

# Deep Learning for NLP

(without Magic)



**Richard Socher and Christopher Manning**

**Stanford University**

NAACL 2013, Atlanta

<http://nlp.stanford.edu/courses/NAACL2013/>

\*with a big thank you to Yoshua Bengio, with whom we participated in the previous ACL 2012 version of this tutorial

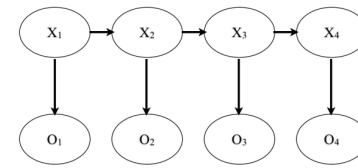
# Deep Learning

Most current machine learning works well because of human-designed representations and input features

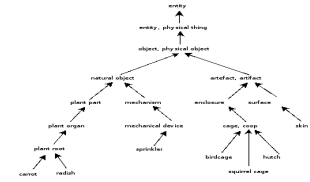
Machine learning becomes just optimizing weights to best make a final prediction

Representation learning attempts to automatically learn good features or representations

Deep learning algorithms attempt to learn multiple levels of representation of increasing complexity/abstraction



NER

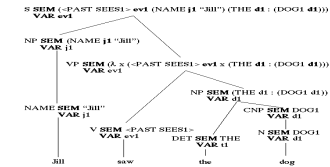


WordNet

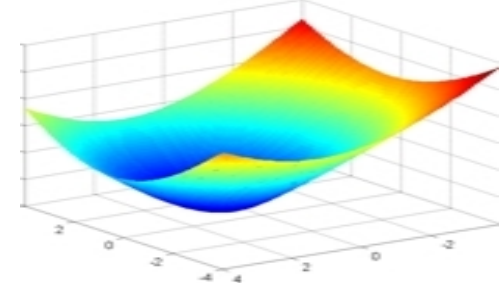
```

<DOCID> ws94_008_0212 </DOCID>
<DOCNO> 940415-0062 </DOCNO>
<RT> Who's News
@
<DATE TIME> 1994 FEB 12 </DATE TIME>
<FILE NAME> 19940212.FTD
<EO> WALL STREET JOURNAL (J), PAGE B10 </EO>
<CO> MER </CO>
<TR> SECURITIES (SCR) </TR>
<TEXT>
[
  <P>
    <STRONG> <STRONG> [ <STRONG> <STRONG> -- Donald Wright, 46 years old, was named executive vice president and director of fixed income at this brokerage firm. Mr. Wright was named as president of Merrill Lynch, Pierce, Fenner & Smith, a unit of Merrill Lynch, Pierce, Fenner & Smith, who is expected to begin his new position by the end of the month.
  ]
]
</TEXT>
</DOC>
  
```

SRL



Parser



# A Deep Architecture

Mainly, work has explored [deep belief networks \(DBNs\)](#), Markov Random Fields with multiple layers, and various types of multiple-layer neural networks

Output layer

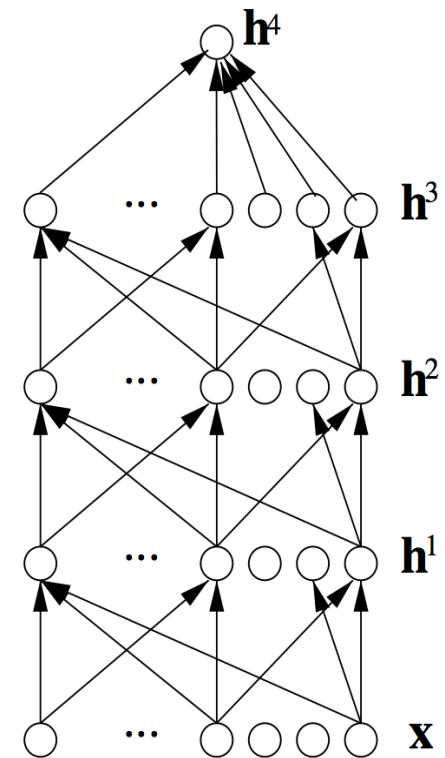
Here predicting a supervised target

Hidden layers

These learn more abstract representations as you head up

Input layer

3 Raw sensory inputs (roughly)



Part 1.1: The Basics

# Five Reasons to Explore Deep Learning



# #1 Learning representations

Handcrafting features is time-consuming

The features are often both over-specified and incomplete

The work has to be done again for each task/domain/...

We must move beyond handcrafted features and simple ML

Humans develop representations for learning and reasoning

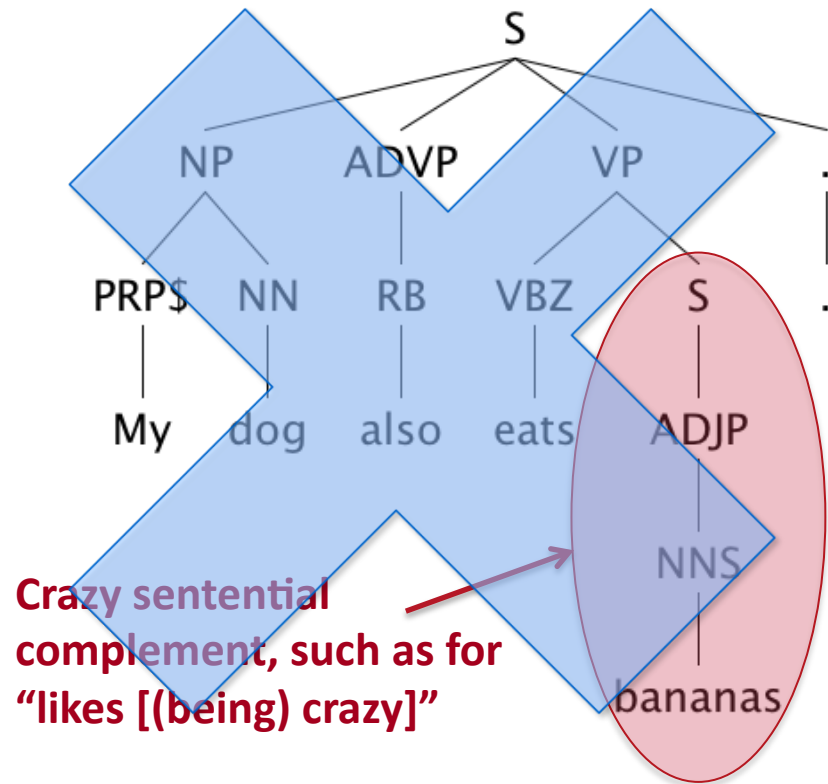
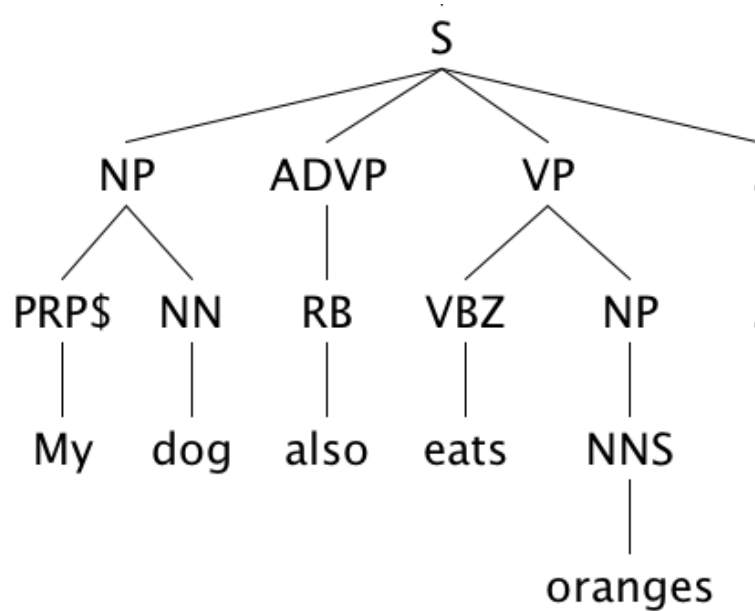
Our computers should do the same

Deep learning provides a way of doing this



# #2 The need for distributed representations

Current NLP systems are incredibly fragile because of their atomic symbol representations



# #2 The need for distributional & distributed representations

Learned word representations help enormously in NLP

They provide a powerful similarity model for words

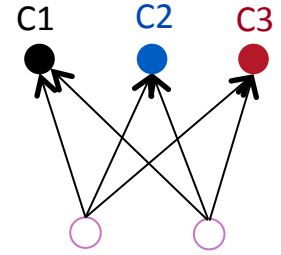
**Distributional similarity** based word clusters greatly help most applications

+1.4% F1 Dependency Parsing **15.2% error reduction** (Koo & Collins 2008, Brown clustering)

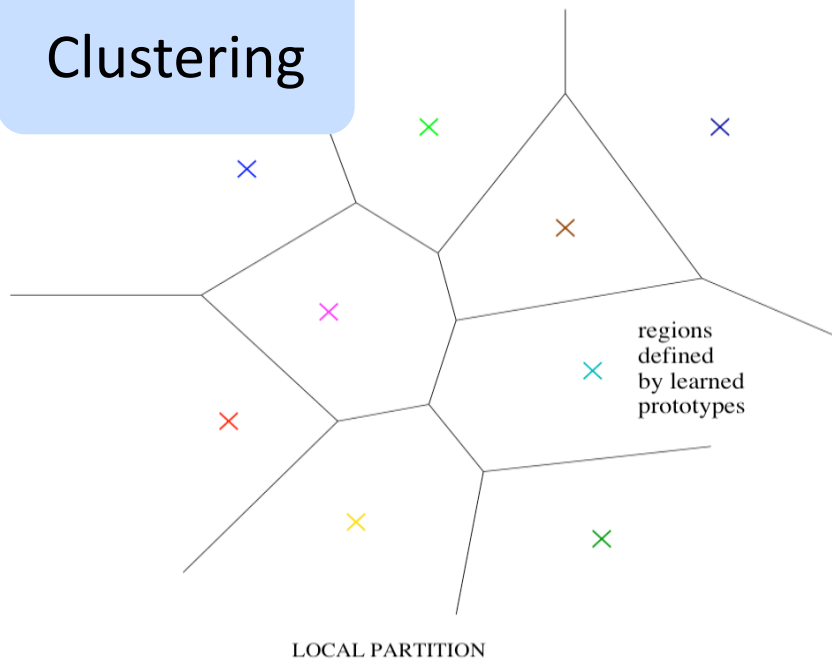
+3.4% F1 Named Entity Recognition **23.7% error reduction** (Stanford NER, exchange clustering)

**Distributed representations** can do even better by representing more dimensions of similarity

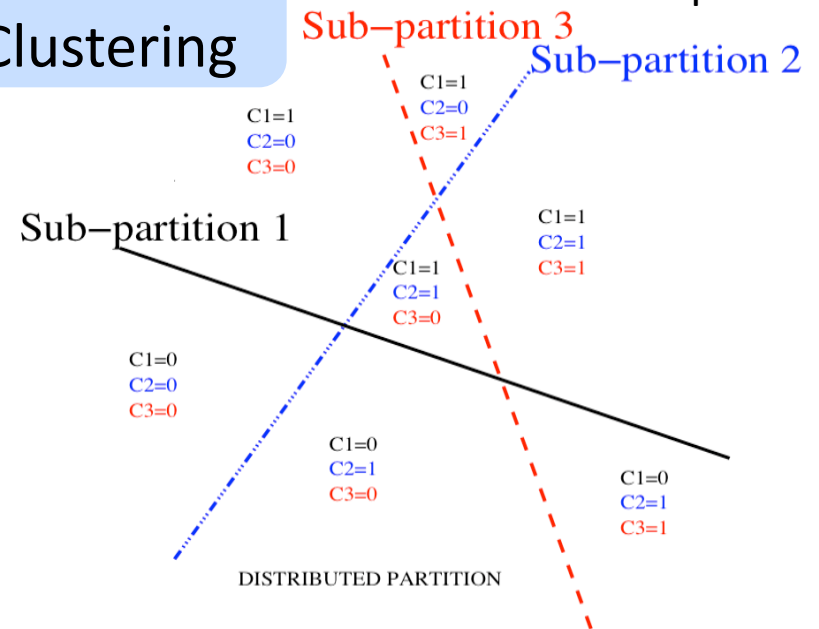
# #2 The need for distributed representations



Clustering



Multi-Clustering



Learning features that are not mutually exclusive can be **exponentially more efficient** than nearest-neighbor-like or clustering-like models

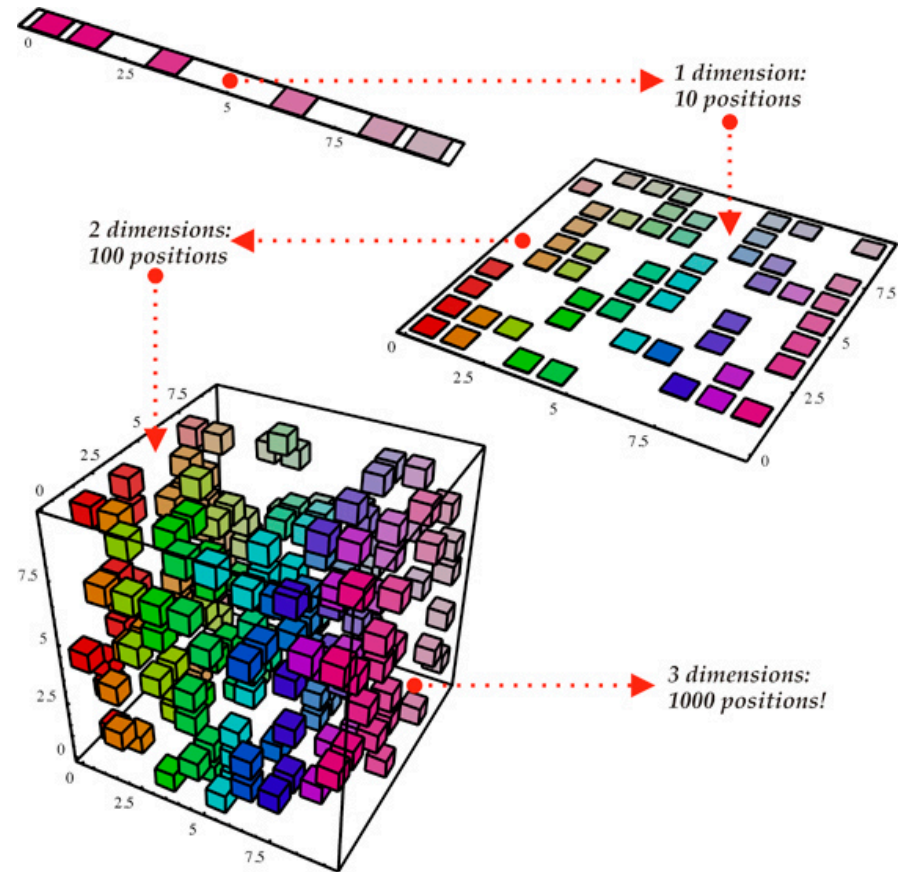
# Distributed representations deal with the curse of dimensionality

Generalizing locally (e.g., nearest neighbors) requires representative examples for all relevant variations!

## Classic solutions:

- Manual feature design
- Assuming a smooth target function (e.g., linear models)
- Kernel methods (linear in terms of kernel based on data points)

Neural networks parameterize and learn a “similarity” kernel



# #3 Unsupervised feature and weight learning

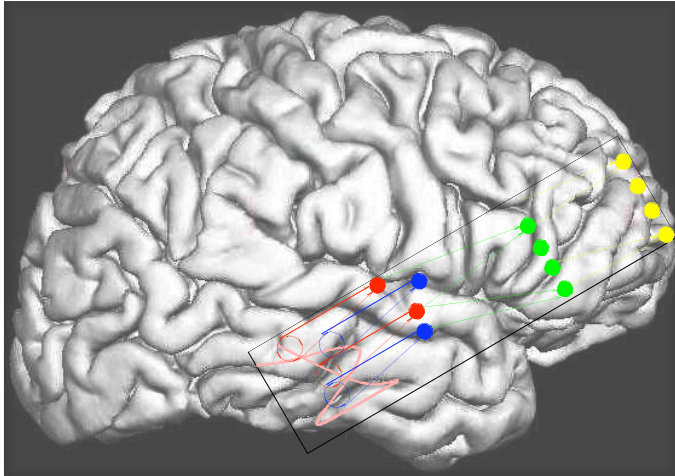
Today, most practical, good NLP& ML methods require labeled training data (i.e., supervised learning)

But almost all **data is unlabeled**

Most information must be acquired **unsupervised**

Fortunately, a good model of observed data can really help you learn classification decisions

# #4 Learning multiple levels of representation



Biologically inspired learning

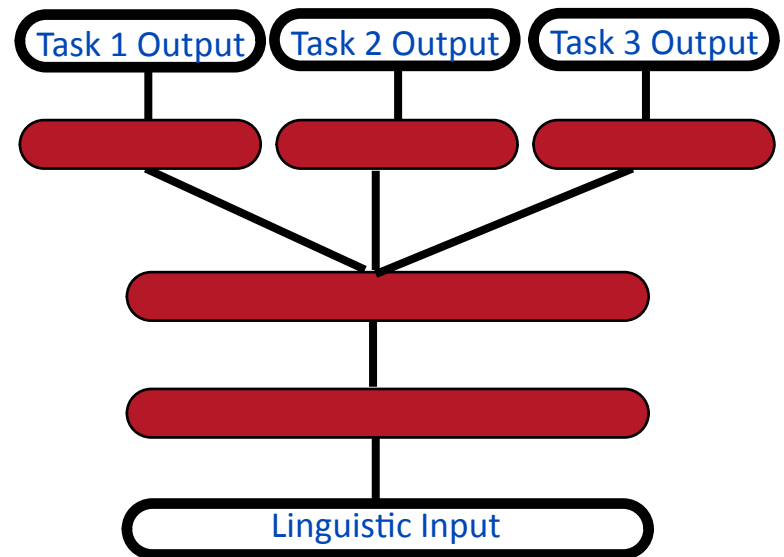
The cortex seems to have a generic learning algorithm

The brain has a deep architecture

We need good intermediate representations that can be shared across tasks

Multiple levels of latent variables allow combinatorial sharing of statistical strength

Insufficient model depth can be exponentially inefficient

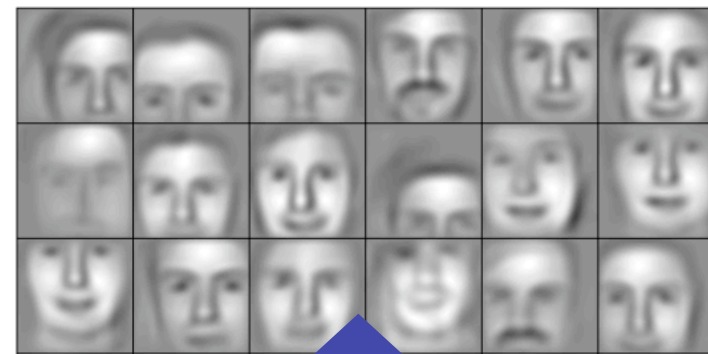


# #4 Learning multiple levels of representation

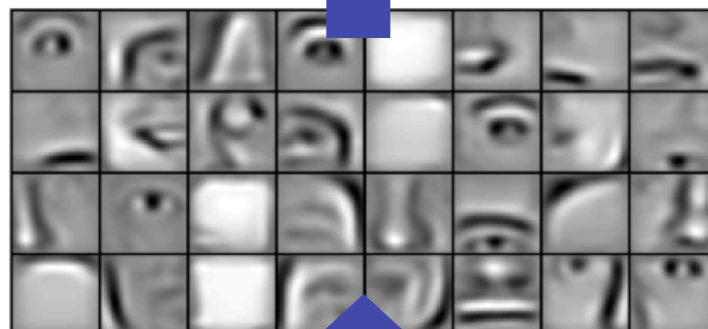


[Lee et al. ICML 2009; Lee et al. NIPS 2009]

Successive model layers learn deeper intermediate representations



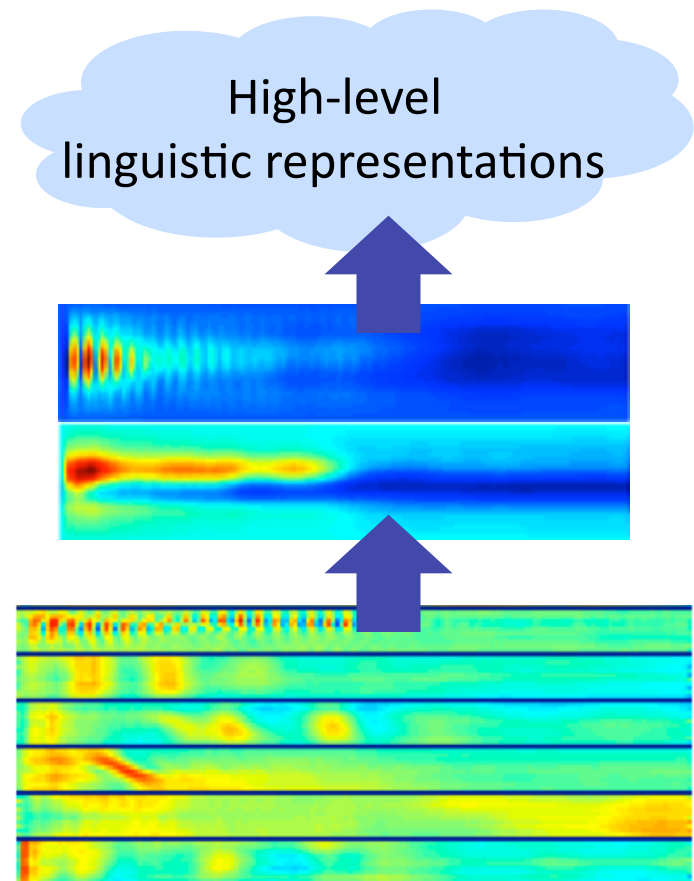
Layer 3



Layer 2



Layer 1



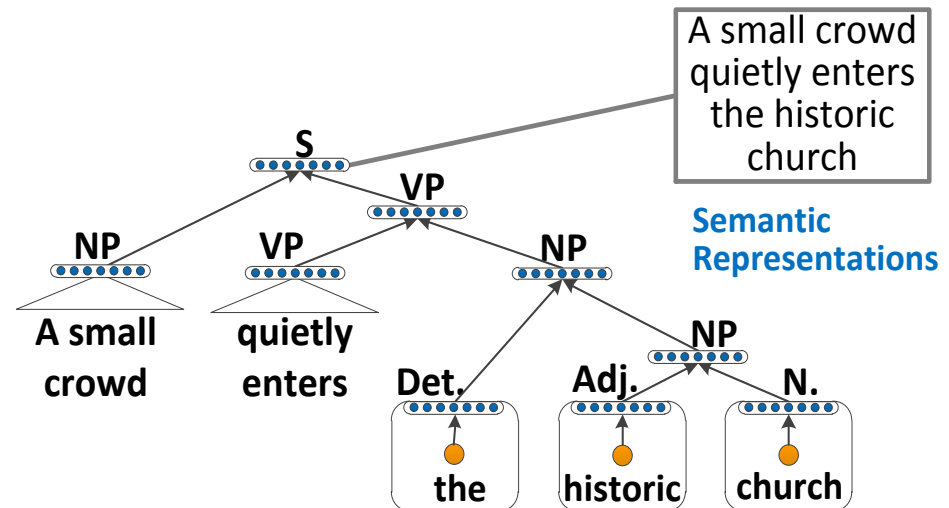
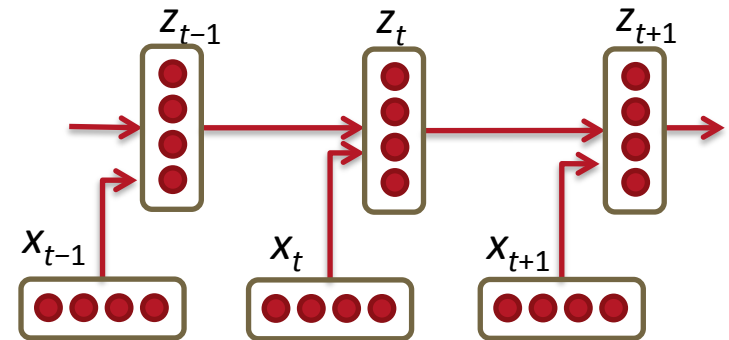


# Handling the recursivity of human language

Human sentences are composed from words and phrases

We need **compositionality** in our ML models

**Recursion**: the same operator (same parameters) is applied repeatedly on different components



# #5 Why now?

Despite prior investigation and understanding of many of the algorithmic techniques ...

Before 2006 training deep architectures was **unsuccessful** 😞

What has changed?

- New methods for unsupervised pre-training have been developed (Restricted Boltzmann Machines = RBMs, autoencoders, contrastive estimation, etc.)
- More efficient parameter estimation methods
- Better understanding of model regularization

# Deep Learning models have already achieved impressive results for HLT

## Neural Language Model [Mikolov et al. Interspeech 2011]



Model \ WSJ ASR task	Eval WER
KN5 Baseline	17.2
Discriminative LM	16.9
Recurrent NN combination	14.4

## MSR MAVIS Speech System [Dahl et al. 2012; Seide et al. 2011; following Mohamed et al. 2011]



“The algorithms represent the first time a company has released a deep-neural-networks (DNN)-based speech-recognition algorithm in a commercial product.”

Acoustic model & training	Recog \ WER	RT03S FSH	Hub5 SWB
GMM 40-mix, BMMI, SWB 309h	1-pass -adapt	27.4	23.6
DBN-DNN 7 layer x 2048, SWB 309h	1-pass -adapt	18.5 (-33%)	16.1 (-32%)
GMM 72-mix, BMMI, FSH 2000h	k-pass +adapt	18.6	17.1

# Deep Learn Models Have Interesting Performance Characteristics

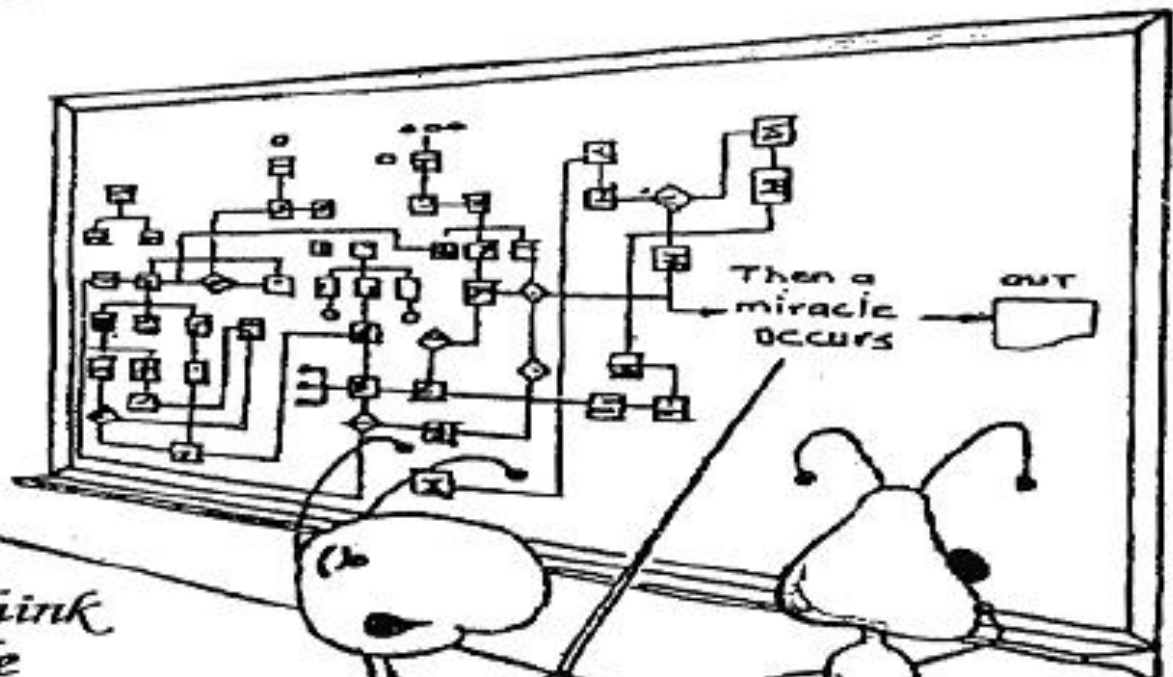
Deep learning models can now be very fast *in some circumstances*

- SENNA [Collobert et al. 2011] can do POS or NER faster than other SOTA taggers (16x to 122x), using 25x less memory
- WSJ POS 97.29% acc; CoNLL NER 89.59% F1; CoNLL Chunking 94.32% F1

Changes in computing technology favor deep learning

- In NLP, speed has traditionally come from exploiting sparsity
- But with modern machines, branches and widely spaced memory accesses are costly
- Uniform parallel operations on dense vectors are faster

These trends are even stronger with multi-core CPUs and GPUs



*Good work -- but I think we might need a little more detail right here.*



# Outline of the Tutorial

1. The Basics
  1. Motivations
  2. From logistic regression to neural networks
  3. Word representations
  4. Unsupervised word vector learning
  5. Backpropagation Training
  6. Learning word-level classifiers: POS and NER
  7. Sharing statistical strength
2. Recursive Neural Networks
3. Applications, Discussion, and Resources

# Outline of the Tutorial

1. The Basics
2. Recursive Neural Networks
  1. Motivation
  2. Recursive Neural Networks for Parsing
  3. Optimization and Backpropagation Through Structure
  4. Compositional Vector Grammars: Parsing
  5. Recursive Autoencoders: Paraphrase Detection
  6. Matrix-Vector RNNs: Relation classification
  7. Recursive Neural Tensor Networks: Sentiment Analysis
3. Applications, Discussion, and Resources

# Outline of the Tutorial

1. The Basics
2. Recursive Neural Networks
3. Applications, Discussion, and Resources
  1. Assorted Speech and NLP Applications
  2. Deep Learning: General Strategy and Tricks
  3. Resources (readings, code, ...)
  4. Discussion



Part 1.2: The Basics

# From logistic regression to neural nets

# Demystifying neural networks

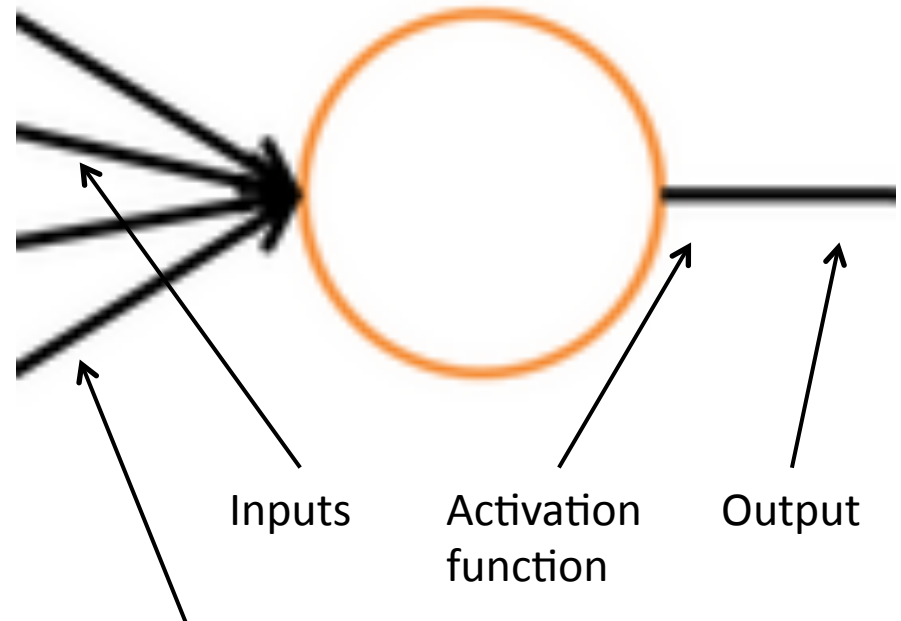
Neural networks come with their own terminological baggage

... just like SVMs

But if you understand how logistic regression or maxent models work

Then **you already understand** the operation of a basic neural network neuron!

**A single neuron**  
A computational unit with  $n$  (3) inputs and 1 output and parameters  $W, b$



Bias unit corresponds to intercept term

# From Maxent Classifiers to Neural Networks

In NLP, a maxent classifier is normally written as:

$$P(c | d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c' \in C} \exp \sum_i \lambda_i f_i(c', d)}$$

Supervised learning gives us a distribution for datum  $d$  over classes in  $C$

Vector form: 
$$P(c | d, \lambda) = \frac{e^{\lambda^\top f(c, d)}}{\sum_{c'} e^{\lambda^\top f(c', d)}}$$

Such a classifier is used as-is in a neural network (“a softmax layer”)

- Often as the top layer:  $J = \text{softmax}(\lambda \cdot x)$

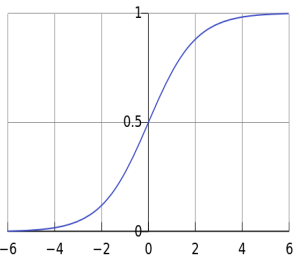
But for now we’ll derive a two-class logistic model for one neuron

# From Maxent Classifiers to Neural Networks

Vector form: 
$$P(c | d, \lambda) = \frac{e^{\lambda^T f(c,d)}}{\sum_{c'} e^{\lambda^T f(c',d)}}$$

Make two class:

$$\begin{aligned} P(c_1 | d, \lambda) &= \frac{e^{\lambda^T f(c_1,d)}}{e^{\lambda^T f(c_1,d)} + e^{\lambda^T f(c_2,d)}} = \frac{e^{\lambda^T f(c_1,d)}}{e^{\lambda^T f(c_1,d)} + e^{\lambda^T f(c_2,d)}} \cdot \frac{e^{-\lambda^T f(c_1,d)}}{e^{-\lambda^T f(c_1,d)}} \\ &= \frac{1}{1 + e^{\lambda^T [f(c_2,d) - f(c_1,d)]}} = \frac{1}{1 + e^{-\lambda^T x}} \quad \text{for } x = f(c_1,d) - f(c_2,d) \\ &= f(\lambda^T x) \end{aligned}$$



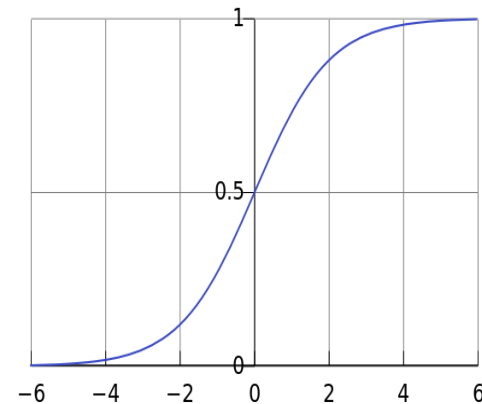
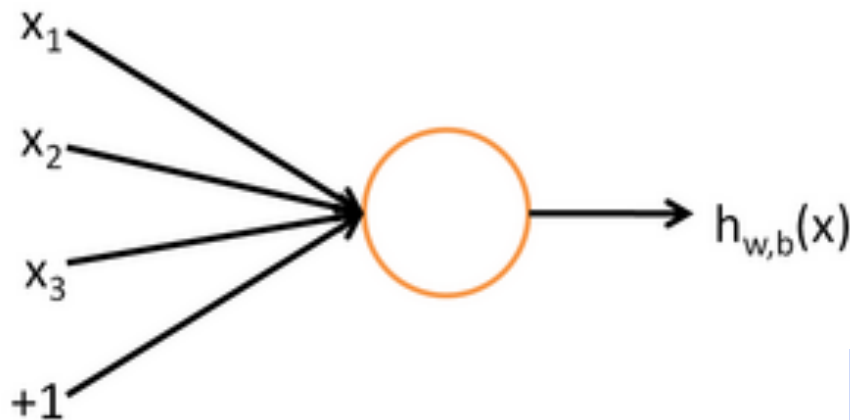
for  $f(z) = 1/(1 + \exp(-z))$ , the logistic function – a sigmoid **non-linearity**.

