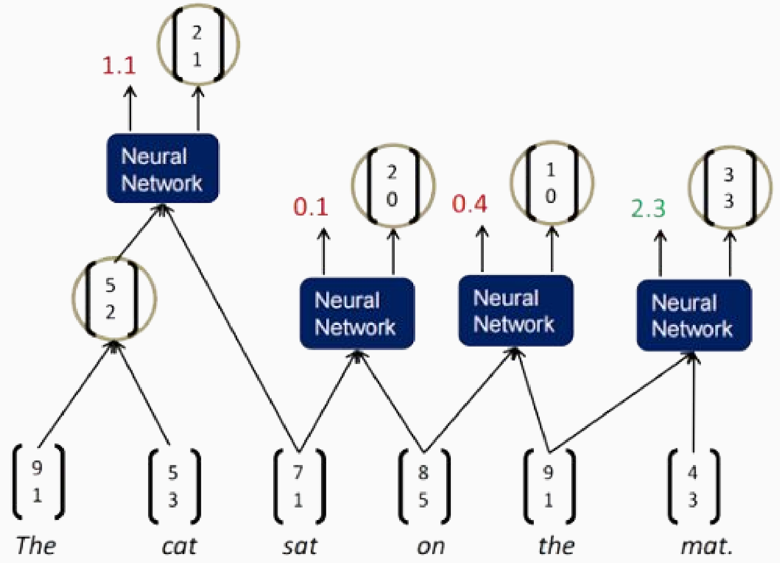
Manning, Socher, et al. 108 - 110

Sentence Parsing with Recursive Neural Networks

To do this, we:

1. Multiply the node's value by the weight
2. Add the bias
3. Run the result through a softmax classifier

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

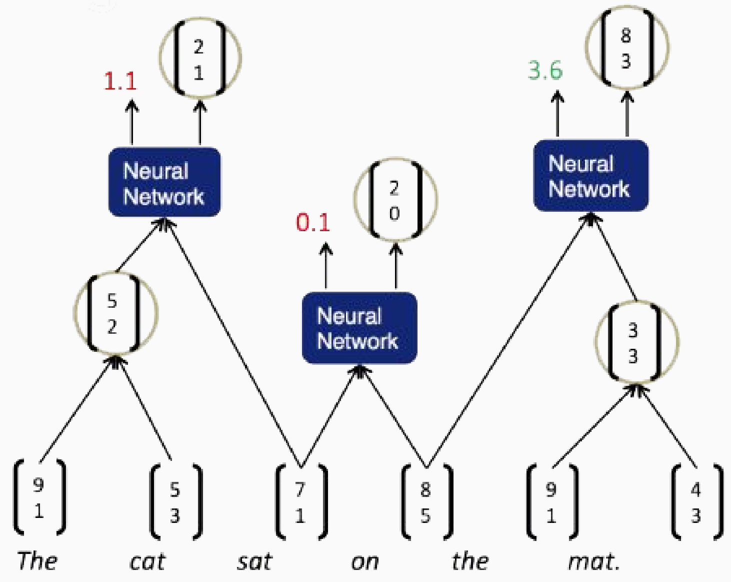


Manning, Socher, et al. 108 - 110

Sentence Parsing with Recursive Neural Networks

To do this, we:

