Natural language processing and weak supervision

Léon Bottou

COS 424 – 4/27/2010

Natural language processing "from scratch"

– Natural language processing systems are heavily engineered.
– How much engineering can we avoid by using more data ?
– Work by Ronan Collobert, Jason Weston, and the NEC team.

Summary

– Natural language processing
– Embeddings and models
– Lots of unlabeled data
– Task dependent hacks

I. Natural language processing

The Goal

- We want to have a conversation with our computer

...still a long way before HAL 9000 ...

- Convert a piece of English into a computer-friendly data structure
- How to measure if the computer "understands" something?

Natural Language Processing Tasks

Intermediate steps to reach the goal?

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

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

NLP Benchmarks

- Datasets:
  - ⋆ POS, CHUNK, SRL: WSJ (≈ up to 1M labeled words)
  - ⋆ NER: Reuters (≈ 200K labeled words)

System	Accuracy
Shen, 2007	97.33%
Toutanova, 2003	**97.24%**
Gimenez, 2004	97.16%

(a) POS: As in (Toutanova, 2003)

System	F1
Shen, 2005	95.23%
Sha, 2003	**94.29%**
Kudoh, 2001	93.91%

(b) CHUNK: CoNLL 2000

System	F1
Ando, 2005	**89.31%**
Florian, 2003	88.76%
Kudoh, 2001	88.31%

(c) NER: CoNLL 2003

System	F1
Koomen, 2005	**77.92%**
Pradhan, 2005	77.30%
Haghighi, 2005	77.04%

(d) SRL: CoNLL 2005

- We chose as benchmark systems:
  - ⋆ Well-established systems
  - ⋆ Systems avoiding external labeled data
- Notes:
  - ⋆ Ando, 2005 uses external unlabeled data
  - ⋆ Koomen, 2005 uses 4 parse trees not provided by the challenge

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

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

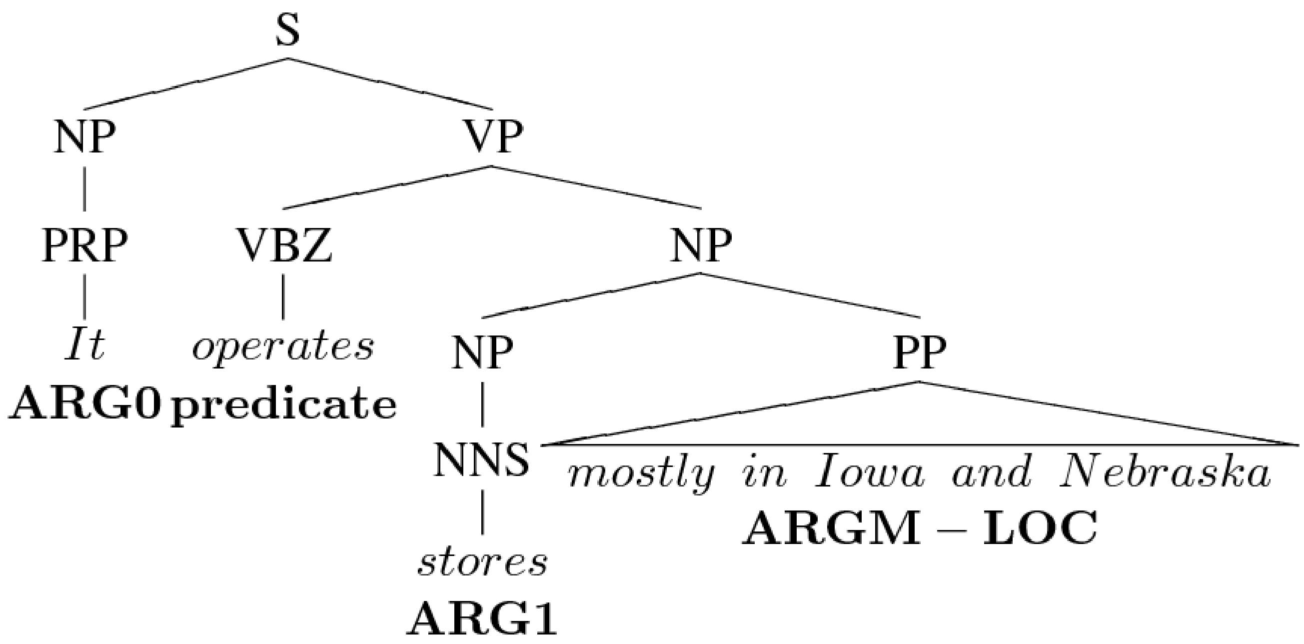
NLP: Large Scale Engineering (1/2)

- Choose some good hand-crafted features

Predicate and POS tag of predicate	Voice: active or passive (hand-built rules)
Phrase type: adverbial phrase, prepositional phrase, . . .	Governing category: Parent node's phrase type(s)
Head word and POS tag of the head word	Position: left or right of verb
Path: traversal from predicate to constituent	Predicted named entity class
Word-sense disambiguation of the verb	Verb clustering
Length of the target constituent (number of words)	NEG feature: whether the verb chunk has a "not"
Partial Path: lowest common ancestor in path	Head word replacement in prepositional phrases
First and last words and POS in constituents	Ordinal position from predicate + constituent type
Constituent tree distance	Temporal cue words (hand-built rules)
Dynamic class context: previous node labels	Constituent relative features: phrase type
Constituent relative features: head word	Constituent relative features: head word POS
Constituent relative features: siblings	Number of pirates existing in the world. . .

