CS290D – 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

Sources of slides

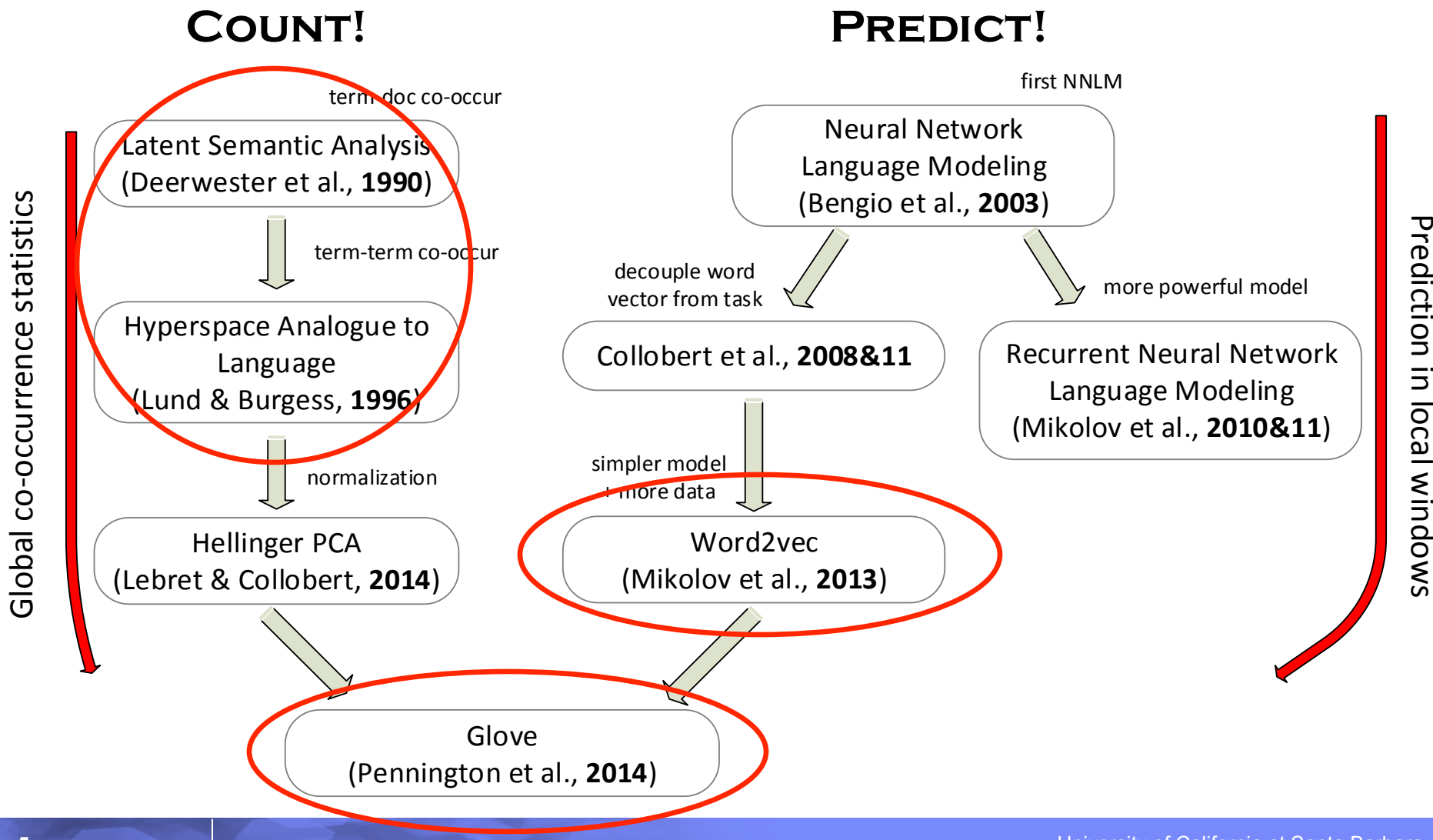
- Dan Jurafsky's lecture about language modeling
- Tutorial at EMNLP'14: Embedding Methods for NLP
- Tutorial at ACL'14: New Directions in Vector Space Models of Meaning

How to let a computer understand meaning?

