

Deep Learning for NLP

(without Magic)



Université 
de Montréal

Richard Socher, Yoshua Bengio and Chris Manning

ACL 2012



Deep Learning

Most of current machine learning learns the predictive value (the weights) of human-given features

Representation learning systems attempt to automatically learn good representations (or features)

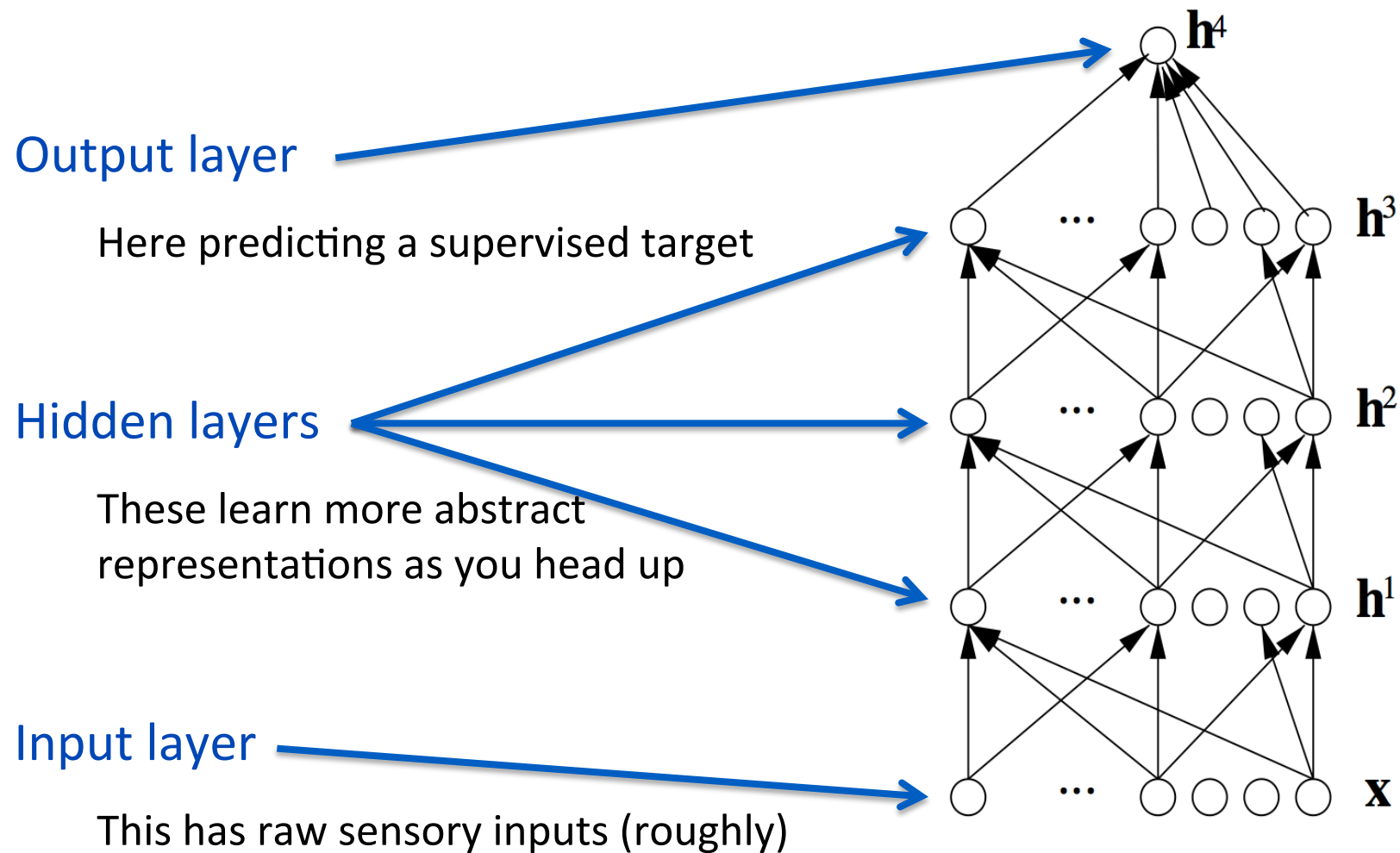
Deep learning algorithms attempt to learn multiple levels of representation of increasing complexity/abstraction

A system with such a sequence is a **deep architecture**

The vast majority of such work has explored **deep belief networks (DBNs)** which are Markov Random Fields with multiple layers

Several other variants of deep learning have seen a revival due to improved optimization methods of multiple layer neural networks

A Deep Architecture



Part 1.1: The Basics

Six Reasons to Explore Deep Learning

#1 Learning features, not just handcrafting them

Most NLP systems use very carefully hand-designed features and representations

Many of us are very experienced – and good – at such feature design

In this world, “machine learning” reduces mostly to linear models (including CRFs) and nearest-neighbor-like features/models (including n-grams)

Hand-crafting features is time-consuming and brittle; the features are often over-specified and incomplete

How can we automatically learn good features?

We need to move the scope of machine learning beyond hand-crafted features and simple ML

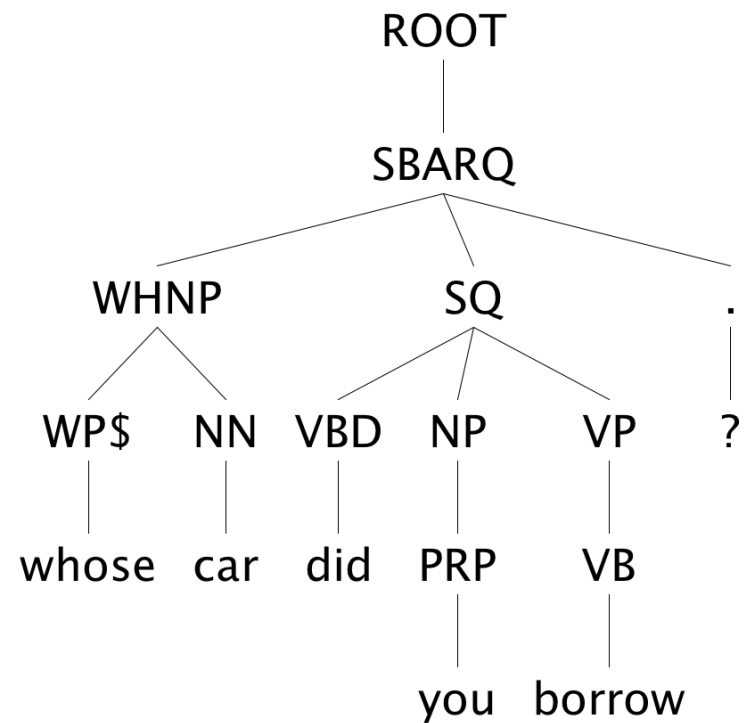
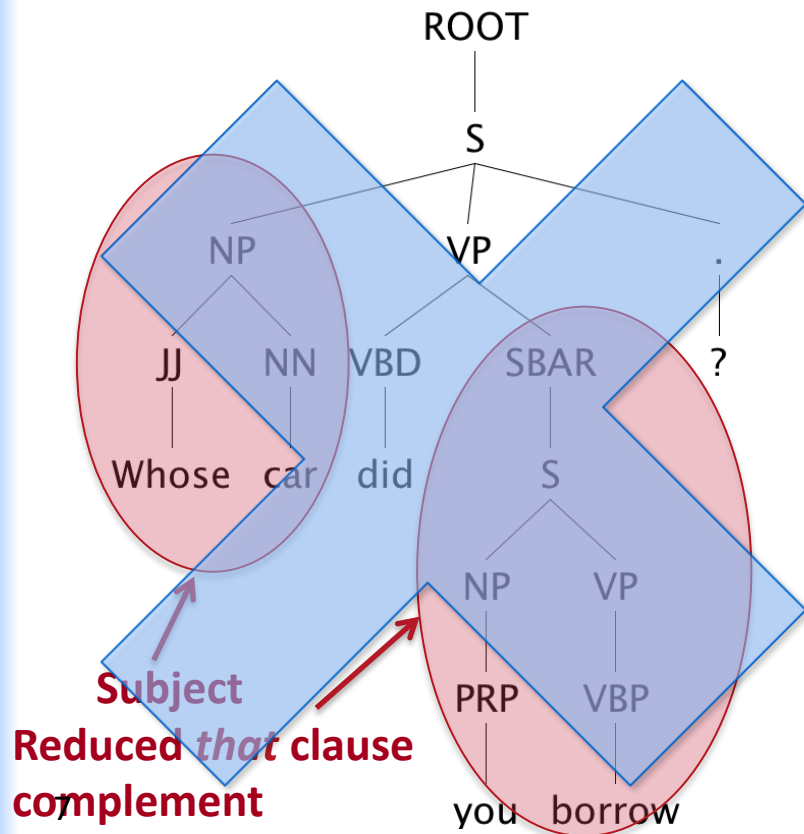
Humans develop representations to enable learning and reasoning; our computers should do the same

Deep learning provides a mechanism for learning good features and representations

Handcrafted features can be combined with learned features, or new more abstract features learned on top of handcrafted features

#2 The need for distributed representations

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



#2 The need for distributed representations

Learned word representations, which model similarities, help all NLP tasks enormously

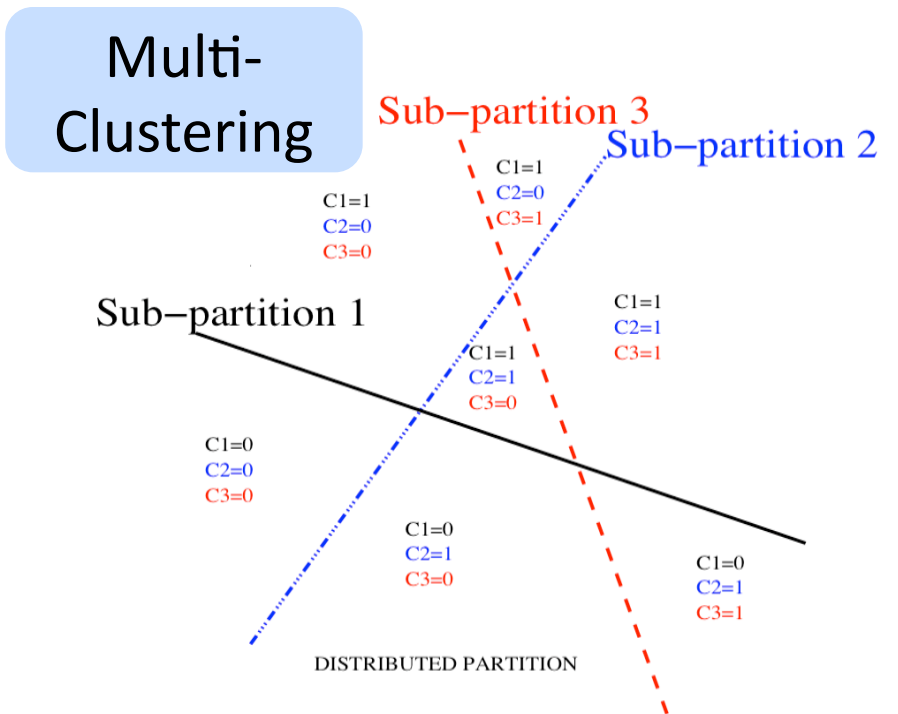
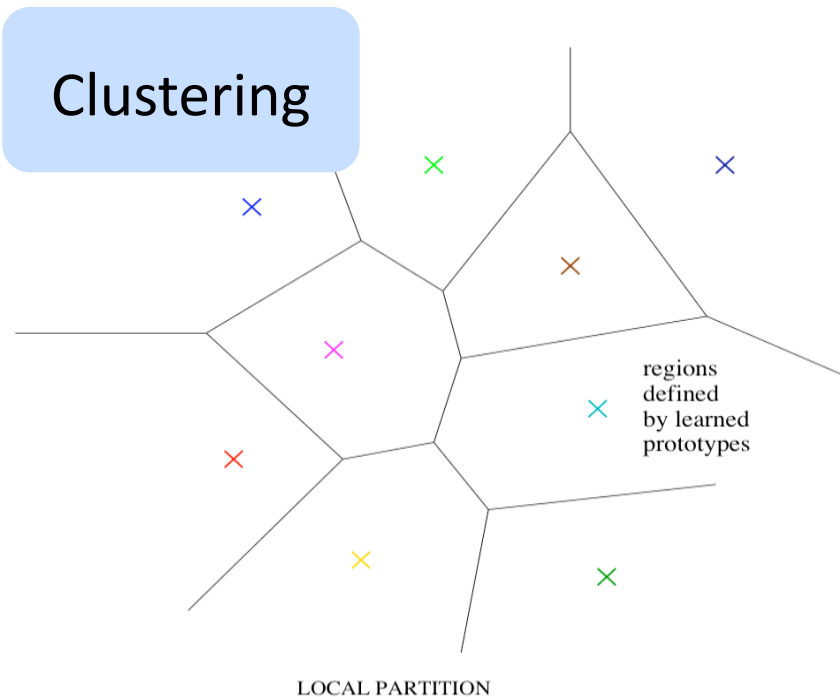
E.g., **distributional similarity** based word clusters greatly help all applications

+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)

E.g., paraphrasing, word-sense disambiguation, language modeling...

#2 The need for distributed representations



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

#3 Unsupervised feature and weight learning

Today, most practical, good machine learning methods require labeled training data

But almost all **data is unlabeled**

The brain needs to learn about 10^{14} connection weights

... in about 10^9 seconds

Labels cannot possibly provide enough information

(unless the weights were highly redundant)

Most information acquired in an **unsupervised** fashion

#4 Learning multiple levels of representation

There is theoretical and empirical evidence in favor of multiple levels of representation

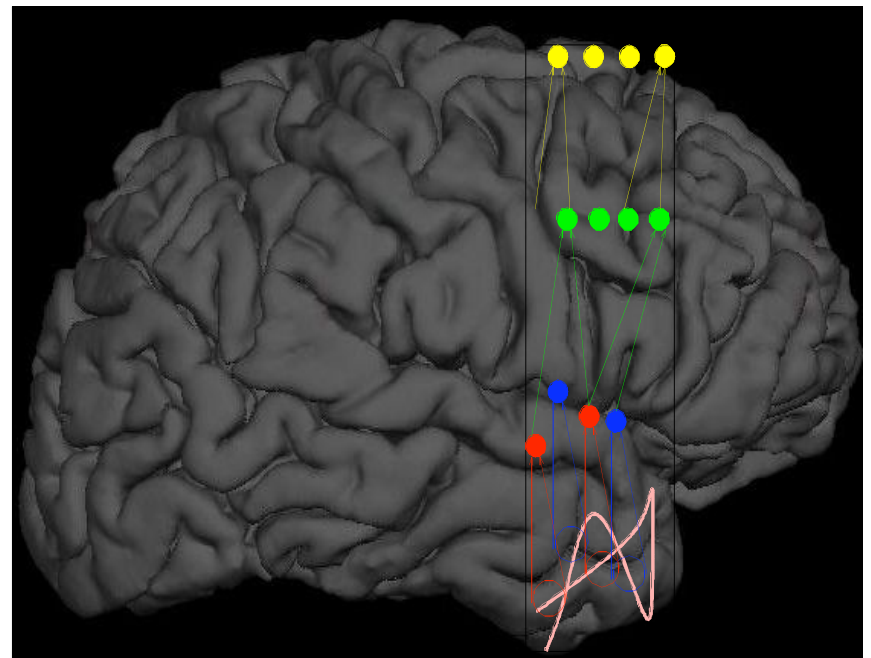
Exponential gain for some families of functions

Biologically inspired learning

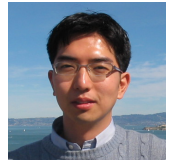
Brain has a deep architecture

Cortex seems to have a generic learning algorithm

Humans first learn simpler concepts and then compose them to more complex ones

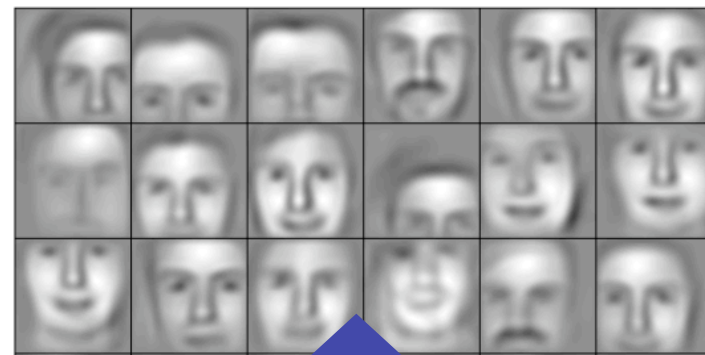


#4 Learning multiple levels of representation

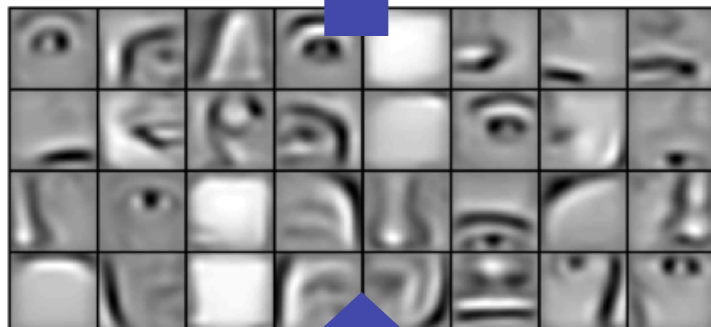


[Lee, Largman, Pham & Ng, NIPS 2009]

Successive model layers learn deeper intermediate representations



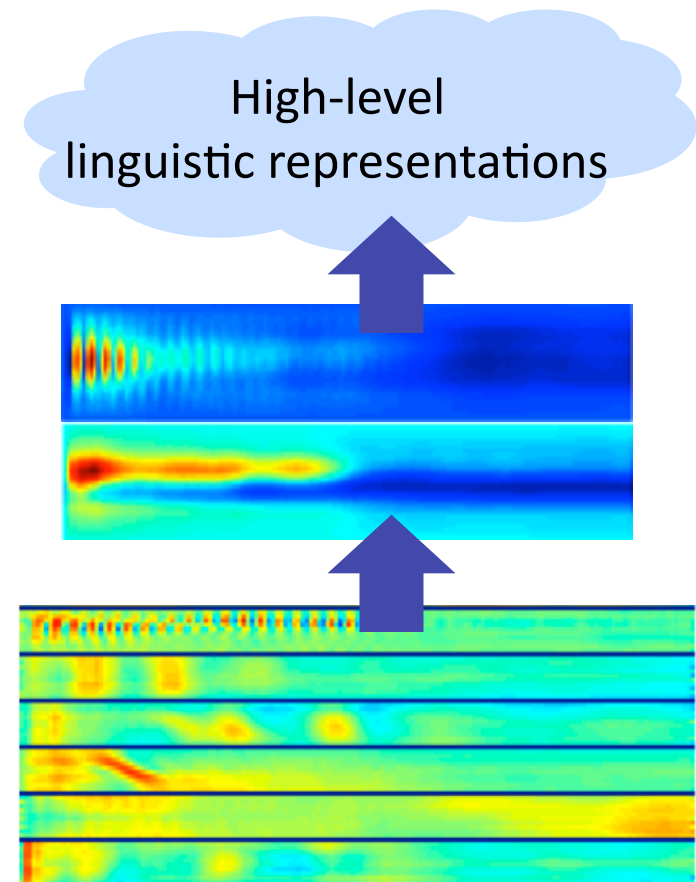
Layer 3



Layer 2

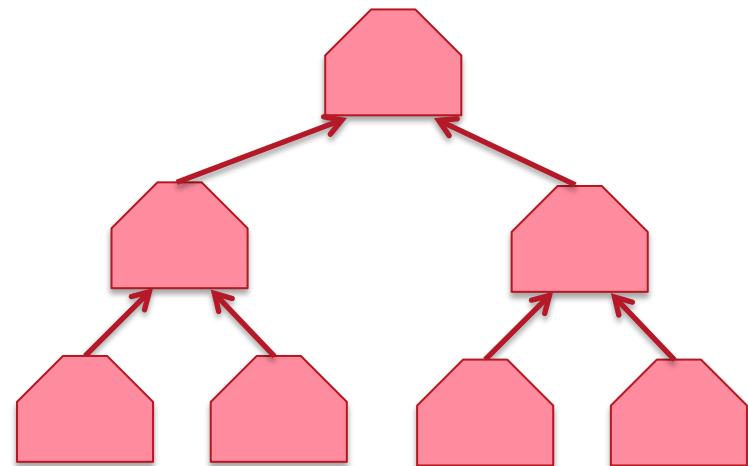
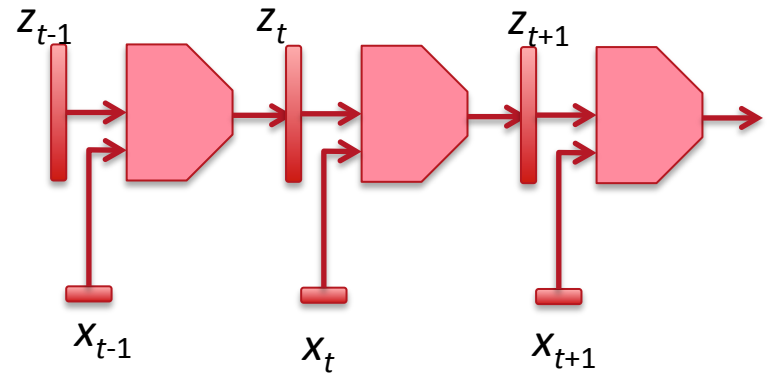


Layer 1



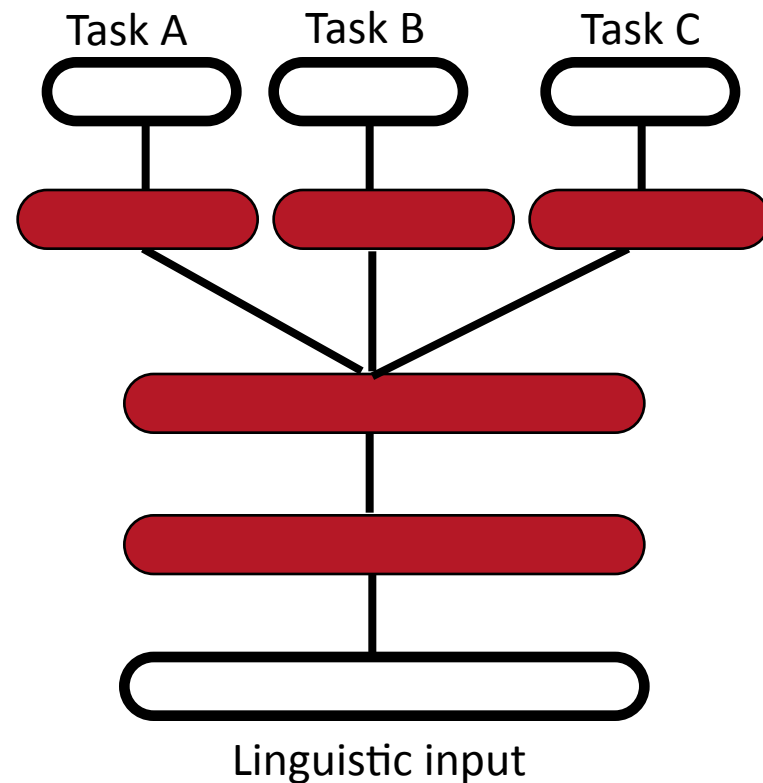
#4 Handling the recursivity of human language

- Human languages, ideas, and artifacts are composed from simpler components
- Recursion: the same operator (same parameters) is applied repeatedly on different states/components of the computation



#4 Using a deep architecture

- Deep architectures learn good intermediate representations that can be shared across tasks
- Insufficient depth of representation can be exponentially inefficient
- Multiple levels of latent variables allow combinatorial sharing of statistical strength



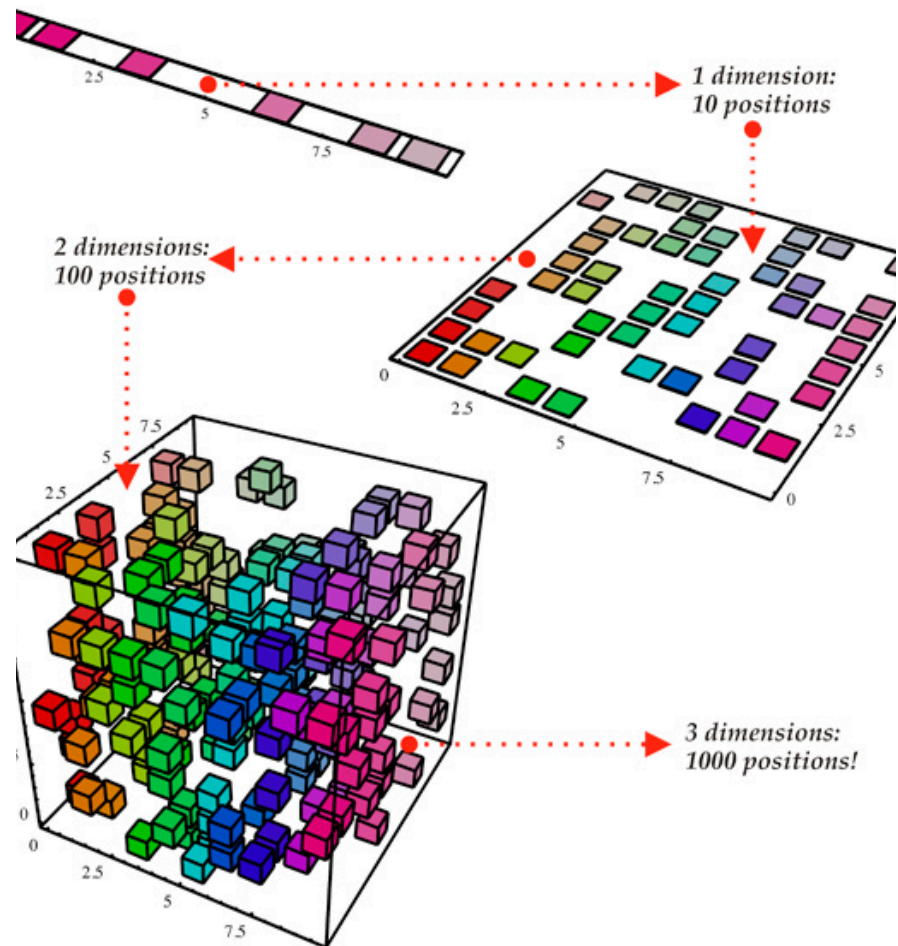
#5 The curse of dimensionality

To generalize locally (e.g., nearest neighbors), we need representative examples for all relevant variations!

We cannot hope to learn from raw signals or random bases

Classical solutions:

- Manual feature design
- Hoping for a smooth enough target function (such as assuming a linear model) to allow simple prediction



#5 Solving the curse of dimensionality

Three approaches dominate recent work:

(Log-)Linear models

Flexibility achieved by adding more manually engineered features

Kernel methods

The basis functions are associated with data points, limiting complexity

Selection of a subset of data points further limits complexity

Output is linear in the output of near-neighbor detectors (kernel fn)

Neural networks

Parameterize and learn the kernel

Can be nested to create a hierarchy of abstraction levels

#5 Solving the curse of dimensionality

We need to build **compositionality** into our ML models

Just as human languages exploit compositionality to give representations and meanings to complex ideas

Exploiting compositionality gives an exponential gain in representational power

Distributed representations / embeddings: **feature learning**

Deep architecture: **multiple levels of feature learning**

#6 Why now?

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

Before 2006 training deep architectures was **unsuccessful**
(except for convolutional neural nets when used by people with French names)

What has changed?

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

#6 Deep NLP Learning models already work well

Neural Language Model

[Mikolov et al. Interspeech 2011]



Model	Eval WER (WSJ task)
KN5 Baseline	17.2
Discriminative LM	16.9
Recurrent NN combination	14.4

State of the art performance for POS, NER, Chunking

[Collobert et al. 2011]

Task		Benchmark	SENNA
Part of Speech (POS)	(Accuracy)	97.24 %	97.29 %
Chunking (CHUNK)	(F1)	94.29 %	94.32 %
Named Entity Recognition (NER)	(F1)	89.31 %	89.59 %

#6 Deep NLP Learning models already work well

Sentiment analysis of
Sentences [Socher et al. 2011a]

Paraphrase Detection [Socher et
al. 2011b]

Relation Classification [Socher et
al. 2012]

	Bench- mark	Recursive NN
MPQA sentiment	86.1	86.4
MSFT Paraphrase	82.3	83.6
SemEval Relation	82.2	82.4

#6 Deep NLP Learning models already work well

MSR MAVIS Speech System

Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition (Dahl, Yu, Deng & Acero TASLP 2012)

Conversational Speech Transcription Using Context-Dependent Deep Neural Networks (Seide, Li & Yu, Interspeech 2011)

acoustic model & training	recognition mode	RT03S		Hub5'00	voicemails		tele-
		FSH	SW	SWB	MS	LDC	conf
GMM 40-mix, ML, SWB 309h	single-pass SI	30.2	40.9	26.5	45.0	33.5	35.2
GMM 40-mix, BMMI, SWB 309h	single-pass SI	27.4	37.6	23.6	42.4	30.8	33.9
CD-DNN 7 layers x 2048, SWB 309h, this paper (rel. change GMM BMMI → CD-DNN)	single-pass SI	18.5 (-33%)	27.5 (-27%)	16.1 (-32%)	32.9 (-22%)	22.9 (-26%)	24.4 (-28%)
GMM 72-mix, BMMI, Fisher 2000h	multi-pass adaptive	18.6	25.2	17.1	-	-	-

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

#6 Why now?

