CS291K – Advanced Data Mining

Instructor: Xifeng Yan
Computer Science
University of California at Santa Barbara

Word Embeddings - 2

Lecturer: Yu Su
Computer Science
University of California at Santa Barbara

How to let a computer understand meaning?

A cat sits on a mat.

#_$@^_&*^&_()_@_+@^=

Distributional semantics

- You can get a lot of value by representing a word by means of its neighbors (context)

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

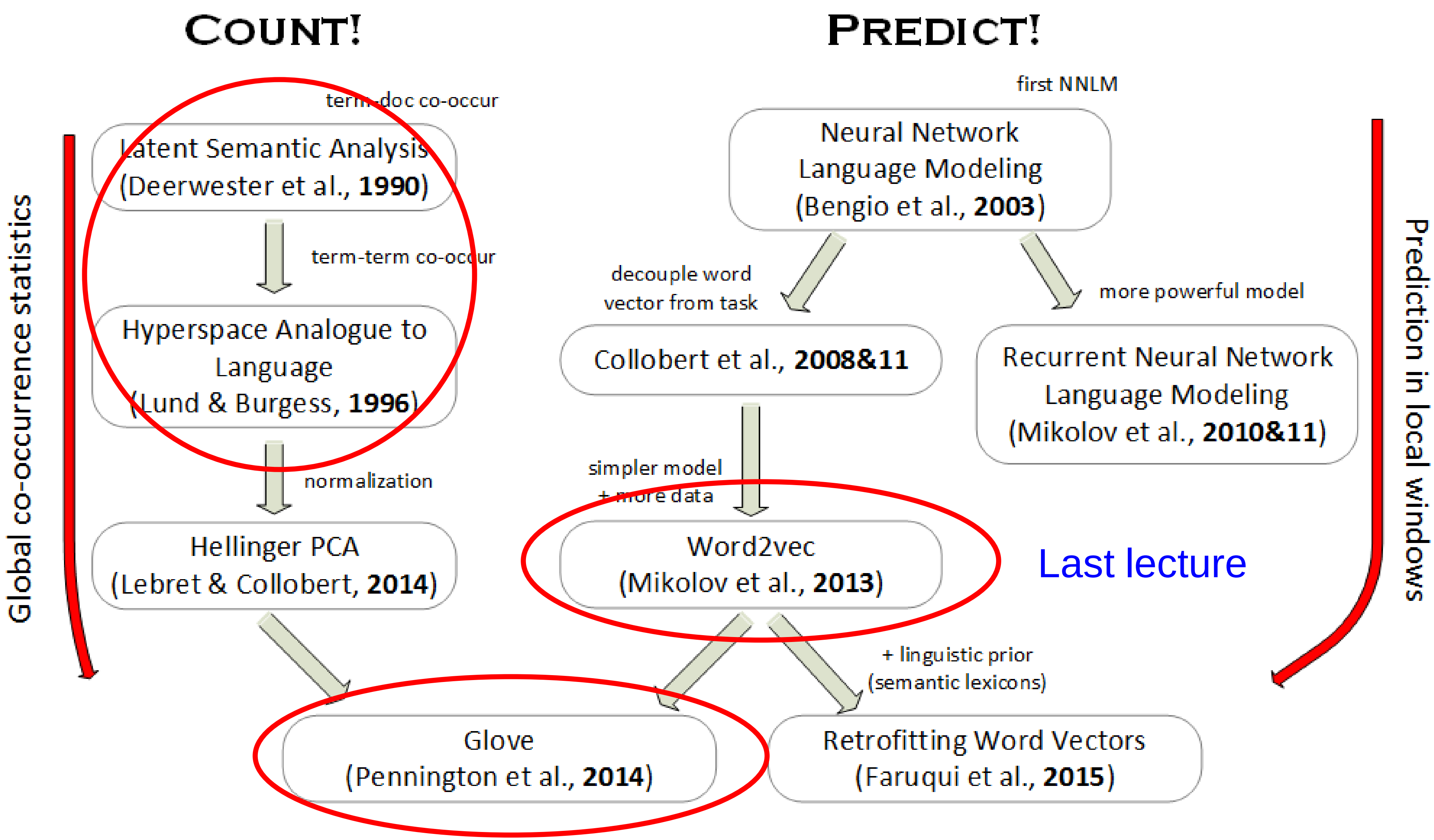
- One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in

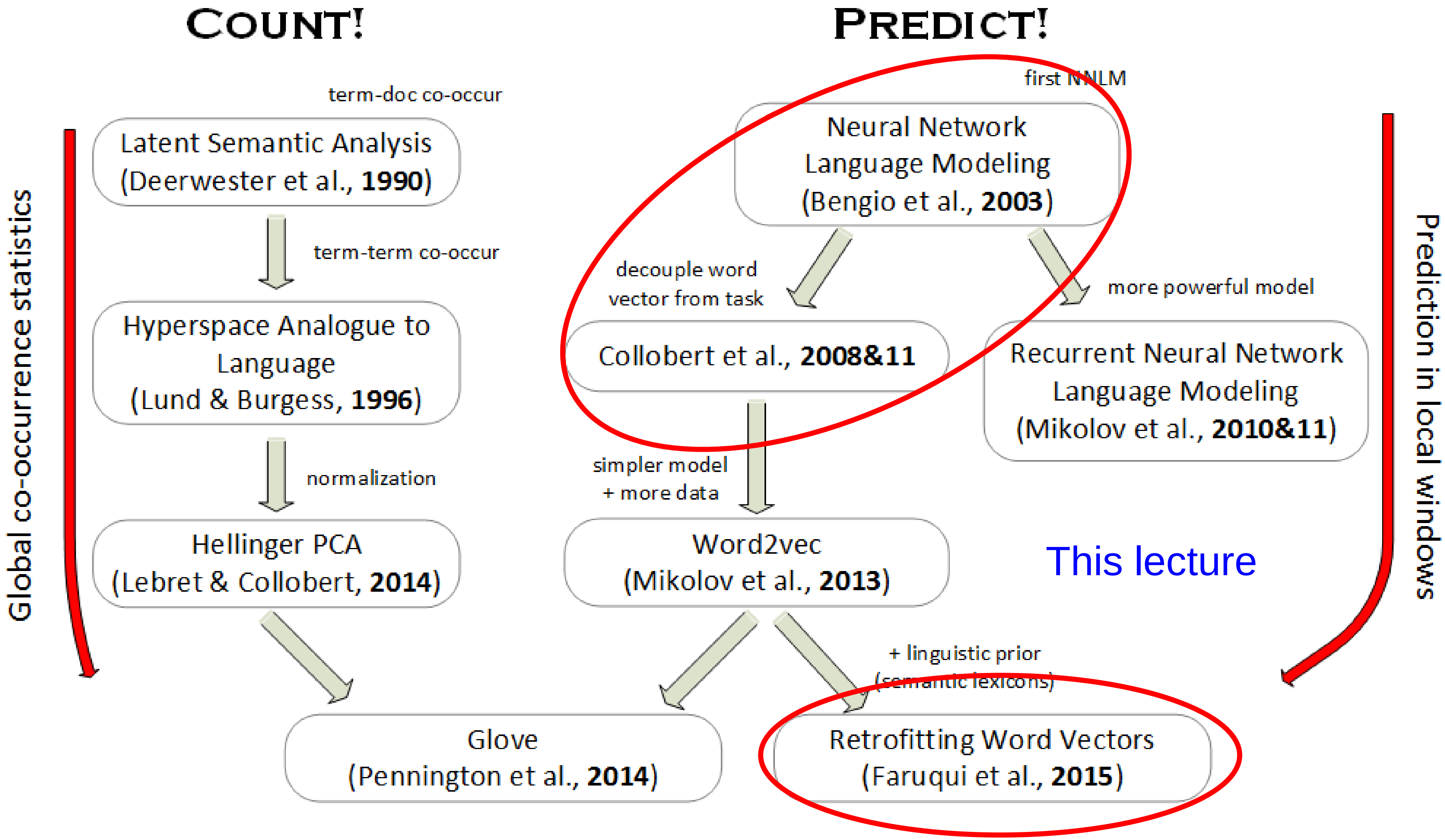
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

History of word embedding



History of word embedding



Different embeddings are based on different priors

Latent semantic analysis

"Words occur in same documents should be similar"

Word2vec

"Words occur in similar contexts should be similar"

Neural Network Language Modeling

"Word vectors should give plausible sentences high probability"

Collabert et al., 2008 & 2011

"Word vectors should facilitate downstream classification tasks"

Faruqui et al., 2015

"Words should follow linguistic constraints from semantic lexicons"

"Words occur in similar contexts should be similar"

I just played with my dog.
I just played with my cat.
My dog likes to sleep on my bed.
My cat likes to sleep on my bed.