- Feed them to a simple classifier like a SVM

- Cascade features: e.g. extract POS, construct a parse tree



- Extract hand-made features from the parse tree
- Feed these features to a simple classifier like a SVM

Goals

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

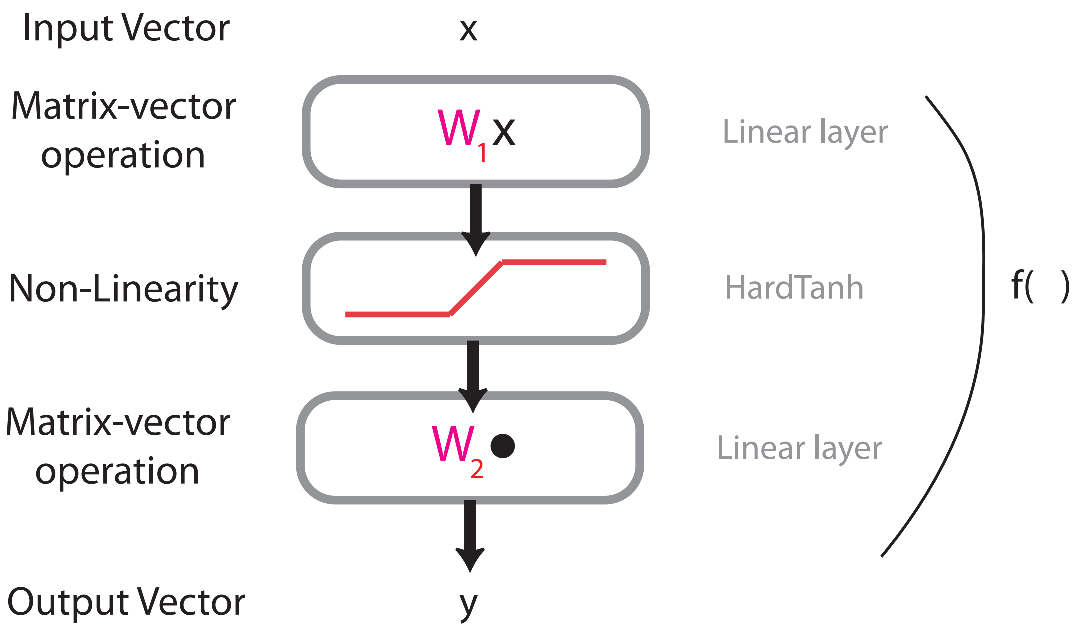
Means

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

II. Embeddings and models

Multilayer Networks

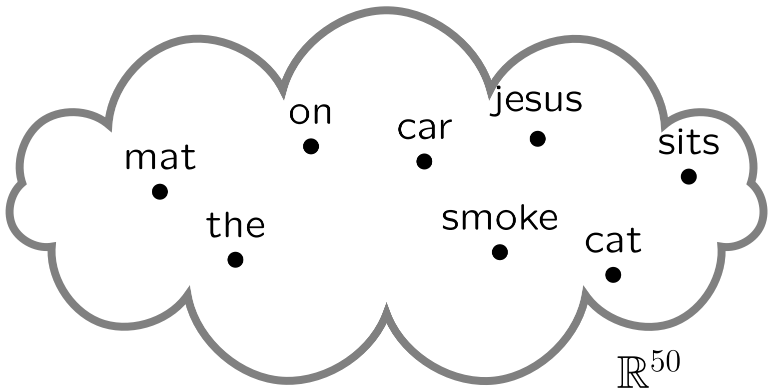
- Stack several layers together



- Increasing level of abstraction at each layer
- Requires simpler features than "shallow" classifiers
- The "weights" W_i are trained by gradient descent
- How can we feed words?

Idea

- Words are embedded in a vector space



- Embeddings are trained

Implementation

- A word w is an index in a dictionary $\mathcal{D} \in \mathbb{N}$
- Use a lookup-table ($W \sim$ feature size $\times$ dictionary size)

$$LT_W(w) = W_{\bullet\, w}$$

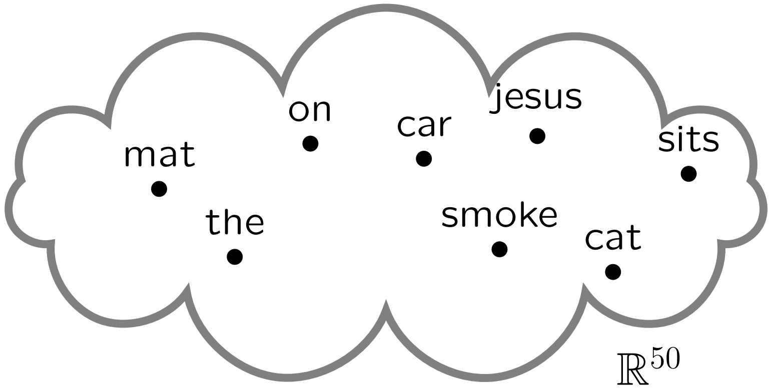
Remarks

- Applicable to any discrete feature (words, caps, stems...)
- See (Bengio et al, 2001)