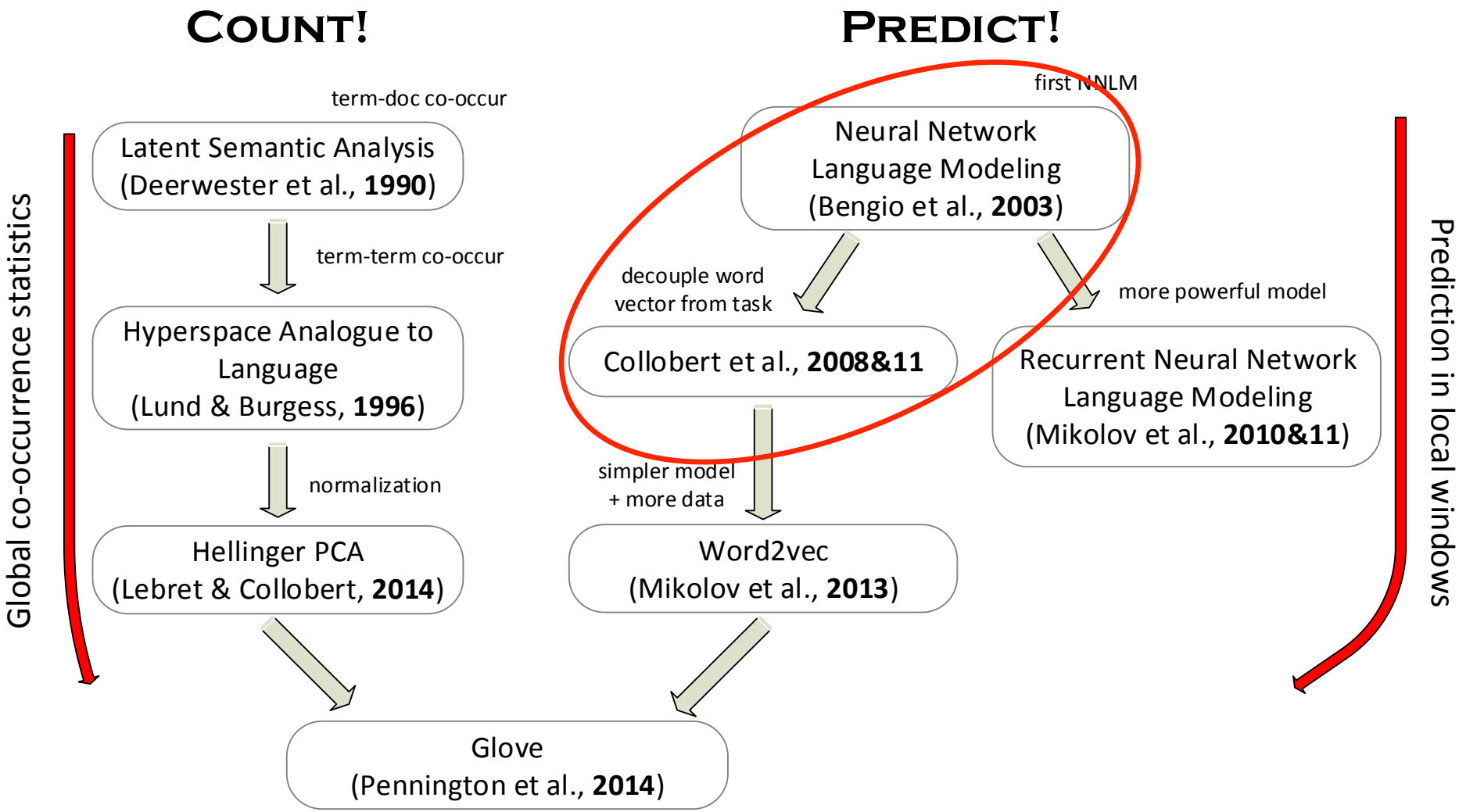
A cat sits on a mat.

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

History of word embeddings



History of word embeddings



In this lecture…

- More advanced models for learning word embeddings
  - Neural Language Model (Bengio et al., 2003)
  - Collabert et al., 2008 & 2011
- 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

Probabilistic Language Modeling

- Goal: assign a probability to a sentence
  - Machine Translation:
    - P(**high** winds tonight) > P(**large** 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:

$$P(w_1^m) = P(w_1, w_2, ..., w_m)$$

- How to compute the joint probability?

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

- Chain rule:

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

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

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

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!
- 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