# This is exactly what a neuron computes

$$h_{w,b}(x) = f(w^T x + b)$$

$b$ : We can have an “always on” feature, which gives a class prior, or separate it out, as a bias term

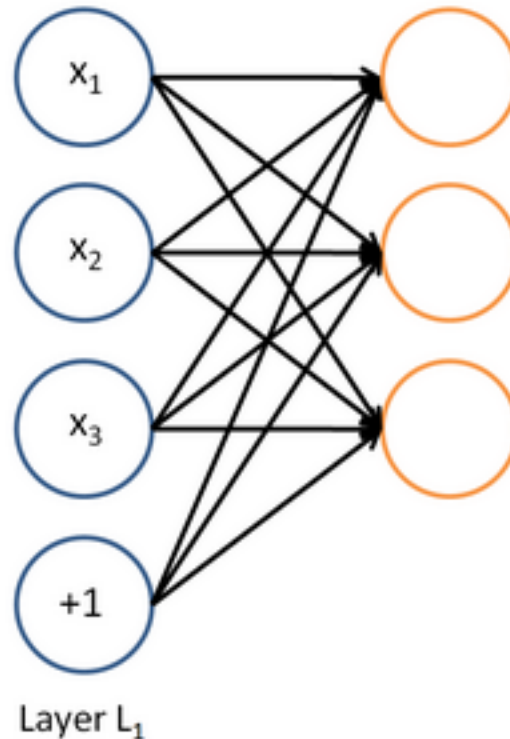
$$f(z) = \frac{1}{1 + e^{-z}}$$



$w, b$  are the parameters of this neuron  
i.e., this logistic regression model

# A neural network = running several logistic regressions at the same time

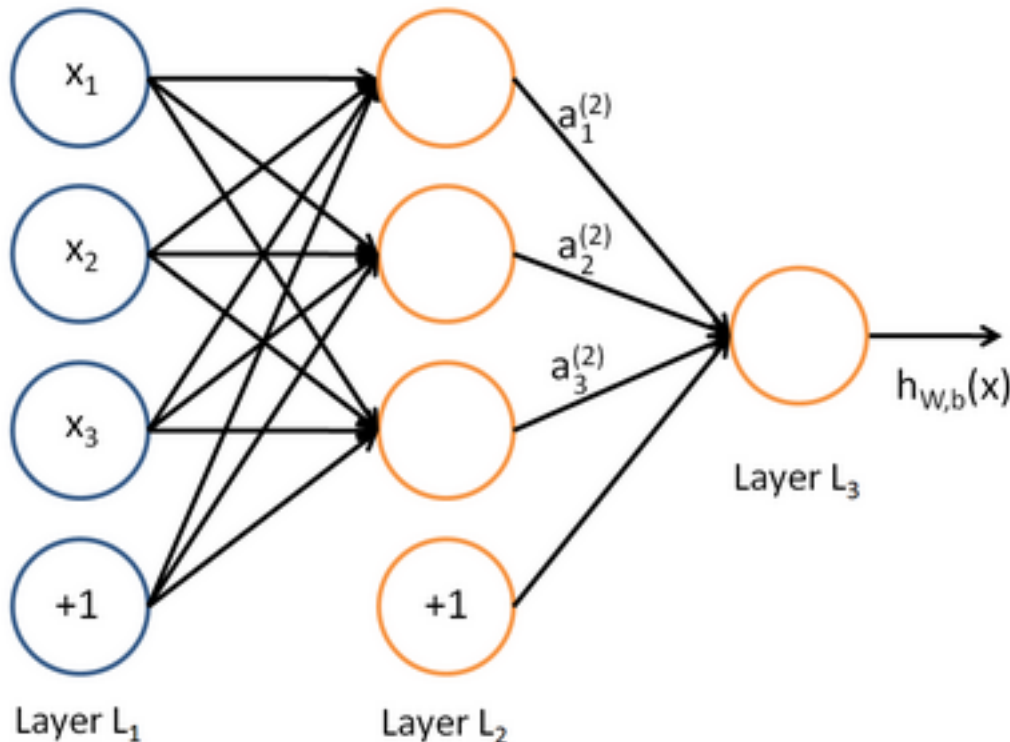
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs ...



*But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!*

# A neural network = running several logistic regressions at the same time

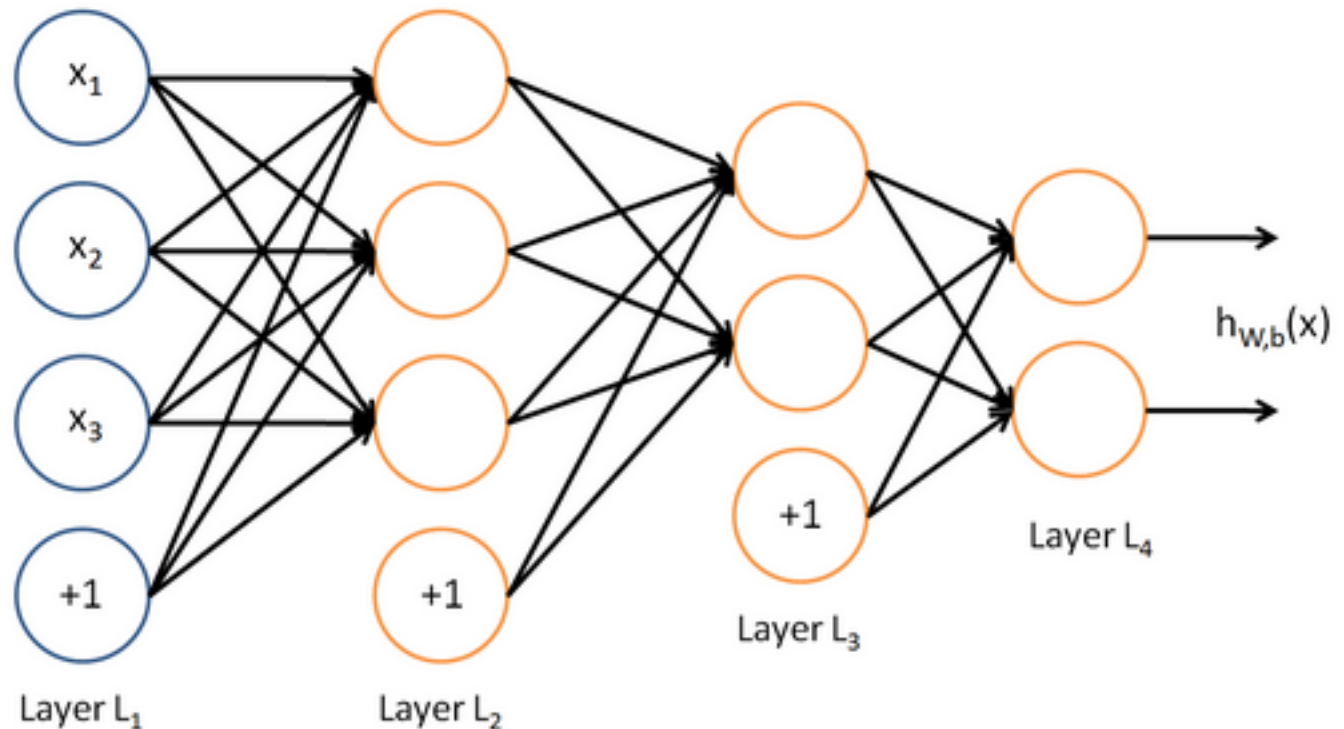
... which we can feed into another logistic regression function



*It is the training criterion that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.*

# A neural network = running several logistic regressions at the same time

Before we know it, we have a multilayer neural network....





# Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

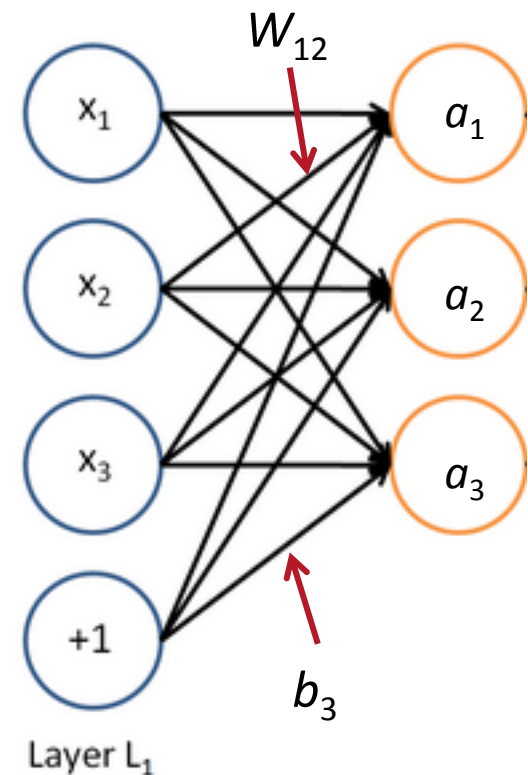
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

where  $f$  is applied element-wise:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

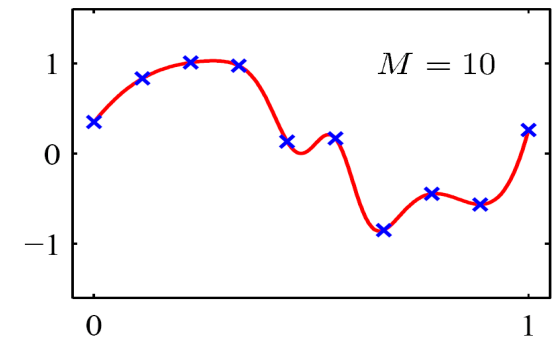
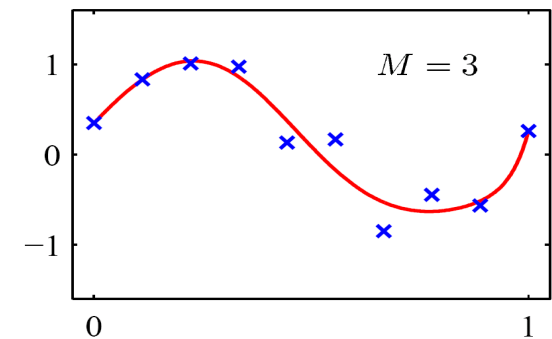
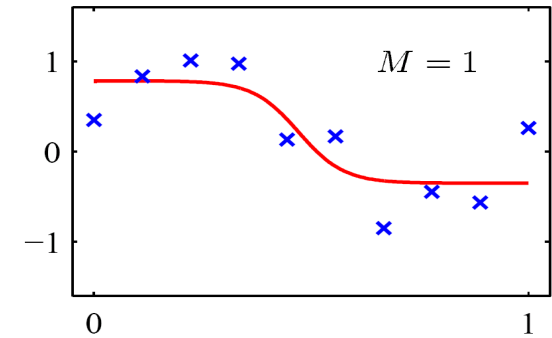


# How do we train the weights $W$ ?

- For a single supervised layer, we train just like a maxent model – we calculate and use error derivatives (gradients) to improve
  - Online learning: Stochastic gradient descent (SGD)
    - Or improved versions like AdaGrad (Duchi, Hazan, & Singer 2010)
  - Batch learning: Conjugate gradient or L-BFGS
- A multilayer net could be more complex because the internal (“hidden”) logistic units make the function non-convex ... just as for hidden CRFs [Quattoni et al. 2005, Gunawardana et al. 2005]
  - But we can use the same ideas and techniques
    - Just without guarantees ...
  - We “backpropagate” error derivatives through the model

# Non-linearities: Why they're needed

- For logistic regression: map to probabilities
- Here: function approximation, e.g., regression or classification
  - Without non-linearities, deep neural networks can't do anything more than a linear transform
    - Extra layers could just be compiled down into a single linear transform
  - Probabilistic interpretation unnecessary except in the Boltzmann machine/graphical models
    - People often use other non-linearities, such as **tanh**, as we'll discuss in part 3



# Summary

## Knowing the meaning of words!

You now understand the basics and the relation to other models

- Neuron = logistic regression or similar function
- Input layer = input training/test vector
- Bias unit = intercept term/always on feature
- Activation = response
- Activation function is a logistic (or similar “sigmoid” nonlinearity)
- Backpropagation = running stochastic gradient descent backward layer-by-layer in a multilayer network
- Weight decay = regularization / Bayesian prior

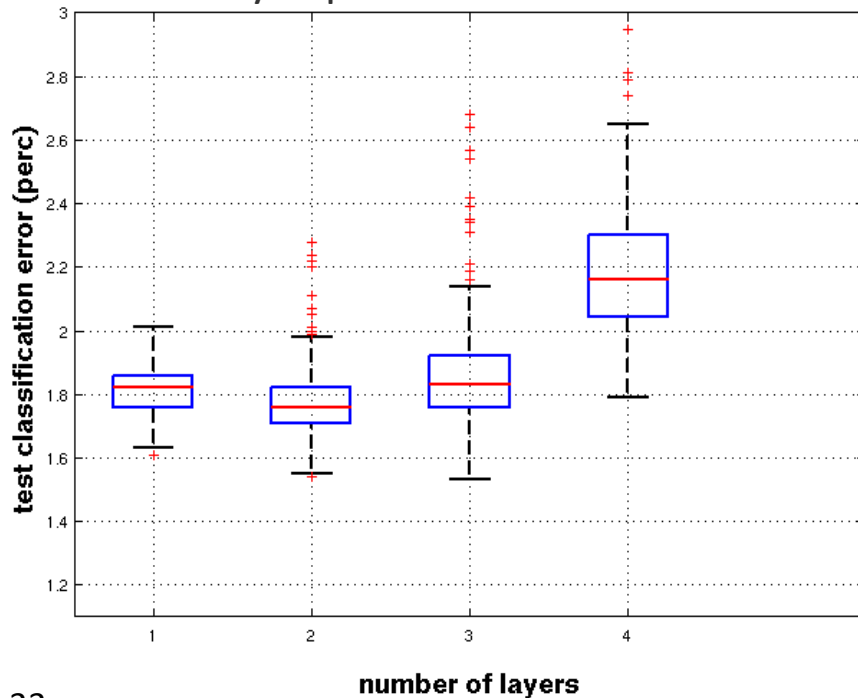
# Effective deep Learning became possible through unsupervised pre-training

[Erhan et al., JMLR 2010]

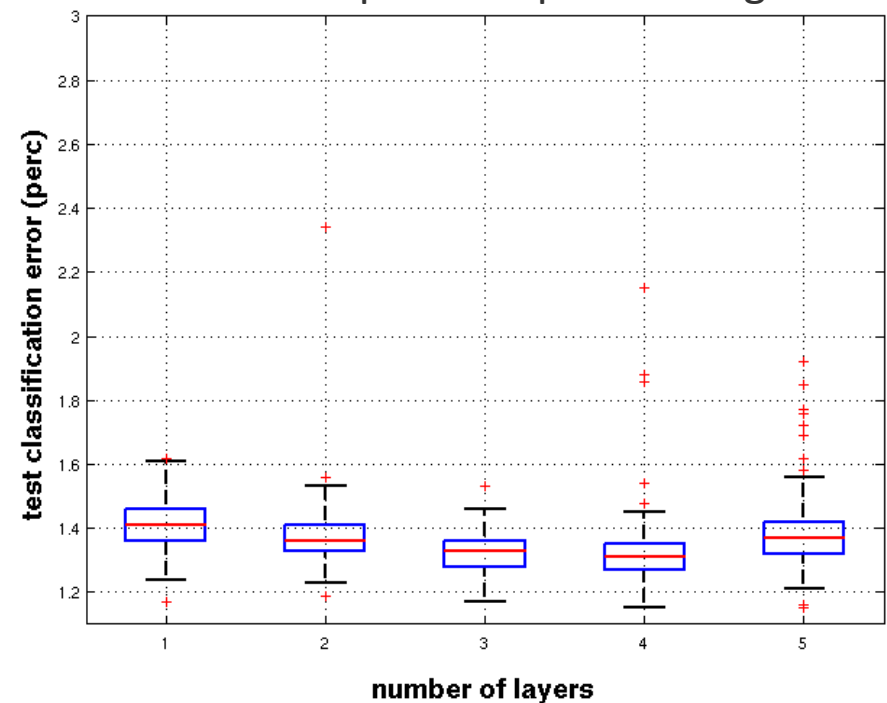


(with RBMs and Denoising Auto-Encoders)

Purely supervised neural net



With unsupervised pre-training



Part 1.3: The Basics

# Word Representations

# The standard word representation

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: *hotel*, *conference*, *walk*

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a “one-hot” representation. Its problem:

*motel* [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND  
*hotel* [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

# Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

You can vary whether you use local or large context to get a more syntactic or semantic clustering



# Class-based (hard) and soft clustering word representations

Class based models learn word classes of similar words based on distributional information (  $\sim$  class HMM)

- Brown clustering (Brown et al. 1992)
- Exchange clustering (Martin et al. 1998, Clark 2003)
- Desparsification and great example of unsupervised pre-training

Soft clustering models learn for each cluster/topic a distribution over words of how likely that word is in each cluster

- Latent Semantic Analysis (LSA/LSI), Random projections
- Latent Dirichlet Analysis (LDA), HMM clustering

# Neural word embeddings as a distributed representation

Similar idea

Combine vector space semantics with the prediction of probabilistic models (Bengio et al. 2003, Collobert & Weston 2008, Turian et al. 2010)

In all of these approaches, including deep learning models, a word is represented as a dense vector

*linguistics* =

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

# Neural word embeddings - visualization



# Stunning new result at this conference! Mikolov, Yih & Zweig (NAACL 2013)

These representations are *way* better at encoding dimensions of similarity than we realized!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

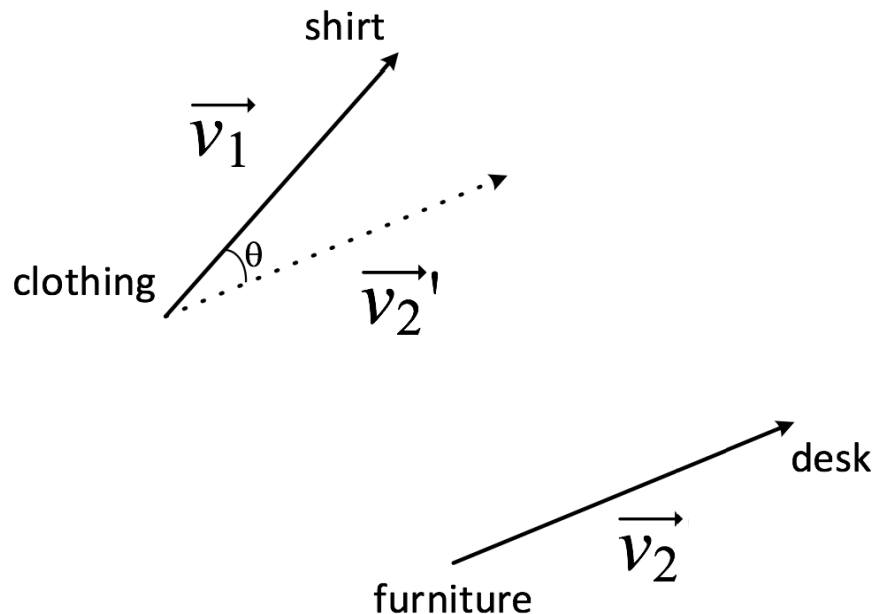
Syntactically

- $X_{apple} - X_{apples} \approx X_{car} - X_{cars} \approx X_{family} - X_{families}$
- Similarly for verb and adjective morphological forms

Semantically (Semeval 2012 task 2)

- $X_{shirt} - X_{clothing} \approx X_{chair} - X_{furniture}$

# Stunning new result at this conference! Mikolov, Yih & Zweig (NAACL 2013)



Method	Syntax % correct
LSA 320 dim	16.5 [best]
RNN 80 dim	16.2
RNN 320 dim	28.5
RNN 1600 dim	39.6
Method	Semantics Spearman $\rho$
UTD-NB (Rink & H. 2012)	0.230 [Semeval win]
LSA 640	0.149
RNN 80	0.211
RNN 1600	0.275 [new SOTA]

# Advantages of the neural word embedding approach

Compared to a method like LSA, neural word embeddings can become **more meaningful** through adding supervision from one or multiple tasks

“Discriminative fine-tuning”

For instance, sentiment is usually not captured in unsupervised word embeddings but can be in neural word vectors

We can build representations for large linguistic units

See part 2

Part 1.4: The Basics

# Unsupervised word vector Learning

# A neural network for learning word vectors

(Collobert et al. JMLR 2011)

Idea: A word and its context is a positive training sample; a random word in that same context gives a negative training sample:

+ cat chills on a mat      - cat chills Jeju a mat

Similar: Implicit negative evidence in Contrastive Estimation, (Smith and Eisner 2005)





# A neural network for learning word vectors

How do we formalize this idea? Ask that

score(cat chills on a mat) > score(cat chills Jeju a mat)

How do we compute the score?

- With a neural network
- Each word is associated with an  $n$ -dimensional vector



# Word embedding matrix

- Initialize all word vectors randomly to form a word embedding matrix  $L \in \mathbb{R}^{n \times |V|}$

$$L = \begin{bmatrix} \bullet & \bullet & \bullet & \dots & \bullet & \bullet \\ \bullet & \bullet & \bullet & \dots & \bullet & \bullet \\ \bullet & \bullet & \bullet & \dots & \bullet & \bullet \\ \bullet & \bullet & \bullet & \dots & \bullet & \bullet \end{bmatrix}_n$$

the   cat   mat   ...

- These are the word features we want to learn
- Also called a look-up table
  - Conceptually you get a word's vector by left multiplying a one-hot vector  $e$  by  $L$ :  $x = Le$

# Word vectors as input to a neural network

- $\text{score}(\text{cat chills on a mat})$
- To describe a phrase, retrieve (via index) the corresponding vectors from  $L$



- Then concatenate them to  $5n$  vector:
- $x = [ \text{●●●●} \text{●●●●} \text{●●●●} \text{●●●●} \text{●●●●} ]$
- How do we then compute  $\text{score}(x)$ ?

# A Single Layer Neural Network

- A single layer was a combination of a linear layer and a nonlinearity: 
$$z = Wx + b$$
$$a = f(z)$$

- The neural activations  $a$  can then be used to compute some function

- For instance, the score we care about:

$$\text{score}(x) = U^T a \in \mathbb{R}$$

# Summary: Feed-forward Computation

Computing a window's score with a 3-layer Neural Net:  $s = \text{score}(\text{cat chills on a mat})$

$$s = U^T f(Wx + b) \quad x \in \mathbb{R}^{20 \times 1}, W \in \mathbb{R}^{8 \times 20}, U \in \mathbb{R}^{8 \times 1}$$

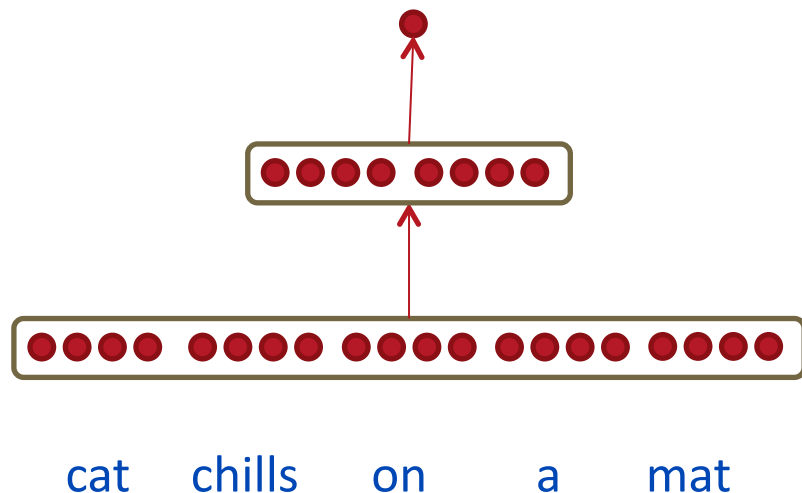
$$s = U^T a$$

$$a = f(z)$$

$$z = Wx + b$$

$$x = [x_{cat} \ x_{chills} \ x_{on} \ x_a \ x_{mat}]$$

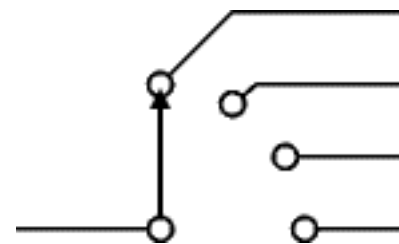
$$L \in \mathbb{R}^{n \times |V|}$$



# Summary: Feed-forward Computation

- $s = \text{score}(\text{cat chills on a mat})$
- $s_c = \text{score}(\text{cat chills Jeju a mat})$
- Idea for training objective: make score of true window larger and corrupt window's score lower (until they're good enough): minimize

$$J = \max(0, 1 - s + s_c)$$



- This is continuous, can perform SGD

# Training with Backpropagation

$$J = \max(0, 1 - s + s_c)$$

$$s = U^T f(Wx + b)$$

$$s_c = U^T f(Wx_c + b)$$

Assuming cost  $J$  is  $> 0$ , it is simple to see that we can compute the derivatives of  $s$  and  $s_c$  wrt all the involved variables:  $U, W, b, x$

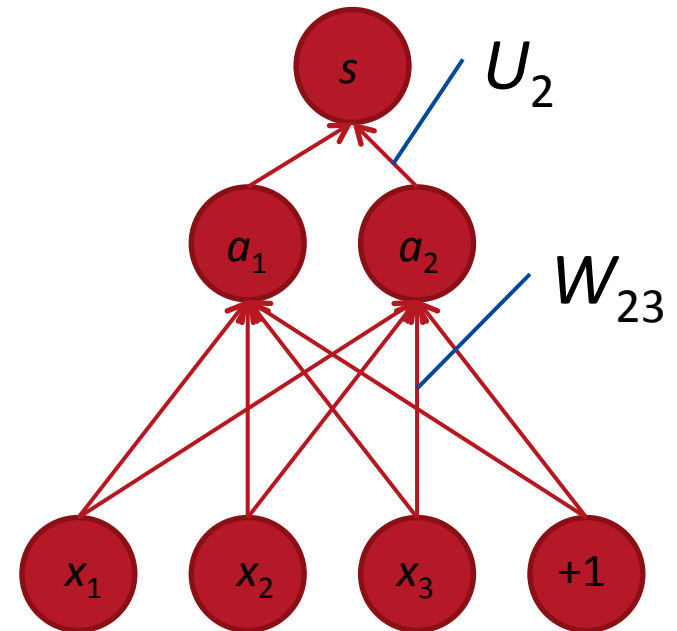
$$\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a \qquad \frac{\partial s}{\partial U} = a$$

# Training with Backpropagation

- Let's consider the derivative of a single weight  $W_{ij}$

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

- This only appears inside  $a_i$
- For example:  $W_{23}$  is only used to compute  $a_2$





# Training with Backpropagation

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

Derivative of weight  $W_{ij}$ :

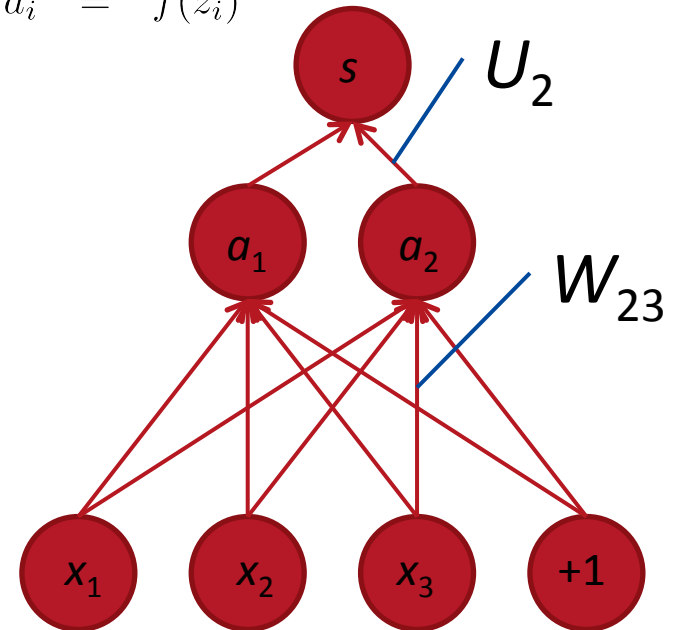
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

$$\frac{\partial}{\partial W_{ij}} U^T a \rightarrow \frac{\partial}{\partial W_{ij}} U_i a_i$$

$$z_i = W_{i \cdot} x + b_i = \sum_{j=1}^3 W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$

$$\begin{aligned} U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial W_{i \cdot} x + b_i}{\partial W_{ij}} \end{aligned}$$



# Training with Backpropagation

Derivative of single weight  $W_{ij}$ :

$$= U_i f'(z_i) \frac{\partial W_i \cdot x + b_i}{\partial W_{ij}}$$

$$= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k$$

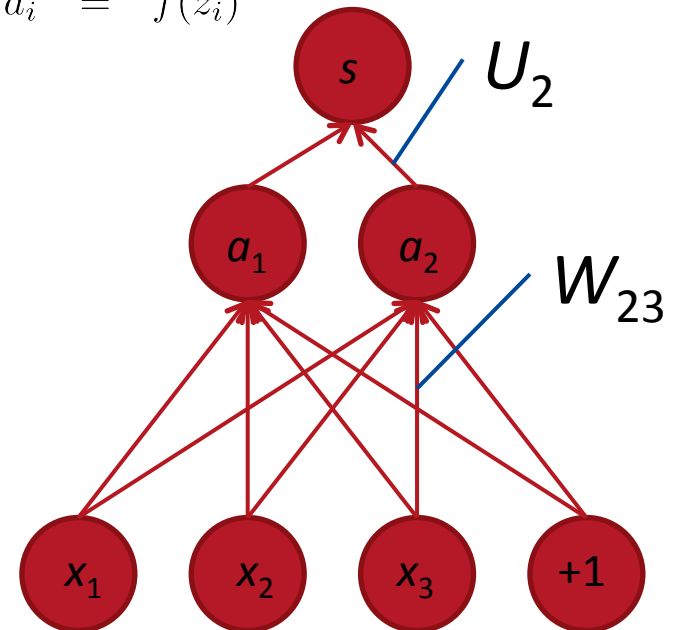
$$= \underbrace{U_i f'(z_i)}_{\delta_i} x_j$$

$$= \underbrace{\delta_i}_{\text{Local error signal}} \underbrace{x_j}_{\text{Local input signal}}$$

$$U_i \frac{\partial}{\partial W_{ij}} a_i$$

$$z_i = W_i \cdot x + b_i = \sum_{j=1}^3 W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$



# Training with Backpropagation

- From single weight  $W_{ij}$  to full  $W$ :

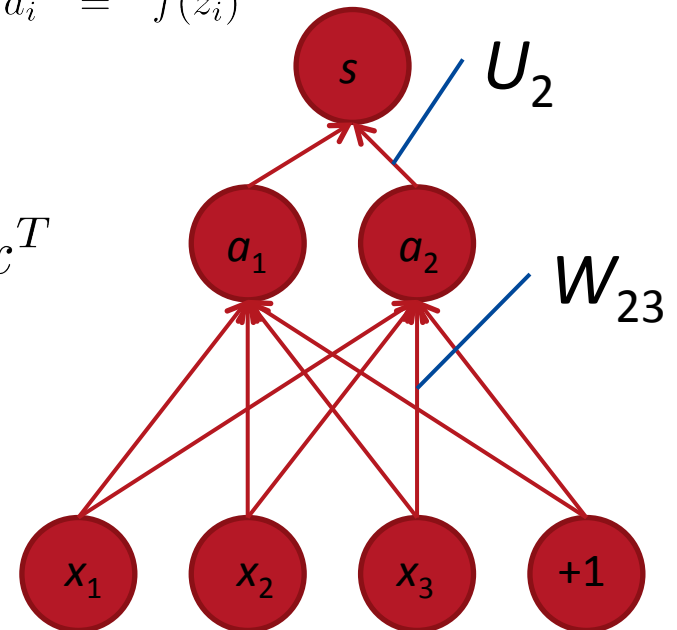
$$\begin{aligned}\frac{\partial J}{\partial W_{ij}} &= \underbrace{U_i f'(z_i)}_{\delta_i} x_j \\ &= \delta_i x_j\end{aligned}$$

$$z_i = W_i \cdot x + b_i = \sum_{j=1}^3 W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$

- We want all combinations of  $i = 1, 2$  and  $j = 1, 2, 3$

- Solution: Outer product:  $\frac{\partial J}{\partial W} = \delta x^T$   
where  $\delta \in \mathbb{R}^{2 \times 1}$  is the “responsibility” coming from each activation  $a$



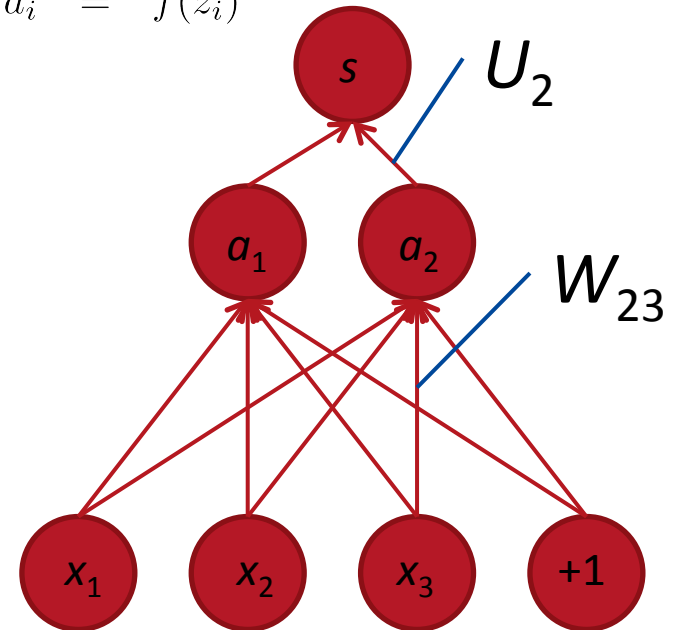
# Training with Backpropagation

- For biases  $b$ , we get:

$$\begin{aligned} & U_i \frac{\partial}{\partial b_i} a_i \\ = & U_i f'(z_i) \frac{\partial W_i \cdot x + b_i}{\partial b_i} \\ = & \delta_i \end{aligned}$$

$$z_i = W_i \cdot x + b_i = \sum_{j=1}^3 W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$



# Training with Backpropagation

That's almost backpropagation

It's simply taking derivatives and using the chain rule!

Remaining trick: we can re-use derivatives computed for higher layers in computing derivatives for lower layers

Example: last derivatives of model, the word vectors in  $x$

# Training with Backpropagation

- Take derivative of score with respect to single word vector (for simplicity a 1d vector, but same if it was longer)
- Now, we cannot just take into consideration one  $a_i$  because each  $x_j$  is connected to all the neurons above and hence  $x_j$  influences the overall score through all of these, hence:

$$\begin{aligned}
 \frac{\partial s}{\partial x_j} &= \sum_{i=1}^2 \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
 &= \sum_{i=1}^2 \frac{\partial U^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
 &= \sum_{i=1}^2 U_i \frac{\partial f(W_i \cdot x + b)}{\partial x_j} \\
 &= \sum_{i=1}^2 \underbrace{U_i f'(W_i \cdot x + b)}_{\delta_i} \frac{\partial W_i \cdot x}{\partial x_j} \\
 &= \sum_{i=1}^2 \delta_i W_{ij} \\
 &= \underbrace{\delta^T}_{\text{Re-used part of previous derivative}} W_{\cdot j}
 \end{aligned}$$

# Training with Backpropagation: softmax

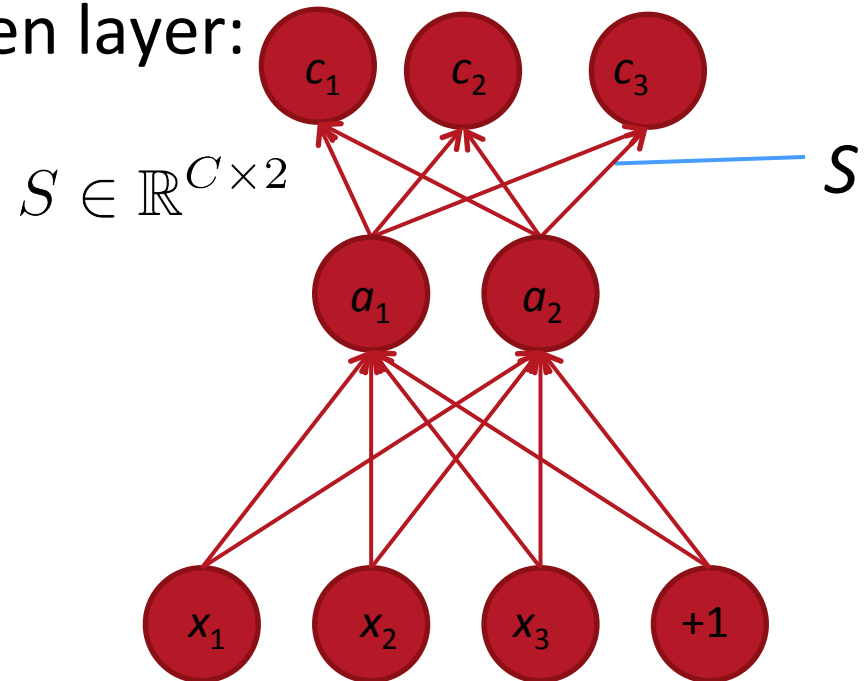
What is the major benefit of deep learned word vectors?