- Word2vec will adjust the vector of a word to be similar to the vectors of its context words
- Words with similar contexts thus end up with similar vectors

Different embeddings are based on different priors

Latent semantic analysis

“Words occur in same documents should be similar”

Word2vec

“Words occur in similar contexts should be similar”

Neural Network Language Modeling

“Word vectors should assign high probability to plausible sentences”

Collabert et al., 2008 & 2011

“Word vectors should facilitate downstream classification tasks”

Faruqui et al., 2015

“Words should follow linguistic constraints from semantic lexicons”

Probabilistic Language Modeling

- Goal: assign a probability to a sentence
  - Machine Translation:
    - P(**large** winds tonight) < P(**strong** winds tonight)
  - Spell Correction
    - The office is about fifteen **minuets** from my house
      - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - Speech Recognition
    - P(I saw a van) >> P(eyes awe of an)
  - +Summarization, question answering, etc.

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or a sequence of words:

$$P(w_1^m) = P(w_1, w_2, \ldots, w_m)$$

- How to compute the joint probability?

$$P(a, dog, is, running, in, a, room)$$

- Chain rule:

$$P(w_1, w_2, \ldots, w_m) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \ldots P(w_m \mid w_1, \ldots w_{m-1})$$

$$P(a, dog, is, running) =$$

$$P(a)P(dog \mid a)P(is \mid a, dog)P(running \mid a, dog, is)$$

Probabilistic Language Modeling

$$P(w_1, w_2, \ldots, w_m) = \prod_t P(w_t \mid w_1, \ldots w_{t-1})$$

- Key: $P(w_t \mid w_1, \ldots w_{t-1})$
- Just count? Exponential number of entries and sparsity.
- Markov assumption:

$$P(w_t \mid w_1, \ldots w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots w_{t-1})$$

Probabilistic Language Modeling

- N-gram (bigram)

$$P(running \mid a, dog, is) \approx P(running \mid is) = \frac{count(is, running)}{count(is)}$$

- What's the problem?
  - Small context window (typically bigram or trigram)
  - Not utilizing word similarity
    - Seeing "A dog is running in a room" should increase probability of
    - "The dog is walking in a room" and
    - "A cat is running in the room" and
    - "Some cats are running in the room"

- Solution: Neural Network Language Modeling!

Neural Network Language Model

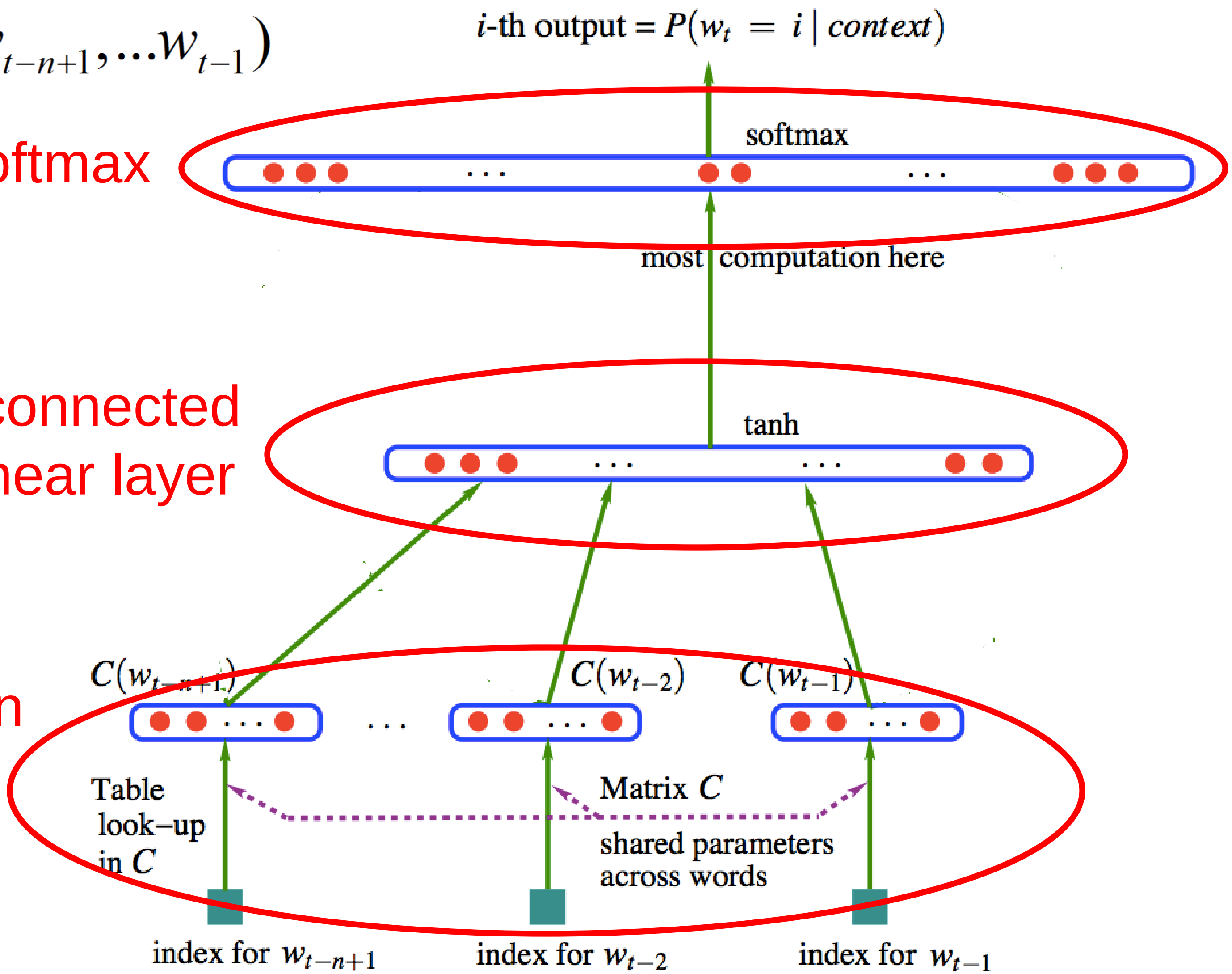
A Neural Probabilistic Language Model. Bengio et al. JMLR 2003.

Learn $P(w_t \mid w_{t-n+1}, \ldots w_{t-1})$

Softmax

Fully connected non-linear layer

Projection



The Lookup Table

- Each word in vocabulary maps to a vector in $\mathbb{R}^d$
- LookupTable: input of the i^th word is