1. Multiply the node's value by the weight
2. Add the bias
3. Run the result through a softmax classifier

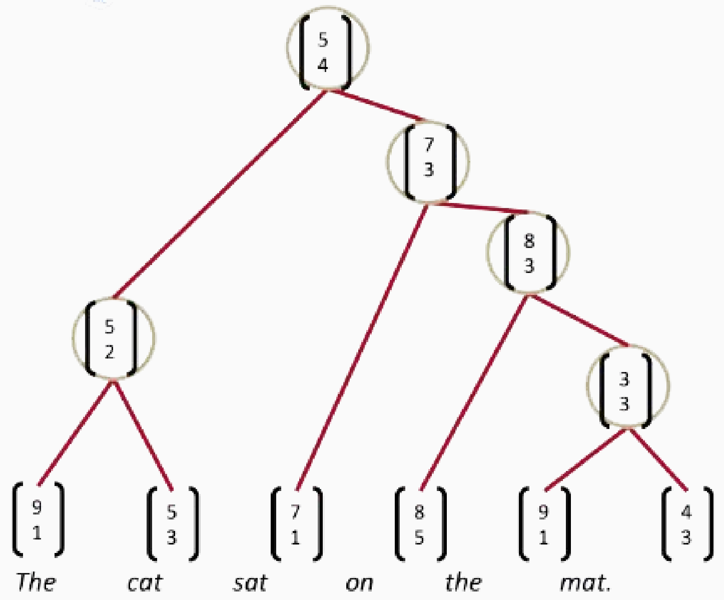


Manning, Socher, et al. 108 - 110

Sentence Parsing with Recursive Neural Networks

The score of the wholistic tree is the sum of local decisions

The loss increases when a segment merges with another one of a different label before merging with all its neighbors of the same label



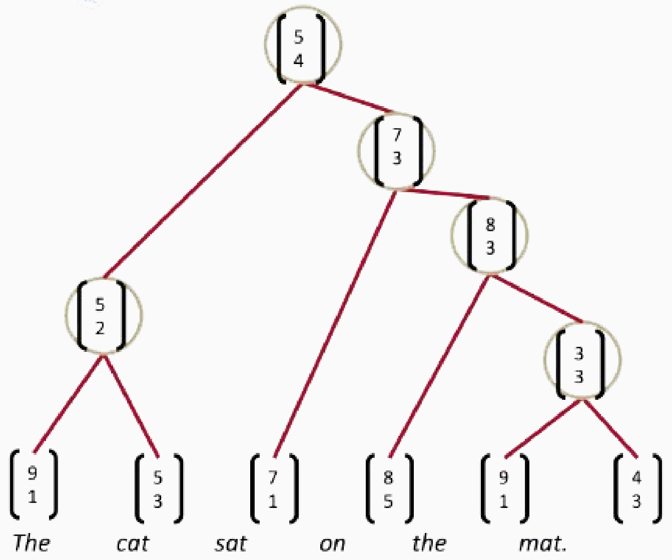
Manning, Socher, et al. 108 - 110

Sentence Parsing with Recursive Neural Networks

The score of the wholistic tree is the sum of local decisions

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

Xi = sentence
Yi = correct tree structure for the sentence

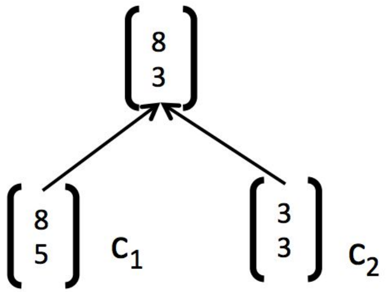


Manning, Socher, et al. 108 - 110

Backpropagation with Recursive Neural Networks

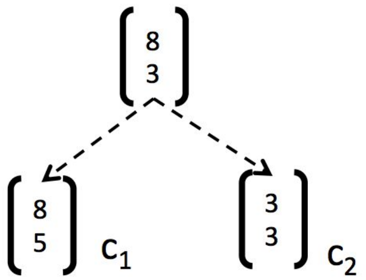
- Training Recursive Neural Networks can be done through backpropagation
- Since we have a tree structure, we need to backpropagate the error from any given parent into both the left and right child nodes
- Since we have the same W at each node, we can simply sum the derivatives of W from each node

Forward Propagation



$\begin{bmatrix}8\\3\end{bmatrix}$
$\begin{bmatrix}8\\5\end{bmatrix}$
c_1
$\begin{bmatrix}3\\3\end{bmatrix}$
c_2

Backpropagation



$\begin{bmatrix}8\\3\end{bmatrix}$
$\begin{bmatrix}8\\5\end{bmatrix}$
c_1
$\begin{bmatrix}3\\3\end{bmatrix}$
c_2