Ability to also propagate labeled information into them,  
via softmax/maxent and hidden layer:

$$P(c|d, \lambda) = \frac{e^{\lambda^T f(c,d)}}{\sum_{c'} e^{\lambda^T f(c',d)}}$$



$$p(c|x) = \frac{\exp(S_c \cdot a)}{\sum_{c'} \exp(S_{c'} \cdot a)}$$



Part 1.5: The Basics

# Backpropagation Training



# Back-Prop

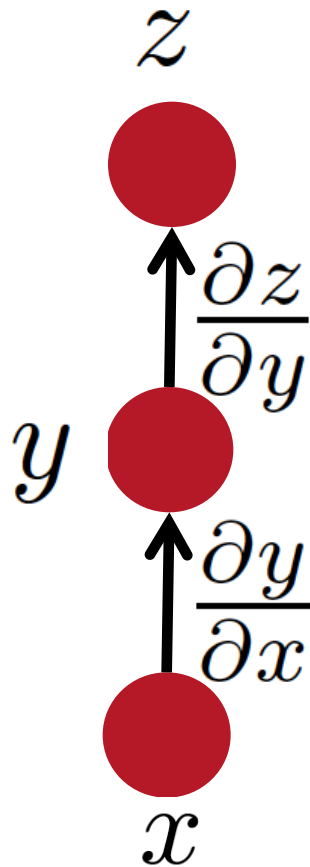
- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivative chain rule wisely

$$z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

- If computing the loss(example, parameters) is  $O(n)$  computation, then so is computing the gradient

# Simple Chain Rule



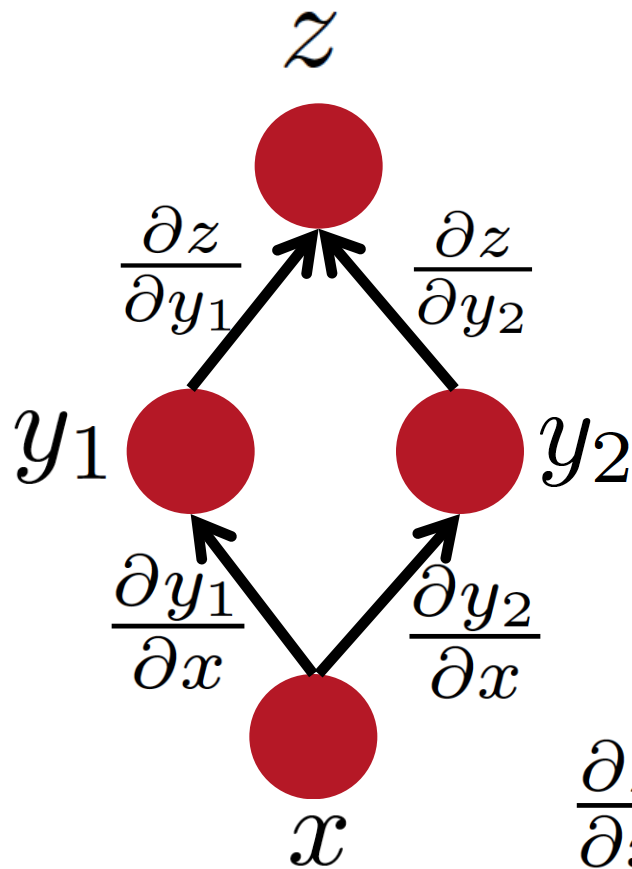
$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

$$\Delta y = \frac{\partial y}{\partial x} \Delta x$$

$$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$$

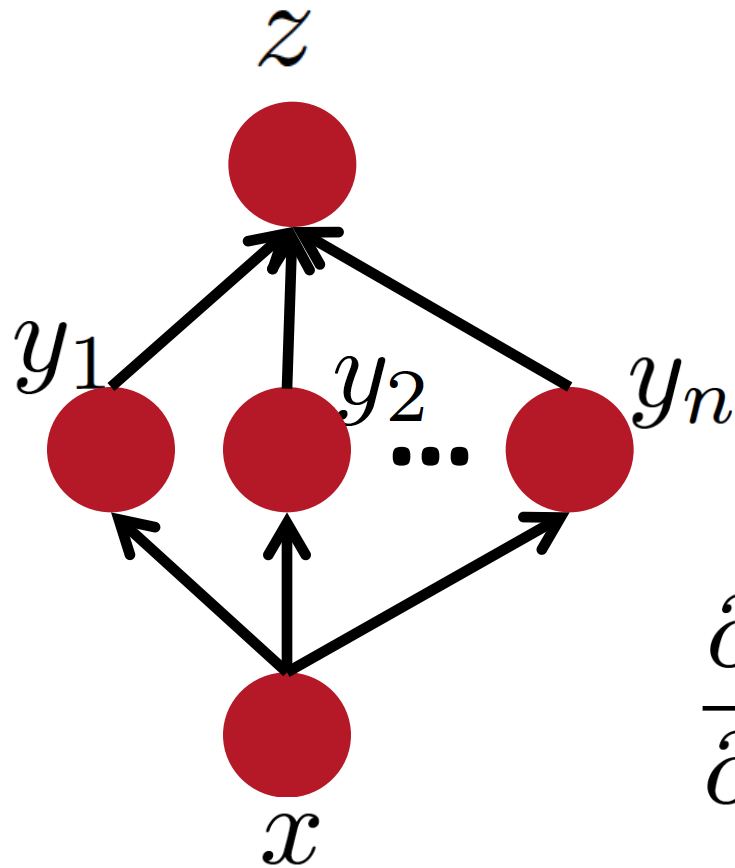
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

# Multiple Paths Chain Rule



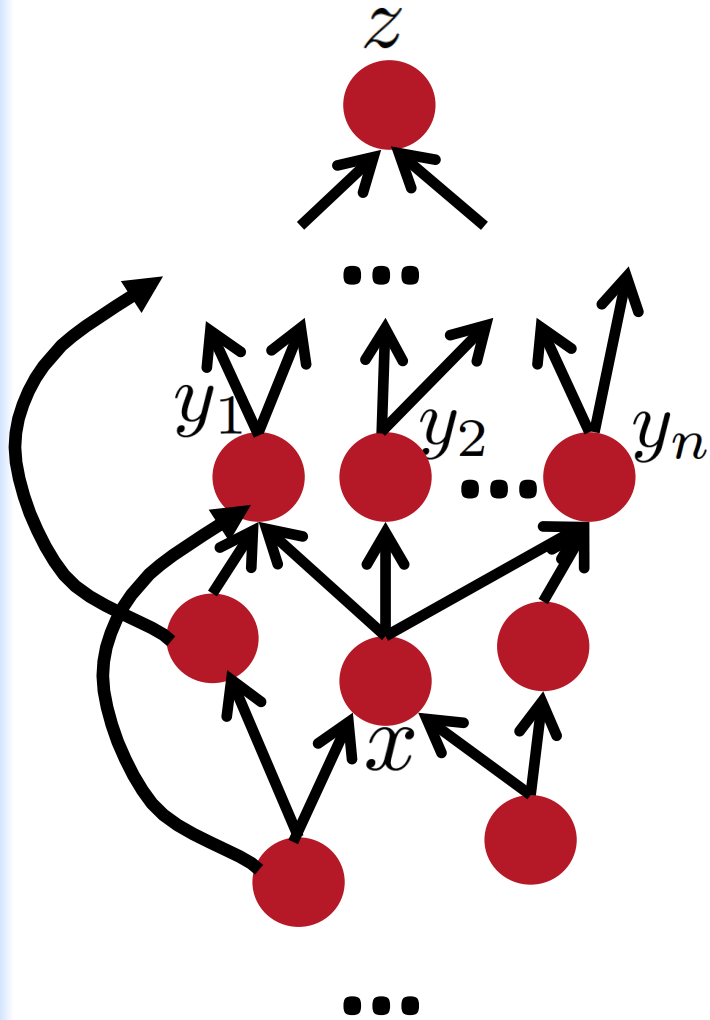
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$

# Multiple Paths Chain Rule - General



$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Chain Rule in Flow Graph

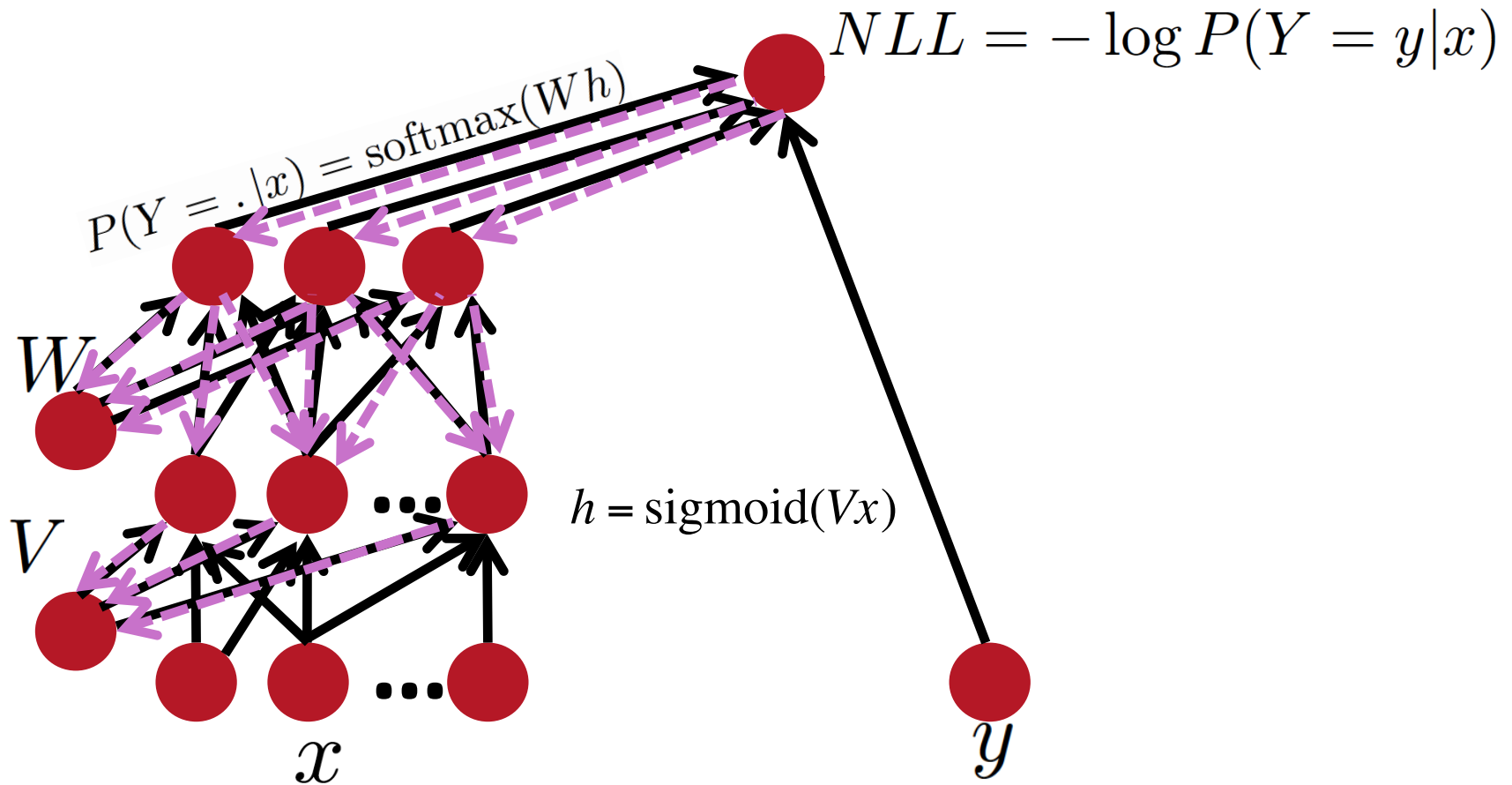


Flow graph: any directed acyclic graph  
node = computation result  
arc = computation dependency

$\{y_1, y_2, \dots, y_n\}$  = successors of  $x$

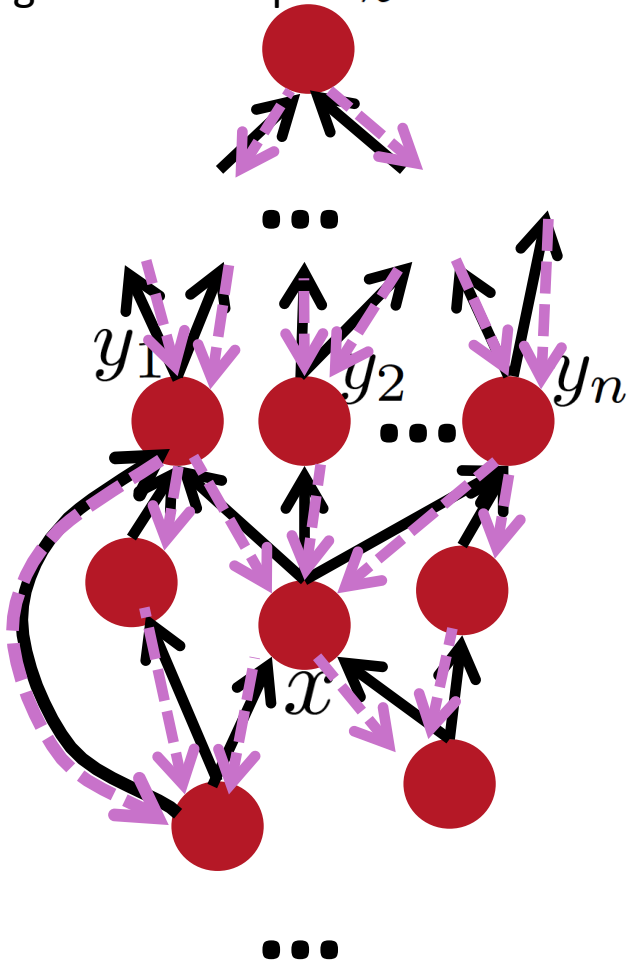
$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Back-Prop in Multi-Layer Net



# Back-Prop in General Flow Graph

Single scalar output  $z$

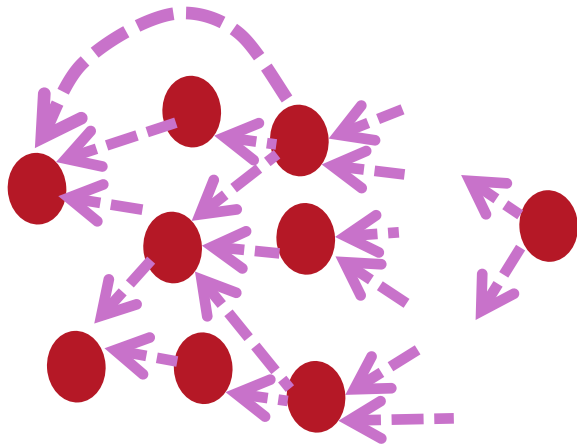
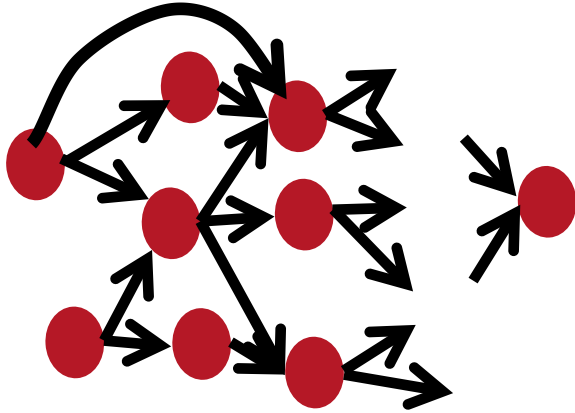


1. Fprop: visit nodes in topo-sort order
  - Compute value of node given predecessors
2. Bprop:
  - initialize output gradient = 1
  - visit nodes in reverse order:
    - Compute gradient wrt each node using gradient wrt successors

$\{y_1, y_2, \dots, y_n\}$  = successors of  $x$

$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

# Automatic Differentiation



- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping



Part 1.6: The Basics

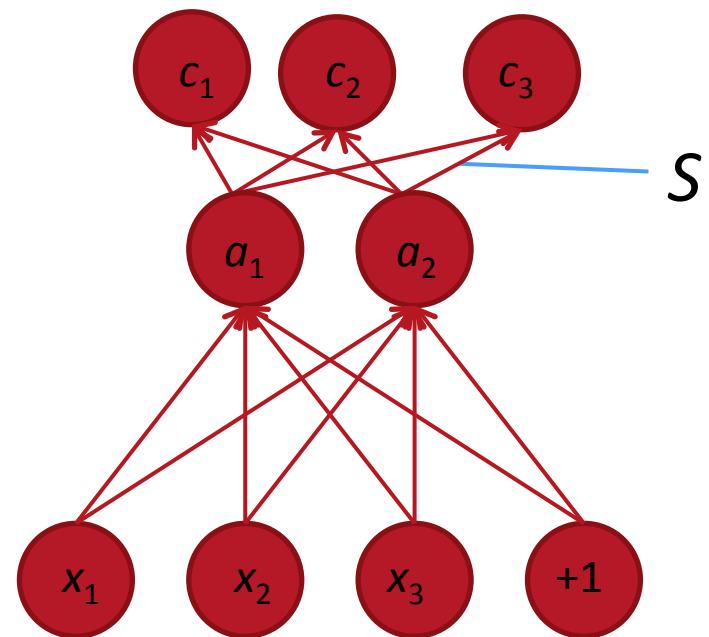
# Learning word-level classifiers: POS and NER

# The Model

(Collobert & Weston 2008;  
Collobert et al. 2011)

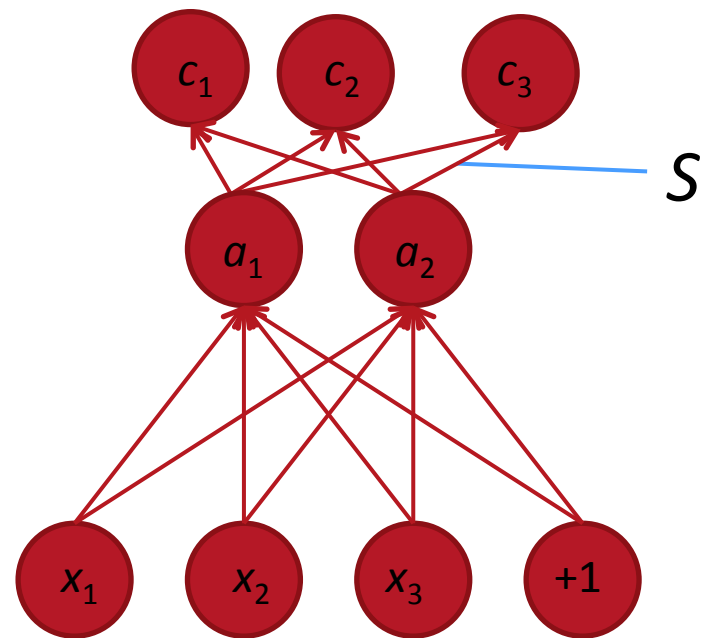
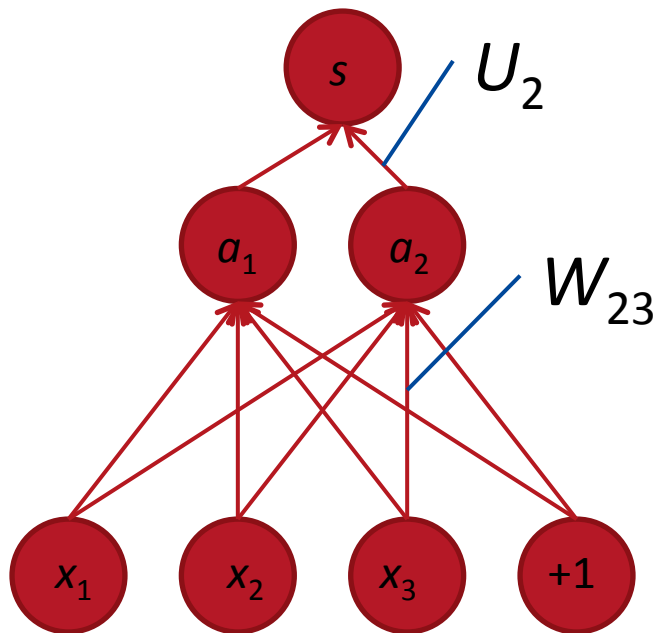


- Similar to word vector learning but replaces the single scalar score with a *Softmax/Maxent* classifier
- Training is again done via backpropagation which gives an error similar to the score in the unsupervised word vector learning model



# The Model - Training

- We already know the softmax classifier and how to optimize it
- The interesting twist in deep learning is that the input features are also learned, similar to learning word vectors with a score:



# The secret sauce is the unsupervised pre-training on a large text collection

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	<b>96.37</b>	<b>81.47</b>
Unsupervised pre-training followed by supervised NN**	<b>97.20</b>	<b>88.87</b>
+ hand-crafted features***	97.29	89.59

\* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)

\*\* 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – **for 7 weeks!** – then supervised task training

\*\*\* Features are character suffixes for POS and a gazetteer for NER

# Supervised refinement of the unsupervised word representation helps

	POS WSJ (acc.)	NER CoNLL (F1)
Supervised NN	96.37	81.47
NN with Brown clusters	96.92	87.15
Fixed embeddings*	<b>97.10</b>	<b>88.87</b>
<b>C&amp;W 2011**</b>	<b>97.29</b>	<b>89.59</b>

\* Same architecture as C&W 2011, but word embeddings are kept constant during the supervised training phase

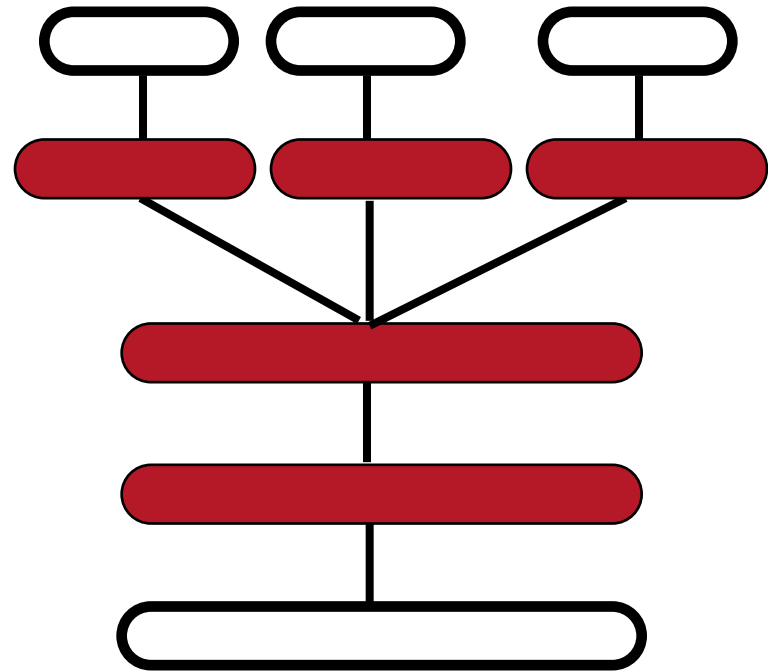
\*\* C&W is unsupervised pre-train + supervised NN + features model of last slide

Part 1.7

# Sharing statistical strength

# Multi-Task Learning

- Generalizing better to new tasks is crucial to approach AI
- Deep architectures learn good intermediate representations that can be shared across tasks
- Good representations make sense for many tasks



# Combining Multiple Sources of Evidence with Shared Embeddings

- Relational learning
- Multiple sources of information / relations
- Some symbols (e.g. words, wikipedia entries) shared
- Shared embeddings help propagate information among data sources: e.g., WordNet, XWN, Wikipedia, FreeBase, ...



# Sharing Statistical Strength

- Besides very fast prediction, the main advantage of deep learning is **statistical**
- Potential to learn from less labeled examples because of sharing of statistical strength:
  - Unsupervised pre-training & multi-task learning
  - Semi-supervised learning →

# Semi-Supervised Learning

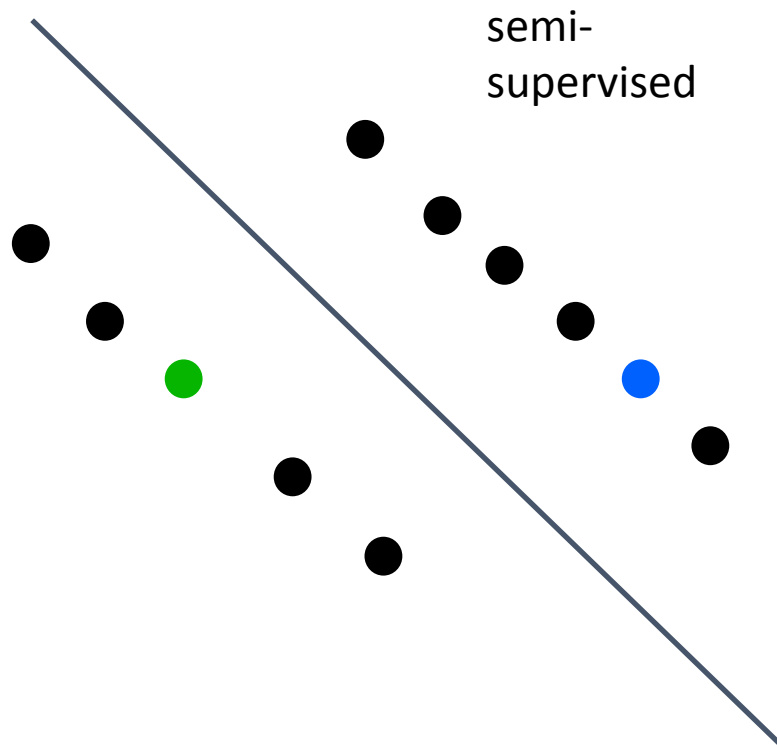
- Hypothesis:  $P(c|x)$  can be more accurately computed using shared structure with  $P(x)$



purely supervised

# Semi-Supervised Learning

- Hypothesis:  $P(c|x)$  can be more accurately computed using shared structure with  $P(x)$



# Deep autoencoders

Alternative to contrastive unsupervised word learning

- Another is RBMs ([Hinton et al. 2006](#)), which we don't cover today

Works well for fixed input representations

1. Definition, intuition and variants of autoencoders
2. Stacking for deep autoencoders
3. Why do autoencoders improve deep neural nets so much?

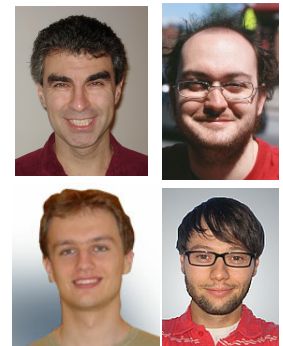
# Auto-Encoders

- Multilayer neural net with target output = input
- Reconstruction=decoder(encoder(input))

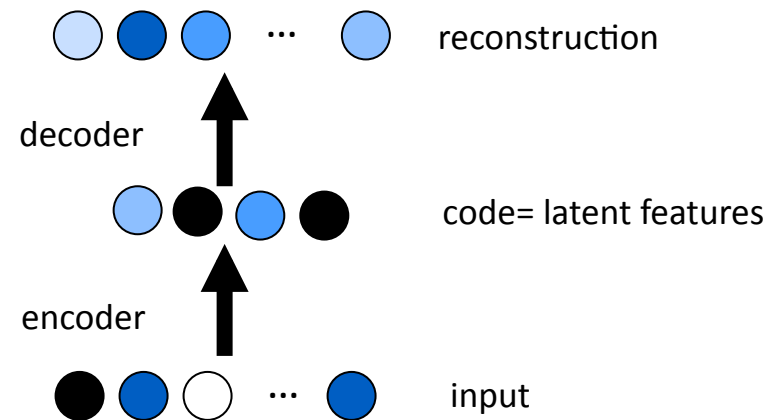
$$a = \tanh(Wx + b)$$

$$x' = \tanh(W^T a + c)$$

$$cost = \|x' - x\|^2$$



- Probable inputs have small reconstruction error



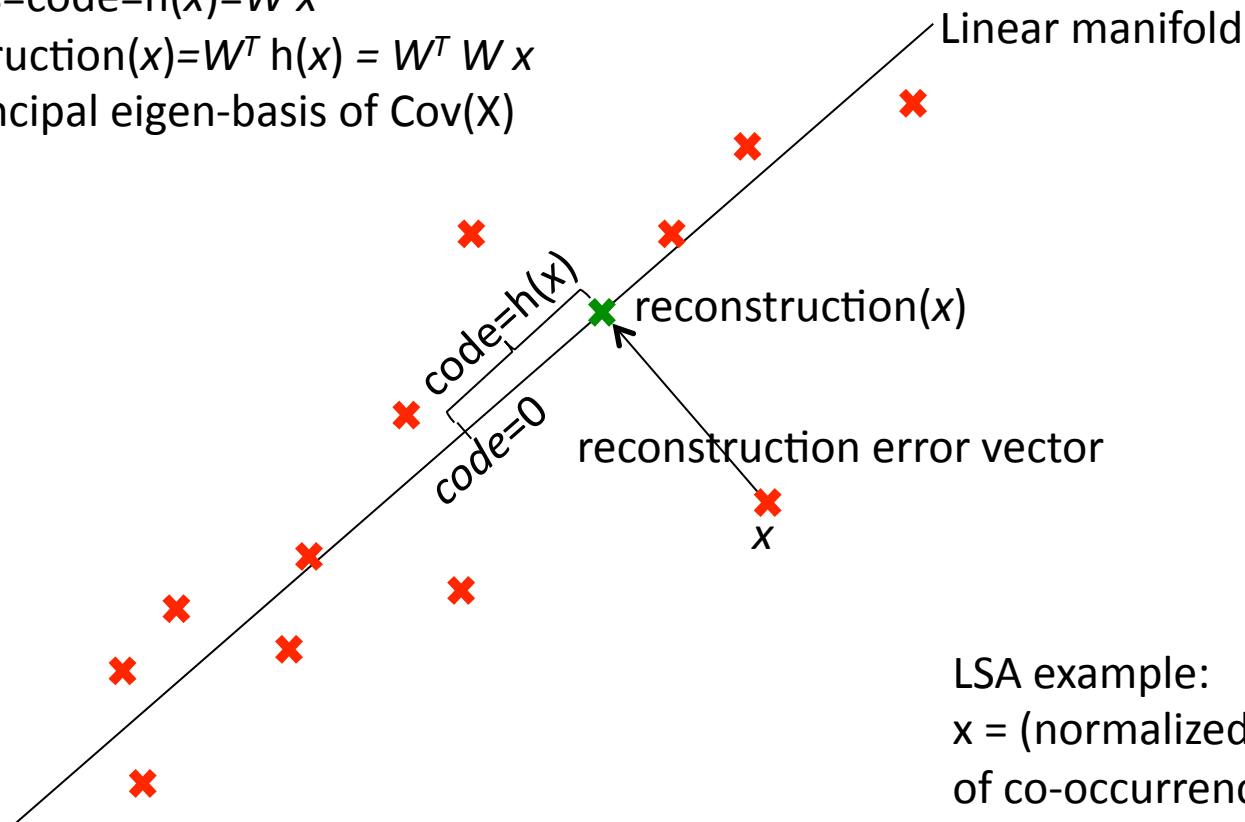
# PCA = Linear Manifold = Linear Auto-Encoder

input  $x$ , 0-mean

features=code= $h(x)=W x$

reconstruction( $x$ )= $W^T h(x) = W^T W x$

$W$  = principal eigen-basis of  $\text{Cov}(X)$

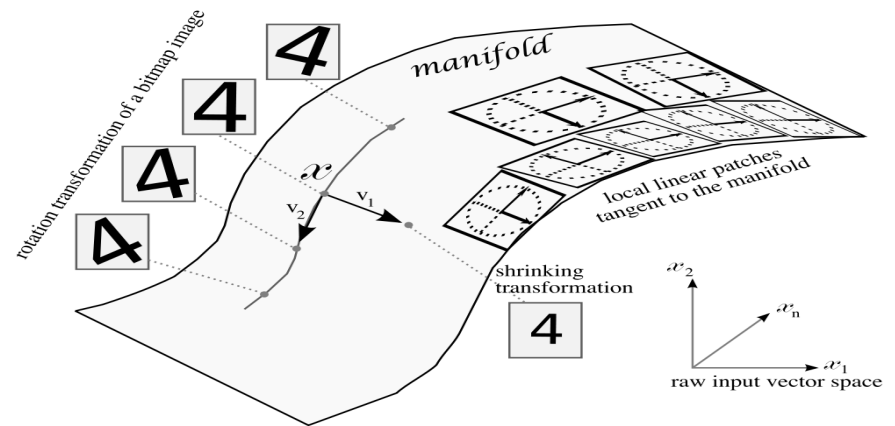
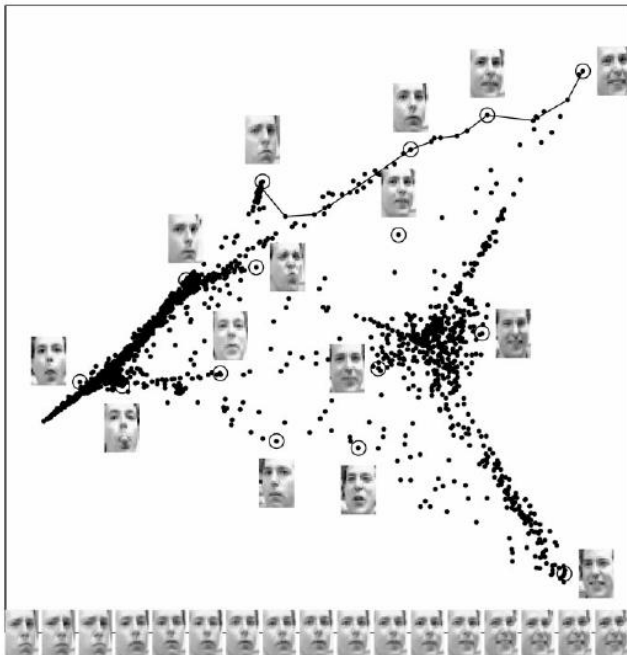


LSA example:

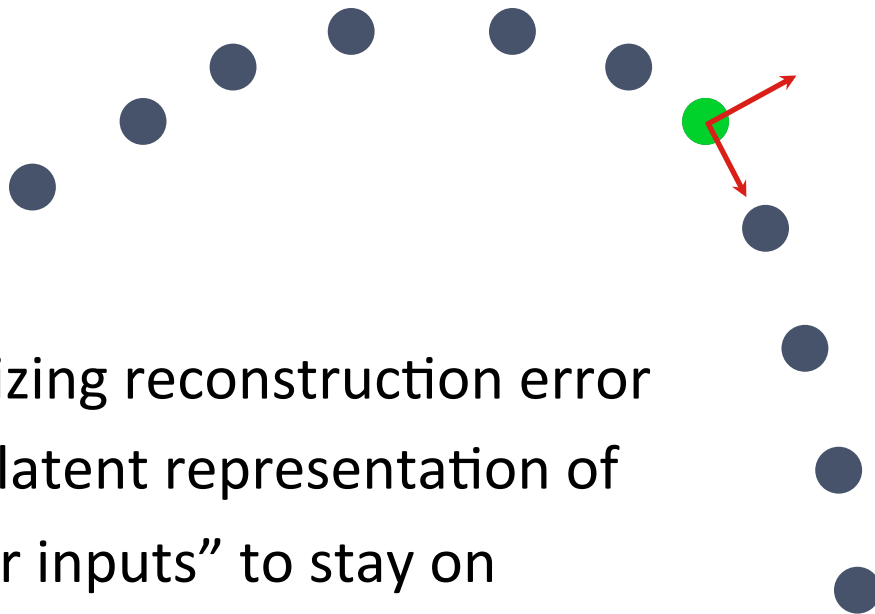
$x$  = (normalized) distribution  
of co-occurrence frequencies

# The Manifold Learning Hypothesis

- Examples concentrate near a lower dimensional “manifold” (region of high density where small changes are only allowed in certain directions)



# Auto-Encoders Learn Salient Variations, like a non-linear PCA



Minimizing reconstruction error forces latent representation of “similar inputs” to stay on manifold