$$x = (0, 0, \ldots, 1, 0, \ldots, 0) \quad 1 \text{ at position } i$$

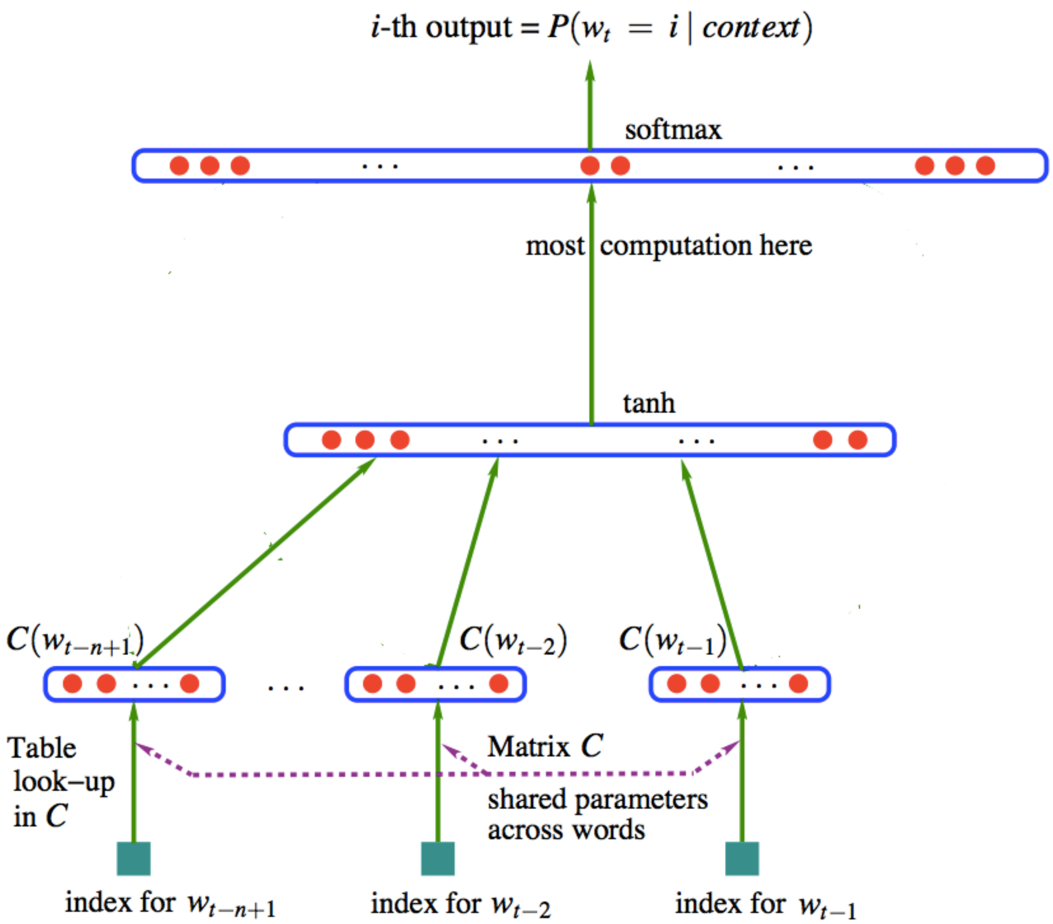
In the original space words are orthogonal.

cat = (0,0,0,0,0,0,0,0,0,1,0,0,0,0, …)
dog = (0,0,1,0,0,0,0,0,0,0,0,0,0,0, …)

To get the $\mathbb{R}^d$ embedding vector for the word we multiply *Cx* where *C* is a *d x N* matrix with *N* words in the vocabulary

C contains the word vectors!

Neural Network Language Model



$$P(w_t = i) = \frac{\exp(y_i)}{\sum_{j=1}^{D} \exp(y_j)}$$

softmax

$$y = Uz + b_2$$

output

$$z = \tanh(Hx + b_1)$$

non-linearity

$$x = (Cw_{t-n+1}, Cw_{t-n+2}, \ldots, Cw_{t-1})^T$$

projection

$$w_{t-n+1}, w_{t-n+2}, \ldots, w_{t-1}$$

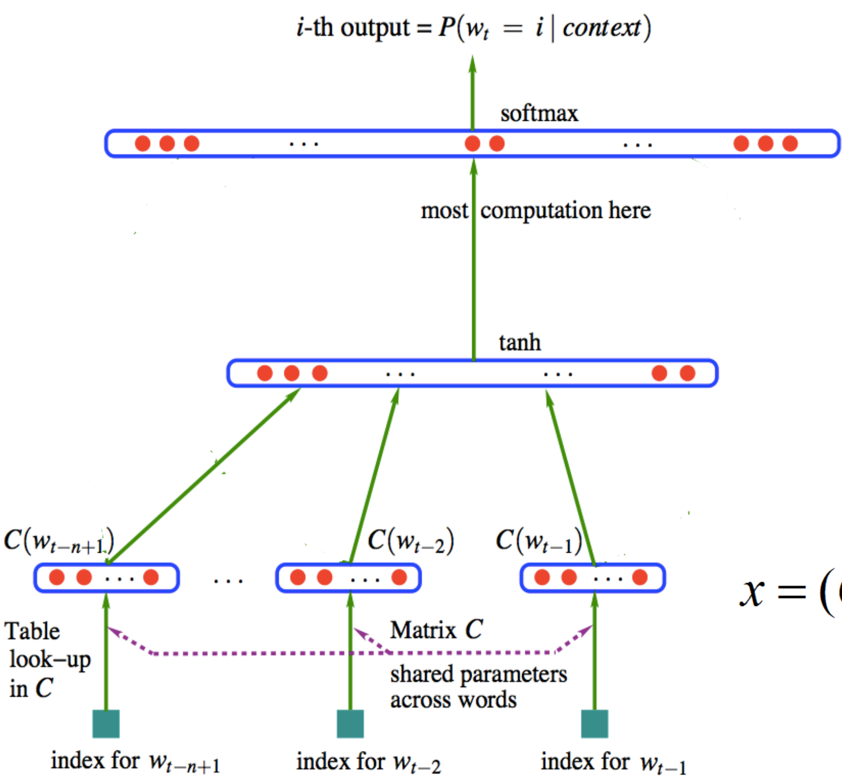
d : word vector dimensionality

n: window size

D: vocabulary size

h: # of hidden units

Dimensionality of each layer?



$$P(w_t = i) = \frac{\exp(y_i)}{\sum_{j=1}^{D} \exp(y_j)}$$

softmax

$$y = Uz + b_2 \qquad \boxed{D}$$

output

$$z = \tanh(Hx + b_1) \qquad \boxed{h}$$

non-linearity

$$x = (Cw_{t-n+1}, Cw_{t-n+2}, \ldots, Cw_{t-1})^T \qquad \boxed{n * d}$$

projection

$$w_{t-n+1}, w_{t-n+2}, \ldots, w_{t-1}$$

d : word vector dimensionality

n: window size

D: vocabulary size

h: # of hidden units

# of parameters in each layer?

$$P(w_t = i) = \frac{\exp(y_i)}{\sum_{j=1}^{D} \exp(y_j)}$$

softmax

$$y = Uz + b_2$$ $\boxed{h * D + D}$

output

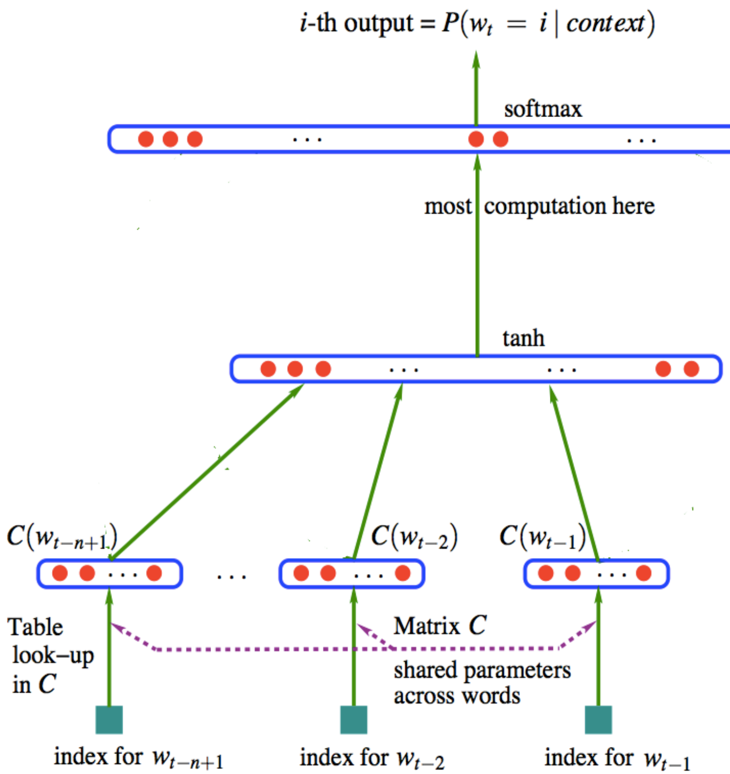
$$z = \tanh(Hx + b_1)$$ $\boxed{n * d * h + h}$

non-linearity

$$x = (Cw_{t-n+1}, Cw_{t-n+2}, \ldots, Cw_{t-1})^T$$ $\boxed{n * d}$

projection

$$w_{t-n+1}, w_{t-n+2}, \ldots, w_{t-1}$$



Training

- All free parameters

$$\boldsymbol{\theta} = (C, H, U, b_1, b_2)$$

- Backpropagation + Stochastic Gradient Ascent:

Costly!