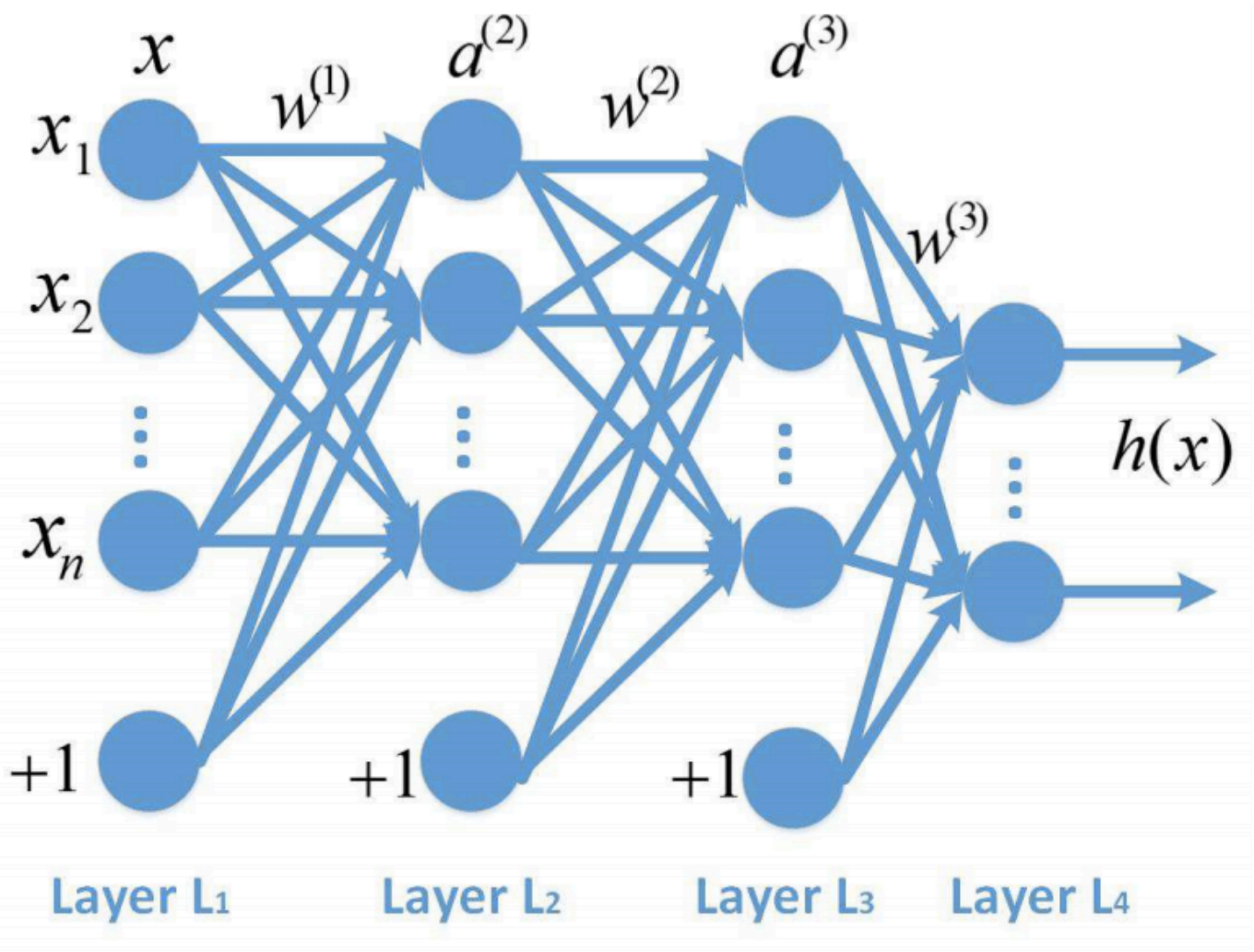
- N-gram (bigram)

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

- What's the problem?
  - Small context window (typically 1~2)
  - Not utilizing word similarity
    - Seeing "A dog is running in a room" should increase probability of
    - "The dog is walking in the room" and
    - "A cat is running in the room" and
    - "Some cats are running in the room"

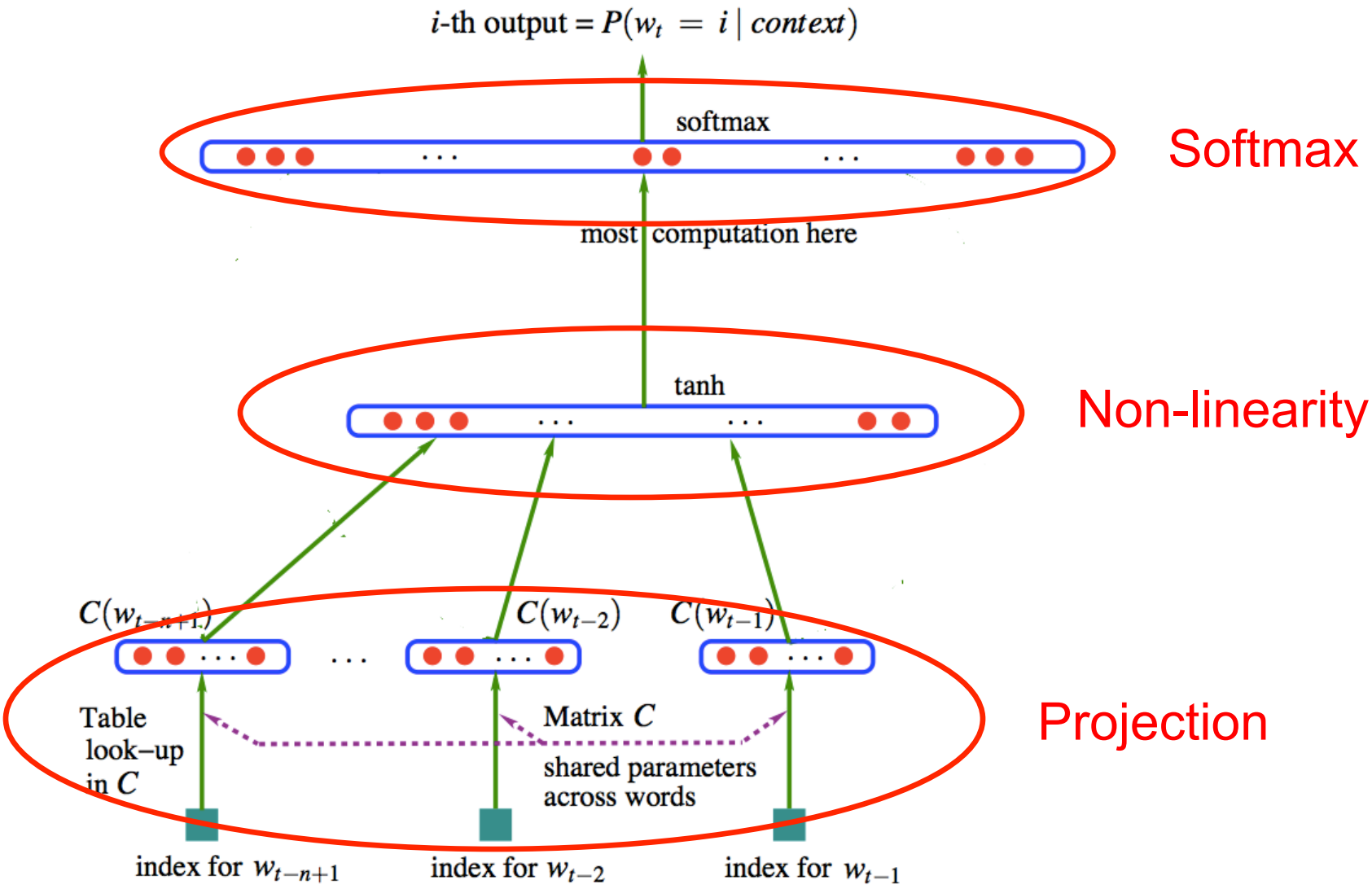
- Solution: Neural language Model!