Socher, pg. 35

Backpropagation with Recursive Neural Networks

- If we assume that the W matrices at each node are different and then sum up the derivatives of these different W's, this turns out to be the same as taking derivatives with respect to the same W inside the recursion.
- This condition holds true for all RNNs

Derivative of a Small Recursive Network

$$\begin{aligned} & \frac{\partial}{\partial W} f(W(f(Wx)) \\ = \;& f'(W(f(Wx)) \left(\left(\frac{\partial}{\partial W} W \right) f(Wx) + W \frac{\partial}{\partial W} f(Wx) \right) \\ = \;& f'(W(f(Wx)) \left(f(Wx) + W f'(Wx) x \right). \end{aligned}$$

Same Network with Derivatives at Each Occurrence

$$\begin{aligned} & \frac{\partial}{\partial W_2} f(W_2(f(W_1x)) + \frac{\partial}{\partial W_1} f(W_2(f(W_1x)) \\ = \;& f'(W_2(f(W_1x)) \left(f(W_1x) \right) + f'(W_2(f(W_1x)) \left(W_2 f'(W_1x) x \right) \\ = \;& f'(W_2(f(W_1x)) \left(f(W_1x) + W_2 f'(W_1x) x \right) \\ = \;& f'(W(f(Wx)) \left(f(Wx) + W f'(Wx) x \right). \end{aligned}$$

Socher, pg. 35

RNN Optimization

- Best to use mini-batched limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) or AdaGrad for minimization.
- AdaGrad gives best results

L-BFGS

$$H_j^{-1} = \left(I - \frac{s_j y_j^T}{y_j^T s_j}\right) H_{j-1}^{-1} \left(I - \frac{y_j s_j^T}{y_j^T s_j}\right) + \frac{s_j s_j^T}{y_j^T s_j}$$

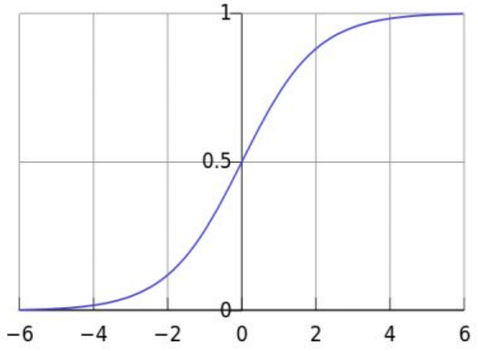
AdaGrad

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$$

Activation Function

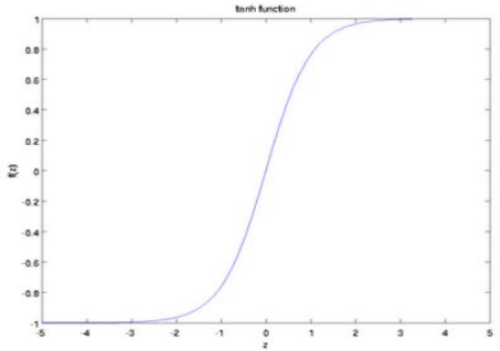
TNN’s can use any standard activation function, such as ReLu, Tanh, etc.