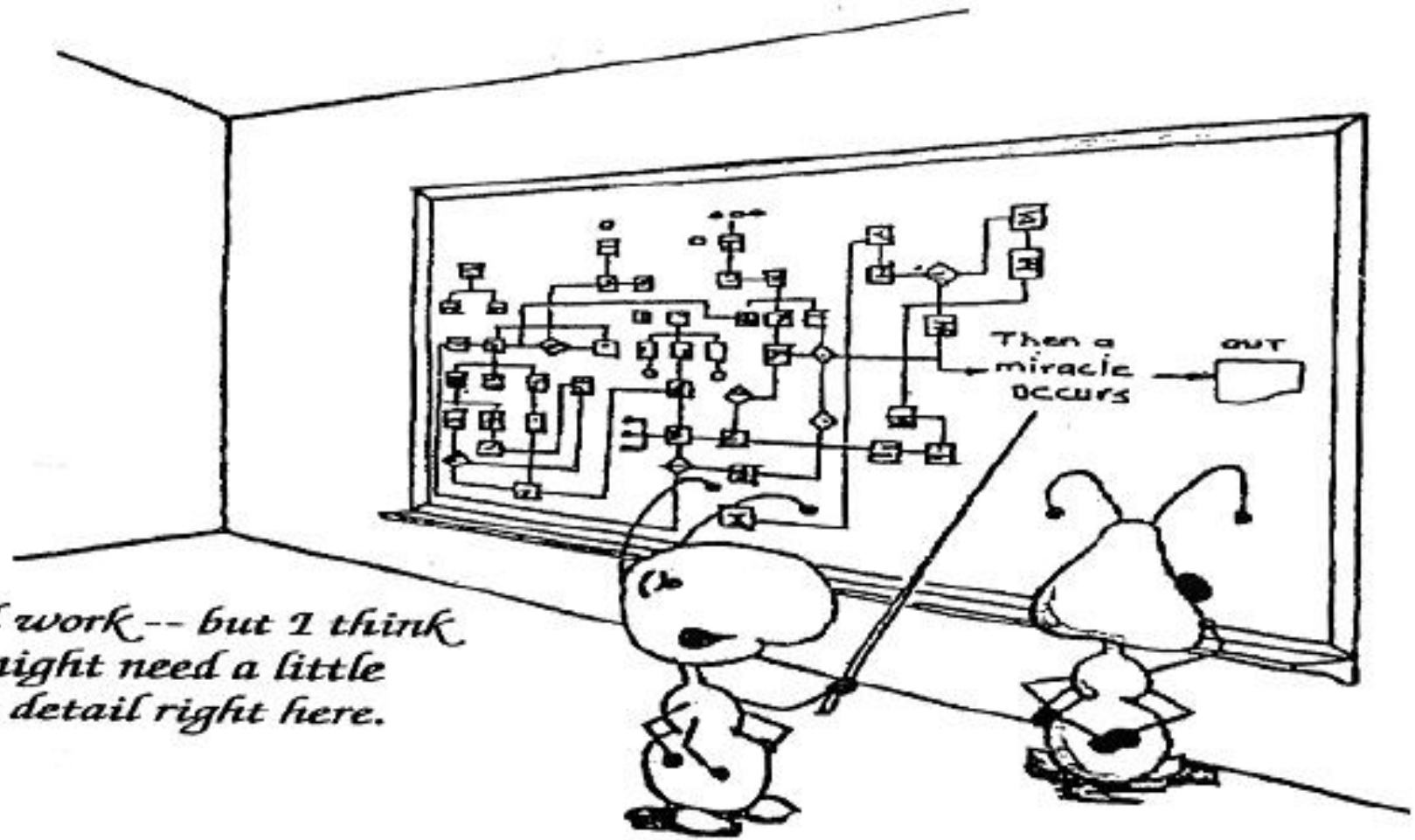
Deep learning models can now be very fast *in some circumstances*

- SENNA can do POS or NER extremely fast (16x to 122x) compared to other SOTA taggers, using 25x less memory

Changes in computing technology favor deep learning

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

These trends hold even more strongly when using 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. Theory: Backpropagation Through Structure
 4. Recursive Autoencoders
 5. Application to Sentiment Analysis and Paraphrase Detection
 6. Compositionality Through Recursive Matrix-Vector Spaces
 7. Relation classification
3. Applications, Discussion, and Resources

Outline of the Tutorial

1. The Basics
2. Recursive Neural Networks
3. Applications, Discussion, and Resources
 1. Applications
 1. Neural language models
 2. Structured embedding of knowledge bases
 3. Assorted other speech and NLP applications
 2. Resources (readings, code, ...)
 3. Discussion
 1. Tricks of the trade
 2. Limitations, advantages, future directions

Part 1.2: The Basics

From logistic regression to neural nets

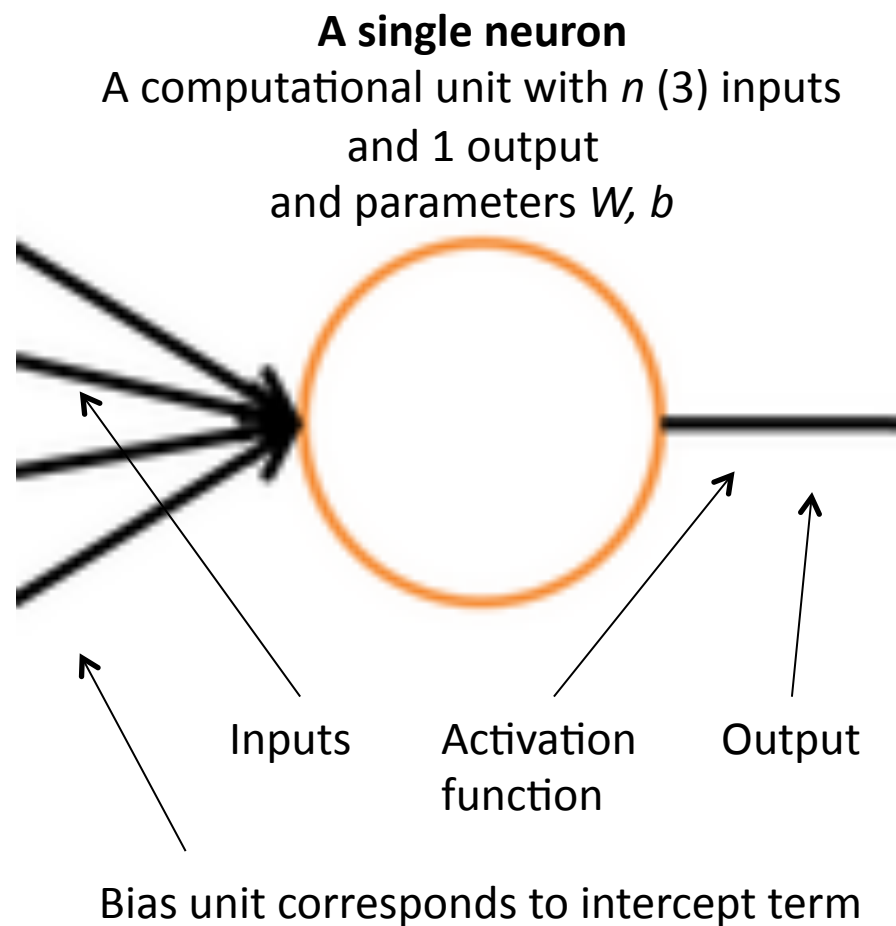
Demystifying neural networks

Neural networks come with their own terminological baggage (of “neurons”, “activation functions”, and “weight decay”)

... just like SVMs

But if you understand how maxent/logistic regression models work

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



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: $P(c | d, \lambda) = \frac{e^{\lambda \cdot f(c, d)}}{\sum_{c'} e^{\lambda \cdot f(c', d)}}$
- Such a classifier is sometimes used as-is in a neural network
 - Often as the top layer (“a softmax layer”)
- But for now we’ll derive a two-class logistic model for one neuron

From Maxent Classifiers to Neural Networks

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

- Make two class:

$$P(c_1 | d, \lambda) = \frac{e^{\lambda \cdot f(c_1, d)}}{e^{\lambda \cdot f(c_1, d)} + e^{\lambda \cdot f(c_2, d)}} = \frac{e^{\lambda \cdot f(c_1, d)}}{e^{\lambda \cdot f(c_1, d)} + e^{\lambda \cdot f(c_2, d)}} \cdot \frac{e^{-\lambda \cdot f(c_1, d)}}{e^{-\lambda \cdot f(c_1, d)}}$$

$$= \frac{1}{1 + e^{\lambda \cdot [f(c_2, d) - f(c_1, d)]}} = \frac{1}{1 + e^{-\lambda \cdot x}} \quad \text{for } x = f(c_1, d) - f(c_2, d)$$

$$P(c_2 | d, \lambda) = \frac{e^{\lambda \cdot f(c_2, d)}}{e^{\lambda \cdot f(c_1, d)} + e^{\lambda \cdot f(c_2, d)}} = \frac{e^{\lambda \cdot f(c_2, d)}}{e^{\lambda \cdot f(c_1, d)} + e^{\lambda \cdot f(c_2, d)}} \cdot \frac{e^{-\lambda \cdot f(c_1, d)}}{e^{-\lambda \cdot f(c_1, d)}}$$

$$= \frac{e^{\lambda \cdot [f(c_2, d) - f(c_1, d)]}}{1 + e^{\lambda \cdot [f(c_2, d) - f(c_1, d)]}} = \frac{e^{-\lambda \cdot x}}{1 + e^{-\lambda \cdot x}}$$

From Maxent Classifiers to Neural Networks

- Output of one class: $P(c_1 | x, \lambda) = \frac{1}{1 + e^{-\lambda \cdot x}}$
- We often have an “always on” feature for a class, which gives a class prior. We can separate it out as a bias term:

$$P(c_1 | x, \lambda) = \frac{1}{1 + e^{-[\lambda \cdot x + b]}}$$

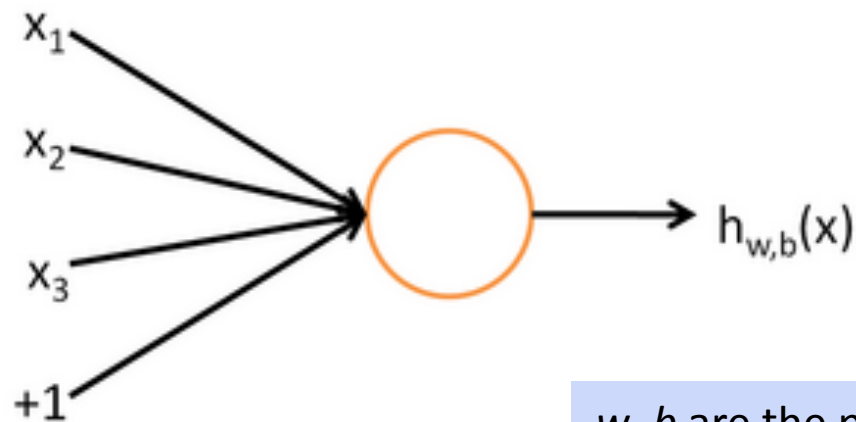
- Or we can have x_1 be an always-on input
- We separate things into the vector dot product and applying a non-linearity. Let $f(z) = 1/(1 + \exp(-z))$, the logistic function.
- Then:

$$P(c_1 | x, \lambda) = f(\lambda \cdot x + b)$$

This is exactly what a neuron computes

$$h_{w,b}(x) = f(w \cdot x + b)$$

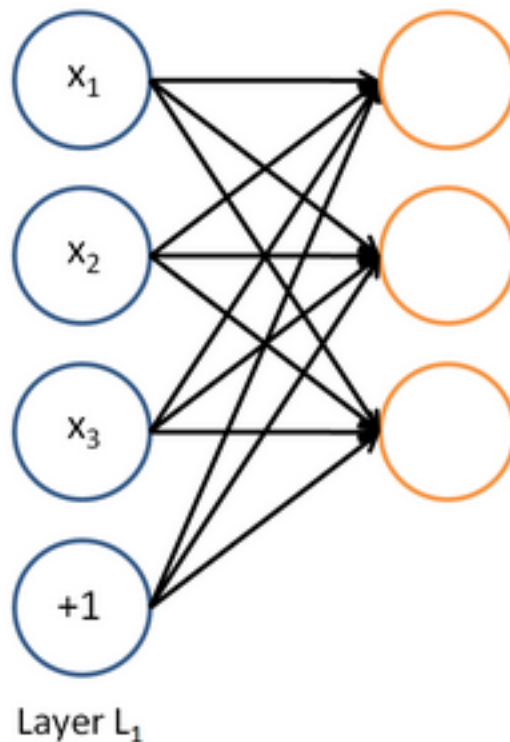
$$f(z) = \frac{1}{1 + e^{-z}}$$



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

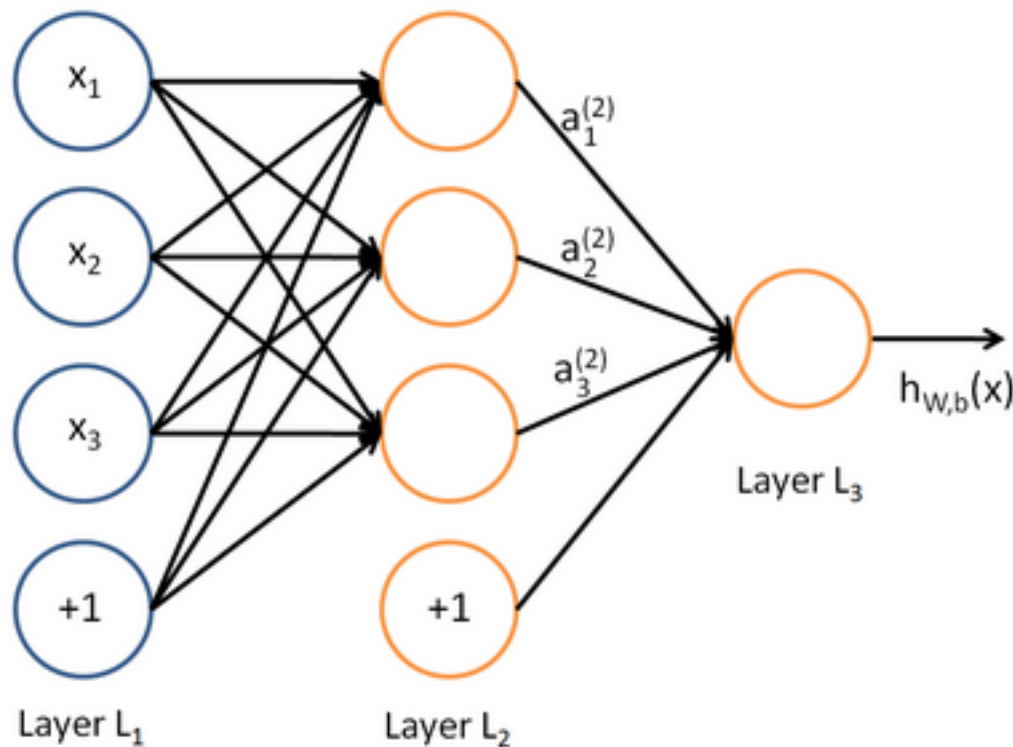
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



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

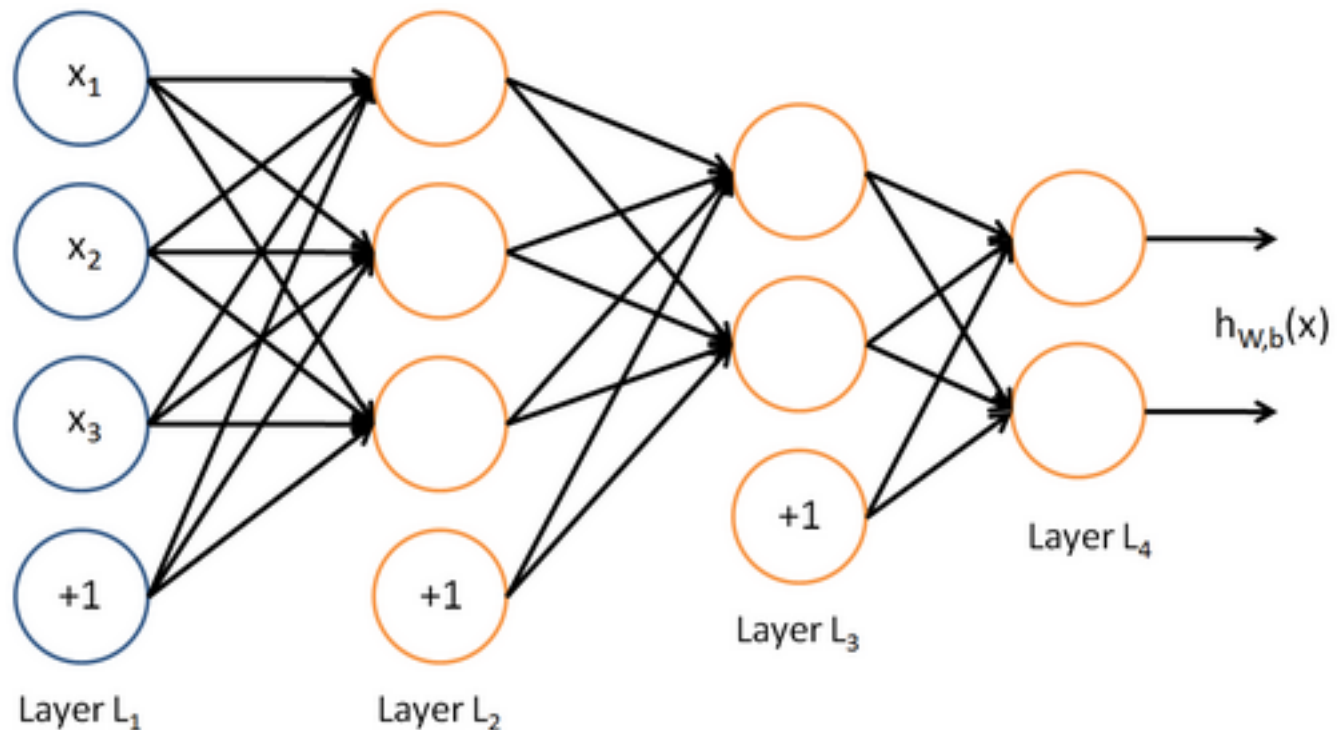
... which we can feed into another logistic regression function



and it is the training criterion that will decide what those intermediate binary target variables should be, so as to make a good job of 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.

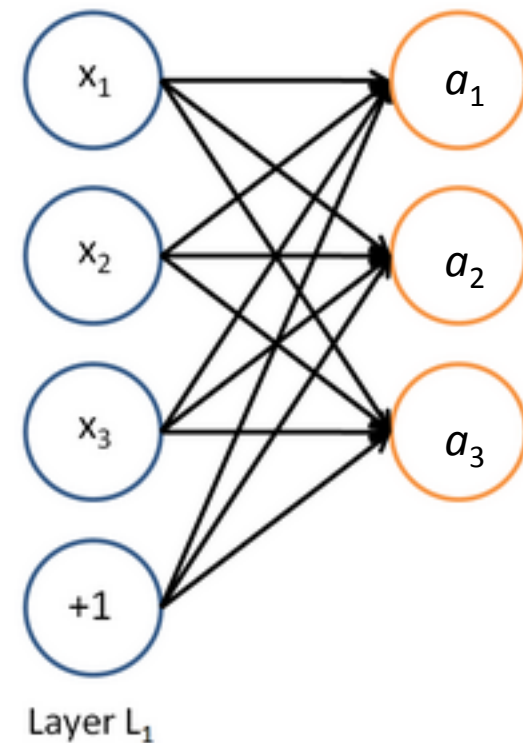
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

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 layer neural net, we can train the model just like a logistic regression model
 - We can do stochastic gradient descent
 - We can use fancier methods, as we commonly do for maxent models like conjugate gradient or L-BFGS
- For a multilayer net it is potentially more complex because the internal (“hidden”) logistic units make the function non-convex ... just as for hidden CRFs [Quattoni et al. 04, Gunawardana et al. 05]
 - But we use the same ideas
 - This leads into “backpropagation”, which we cover later

Non-linearities: Why they're needed

- For logistic regression, they're motivated by mapping to probabilities: $[0,1]$
- Here, they're motivated by being able to do function approximation
 - Without non-linearities, neural networks can't do anything more than a linear transform: extra layers could just be compiled down into a single linear transform
 - The probabilistic interpretation for hidden units is usually unnecessary except in the Boltzmann machine models.

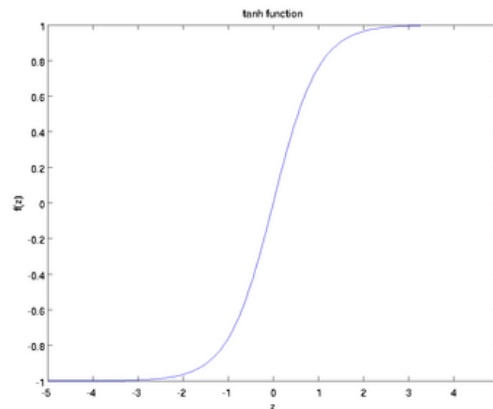
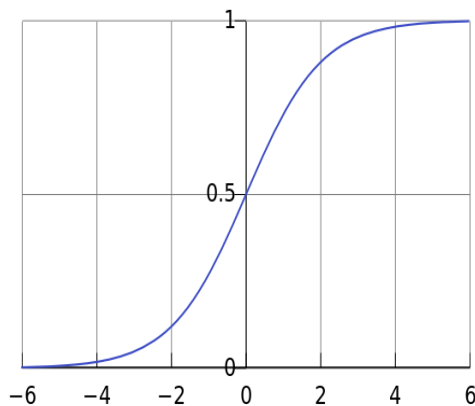
Non-linearities: What's used

logistic (“sigmoid”)

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

tanh

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



tanh is just a rescaled and shifted sigmoid ($2 \times$ as steep, $[-1,1]$):

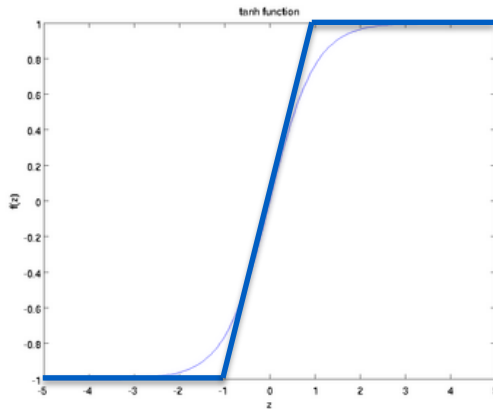
$$\tanh(z) = 2\text{logistic}(2z) - 1$$

tanh is what is most used and often performs best for deep nets
[Glorot and Bengio *AISTATS* 2010]

Non-linearities: There are various other choices

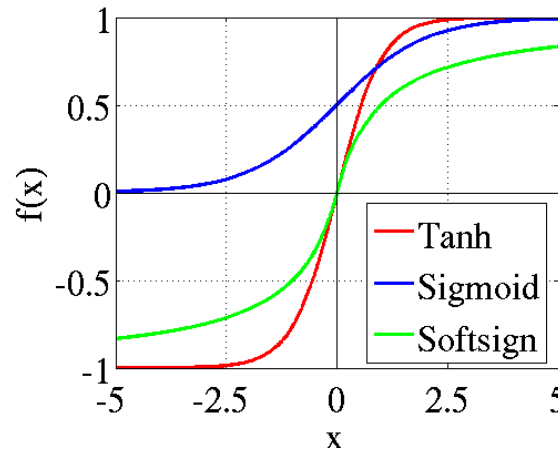
hard tanh

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



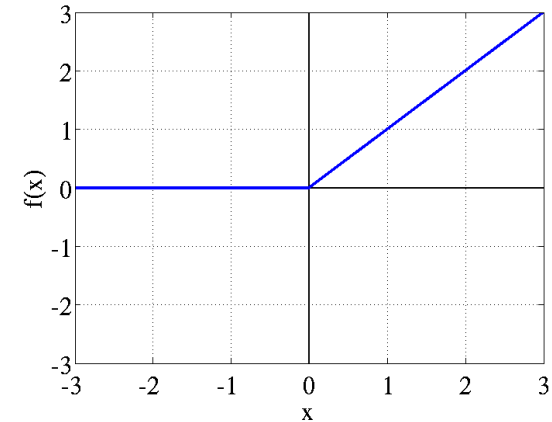
soft sign

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



rectifier

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



- hard tanh is a mathematically awkward but computationally cheap tanh
- [Glorot and Bengio AISTATS 2010] discuss uses of softsign and rectifier

Summary: Knowing the meaning of words!

You should now understand the basics and how they relate to other models you use

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

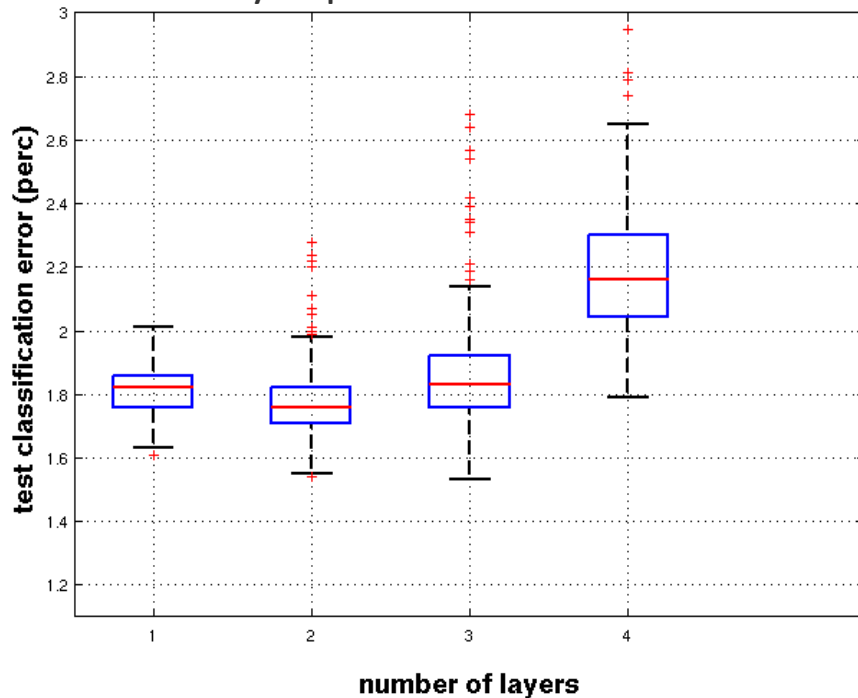
Effective deep learning became possible through unsupervised pre-training

[Erhan et al., JMLR 2010]

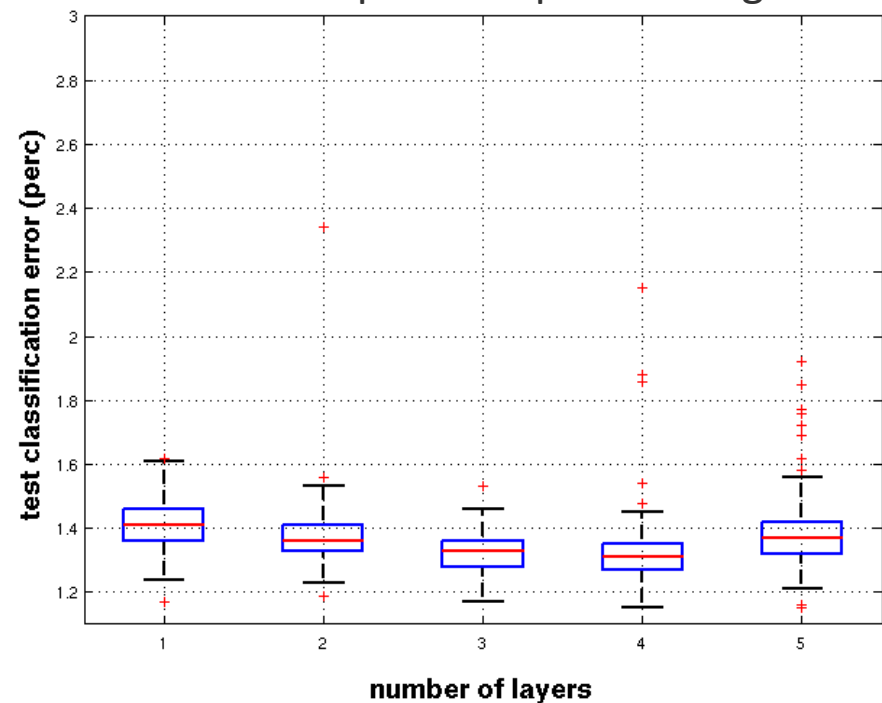


(with RBMs and Denoising Auto-Encoders)

Purely supervised neural net



With unsupervised pre-training



Part 1.3: The Basics

Word Representations

Word representations

What is the main source of information in NLP?

Words.

How do most systems handle words?

Not very well.

The standard word representation

The vast majority of NLP work regards words as atomic symbols:
hotel, conference, walk

Regardless of whether it is rule-based NLP or statistical NLP

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) – 3M (Google 1T)

We call this a “one-hot” representation

Can one do better?

The standard word representation

A symbolic representation looks ridiculous as a vector ...

But it's what vector space model IR actually uses (conceptually)

$$\begin{array}{l} \text{motel} \quad [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] \\ \text{AND hotel} \quad [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ \quad \quad \quad = \quad 0 \end{array}$$

How does conventional IR try to solve this problem?

Query expansion (synonyms)

EXPAND(motel) [0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0]

AND hotel [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Softer methods like pseudo-relevance feedback

PRF(motel) [0 0 0 0 0.1 0 0 0.2 0 0 1 0 0 0 0.4]

AND hotel [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Making representations for whole documents, which are denser

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

AND doc vec [0 0 1 0 3 0 0 2 0 0 1 1 0 0 5]

Dealing with the Sparsity of Language

This problem of seeing connections between words only becomes worse when we move to dealing with relations between word pairs and triples

Language modeling methods of backoff and interpolation provide a poor way of dealing with a lack of information about word similarity

We need a method that generalizes based on other words that are semantically and syntactically similar

Distributional similarity based representations

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

One of the most successful ideas of modern statistical NLP

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

(J. R. Firth 1957: 11)

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* ↗

Distributional similarity based representations

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

(J. R. Firth 1957: 11)

Methods vary in the kind of context they exploit

Using a word or a few words to the left and/or right gives largely syntactic word classes, with semantic coloring

players \approx musicians

Using the whole document gives much more “topical” semantic similarity

inning \approx homered

Class-based (hard clustering) word representations

These models learn word classes based on distributional information, roughly as a class HMM

- Brown clustering (Brown et al. 1992)
- Exchange clustering (Martin et al. 1998, Clark 2003)

Words are similar if they are put in the same class

This is a simplistic notion of similarity, but it is sufficient to give a very useful desparsification of word data

- It's not what we're talking about today, but they are an example of unsupervised pre-training
 - And, in practice, these are still great features!

Soft clustering word representations

These models learn for each cluster/dimension/factor a distribution over words as probabilities or strengths

- Latent Semantic Analysis (LSA/LSI)
 - Random projections
 - Latent Dirichlet Analysis (LDA)
 - HMM clustering
- } Vector space
- } Probabilistic

Broadly, neural word embeddings are in this space, combining vector space semantics with the prediction of probabilistic models
(Bengio et al. 2003, Collobert & Weston 2008, Turian et al. 2010)

Distributed Representations

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

$$\textit{linguistics} = \begin{bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

Some approaches aim for some sparsity in the vector.
In probabilistic approaches, all the numbers are non-negative.

Distributed Representations

These are **distributed representations**: each word has a weighting in each dimension or cluster

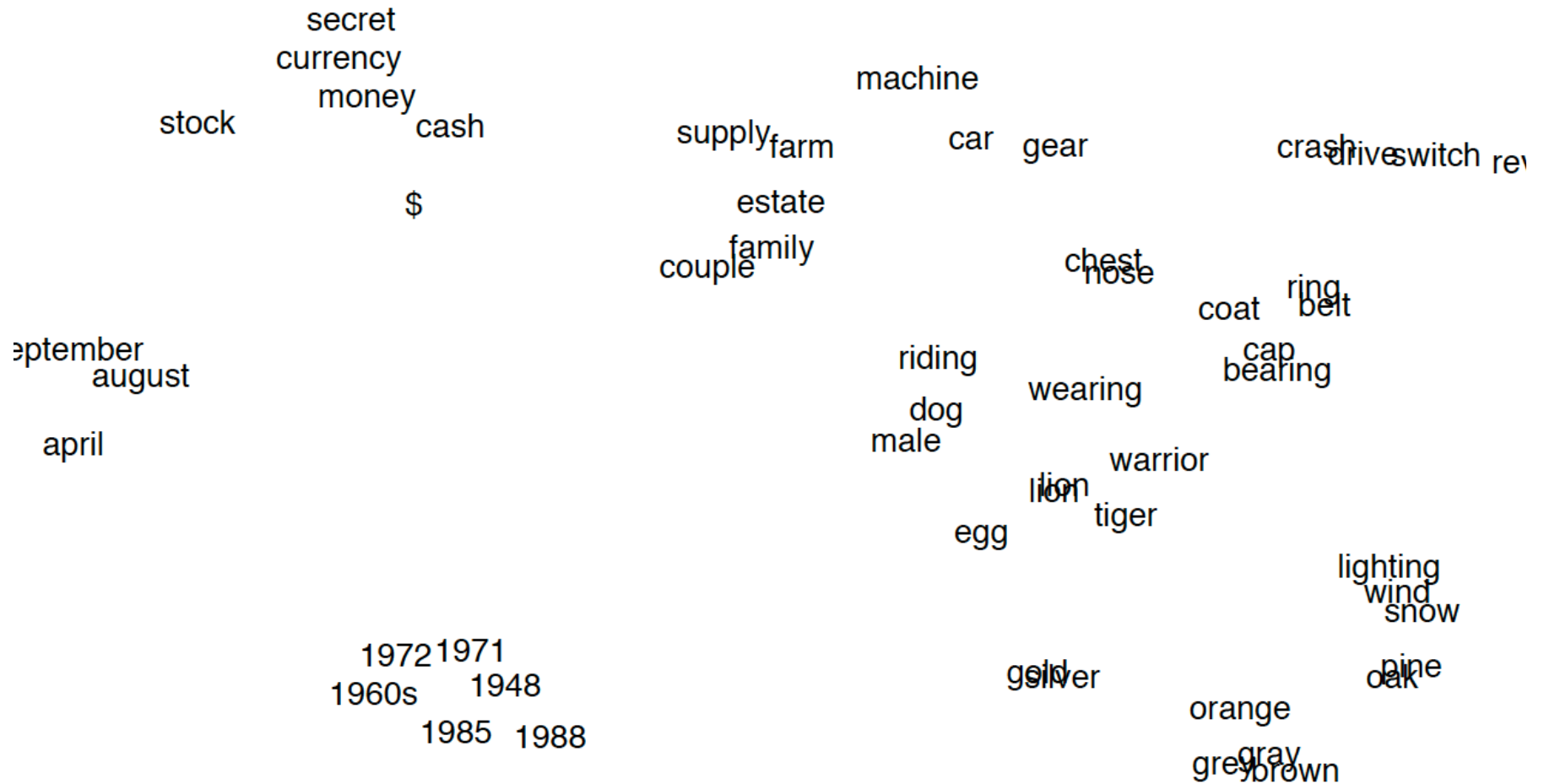
In contrast to the the “atomic” or “localist” representations employed in most of NLP, a distributed representation is one in which “each entity is represented by a pattern of activity distributed over many computing elements, and each computing element is involved in representing many different entities”

(Hinton “Distributed representations” CMU-CS-84-157, 1984)

Neural word embeddings



Neural word embeddings



Neural word embeddings

Most similar words to a few words

Spain
France
England Italy
Germany
Denmark

Jesus
God Christ
Sin Prayer

France	Jesus	XBOX	Reddish	Scratched
Spain	Christ	Playstation	Yellowish	Smashed
Italy	God	Dreamcast	Greenish	Ripped
Russia	Resurrection	PS###	Brownish	Brushed
Poland	Prayer	SNES	Bluish	Hurled
England	Yahweh	WH	Creamy	Grabbed
Denmark	Josephus	NES	Whitish	Tossed
Germany	Moses	Nintendo	Blackish	Squeezed
Portugal	Sin	Gamecube	Silvery	Blasted
Sweden	Heaven	PSP	Greyish	Tangled
Austria	Salvation	Amiga	Paler	Slashed

(Collobert & Weston, *ICML* 2008)

Advantages of the neural word embedding approach

Compared to a method like LSA:

- A neural embedding can learn higher-level features/abstractions (beyond the word)
- Learning such an embedding forces a representation on the words themselves that is better and more meaningful
 - Because everything is trained together
- By adding supervision from either one task or multiple tasks simultaneously, we can improve the representation of the words for handling language analysis tasks

Part 1.4: The Basics

Unsupervised word vector Learning

A neural network for learning word vectors

(Collobert & Weston 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

- Idea: A word and its context is a positive training sample, a random word in that same context is a negative training sample.
- $\text{score}(\text{cat chills on a mat}) > \text{score}(\text{cat chills Jeju a mat})$
- How to 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 o by L : $x = Lo$

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 of a Neural Network

- A single layer is a combination of a linear layer and a nonlinearity:

$$z = Wx + b$$

$$a = f(z)$$

- The neural activations can then be used to compute some function.
- For instance, the score we care about:

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

Summary: Feed-forward Computation

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

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

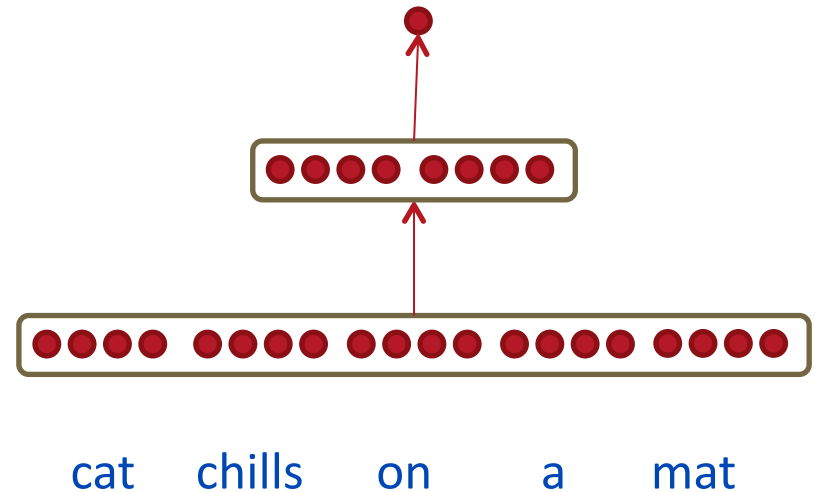
$$s = W_{score}^T a$$

$$z = Wx + b$$

$$a = f(z)$$

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

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



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

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

Training with Backpropagation

$$J = \max(0, 1 - s + s_c) \quad s = W_{score}^T f(Wx + b)$$

- Assuming cost is > 0 , we compute the derivatives of s and s_c wrt all the involved variables: W_{score} , W , b , x

$$\frac{\partial s}{\partial W_{score}} = \frac{\partial}{\partial W_{score}} W_{score}^T a \quad \frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial s}{\partial W_{score}} = a$$