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \varepsilon \frac{\partial \log P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})}{\partial \boldsymbol{\theta}}$$

Speed up training

- Most computations are at the output layer
  - In order to compute the normalization term of softmax, we have to compute the y_i for every word!
  - Cost (almost) linear to vocabulary size.
  - Same problem in Skip-gram

- Solutions: Approximate the normalized probability
  - Negative sampling
  - Noise contrastive estimation
  - Hierarchical softmax
  - …

Speed up training

- Most computations are at the output layer
  - In order to compute the normalization term of softmax, we have to compute the y_i for every word!
  - Cost (almost) linear to vocabulary size.
  - Same problem in Skip-gram

- Solutions: Approximate the normalized probability
  - **Negative sampling**
  - Noise contrastive estimation
  - Hierarchical softmax
  - …

Refresher: Skip-gram

- Given the central word, predict surrounding words in a window of length c

- Objective function:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{t+j} | w_t)$$

- Softmax:

$$p(O \mid I) = \frac{\exp(v_O'^T v_I)}{\sum_{w \in V} \exp(v_w'^T v_I)}$$

$$\frac{\partial \log p(O \mid I)}{\partial v_I} = v'_O - \sum_w p(w \mid I) v'_w$$

Negative sampling

- I: central word. O: a context word
- Original: Maximize $p(O \mid I, \theta)$
- We will derive an alternative which is less costly to compute
- Does pair (I,O) really come from the training data?

$$\theta = \arg\max_{\theta} p(D=1 \mid I, O, \theta)$$

$$\text{where } p(D=1 \mid I, O, \theta) = \sigma(v_I^T v_O^{'}) = \frac{1}{1+e^{-v_I^T v_O^{'}}}$$

- Trivial solution: same (long enough) vector for all words
- Contrast with negative words!

Negative sampling

- Solution: randomly sample k negative words w_i from a noise distribution, assume (I,w_i) are incorrect pairs

$$\text{maximize } p(D=1 \mid I,O,\theta) \bullet \prod_{i=1}^{k} p(D=0 \mid I,w_i,\theta)$$

$$\text{or } \log\sigma(v_I^T v_O^{'}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log\sigma(-v_I^T v_{w_i}^{'})]$$

$$\text{where } P_n(w) = \frac{U(w)^{3/4}}{Z}, U(w) \text{ the unigram distribution}$$

Different embeddings are based on different priors

Latent semantic analysis

"Words occur in same documents should be similar"

Word2vec

"Words occur in similar contexts should be similar"

Neural Network Language Modeling

"Word vectors should give plausible sentences high probability"

Collabert et al., 2008 & 2011

"Word vectors should facilitate downstream classification tasks"

Faruqui et al., 2015

"Words should follow linguistic constraints from semantic lexicons"

What to get from this work

- How to supervise the learning of word embedding using external classification tasks

- How to do semi-supervised learning of word embedding

- How to apply word vectors and neural networks in other traditional NLP tasks

Embedding for other NLP tasks (Collobert et al., 2008&11)

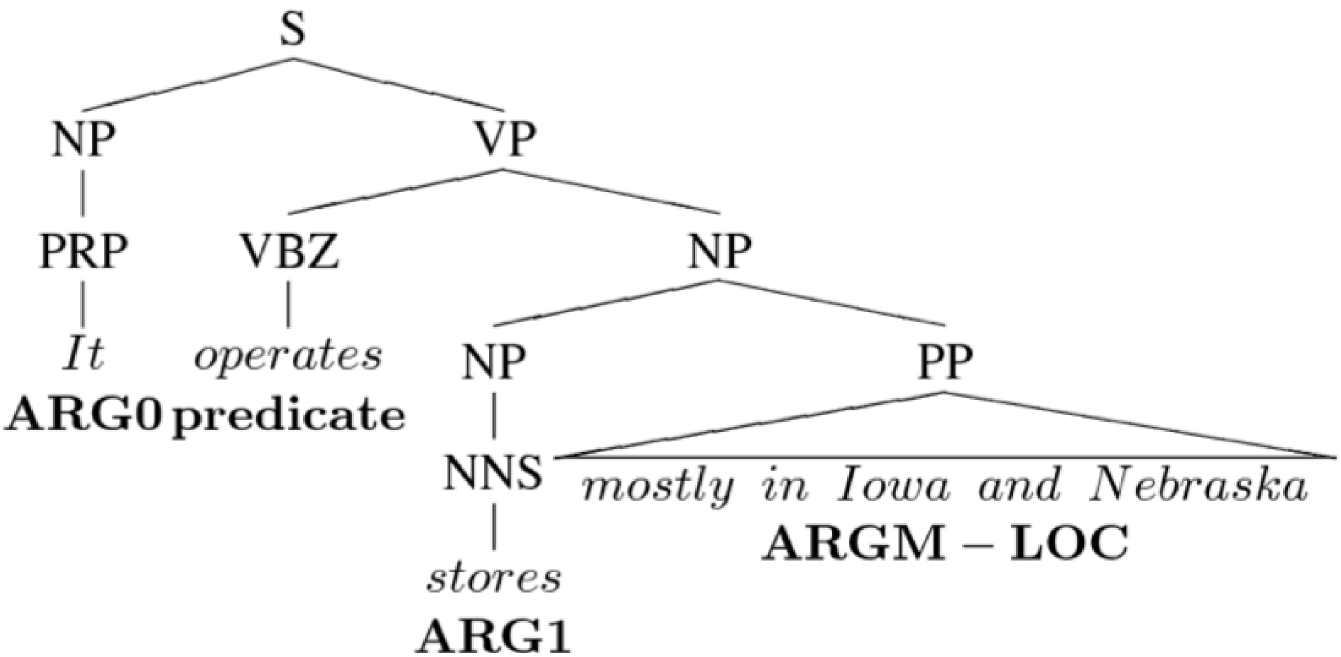
- Part-Of-Speech Tagging (POS): syntactic roles (noun, adverb...)
- Chunking: syntactic constituents (noun phrase, verb phrase...)
- Name Entity Recognition (NER): person/company/location...
- Semantic Role Labeling (SRL):