$$f(z) = \frac{1}{1 + \exp(-z)}.$$



$$f'(z) = f(z)(1 - f(z))$$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

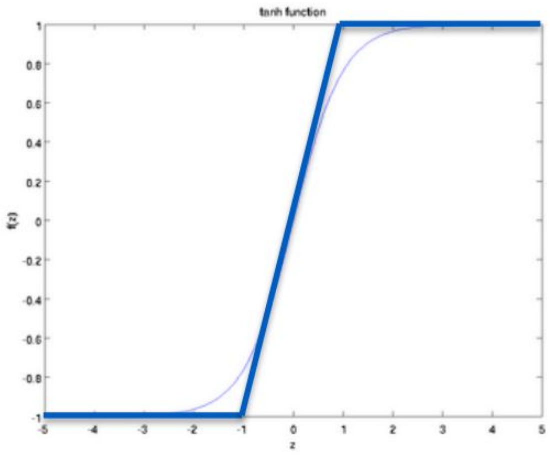


$$f'(z) = 1 - f(z)^2$$

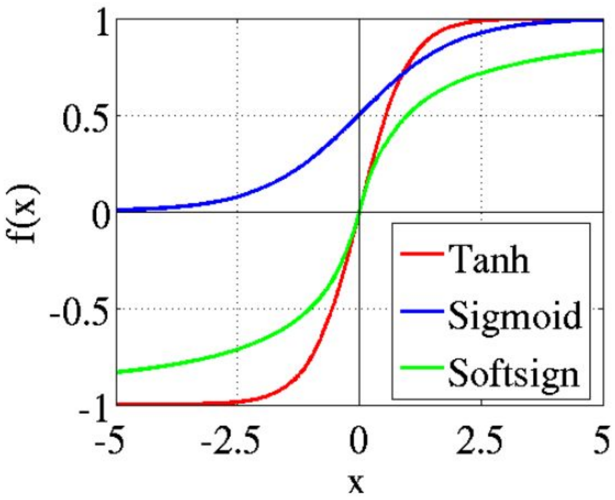
Activation Function

TNN's can use any standard activation function, such as ReLu, Tanh, etc.

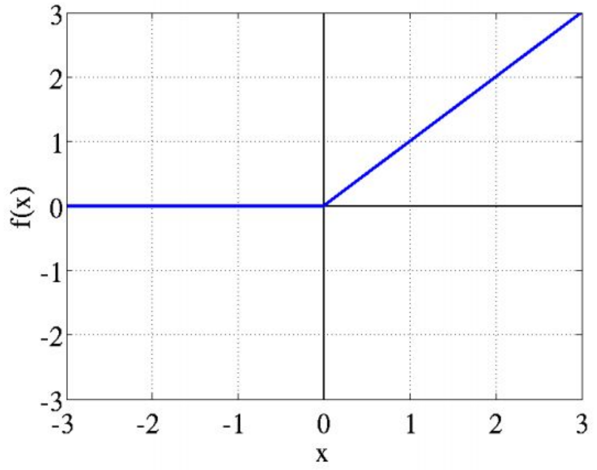
Hard Tanh



Soft Sign



ReLu



RecNN Recap

1. We must begin with a set of embedded word vectors, or activation vectors, which represent the words in a sentence. These can be pre-trained using a Word2Vec or similar model.
2. These word embeddings create a symmetric adjacency matrix A, where A(i, j) = 1, if segment i neighbors j. This matrix defines which elements can be merged. For sentences, this matrix has a special form with 1's only on the first diagonal below and above the main diagonal.
3. We recursively run the composition function over the elements of the parse tree, outputting both parent vectors and scores until we reach a root node which represents the entire sentence.

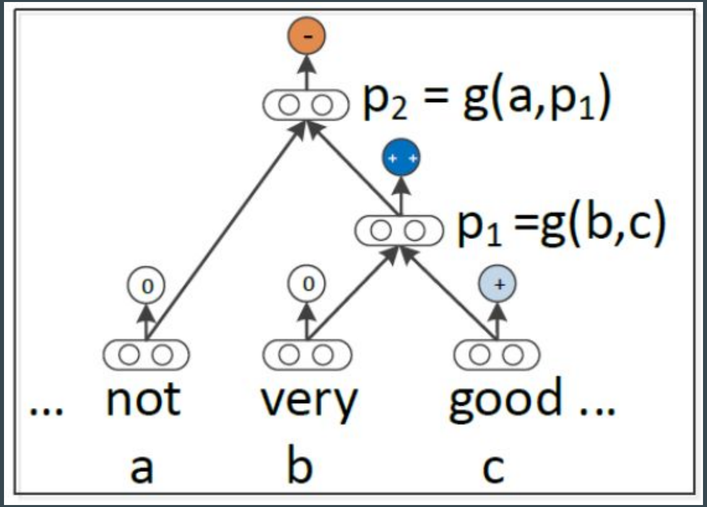
Putting it All Together: RNTN’s

Recursive Neural Tensor Networks

With Recursive Neural Tensor Networks, the fundamental idea is to allow more interaction of word vectors

Recursive Neural Tensor Networks:

- Take as input any phrase of any length
- RNTN’s represent these phrases through a parse tree and word vectors as before
- RNTN’s compute vectors for higher nodes in the tree using the same tensor-based composition function throughout.