Training with Backpropagation

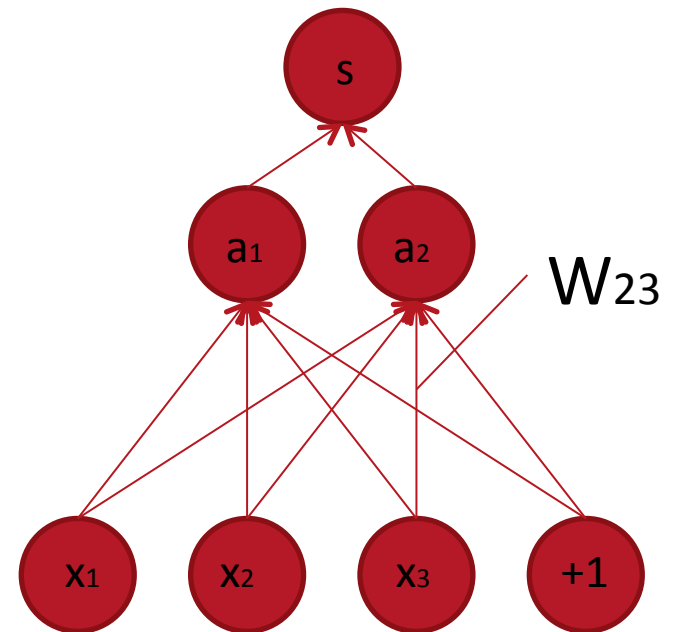
$$s = W_{score}^T f(Wx + b)$$

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

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

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

$$\frac{\partial}{\partial W_{ij}} W_{score}^T a$$



Training with Backpropagation

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

- Derivative of weight W_{ij} :

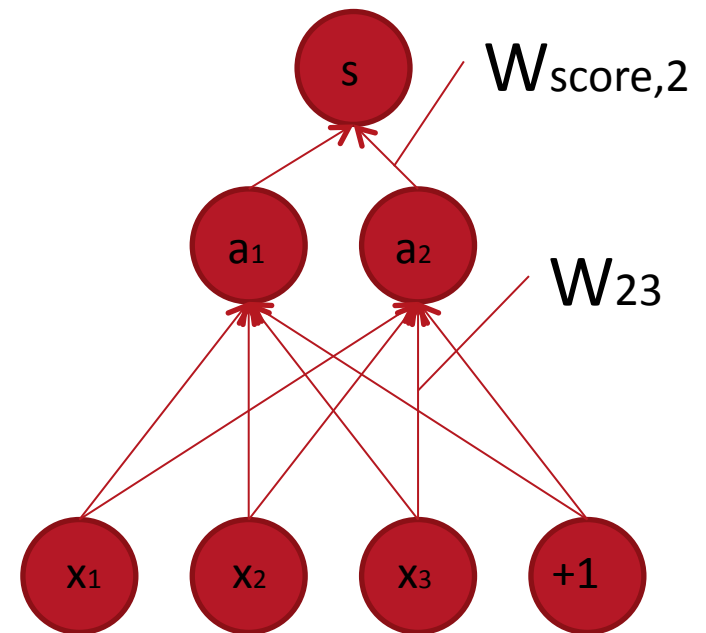
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

$$\frac{\partial}{\partial W_{ij}} W_{score}^T a \rightarrow \frac{\partial}{\partial W_{ij}} W_{score,i} a_i$$

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

$$a_i = f(z_i)$$

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



Training with Backpropagation

- Derivative of single weight W_{ij} : $W_{score,i} \frac{\partial}{\partial W_{ij}} a_i$

$$= W_{score,i} f'(z_i) \frac{\partial W_{i \cdot x} + b_i}{\partial W_{ij}}$$

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

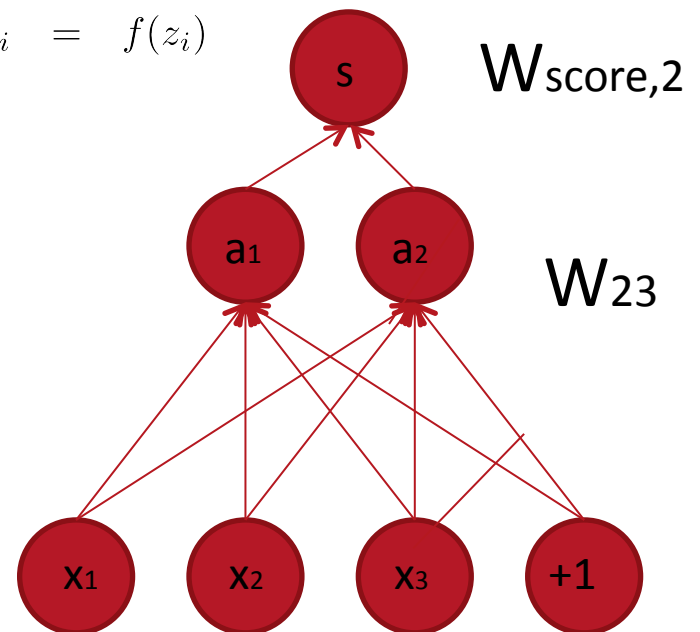
$$= W_{score,i} f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k$$

$$a_i = f(z_i)$$

$$= \underbrace{W_{score,i} f'(z_i)}_{\delta_i} x_j$$

δ_i
Local error signal

x_j
Local input signal



Training with Backpropagation

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

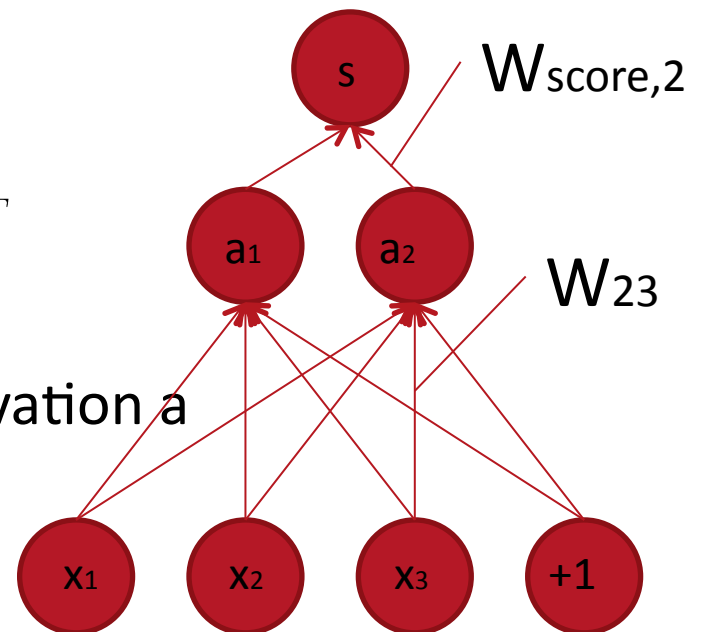
$$\frac{\partial J}{\partial W_{ij}} = \underbrace{W_{score,i} f'(z_i)}_{\delta_i} x_j$$

$$= \delta_i x_j$$

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

$$a_i = f(z_i)$$

- 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

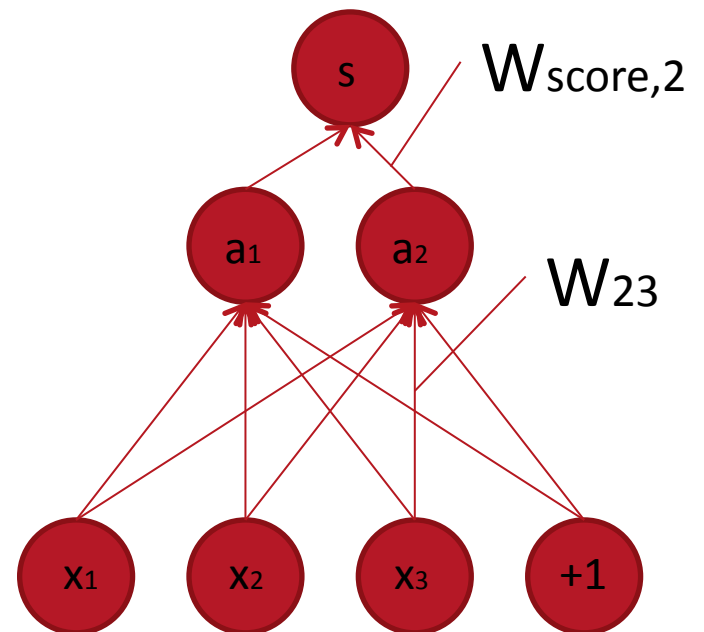


Training with Backpropagation

- For biases b , we get:

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

$$\begin{aligned} z_i &= W_{i \cdot x} + b_i = \sum_{j=1}^3 W_{i,j} x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



Training with Backpropagation

- We just almost learned the backpropagation rule by carefully taking derivatives and using the chain rule!
- The only remaining trick is to re-use derivatives that we computed once, for lower layers.
- Let's apply that idea for the last derivatives of this 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:

$$\begin{aligned}
 \frac{\partial s}{\partial x_j} &= \sum_{i=1}^2 \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
 &= \sum_{i=1}^2 \frac{\partial W_{score}^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
 &= \sum_{i=1}^2 W_{score,i}^T \frac{\partial f(W_i \cdot x + b)}{\partial x_j} \\
 &= \sum_{i=1}^2 \underbrace{W_{score,i}^T f'(W_i \cdot x + b)} \frac{\partial W_i \cdot x}{\partial x_j} \\
 &= \sum_{i=1}^2 \delta_i W_{ij} \\
 &= \delta^T W_{\cdot j}
 \end{aligned}$$

Training with Backpropagation

- So in the last line, we re-used parts of the derivative of a computation from a layer above.
- In the next section, we will see how – with more deeper layers – even more of the computation of higher layers can be re-used in lower layers

Part 1.5: The Basics

Backpropagation Training

Back-Prop

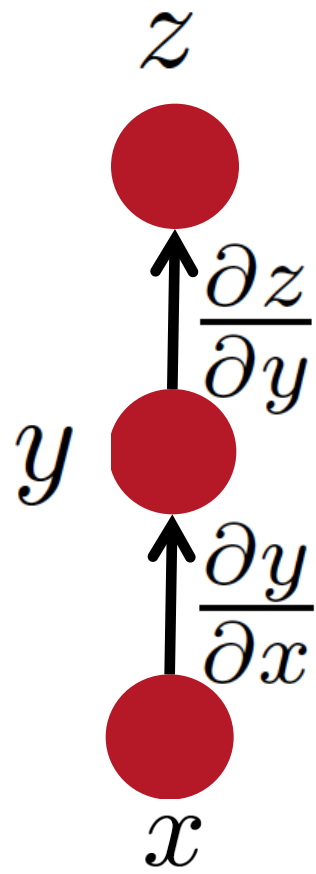
- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivatives 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



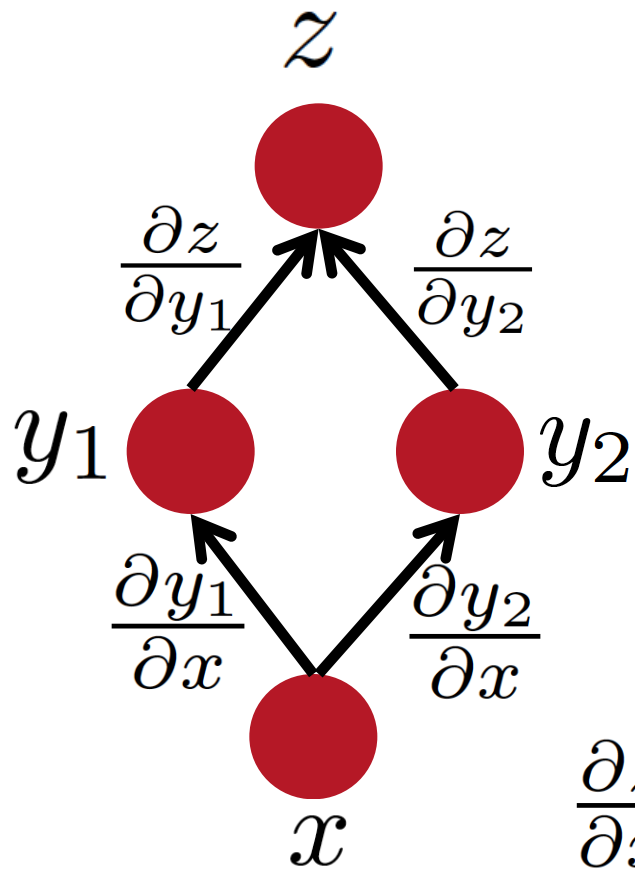
$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

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

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

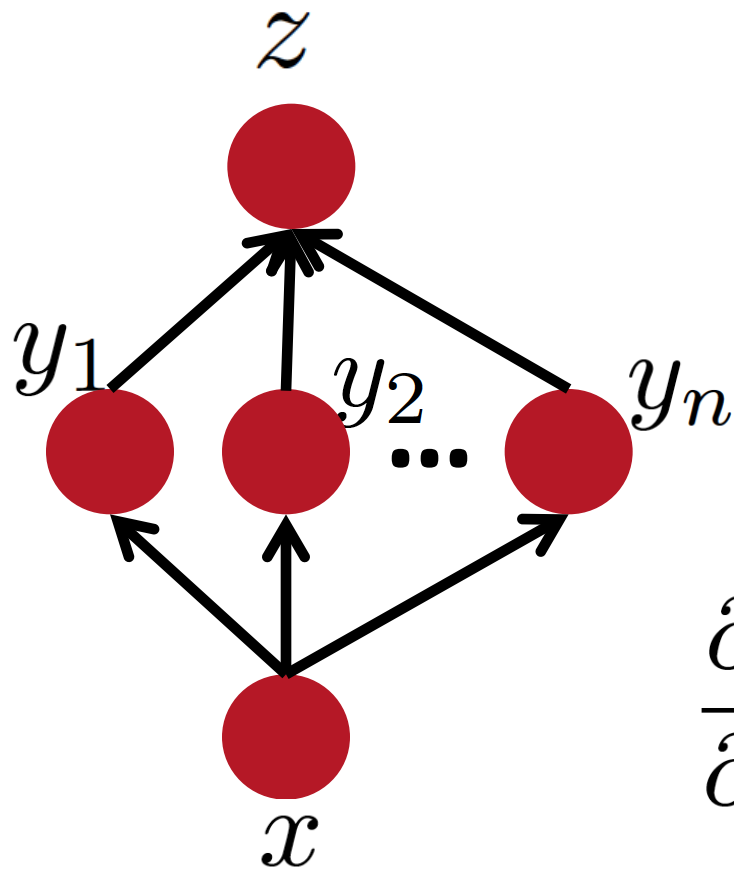
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

Multiple Paths Chain Rule



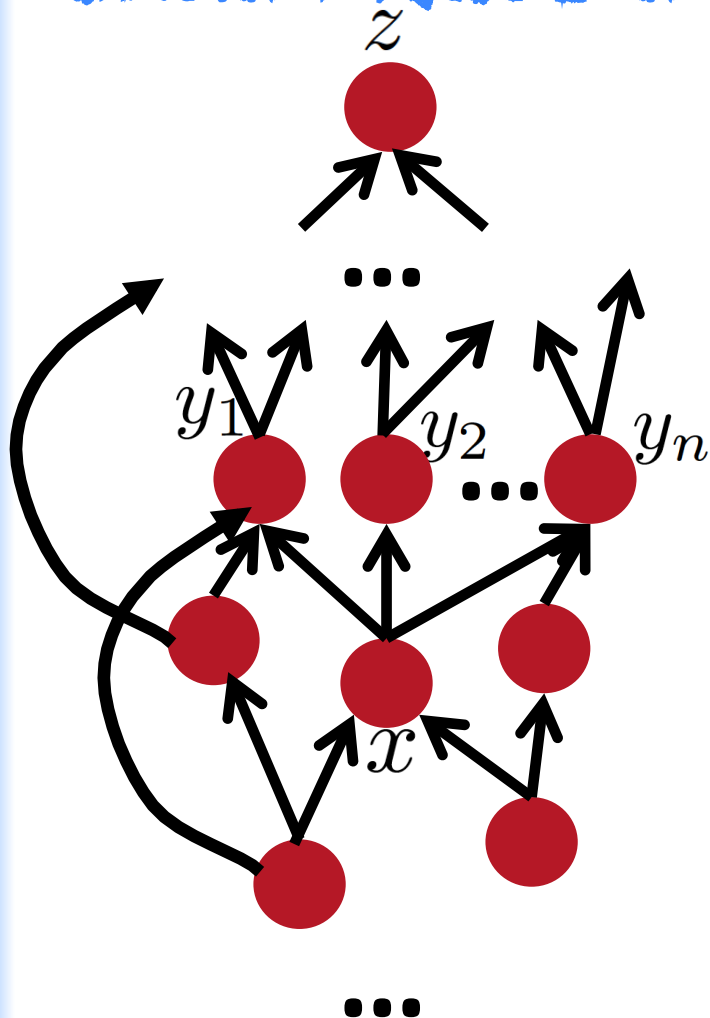
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x}$$

Chain Rule in Flow Graph



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

Chain Rule in Flow Graph

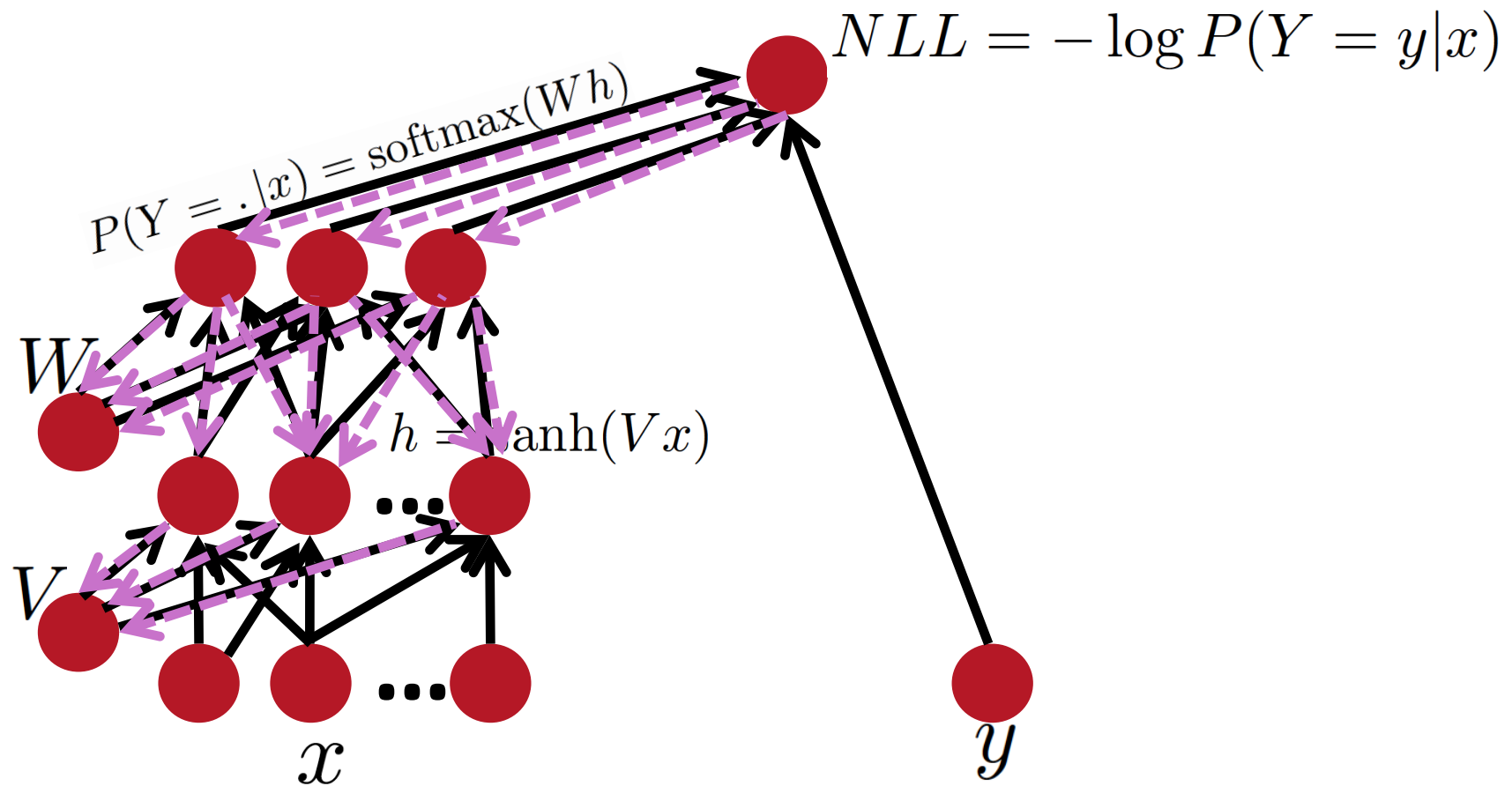


Flow graph: any directed acyclic graph
 node = computation result
 arc = computation dependency

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

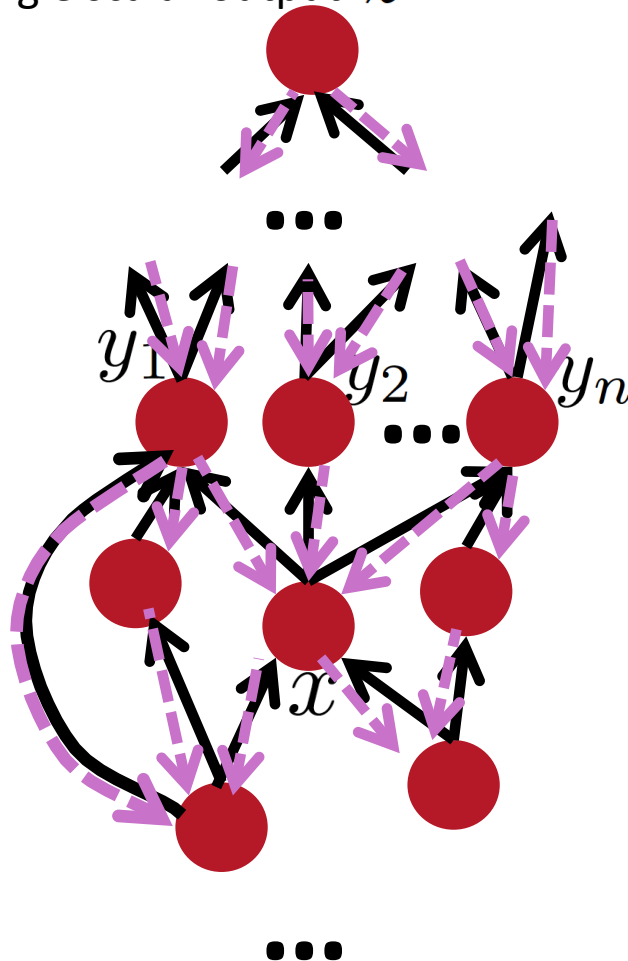
$$\frac{\partial z}{\partial x} = \sum_{i=1}^n \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Back-Prop in Multi-Layer Net



Back-Prop in General Flow Graph

Single scalar output z



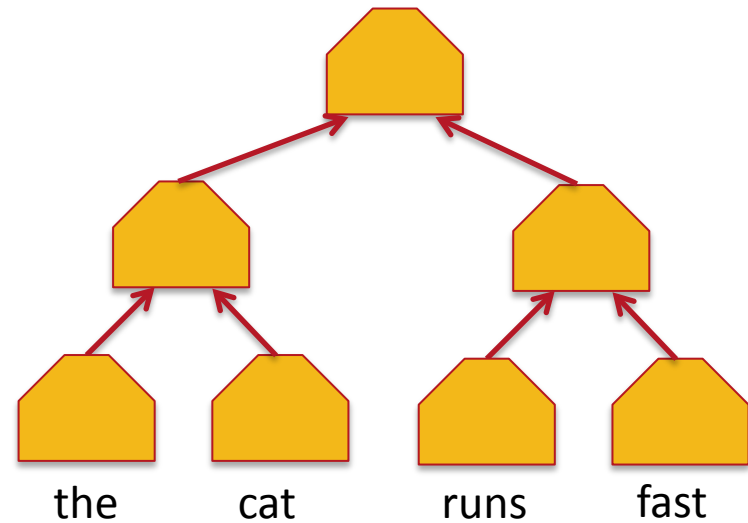
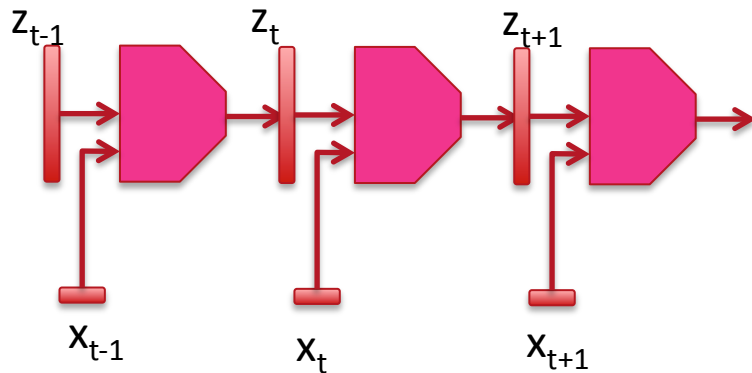
1. Fprop: visit nodes in topo-sort order
 - Compute value of node given predecessors
2. Bprop:
 - initialize output gradient = 1
 - visit nodes in reverse order:
 - Compute gradient wrt each node using gradient wrt successors

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

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

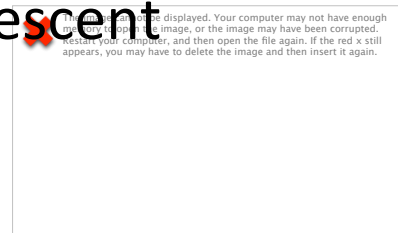
Back-Prop in Recur{sive} Net

- Replicate a parametrized function over different time steps or nodes of a DAG
- Output state at one time-step / node is used as input for another time-step / node
- Very deep once unfolded!

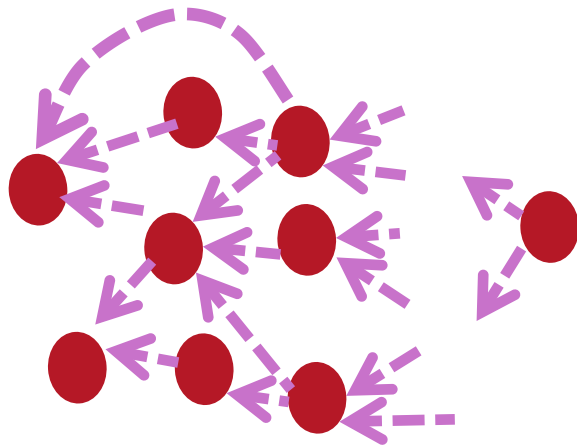
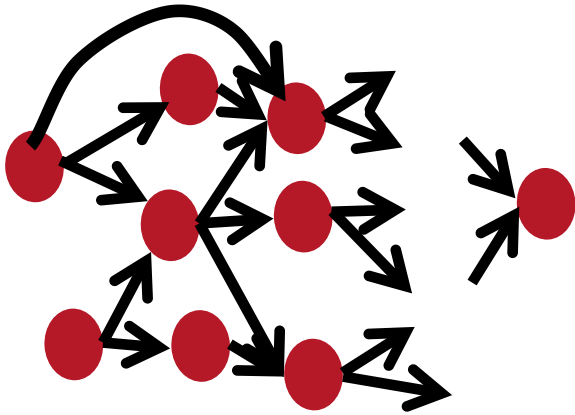


BP Through Structured Inference

- Inference \rightarrow discrete choices
 - (e.g., shortest path in HMM, best output configuration in CRF)
- E.g. Max over configurations or sum weighted by posterior
- The loss to be optimized depends on these choices
- The inference operations are flow graph nodes
- If continuous, can perform stochastic gradient descent
 - $\text{Max}(a,b)$ is continuous.
- Collobert & Weston 2008: max-pooling layer



Automatic Differentiation



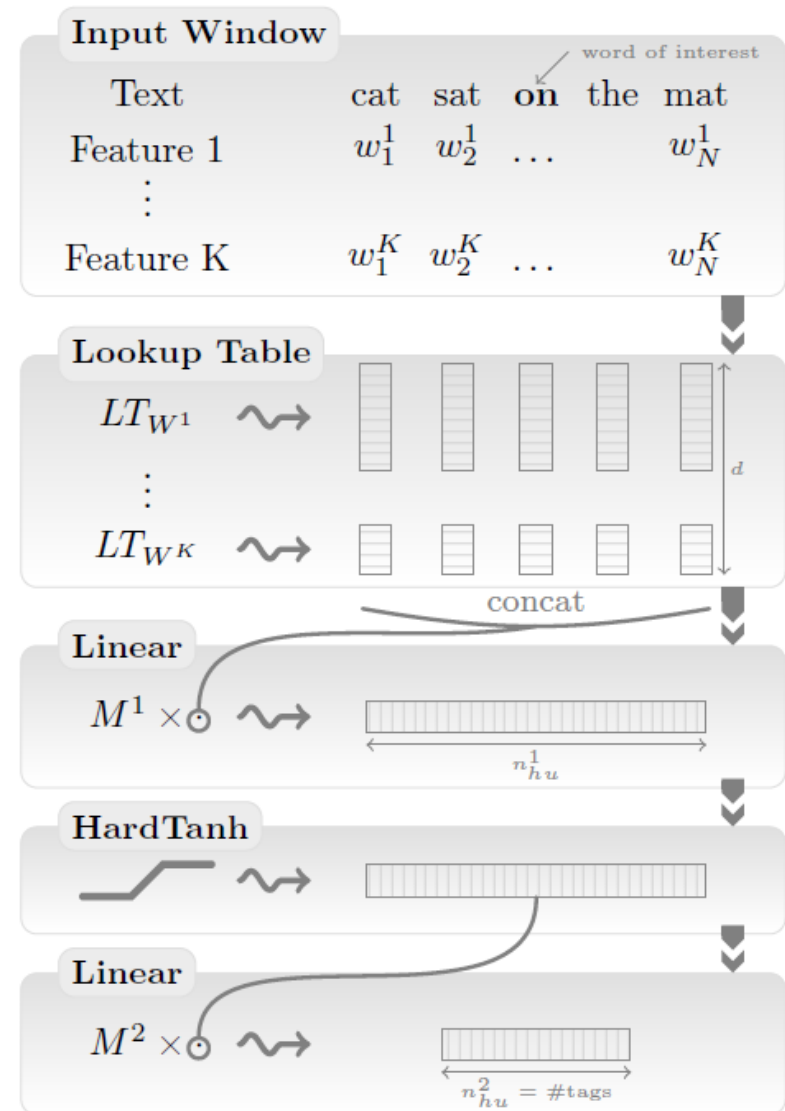
- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Makes it easier to quickly and safely try new models.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Theano Library (python) does it symbolically. Other neural network packages (Torch, Lush) do it numerically (at run-time).