Refresher: Multi-layer Neural Network



Neural Language Model

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



The Lookup Table

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

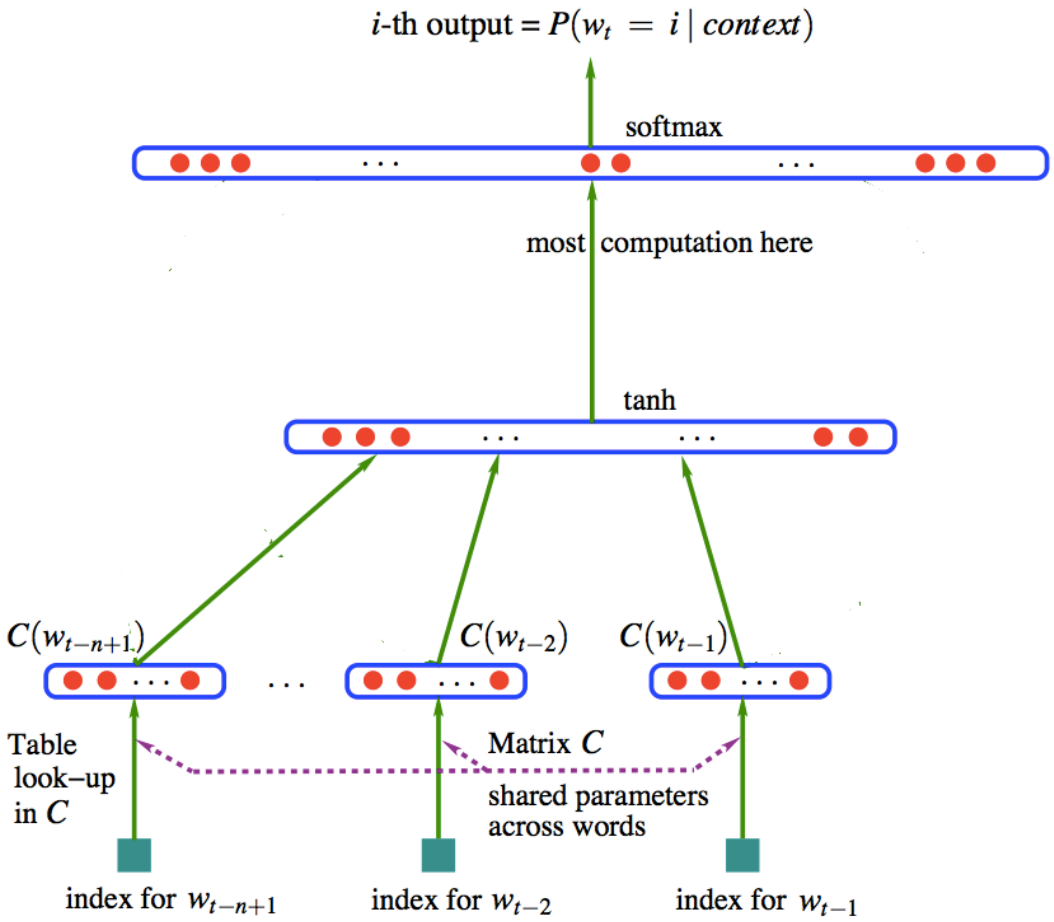
$$x = (0, 0, \ldots, 1, 0, \ldots, 0) \quad 1 \text{ at position } i$$