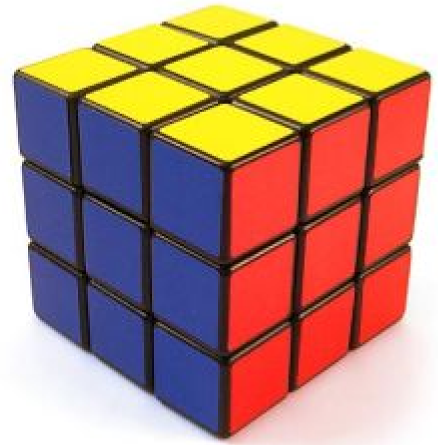


Tensors

A tensor T

- represents a multilinear transformation $U \otimes V \rightarrow W$, $W \rightarrow U \otimes V$, $U \otimes W \rightarrow V$, etc.
- is a multi-way array (here 3-way) with a multilinear action on each "leg" or "side"
- Data arrives in tensor format, and something is lost by flattening

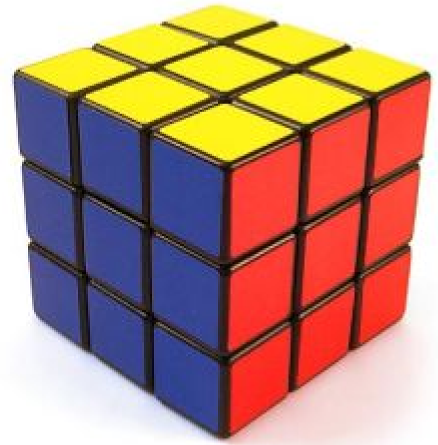
Tensors describe linear relationships between vectors and other tensors



Socher, Manning, Ng, et al. 6.

Neural Tensor Theory

- Developed as a geometric representation of brain function, later expand to the “geometrization” of natural states
- Highly researched in physics and chemistry in the last 20+ years
- Tensors allow for multi-linear operations in deep learning

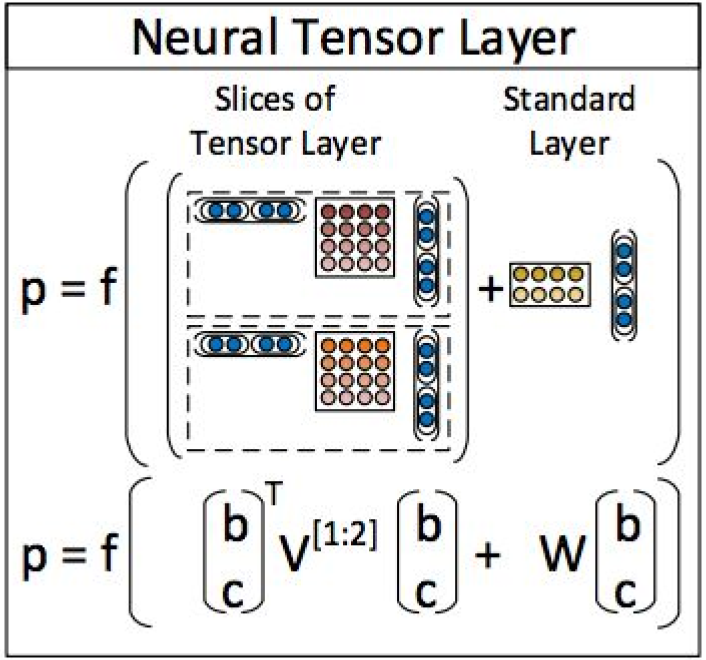


Neural Tensor Layers

- In the standard RNN, the input vectors only implicitly interact through the nonlinearity (squashing) function.