  [John]$_{ARG0}$ [ate]$_{REL}$ [the apple]$_{ARG1}$ [in the garden]$_{ARGM-LOC}$

Complex Systems

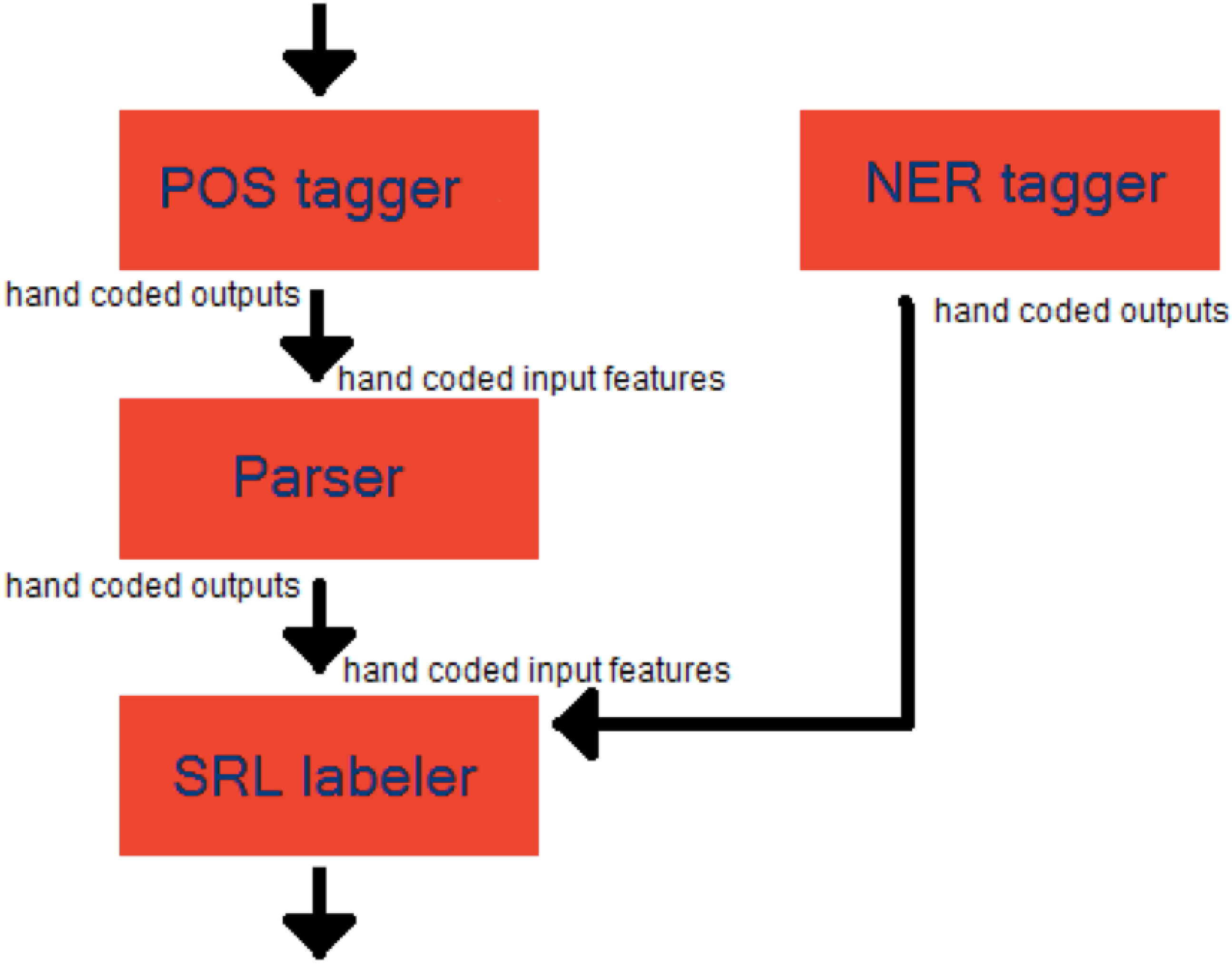
- Two extreme choices to get a complex system
  - ⋆ Large Scale Engineering: design a lot of complex features, use a fast existing linear machine learning algorithm
  - ⋆ Large Scale Machine Learning: use simple features, design a complex model which will implicitly learn the right features

The Large Scale Feature Engineering Way



- Extract **hand-made features** e.g. from the parse tree
- Disjoint: all tasks trained separately, Cascade features
- Feed these features to a shallow classifier like SVM

The sub-optimal cascade



NLP: Large scale machine learning

Goals

- Task-specific engineering limits NLP scope
- Can we find unified hidden representations?
- Can we build unified NLP architecture?

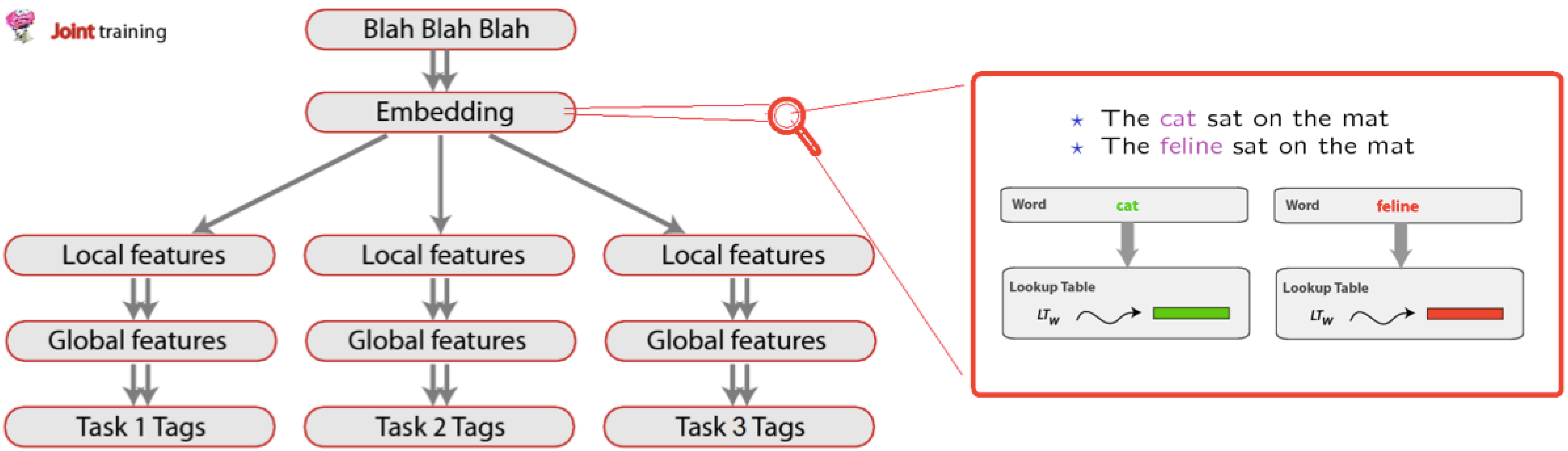
Means

- Start from scratch: forget (most of) NLP knowledge
- Compare against classical NLP benchmarks
- Our dogma: avoid task-specific engineering

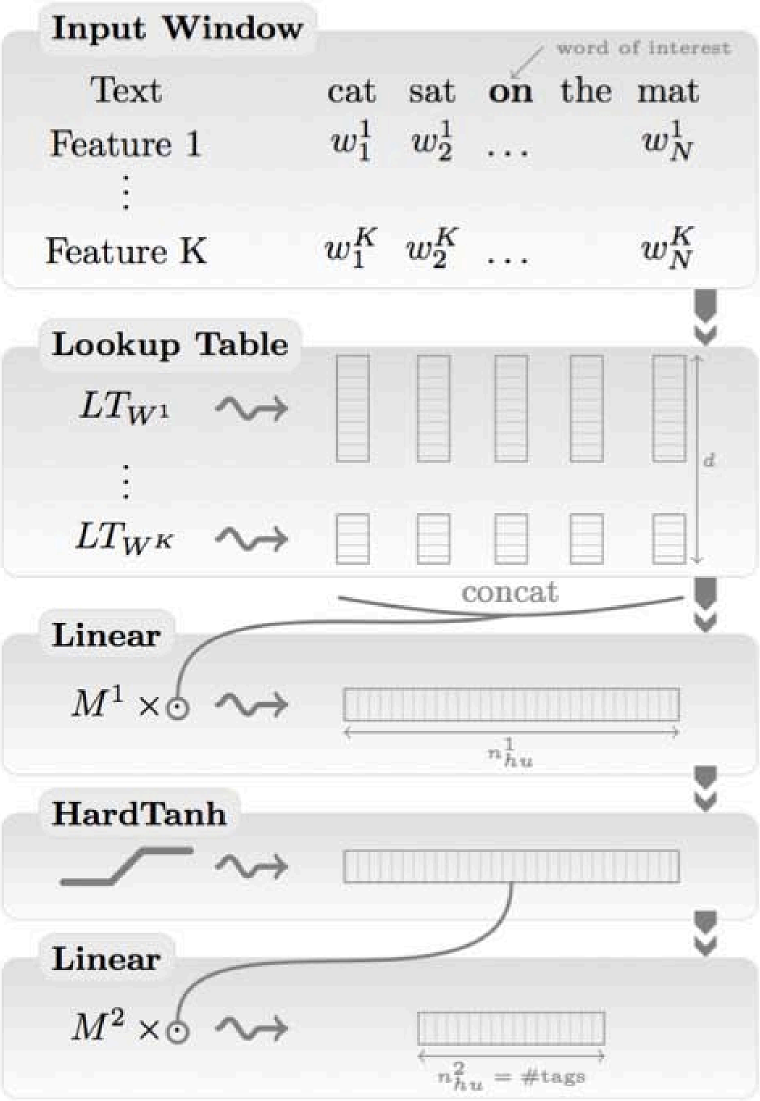
The big picture

A unified architecture for all NLP (labeling) tasks:

Sentence:	*Felix*	*sat*	*on*	*the*	*mat*	.
POS:	NNP	VBD	IN	DT	NN	.
CHUNK:	NP	VP	PP	NP	NP-I	.
NER:	PER	-	-	-	-	-
SRL:	ARG1	REL	ARG2	ARG2-I	ARG2-I	-



Window approach

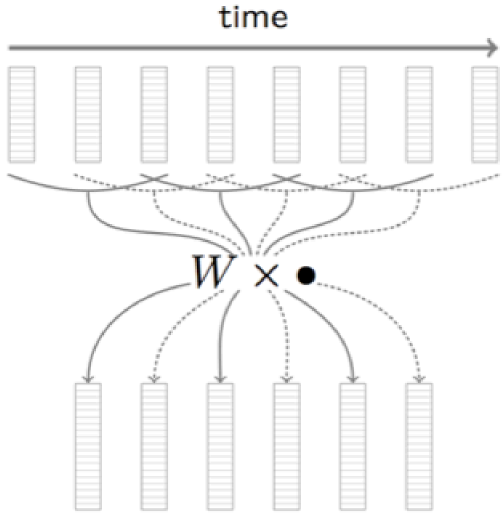


- Tags one word at the time
- Feed a fixed-size window of text around each word to tag
- Works fine for most tasks
- How do deal with long-range dependencies?