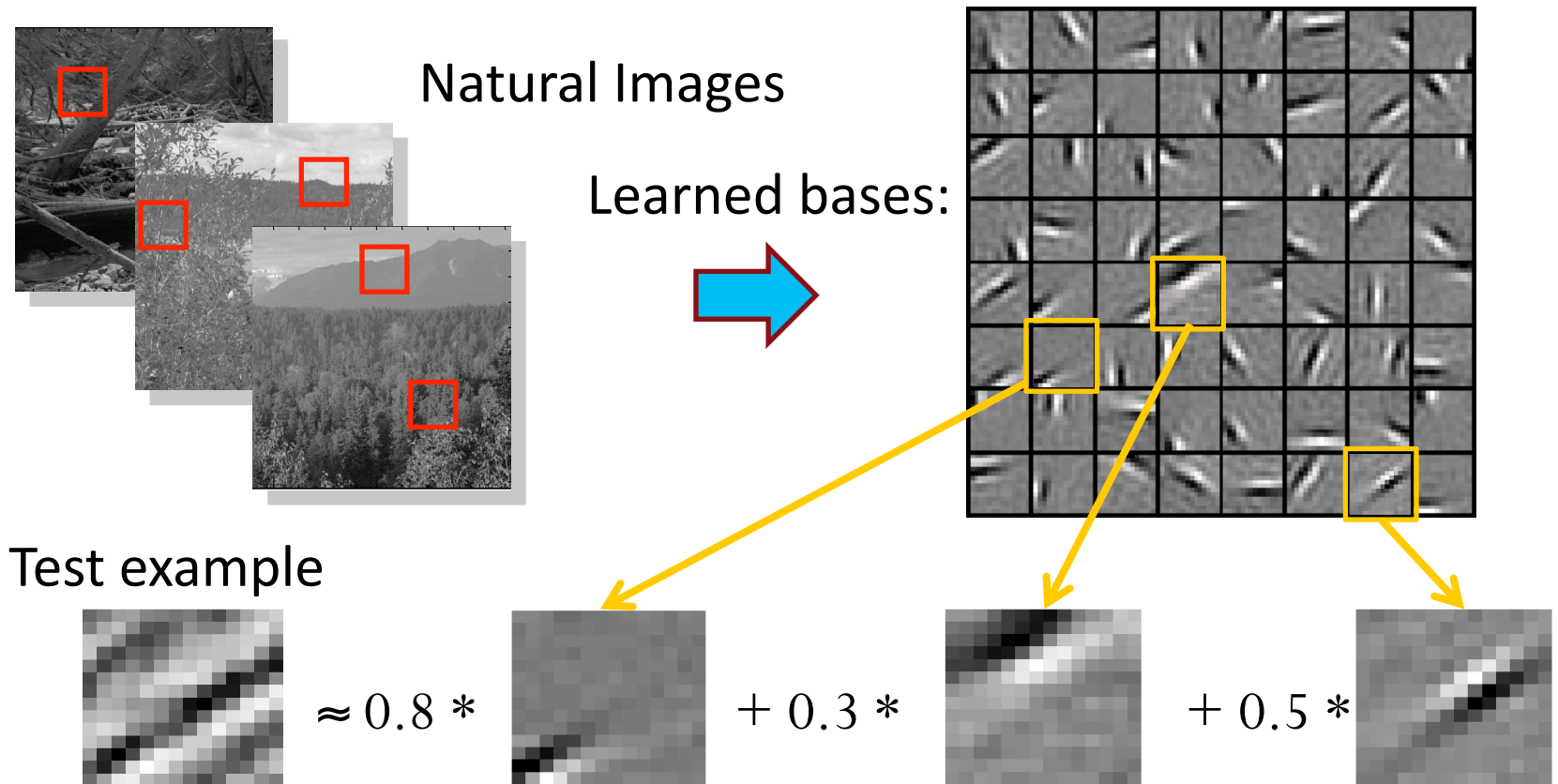


# Auto-Encoder Variants

- Discrete inputs: cross-entropy or log-likelihood reconstruction criterion (similar to used for discrete targets for MLPs)
- Preventing them to learn the identity everywhere:
  - Undercomplete (eg PCA): bottleneck code smaller than input
  - Sparsity: penalize hidden unit activations so at or near 0  
[Goodfellow et al 2009]
  - Denoising: predict true input from corrupted input  
[Vincent et al 2008]
  - Contractive: force encoder to have small derivatives  
[Rifai et al 2011]



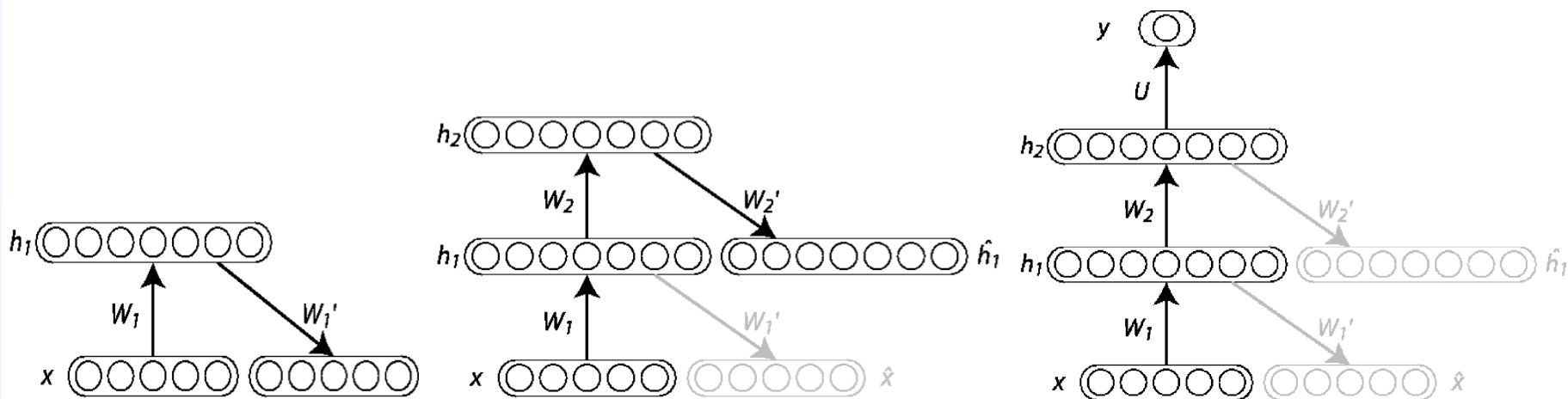
# Sparse autoencoder illustration for images



$$[a_1, \dots, a_{64}] = [0, 0, \dots, 0, \mathbf{0.8}, 0, \dots, 0, \mathbf{0.3}, 0, \dots, 0, \mathbf{0.5}, 0]$$

# Stacking Auto-Encoders

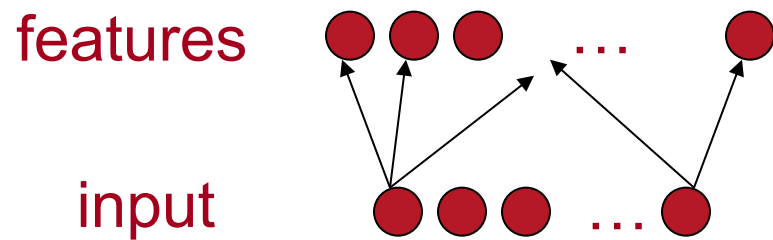
- Can be stacked successfully (Bengio et al NIPS'2006) to form highly non-linear representations



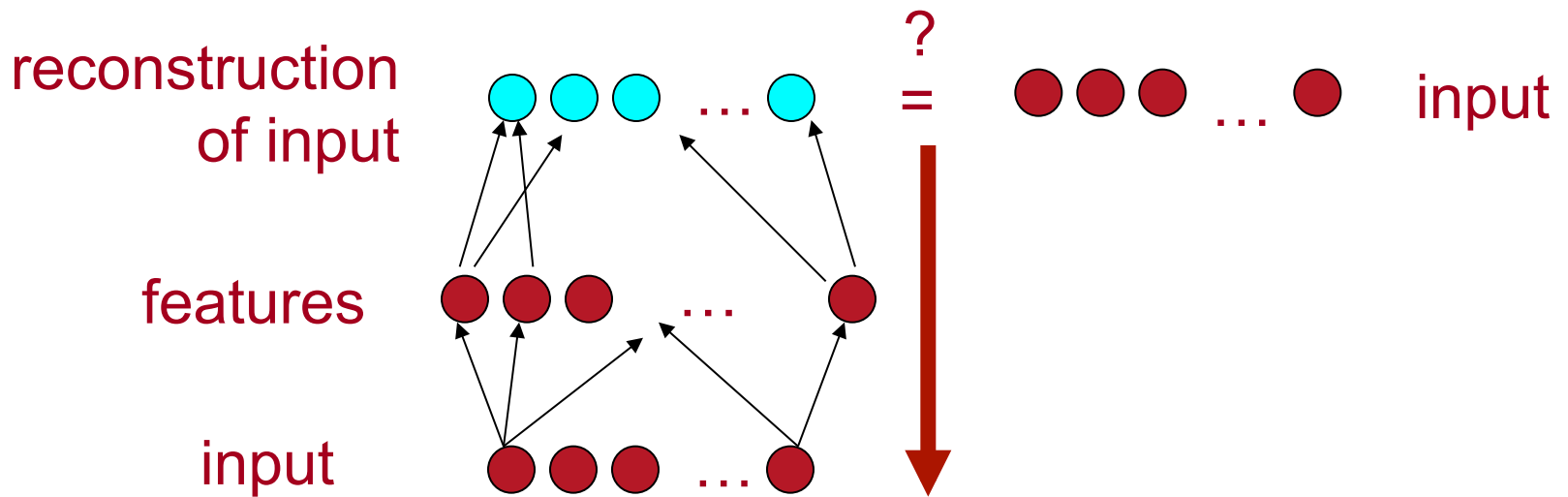
# Layer-wise Unsupervised Learning

input      ● ● ● ... ●

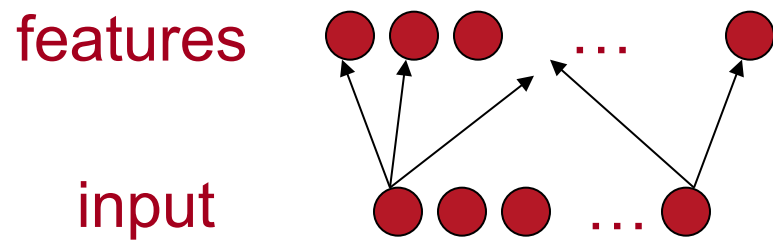
# Layer-wise Unsupervised Pre-training



# Layer-wise Unsupervised Pre-training



# Layer-wise Unsupervised Pre-training

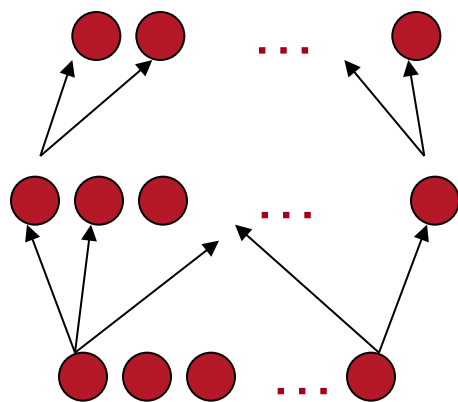


# Layer-wise Unsupervised Pre-training

More abstract  
features

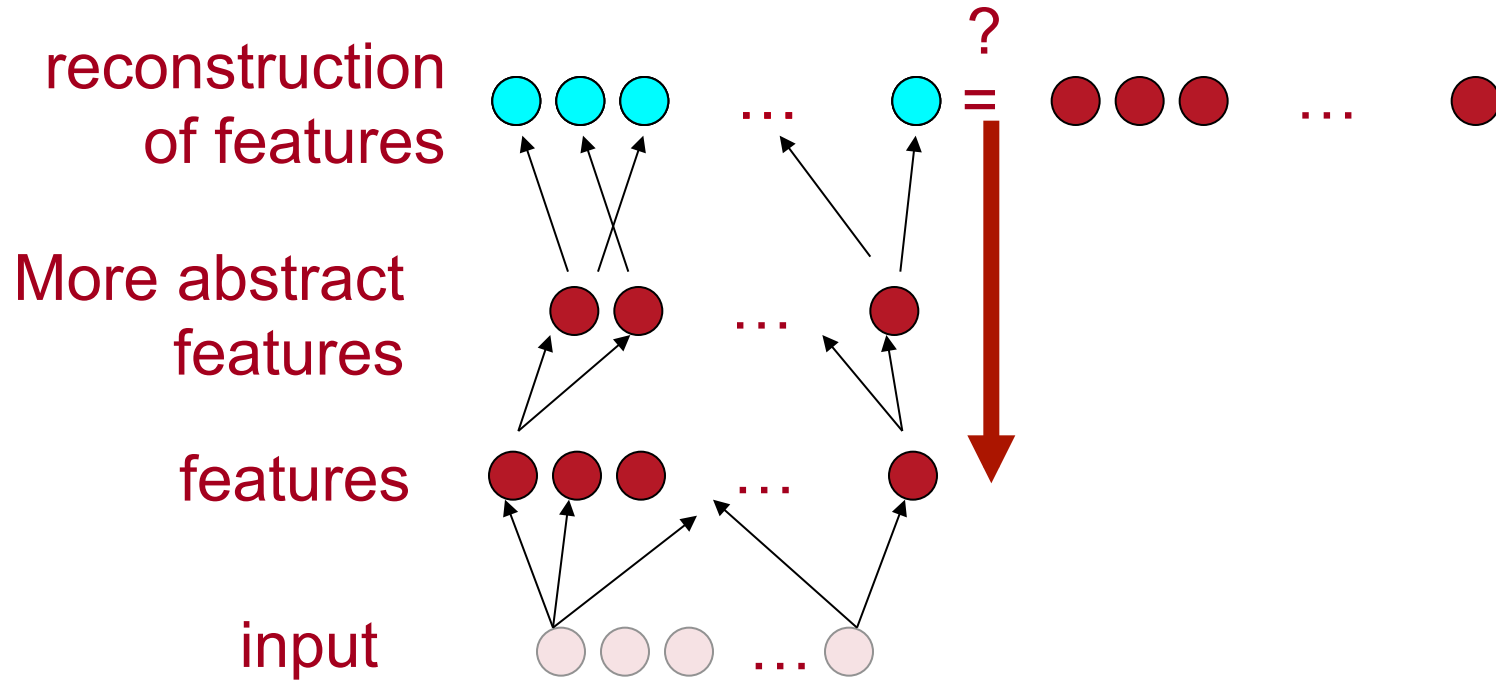
features

input





# Layer-wise Unsupervised Learning

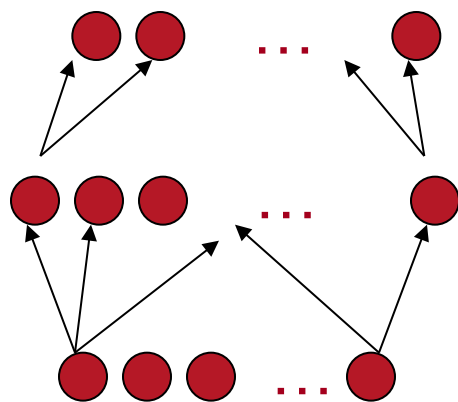


# Layer-wise Unsupervised Pre-training

More abstract  
features

features

input



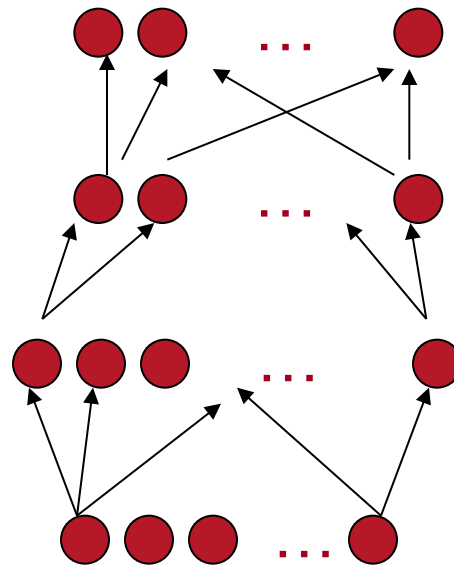
# Layer-wise Unsupervised Learning

Even more abstract  
features

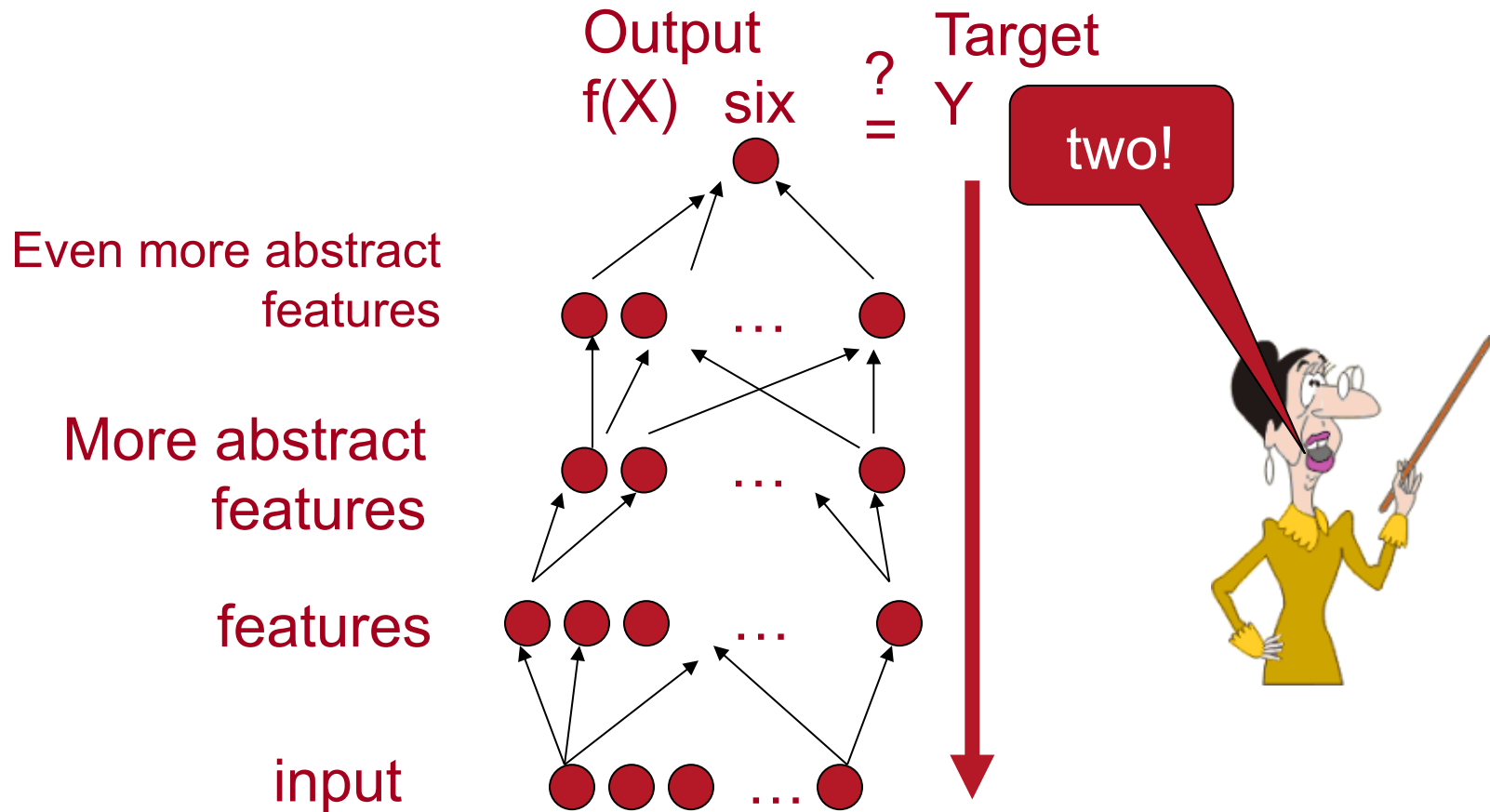
More abstract  
features

features

input

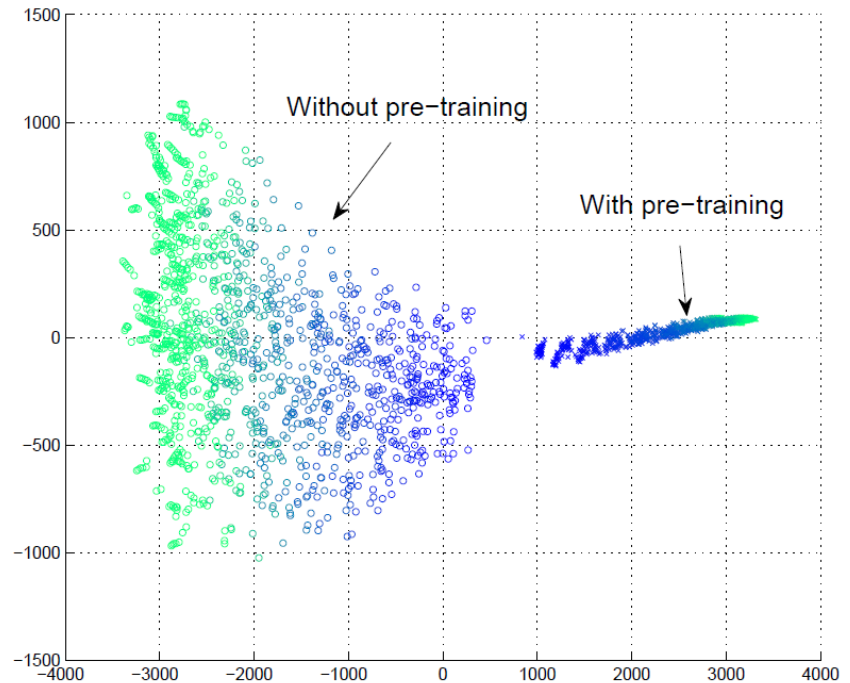


# Supervised Fine-Tuning



# Why is unsupervised pre-training working so well?

- Regularization hypothesis:
  - Representations good for  $P(x)$  are good for  $P(y|x)$
- Optimization hypothesis:
  - Unsupervised initializations start near better local minimum of supervised training error
  - Minima otherwise not achievable by random initialization



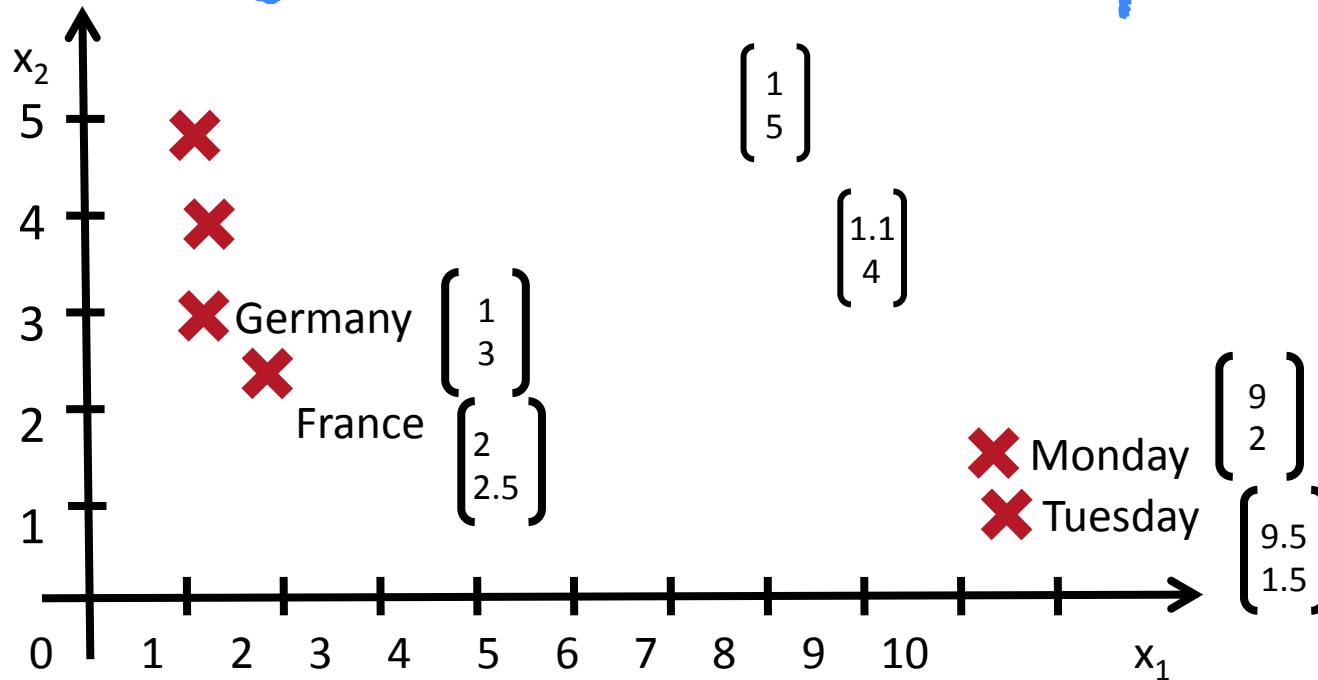
Erhan, Courville, Manzagol,  
Vincent, Bengio (JMLR, 2010)



Part 2

# Recursive Deep Learning

# Building on Word Vector Space Models



the country of my birth  
the place where I was born

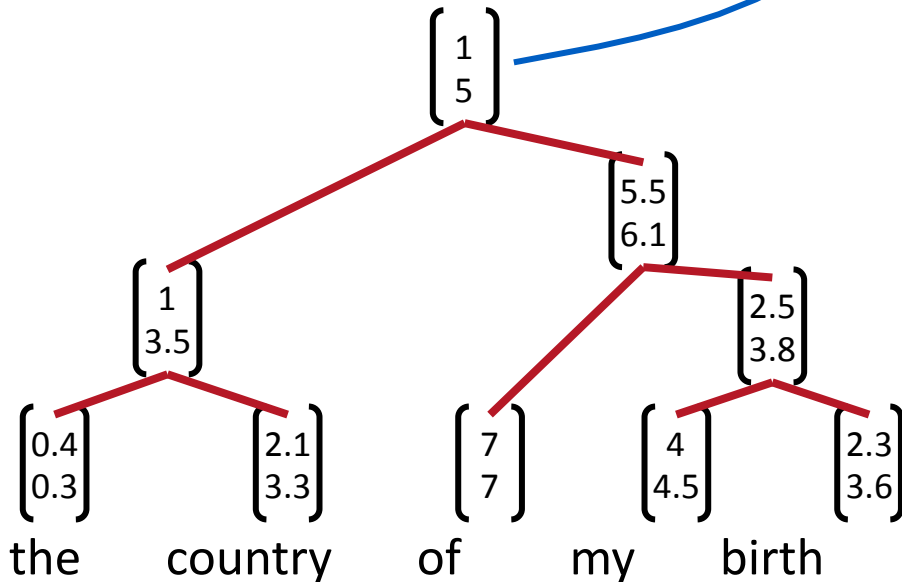
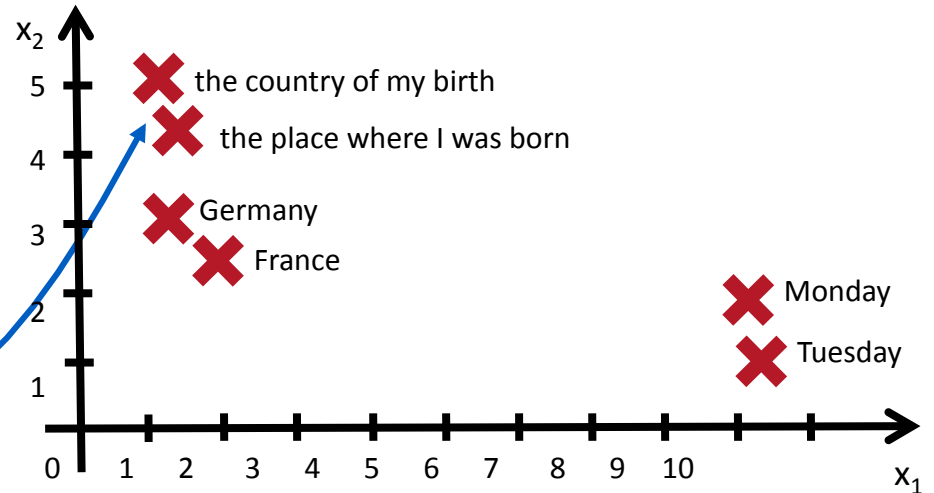
But how can we represent the meaning of longer phrases?

99 By mapping them into the same vector space!

# 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.



Models in this section can jointly learn parse trees and compositional vector representations



# Semantic Vector Spaces

Vectors  
representing  
Phrases and Sentences  
that do not ignore word order  
and capture semantics for NLP tasks



## Single Word Vectors

- Distributional Techniques
- Brown Clusters
- Useful as features inside models, e.g. CRFs for NER, etc.
- Cannot capture longer phrases

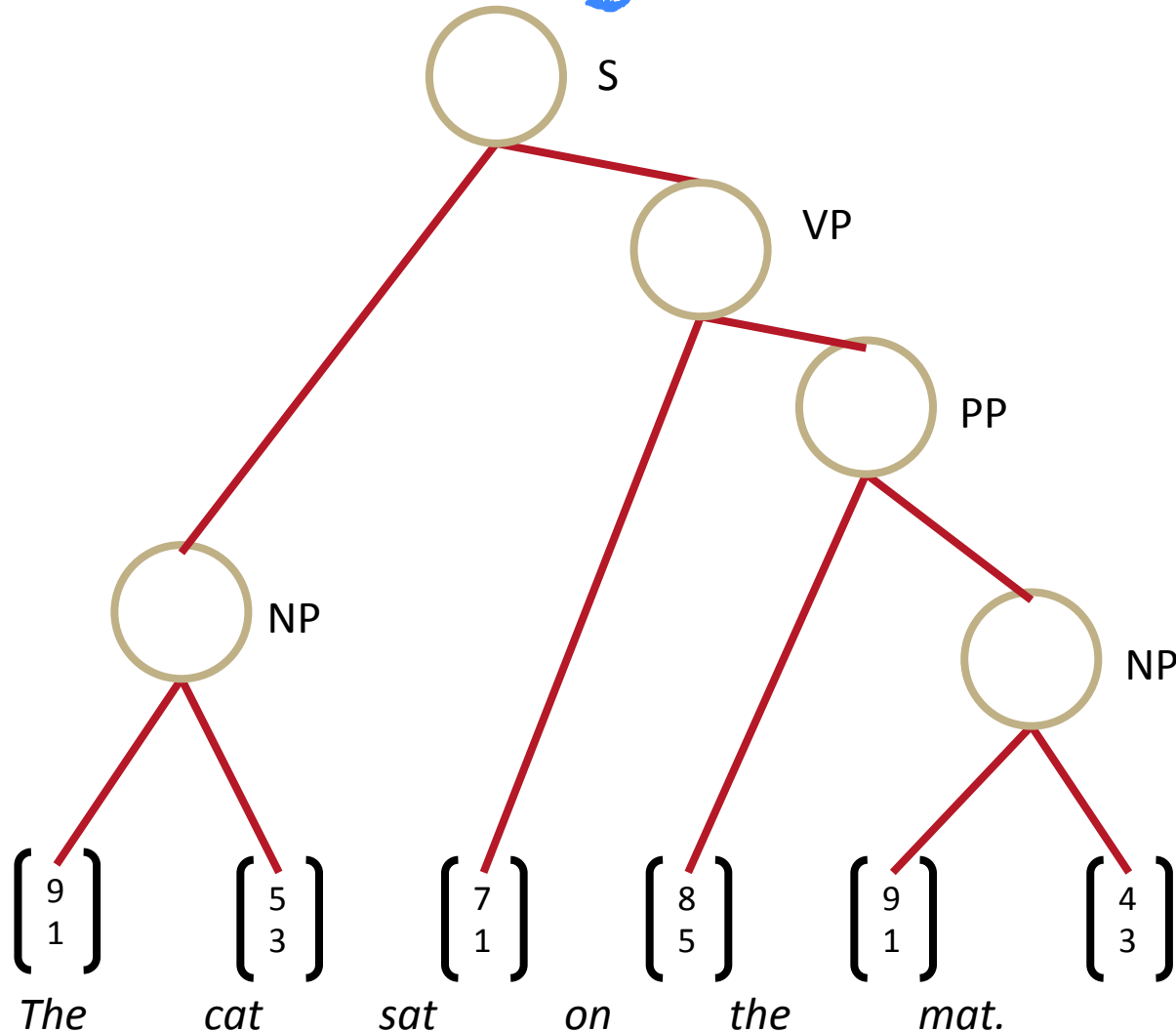
## Documents Vectors

- Bag of words models
- LSA, LDA
- Great for IR, document exploration, etc.
- Ignore word order, no detailed understanding

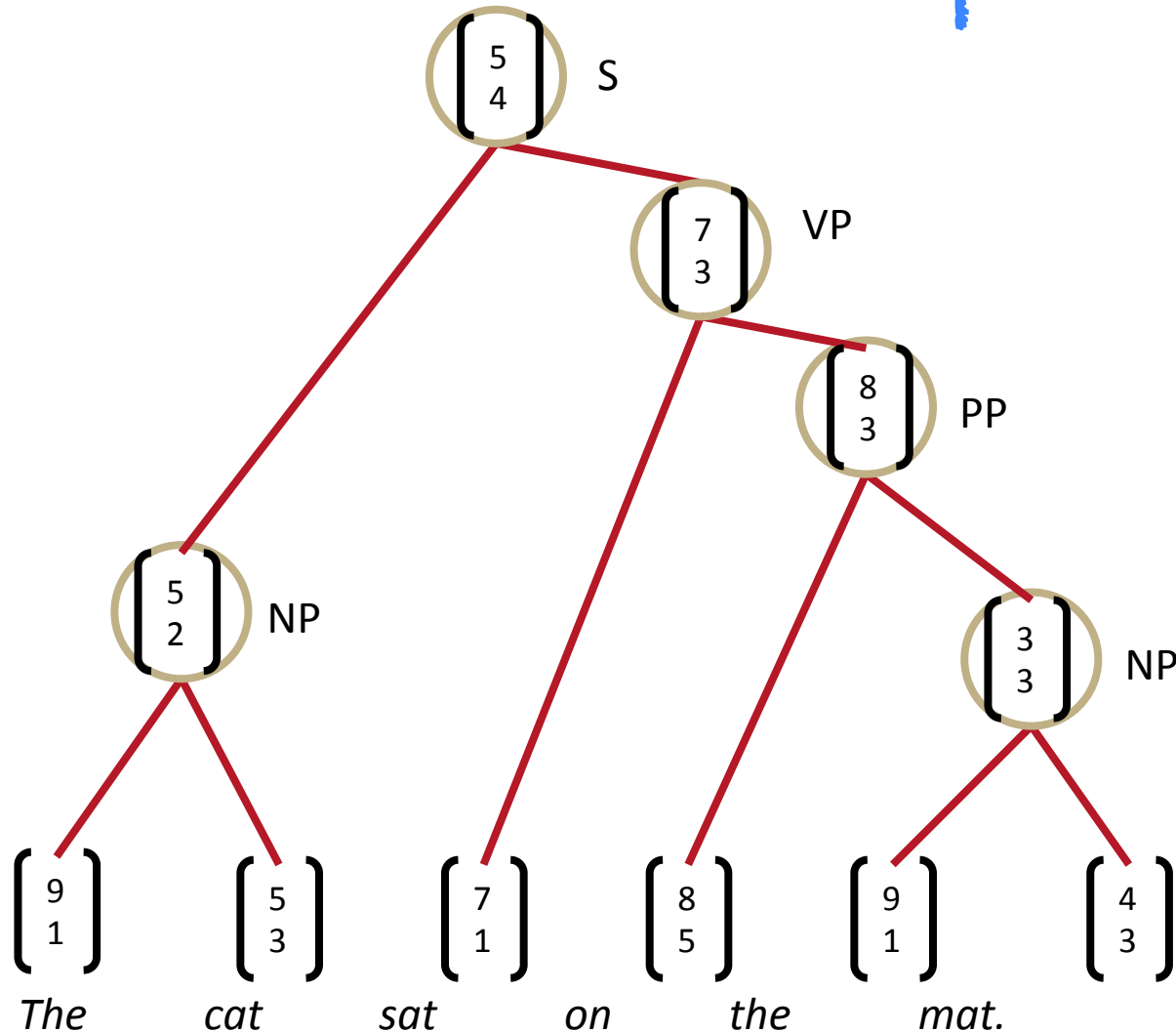
# Recursive Deep Learning

1. Motivation
2. Recursive Neural Networks for Parsing
3. Optimization and Backpropagation Through Structure
4. Compositional Vector Grammars: Parsing
5. Recursive Autoencoders: Paraphrase Detection
6. Matrix-Vector RNNs: Relation classification
7. Recursive Neural Tensor Networks: Sentiment Analysis