- The composition function, otherwise known as the "layer", computes vectors for longer phrases based on the words in a phrase.

- RNTN's assume that a tree structure is already provided and traverses that tree, allowing the model to focus solely on the semantic content of a sentence and the prediction task


Neural Tensor Layer
Slices of Tensor Layer
Standard Layer
p = f
+
p = f
$\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}$

Socher, Manning, Ng, et al. 6.

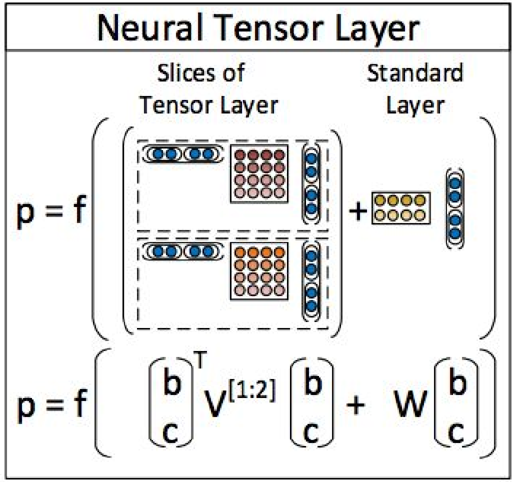
Neural Tensor Layers

- The main advantage over the previous RNN model, which is a special case of the RNTN when V is set to 0, is that the tensor can directly relate input vectors.

- One vector representation modifies the other vector representation as you combine them.

- $V^{[1:d]} \in R^{2d\times 2d\times d}$ is the tensor that defines multiple bilinear forms, where d reflects the dimensions of the children vector

Another way to interpret each tensor slice is that it mediates the relationship between the two entity vectors differently.



$$p = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right)$$

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

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

Socher, Manning, Ng, et al. 6.

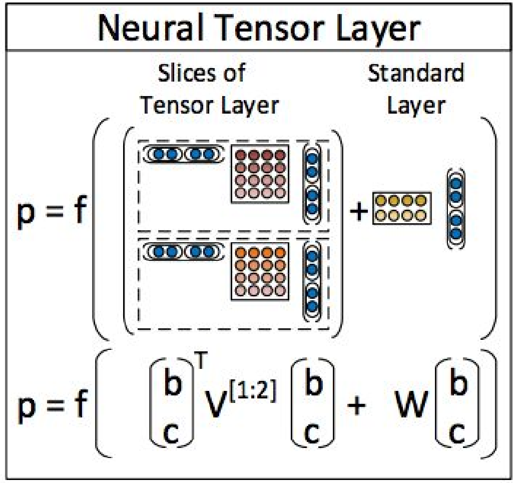
Neural Tensor Layers

- Each tensor slice captures one way that the two vectors could interact with each other

We then add the product of the Weight matrix and the concatenated vector as with a standard RecNN.

Parameters

W_s = sentiment classification matrix
W = Weighting parameter
L = Word embedding matrix
V = Bilinear Tensor


Neural Tensor Layer
Slices of Tensor Layer
Standard Layer
p = f
+
p = f
$\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}$

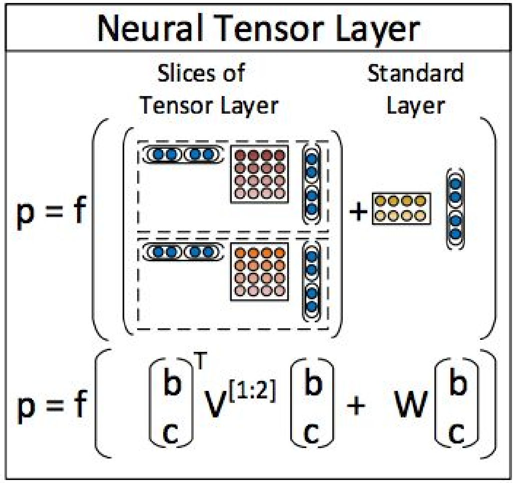
$$p_1 = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right),$$

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

Socher, 121

Backprop with RNTN's

- With RNTN Backprop, we want to minimize the cross entropy error
- Each node backpropagates its error through to the recursively used weights V, W
  - The full derivative for V and W is the sum of the derivatives at each of the nodes
- Utilizing AdaGrad, converges to a local optimum on the Stanford Treebank in 3 hours.