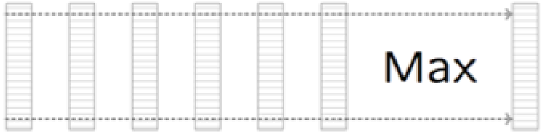
  E.g. in SRL, the verb of interest might be outside the window!

Sentence approach

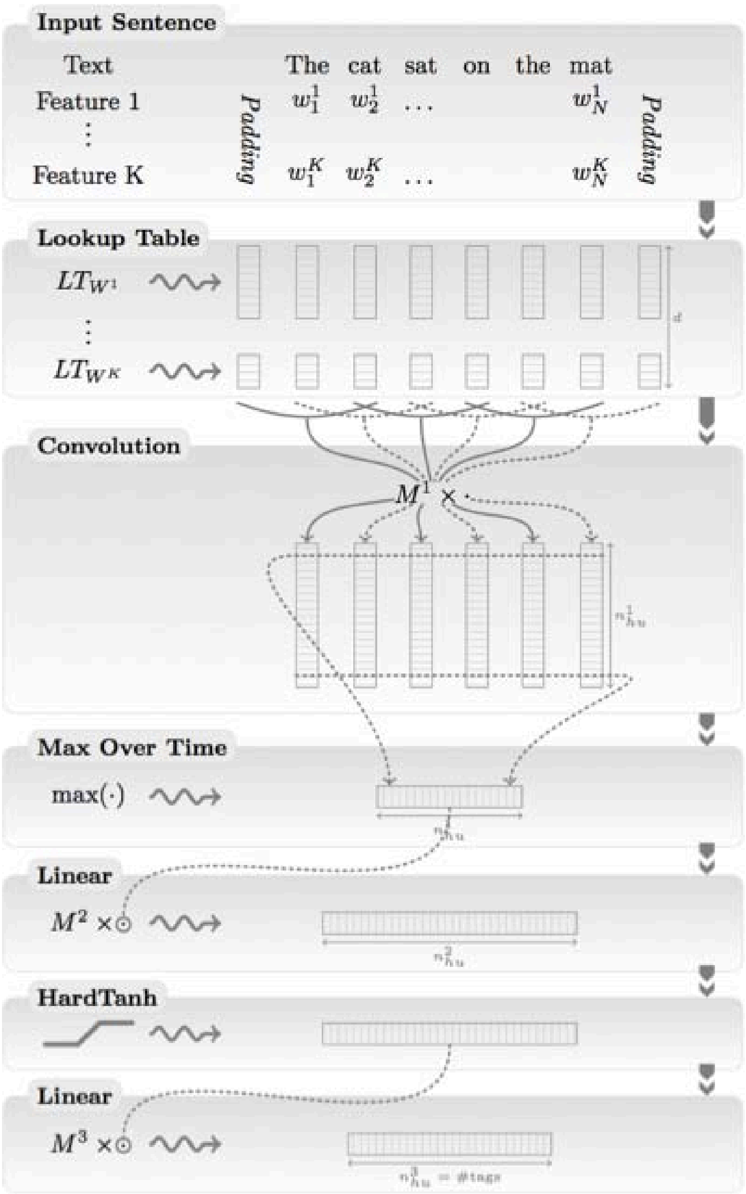
- Feed the whole sentence to the network
- Tag one word at the time: add extra position features
- Convolutions to handle variable-length inputs



- Produces local features with higher level of abstraction
- Max over time to capture most relevant features



Outputs a fixed-sized feature vector



Input Sentence
Text
The cat sat on the mat
Feature 1
w_1^1 w_2^1 ... w_N^1
Feature K
w_1^K w_2^K ... w_N^K
Padding
Padding
Lookup Table
LT_{W^1}
LT_{W^K}
Convolution
$M^1 \times \cdot$
Max Over Time
max(·)
Linear
$M^2 \times \odot$
HardTanh
Linear
$M^3 \times \odot$

Supervised benchmark results

- Window for POS tagging, Chunking and NER
- Convolution for SRL

Approach	POS (PWA)	Chunking (F1)	NER (F1)	SRL (F1)
Benchmark Systems	**97.24**	**94.29**	**89.31**	**77.92**
NN+WTL	96.31	89.13	79.53	55.40
NN+STL	96.37	90.33	81.47	*70.99*

- Can we do better?

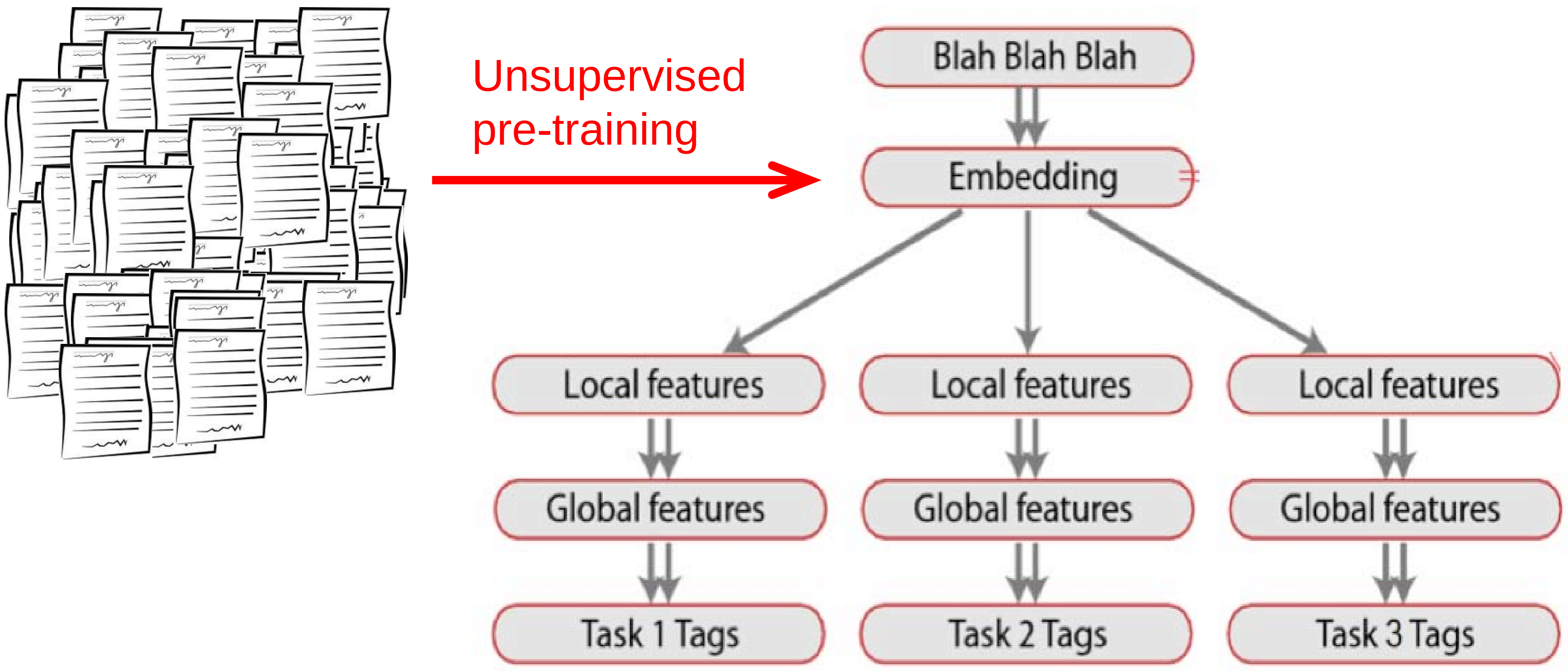
Supervised word embeddings

- Sentences with similar words should be tagged in the same way:
  - ⋆ The cat sat on the mat
  - ⋆ The feline sat on the mat