# Sentence Parsing: What we want



# Learn Structure and Representation

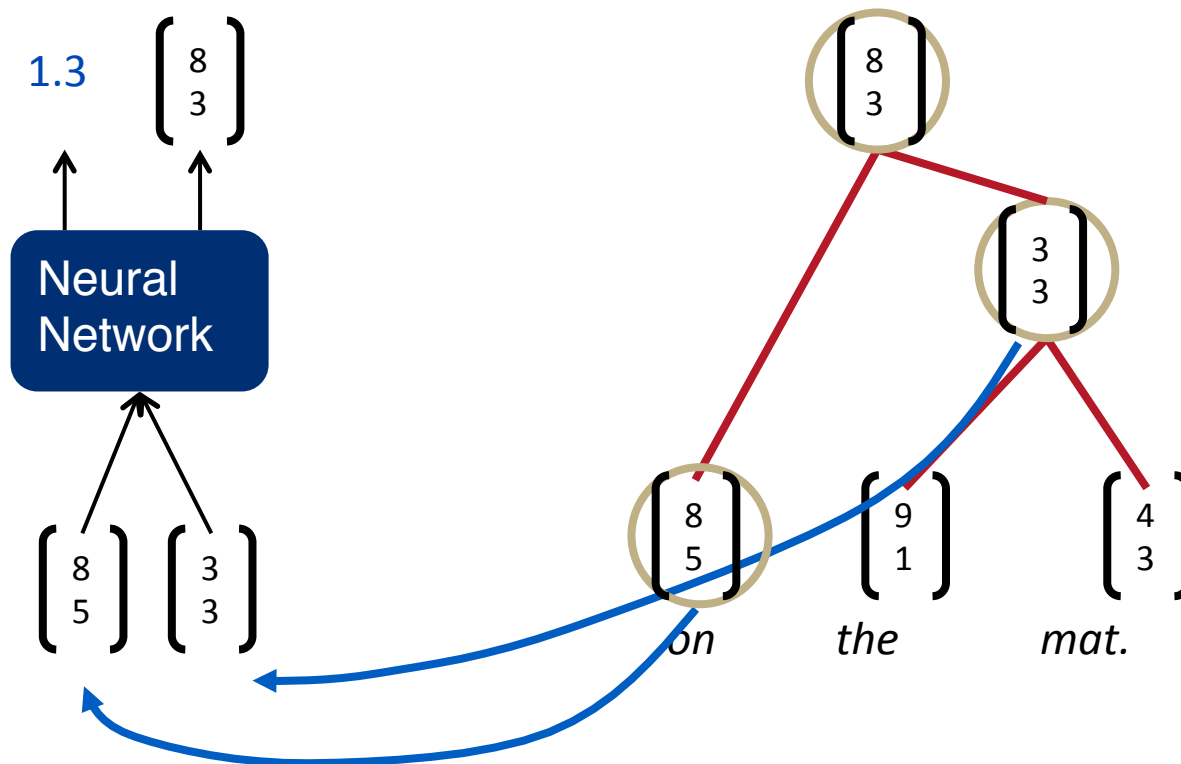


# Recursive Neural Networks for Structure Prediction

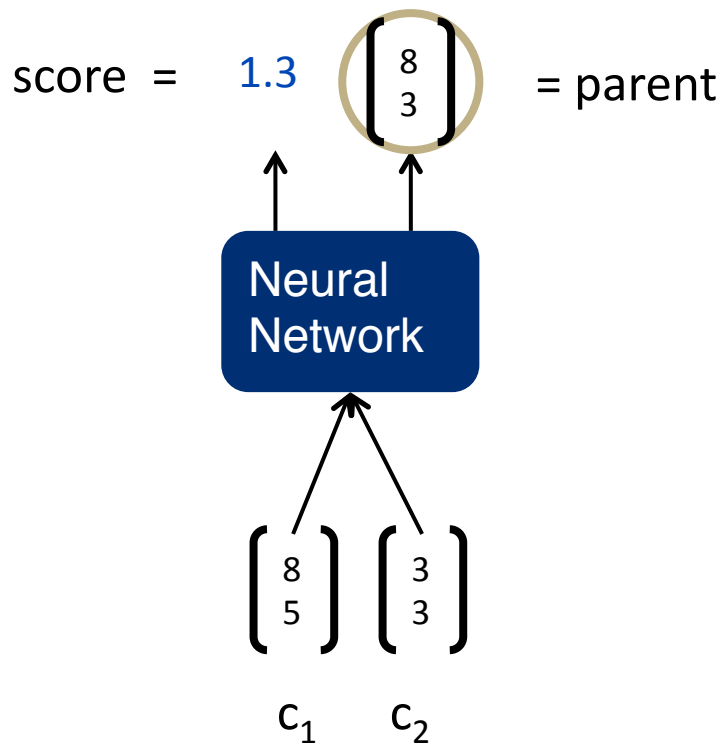
Inputs: two candidate children's representations

Outputs:

1. The semantic representation if the two nodes are merged.
2. Score of how plausible the new node would be.



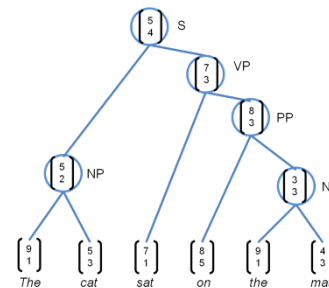
# Recursive Neural Network Definition



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

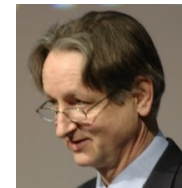
$$p = \tanh\left(W \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + b\right),$$

Same  $W$  parameters at all nodes of the tree

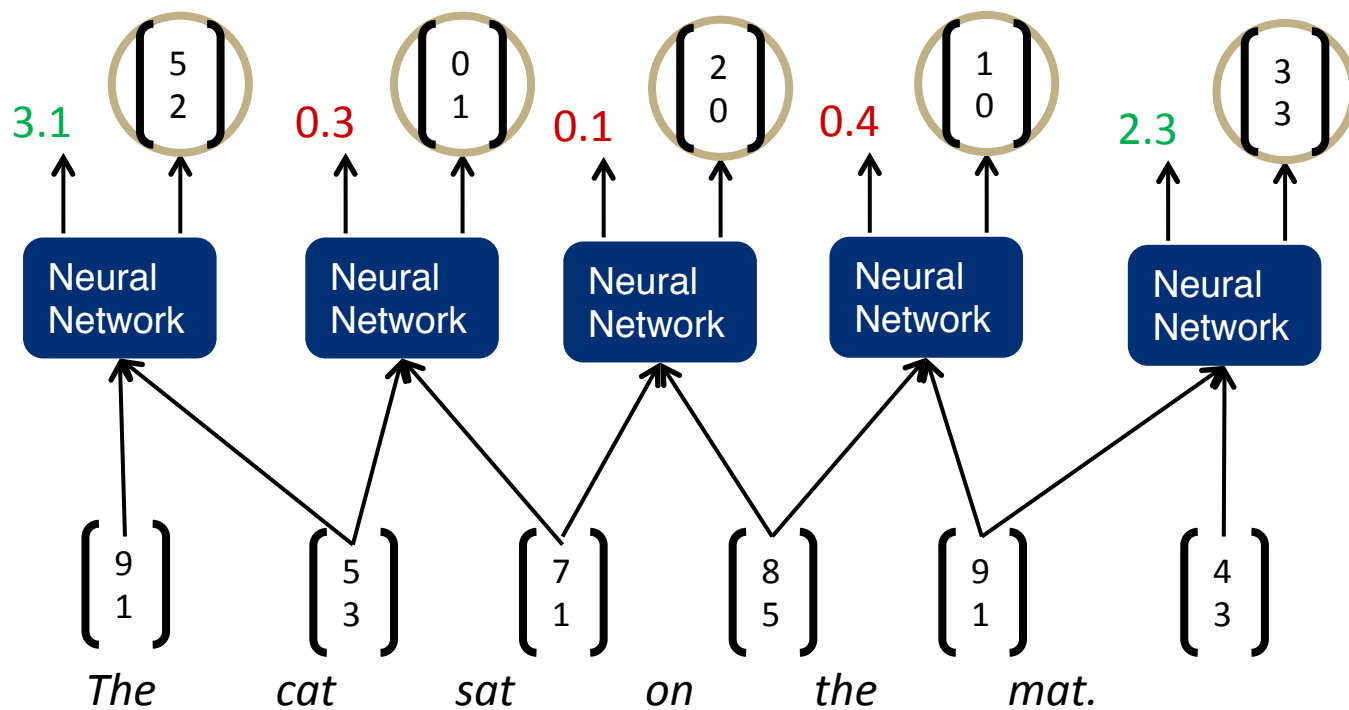


# Related Work to Socher et al. (ICML 2011)

- Pollack (1990): Recursive auto-associative memories
- Previous Recursive Neural Networks work by Goller & Küchler (1996), Costa et al. (2003) assumed fixed tree structure and used one hot vectors.
- Hinton (1990) and Bottou (2011): Related ideas about recursive models and recursive operators as smooth versions of logic operations

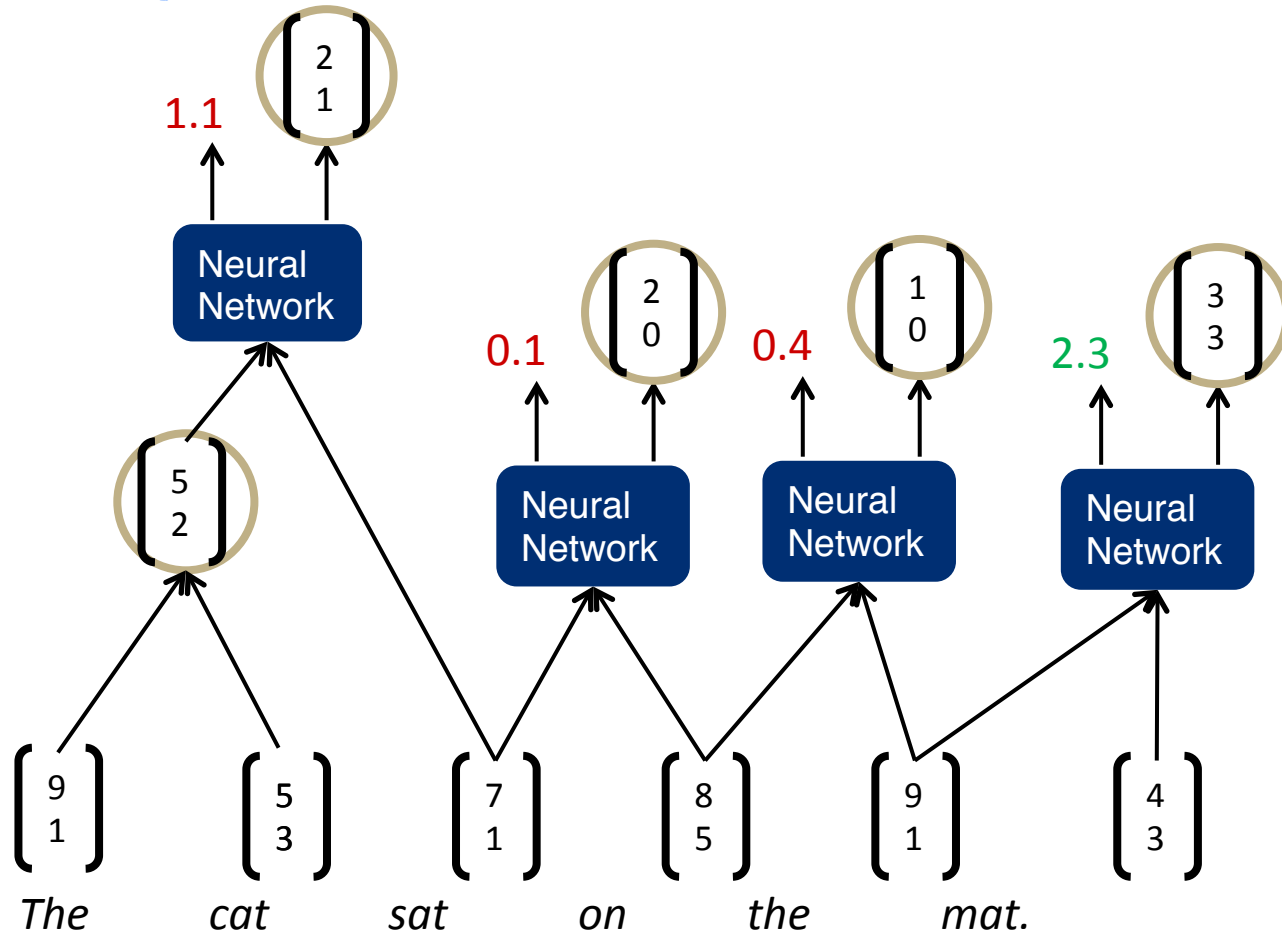


# Parsing a sentence with an RNN

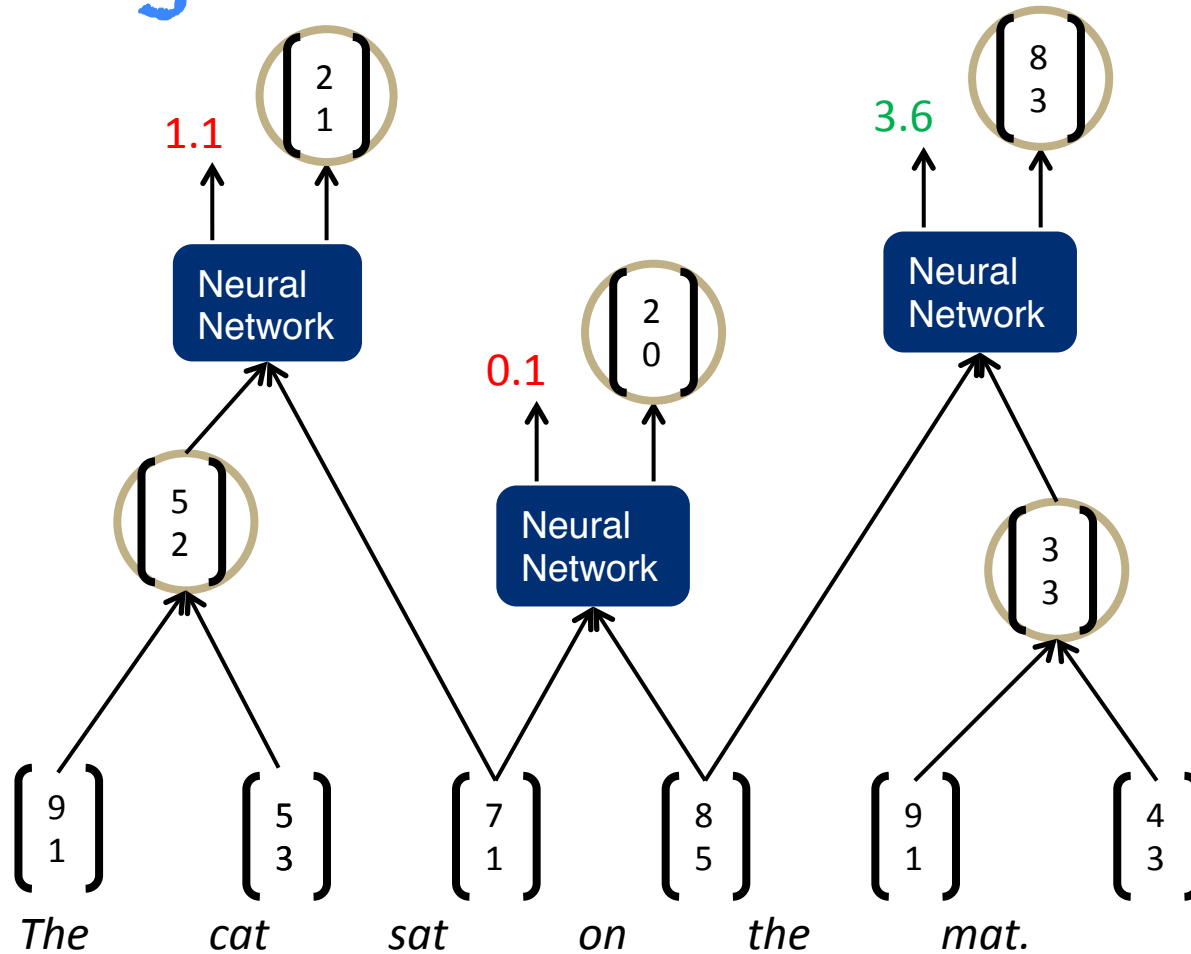




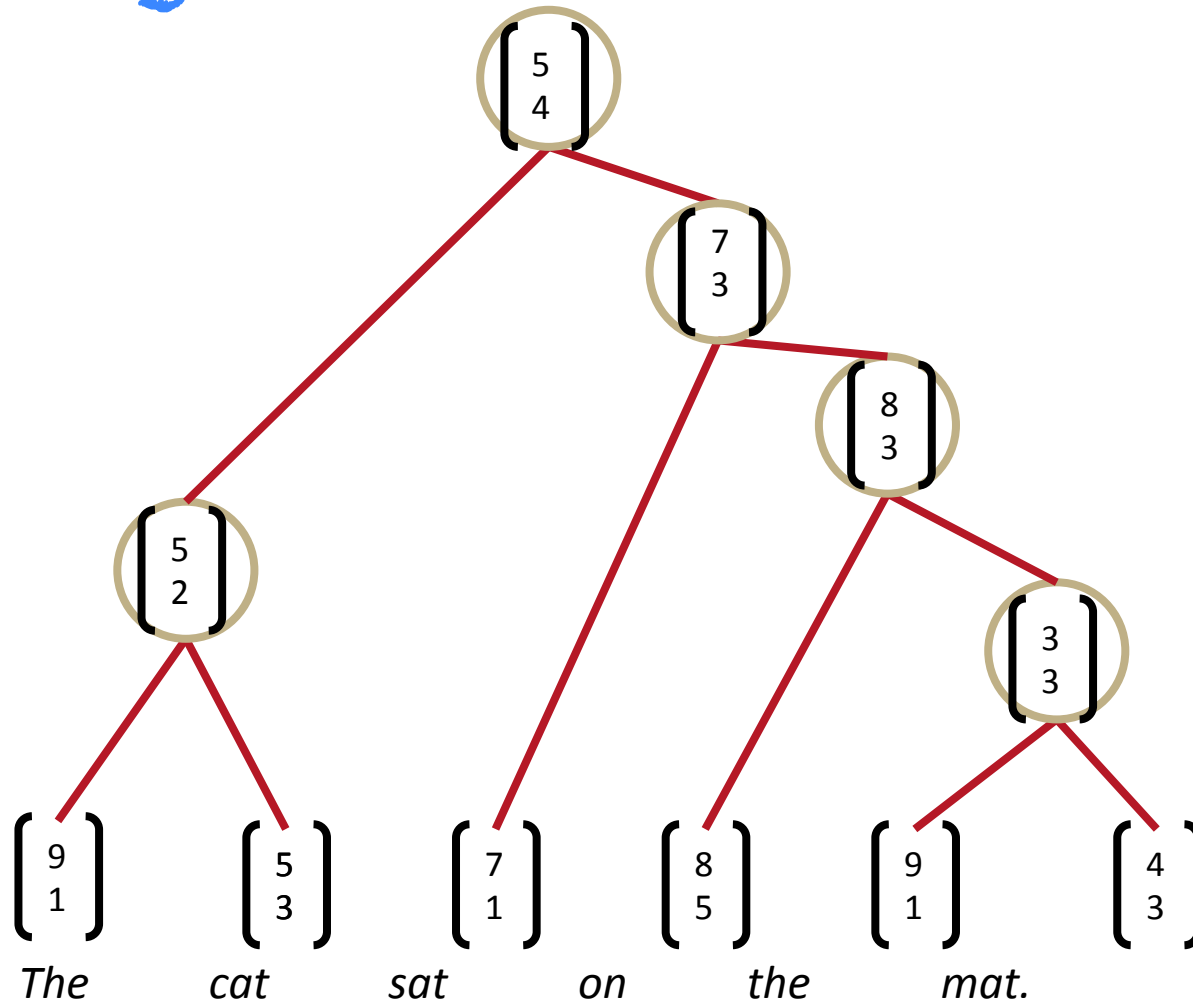
# Parsing a sentence



# Parsing a sentence



# Parsing a sentence



# Max-Margin Framework - Details

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



- Similar to max-margin parsing (Taskar et al. 2004), a supervised max-margin objective

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

- The loss  $\Delta(y, y_i)$  penalizes all incorrect decisions
- Structure search for  $A(x)$  was maximally greedy
  - Instead: Beam Search with Chart

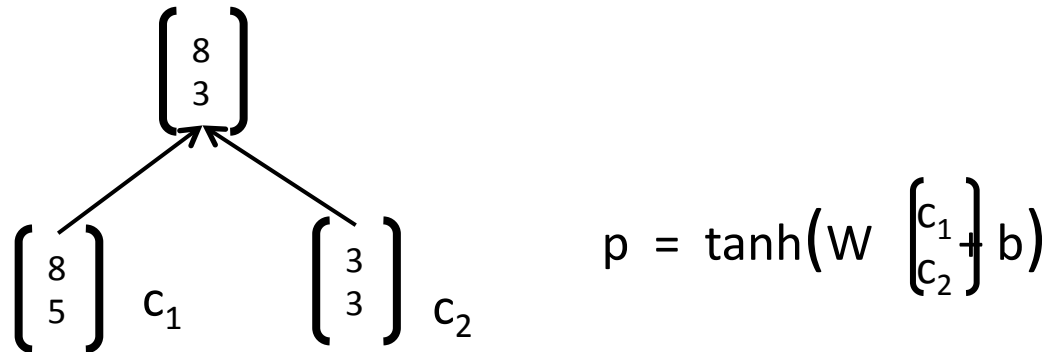
# Backpropagation Through Structure



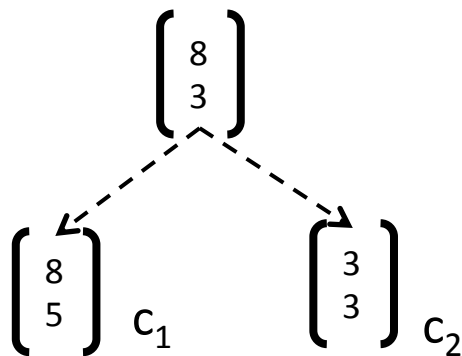
- Introduced by Goller & Küchler (1996)
- Principally the same as general backpropagation
- Two differences resulting from the tree structure:
  - Split derivatives at each node
  - Sum derivatives of  $W$  from all nodes

# BTS: Split derivatives at each node

- During forward prop, the parent is computed using 2 children



- Hence, the errors need to be computed wrt each of them:



where each child's error is n-dimensional

# BTS: Sum derivatives of all nodes

- You can actually assume it's a different  $W$  at each node
- Intuition via example:

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

- If take separate derivatives of each occurrence, we get same:

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

# BTS: Optimization

- As before, we can plug the gradients into a standard off-the-shelf L-BFGS optimizer
- Best results with AdaGrad (Duchi et al, 2011):

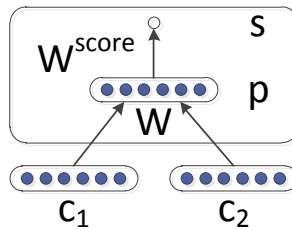
$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^t g_{\tau,i}^2}} g_{t,i}$$

- For non-continuous objective use subgradient *method* (Ratliff et al. 2007)



# Discussion: Simple RNN

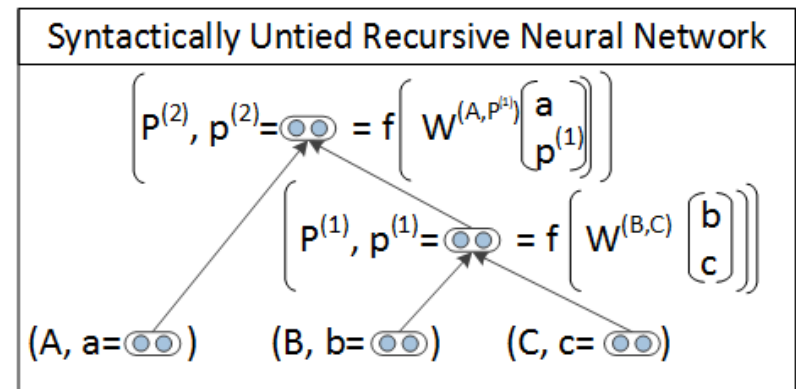
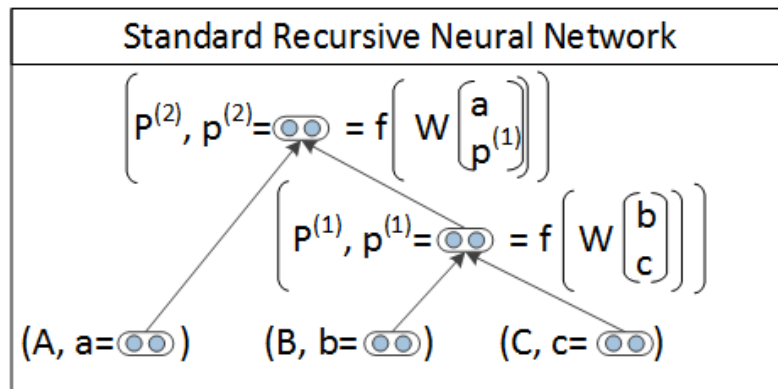
- Good results with single matrix RNN (more later)
- Single weight matrix RNN could capture some phenomena but not adequate for more complex, higher order composition and parsing long sentences



- The composition function is the same for all syntactic categories, punctuation, etc

# Solution: Syntactically-Untied RNN

- Idea: Condition the composition function on the syntactic categories, “untie the weights”
- Allows for different composition functions for pairs of syntactic categories, e.g. Adv + AdjP, VP + NP
- Combines discrete syntactic categories with continuous semantic information



# Solution: CVG = PCFG + Syntactically-Untied RNN

- Problem: Speed. Every candidate score in beam search needs a matrix-vector product.
- Solution: Compute score using a linear combination of the log-likelihood from a simple PCFG + RNN
  - Prunes very unlikely candidates for speed
  - Provides coarse syntactic categories of the children for each beam candidate
- Compositional Vector Grammars:  $CVG = PCFG + RNN$

# Details: Compositional Vector Grammar

- Scores at each node 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:

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

- Socher et al (2013): More details at ACL

# Related Work

- Resulting CVG Parser is related to previous work that extends PCFG parsers
- Klein and Manning (2003a) : manual feature engineering
- Petrov et al. (2006) : learning algorithm that splits and merges syntactic categories
- Lexicalized parsers (Collins, 2003; Charniak, 2000): describe each category with a lexical item
- Hall and Klein (2012) combine several such annotation schemes in a factored parser.
- CVGs extend these ideas from discrete representations to richer continuous ones
- Hermann & Blunsom (2013): Combine Combinatory Categorical Grammars with RNNs and also untie weights, see upcoming ACL 2013

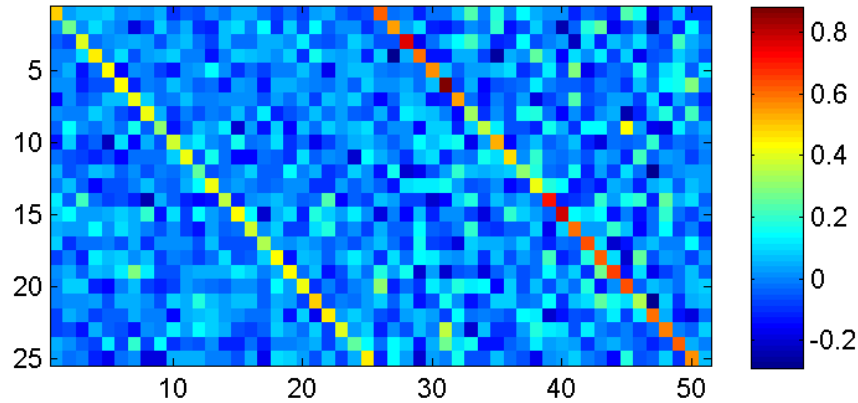
# Experiments

- 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 parser

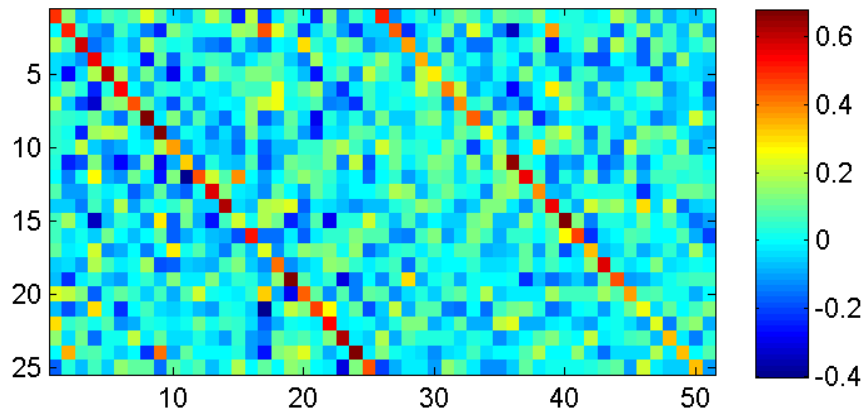
Parser	Test, All Sentences
Stanford PCFG, (Klein and Manning, 2003a)	85.5
Stanford Factored (Klein and Manning, 2003b)	86.6
Factored PCFGs (Hall and Klein, 2012)	89.4
Collins (Collins, 1997)	87.7
SSN (Henderson, 2004)	89.4
Berkeley Parser (Petrov and Klein, 2007)	90.1
CVG (RNN) (Socher et al., ACL 2013)	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 Analysis

- Learns notion of soft head words



DT-NP



VP-NP

# Analysis of resulting vector representations

All the figures are adjusted for seasonal variations

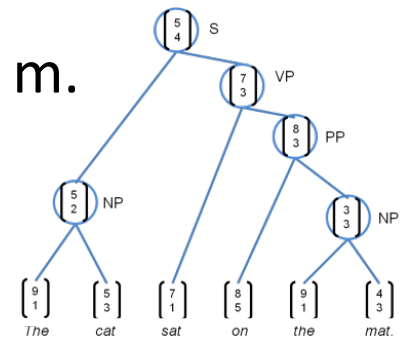
1. All the numbers are adjusted for seasonal fluctuations
2. All the figures are adjusted to remove usual seasonal patterns

Knight-Ridder wouldn't comment on the offer

1. Harsco declined to say what country placed the order
2. Coastal wouldn't disclose the terms

Sales grew almost 7% to \$UNK m. from \$UNK m.

1. Sales rose more than 7% to \$94.9 m. from \$88.3 m.
2. Sales surged 40% to UNK b. yen from UNK b.



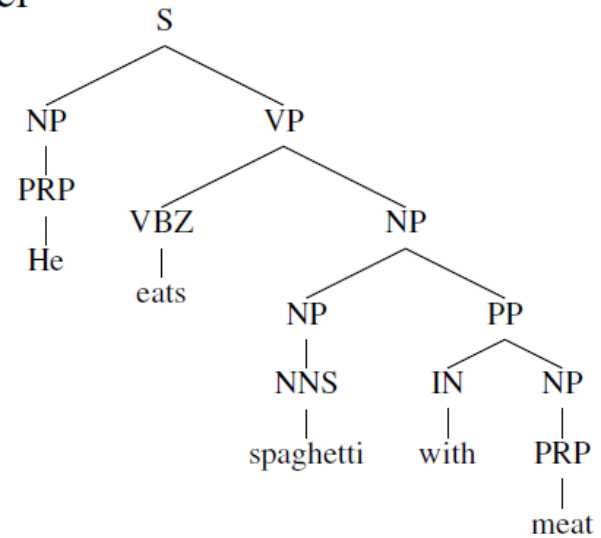
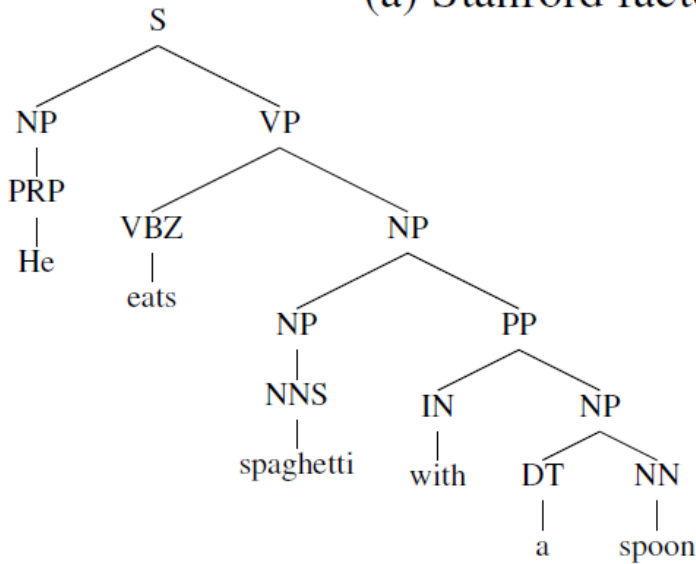


# SU-RNN Analysis

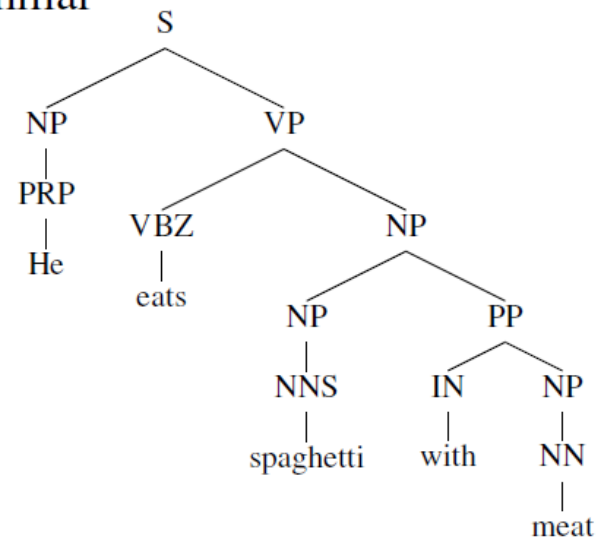
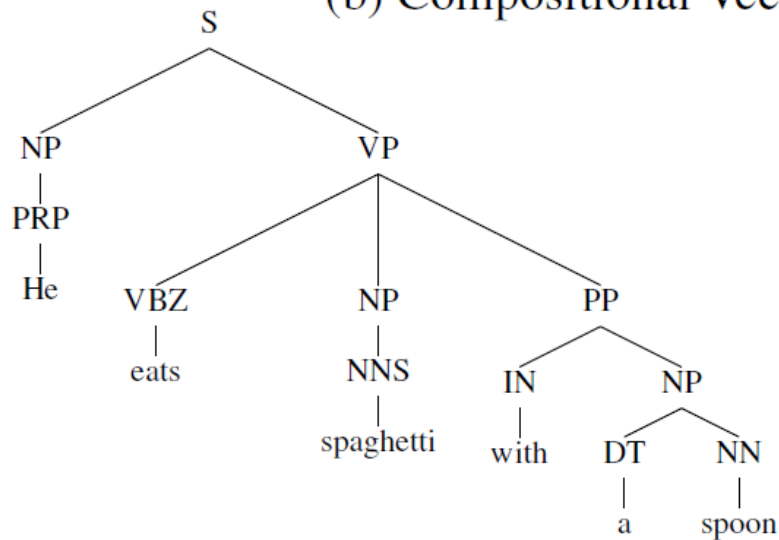
- Can transfer semantic information from single related example
- Train sentences:
  - He eats spaghetti with a fork.
  - She eats spaghetti with pork.
- Test sentences
  - He eats spaghetti with a spoon.
  - He eats spaghetti with meat.