Words into Vectors

Idea

- Words are embedded in a vector space



- Embeddings are trained

Implementation

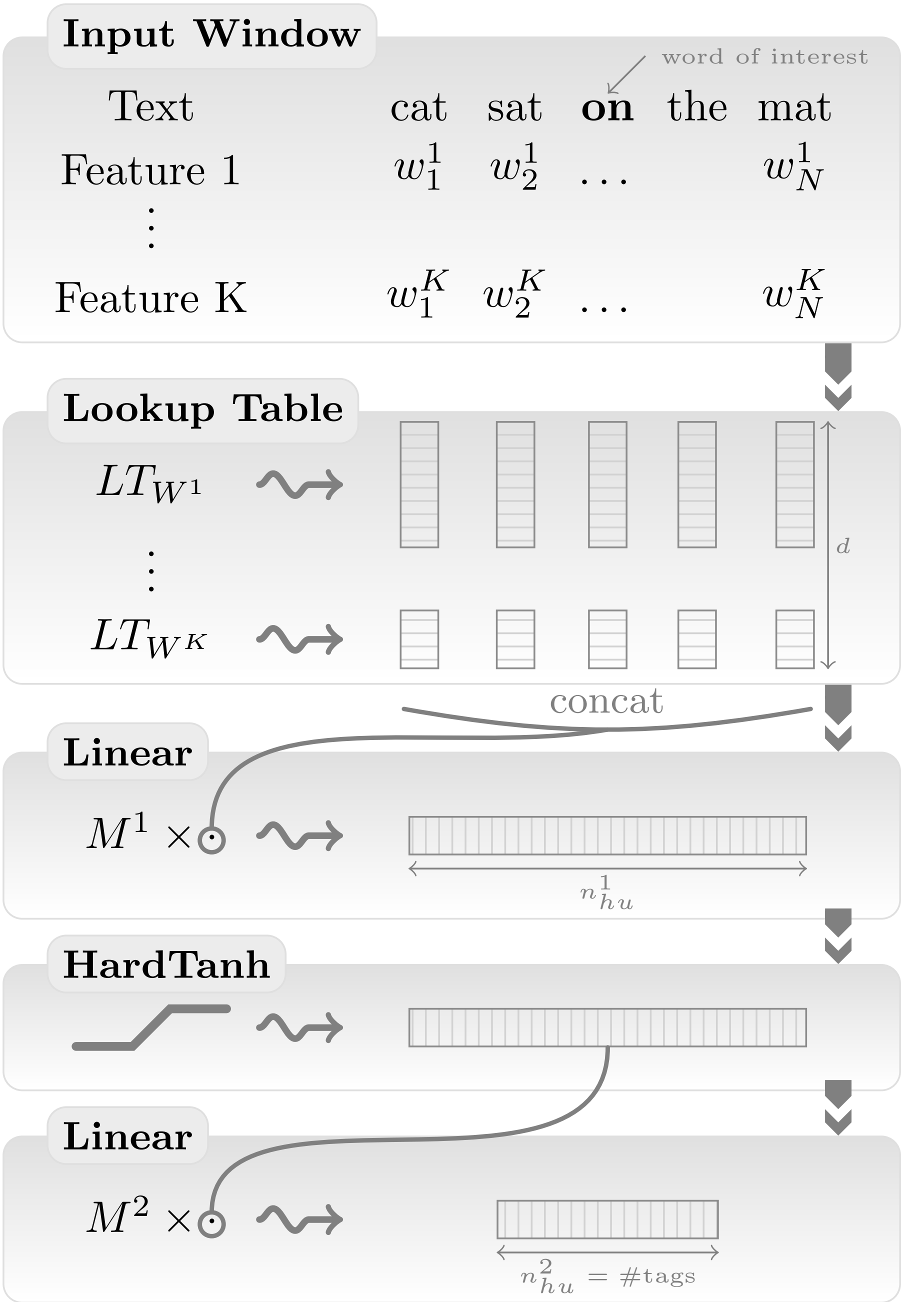
- A word w is an index in a dictionary $\mathcal{D} \in \mathbb{N}$
- Use a lookup-table ($W \sim$ feature size $\times$ dictionary size)

$$LT_W(w) = W_{\bullet\, w}$$

Remarks

- Applicable to any discrete feature (words, caps, stems...)
- See (Bengio et al, 2001)

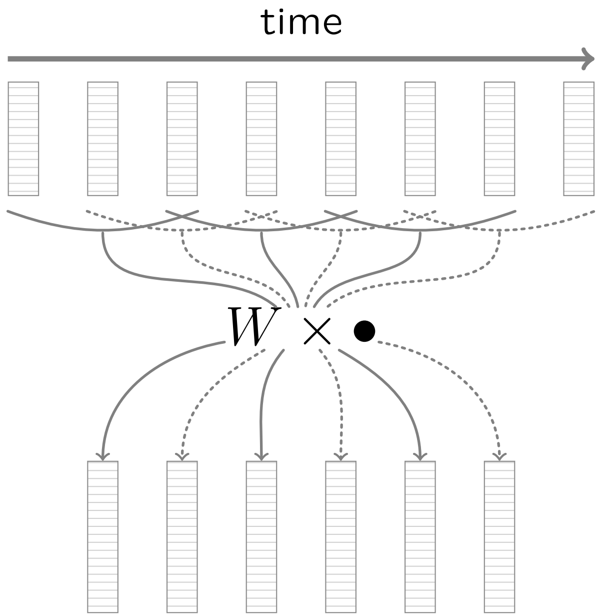
Window Approach



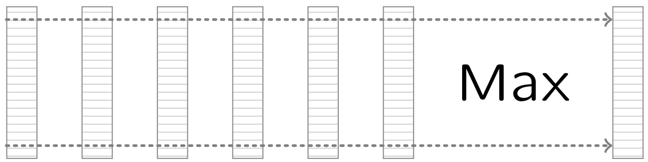
- Tags one word at the time
- Feed a fixed-size window of text around each word to tag
- Works fine for most tasks
- How do deal with long-range dependencies?