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

Neural 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

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

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 Descent:

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 linear to |V|.
  - 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 linear to |V|.
  - 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:

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

- Softmax:

$$p(w_O \mid w_I) = \frac{\exp(v'^{T}_{w_O} v_{w_I})}{\sum_{w=1}^{W} \exp(v'^{T}_{w} v_{w_I})}$$

Negative sampling

- w: central word. c: a context word
- Original: Maximize $p(w \mid c,\theta)$
- Or: Does pair (w,c) really come from the training data?

$$\theta = \arg\max_\theta p(D=1 \mid w,c,\theta)$$

$$\text{where } p(D=1 \mid w,c,\theta) = \sigma(v_w^T v_c^{'}) = \frac{1}{1+e^{-v_w^T v_c^{'}}}$$

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

Negative sampling

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

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

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

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

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

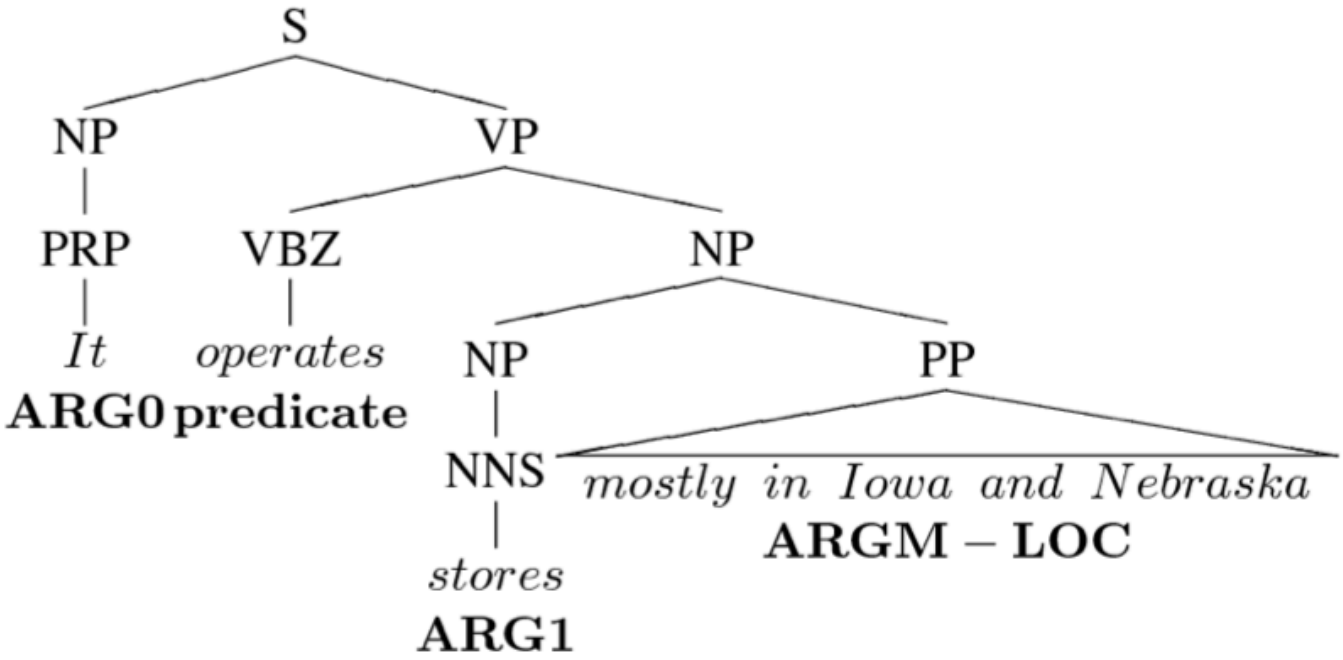
- Part-Of-Speech Tagging (POS): syntactic roles (noun, adverb...)
- Chunking: syntactic constituents (noun phrase, verb phrase...)
- Name Entity Recognition (NER): person/company/location...
- Semantic Role Labeling (SRL):