Part 1.6: The Basics

Learning word-level classifiers: POS and NER

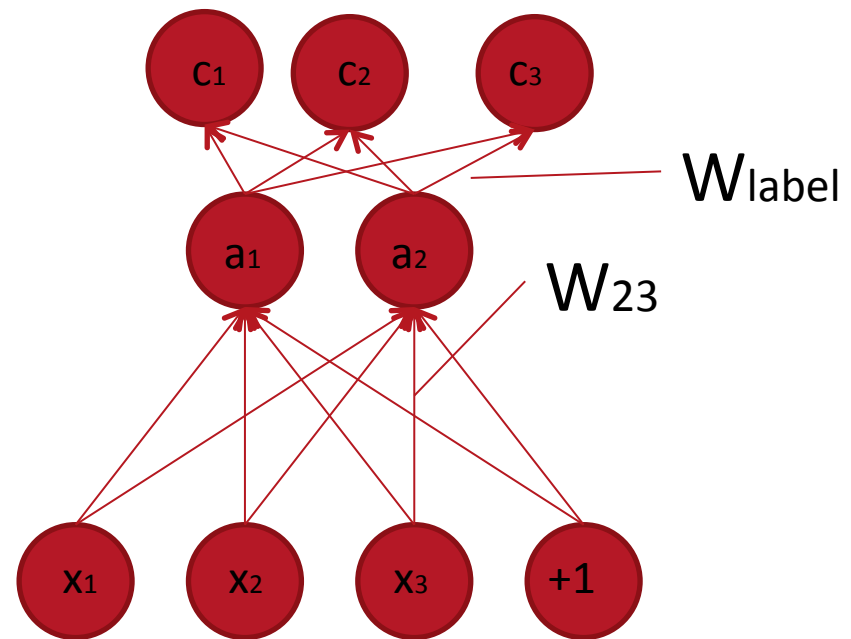
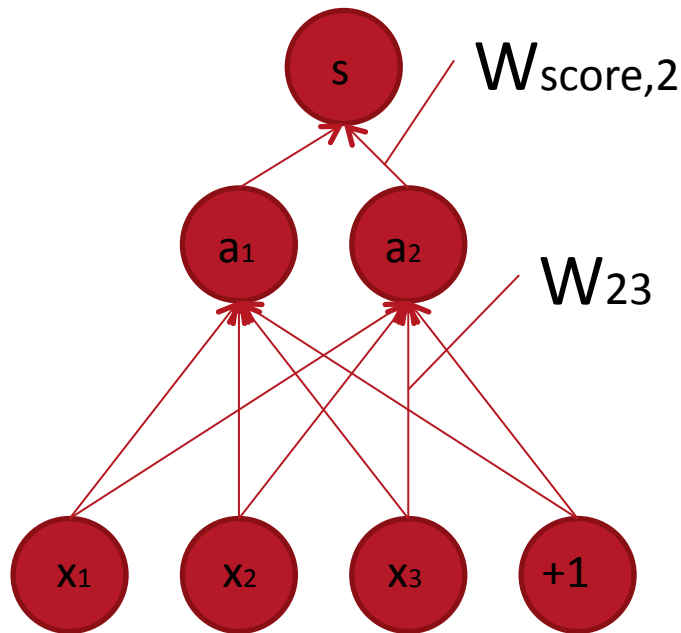
The Model

- Collobert & Weston (2008)
- Similar to the 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



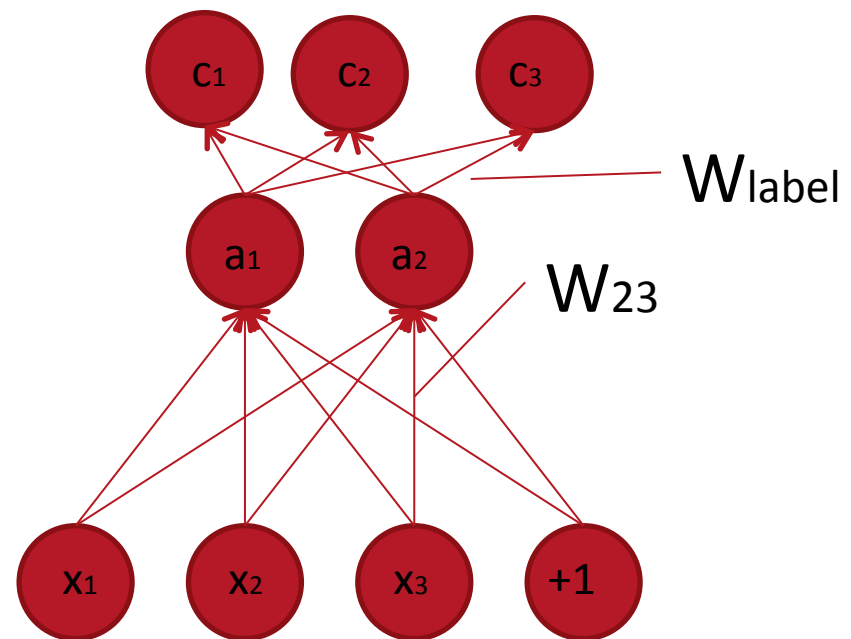
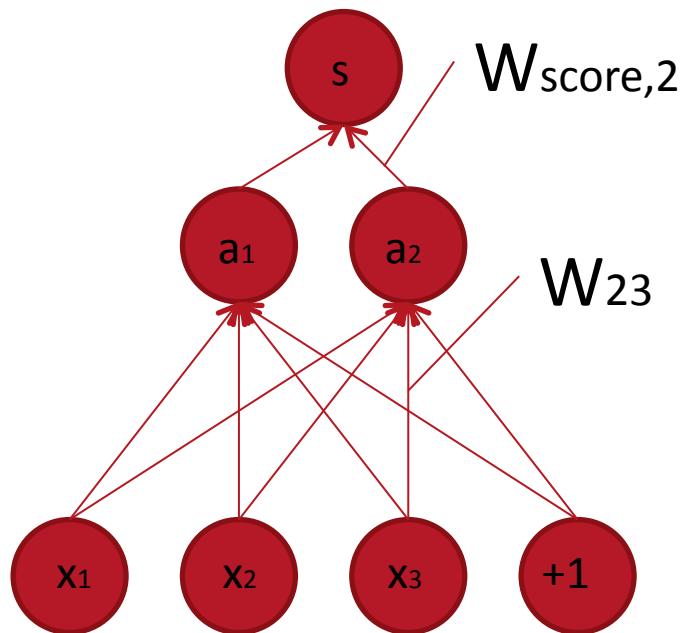
The Model - Training

- 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:



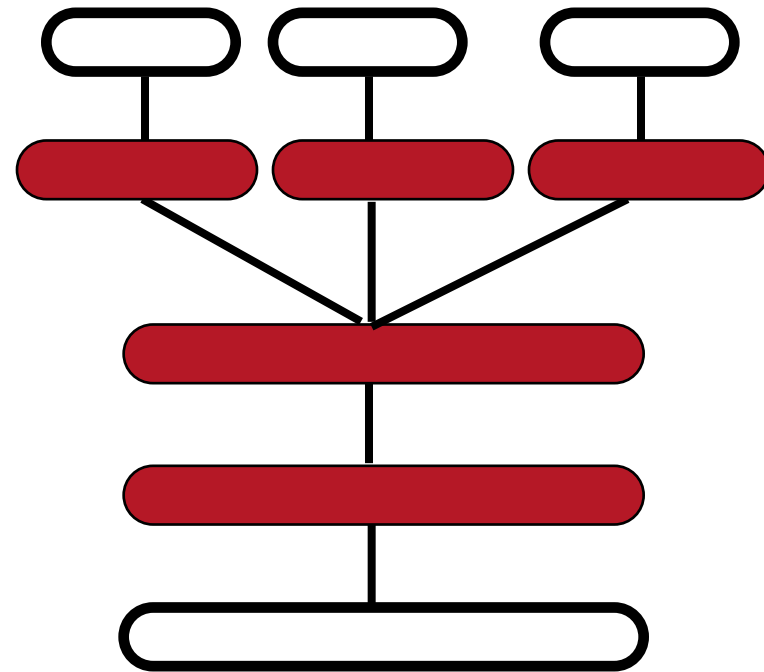
The Model - Training

- All derivatives of layers beneath the score were multiplied by $W_{\text{score},2}$, for the softmax, the error vector becomes the difference between predicted and gold standard distributions



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



Experiments

- Collobert et al (2011) share word embeddings across:
 - Language modeling (predict next word)
 - POS
 - Chunking
 - SRL
 - NER
 - Parsing

	POS (PWA)	Chunk (F1)	NER (F1)	SRL (F1)
SOTA	97.24	94.29	89.31	77.92
Supervised	96.37	90.33	81.47	70.99
Semi-supervised/ multi-task	97.20	93.63	88.87	74.15
+ hand-crafted features	97.37	94.35	89.70	76.06

Part 1.7

Sharing statistical strength

Sharing Statistical Strength

- Besides very fast predictors, the main advantage of deep learning is **statistical**: potential to learn from less examples because of sharing of statistical strength:
 - Unsupervised pre-training and semi-supervised training
 - Multi-task learning
 - Multi-data sharing, learning about symbolic objects and their relations

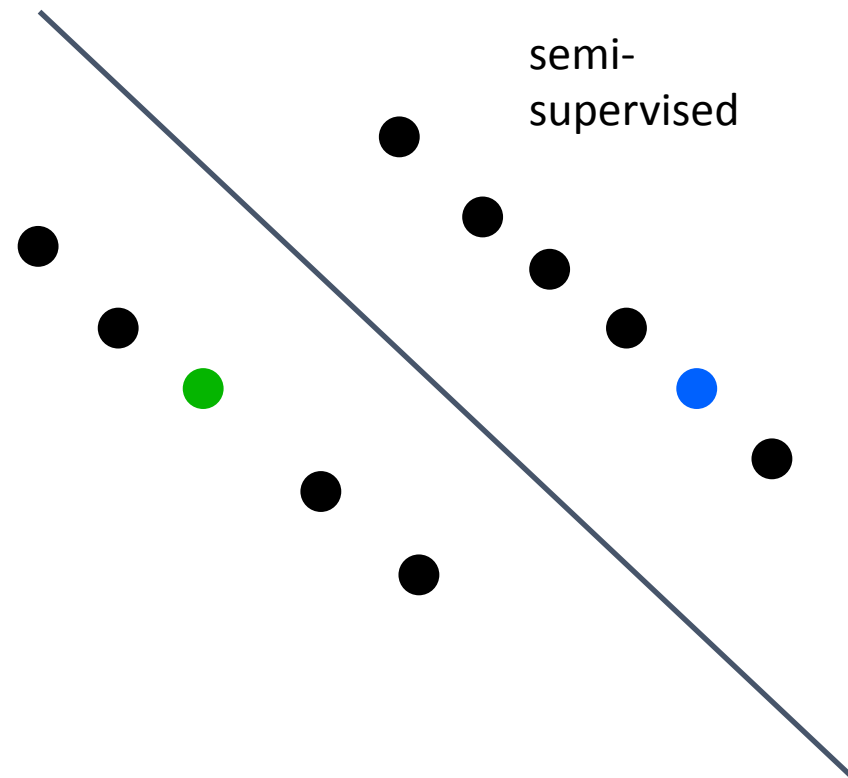
Unsupervised Learning

- Major breakthrough in 2006:
 - Ability to train deep architectures by using layer-wise unsupervised learning, whereas previous supervised attempts had failed
 - Unsupervised feature learners:
 - RBMs
 - Auto-encoder variants
 - Sparse coding variants



Sharing Statistical Strength by Semi-Supervised Learning

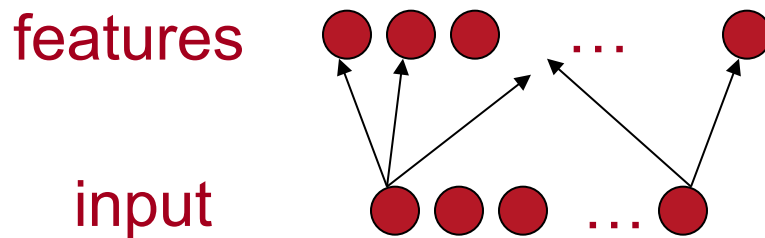
- Hypothesis: $P(x)$ shares structure with $P(y|x)$



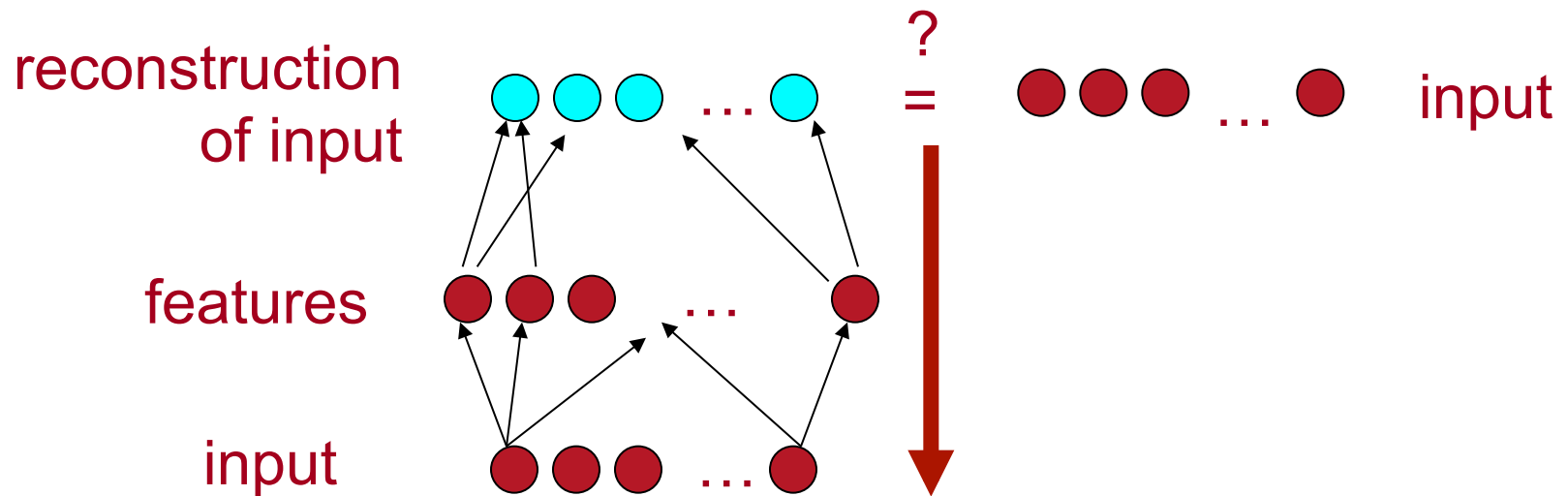
Layer-wise Unsupervised Learning

input ● ● ● ... ●

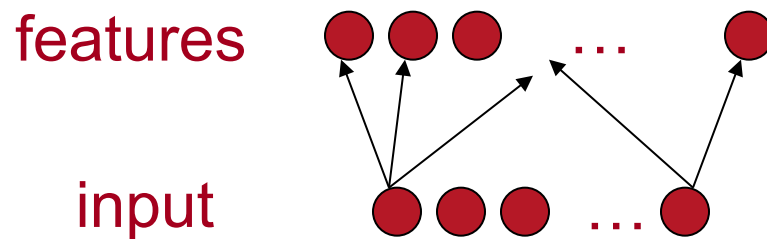
Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training

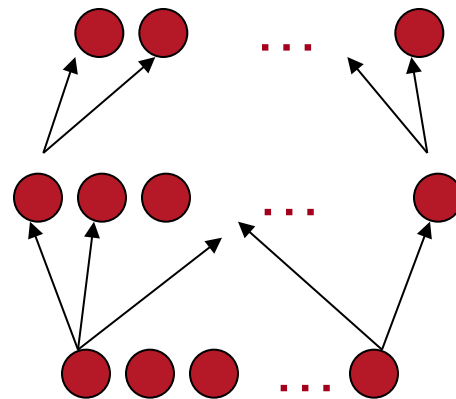


Layer-Wise Unsupervised Pre-training

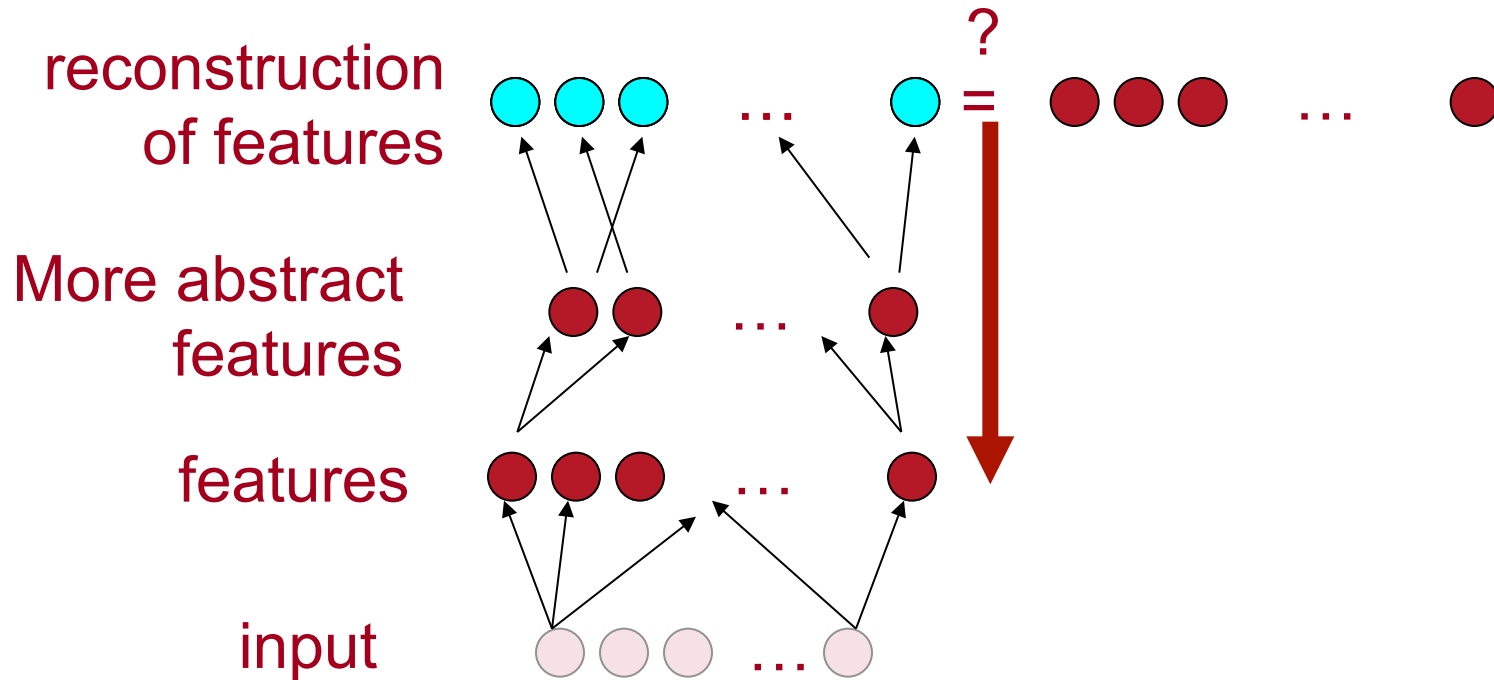
More abstract
features

features

input



Layer-wise Unsupervised Learning

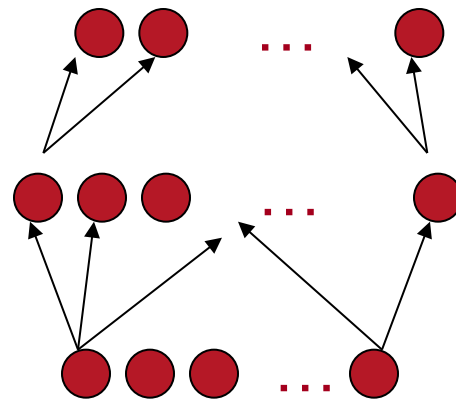


Layer-Wise Unsupervised Pre-training

More abstract
features

features

input



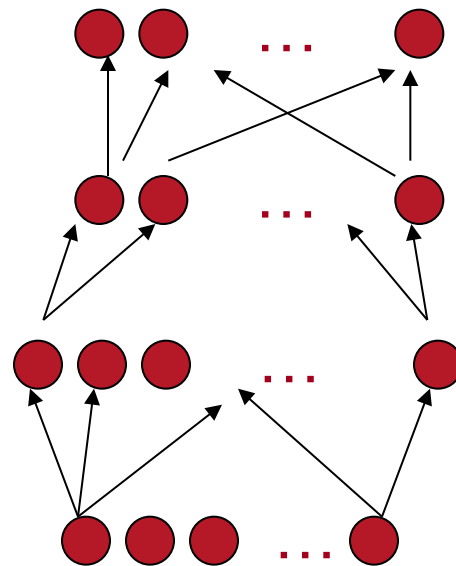
Layer-wise Unsupervised Learning

Even more abstract
features

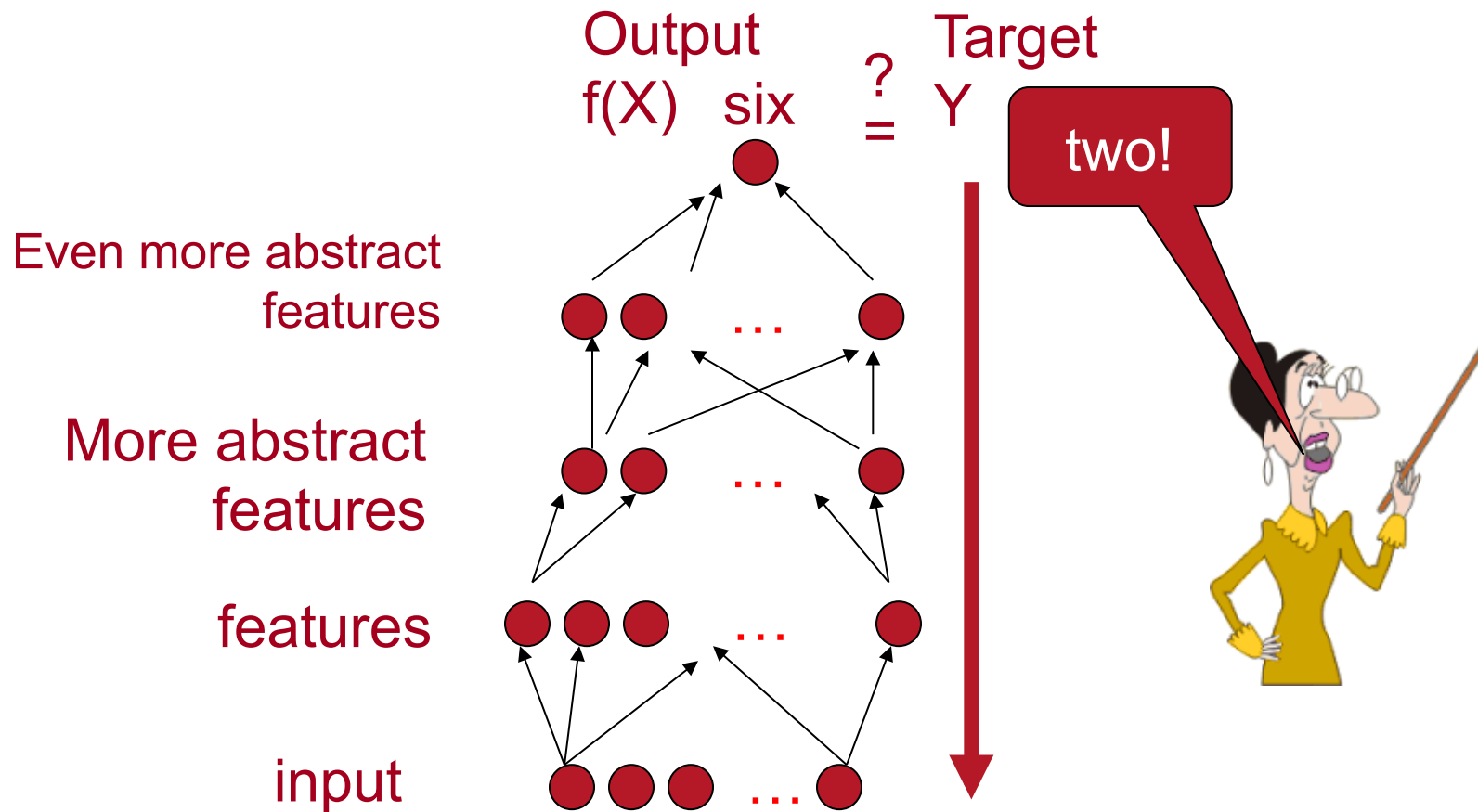
More abstract
features

features

input



Supervised Fine-Tuning



- Additional hypothesis: features good for $P(x)$ good for $P(y|x)$

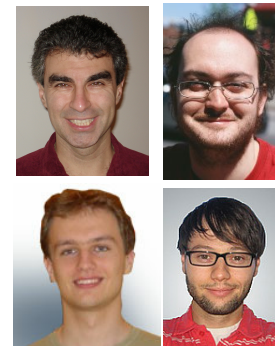
Auto-Encoders

- MLP whose target output = input
- Reconstruction=decoder(encoder(input)), e.g.

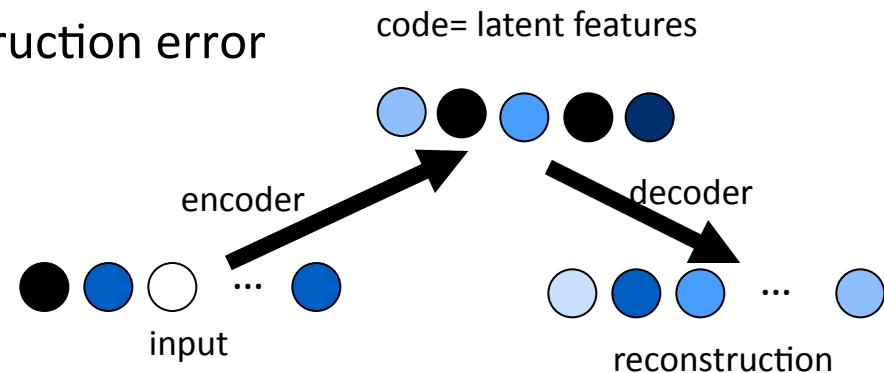
$$h = \tanh(b + Wx)$$

$$\text{reconstruction} = \tanh(c + W^T h)$$

$$\text{cost} = \| \text{reconstruction} - x \|^2$$



- Probable inputs have small reconstruction error



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

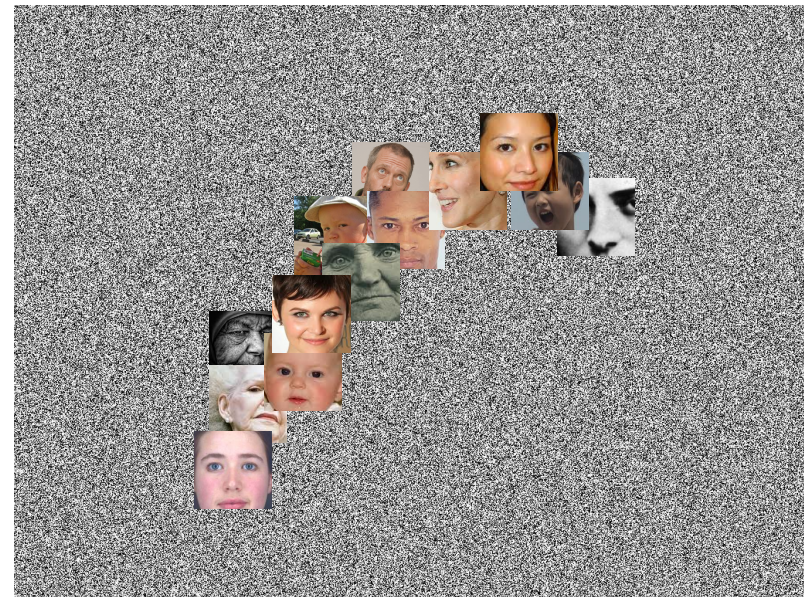
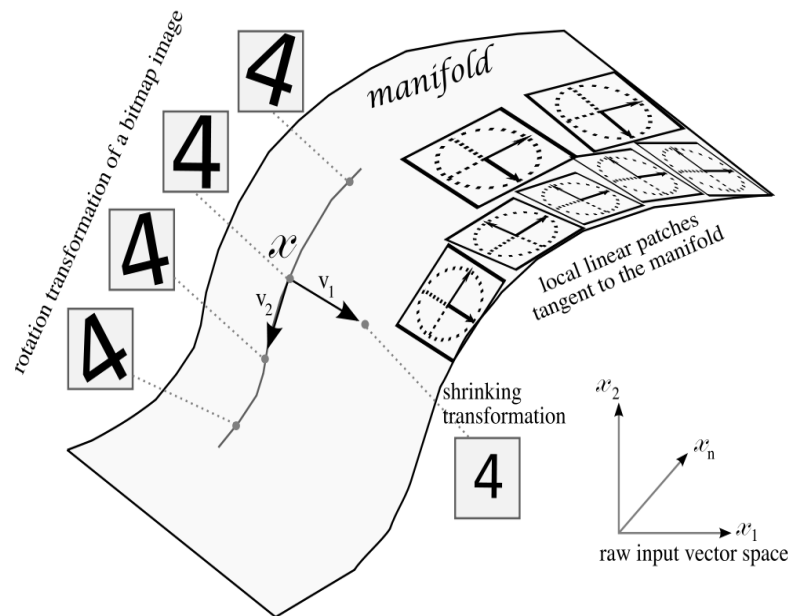
Auto-Encoder Variants

- 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]



Manifold Learning

- Additional hypothesis: examples concentrate near a lower dimensional “manifold” (region of high density with only few operations allowed which allow small changes while staying on the manifold)



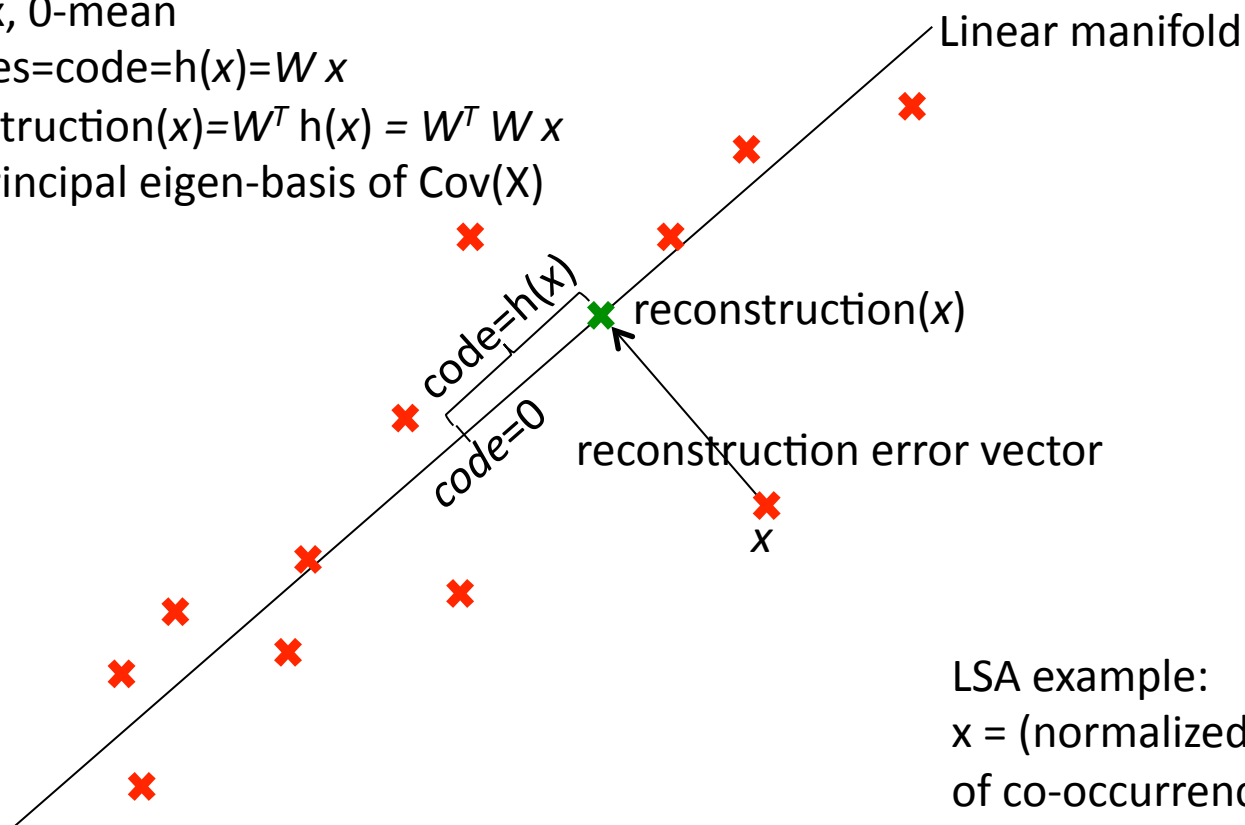
PCA = Linear Manifold = Linear Auto-Encoder

input x , 0-mean

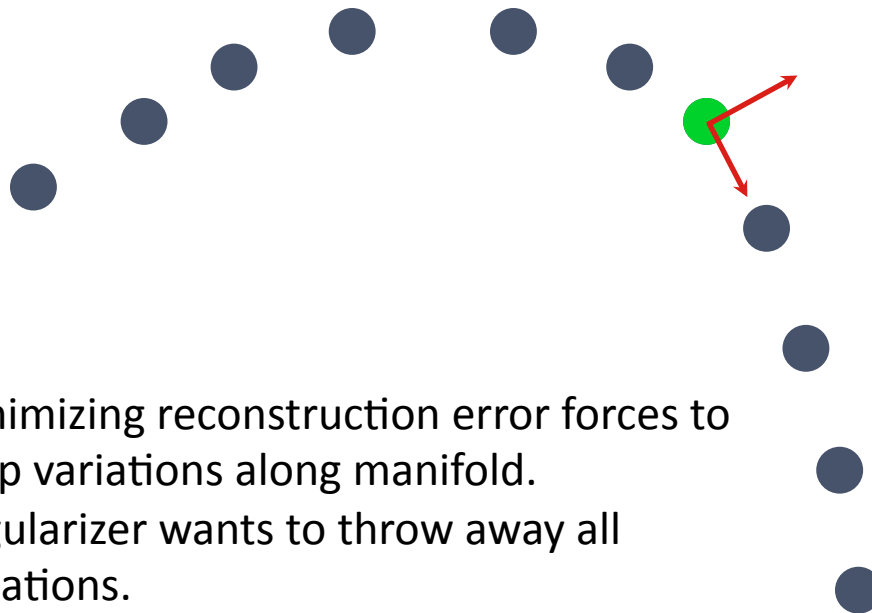
features=code= $h(x)=W x$

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

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

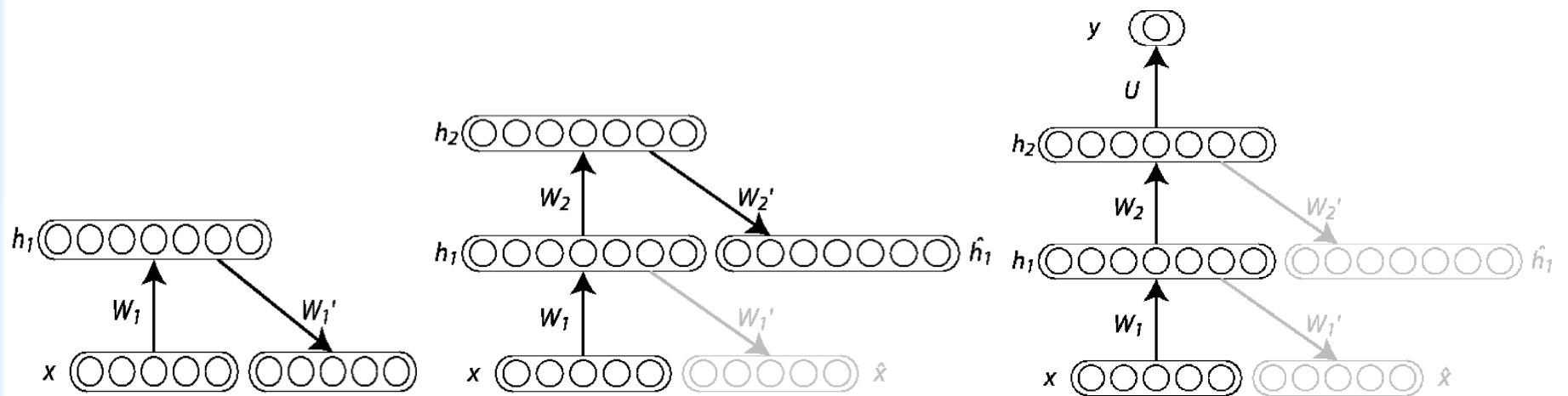


Auto-Encoders Learn Salient Variations, like a non-linear PCA



- Minimizing reconstruction error forces to keep variations along manifold.
- Regularizer wants to throw away all variations.
- Both: keep ONLY sensitivity to variations ON the manifold.