Neural Tensor Layer
Slices of Tensor Layer
Standard Layer
p = f
+
p = f
$\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}$

RNTN Error as a function of its parameters (theta = V, W, W_s, L)

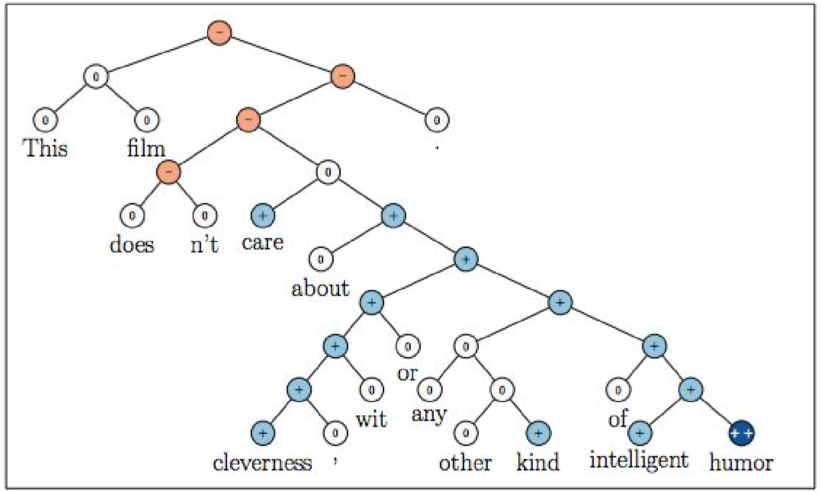
$$E(\theta) = \sum_i \sum_j t_j^i \log y_j^i + \lambda \|\theta\|^2$$

Socher, Manning, Ng, et al. 6.

RNTN

- In their current architecture, RNTN’s use a single, composition function (layer).
- Studies testing the effect of additional layers found that the networks are hard to optimize

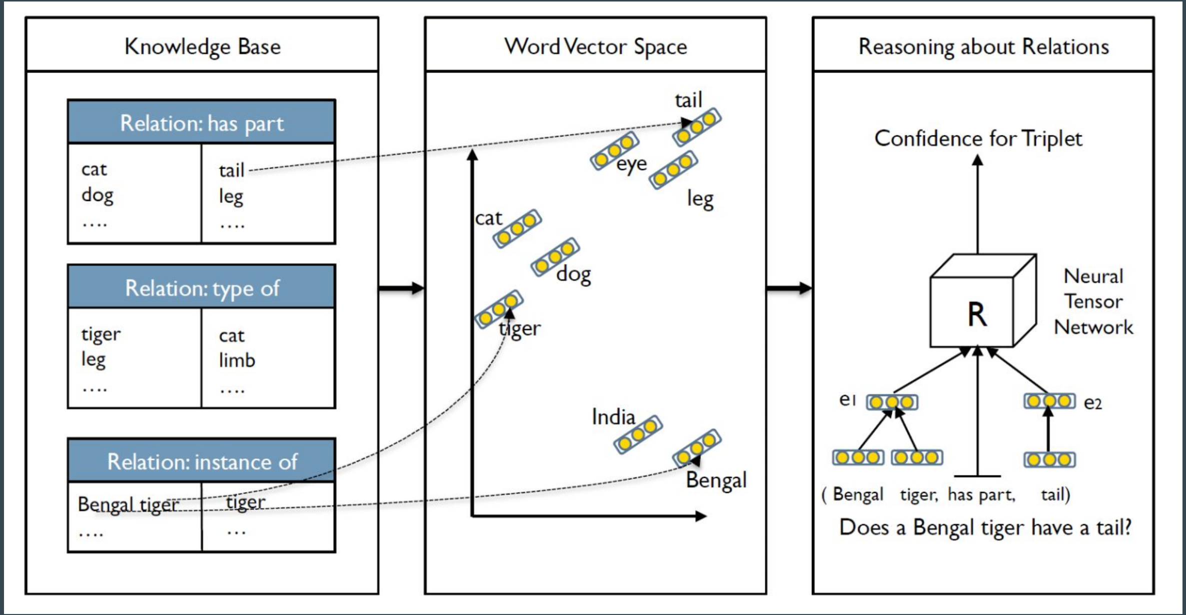
Example of the Recursive Neural Tensor Network accurately predicting 5 sentiment classes