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

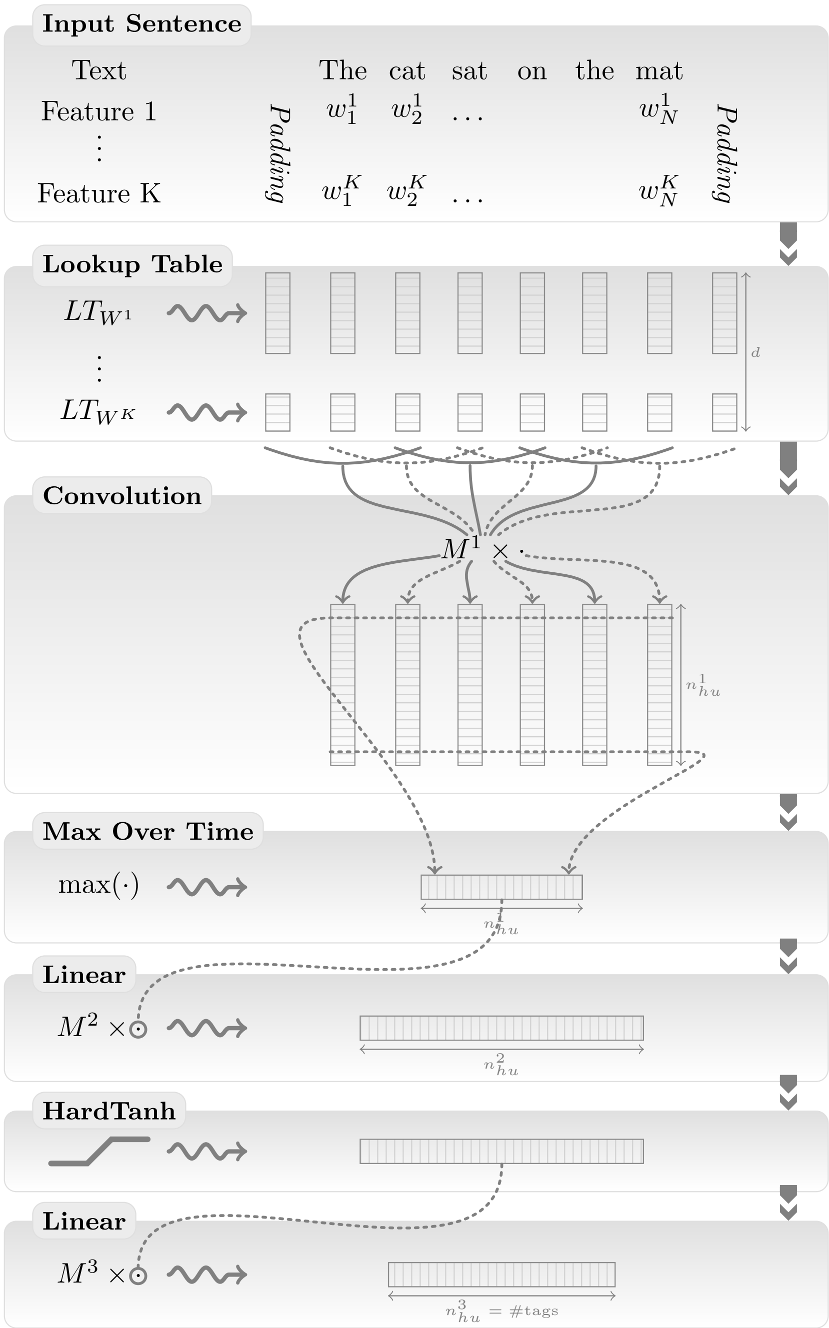
- Feed the whole sentence to the network
- Tag one word at the time: add extra position features
- Convolutions to handle variable-length inputs



- 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}
d
Convolution
$M^1 \times \cdot$
n^1_{hu}
Max Over Time
$\max(\cdot)$
n^1_{hu}
Linear
$M^2 \times \odot$
n^2_{hu}
HardTanh
Linear
$M^3 \times \odot$
$n^3_{hu} = \#\text{tags}$

Training

- Given a training set $\mathcal{T}$
- Convert network outputs into probabilities
- Maximize a log-likelihood

$$\boldsymbol{\theta} \longmapsto \sum_{(\boldsymbol{x},y)\in\mathcal{T}} \log p(y \,|\, \boldsymbol{x},\, \boldsymbol{\theta})$$

- Use stochastic gradient (See Bottou, 1991)

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \lambda \frac{\partial \log p(y \,|\, \boldsymbol{x},\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Fixed learning rate. "Tricks":

⋆ Divide learning by "fan-in"

⋆ Initialization according to "fan-in"

- Use chain rule ("back-propagation") for efficient gradient computation

Network $f(\cdot)$ has L layers

$$f = f_L \circ \cdots \circ f_1$$

Parameters

$$\boldsymbol{\theta} = (\boldsymbol{\theta_L},\, \ldots,\, \boldsymbol{\theta_1})$$

$$\frac{\partial \log p(y \,|\, \boldsymbol{x},\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta_i}} = \frac{\partial \log p(y \,|\, \boldsymbol{x},\, \boldsymbol{\theta})}{\partial f_i} \cdot \frac{\partial f_i}{\partial \boldsymbol{\theta_i}}$$

$$\frac{\partial \log p(y \,|\, \boldsymbol{x},\, \boldsymbol{\theta})}{\partial f_{i-1}} = \frac{\partial \log p(y \,|\, \boldsymbol{x},\, \boldsymbol{\theta})}{\partial f_i} \cdot \frac{\partial f_i}{\partial f_{i-1}}$$

- How to interpret neural networks outputs as probabilities?

Word Tag Likelihood (WTL)

- The network has one output $f(\boldsymbol{x}, i, \boldsymbol{\theta})$ per tag i
- Interpreted as a probability with a softmax over all tags

$$p(i \,|\, \boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{f(\boldsymbol{x}, i, \boldsymbol{\theta})}}{\sum_j e^{f(\boldsymbol{x}, j, \boldsymbol{\theta})}}$$