Stacking Auto-Encoders



Why is Unsupervised Pre-Training Working So Well?

- Regularization hypothesis:
 - Unsupervised component forces model close to $P(x)$
 - Representations good for $P(x)$ are good for $P(y|x)$
- Optimization hypothesis:
 - Unsupervised initialization near better local minimum of supervised training error
 - Can reach lower local minimum otherwise not achievable by random initialization

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



Unsupervised Learning: Disentangling Factors of Variation

- *[Goodfellow et al NIPS'2009]*: some hidden units more invariant (with more depth) to input geometry variations
- *[Glorot et al ICML'2011]*: some hidden units specialize on one aspect (domain) while others on another (sentiment)
- We don't want invariant representations because it is not clear to what aspects, but disentangling factors would help a lot
- Sparse/saturated units seem to help
- Why?
- **How to train more towards that objective?**



Invariance and Disentangling

- Invariant features
- Which invariances?
- Alternative: learning to disentangle factors
- Good disentangling →
avoid the curse of dimensionality



Advantages of Sparse Representations

- Just add a penalty on learned representation
- Information disentangling (compare to dense compression)
- More likely to be linearly separable (high-dimensional space)
- Locally low-dimensional representation = local chart
- Hi-dim. sparse = efficient **variable size** representation
= **data structure**

Few bits of information

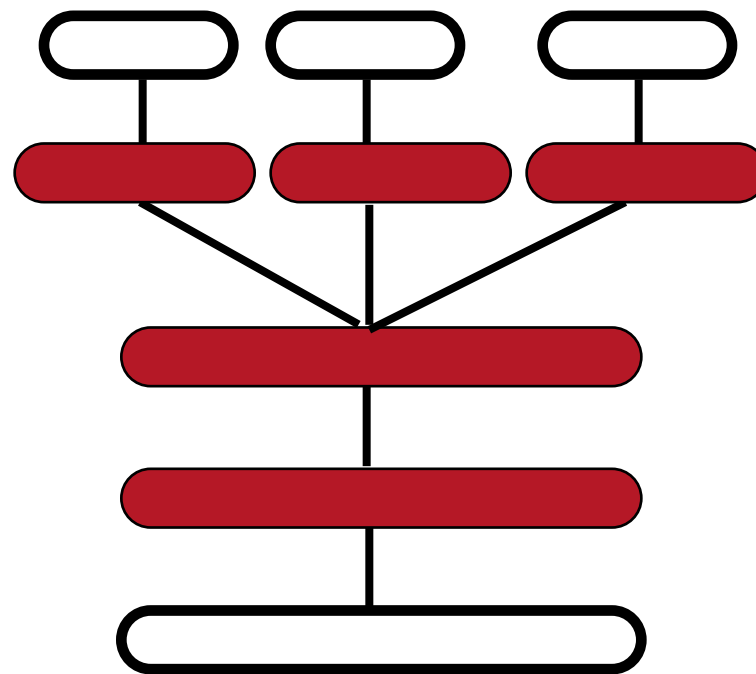


Many bits of information



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



Multi-Task Learning

- Collobert et al (2011) share word embeddings across:
 - Language modeling (predict next word)
 - POS
 - Chunking
 - SRL
 - NER
 - Parsing

	POS (PWA)	Chunk (F1)	NER (F1)	SRL (F1)
SOTA	97.24	94.29	89.31	77.92
Supervised	96.37	90.33	81.47	70.99
Semi-supervised/ multi-task	97.20	93.63	88.87	74.15
+ hand-crafted features	97.37	94.35	89.70	76.06

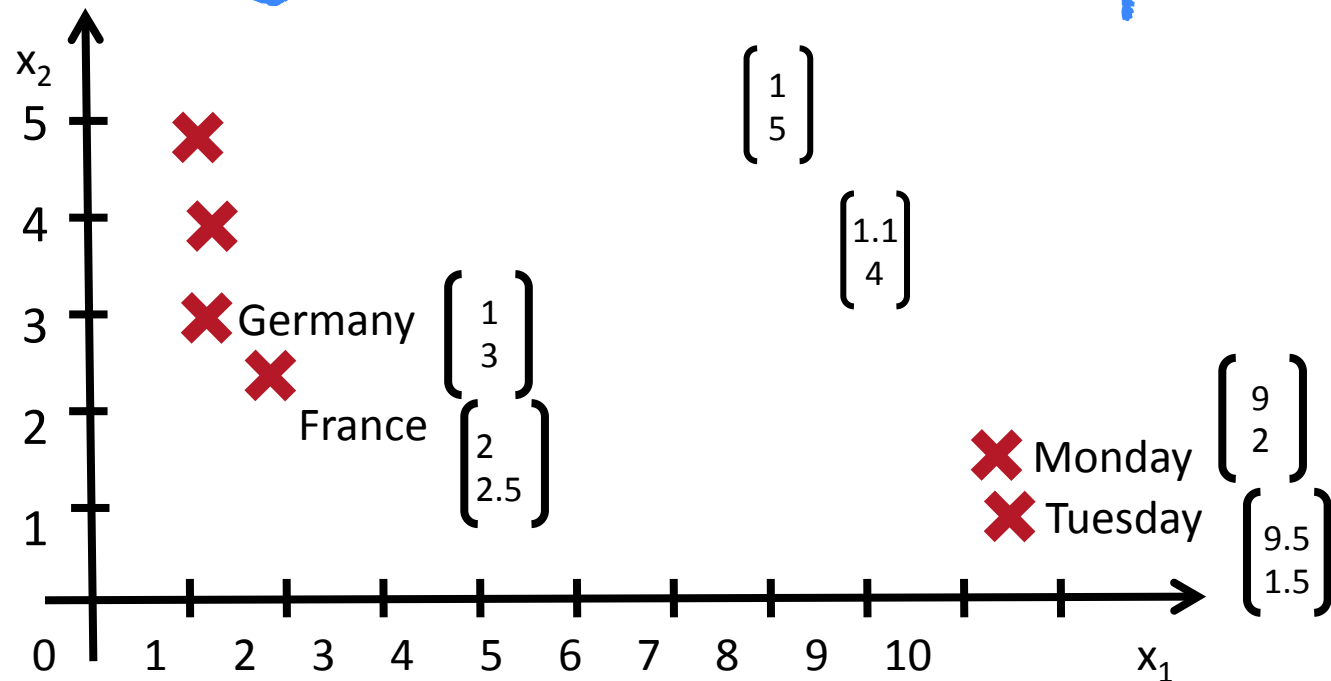
Combining Multiple Sources of Evidence with Shared Embeddings

- 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,...

Part 2

Recursive Neural Networks

Building on Word Vector Space Models



the country of my birth
the place where I was born

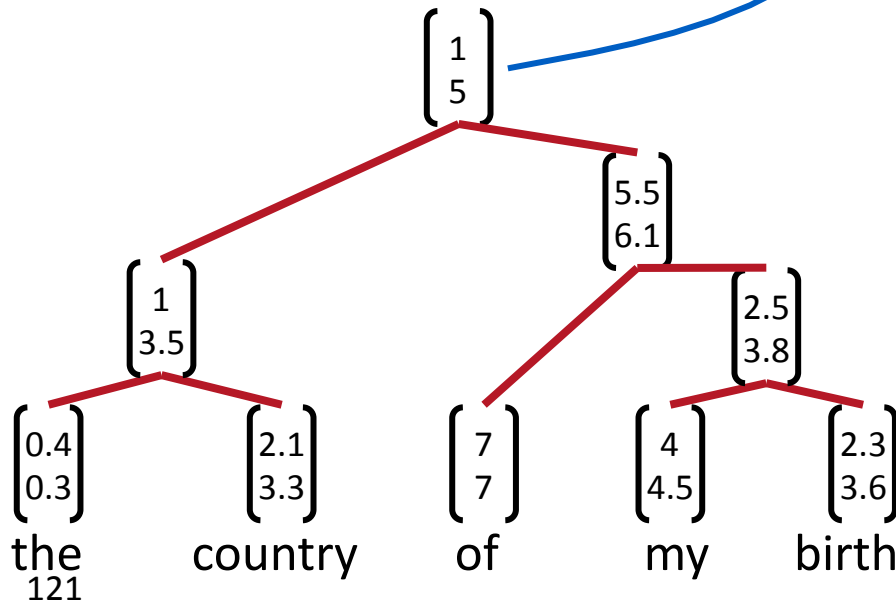
But how can we represent the meaning of longer phrases?

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.

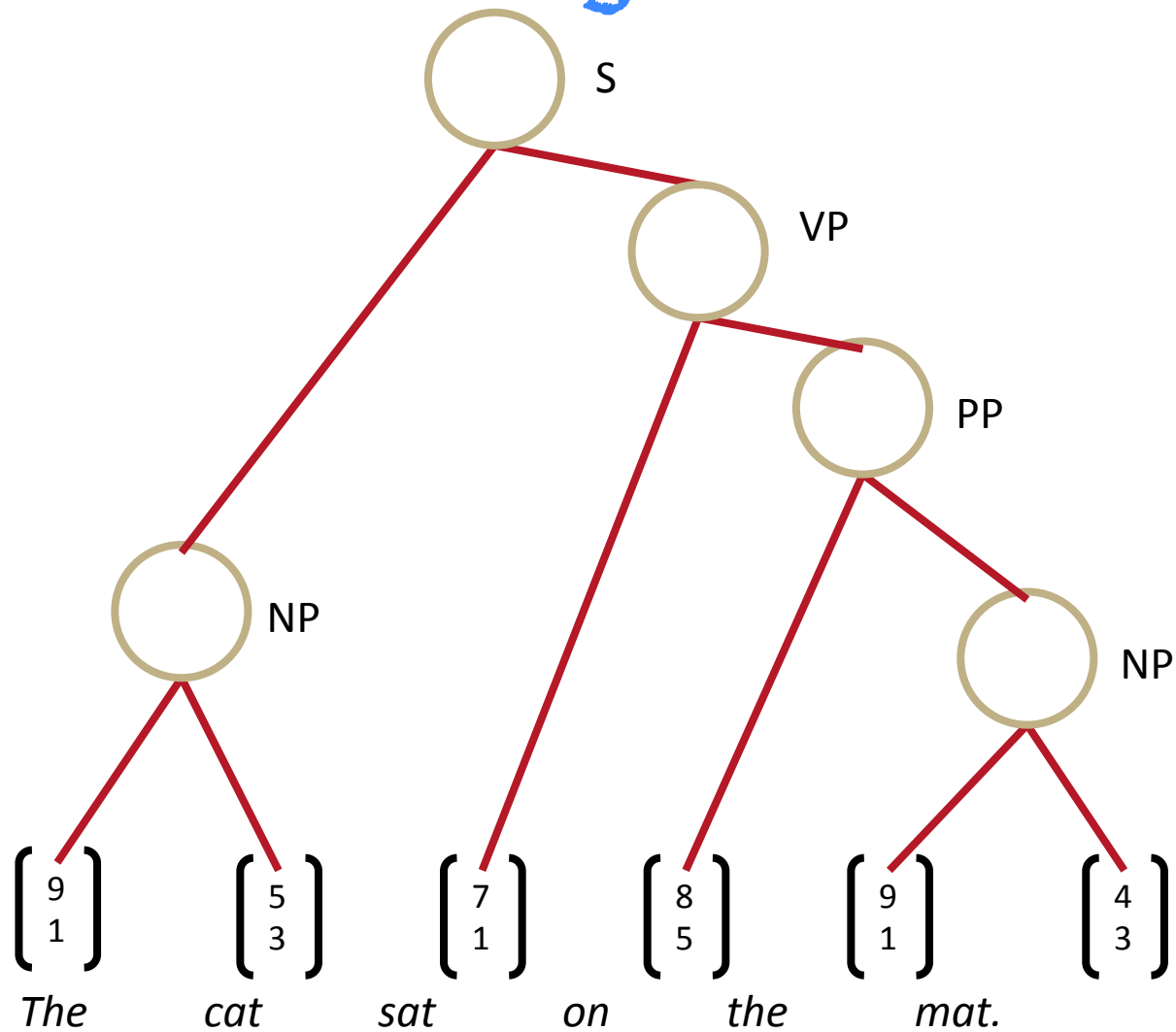


Recursive Neural Nets
can jointly learn
compositional vector
representations and
parse trees

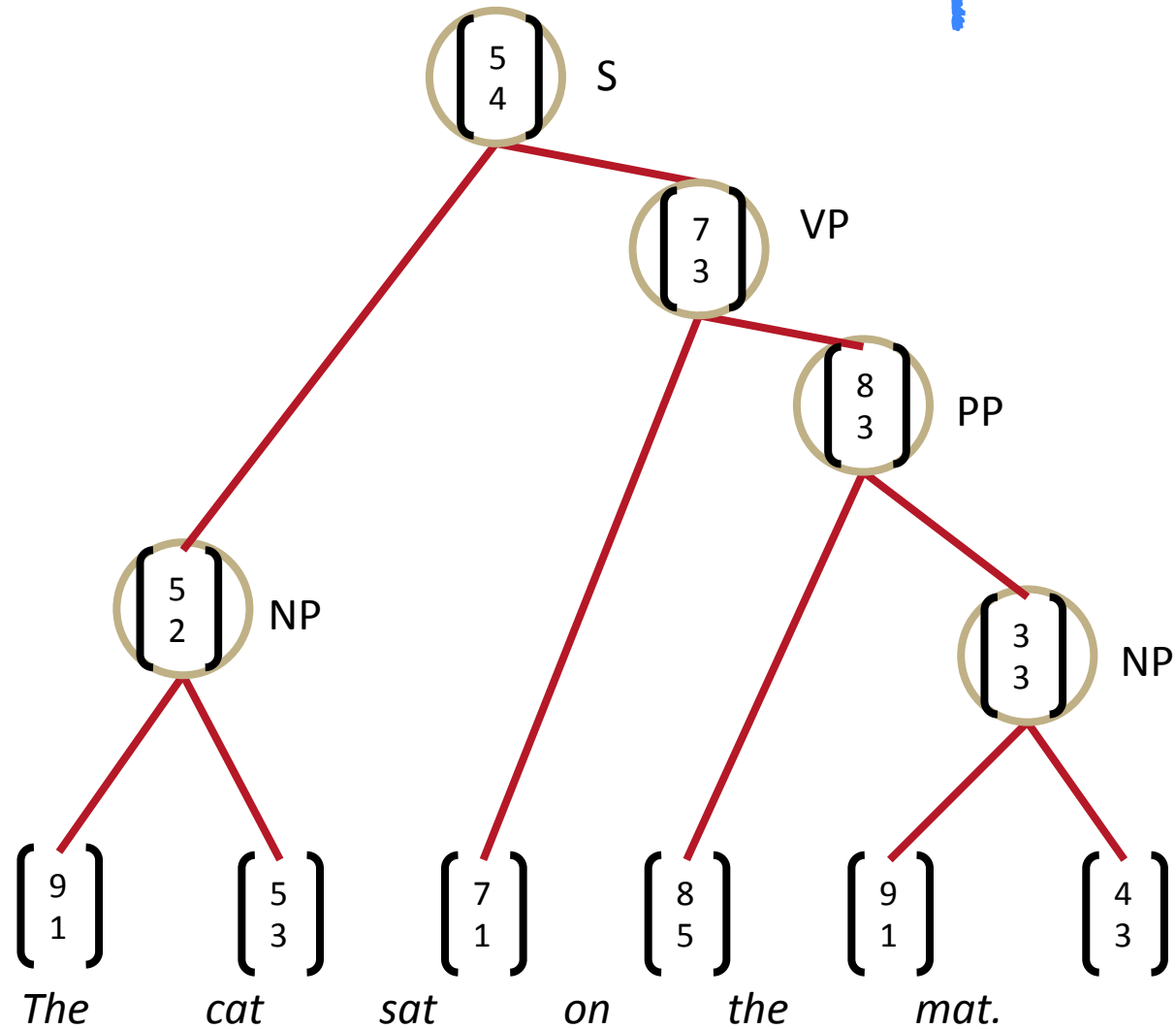
Recursive Neural Networks

1. Motivation
2. Recursive Neural Networks for Parsing
3. Theory: Backpropagation Through Structure
4. Recursive Autoencoders
5. Application to Sentiment Analysis and Paraphrase Detection
6. Compositionality Through Recursive Matrix-Vector Spaces
7. Relation classification

Sentence Parsing: What we want



Learn Structure and Representation

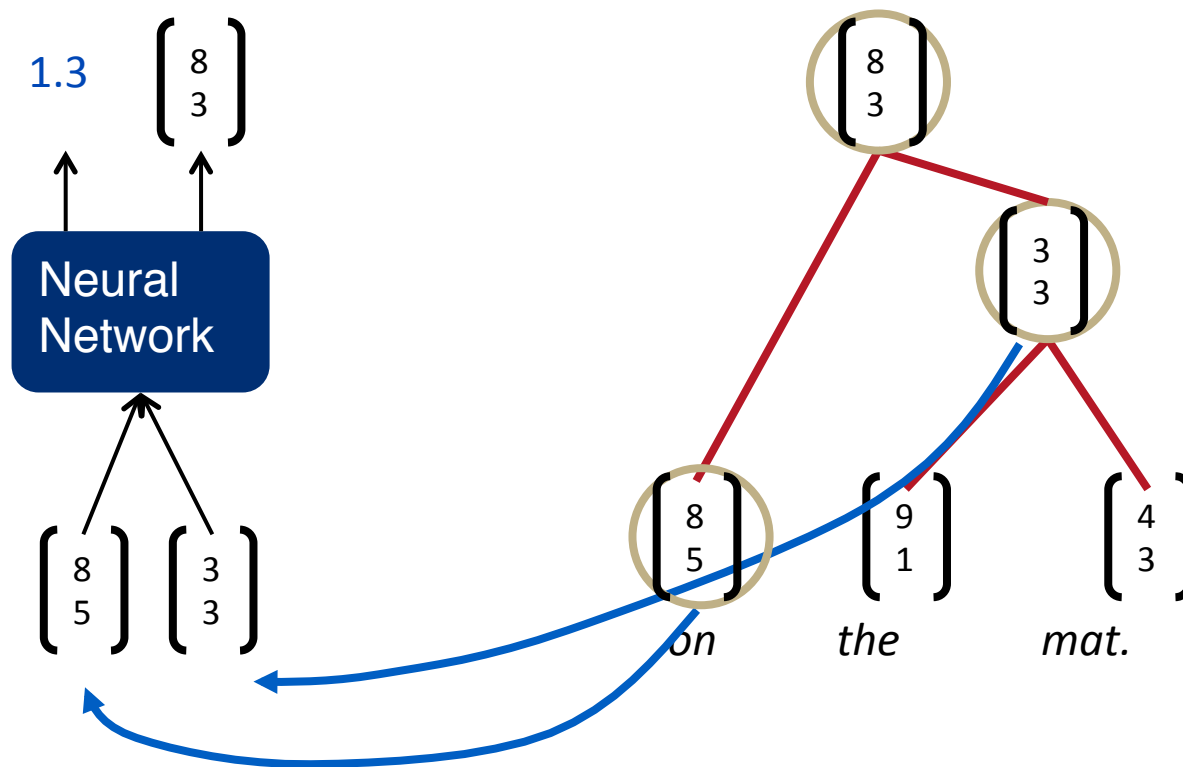


Recursive Neural Networks for Structure Prediction

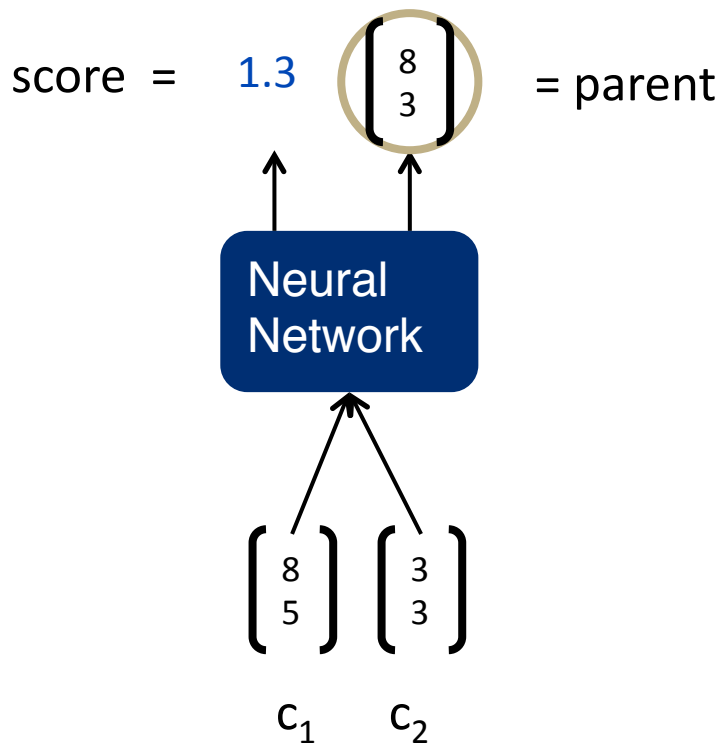
Inputs: two candidate children's representations

Outputs:

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



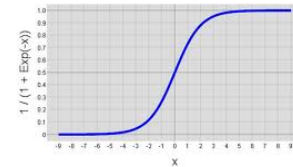
Recursive Neural Network Definition



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

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

where sigmoid:



Related Work to Socher et al, 2011a

- Pollack (1990): Recursive auto-associative memories



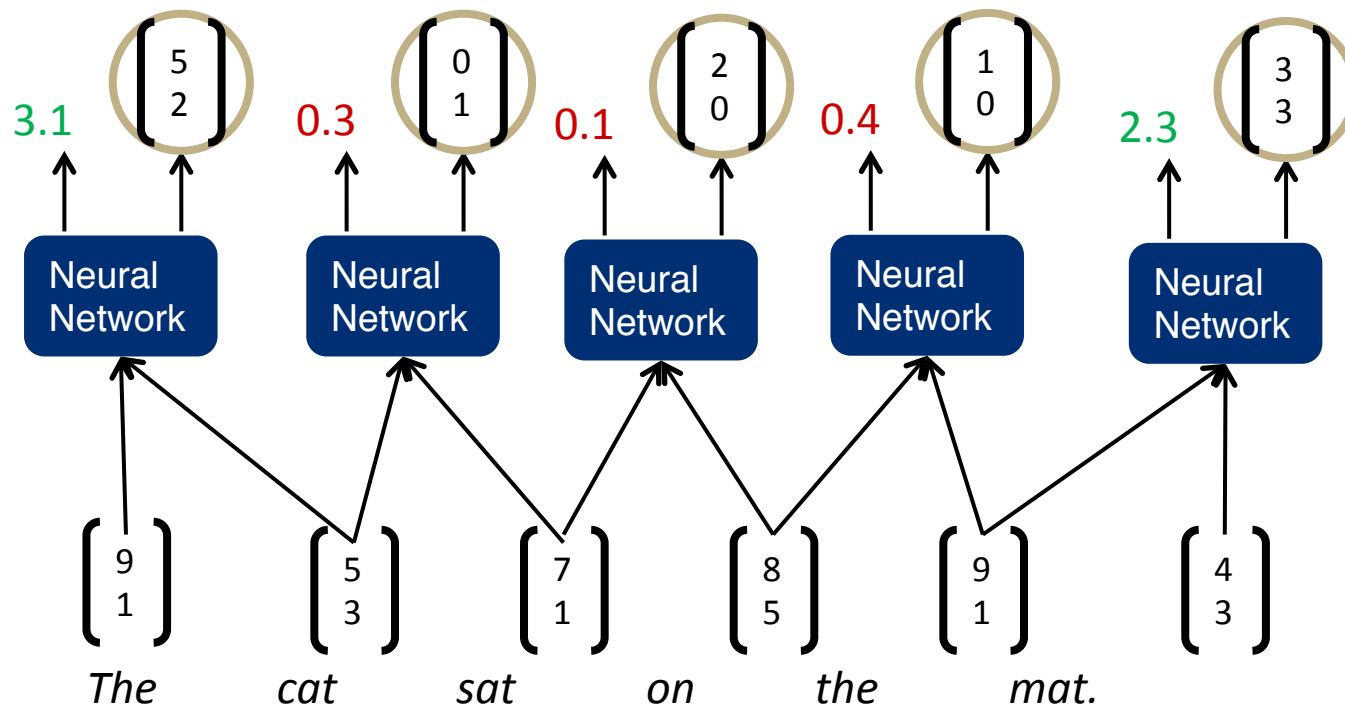
- Previous Recursive Neural Networks work by [Goller & Küchler 1996, Costa et al. 2003] assumed fixed tree structure and used one hot vectors.



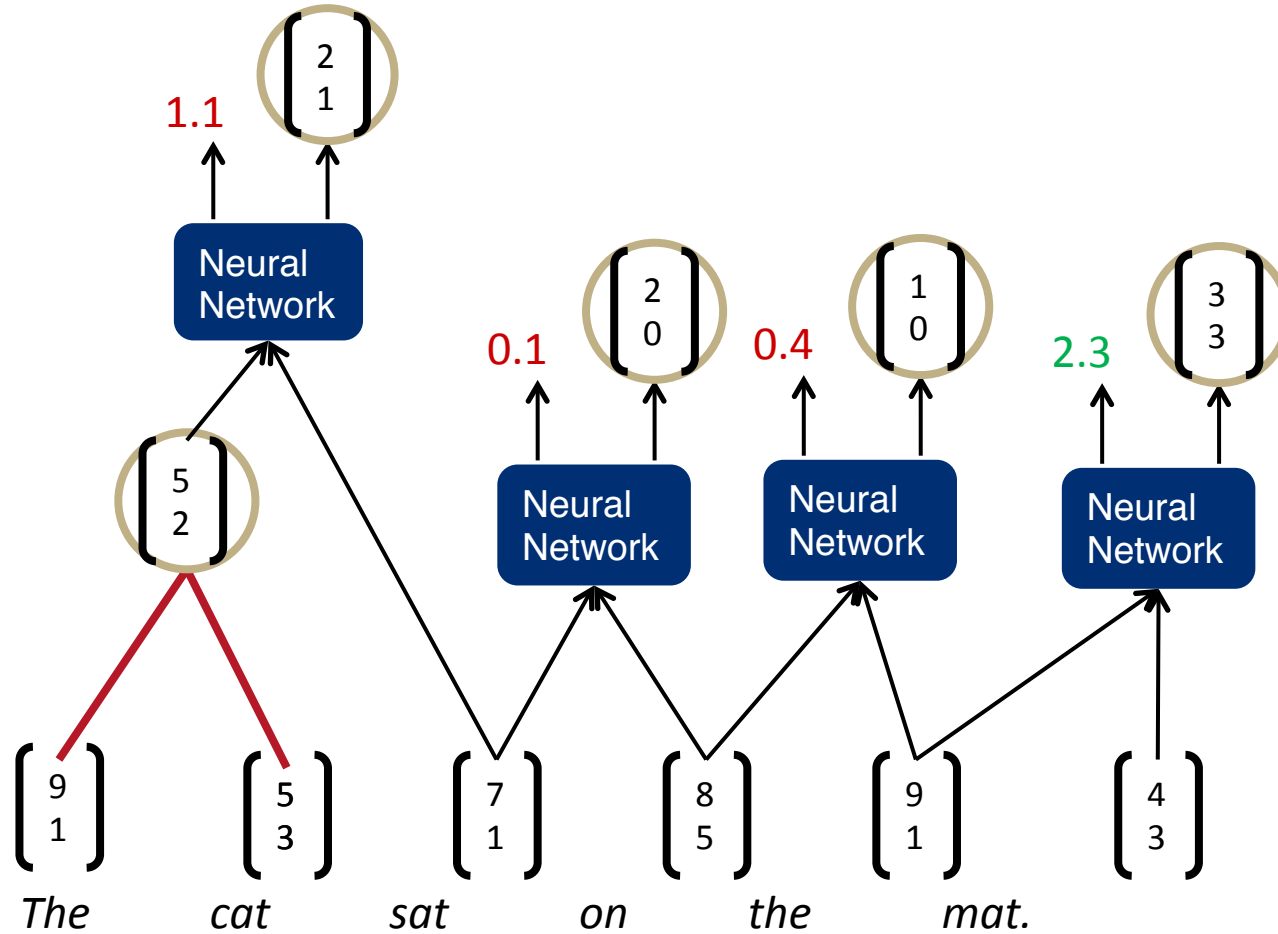
- Hinton (1990) and Bottou (2011): Related ideas about recursive models and recursive operators as smooth versions of logic operations



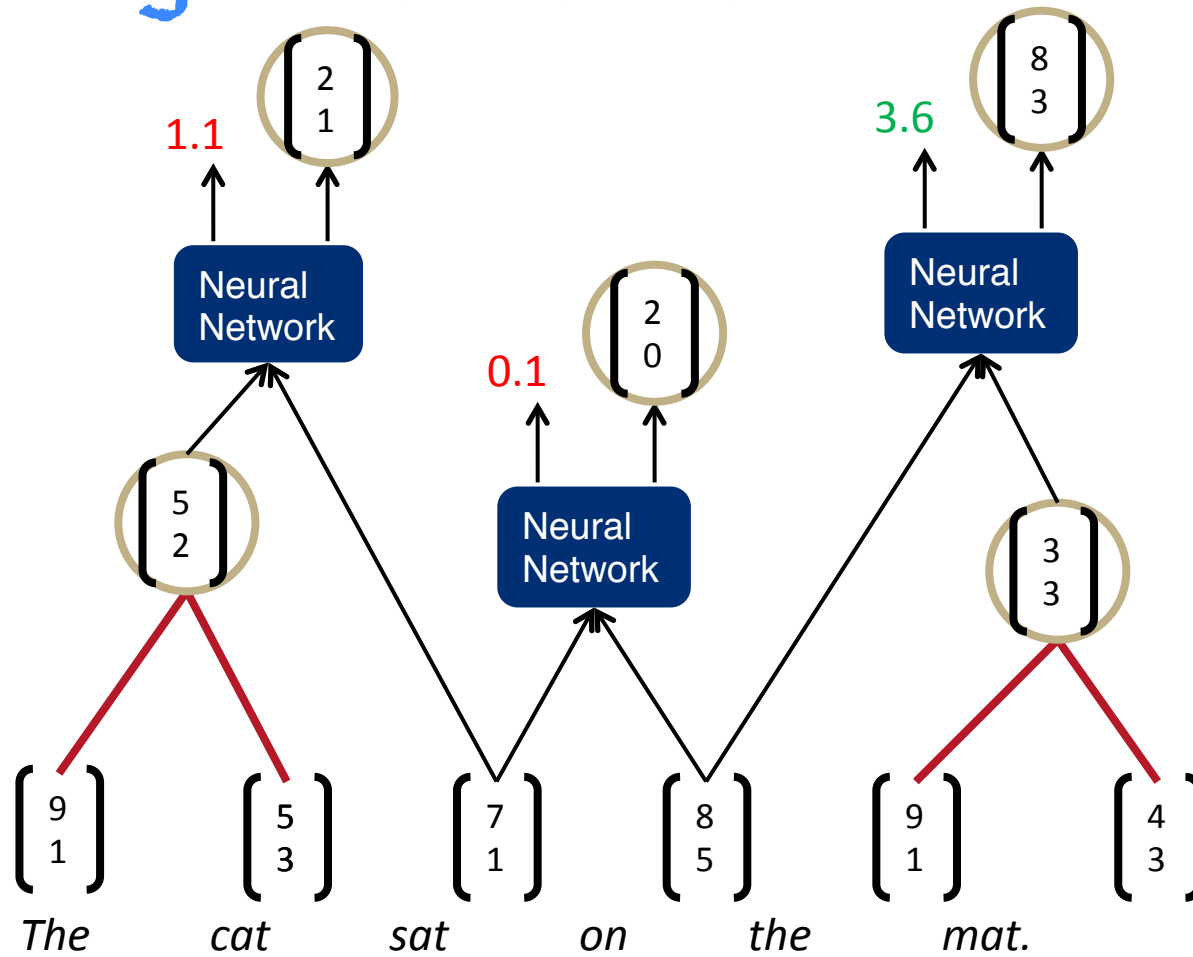
Parsing a sentence



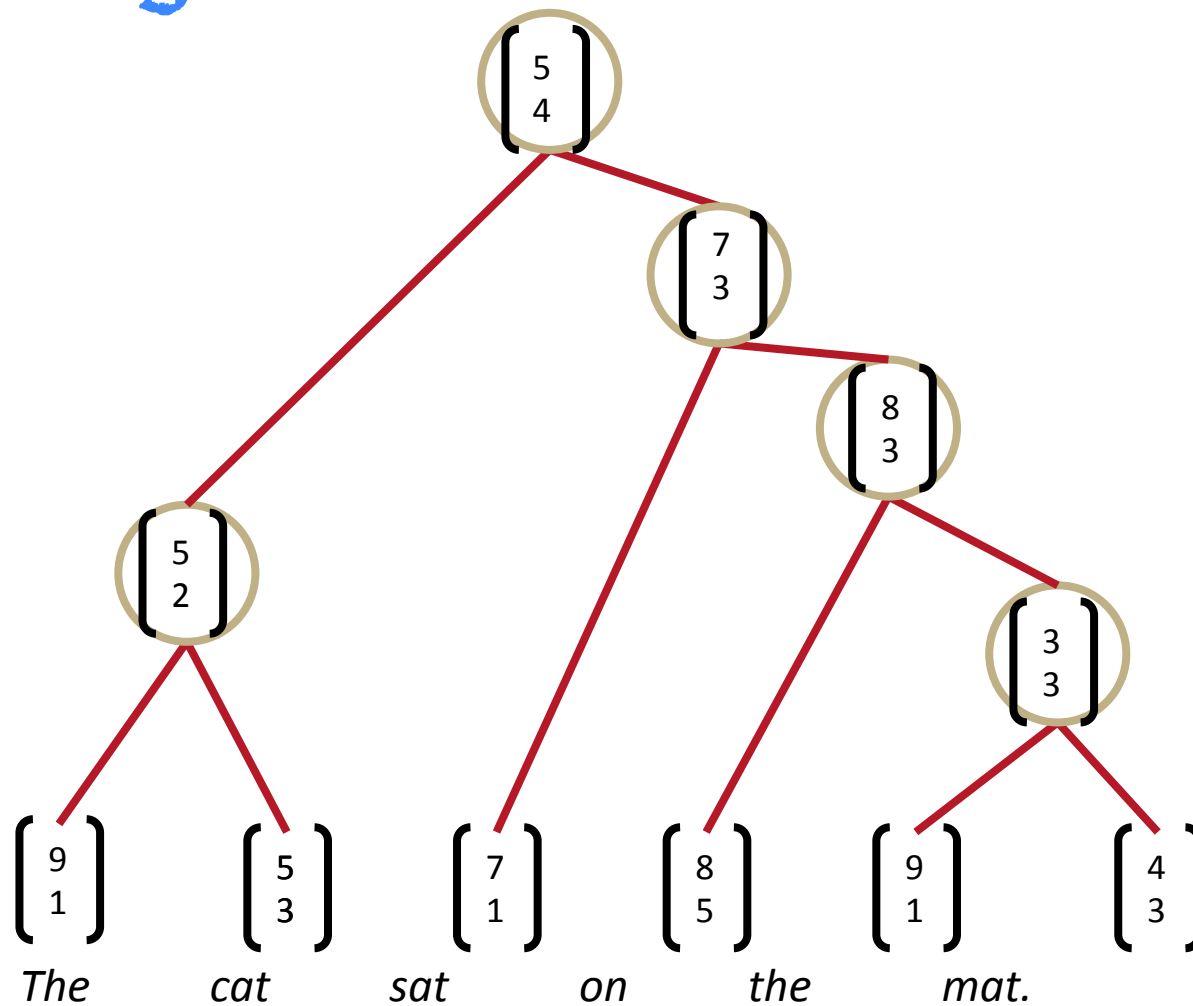
Parsing a sentence



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), we can formulate 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