# SU-RNN Analysis

(a) Stanford factored parser



(b) Compositional Vector Grammar

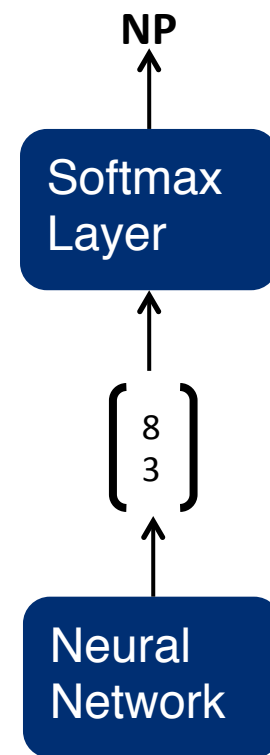


# Labeling in Recursive Neural Networks

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

$$p(c|p) = \textit{softmax}(Sp)$$

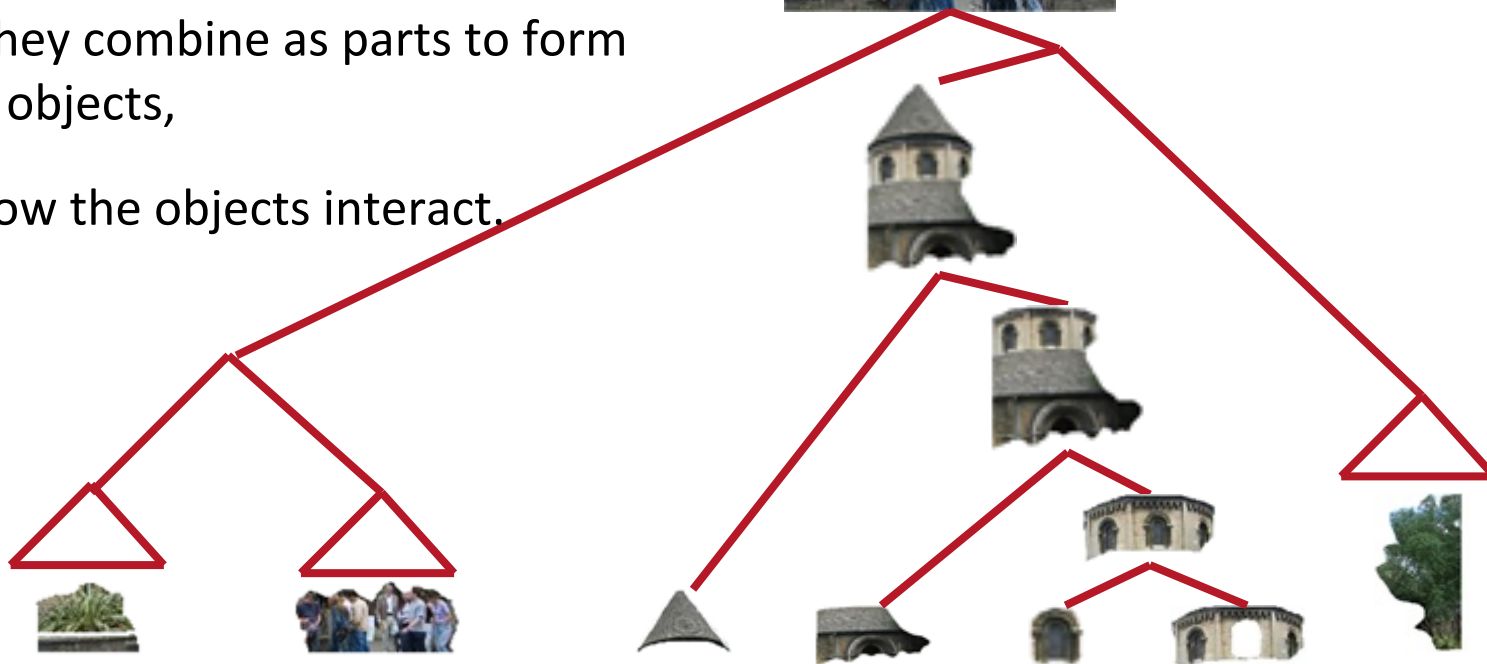
- Training similar to model in part 1 with standard cross-entropy error + scores



# Scene Parsing

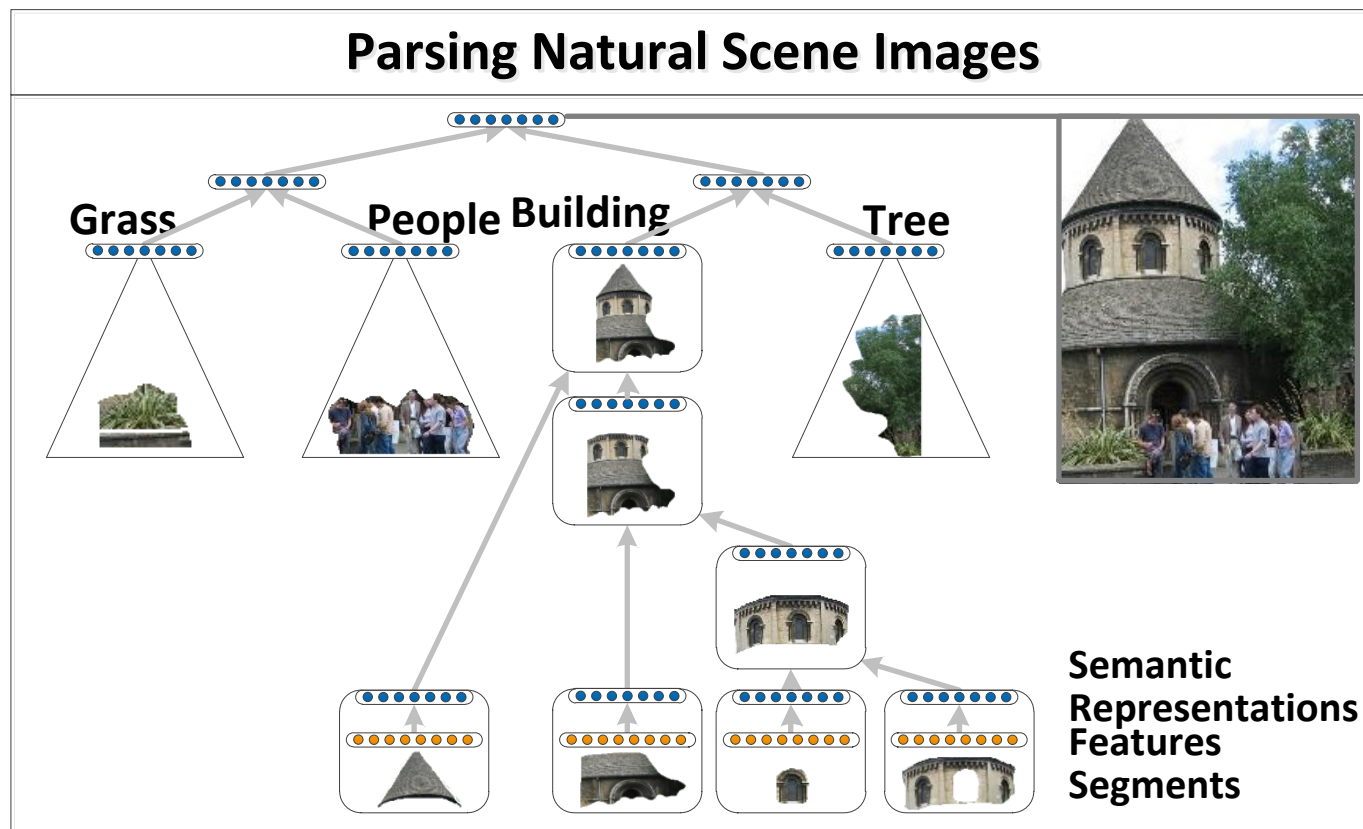
Similar principle of compositionality.

- The meaning of a scene image is also a function of smaller regions,
- how they combine as parts to form larger objects,
- and how the objects interact.



# Algorithm for Parsing Images

Same Recursive Neural Network as for natural language parsing!  
(Socher et al. ICML 2011)



# Multi-class segmentation



■ sky ■ tree ■ road ■ grass ■ water ■ bldg ■ mntn ■ fg obj.

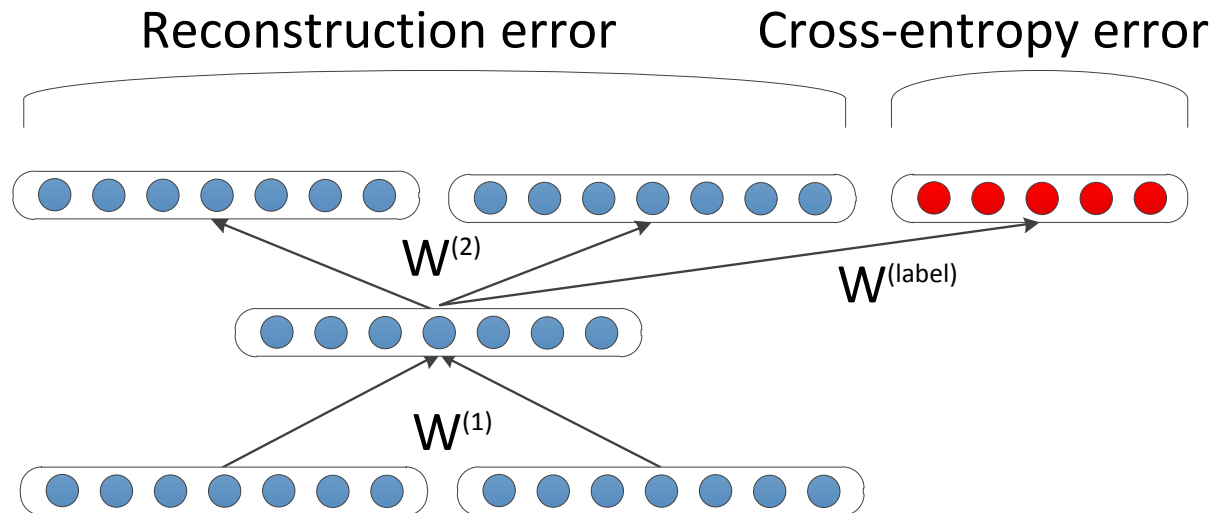
Method	Accuracy
Pixel CRF (Gould et al., ICCV 2009)	74.3
Classifier on superpixel features	75.9
Region-based energy (Gould et al., ICCV 2009)	76.4
Local labelling (Tighe & Lazebnik, ECCV 2010)	76.9
Superpixel MRF (Tighe & Lazebnik, ECCV 2010)	77.5
Simultaneous MRF (Tighe & Lazebnik, ECCV 2010)	77.5
Recursive Neural Network	<b>78.1</b>

# Recursive Deep Learning

1. Motivation
2. Recursive Neural Networks for Parsing
3. Theory: Backpropagation Through Structure
4. Compositional Vector Grammars: Parsing
5. Recursive Autoencoders: Paraphrase Detection
6. Matrix-Vector RNNs: Relation classification
7. Recursive Neural Tensor Networks: Sentiment Analysis

# Semi-supervised Recursive Autoencoder

- To capture sentiment and solve antonym problem, add a softmax classifier
- Error is a weighted combination of reconstruction error and cross-entropy
- Socher et al. (EMNLP 2011)





# Paraphrase Detection

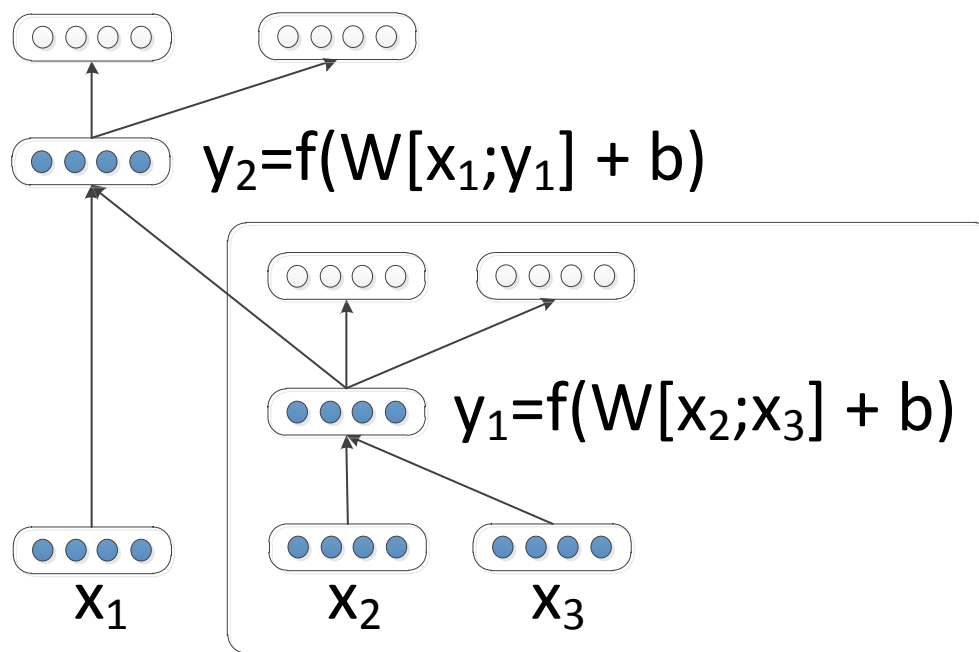
- Pollack said the plaintiffs failed to show that Merrill and Blodget directly caused their losses
- Basically , the plaintiffs did not show that omissions in Merrill's research caused the claimed losses
- The initial report was made to Modesto Police December 28
- It stems from a Modesto police report

How to compare  
the meaning  
of two sentences?

# Unsupervised Recursive Autoencoders

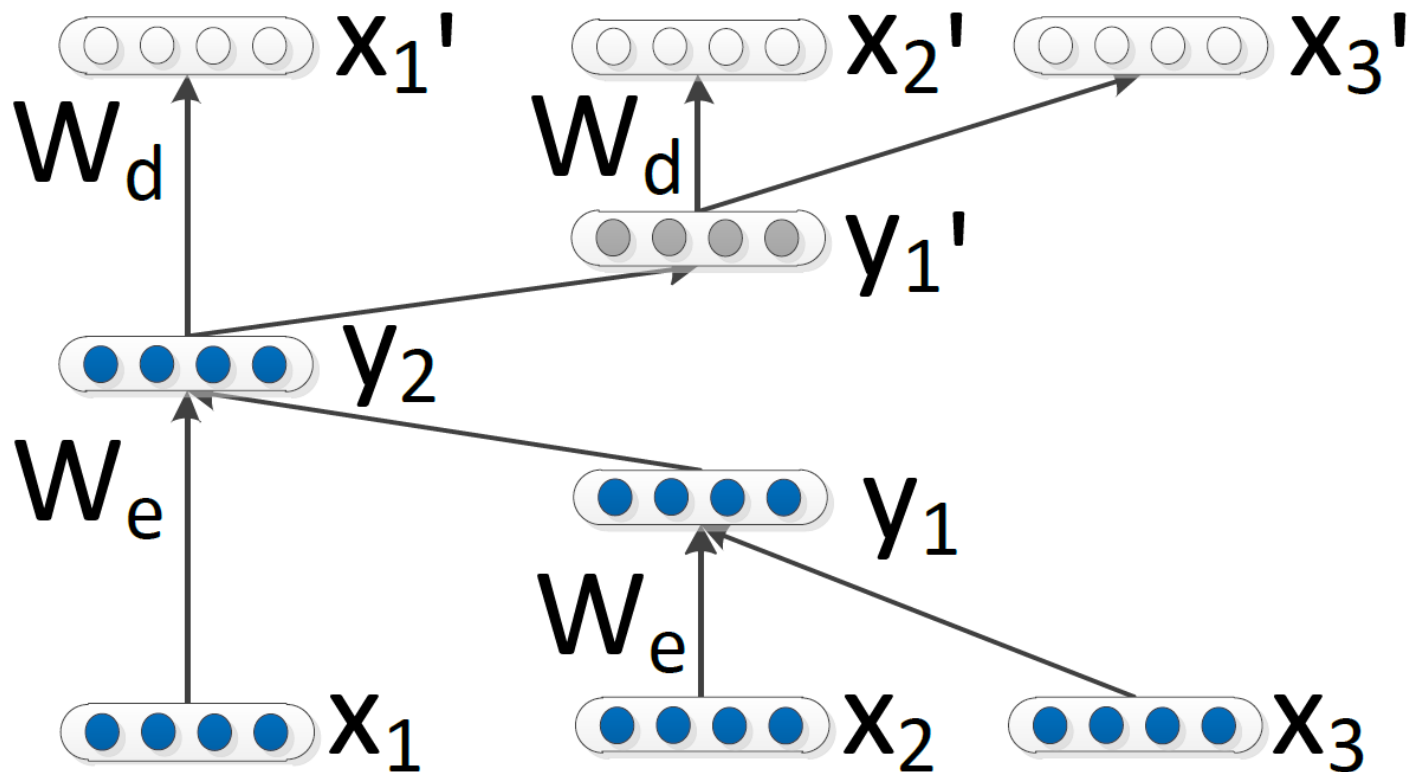
- Similar to Recursive Neural Net but instead of a supervised score we compute a reconstruction error at each node. Socher et al. (EMNLP 2011)

$$E_{rec}([c_1; c_2]) = \frac{1}{2} \left\| [c_1; c_2] - [c'_1; c'_2] \right\|^2$$



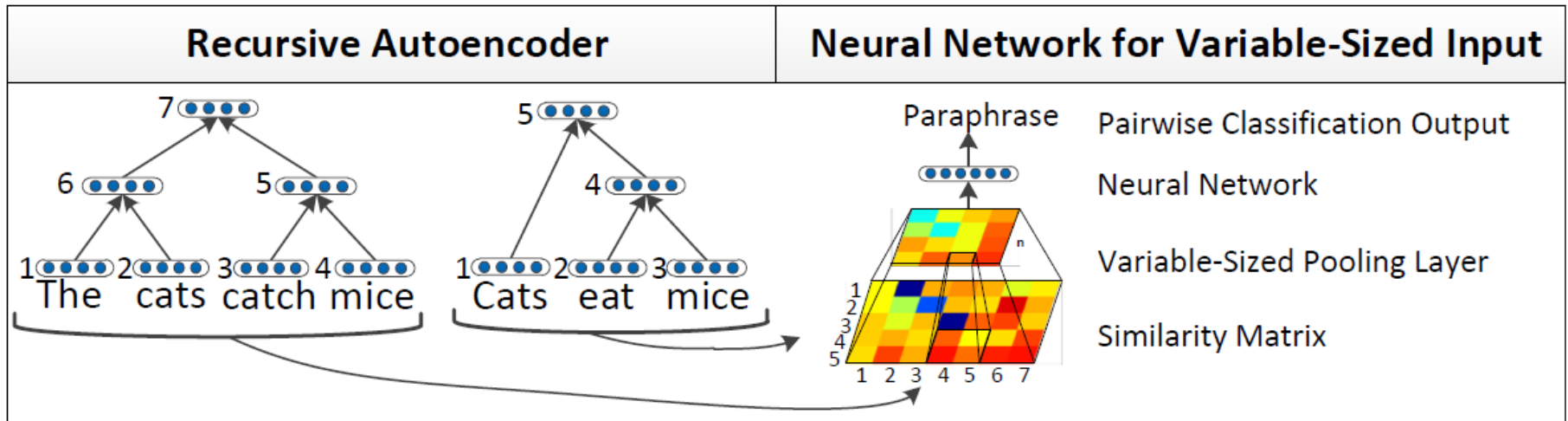
# Unsupervised unfolding RAE

- Attempt to encode entire tree structure at each node



# Recursive Autoencoders for Full Sentence Paraphrase Detection

- Unsupervised Unfolding RAE and a pair-wise sentence comparison of nodes in parsed trees
- Socher et al. (NIPS 2011)



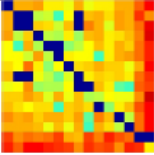
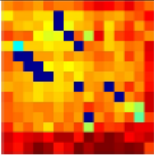
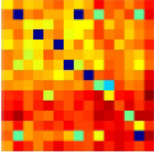
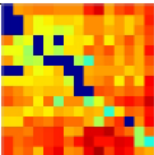
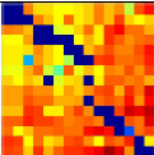
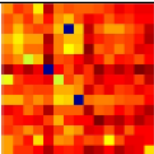
# Recursive Autoencoders for Full Sentence Paraphrase Detection

- Experiments on Microsoft Research Paraphrase Corpus
- (Dolan et al. 2004)

Method	Acc.	F1
Rus et al.(2008)	70.6	80.5
Mihalcea et al.(2006)	70.3	81.3
Islam et al.(2007)	72.6	81.3
Qiu et al.(2006)	72.0	81.6
Fernando et al.(2008)	74.1	82.4
Wan et al.(2006)	75.6	83.0
Das and Smith (2009)	73.9	82.3
Das and Smith (2009) + 18 Surface Features	76.1	82.7
F. Bu et al. (ACL 2012): String Re-writing Kernel	76.3	--
Unfolding Recursive Autoencoder (NIPS 2011)	<b>76.8</b>	<b>83.6</b>



# Recursive Autoencoders for Full Sentence Paraphrase Detection

L	Pr	Sentences	Sim.Mat.
P	0.95	(1) LLEYTON Hewitt yesterday traded his tennis racquet for his first sporting passion - Australian football - as the world champion relaxed before his Wimbledon title defence (2) LLEYTON Hewitt yesterday traded his tennis racquet for his first sporting passion-Australian rules football-as the world champion relaxed ahead of his Wimbledon defence	
P	0.82	(1) The lies and deceptions from Saddam have been well documented over 12 years (2) It has been well documented over 12 years of lies and deception from Saddam	
P	0.67	(1) Pollack said the plaintiffs failed to show that Merrill and Blodget directly caused their losses (2) Basically , the plaintiffs did not show that omissions in Merrill's research caused the claimed losses	
N	0.49	(1) Prof Sally Baldwin, 63, from York, fell into a cavity which opened up when the structure collapsed at Tiburtina station, Italian railway officials said (2) Sally Baldwin, from York, was killed instantly when a walkway collapsed and she fell into the machinery at Tiburtina station	
N	0.44	(1) Bremer, 61, is a onetime assistant to former Secretaries of State William P. Rogers and Henry Kissinger and was ambassador-at-large for counterterrorism from 1986 to 1989 (2) Bremer, 61, is a former assistant to former Secretaries of State William P. Rogers and Henry Kissinger	
N	0.11	(1) The initial report was made to Modesto Police December 28 (2) It stems from a Modesto police report	

# Recursive Deep Learning

1. Motivation
2. Recursive Neural Networks for Parsing
3. Theory: Backpropagation Through Structure
4. Compositional Vector Grammars: Parsing
5. Recursive Autoencoders: Paraphrase Detection
6. Matrix-Vector RNNs: Relation classification
7. Recursive Neural Tensor Networks: Sentiment Analysis



# Compositionality Through Recursive Matrix-Vector Spaces

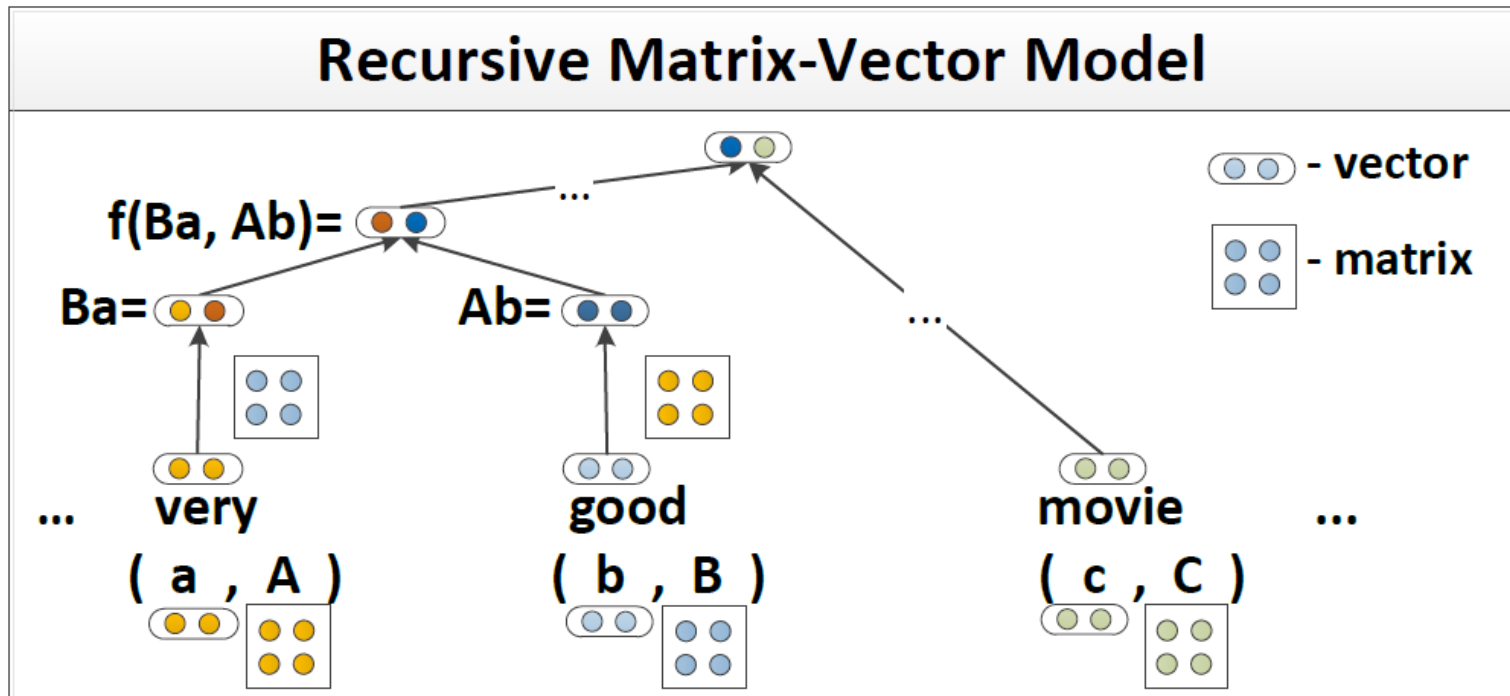
$$p = \tanh\left(W \begin{bmatrix} c_1 \\ + \\ c_2 \end{bmatrix} b\right)$$

- 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
- Proposal: A new composition function

# Compositionality Through Recursive Matrix-Vector Recursive Neural Networks

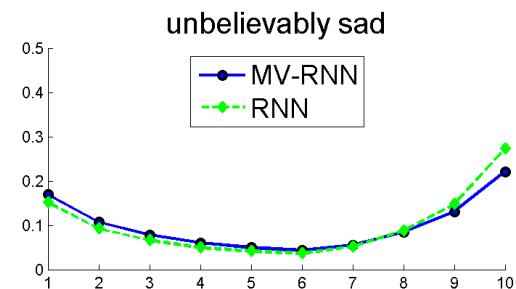
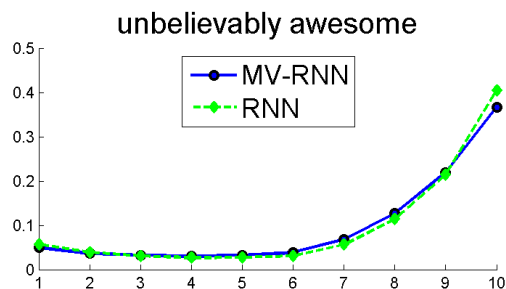
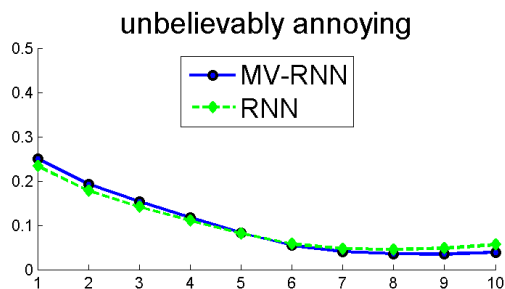
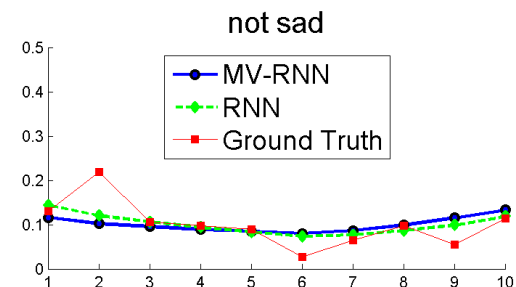
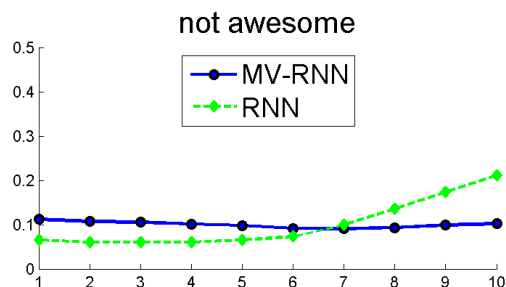
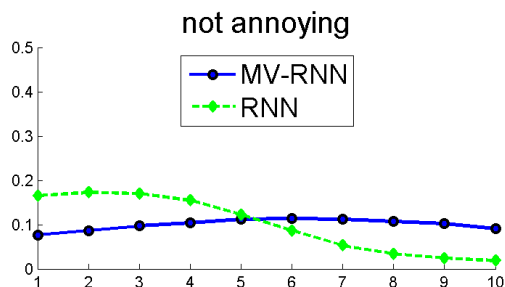
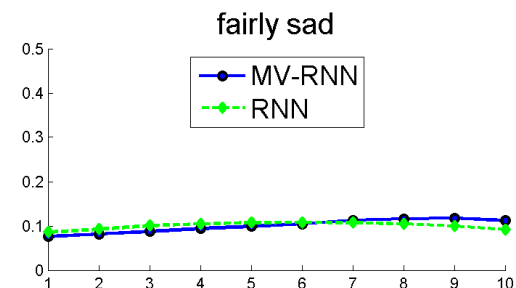
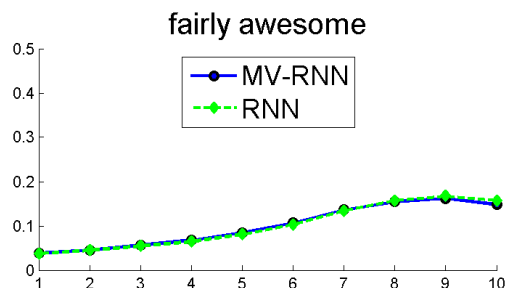
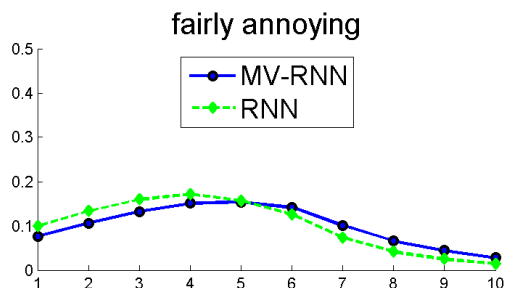
$$p = \tanh\left(W \begin{pmatrix} c_1 \\ + \\ c_2 \end{pmatrix} b\right)$$

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

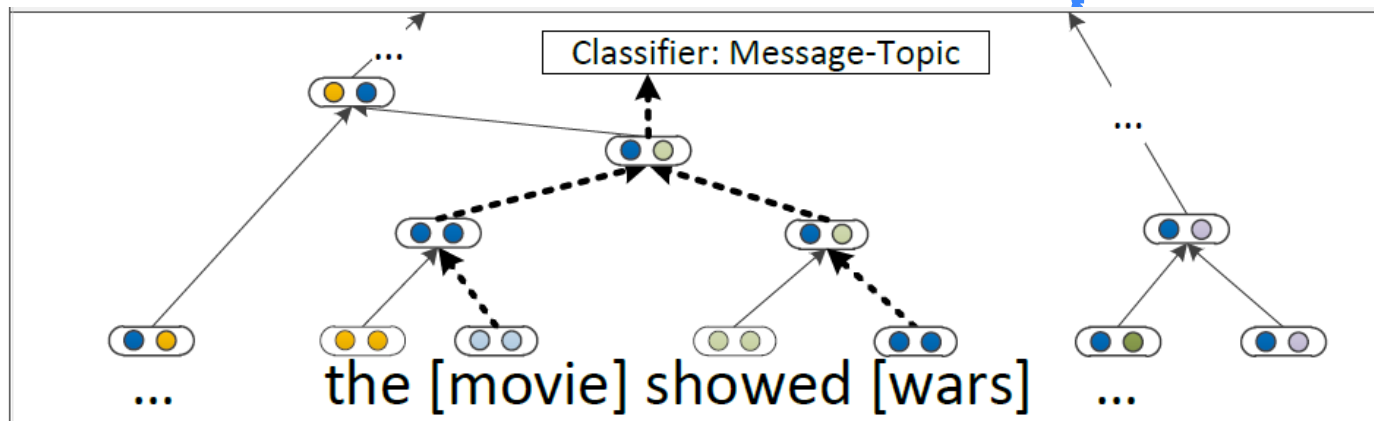


# Predicting Sentiment Distributions

- Good example for non-linearity in language



# MV-RNN for Relationship Classification



Relationship	Sentence with labeled nouns for which to predict relationships
Cause-Effect(e2,e1)	Avian [influenza] <sub>e1</sub> is an infectious disease caused by type a strains of the influenza [virus] <sub>e2</sub> .
Entity-Origin(e1,e2)	The [mother] <sub>e1</sub> left her native [land] <sub>e2</sub> about the same time and they were married in that city.
Message-Topic(e2,e1)	Roadside [attractions] <sub>e1</sub> are frequently advertised with [billboards] <sub>e2</sub> to attract tourists.

Classifier	Feature Sets	F1
SVM	POS, stemming, syntactic patterns	60.1
SVM	word pair, words in between	72.5
SVM	POS, WordNet, stemming, syntactic patterns	74.8
SVM	POS, WordNet, morphological features, thesauri, Google <i>n</i> -grams	77.6
MaxEnt	POS, WordNet, morphological features, noun compound system, thesauri, Google <i>n</i> -grams	77.6
SVM	POS, WordNet, prefixes and other morphological features, POS, dependency parse features, Levin classes, PropBank, FrameNet, NomLex-Plus, Google <i>n</i> -grams, paraphrases, TextRunner	82.2
RNN	-	74.8
Lin.MVR	-	73.0
MV-RNN	-	79.1
RNN	POS,WordNet,NER	77.6
Lin.MVR	POS,WordNet,NER	78.7
MV-RNN	POS,WordNet,NER	<b>82.4</b>

# Sentiment Detection

- Sentiment detection is crucial to business intelligence, stock trading, ...



Maybe she'll change her name to Halliburton. Just to see.

3/18/11 at 4:00 PM | 17 Comments  
Mentions of the Name 'Anne Hathaway' May Drive Berkshire Hathaway Stock

By Patrick Huguenin



The Huffington Post recently [pointed out](#) that whenever Anne Hathaway is in the news, the stock price for Warren Buffett's Berkshire Hathaway goes up. Really. When *Bride Wars* opened, the stock rose 2.61 percent. (*Rachel*

*Getting Married* only kicked it up 0.44 percent, but, you know, that one was so light on plot compared to *Bride Wars*.)

# Sentiment Detection and Bag-of-Words Models

- Most methods start with a bag of words + linguistic features/processing/lexica
- But such methods (including tf-idf) can't distinguish:
  - + white blood cells destroying an infection
  - an infection destroying white blood cells

# Sentiment Detection and Bag-of-Words Models

- Sentiment is that sentiment is “easy”
- Detection accuracy for longer documents ~90%
- Lots of easy cases (... horrible... or ... awesome ...)
- For dataset of single sentence movie reviews (Pang and Lee, 2005) accuracy never reached above 80% for >7 years
- Harder cases require actual understanding of negation and its scope and other semantic effects

## Data: Movie Reviews

Stealing Harvard doesn't care about cleverness, wit or any other kind of intelligent humor.

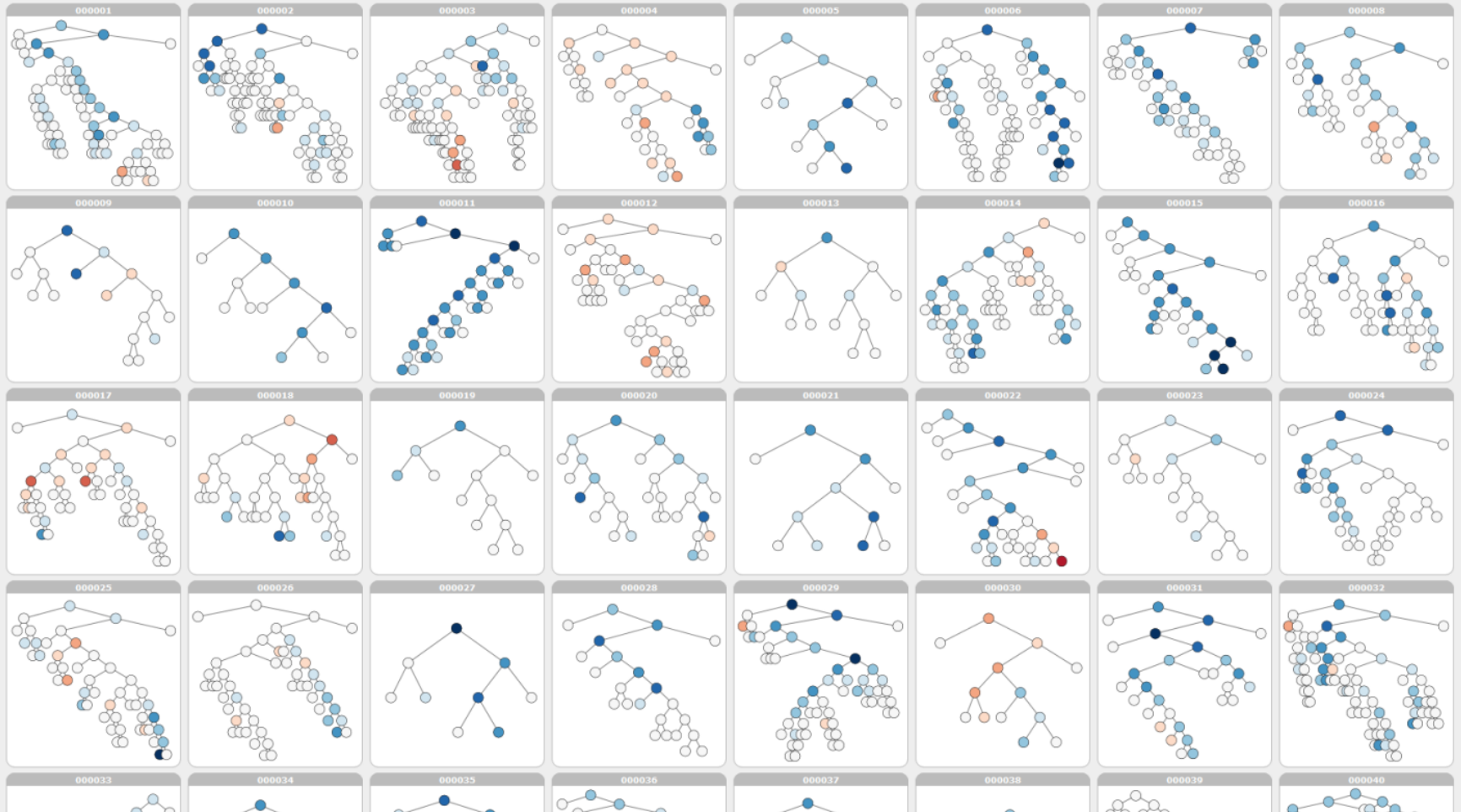
There are slow and repetitive parts but it has just enough spice to keep it interesting.