- Define the logadd operation

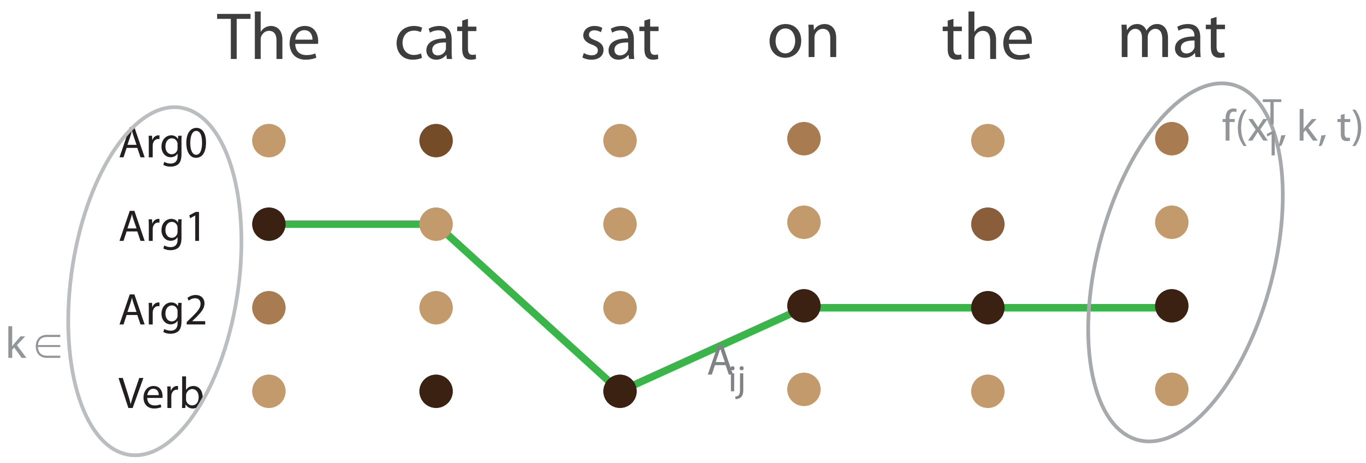
$$\operatorname*{logadd}_i z_i = \log(\sum_i e^{z_i})$$

- Log-likelihood for example $(\boldsymbol{x}, y)$

$$\log p(y \,|\, \boldsymbol{x}, \boldsymbol{\theta}) = f(\boldsymbol{x}, y, \boldsymbol{\theta}) - \operatorname*{logadd}_j f(\boldsymbol{x}, j, \boldsymbol{\theta})$$

- How to leverage the sentence structure?

- The network score for tag k at the t^{th} word is $f(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, k,\, t,\, \boldsymbol{\theta})$
- A_{kl} transition score to jump from tag k to tag l



- Sentence score for a tag path $i_1 \ldots i_T$

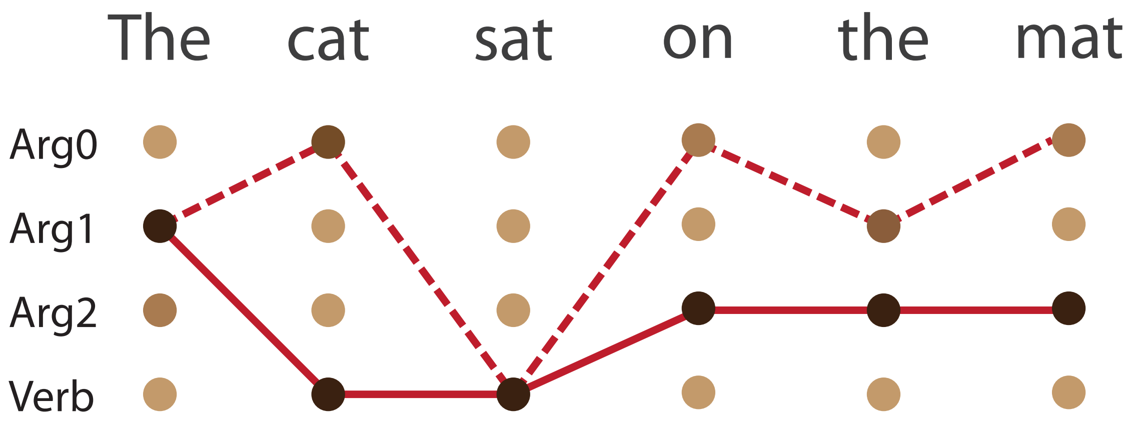
$$s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, i_1 \ldots i_T,\, \tilde{\boldsymbol{\theta}}) = \sum_{t=1}^{T} \left(A_{i_{t-1} i_t} + f(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, i_t,\, t,\, \boldsymbol{\theta}) \right)$$

- Conditional likelihood by normalizing w.r.t all possible paths:

$$\log p(y_1 \ldots y_T \,|\, \boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, \tilde{\boldsymbol{\theta}}) = s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, y_1 \ldots y_T,\, \tilde{\boldsymbol{\theta}}) - \underset{j_1 \ldots j_T}{\operatorname{logadd}}\, s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, j_1 \ldots j_T,\, \tilde{\boldsymbol{\theta}})$$

- How to efficiently compute the normalization?

- The network score for tag k at the t^{th} word is $f(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, k,\, t,\, \boldsymbol{\theta})$
- A_{kl} transition score to jump from tag k to tag l



- Sentence score for a tag path $i_1 \ldots i_T$

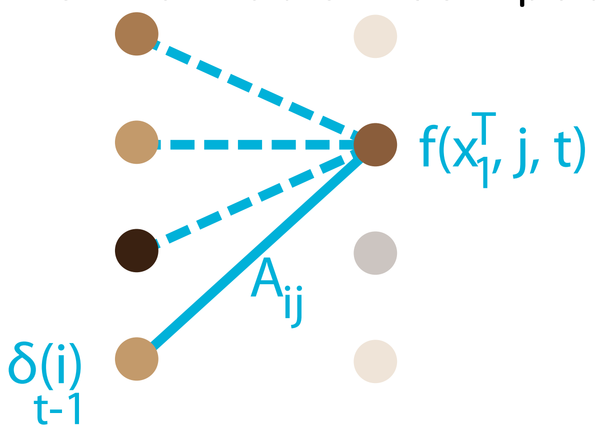
$$s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, i_1 \ldots i_T,\, \tilde{\boldsymbol{\theta}}) = \sum_{t=1}^{T} \Big(A_{i_{t-1} i_t} + f(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, i_t,\, t,\, \boldsymbol{\theta}) \Big)$$

- Conditional likelihood by normalizing w.r.t all possible paths:

$$\log p(y_1 \ldots y_T \,|\, \boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, \tilde{\boldsymbol{\theta}}) = s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, y_1 \ldots y_T,\, \tilde{\boldsymbol{\theta}}) - \underset{j_1 \ldots j_T}{\operatorname{logadd}}\, s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, j_1 \ldots j_T,\, \tilde{\boldsymbol{\theta}})$$

- How to efficiently compute the normalization?