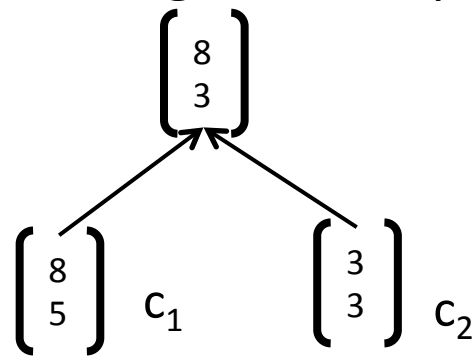
Backpropagation Through Structure (BTS)



- Introduced by [Goller et al. 1996]
- Principally the same as general backpropagation (efficient matrix derivative)
- Two differences resulting from the tree structure:
 - Split derivatives at each node
 - Sum derivatives of W from all nodes

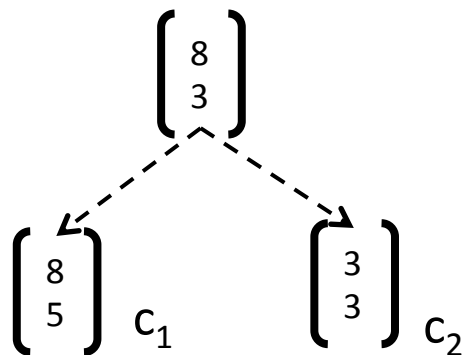
BTS: Split derivatives at each node

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



$$p = \text{sigmoid}(W \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + b)$$

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



$$\delta_{p \rightarrow c_1 c_2} = [\delta_{p \rightarrow c_1} \delta_{p \rightarrow c_2}]$$

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:

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

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

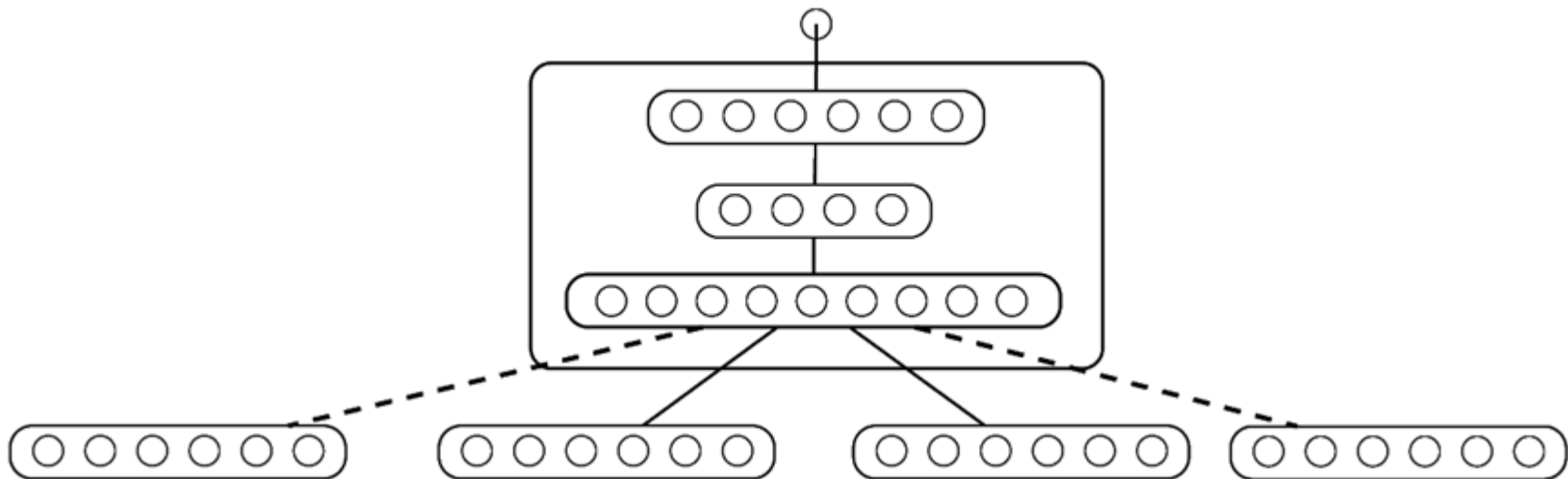
$$\begin{aligned} & \frac{\partial}{\partial W_2} f(W_2(f(W_1x))) + \frac{\partial}{\partial W_1} f(W_2(f(W_1x))) \\ = & f'(W_2(f(W_1x))) (f(W_1x)) + f'(W_2(f(W_1x))) (W_2 f'(W_1x)x) \\ = & f'(W_2(f(W_1x))) (f(W_1x) + W_2 f'(W_1x)x) \\ = & f'(W(f(Wx))) (f(Wx) + W f'(Wx)x) \end{aligned}$$

BTS: Optimization

- As before, we can plug the gradients into a standard off-the-shelf L-BFGS optimizer
- For non-continuous objective use subgradient *method* [Ratliff et al. 2007]

Details of Recursive Neural Networks

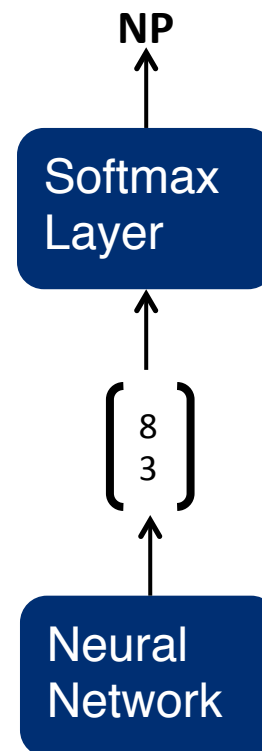
- Structure search was maximally **greedy**
 - Instead: Beam Search with Chart
- Include **context**: the word to the left and right



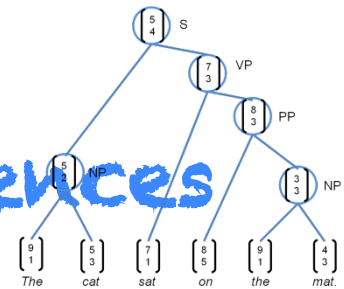
Labeling in Recursive Neural Networks

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

$$label_p = softmax(W^{label_p})$$



Experiments: Parsing Short Sentences



- Standard *WSJ* train/test
- Good results on short sentences
- More work is needed for longer sentences

Model	L15 Dev	L15 Test
Recursive Neural Network	92.1	90.3
Sigmoid NN (Titov & Henderson 2007)	89.5	89.3
Berkeley Parser (Petrov & Klein 2006)	92.1	91.6

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

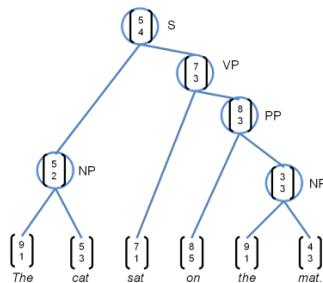
1. Harsco declined to say what country placed the order
2. Coastal wouldn't disclose the terms

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

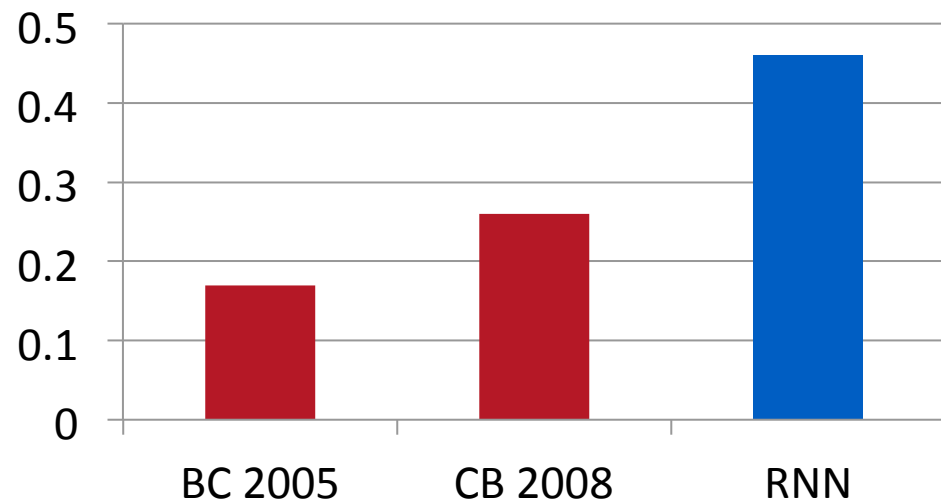
1. Sales rose more than 7% to \$94.9 m. from \$88.3 m.
2. Sales surged 40% to UNK b. yen from UNK b.

Paraphrase detection task

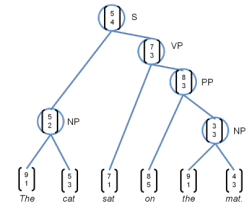
- Goal is to say which of candidate phrases are a good paraphrase of a given phrase
 - Motivated by Machine Translation
 - Initial algorithms: **Bannard & Callison-Burch 2005** (BC 2005), **Callison-Burch 2008** (CB 2008) exploit bilingual sentence-aligned corpora and hand-built linguistic constraints
 - We simply re-use our system learned on parsing the WSJ



F1 of Paraphrase Detection



Paraphrase detection task



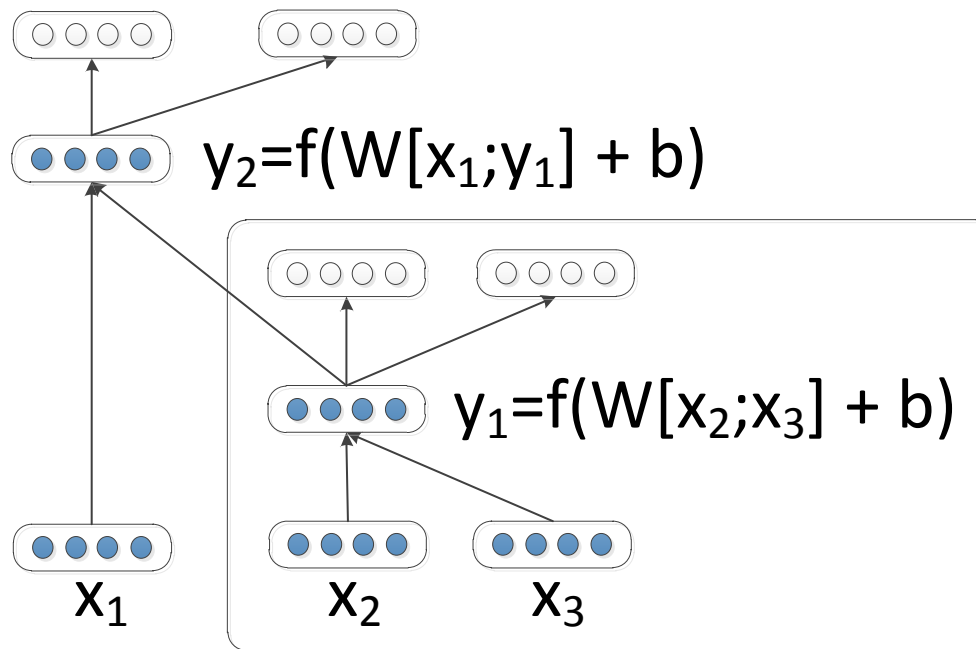
Target	Candidates with human goodness label (1–5) ordered by our system
the united states	the usa (5) the us (5) united states (5) north america (4) united (1) the (1) of the united states (3) america (5) nations (2) we (3)
around the world	around the globe(5) throughout the world(5) across the world(5) over the world(2) in the world(5) of the budget(2) of the world(5)
it would be	it would represent (5) there will be (2) that would be (3) it would be ideal (2) it would be appropriate (2) it is (3) it would (2)
of capital punishment	of the death penalty (5) to death (2) the death penalty (2) of (1)
in the long run	in the long term (5) in the short term (2) for the longer term (5) in the future (5) in the end (3) in the long-term (5) in time (5) of the (1)

Recursive Neural Networks

1. Motivation
2. Recursive Neural Networks for Parsing
3. Theory: Backpropagation Through Structure
4. Recursive Autoencoders
5. Application to Sentiment Analysis and Paraphrase Detection
6. Compositionality Through Recursive Matrix-Vector Spaces
7. Relation classification

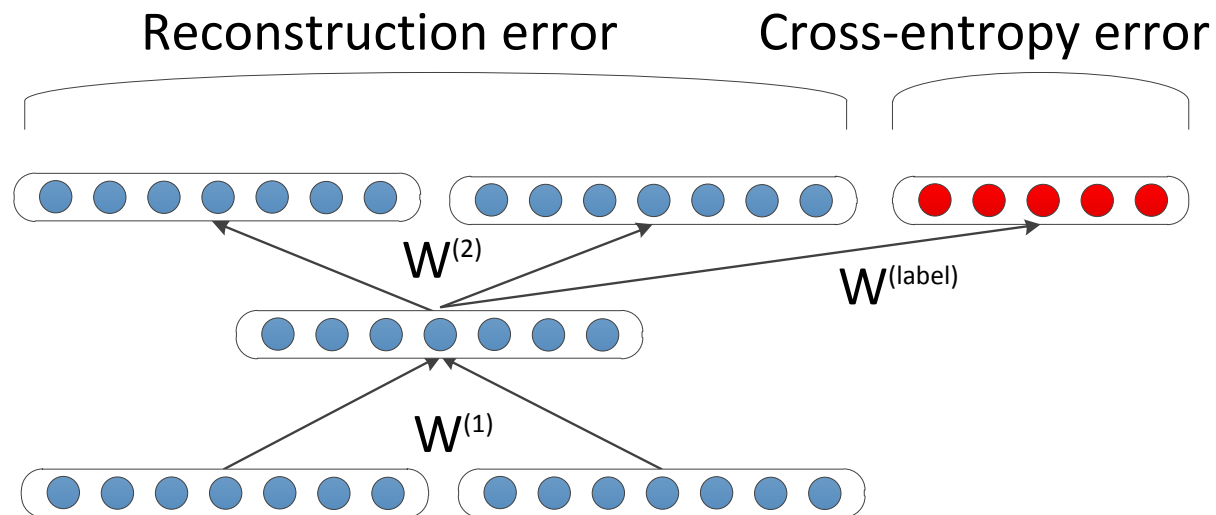
Recursive Autoencoders

- Similar to Recursive Neural Net but instead of a supervised score we compute a reconstruction error at each node. $E_{rec}([c_1; c_2]) = \frac{1}{2} ||[c_1; c_2] - [c'_1; c'_2]||^2$



Semi-supervised Recursive Autoencoder

- In order for representations to capture sentiment, we add a softmax classifier
- Error is a weighted combination of reconstruction error and cross-entropy (distribution likelihood)
- Socher et al. 2011b



Sentiment Detection and Bag-of-Words Models

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

Single Scale Experiments: Movies

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

a film of ideas and wry comic mayhem.

Accuracy of Positive/Negative Sentiment Classification

- Results on movie reviews (MR) and opinions (MPQA).
- All other methods use hand-designed polarity shifting rules or sentiment lexica.
- RAE: no hand-designed features, learns vector

Method	MR	MPQA
Phrase voting with lexicons	63.1	81.7
Bag of features with lexicons	76.4	84.1
Tree-CRF (Nakagawa et al. 2010)	77.3	86.1
RAE (this work)	77.7	86.4



Sorted Negative and Positive N-grams

Most Negative N-grams	Most Positive N-grams
bad; boring; dull; flat; pointless	touching; enjoyable; powerful
that bad; abysmally pathetic	the beautiful; with dazzling
is more boring; manipulative and contrived	funny and touching; a small gem
boring than anything else.; a major waste ... generic	cute, funny, heartwarming; with wry humor and genuine
loud, silly, stupid and pointless. ; dull, dumb and derivative horror film.	, deeply absorbing piece that works as a; ... one of the most ingenious and entertaining;

Learning Compositionality from Movie Reviews

- Probability of being positive of several n-grams

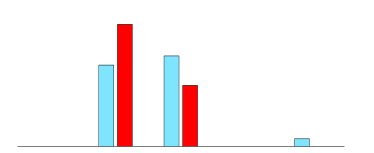
n-gram	P(positive n-gram)
good	0.45
not good	0.20
very good	0.61
not very good	0.15
not	0.03
very	0.23

Sentiment Distribution Experiments

- Learn distributions over multiple complex sentiments → New dataset and task
- Experience Project
 - <http://www.experienceproject.com>
 - “I walked into a parked car”
 - Sorry, Hugs; You rock; Tee-hee ; I understand; Wow just wow
 - Over 31,000 entries with 113 words on average

Sentiment distributions

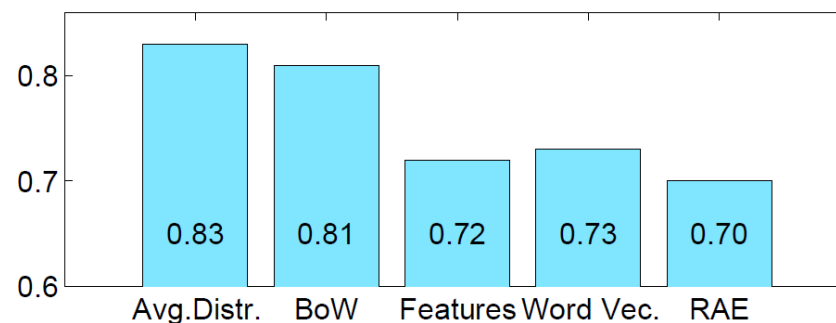
- Sorry, Hugs; You rock; Tee-hee ; I understand; Wow just wow

Predicted and Gold Distribution	Anonymous Confession
	i am a very succesfull business man. i make good money but i have been addicted to crack for 13 years. i moved 1 hour away from my dealers 10 years ago to stop using now i dont use daily but ...
	well i think hairy women are attractive
	Dear Love, I just want to say that I am looking for you. Tonight I felt the urge to write, and I am becoming more and more frustrated that I have not found you yet. I'm also tired of spending so much heart on an old dream. ...

Experience Project most votes results


Method	Accuracy %
Random	20
Most frequent class	38
Bag of words; MaxEnt classifier	46
Spellchecker, sentiment lexica, SVM	47
SVM on neural net word features	46
RAE (this work)	50

Average KL between gold and predicted label distributions:



Paraphrase Detection

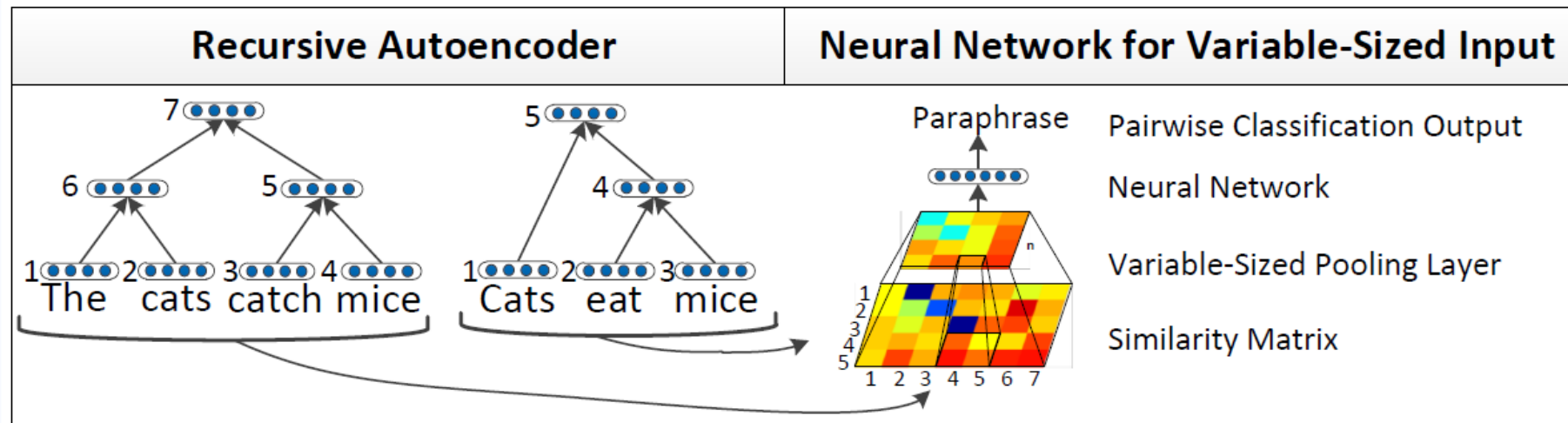
- Pollack said the plaintiffs failed to show that Merrill and Blodget directly caused their losses
- Basically , the plaintiffs did not show that omissions in Merrill's research caused the claimed losses
- The initial report was made to Modesto Police December 28
- It stems from a Modesto police report



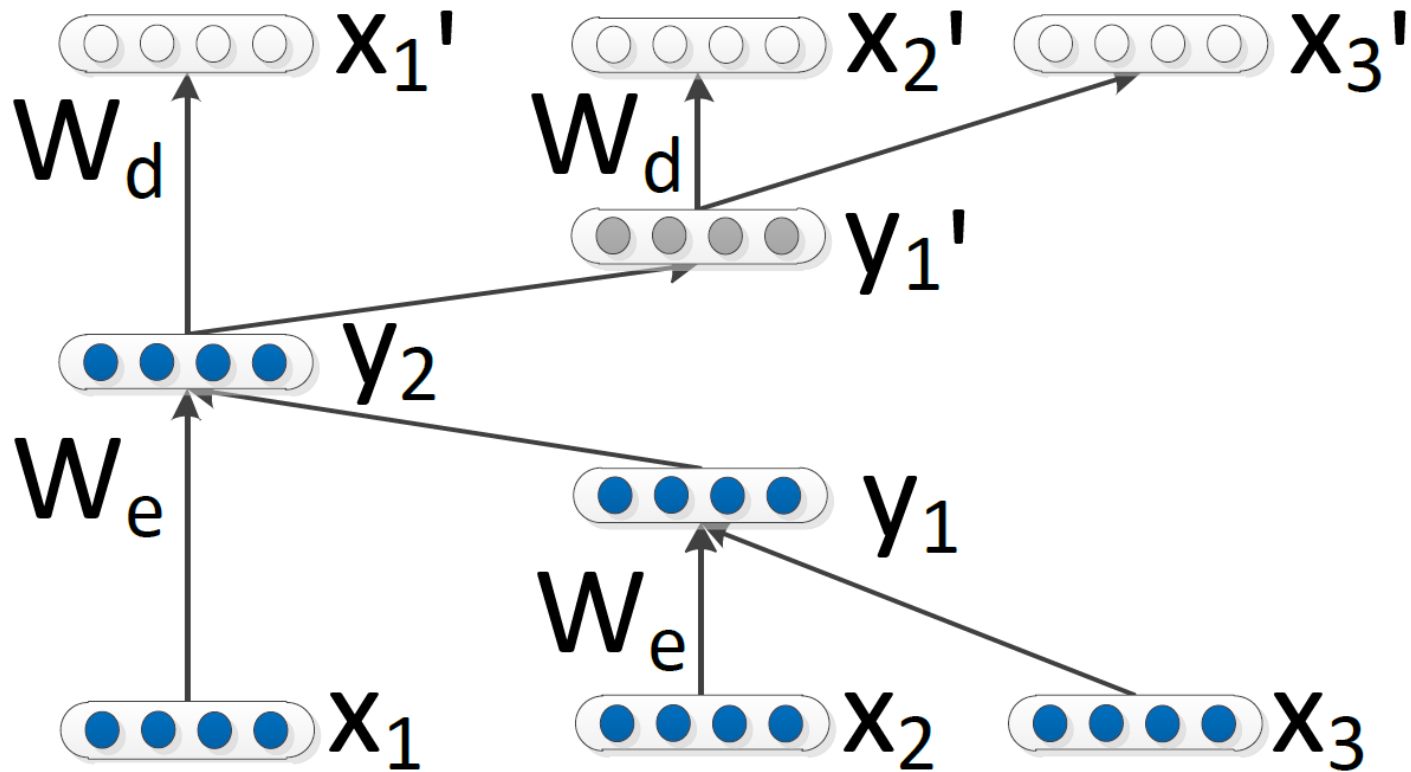
How to compare the
meaning of two
sentences?

Recursive Autoencoders for Full Sentence Paraphrase Detection

- Unsupervised RAE and a pair-wise sentence comparison of nodes in parsed trees
- Socher et al. 2011c



Unsupervised unfolding RAE



Nearest Neighbors of the Unfolding RAE

Center Phrase	RAE	Unfolding RAE
the U.S.	the Swiss	the former U.S.
suffering low morale	suffering due to no fault of my own	suffering heavy casualties
advance to the next round	advance to the final of the UNK 1.1 million Kremlin Cup	advance to the semis
a prominent political figure	the second high-profile opposition figure	a powerful business figure
Seventeen people were killed	Fourteen people were killed	Fourteen people were killed
conditions of his release	conditions of peace, social stability and political harmony	negotiations for their release

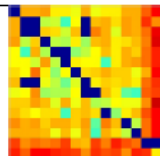
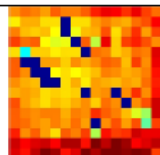
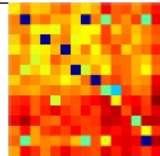
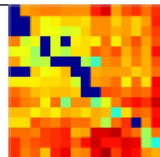
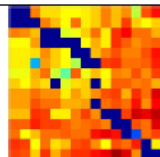
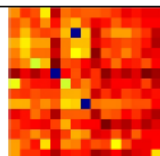
Recursive Autoencoders for Full Sentence Paraphrase Detection

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

Method	Acc.	F1
All Paraphrase Baseline	66.5	79.9
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
Unfolding Recursive Autoencoder (our method)	76.4	83.4



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 Neural Networks

1. Motivation
2. Recursive Neural Networks for Parsing
3. Theory: Backpropagation Through Structure
4. Recursive Autoencoders
5. Application to Sentiment Analysis and Paraphrase Detection
6. Compositionality Through Recursive Matrix-Vector Spaces
7. Relation classification

Compositionality Through Recursive Matrix-Vector Spaces

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

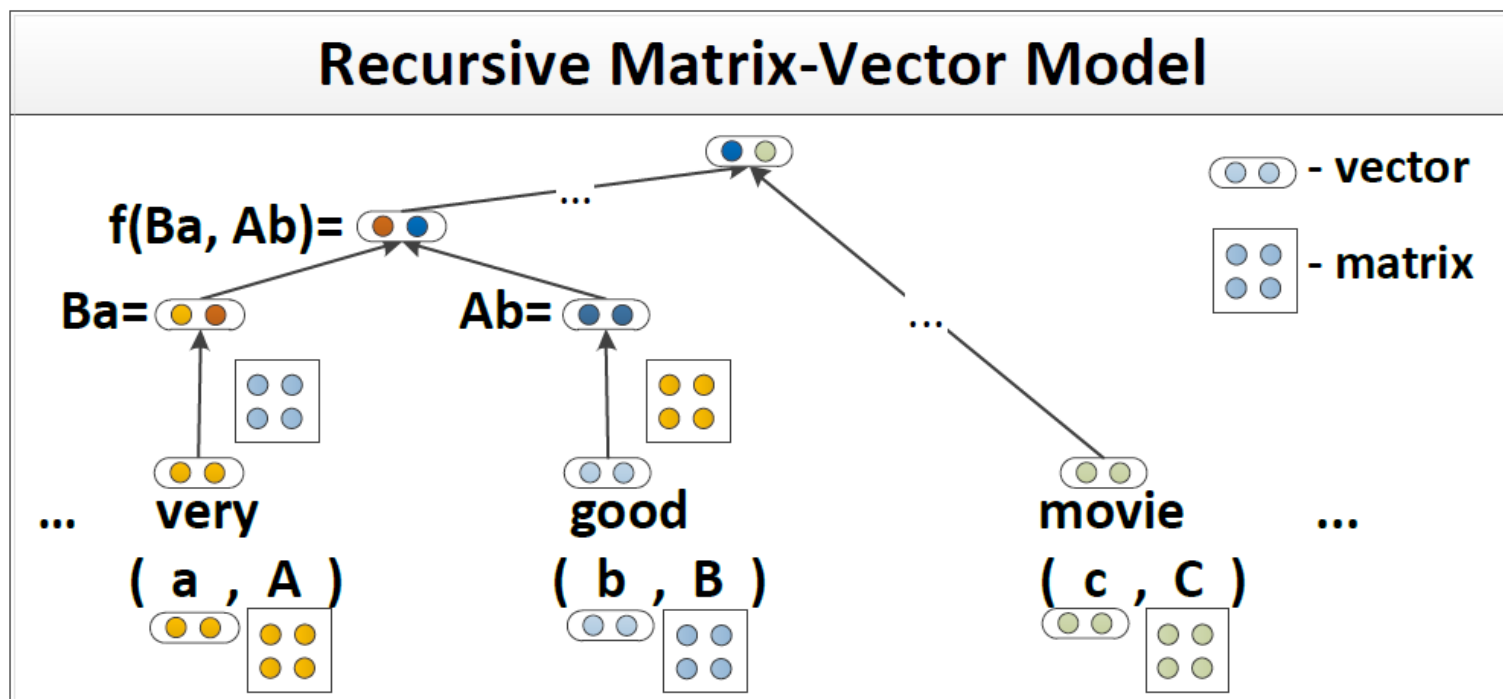
- But what if words act mostly as an operator, e.g. “very” in

very good

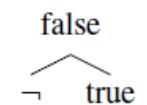
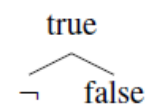
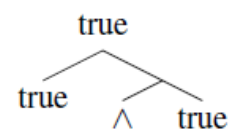
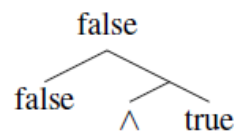
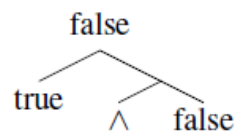
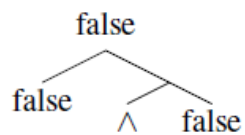
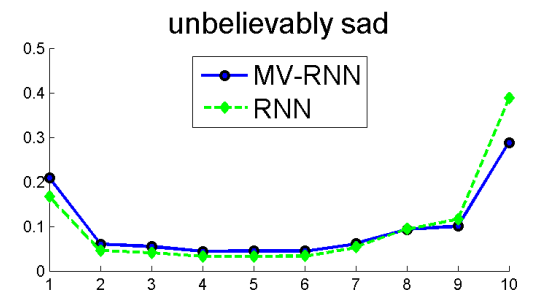
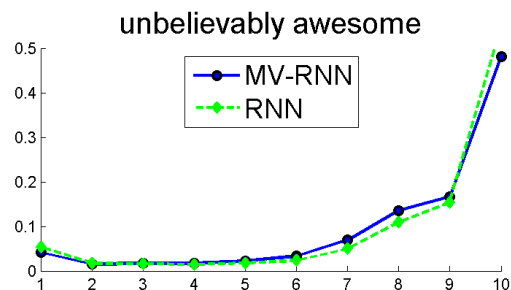
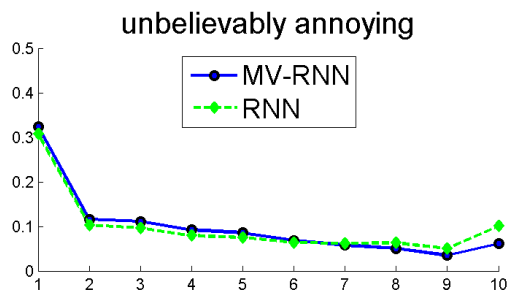
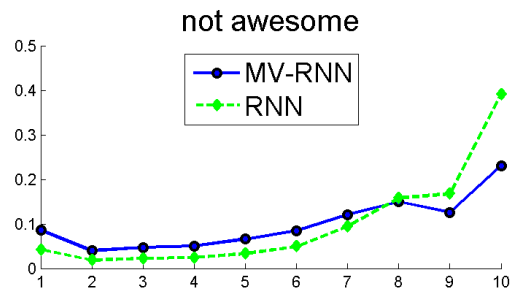
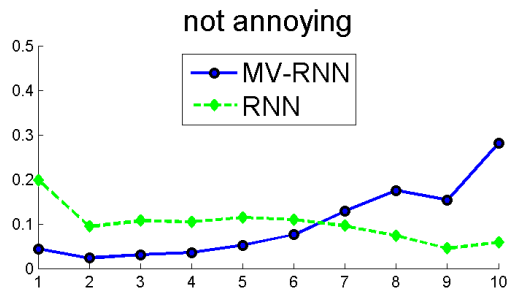
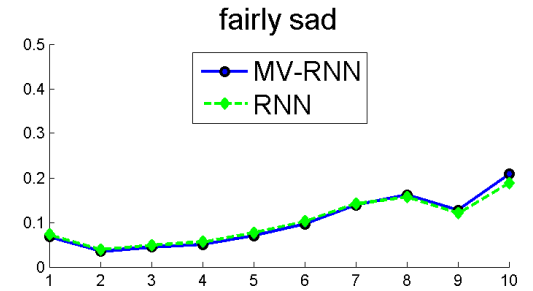
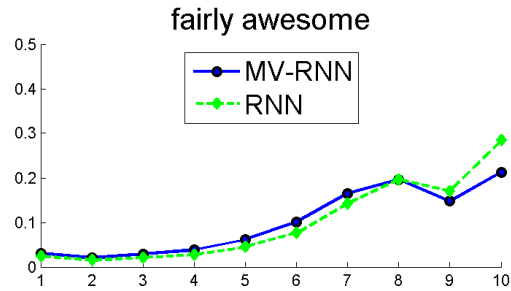
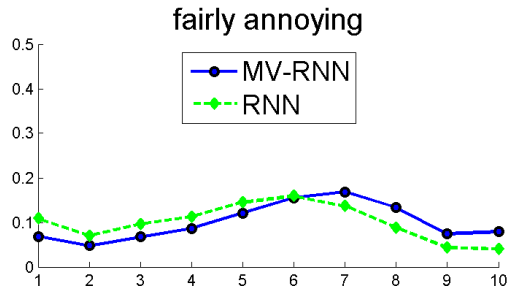
Compositionality Through Recursive Matrix-Vector Recursive Neural Networks