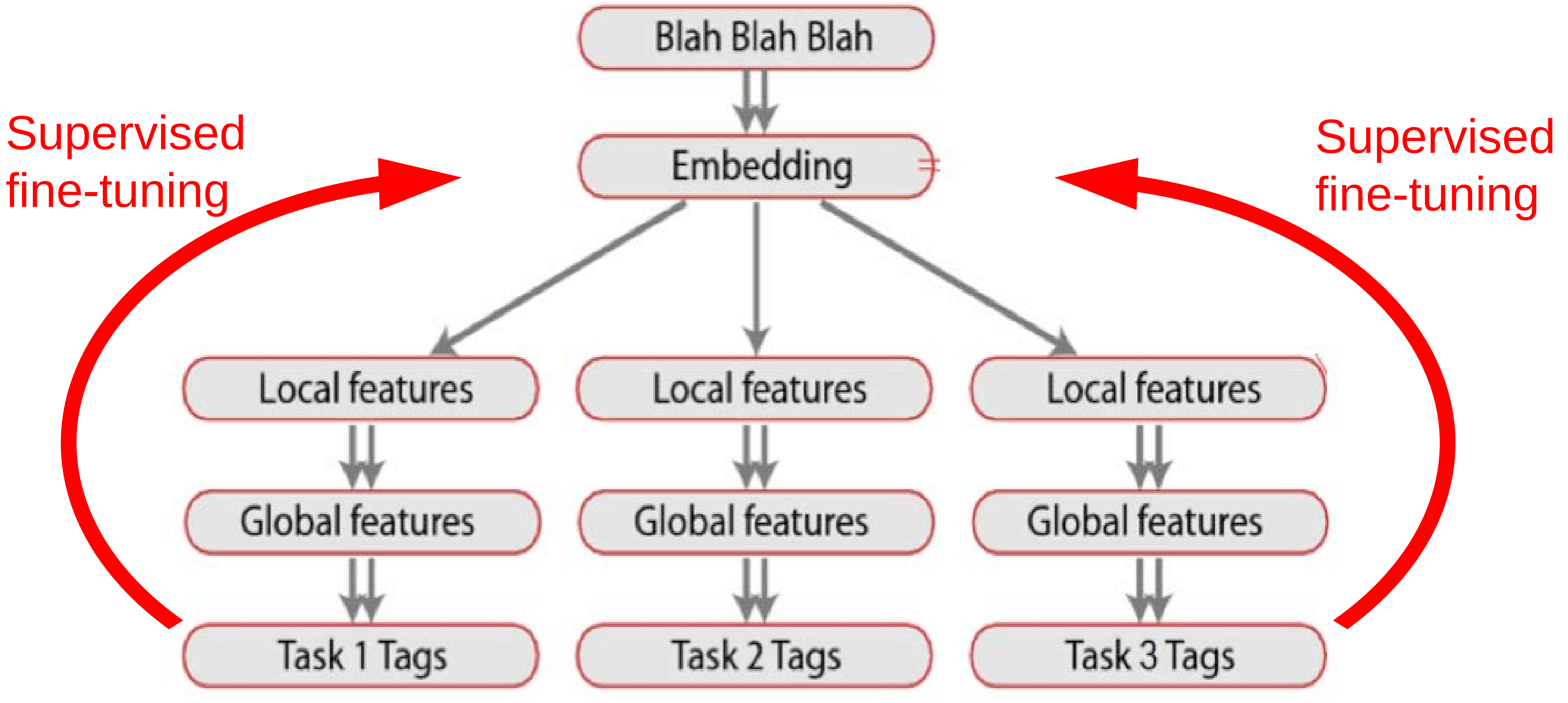
france	jesus	xbox	reddish	scratched	megabits
454	1973	6909	11724	29869	87025
persuade	thickets	decadent	widescreen	odd	ppa
faw	savary	divo	antica	anchieta	uddin
blackstock	sympathetic	verus	shabby	emigration	biologically
giorgi	jfk	oxide	awe	marking	kayak
shaheed	khwarazm	urbina	thud	heuer	mclarens
rumelia	stationery	epos	occupant	sambhaji	gladwin
planum	ilias	eglinton	revised	worshippers	centrally
goa'uld	gsNUMBER	edging	leavened	ritsuko	indonesia
collation	operator	frg	pandionidae	lifeless	moneo
bacha	w.j.	namsos	shirt	mahan	nilgiris

- About 1M of words in WSJ
- 15% of most frequent words in the dictionary are seen 90% of the time
- Cannot expect words to be trained properly!

Semi-supervised learning with unlabeled text



Semi-supervised learning with unlabeled text



Semi-supervised word embeddings

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869
SPAIN	CHRIST	PLAYSTATION	YELLOWISH	SMASHED
ITALY	GOD	DREAMCAST	GREENISH	RIPPED
RUSSIA	RESURRECTION	PSNUMBER	BROWNISH	BRUSHED
POLAND	PRAYER	SNES	BLUISH	HURLED
ENGLAND	YAHWEH	WII	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

(Even fairly rare words are embedded well.)

Semi-supervised benchmark results

Algorithm	POS (PWA)	CHUNK (F1)	NER (F1)	SRL (F1)
Baselines	97.24 [Toutanova '03]	94.29 [Sha '03]	89.31 [Ando '05]	77.92 [Koomen '05]
NN + WTL	96.31	89.13	79.53	55.40
NN + STL	96.37	90.33	81.47	70.99
NN + LM + STL	97.22	94.10	88.67	74.15
NN + . . . + tricks	97.29 [+suffix]	94.32 [+POS]	89.95 [+gazetteer]	76.03 [+Parse Trees]

Different embeddings are based on different priors

Latent semantic analysis

"Words occur in same documents should be similar"

Word2vec

"Words occur in similar contexts should be similar"

Neural Network Language Modeling

"Word vectors should give plausible sentences high probability"