- Normalization computed with recursive forward algorithm:



$$\delta_t(j) = \underset{i}{\text{logadd}} \left[\delta_{t-1}(i) + A_{i,j} + f_\theta(j, \boldsymbol{x}_1 \ldots \boldsymbol{x}_T, t)\right]$$

Termination:

$$\underset{j_1 \ldots j_T}{\text{logadd}}\, s(\boldsymbol{x}_1 \ldots \boldsymbol{x}_T,\, j_1 \ldots j_T,\, \tilde{\boldsymbol{\theta}}) = \underset{i}{\text{logadd}}\, \delta_T(i)$$

- Simply backpropagate through this recursion with chain rule

- Non-linear CRFs: Graph Transformer Networks

- Compared to CRFs, we train features
  (network parameters $\boldsymbol{\theta}$ and transitions scores A_{kl})

- Inference: Viterbi algorithm
  (replace logadd by max)

Supervised Benchmark Results

- Network architectures:
  - ⋆ Window (5) approach for POS, CHUNK & NER (300HU)
  - ⋆ Convolutional (3) for SRL (300+500HU)
  - ⋆ Word Tag Likelihood (WTL) and Sentence Tag Likelihood (STL)
- Network features: lower case words (size 50), capital letters (size 5)
  dictionary size 100,000 words

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*

- STL helps, but... fair performance.
- Capacity mainly in words features... are we training it 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!

III. Lots of unlabeled data

Ranking Language Model

- Language Model: *"is a sentence actually english or not?"*
  Implicitly captures: syntax and semantics.
- Estimating the probability of next word given previous words:
  Overkill because we do not need probabilities here
- Likelihood criterion largely determined by the most frequent phrases
- Rare legal phrases are no less significant that common phrases
- $f()$ a window approach network
- Ranking margin cost:

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

$\mathcal{S}$: 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

- Stochastic training:
  - ⋆ positive example: random corpus sentence
  - ⋆ negative example: replace middle word by random word

Training Language Model

- Two window approach (11) networks (100HU) trained on two corpus:
  - LM1: Wikipedia: **631M** of words
  - LM2: Wikipedia+Reuters RCV1: **631M+221M=852M** of words
- Massive dataset: cannot afford classical training-validation scheme
- Like in biology: breed a couple of network lines
- Breeding decisions according to 1M words validation set
- LM1
  - order dictionary words by frequency
  - increase dictionary size: 5000, $10,000$, $30,000$, $50,000$, $100,000$
  - 4 weeks of training
- LM2
  - initialized with LM1, dictionary size is $130,000$
  - 30,000 additional most frequent Reuters words
  - 3 additional weeks of training

Unsupervised Word Embeddings

france	jesus	xbox	reddish	scratched	megabits
454	1973	6909	11724	29869	87025
austria	god	amiga	greenish	nailed	octets
belgium	sati	playstation	bluish	smashed	mb/s
germany	christ	msx	pinkish	punched	bit/s
italy	satan	ipod	purplish	popped	baud
greece	kali	sega	brownish	crimped	carats
sweden	indra	psNUMBER	greyish	scraped	kbit/s
norway	vishnu	hd	grayish	screwed	megahertz
europe	ananda	dreamcast	whitish	sectioned	megapixels
hungary	parvati	geforce	silvery	slashed	gbit/s
switzerland	grace	capcom	yellowish	ripped	amperes

Semi-Supervised Benchmark Results

- Initialize word embeddings with LM1 or LM2
- Same training procedure

Approach	**POS** (PWA)	**CHUNK** (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*
NN+WTL+LM1	97.05	91.91	85.68	58.18
NN+STL+LM1	97.10	93.65	87.58	*73.84*
NN+WTL+LM2	97.14	92.04	86.96	–
NN+STL+LM2	97.20	93.63	88.67	*74.05*