## Two missing pieces for improving sentiment

1. Compositional Training Data
2. Better Compositional model

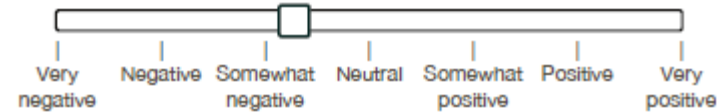
# 1. New Sentiment Treebank



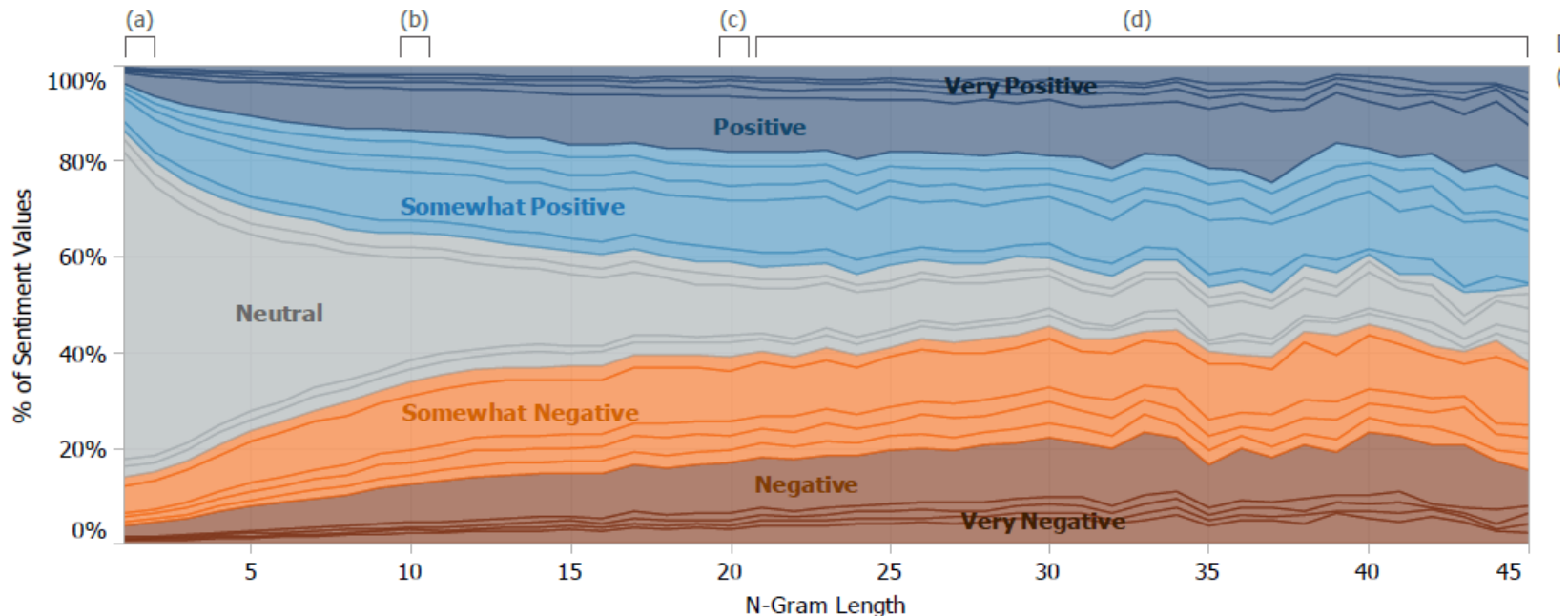
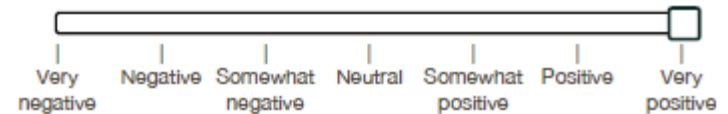
# 1. New Sentiment Treebank

- Parse trees of 11,855 sentences
- 215,154 phrases with labels
- Allows training and evaluating with compositional information

nerdy folks

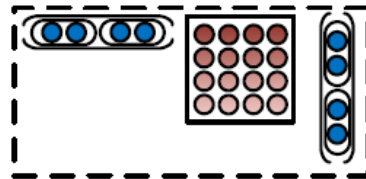
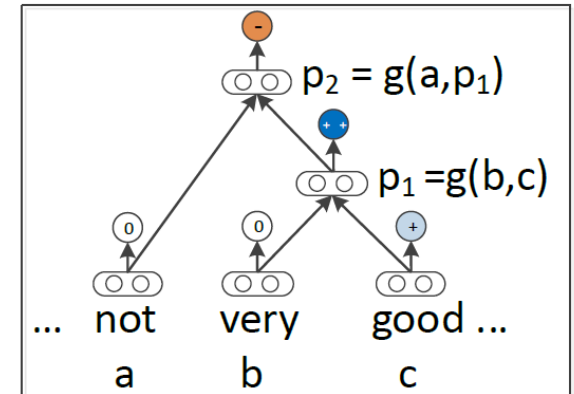


phenomenal fantasy best sellers



## 2. New Compositional Model

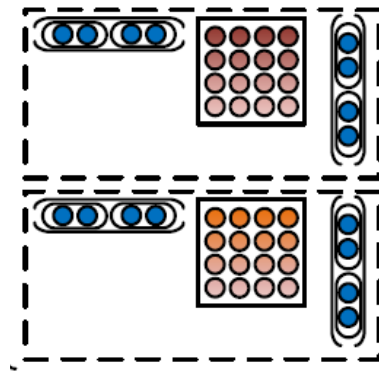
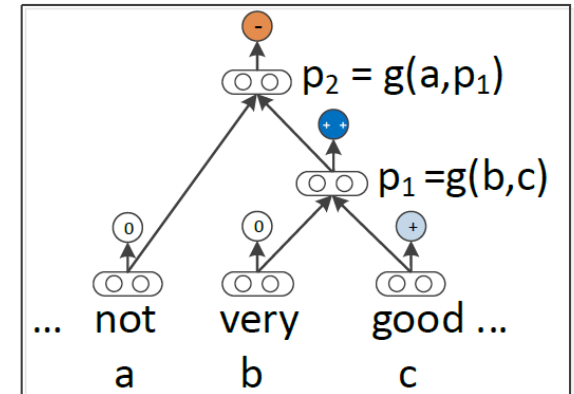
- Recursive Neural Tensor Network
- More expressive than any other RNN so far
- Idea: Allow more interactions of vectors



$$\begin{pmatrix} b \\ c \end{pmatrix}^T \quad \begin{pmatrix} b \\ c \end{pmatrix}$$

## 2. New Compositional Model

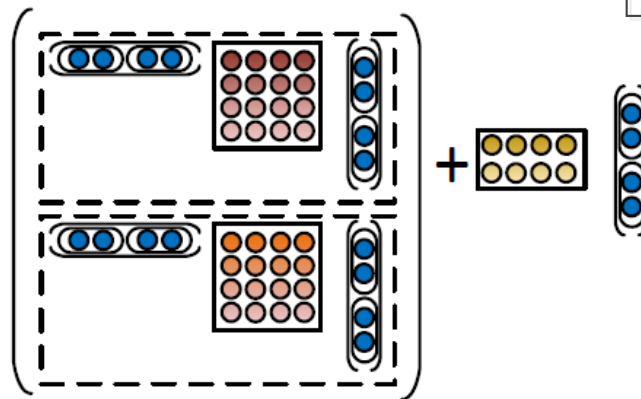
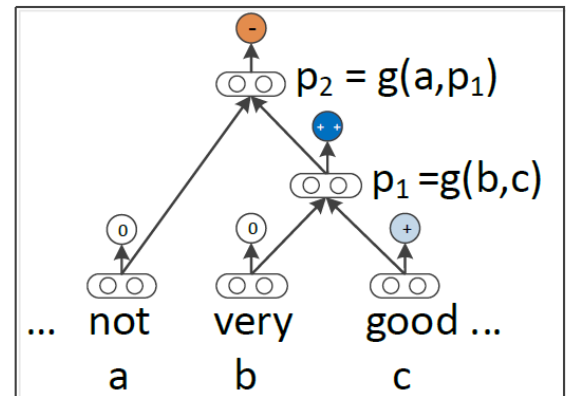
- Recursive Neural Tensor Network



$$\begin{pmatrix} b \\ c \end{pmatrix}^T v^{[1:2]} \begin{pmatrix} b \\ c \end{pmatrix}$$

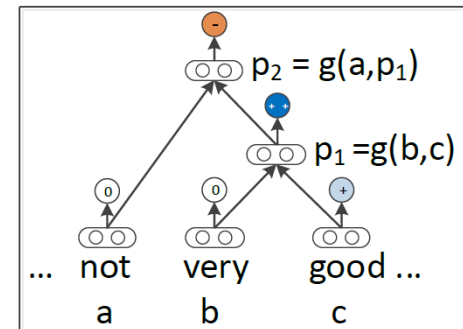
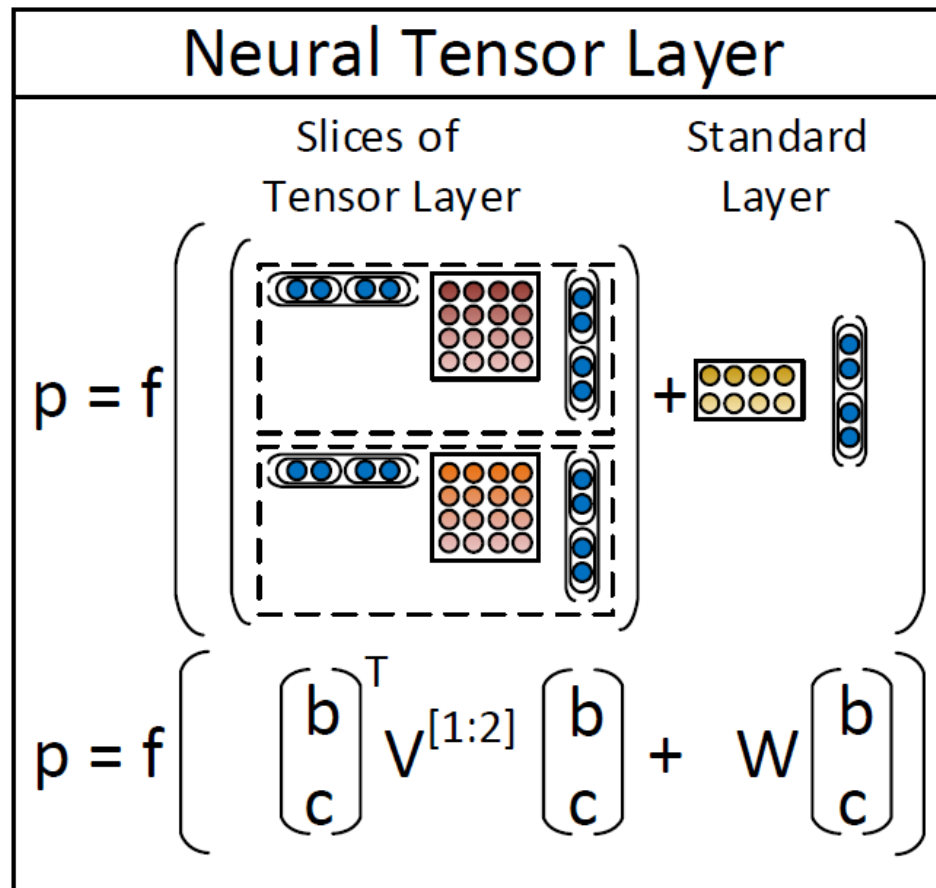
## 2. New Compositional Model

- Recursive Neural Tensor Network

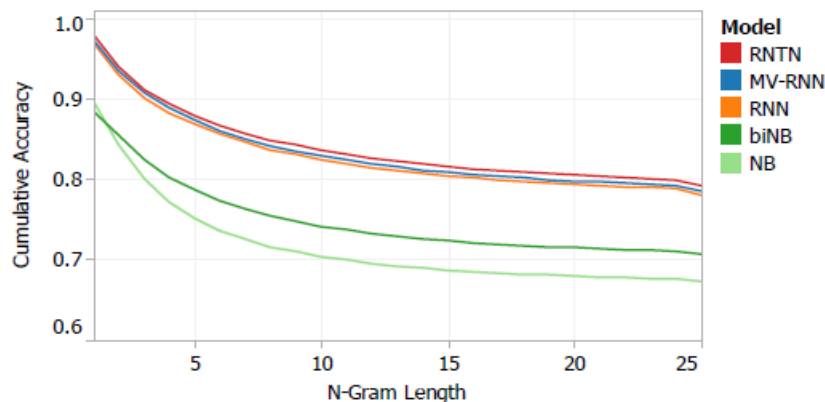
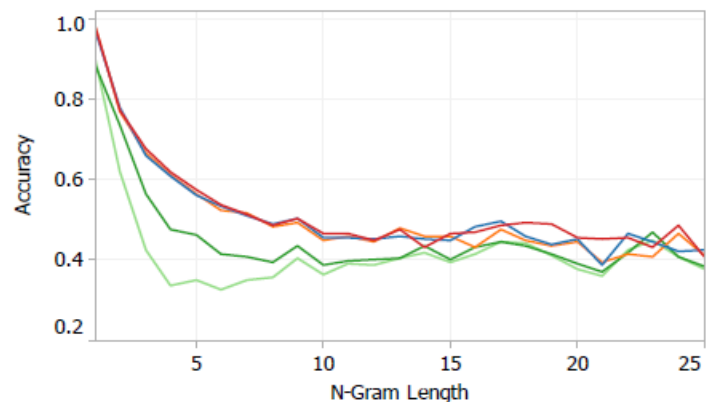


$$\begin{pmatrix} b \\ c \end{pmatrix}^T v^{[1:2]} \begin{pmatrix} b \\ c \end{pmatrix} + w \begin{pmatrix} b \\ c \end{pmatrix}$$

# Recursive Neural Tensor Network



# Experimental Result on Treebank

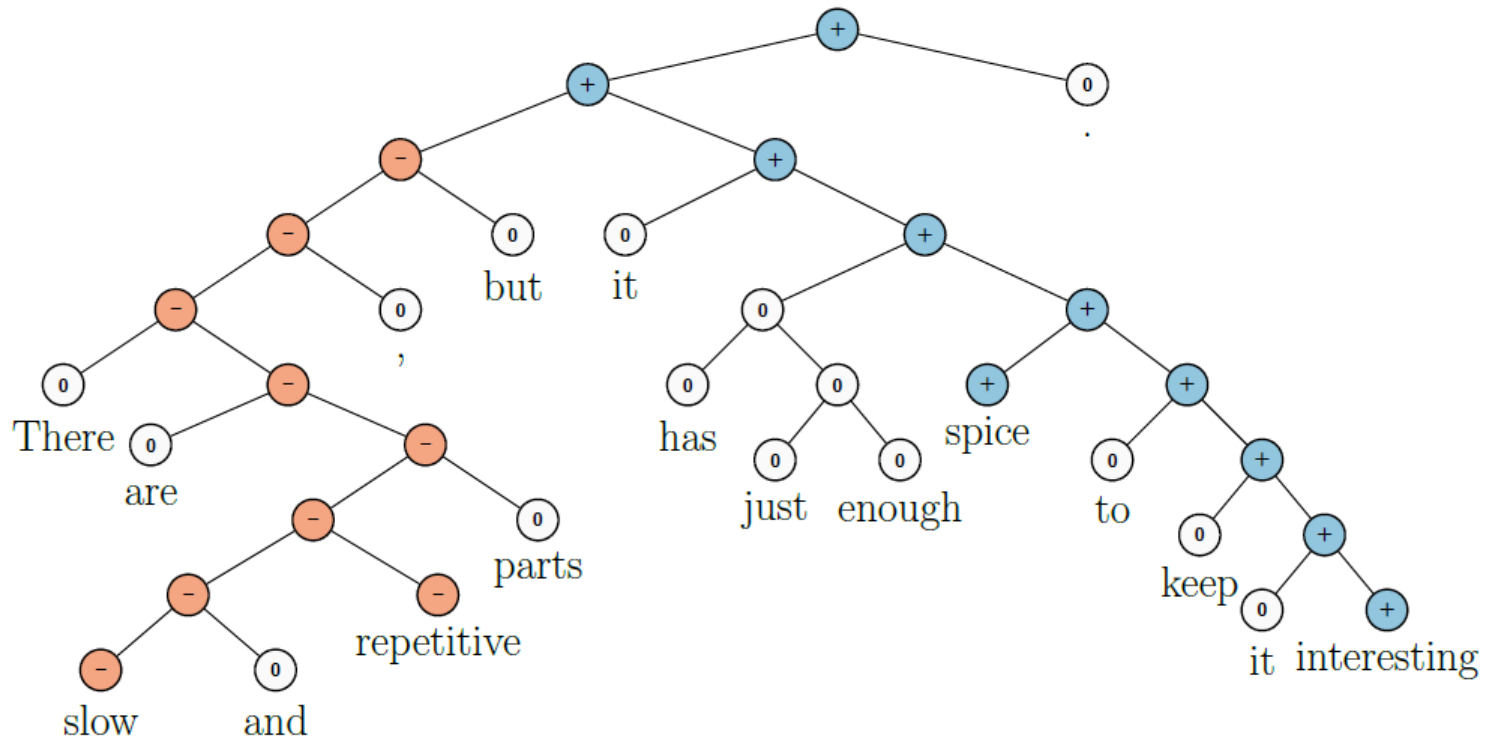


Model	Fine-grained		Positive/Negative	
	All	Root	All	Root
NB	67.2	41.0	82.6	81.8
SVM	64.3	40.7	84.6	79.4
BiNB	71.0	41.9	82.7	83.1
VecAvg	73.3	32.7	85.1	80.1
RNN	79.0	43.2	86.1	82.4
MV-RNN	78.7	44.4	86.8	82.9
RNTN	<b>80.7</b>	<b>45.6</b>	<b>87.6</b>	<b>85.4</b>

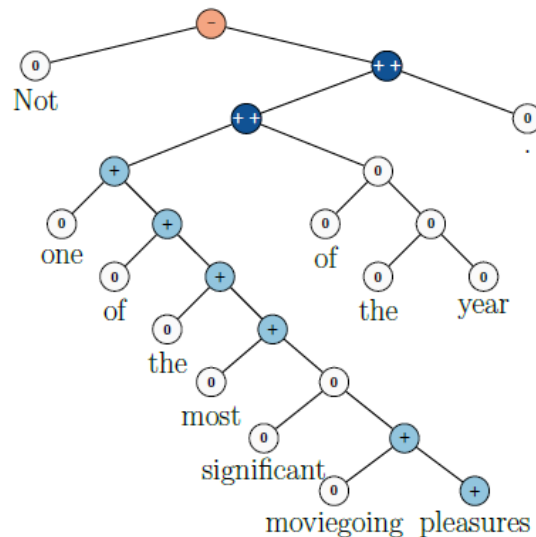
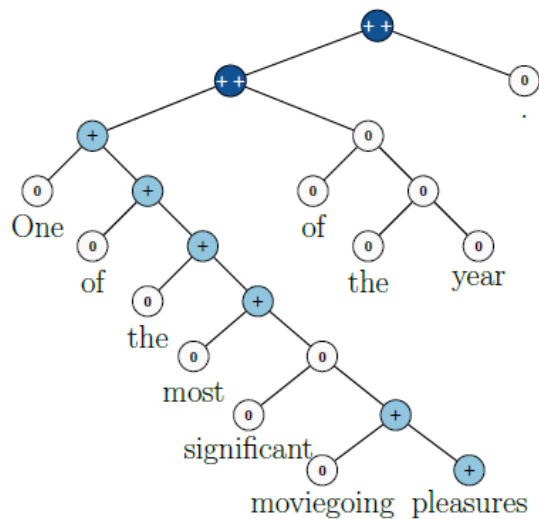
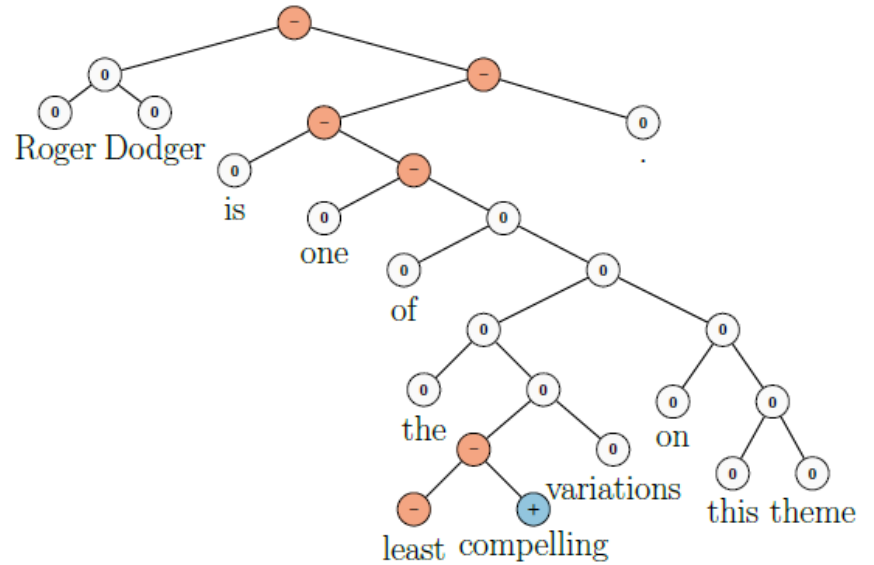
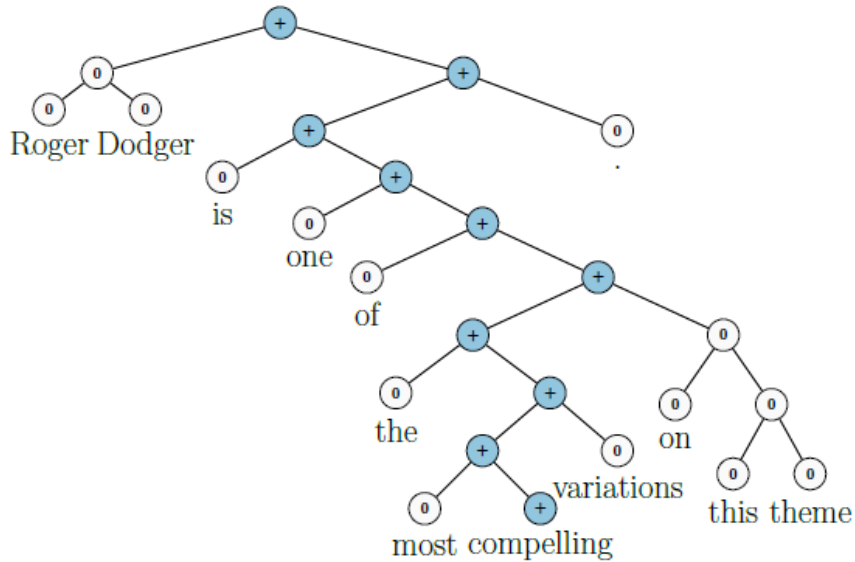


# Experimental Result on Treebank

- RNTN can capture X but Y
- RNTN accuracy of 72%, compared to MV-RNN (65), biNB (58) and RNN (54)

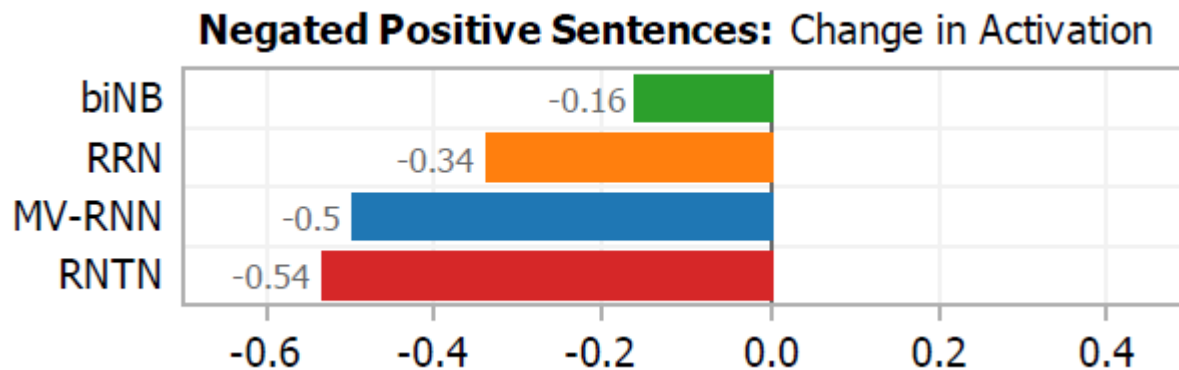


# Negation Results



# Negation Results

- Most methods capture that negation often makes things more negative (See Potts, 2010)
- Analysis on negation dataset



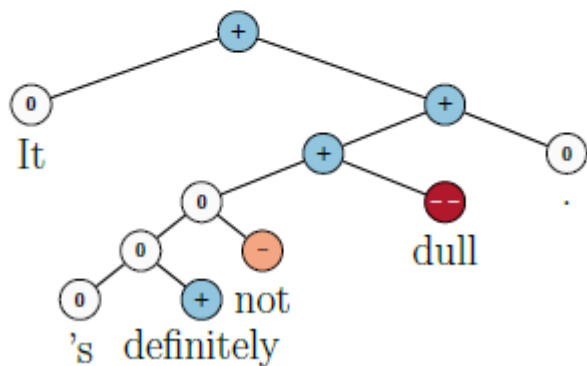
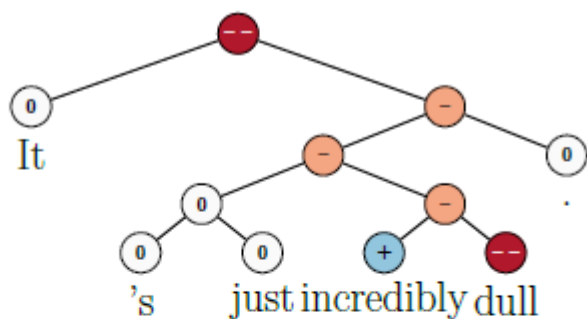
Negated Positive

---

biNB	19.0
RNN	33.3
MV-RNN	52.4
RNTN	<b>71.4</b>

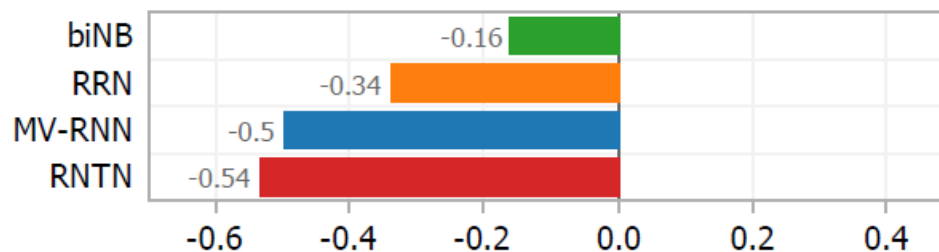
# Negation Results

- But how about negating negatives?
- Positive activation should increase!

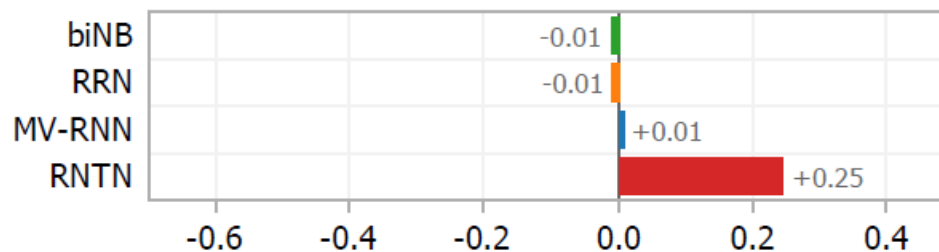


Model	Accuracy	
	Negated Positive	Negated Negative
biNB	19.0	27.3
RRN	33.3	45.5
MV-RNN	52.4	54.6
RNTN	<b>71.4</b>	<b>90.9</b>

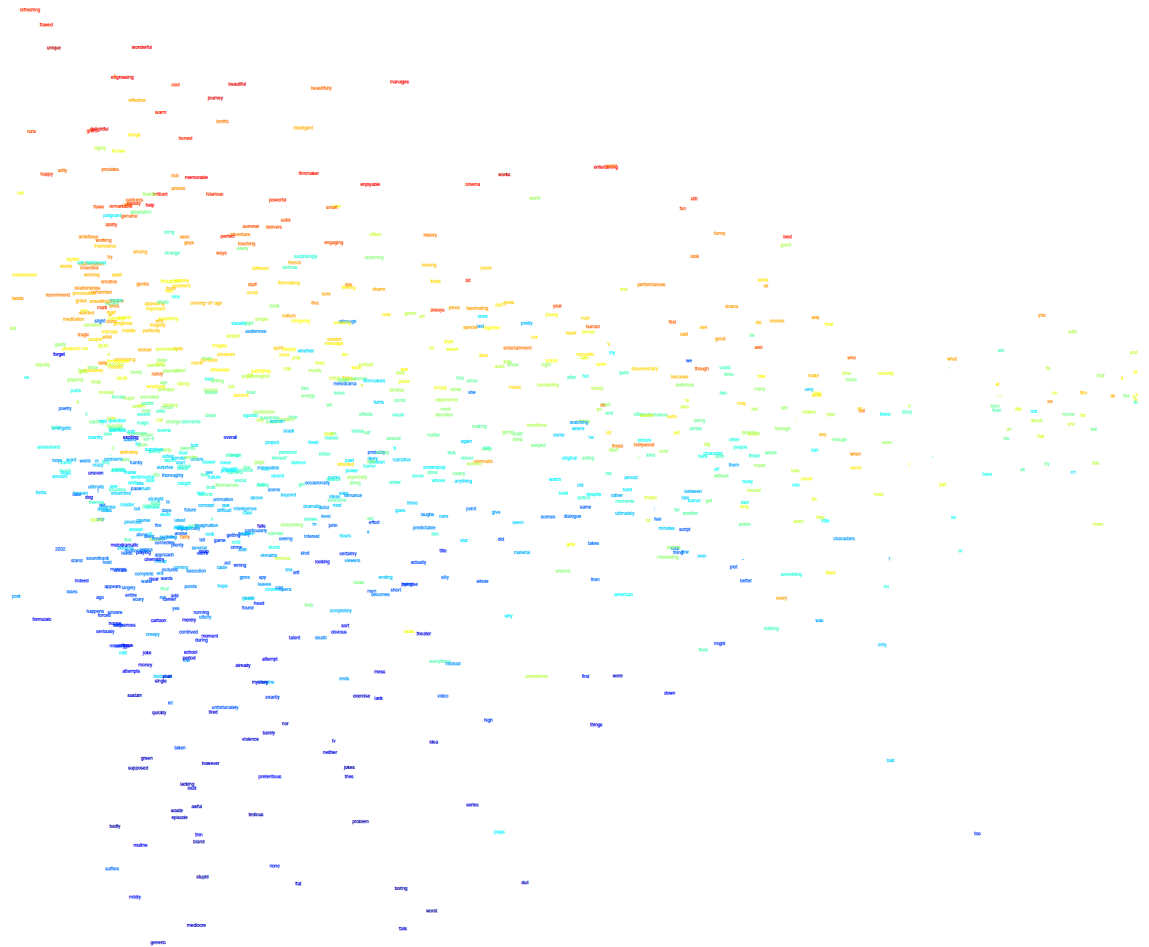
**Negated Positive Sentences: Change in Activation**



**Negated Negative Sentences: Change in Activation**



# Visualizing Deep Learning: Word Embeddings

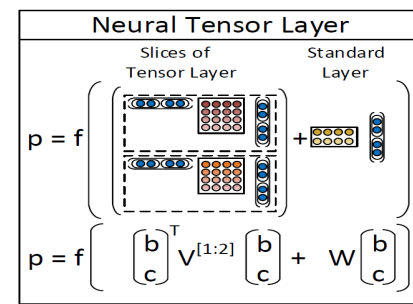
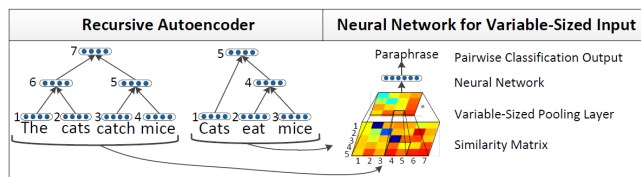
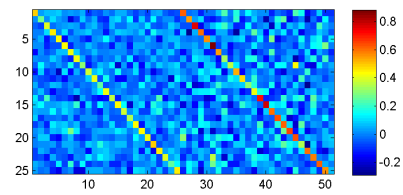
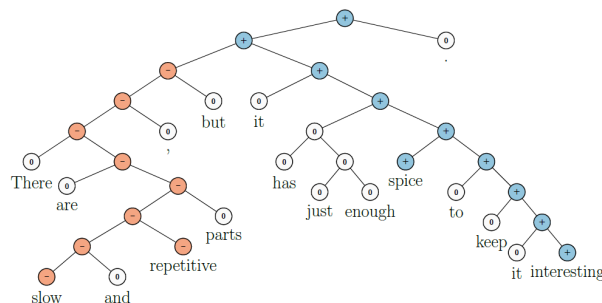
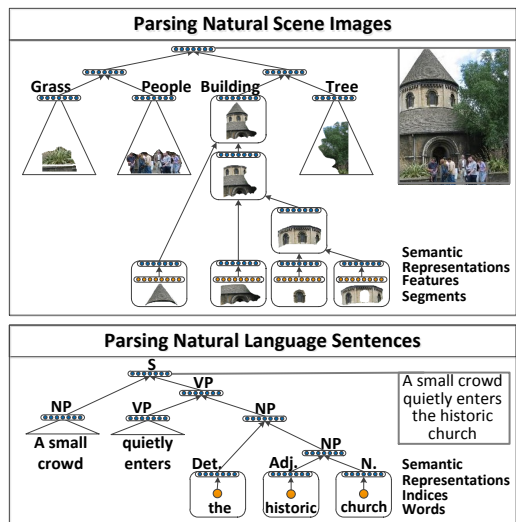


# Overview of RNN Model Variations

- Objective Functions
  - **Supervised Scores for Structure Prediction**
  - **Classifier for Sentiment, Relations, Visual Objects, Logic**
  - **Unsupervised autoencoding immediate children or entire tree structure**
- Composition Functions
  - **Syntactically-Untied Weights**
  - **Matrix Vector RNN**
  - **Tensor-Based Models**
- Tree Structures
  - **Constituency Parse Trees**
  - **Combinatory Categorical Grammar Trees**
  - **Dependency Parse Trees**
  - **Fixed Tree Structures (Connections to CNNs)**

# Summary: Recursive Deep Learning

- Recursive Deep Learning can predict hierarchical structure and classify the structured output using compositional vectors
- State-of-the-art performance (all with code on [www.socher.org](http://www.socher.org))
  - Parsing** on the WSJ (Java code soon)
  - Sentiment Analysis** on multiple corpora
  - Paraphrase detection** with unsupervised RNNs
  - Relation Classification** on SemEval 2011, Task8
  - Object detection** on Stanford background and MSRC datasets



## Part 3

1. Assorted Speech and NLP Applications
2. Deep Learning: General Strategy and Tricks
3. Resources (readings, code, ...)
4. Discussion



Part 3.1: Applications

# Assorted Speech and NLP Applications

# Existing NLP Applications

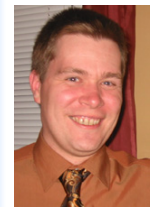
- **Language Modeling** (Speech Recognition, Machine Translation)
- **Word-Sense Learning** and Disambiguation
- **Reasoning over Knowledge Bases**
- Acoustic Modeling
- Part-Of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Parsing
- Sentiment Analysis
- Paraphrasing
- Question-Answering

# Language Modeling

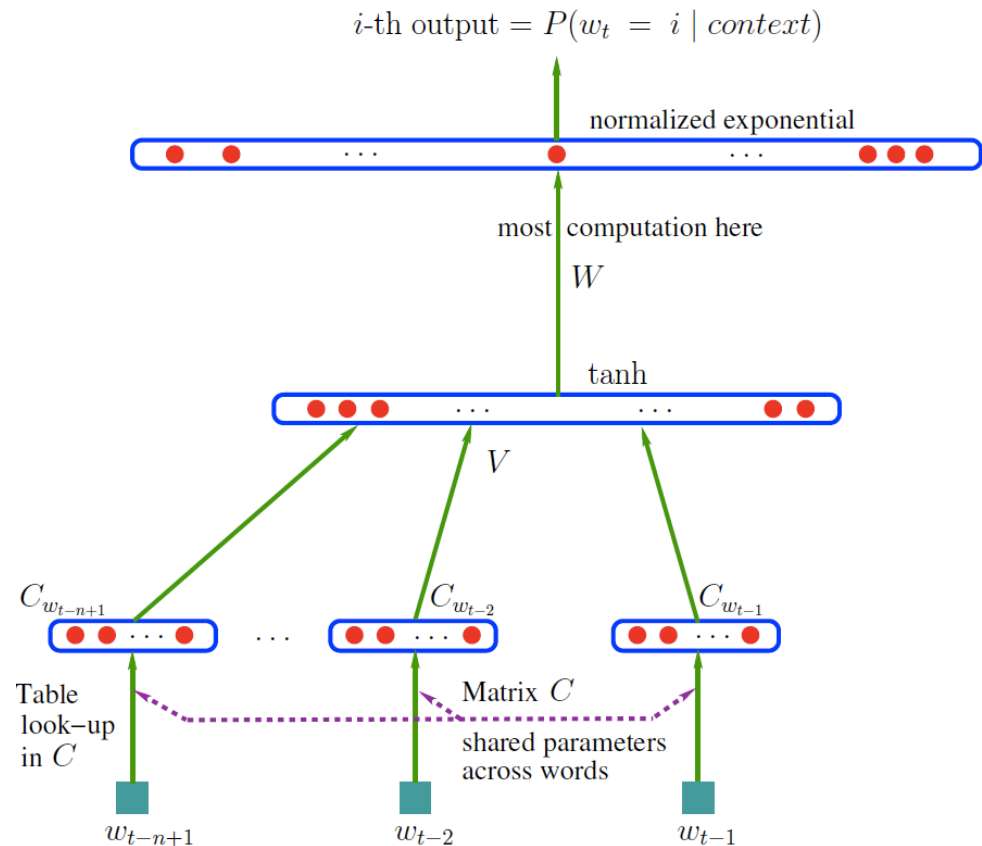
- Predict  $P(\text{next word} \mid \text{previous word})$
- Gives a probability for a longer sequence
- Applications to Speech, Translation and Compression
- Computational bottleneck: large vocabulary  $V$  means that computing the output costs  $\# \text{hidden units} \times |V|$ .