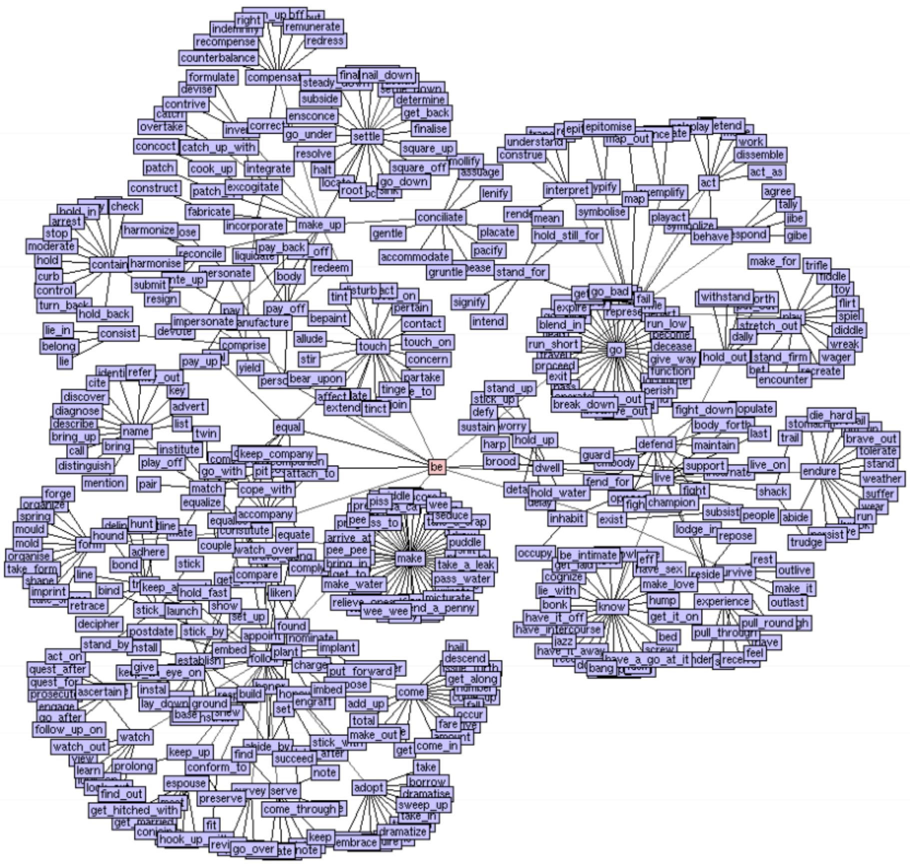
Collabert et al., 2008 & 2011

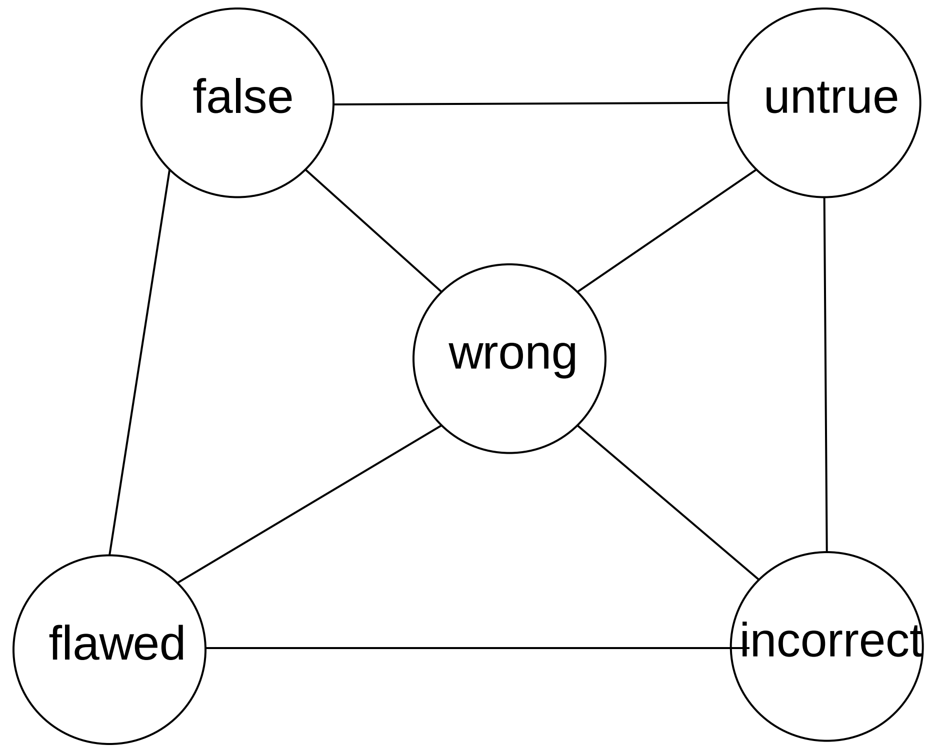
"Word vectors should facilitate downstream classification tasks"

Faruqui et al., 2015

"Words should follow linguistic constraints from semantic lexicons"

Semantic lexicon: WordNet

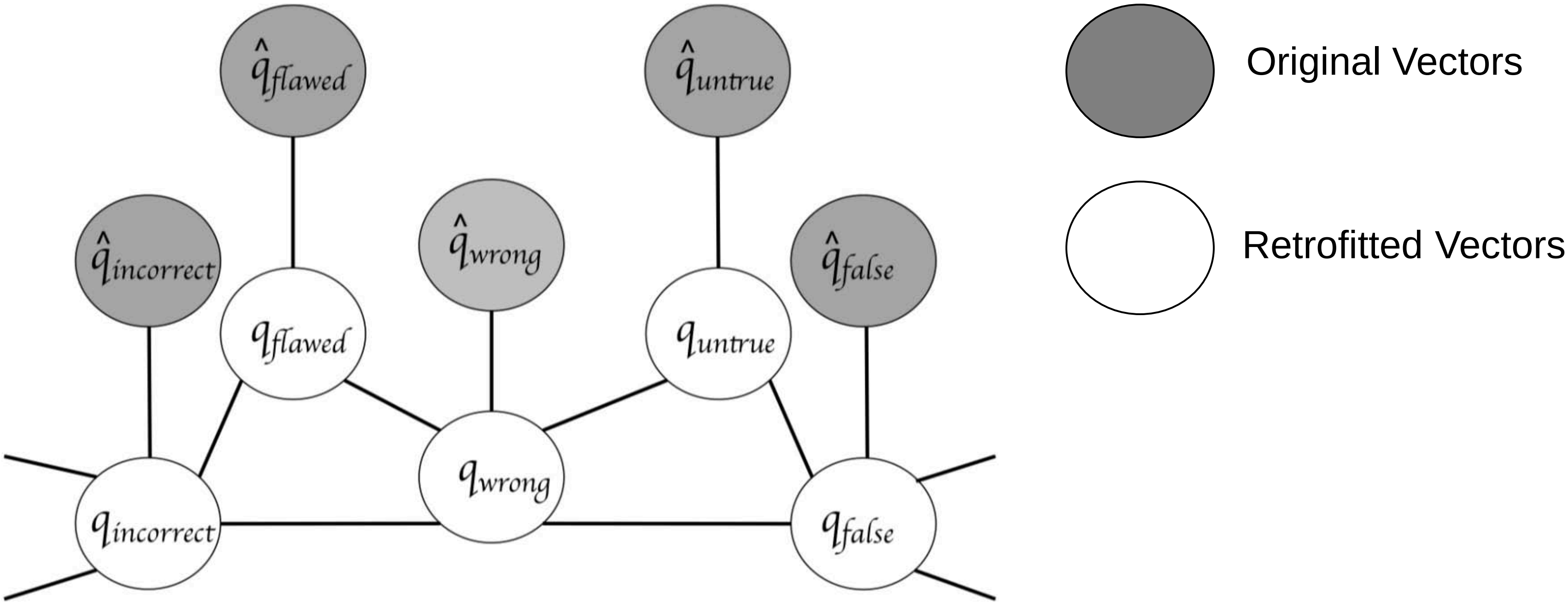




Retrofitting word vectors to semantic lexicons (NAACL'15)

- Incorporates information from lexicons in word vectors
- Post-processing approach
- Applicable to **any** word embedding method
- Applicable to **any** lexicon

Retrofitting



$$\Psi(Q) = \sum_{i=1}^{n} \left[\alpha_i \|q_i - \hat{q}_i\|^2 + \sum_{(i,j)\in E} \beta_{ij} \|q_i - q_j\|^2 \right]$$

Semantic lexicons used in this work

- PPDB: Lexical paraphrases obtained from parallel texts
- WordNet: synonyms, hypernyms and hyponyms
- FrameNet: Cause_change_of_position -> push=raise=growth

Lexicon	Words	Edges
PPDB	102,902	374,555
WordNet_{syn}	148,730	304,856
WordNet_{all}	148,730	934,705
FrameNet	10,822	417,456

Table 1. Approximate size of the graphs obtained from different lexicons

Experiment results

Word Similarity

Synonym Selection

Syntactic Analysis

Sentiment Analysis

Original Embedding

Semantic Lexicons

Lexicon	MEN-3k	RG-65	WS-353	TOEFL	SYN-REL	SA
Glove	73.7	76.7	60.5	89.7	67.0	79.6
+PPDB	1.4	2.9	–1.2	**5.1**	–0.4	**1.6**
+WN$_{syn}$	0.0	2.7	0.5	**5.1**	–12.4	0.7
+WN$_{all}$	**2.2**	**7.5**	**0.7**	2.6	–8.4	0.5
+FN	–3.6	–1.0	–5.3	2.6	–7.0	0.0
SG	67.8	72.8	65.6	85.3	73.9	81.2
+PPDB	**5.4**	3.5	**4.4**	**10.7**	–2.3	**0.9**
+WN$_{syn}$	0.7	3.9	0.0	9.3	–13.6	0.7
+WN$_{all}$	2.5	**5.0**	1.9	9.3	–10.7	–0.3
+FN	–3.2	2.6	–4.9	1.3	–7.3	0.5

In this lecture…

- More types of supervision used in training word embedding
  - Language modeling
  - NLP labeling tasks
  - Semantic lexicons

- Ways to speed up
  - E.g., negative sampling
  - Necessary for training on huge text corpora
  - Scale up from hundreds of millions to hundreds of billions

- How word embeddings help other NLP tasks