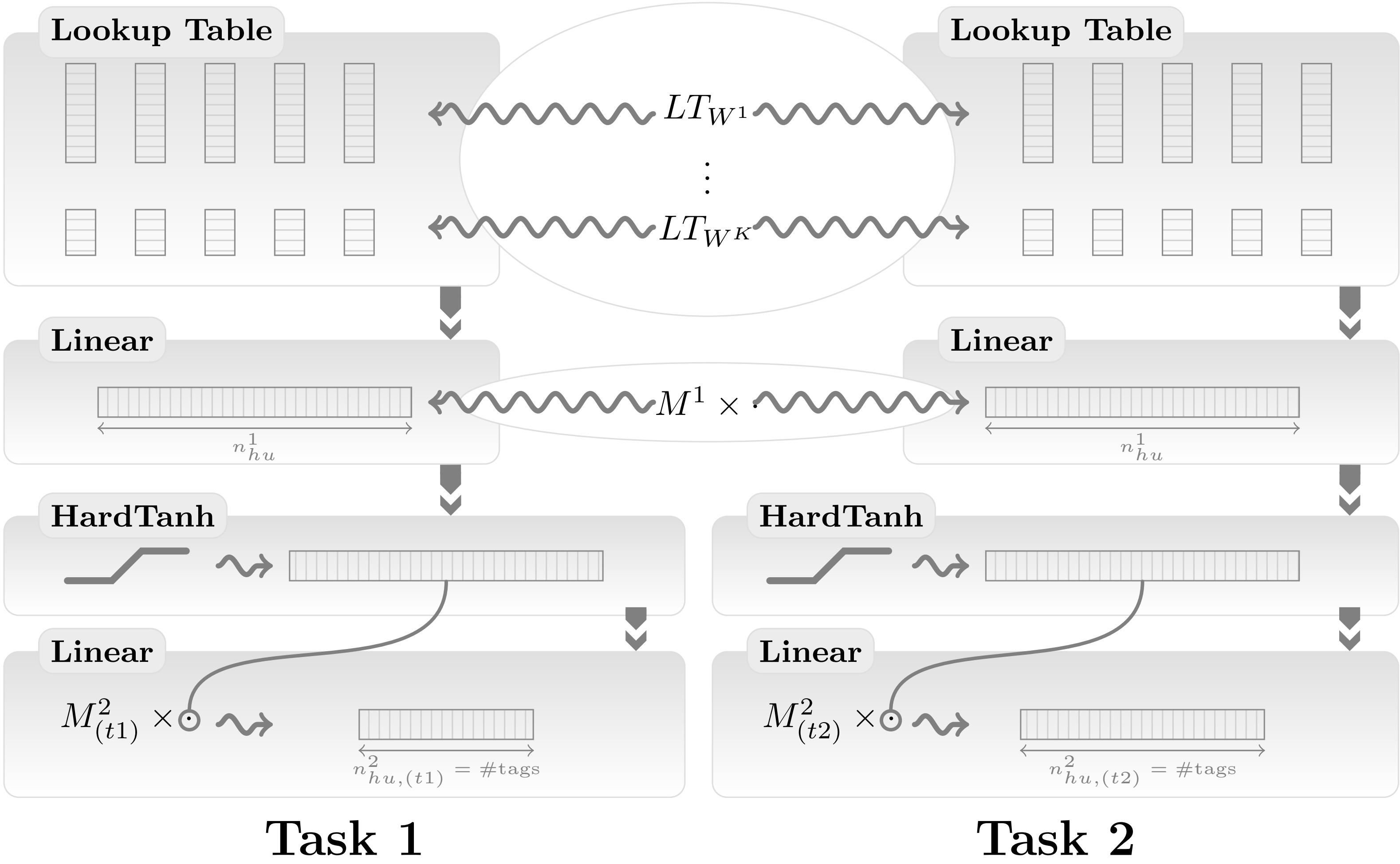
- Huge boost from language models
- Training set word coverage:

	LM1	**LM2**
POS	97.86%	98.83%
CHK	97.93%	98.91%
NER	95.50%	98.95%
SRL	97.98%	98.87%

IV. Multi-task learning

Multi-Task Learning

- Joint training
- Good overview in (Caruana, 1997)



Multi-Task Learning Benchmark Results

Approach	**POS** (PWA)	**CHUNK** (F1)	**NER** (F1)
Benchmark Systems	**97.24**	**94.29**	**89.31**
NN+STC+LM2	97.20	93.63	88.67
NN+STC+LM2+MTL	97.22	94.10	88.62

V. Task dependent hacks

Cascading Tasks

Increase level of engineering by incorporating common NLP techniques

- Stemming for western languages benefits POS (Ratnaparkhi, 1996)
  - ⋆ Use last two characters as feature (455 different stems)
- Gazetteers are often used for NER (Florian, 2003)
  - ⋆ $8,000$ locations, person names, organizations and misc entries from CoNLL 2003
- POS is a good feature for CHUNK & NER (Shen, 2005) (Florian, 2003)
  - ⋆ We feed our own POS tags as feature
- CHUNK is also a common feature for SRL (Koomen, 2005)
  - ⋆ We feed our own CHUNK tags as feature

Cascading Tasks Benchmark Results

Approach	POS (PWA)	CHUNK (F1)	NER (F1)	SRL
Benchmark Systems	**97.24**	**94.29**	**89.31**	**77.92**
NN+STC+LM2	97.20	93.63	88.67	*74.05*
NN+STC+LM2+Suffix2	97.29	–	–	–
NN+STC+LM2+Gazetteer	–	–	89.59	–
NN+STC+LM2+POS	–	94.32	88.67	–
NN+STC+LM2+CHUNK	–	–	–	*74.68*