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

Complex Systems

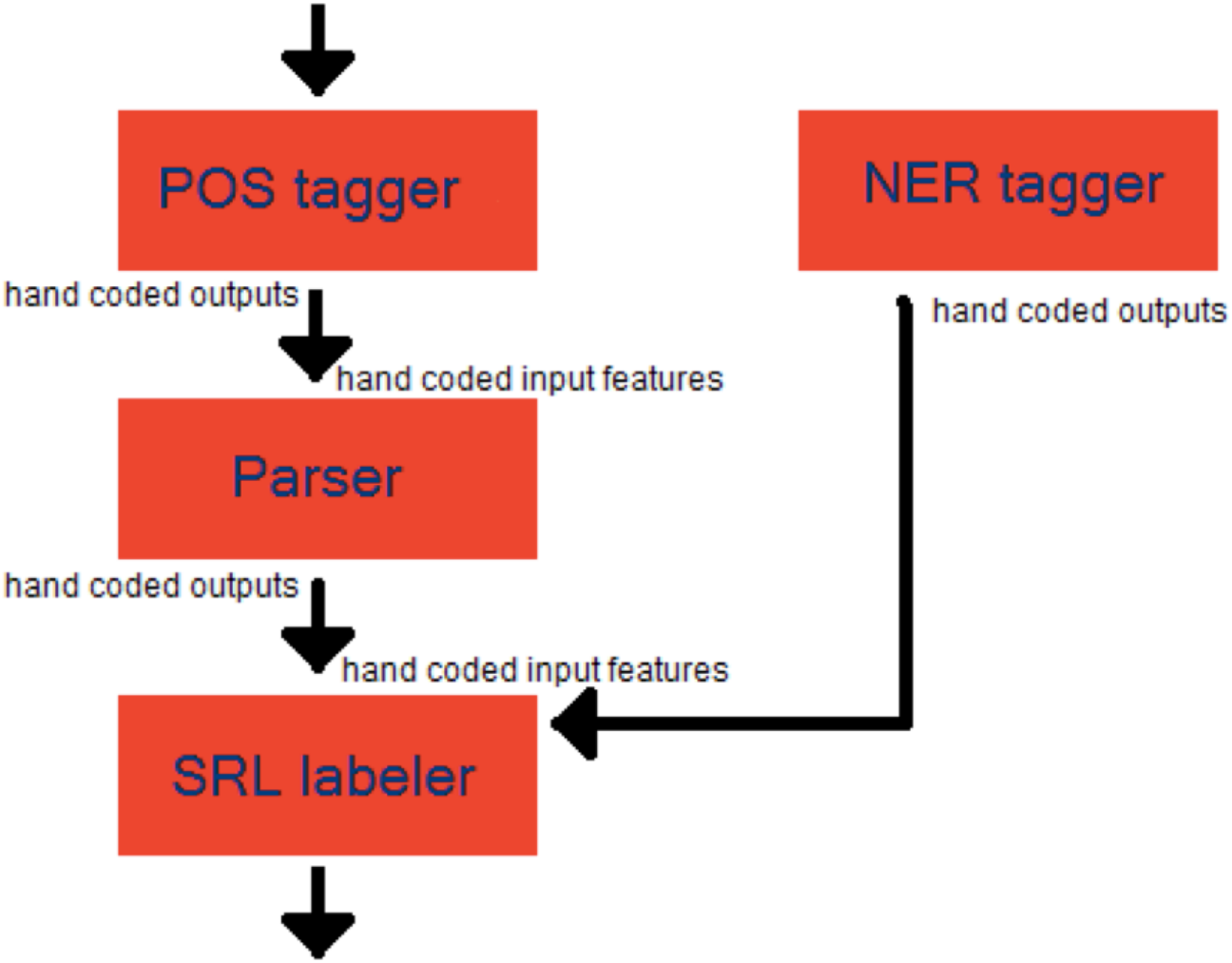
- Two extreme choices to get a complex system
  - ⋆ Large Scale Engineering: design a lot of complex features, use a fast existing linear machine learning algorithm
  - ⋆ Large Scale Machine Learning: use simple features, design a complex model which will implicitly learn the right features

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

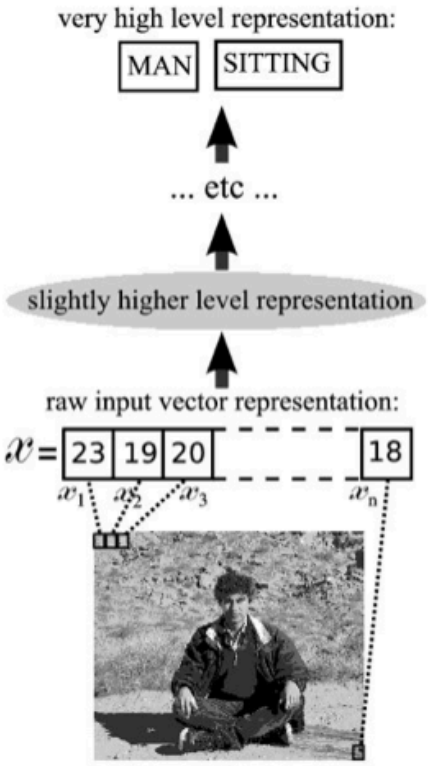
- Task-specific engineering limits NLP scope
- Can we find unified hidden representations?
- Can we build unified NLP architecture?