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

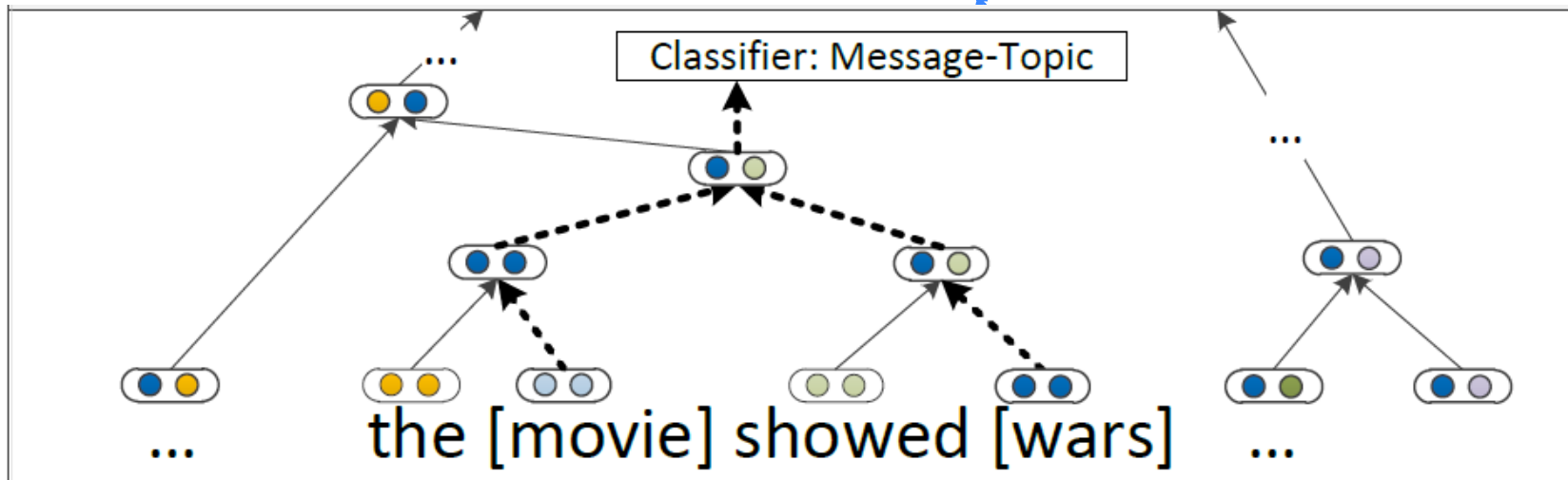
$$p = \text{sigmoid}\left(W \begin{pmatrix} c_2 c_1 \\ c_1 c_2 \end{pmatrix} + b\right)$$



Predicting Sentiment Distributions



MV-RNN for Relationship Classification



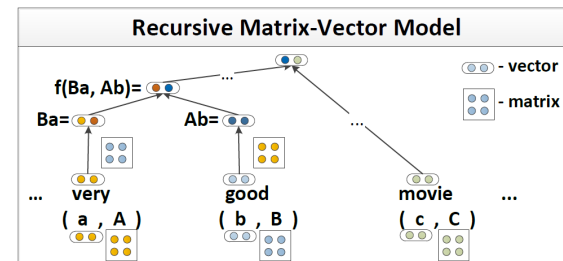
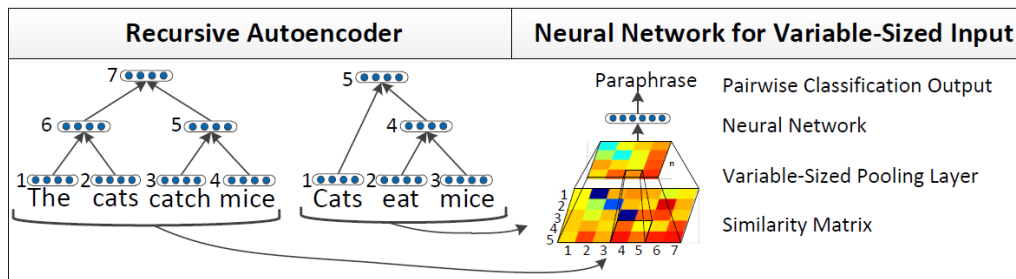
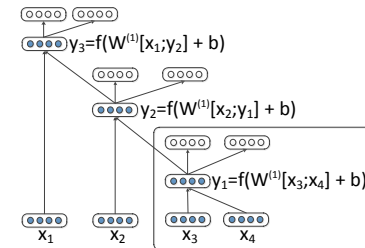
Relationship	Sentence with labeled nouns for which to predict relationships
Cause-Effect(e2,e1)	Avian [influenza] _{e1} is an infectious disease caused by type a strains of the influenza [virus] _{e2} .
Entity-Origin(e1,e2)	The [mother] _{e1} left her native [land] _{e2} about the same time and they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

MV-RNN for Relationship Classification

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	82.4

Summary: Recursive Deep Learning

- Recursive Deep Learning can predict hierarchical structure and classify the structured output using compositional vectors
- State-of-the-art performance on
 - Object detection on Stanford background and MSRC datasets
 - Sentiment Analysis on multiple corpora
 - Paraphrase detection on the MSRP dataset
 - Relation Classification on SemEval 2011, Task8



Part 3

1. Applications
 1. Neural language models
 2. Structured embedding of knowledge bases
 3. Assorted other speech and NLP applications
2. Resources (readings, code, ...)
3. Tricks of the trade
4. Discussion: Limitations, advantages, future directions

Existing NLP Applications

- Language Modeling
 - Speech Recognition
 - Machine Translation
- Part-Of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Sentiment Analysis
- Paraphrasing
- Question-Answering
- Word-Sense Disambiguation

Part 3.1: Applications

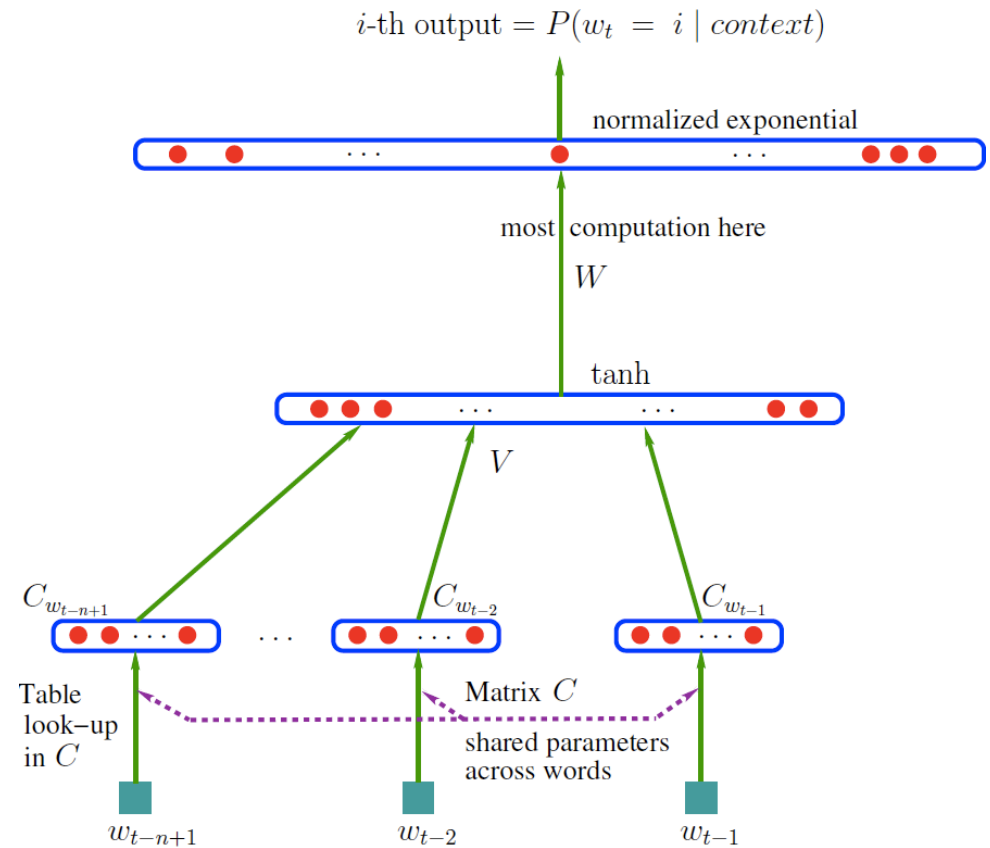
Neural Language Models

Neural Language Model

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

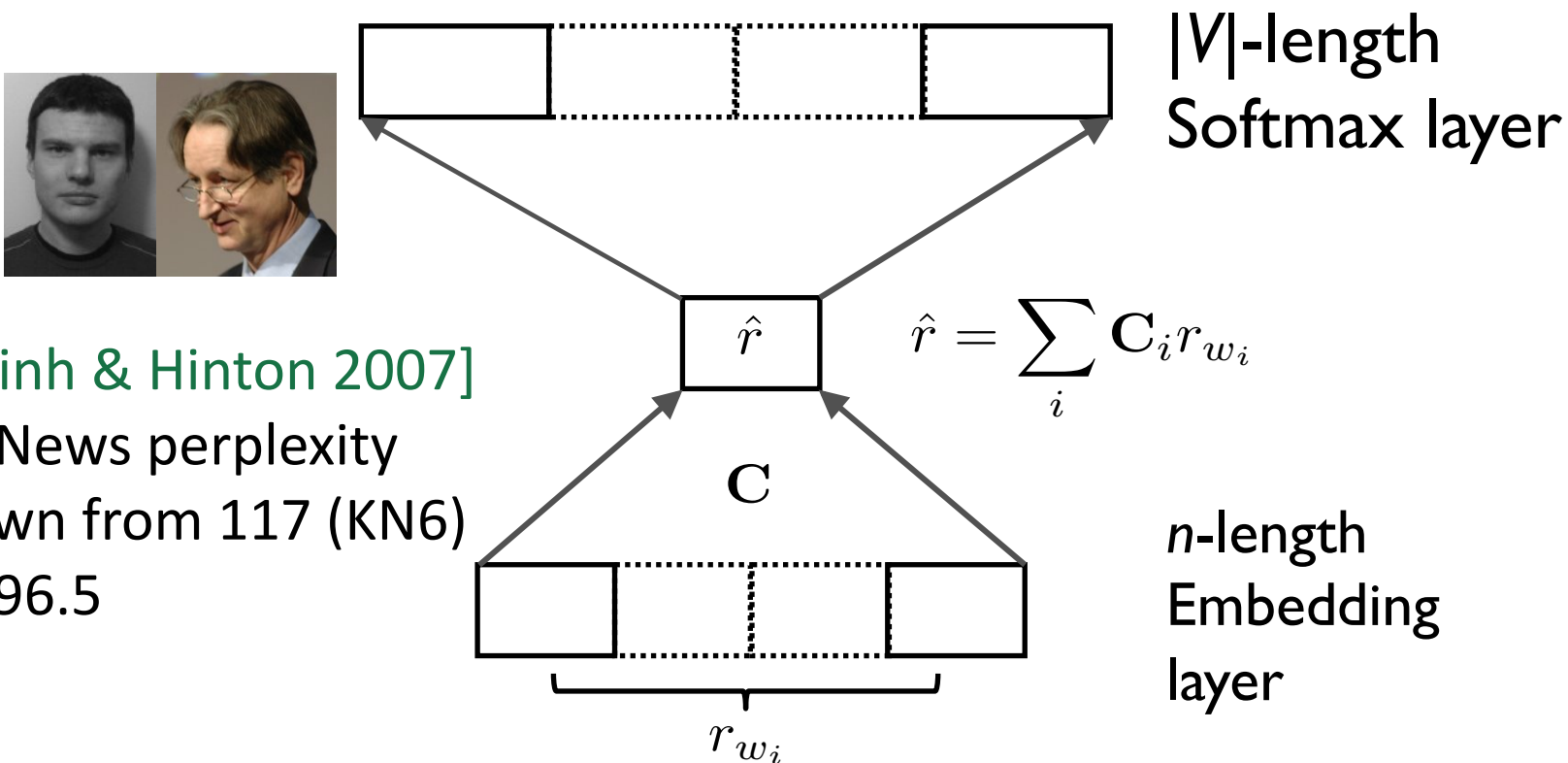


- Each word represented by a distributed continuous-valued code
- Generalizes to sequences of words that are semantically similar to training sequences



Bilinear Language Model

- Even a linear version of the Neural Language Model works better than n-grams



- [Minh & Hinton 2007]
- APNews perplexity down from 117 (KN6) to 96.5

Language Modeling

- Predict next word given previous words
- Standard approach: output = $P(\text{next word} \mid \text{previous word})$
- Applications to Speech, Translation and Compression
- Computational bottleneck: large vocabulary V means that computing the output costs $\# \text{hidden units} \times |V|$.

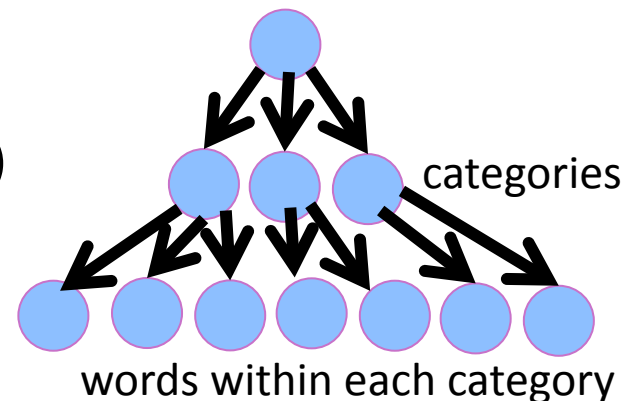
Language Modeling Output Bottleneck

- [Schwenk et al 2002]: only predict most frequent words (short list) and use n-gram for the others



- [Morin & Bengio 2005; Blitzer et al 2005; Mnih & Hinton 2007,2009; Mikolov et al 2011]: **hierarchical representations**, multiple output groups, conditionally computed, predict

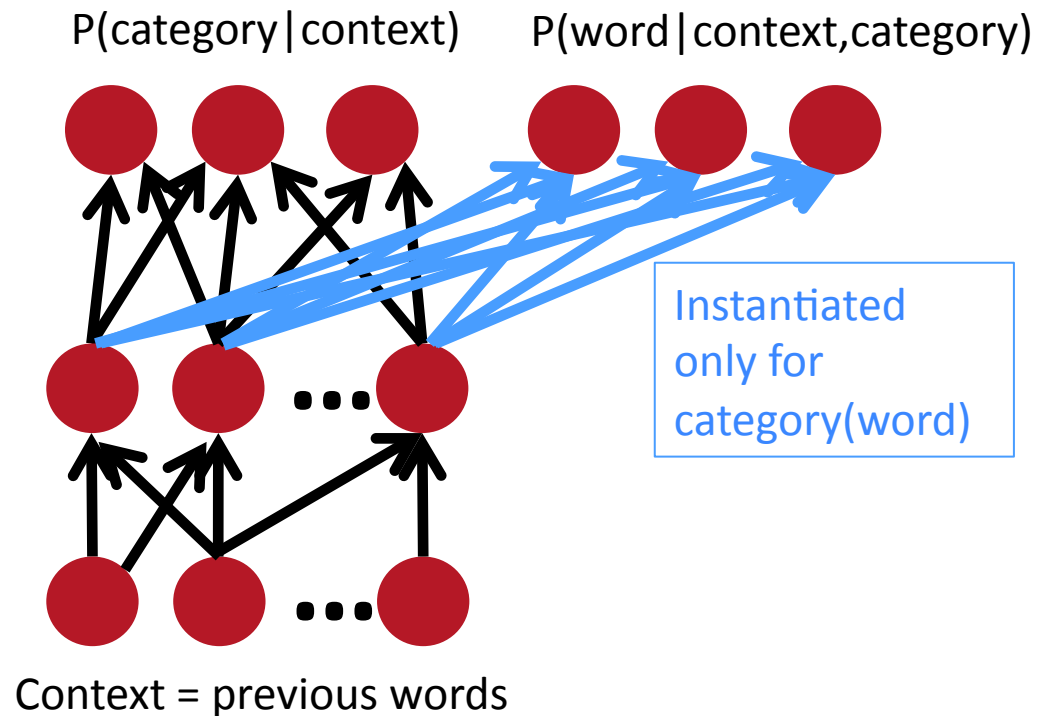
- $P(\text{word category} \mid \text{context})$
- $P(\text{sub-category} \mid \text{context, category})$
- $P(\text{word} \mid \text{context, sub-category, category})$
- With 2 levels, $O(|V|)$ becomes $O(\sqrt{|V|})$
- With d levels, $O(|V|)$ becomes $O(\log(|V|))$
- Hard categories, can be arbitrary [Mikolov et al 2011]



Language Modeling Output Bottleneck: Hierarchical word categories

Compute

$P(\text{word} | \text{category}, \text{context})$
only for
 $\text{category} = \text{category}(\text{word})$



Language Modeling Output Bottleneck: Sampling Methods

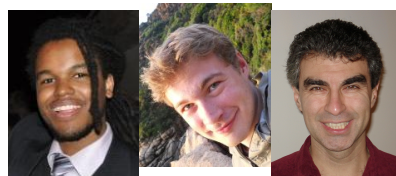
- Importance sampling to recover next-word probabilities [Bengio & Senecal 2003, 2008]



- Sampling a ranking loss [Collobert et al, 2008, 2011]
 - Increase score of observed word's output
 - Decrease score of randomly selected word (negative example)
 - (not anymore outputting probabilities, ok if the goal is just to learn word embeddings)

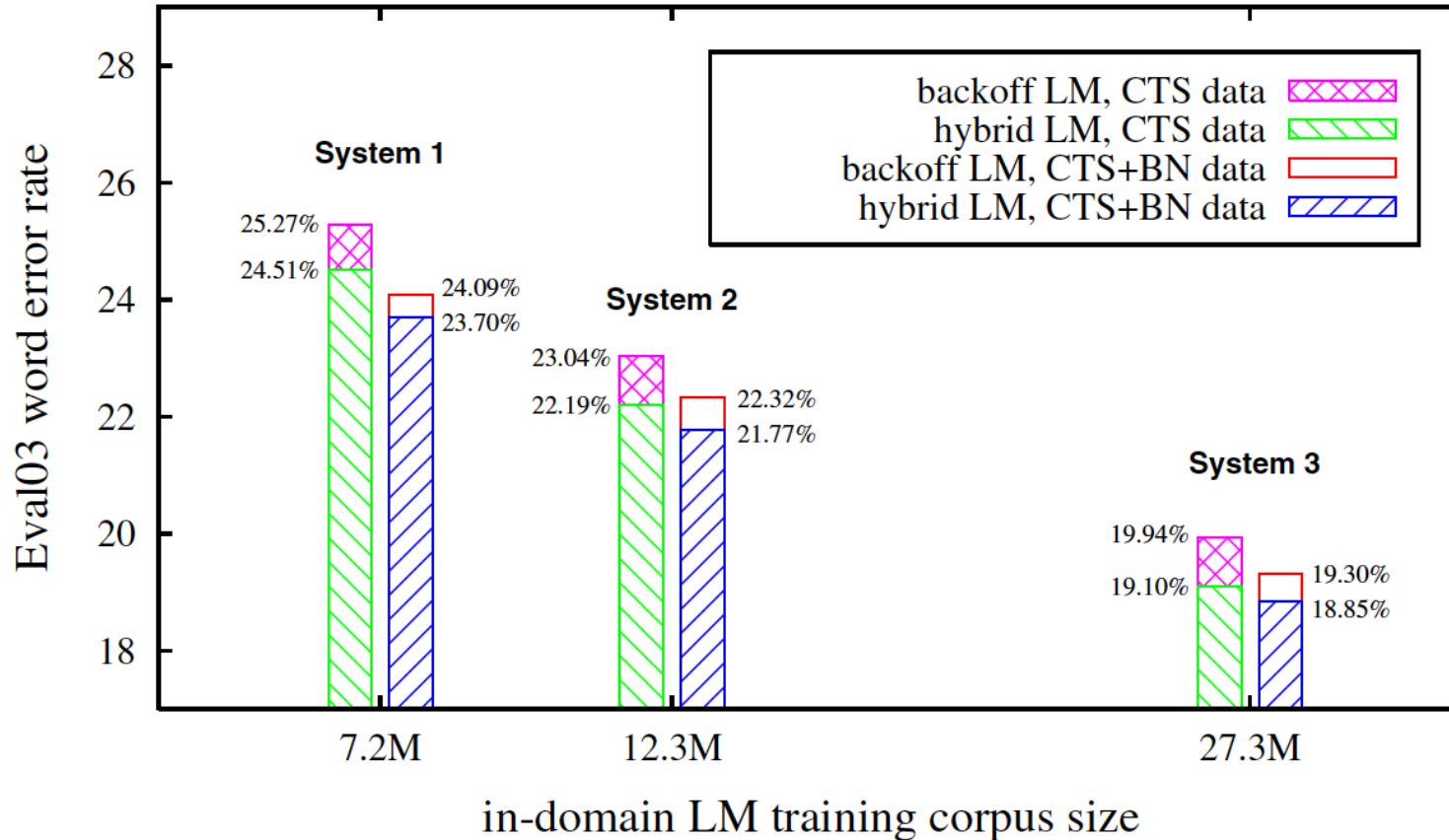


- Importance sampling for reconstructing bag-of-words [Dauphin et al 2011]



Neural Net Language Modeling for ASR

- [Schwenk 2009], real-time ASR, perplexity AND word error rate improve (CTS evaluation set 2003), perplexities go from 50.1 to 45.5



Application to Statistical Machine Translation



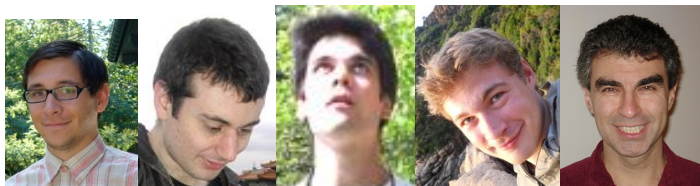
- Schwenk (NAACL 2012 workshop on the future of LM)
 - 41M words, Arabic/English bitexts + 151M English from LDC
- Perplexity down from 71.1 (6 Gig back-off) to 56.9 (neural model, 500M memory)
- +1.8 BLEU score (50.75 to 52.28)
- Can take advantage of longer contexts
- Code: <http://lium.univ-lemans.fr/cs1m/>

Part 3.1: Applications

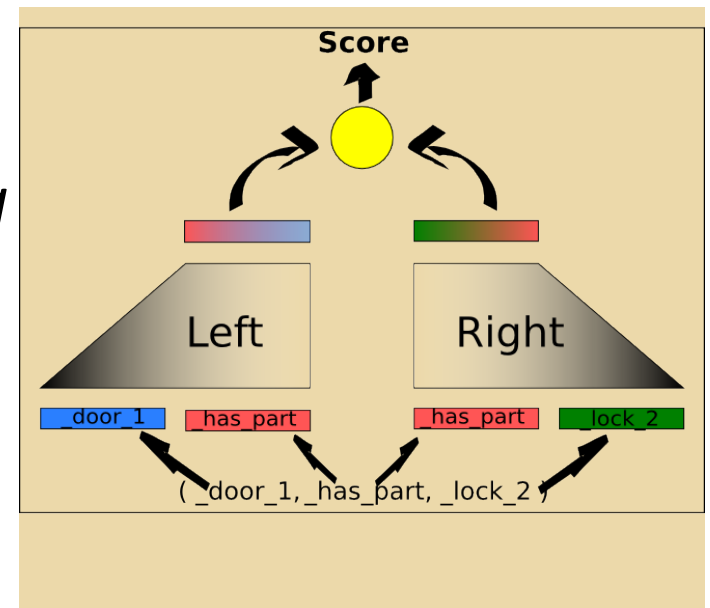
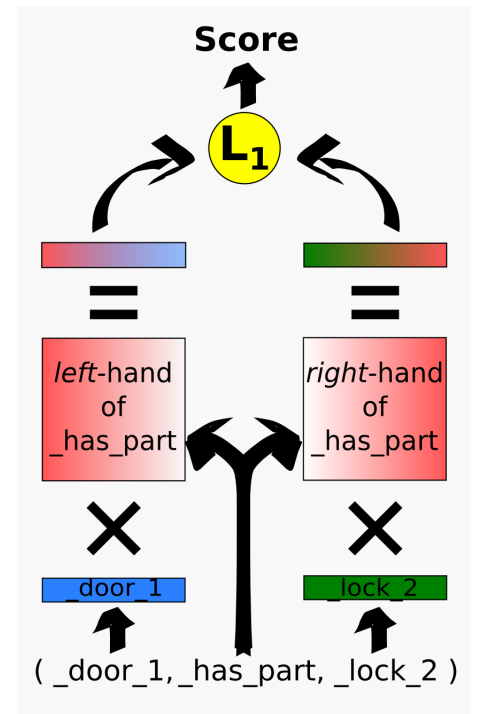
Structured embedding of knowledge bases

Modeling Semantics

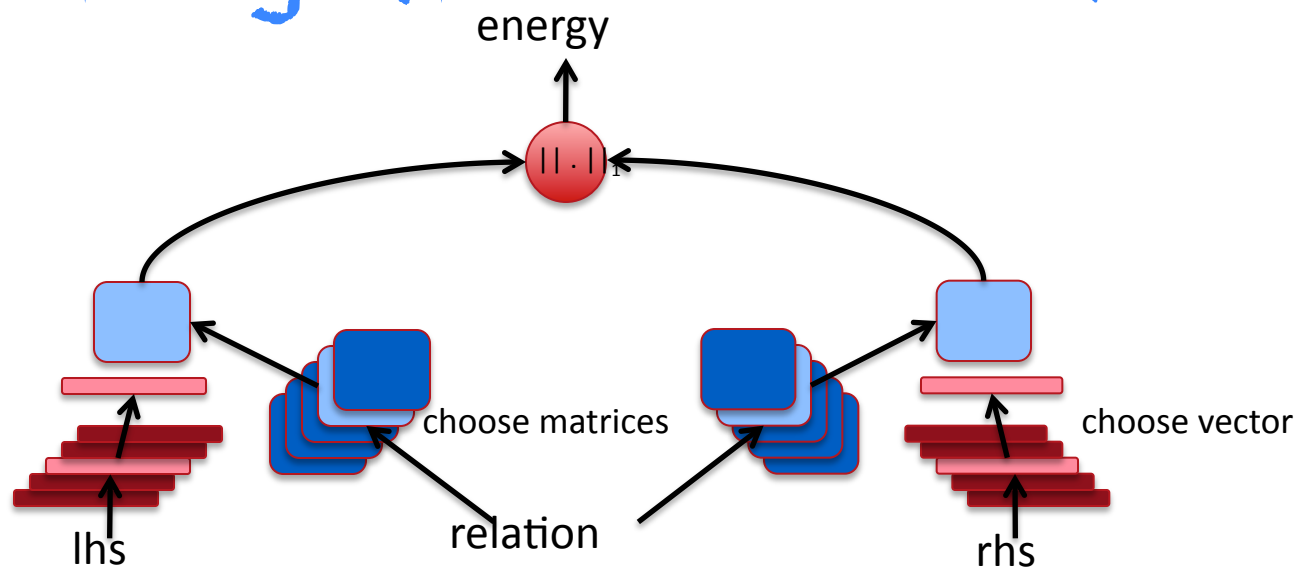
Learning Structured Embeddings of Knowledge Bases, Bordes, Weston, Collobert & Bengio, AAI 2011



Joint Learning of Words and Meaning Representations for Open-Text Semantic Parsing, Bordes, Glorot, Weston & Bengio, AISTATS 2012



Modeling Relations with Matrices



Model (lhs, relation, rhs)

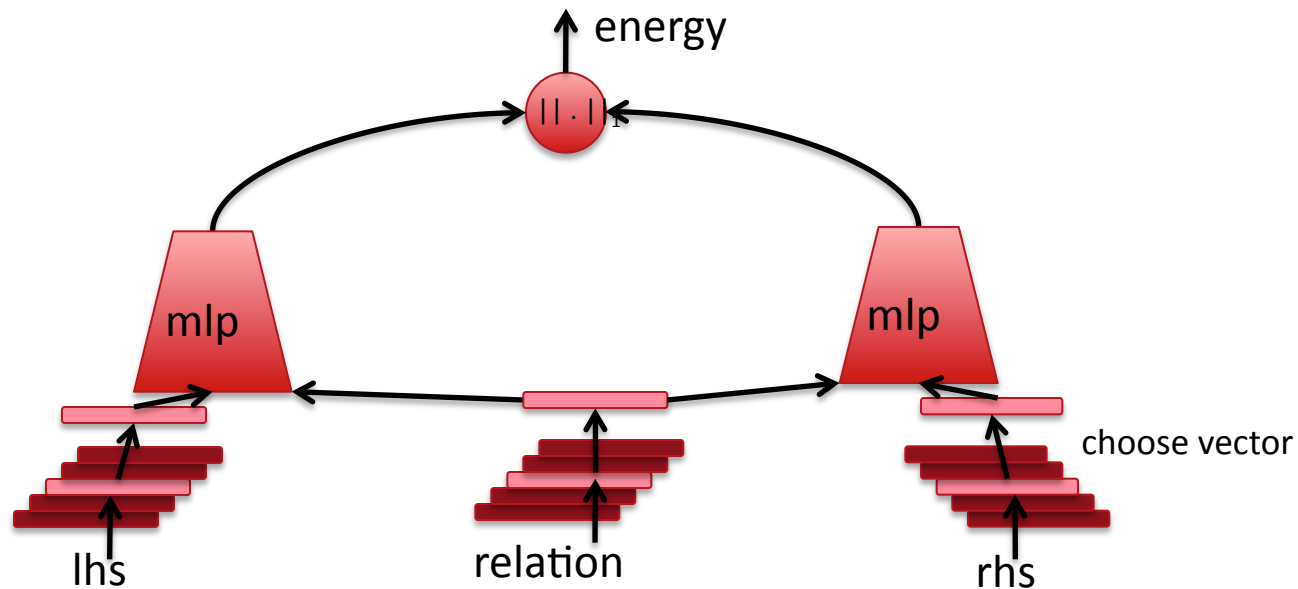
Each concept = 1 embedding vector

Each relation = 2 matrices. **Matrix acts like an operator.**

Ranking criterion

Energy = low for training examples, high o/w

Allowing Relations on Relations



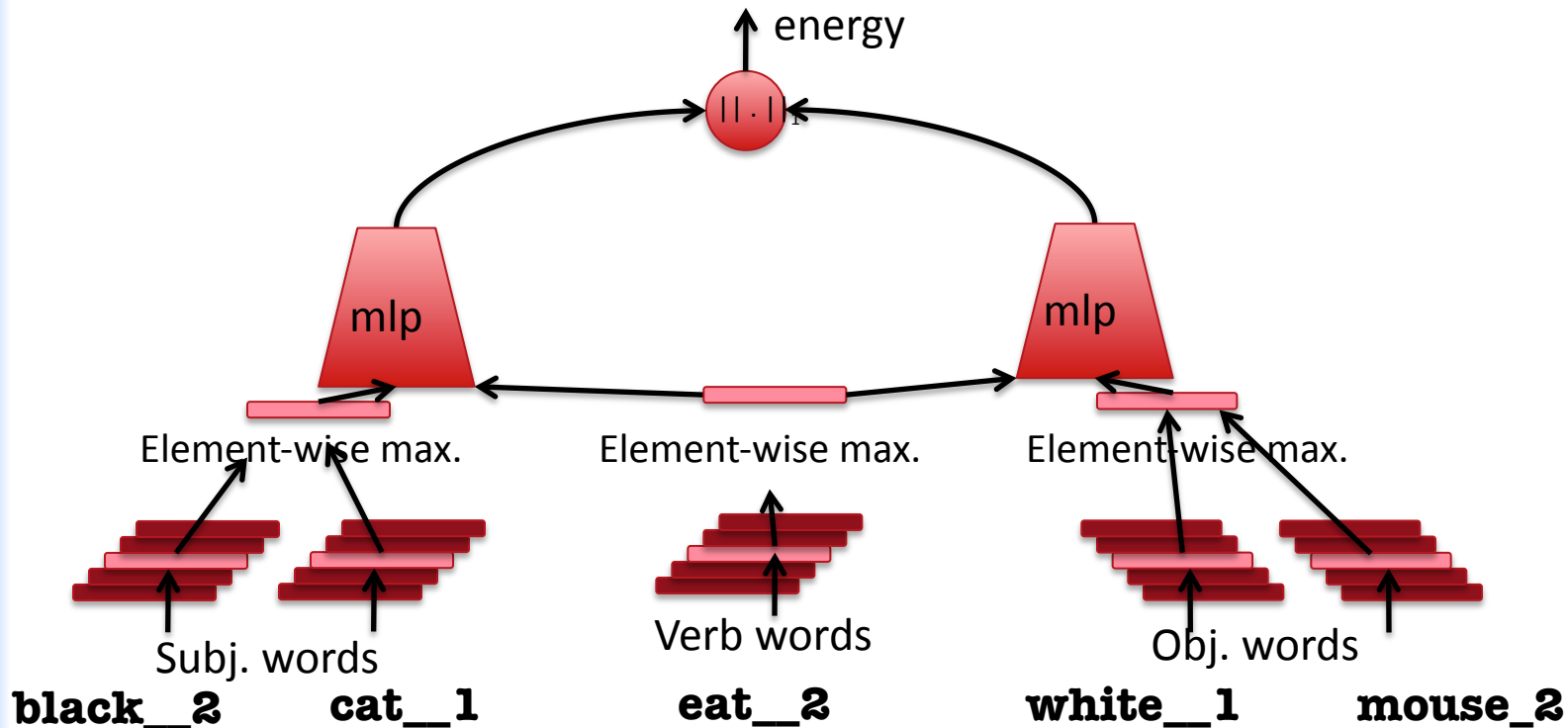
Verb = relation. Too many to have a matrix each.