# Neural Language Model

- *Bengio et al NIPS'2000 and JMLR 2003 "A Neural Probabilistic Language Model"*



- Each word represented by a distributed continuous-valued code
- Generalizes to sequences of words that are semantically similar to training sequences



# Recurrent Neural Net Language Modeling for ASR

- [Mikolov et al 2011]

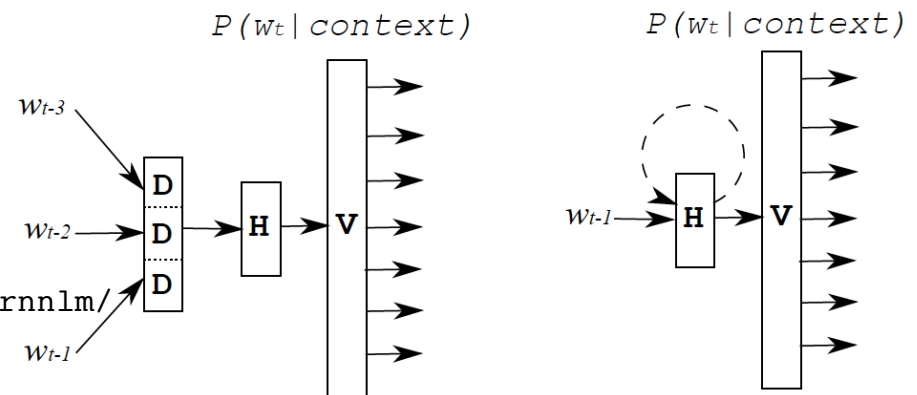
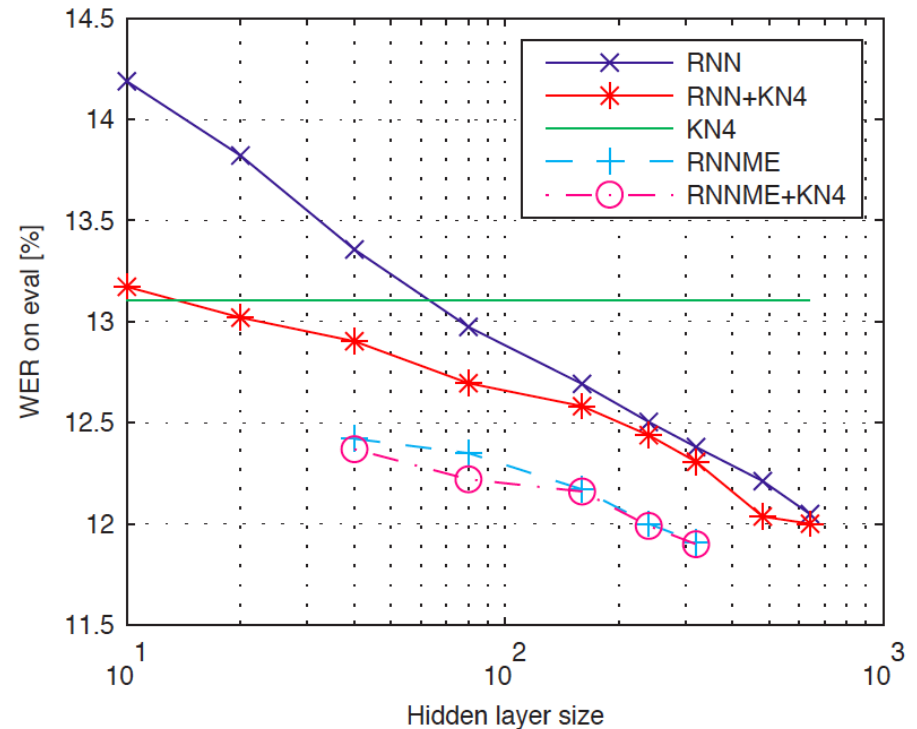


Bigger is better...  
experiments on Broadcast  
News NIST-RT04

perplexity goes from  
140 to 102

Paper shows how to  
train a recurrent neural net  
with a single core in a few  
days, with > 1% absolute  
improvement in WER

Code: <http://www.fit.vutbr.cz/~imikolov/rnnlm/>



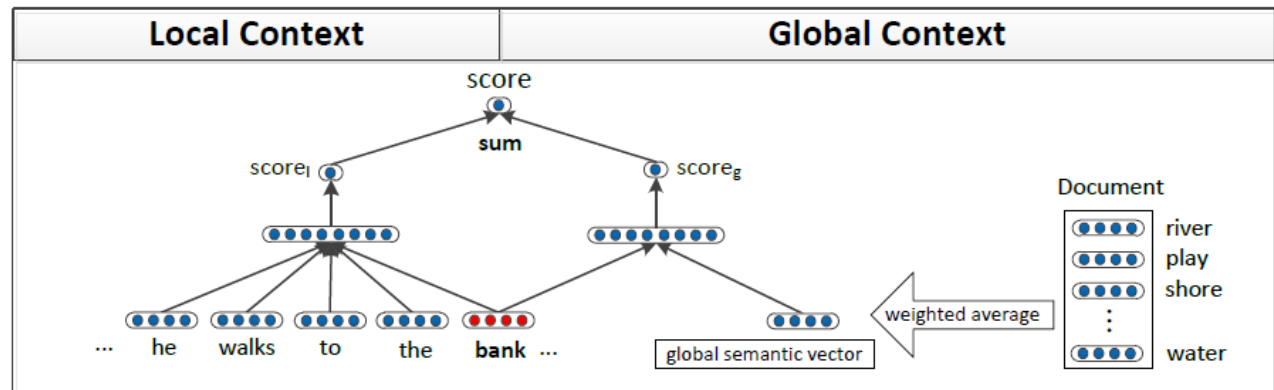
# Application to Statistical Machine Translation



- Schwenk (NAACL 2012 workshop on the future of LM)
  - 41M words, Arabic/English bitexts + 151M English from LDC
- Perplexity down from 71.1 (6 Gig back-off) to 56.9 (neural model, 500M memory)
- +1.8 BLEU score (50.75 to 52.28)
- Can take advantage of longer contexts
- Code: <http://lium.univ-lemans.fr/cs1m/>

# Learning Multiple Word Vectors

- Tackles problems with polysemous words
- Can be done with both standard tf-idf based methods [Reisinger and Mooney, NAACL 2010]
- Recent neural word vector model by [Huang et al. ACL 2012] learns multiple prototypes using both local and global context
- State of the art correlations with human similarity judgments



# Learning Multiple Word Vectors

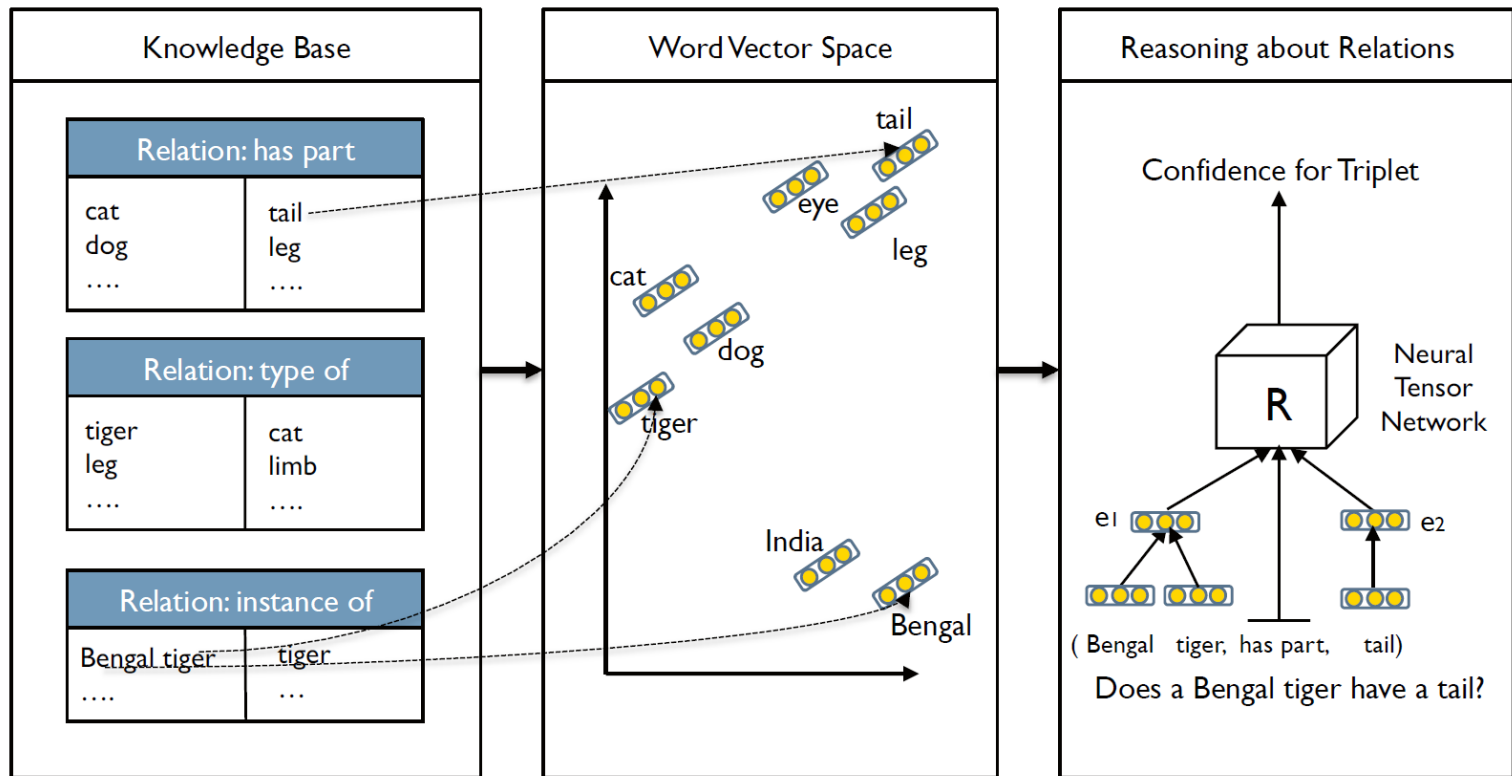
- Visualization of learned word vectors from Huang et al. (ACL 2012)





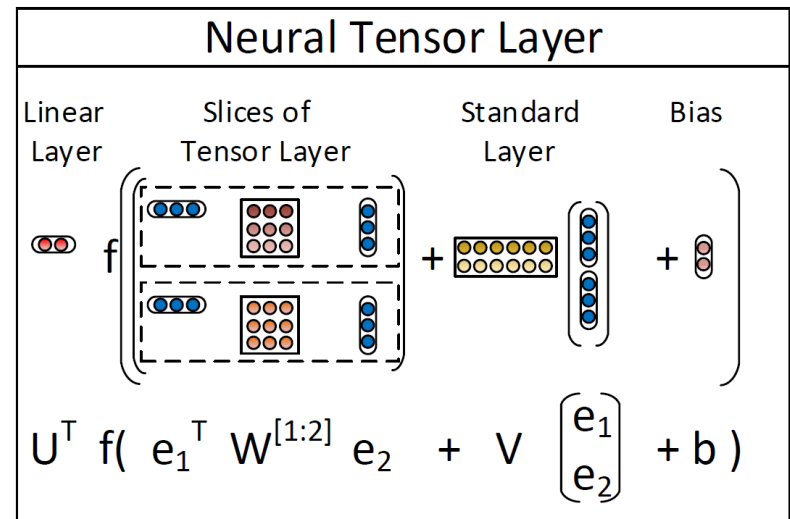
# Common Sense Reasoning Inside Knowledge Bases

- Question: Can Neural Networks learn to capture logical inference, set inclusions, part-of and hypernym relationships?



# Neural Networks for Reasoning over Relationships

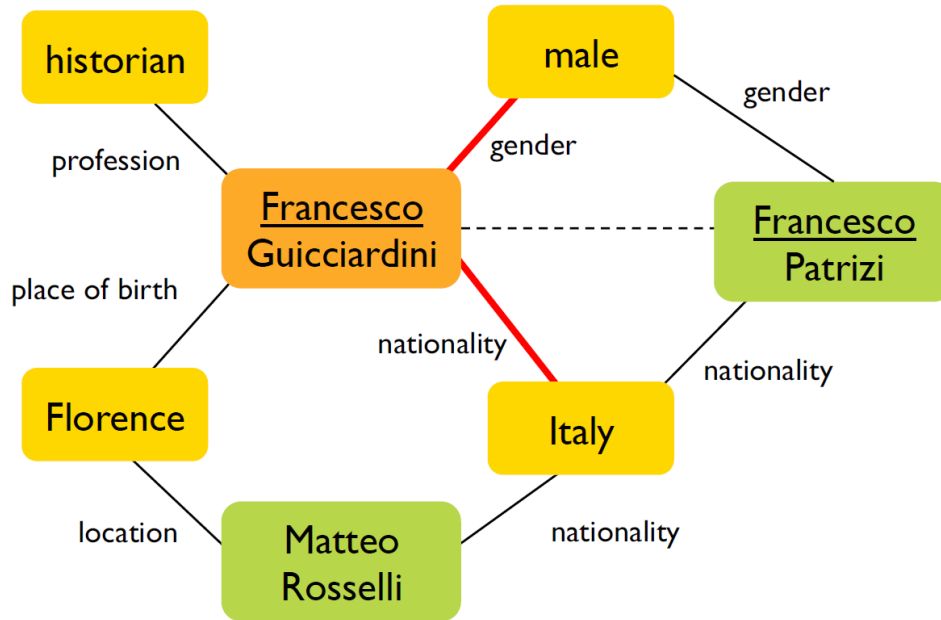
- Higher scores for each triplet  $T = (e_1, R, e_2)$  indicate that entities are more likely in relationship
- Training uses contrastive estimation function, similar to word vector learning



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

- Cost:  $\sum_{i=1}^N \sum_{c=1}^C \max \left( 0, 1 - g \left( T^{(i)} \right) + g \left( T_c^{(i)} \right) \right) + \lambda \|\Omega\|_2^2,$

# Accuracy of Predicting True and False Relationships



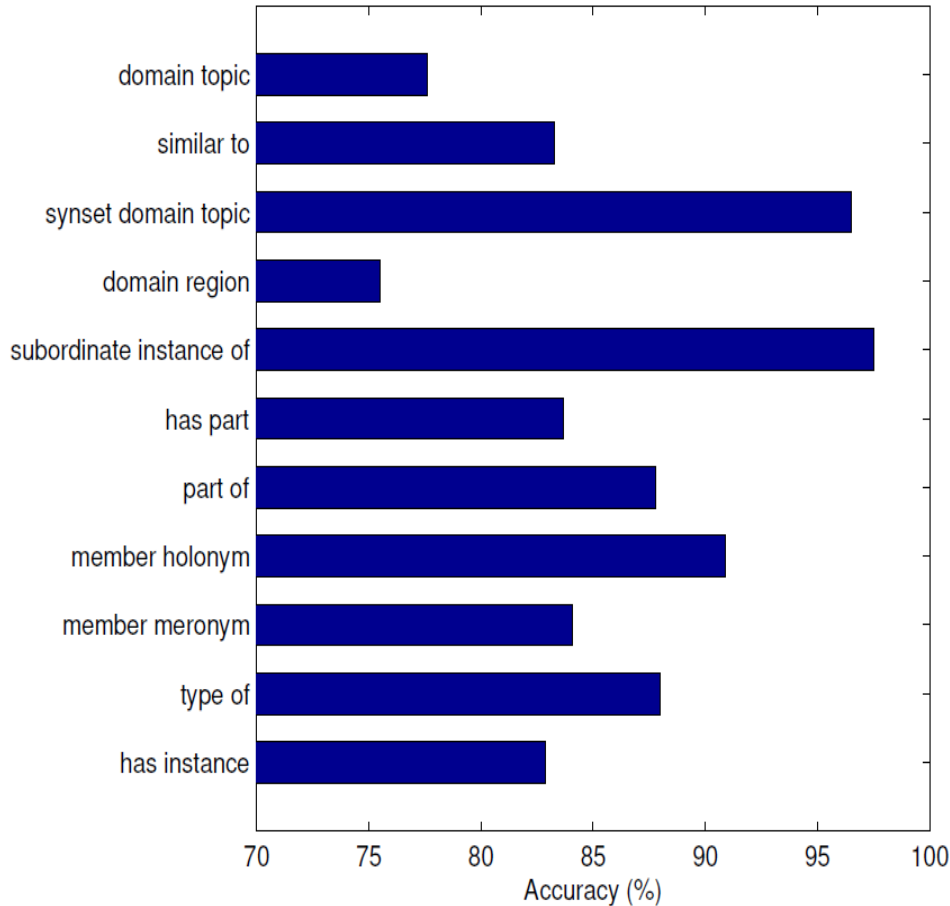
- Related Work
- Bordes, Weston, Collobert & Bengio, AAAI 2011)
- (Bordes, Glorot, Weston & Bengio, AISTATS 2012)



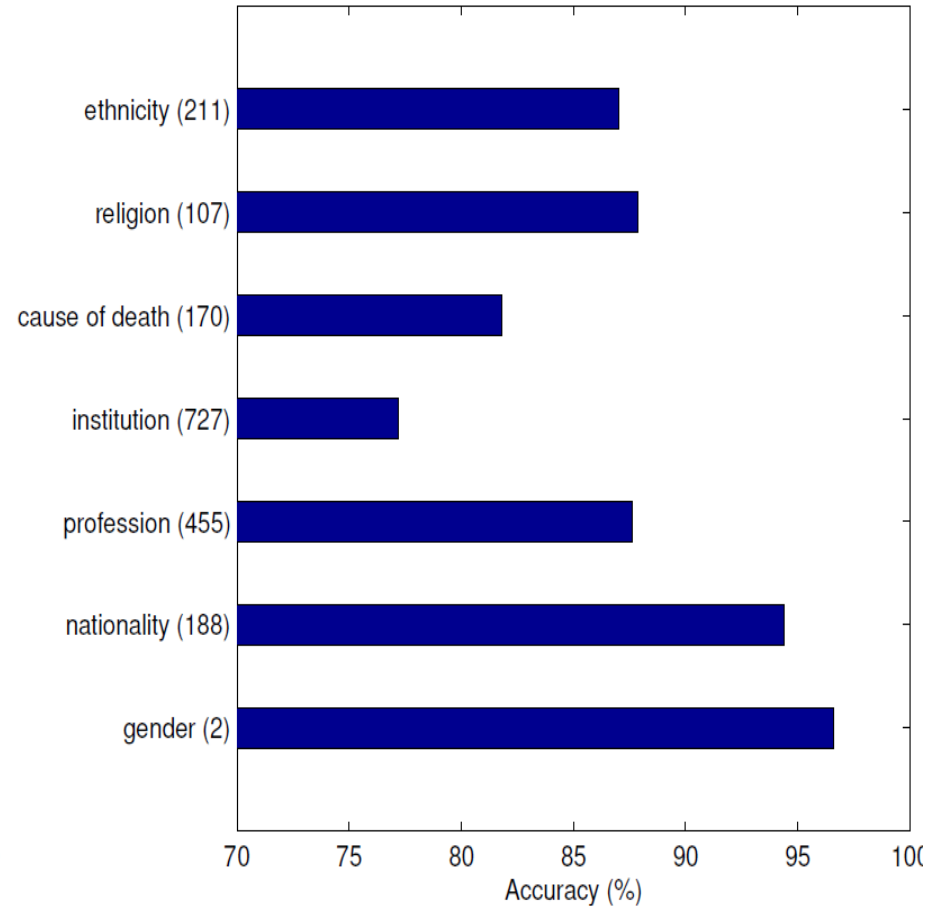
Model	FreeBase	WordNet
Distance Model	68.3	61.0
Hadamard Model	80.0	68.8
Standard Layer Model (<NTN)	76.0	85.3
Bilinear Model (<NTN)	84.1	87.7
Neural Tensor Network (Chen et al. 2013)	<b>86.2</b>	<b>90.0</b>

# Accuracy Per Relationship

WordNet



FreeBase



Part 3.2

# Deep Learning General Strategy and Tricks

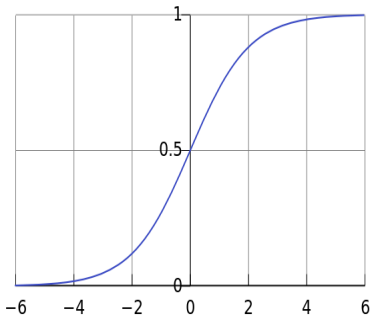
# General Strategy

1. Select network structure appropriate for problem
  1. Structure: Single words, fixed windows vs Recursive Sentence Based vs Bag of words
  2. Nonlinearity
2. Check for implementation bugs with gradient checks
3. Parameter initialization
4. Optimization tricks
5. Check if the model is powerful enough to overfit
  1. If not, change model structure or make model “larger”
  2. If you can overfit: Regularize

# Non-linearities: What's used

logistic (“sigmoid”)

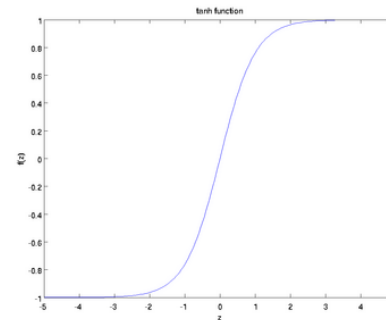
$$f(z) = \frac{1}{1 + \exp(-z)}$$



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

tanh

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



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

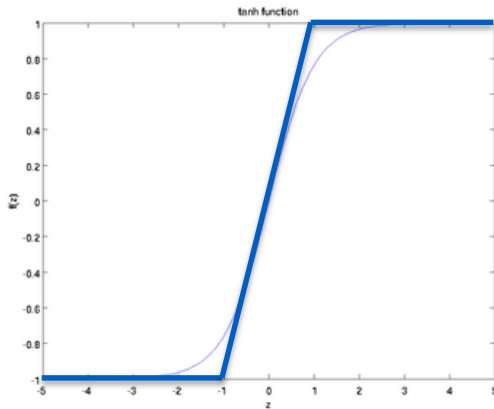
tanh is just a rescaled and shifted sigmoid  $\tanh(z) = 2\text{logistic}(2z) - 1$

tanh is what is most used and often performs best for deep nets

# Non-linearities: There are various other choices

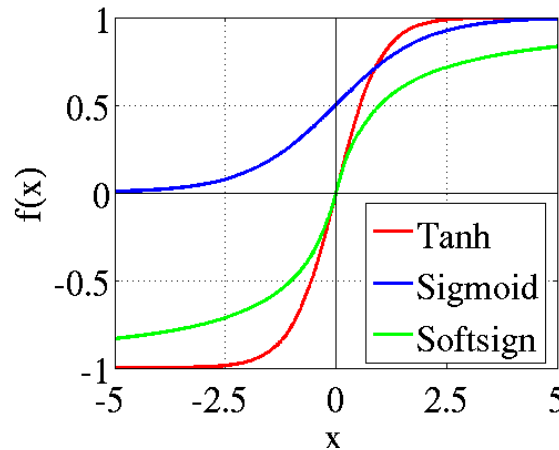
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$



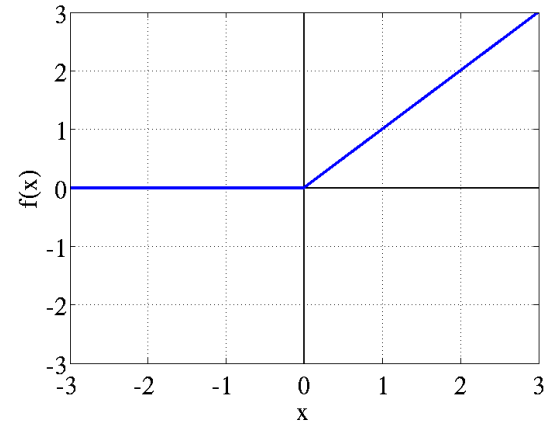
soft sign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$



rectifier

$$\text{rect}(z) = \max(z, 0)$$



- hard tanh similar but computationally cheaper than tanh and saturates hard.
- [Glorot and Bengio *AISTATS* 2010, 2011] discuss softsign and rectifier



# MaxOut Network

- A very recent type of nonlinearity/network
- Goodfellow et al. (2013)

$$f_i(z) = \max_{j \in [1, k]} z_{ij}$$

- Where  $z_{ij} = x^T W_{..ij} + b_{ij}$
- This function too is a universal approximator
- State of the art on several image datasets

# Gradient Checks are Awesome!

- Allows you to know that there are no bugs in your neural network implementation!
- Steps:
  1. Implement your gradient
  2. Implement a finite difference computation by looping through the parameters of your network, adding and subtracting a small epsilon ( $\sim 10^{-4}$ ) and estimate derivatives

$$g_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2 \times \text{EPSILON}}. \quad \theta^{(i+)} = \theta + \text{EPSILON} \times \vec{e}_i$$

3. Compare the two and make sure they are the same

# General Strategy

1. Select appropriate Network Structure
  1. Structure: Single words, fixed windows vs Recursive Sentence Based vs Bag of words
  2. Nonlinearity
2. Check for implementation bugs with gradient check
3. Parameter initialization
4. Optimization tricks
5. Check if the model is powerful enough to overfit
  1. If not, change model structure or make model “larger”
  2. If you can overfit: Regularize

# Parameter Initialization

- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g. mean target or inverse sigmoid of mean target).
- Initialize weights  $\sim$  Uniform(-r,r), r inversely proportional to fan-in (previous layer size) and fan-out (next layer size):

$$\sqrt{6 / (\text{fan-in} + \text{fan-out})}$$

for tanh units, and 4x bigger for sigmoid units [Glorot AISTATS 2010]

- Pre-training with Restricted Boltzmann machines

# Stochastic Gradient Descent (SGD)

- Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}$$

- $L$  = loss function,  $z_t$  = current example,  $\theta$  = parameter vector, and  $\epsilon_t$  = learning rate.
- Ordinary gradient descent as a batch method, very slow, **should never be used**. Use 2<sup>nd</sup> order batch method such as LBFGS. On large datasets, SGD usually wins over all batch methods. On smaller datasets LBFGS or Conjugate Gradients win. Large-batch LBFGS extends the reach of LBFGS [Le et al ICML'2011].

# Learning Rates

- Simplest recipe: keep it fixed and use the same for all parameters.
- Collobert scales them by the inverse of square root of the fan-in of each neuron
- Better results can generally be obtained by allowing learning rates to decrease, typically in  $O(1/t)$  because of theoretical convergence guarantees, e.g.,  
with hyper-parameters  $\epsilon_0$  and  $\tau$  
$$\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$$
- Better yet: No learning rates by using L-BFGS or AdaGrad (Duchi et al. 2011)

# Long-Term Dependencies and Clipping Trick

- In very deep networks such as recurrent networks (or possibly recursive ones), the gradient is a product of Jacobian matrices, each associated with a step in the forward computation. This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down.



- The solution first introduced by Mikolov is to clip gradients to a maximum value. Makes a big difference in RNNs



# General Strategy

1. Select appropriate Network Structure
  1. Structure: Single words, fixed windows vs Recursive Sentence Based vs Bag of words
  2. Nonlinearity
2. Check for implementation bugs with gradient check
3. Parameter initialization
4. Optimization tricks
5. Check if the model is powerful enough to overfit
  1. If not, change model structure or make model “larger”
  2. If you can overfit: Regularize

Assuming you found the right network structure, implemented it correctly, optimize it properly and you can make your model overfit on your training data.

Now, it's time to regularize




# Prevent Overfitting: Model Size and Regularization

- Simple first step: Reduce model size by lower number of units and layers and other parameters
- Standard L1 or L2 regularization on weights
- Early Stopping: Use parameters that gave best validation error
- Sparsity constraints on hidden activations, e.g. add to cost:

$$KL \left( 1/N \sum_{n=1}^N a_i^{(n)} \parallel 0.0001 \right)$$

- Dropout (Hinton et al. 2012):
  - Randomly set 50% of the inputs at each layer to 0
  - At test time half the outgoing weights (now twice as many)
  - Prevents Co-adaptation

# Deep Learning Tricks of the Trade

- Y. Bengio (2012), “Practical Recommendations for Gradient-Based Training of Deep Architectures”
  - Unsupervised pre-training
  - Stochastic gradient descent and setting learning rates
  - Main hyper-parameters
    - Learning rate schedule & early stopping
    - Minibatches
    - Parameter initialization
    - Number of hidden units
    - L1 or L2 weight decay
    - Sparsity regularization
  - Debugging → Finite difference gradient check (Yay)
  - How to efficiently search for hyper-parameter configurations

Part 3.3: Resources

## Resources: Tutorials and Code

# Related Tutorials

- See “*Neural Net Language Models*” **Scholarpedia** entry
- Deep Learning tutorials: <http://deeplearning.net/tutorials>
- Stanford deep learning tutorials with simple programming assignments and reading list <http://deeplearning.stanford.edu/wiki/>
- Recursive Autoencoder class project  
<http://cseweb.ucsd.edu/~elkan/250B/learningmeaning.pdf>
- Graduate Summer School: Deep Learning, Feature Learning  
<http://www.ipam.ucla.edu/programs/gss2012/>
- ICML 2012 Representation Learning tutorial <http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-2012.html>
- **More reading (including tutorial references):**  
<http://nlp.stanford.edu/courses/NAACL2013/>

# Software

- Theano (Python CPU/GPU) mathematical and deep learning library <http://deeplearning.net/software/theano>
  - Can do automatic, symbolic differentiation
- Senna: POS, Chunking, NER, SRL
  - by Collobert et al. <http://ronan.collobert.com/senna/>
  - State-of-the-art performance on many tasks
  - 3500 lines of C, extremely fast and using very little memory
- Recurrent Neural Network Language Model  
<http://www.fit.vutbr.cz/~imikolov/rnnlm/>
- Recursive Neural Net and RAE models for paraphrase detection, sentiment analysis, relation classification [www.socher.org](http://www.socher.org)

# Software: what's next

- Off-the-shelf SVM packages are useful to researchers from a wide variety of fields (no need to understand RKHS).
- One of the goals of deep learning: Build off-the-shelf NLP classification packages that are using as training input only raw text (instead of features) possibly with a label.

Part 3.4:

## Discussion

# Concerns

- Many algorithms and variants (burgeoning field)
- Hyper-parameters (layer size, regularization, possibly learning rate)
  - Use multi-core machines, clusters and random sampling for cross-validation (Bergstra & Bengio 2012)
  - Pretty common for powerful methods, e.g. BM25, LDA
  - Can use (mini-batch) L-BFGS instead of SGD



# Concerns

- Not always obvious how to combine with existing NLP
  - Simple: Add word or phrase vectors as features. Gets close to state of the art for NER, [Turian et al, ACL 2010]
  - Integrate with known problem structures: Recursive and recurrent networks for trees and chains
  - Your research here

# Concerns

- Slower to train than linear models
  - Only by a small constant factor, and much more compact than non-parametric (e.g. n-gram models)
  - Very fast during inference/test time (feed-forward pass is just a few matrix multiplies)
- Need more training data
  - Can *handle and benefit from* more training data, suitable for age of Big Data (Google trains neural nets with a billion connections, [Le et al, ICML 2012])

# CONCERNS

- There aren't many good ways to encode prior knowledge about the structure of language into deep learning models
  - There is some truth to this. However:
  - You can choose architectures suitable for a problem domain, as we did for linguistic structure
  - You can include human-designed features in the first layer, just like for a linear model
  - And the goal is to get the machine doing the learning!

## Concern:

# Problems with model interpretability

- No discrete categories or words, everything is a continuous vector. We'd like have symbolic features like NP, VP, etc. and see why their combination makes sense.
  - True, but most of language is fuzzy and many words have soft relationships to each other. Also, many NLP features are already not human-understandable (e.g., concatenations/ combinations of different features).
  - Can try by projections of weights and nearest neighbors, see part 2

# Concern: non-convex optimization

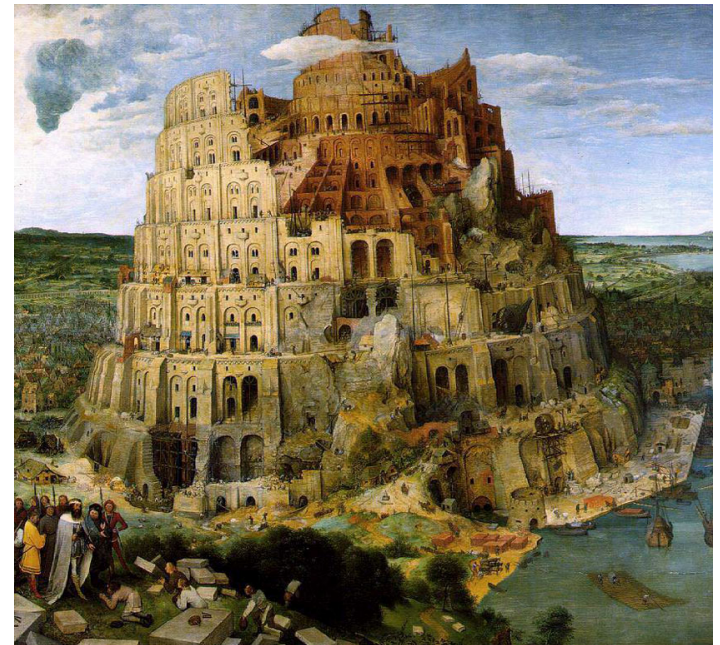
- Can initialize system with convex learner
  - Convex SVM
  - Fixed feature space
- Then optimize non-convex variant (add and tune learned features), can't be worse than convex learner
- Not a big problem in practice (often relatively stable performance across different local optima)

# Advantages

- Despite a small community in the intersection of deep learning and NLP, already many state of the art results on a variety of language tasks
- Often very simple matrix derivatives (backprop) for training and matrix multiplications for testing → fast implementation
- Fast inference and well suited for multi-core CPUs/GPUs and parallelization across machines

# Learning Multiple Levels of Abstraction

- The big payoff of deep learning is to learn feature representations and higher levels of abstraction
- This allows much easier generalization and transfer between domains, languages, and tasks



# The End