Means

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

The “deep learning” way

Neural nets attempt to propose a radically? different end-to-end approach:

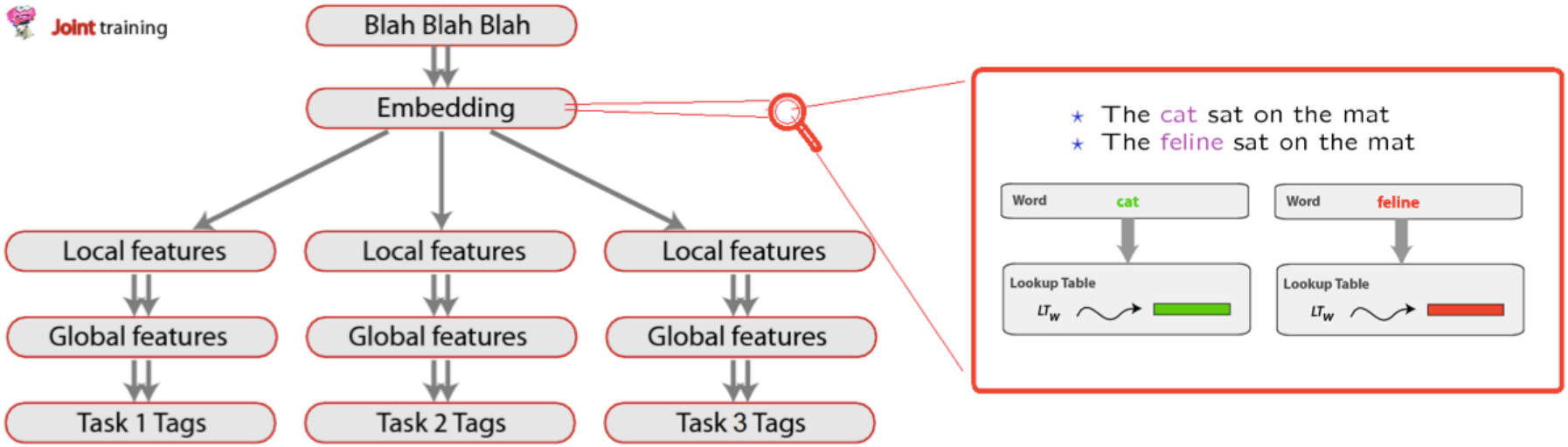


- Avoid building a parse tree. Humans don’t need this to talk.
- Try to avoid all hand-built features → monolithic systems.