Neural Tensor Networks in Knowledge Bases

Higher scores for each triplet indicate that the words are more likely to be in a relationship

$$g(e_1, R, e_2) = u_R^T f\left(e_1^T W_R^{[1:k]} e_2 + V_R \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b_R\right),$$

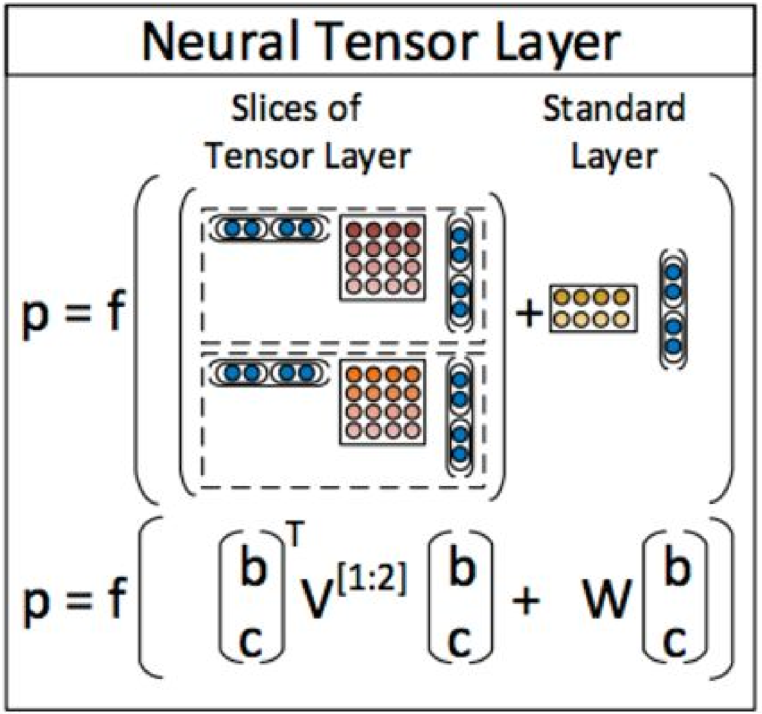


Coding RNTN’s in Python

Wrapping it all up

Wrapping Up

- By using deep learning for NLP, we will continue to learn more about feature representations and higher levels of abstraction
- Intuitively, this should allow for easier generalization and understanding of cross-contextual language



$$p = f\left(\begin{pmatrix} b \\ c \end{pmatrix}^T V^{[1:2]} \begin{pmatrix} b \\ c \end{pmatrix} + W \begin{pmatrix} b \\ c \end{pmatrix} \right)$$

Sources

Deep Recursive Networks for Compositionality in Language. Ozan Irsoy and Claire Cardie.
https://www.cs.cornell.edu/~oirsoy/files/nips14drsv.pdf

Recursive Deep Learning for Natural Language Processing and Computer Vision. Richard Socher, 2014.
http://nlp.stanford.edu/~socherr/thesis.pdf

Deep Learning for NLP. Richard Socher and Christopher Manning, 2013.http://nlp.stanford.edu/courses/NAACL2013/NAACL2013-Socher-Manning-DeepLearning.pdf

Reasoning with Neural Tensor Networks for Knowledge Base Completion. Richard Scher, Danqui Chen, Christopher Manning, Andrew Ng.
https://papers.nips.cc/paper/5028-reasoning-with-neural-tensor-networks-for-knowledge-base-completion.pdf