Each concept = 1 embedding vector

Each relation = 1 embedding vector

Can handle **relations on relations on relations**

Training on Full Sentences



Use SENNA (Collobert 2010) = embedding-based NLP tagger for Semantic Role Labeling, breaks sentence into (subject part, verb part, object part)

→ Use max-pooling to aggregate embeddings of words inside each part

Open-Text Semantic Parsing

- 3 steps:

``A musical score accompanies a television program ."

↓ **Semantic Role Labeling**

(``A musical score", ``accompanies", ``a television program")

↓ **Preprocessing (POS, Chunking, ...)**

((`_musical_JJ` `score_NN`), `_accompany_VB` , `_television_program_NN`)

↓ **Word-sense Disambiguation**

((`_musical_JJ_1` `score_NN_2`), `_accompany_VB_1`, `_television_program_NN_1`)

- last formula defines the Meaning Representation (MR).

Training Criterion

- Intuition: if an entity of a triplet was missing, we would like our model to predict it correctly i.e. to give it the lowest energy. For example, this would allow us to answer questions like “what is part of a car?”
- Hence, for any training triplet $x_i = (lhs_i, rel_i, rhs_i)$ we would like:
 - (1) $E(lhs_i, rel_i, rhs_i) < E(lhs_j, rel_i, rhs_i),$
 - (2) $E(lhs_i, rel_i, rhs_i) < E(lhs_i, rel_j, rhs_i),$
 - (3) $E(lhs_i, rel_i, rhs_i) < E(lhs_i, rel_i, rhs_j),$

That is, the energy function E is trained to rank training samples below all other triplets.

Training Algorithm

pseudo-likelihood + uniform sampling of negative variants

Train by stochastic gradient descent:

1. Randomly select a **positive training triplet** $x_i = (\text{lhs}_i, \text{rel}_i, \text{rhs}_i)$.
2. Randomly select constraint (1), (2) or (3) and an entity \tilde{e} :
 - If constraint (1), construct **negative triplet** $\tilde{x} = (\tilde{e}, \text{rel}_i, \text{rhs}_i)$.
 - Else if constraint (2), construct $\tilde{x} = (\text{lhs}_i, \tilde{e}, \text{rhs}_i)$.
 - Else, construct $\tilde{x} = (\text{lhs}_i, \text{rel}_i, \tilde{e})$.
3. If $E(x_i) > E(\tilde{x}) - 1$ make a **gradient step** to minimize:
 $\max(0, 1 - E(\tilde{x}) + E(x_i))$.
4. Constraint embedding vectors to norm 1

Question Answering: implicitly adding new relations to WN or FB

	Model (All)	<i>TextRunner</i>
<i>lhs</i>	_army_NN_1	<i>army</i>
<i>rel</i>	_attack_VB_1	<i>attacked</i>
top ranked <i>rhs</i>	_troop_NN_4 _armed_service_NN_1 _ship_NN_1 _territory_NN_1 _military_unit_NN_1	<i>Israel</i> <i>the village</i> <i>another army</i> <i>the city</i> <i>the fort</i>
top ranked <i>lhs</i>	_business_firm_NN_1 _person_NN_1 _family_NN_1 _payoff_NN_3 _card_game_NN_1	<i>People</i> <i>Players</i> <i>one</i> <i>Students</i> <i>business</i>
<i>rel</i>	_earn_VB_1	<i>earn</i>
<i>rhs</i>	_money_NN_1	<i>money</i>

MRs inferred from text define triplets between WordNet synsets.

Model captures knowledge about relations between nouns and verbs.

→ Implicit addition of new relations to WordNet!

→ Generalize Freebase!

Embedding Near Neighbors of Words & Senses

<p>_mark_NN</p> <p>_indication_NN _print_NN_3 _print_NN _roll_NN _pointer_NN</p>	<p>_mark_NN_1</p> <p>_score_NN_1 _number_NN_2 _gradation_NN _evaluation_NN_1 _tier_NN_1</p>	<p>_mark_NN_2</p> <p>_marking_NN_1 _symbolizing_NN_1 _naming_NN_1 _marking_NN _punctuation_NN_3</p>
<p>_take_VB</p> <p>_bring_VB _put_VB _ask_VB _hold_VB _provide_VB</p>	<p>_canary_NN</p> <p>_sea_mew_NN_1 _yellowbird_NN_2 _canary_bird_NN_1 _larus_marinus_NN_1 _mew_NN</p>	<p>_different_JJ_1</p> <p>_eccentric_NN _dissimilar_JJ _same_JJ_2 _similarity_NN_1 _common_JJ_1</p>

Word Sense Disambiguation

- Senseval-3 results
(only sentences with
Subject-Verb-Object
structure)

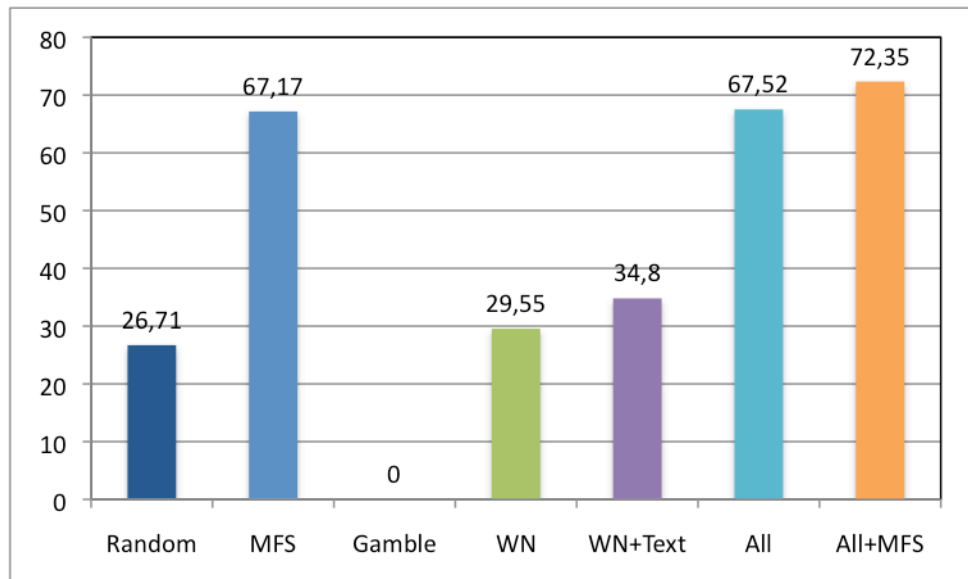
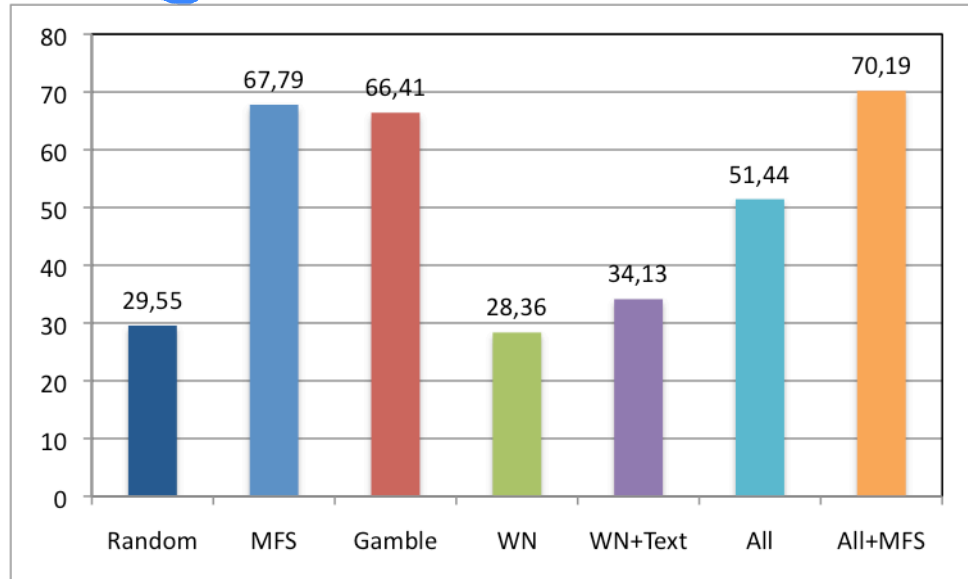
MFS=most frequent sense

All=training from all sources

Gamble=Decadt et al 2004
(Senseval-3 SOA)

- XWN results

XWN = eXtended WN

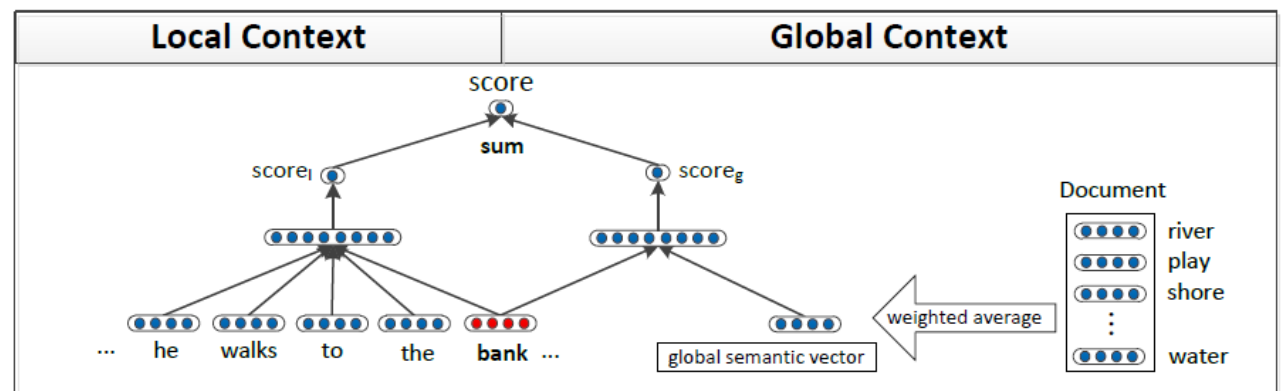


Part 3.1: Applications

Assorted other speech and NLP
applications

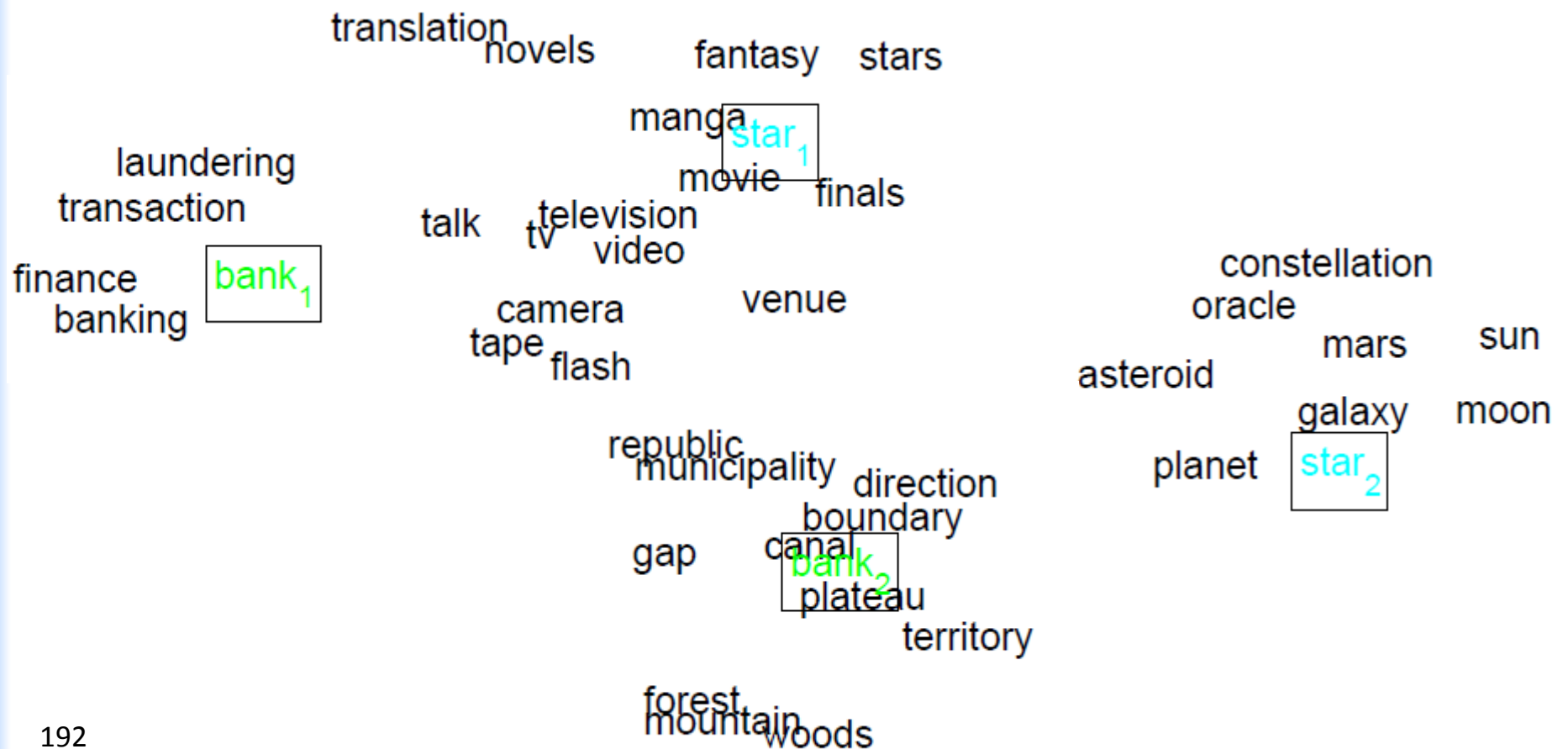
Learning Multiple Word Vectors

- 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)



Recurrent Neural Net Language Modeling for ASR

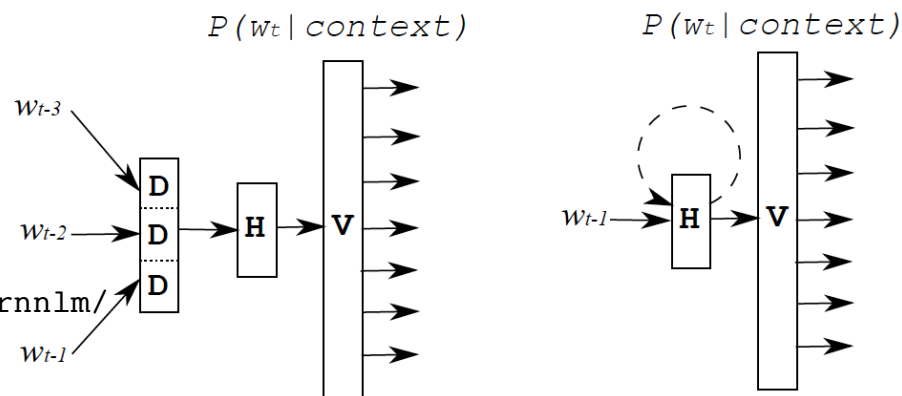
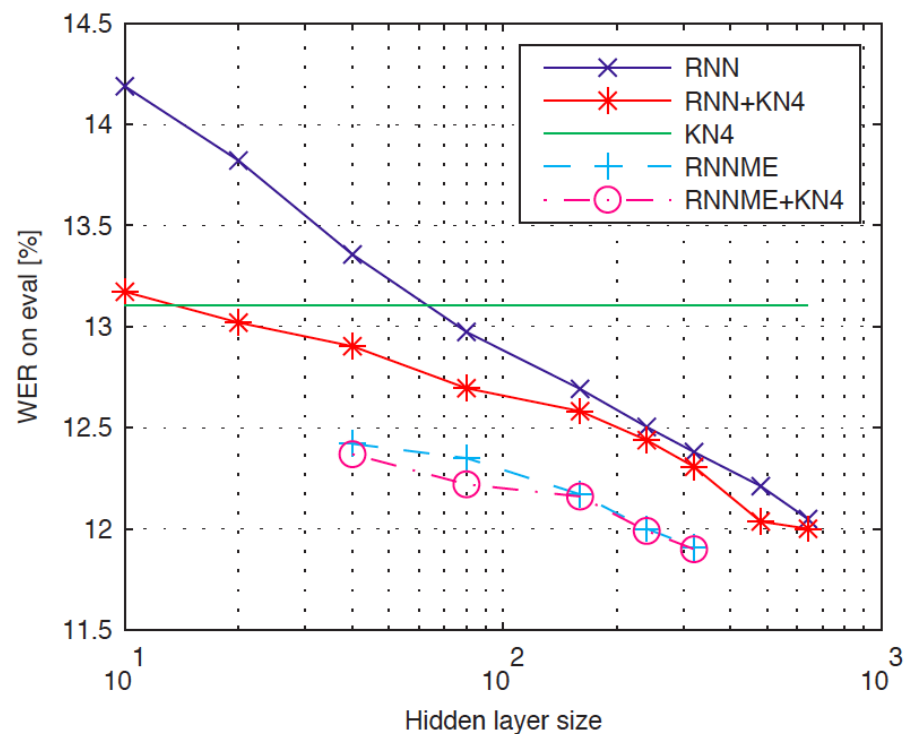
- [Mikolov et al 2011]
Bigger is better...
experiments on Broadcast
News NIST-RT04



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/>



Phoneme-Level Acoustic Models

- [Mohamed et al, 2011, IEEE Tr.ASLP]

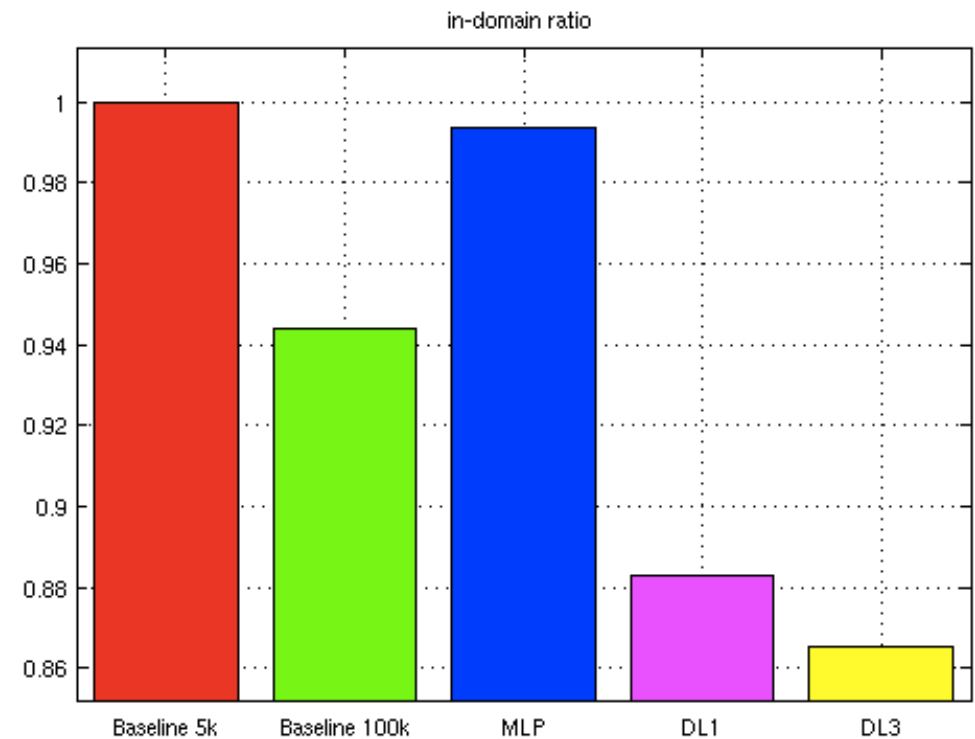


- Unsupervised pre-training as Deep Belief Nets (a stack of RBMs), supervised fine-tuning to predict phonemes
- Phoneme classification on TIMIT:
 - CD-HMM: 27.3% error
 - CRFs: 26.6%
 - Triphone HMMs w. BMML: 22.7%
 - Unsupervised DBNs: 24.5%
 - Fine-tuned DBNs: 20.7%
- Improved version by Dong Yu is **RELEASED IN MICROSOFT'S ASR** system for Audio Video Indexing Service

Sentiment Analysis on Bag-of-word representations

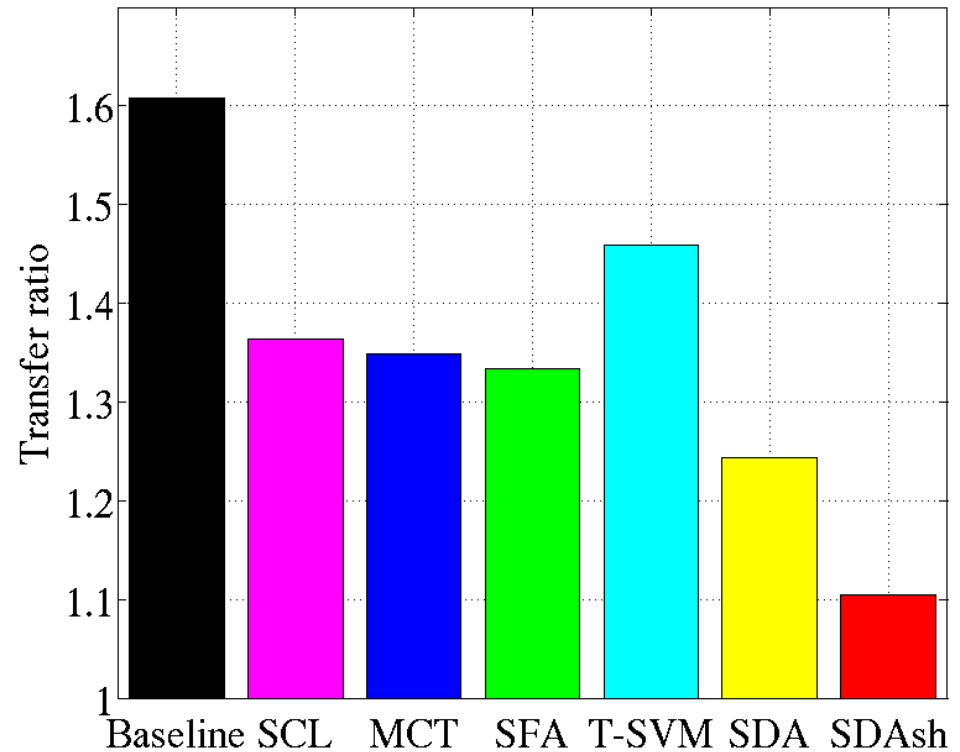


- [Glorot et al, ICML 2011] beats SOTA on Amazon benchmark
- Bag-of-words input
- Embeddings pre-trained in denoising auto-encoder with rectifier activation functions and sampled reconstruction
- Disentangling effect (features specialize to domain or sentiment)



Sentiment Analysis: Transfer Learning

- 25 Amazon.com domains: toys, software, video, books, music, beauty, ...
- Unsupervised pre-training of input space on all domains
- Supervised SVM on 1 domain, generalize out-of-domain
- Baseline: bag-of-words + SVM



Part 3.2: Resources

Resources: Code and References

Papers, 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>

Software


- Theano (Python CPU/GPU mathematical and deep learning library
 - <http://deeplearning.net/software/theano/>
- Senna by (Collobert et al 2011, JMLR)
 - POS, Chunking, NER, SRL
 - State-of-the-art performance on many tasks
 - <http://ronan.collobert.com/senna/>
 - 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



Part 3.3: Deep Learning Tricks

Deep Learning Tricks

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 and L2 weight decay
 - Sparsity regularization
 - Debugging
 - How to efficiently search for hyper-parameter configurations

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, θ = parameter vector, and ϵ_t = learning rate.
- Ordinary gradient descent is a batch method, very slow, **should never be used**. Use 2nd 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.,

$$\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$$

with hyper-parameters ϵ_0 and τ .

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.

$$L = L(s_T(s_{T-1}(\dots s_{t+1}(s_t, \dots))))$$
$$\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_{T-1}} \dots \frac{\partial s_{t+1}}{\partial s_t}$$

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



Early Stopping

- Beautiful **FREE LUNCH** (no need to launch many different training runs for each value of hyper-parameter for #iterations)
- Monitor validation error during training (after visiting # examples a multiple of validation set size)
- Keep track of parameters with best validation error and report them at the end
- If error does not improve enough (with some patience), stop.

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 $\sim \text{Uniform}(-r,r)$, r inversely proportional to fan-in (previous layer size) and fan-out (next layer size):

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

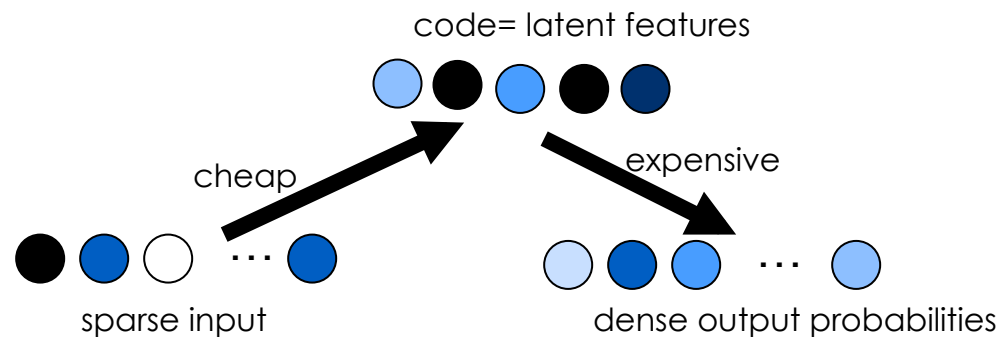
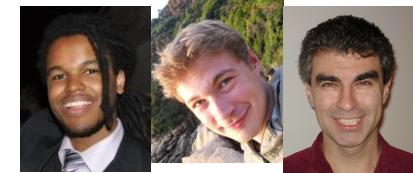
for tanh units (and 4x bigger for sigmoid units).

Note: for embedding weights, fan-in=1 and we don't care about fan-out, Collobert uses $\text{Uniform}(-1,1)$.

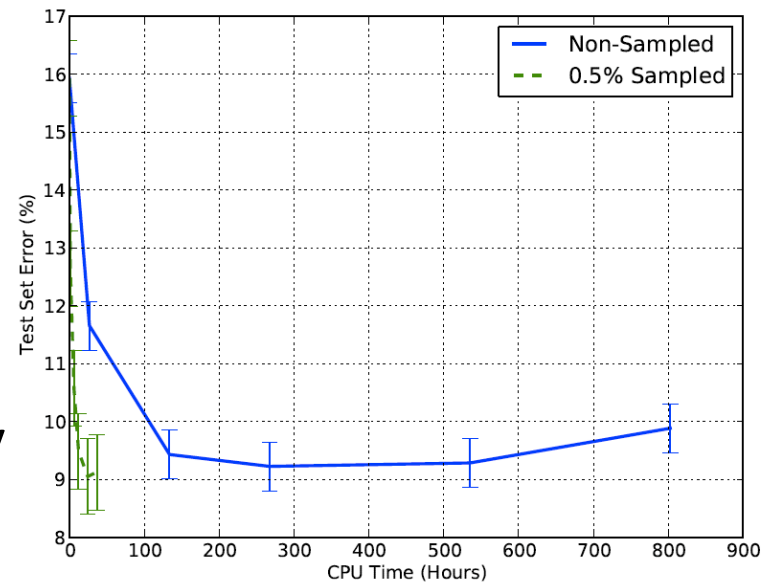
Sampled Reconstruction Trick

[Dauphin et al, ICML 2011]

- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional



- Applied to bag-of-words input for sentiment analysis, with denoising auto-encoders
- Always reconstruct the non-zeros in the input, and reconstruct as many randomly chosen zeros



Part 3.4: Discussion

Discussion: Limitations, Advantages, Future Directions

Outlook

- Current strengths and weaknesses
- Representing meaning at different levels
- Inference challenges
- Reaping the benefits of more abstract features: better transfer, e.g. to other languages

Criticisms

- Many algorithms and variants (burgeoning field)
- Many hyper-parameters → use multi-core machines, clusters and random sampling for cross-validation
- Not always obvious how to combine with existing NLP → learn more abstract features, separate parsing and semantic analysis, **your research here**
- Slower to train linear models → only by a small constant factor, and much more compact than non-parametric (e.g. n-gram models)

Criticisms

- You're learning everything. Better to encode prior knowledge about structure of language.
 - Wasn't there a similar machine learning vs. linguists debate in NLP ~20 years ago....
- Research goal: Just like we now use off-the-shelf SVM packages, we can try to build off-the-shelf NLP classification packages that are given as input only raw text and a label.

Problems with model interpretability

- Ways to interpret the output of models/ weights on features
 - More difficult compared to systems with small sets of hand-designed features, but possible through low-dimensional projections of word vectors
- Learning about language through doing nlp
 - Can be done by careful visualization, e.g. “flawed” in sentiment analysis
- 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).

Inference Challenges

- Many latent variables involved in understanding language (sense ambiguity, parsing, semantic role)
- Almost any inference mechanism can be combined with deep learning
- See [Bottou, Bengio, LeCun 97], [Graves 2012]



- Complex inference can be hard (exponentially) and needs to be approximate → learn to perform inference

Learning Multiple Levels of Abstraction

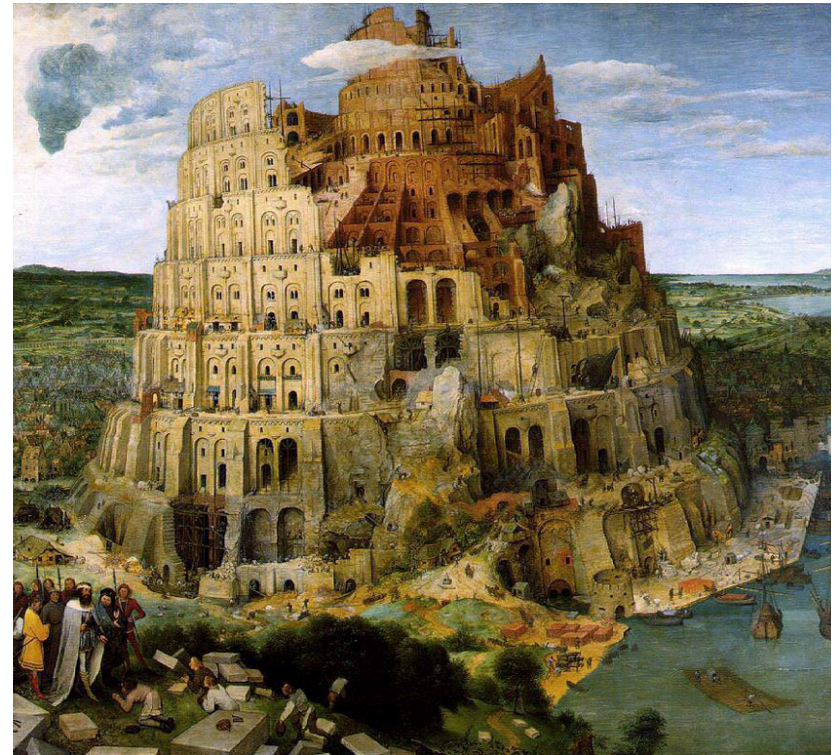
- The big payoff of deep learning is to allow learning higher levels of abstraction
- Going from symbols to embeddings already helps, and recent years have shown that phrase and sentence representations work too, but the space of possibilities is much wider there
- Higher-level abstractions disentangle the factors of variation, which allows much easier generalization and transfer

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 inference and well suited for multi-core CPUs/GPUs

Transfer Learning

- Application of deep learning could be in areas where there are not enough labeled data but a transfer is possible
- Domain adaptation already showed that effect, thanks to **unsupervised feature learning**
- Transfer to resource-poor languages would be a great application [Gouws, PhD proposal 2012]



The End