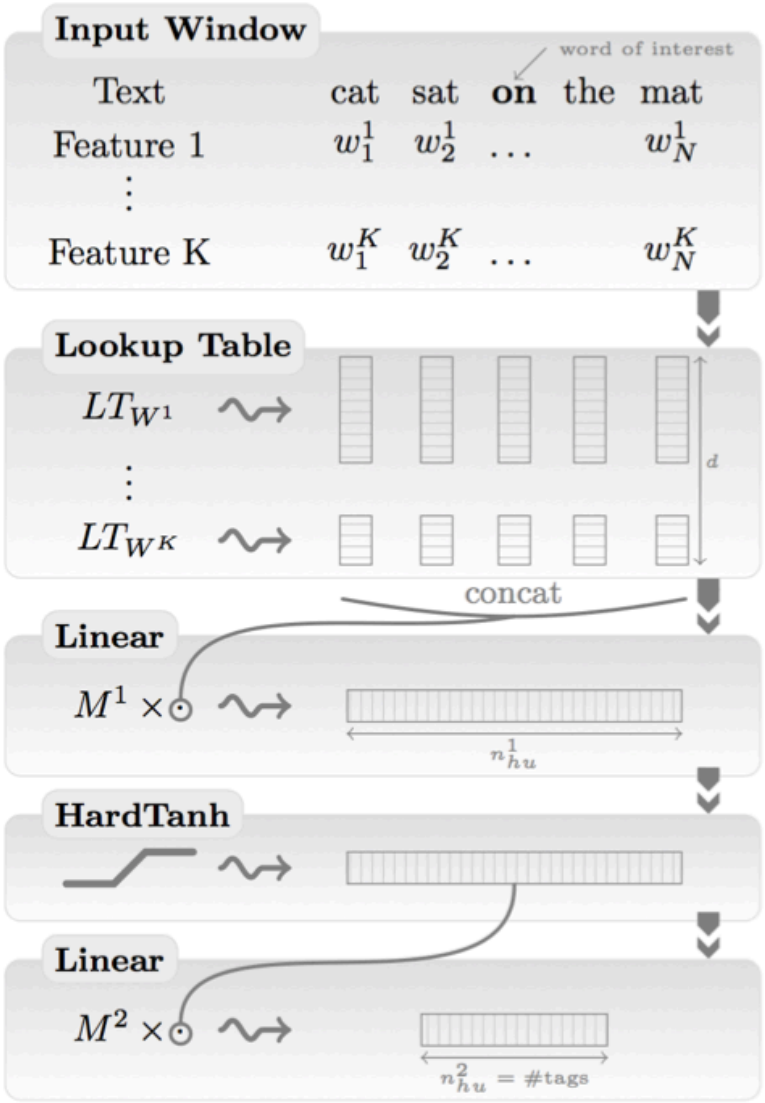
The big picture

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

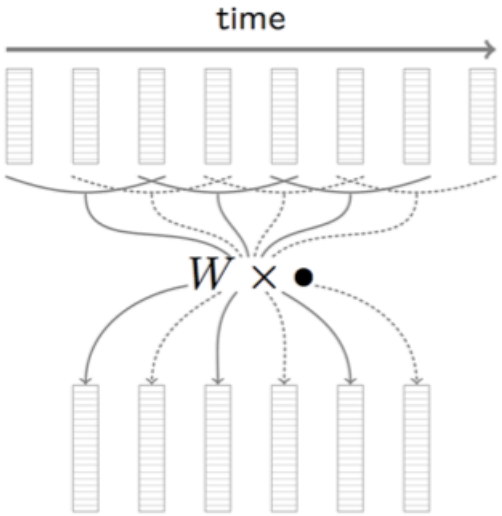


- Tags one word at the time
- Feed a fixed-size window of text around each word to tag
- Works fine for most tasks
- How do deal with long-range dependencies?

  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

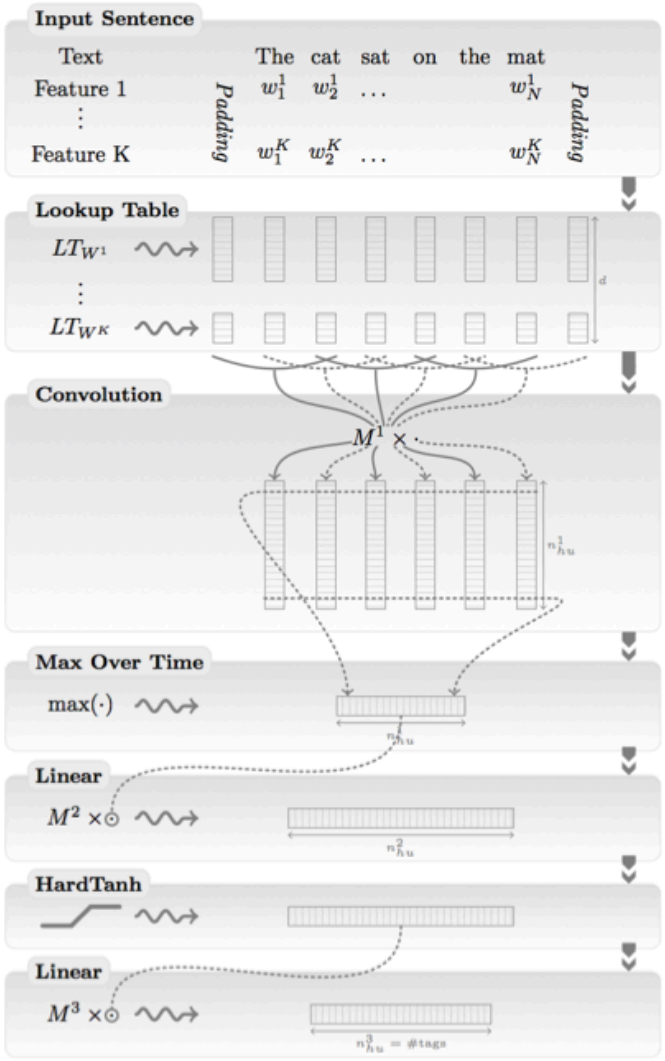


See (Bottou, 1989) or (LeCun, 1989).

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

Outputs a fixed-sized feature vector

Sentence approach



Supervised benchmark results

- Window (5) for POS, Chunking and NER
- Convolutional (3) 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*

- Are we training word embeddings right?

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

- Language Model: "*is a sentence actually english or not?*"
  Implicitly captures: * syntax * semantics
- Bengio & Ducharme (2001) Probability of next word given previous words. Overcomplicated – we do not need probabilities here
- English sentence windows: Wikipedia ($\sim 631M$ words)
  Non-english sentence windows: middle word randomly replaced

  the champion federer wins wimbledon
  vs. the champion saucepan wins wimbledon
- Multi-class margin cost:

$$\sum_{s \in \mathcal{S}} \sum_{w \in \mathcal{D}} \max(0, 1 - f(s, w_s^\star) + f(s, w))$$

$\mathcal{S}$: sentence windows $\mathcal{D}$: dictionary

$w_s^\star$: true middle word in s

$f(s, w)$: network score for sentence s and middle word w

Un-supervised word embeddings

Nearest neighbors in 100-dim. embedding space:

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]

In this lecture…

- More advanced models for learning word embeddings
  - Neural Language Model (Bengio et al., 2003)
  - Collabert et al., 2008 & 2011
- 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