Variance

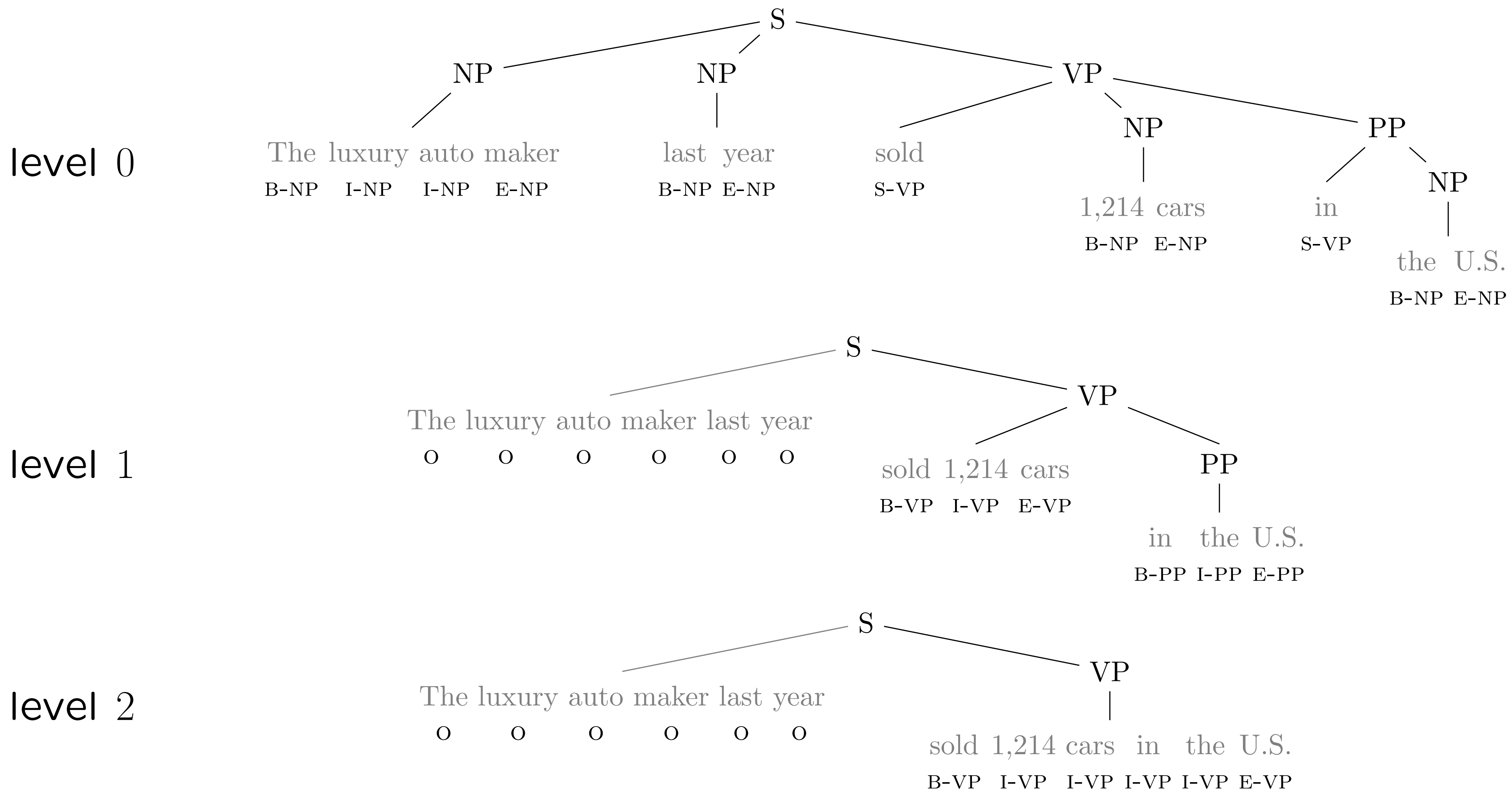
- Train 10 networks

Approach		POS (PWA)	CHUNK (F1)	NER (F1)
Benchmark Systems		**97.24%**	**94.29%**	**89.31%**
NN+STC+LM2+POS	worst	97.29%	93.99%	89.35%
NN+STC+LM2+POS	mean	97.31%	94.17%	89.65%
NN+STC+LM2+POS	best	97.35%	94.32%	89.86%

- Previous experiments:
  same seed was used for all networks to reduce variance

Parsing

- Parsing is essential to SRL (Punyakanok, 2005) (Pradhan, 2005)
- State-of-the-art SRL systems use several parse trees (up to 6!!)
- We feed our network several levels of the Charniak parse tree provided by CoNLL 2005



SRL Benchmark Results With Parsing

Approach	SRL (test set F1)
Benchmark System (six parse trees)	**77.92**
Benchmark System (top Charniak only)	**74.76**†
NN+STC+LM2	*74.05*
NN+STC+LM2+CHUNK	*74.68*
NN+STC+LM2+Charniak (level 0 only)	*75.45*
NN+STC+LM2+Charniak (levels 0 & 1)	*75.86*
NN+STC+LM2+Charniak (levels 0 to 2)	*75.79*
NN+STC+LM2+Charniak (levels 0 to 3)	*75.90*
NN+STC+LM2+Charniak (levels 0 to 4)	*75.66*

†on the validation set

Engineering a Sweet Spot

- SENNA: implements our networks in simple C ($\approx$ 2500 lines)
- Neural networks mainly perform matrix-vector multiplications: use BLAS
- All networks are fed with lower case words (130,000) and caps features
- POS uses prefixes
- CHUNK uses POS tags
- NER uses gazetteer
- SRL uses level 0 of parse tree
  - ⋆ We trained a network to predict level 0 (uses POS tags): 92.25% F1 score against 91.94% for Charniak
  - ⋆ We trained a network to predict verbs as in SRL
  - ⋆ Optionaly, we can use POS verbs

SENNA Speed

System	RAM (Mb)	Time (s)
Toutanova, 2003	1100	1065
Shen, 2007	2200	833
SENNA	32	4

(a) POS

System	RAM (Mb)	Time (s)
Koomen, 2005	3400	6253
SENNA	124	52

(b) SRL

SENNA Demo

- Will be available in January at

  `http://ml.nec-labs.com/software/senna`

- If interested: email `ronan@collobert.com`

Terminal — emacs — 109×32

```
void SENNA_nn_viterbi(int *path, float *init, float *transition, float *emission, int N, int T)
{
  float *delta, *deltap;
  int *phi;
  int i, j, t;

  /* misc allocations */
  delta = SENNA_malloc(sizeof(float), N);
  deltap = SENNA_malloc(sizeof(float), N);
  phi = SENNA_malloc(sizeof(float), N*T);

  /* init */
  for(i = 0; i < N; i++)
    deltap[i] = init[i] + emission[i];

  /* recursion */
  for(t = 1; t < T; t++)
  {
    float *deltan = delta;
    for(j = 0; j < N; j++)
    {
      float maxValue = -FLT_MAX;
      int maxIndex = 0;
      for(i = 0; i < N; i++)
      {
        float z = deltap[i] + transition[i+j*N];
        if(z > maxValue)
        {
          maxValue = z;
          maxIndex = i;
-uu-:---F1  SENNA_nn.c      73% (165,0)     (C/l Abbrev)--------------------------------------------
```

Conclusion

Results

- "All purpose" neural network architecture for NLP
- Limit task-specific engineering
- Rely on very large unlabeled datasets
- Still room for improvements

Criticism

- Why forgetting NLP expertise for neural network training skills?
  - ⋆ NLP goals are not limited to existing NLP task
  - ⋆ Excessive task-specific engineering is not desirable
- Why neural networks?
  - ⋆ Scale on massive datasets
  - ⋆ Discover hidden representations
  - ⋆ Most of neural network technology existed in 1997

If we had started in 1997 with vintage computers,
training would be near completion today!!