Deep Learning for Natural Language Processing

Ronan Collobert
NEC Labs America, Princeton, USA

Jason Weston
Google, New York, USA

Joint work with Leon Bottou, David Grangier, Bing Bai, Yanjun Qi, Antoine Bordes, Nicolas Usunier, Koray Kavukcuoglu, Pavel Kuksa, Corinna Cortes and Mehryar Mohri.

Deep Learning
for Natural Language Processing

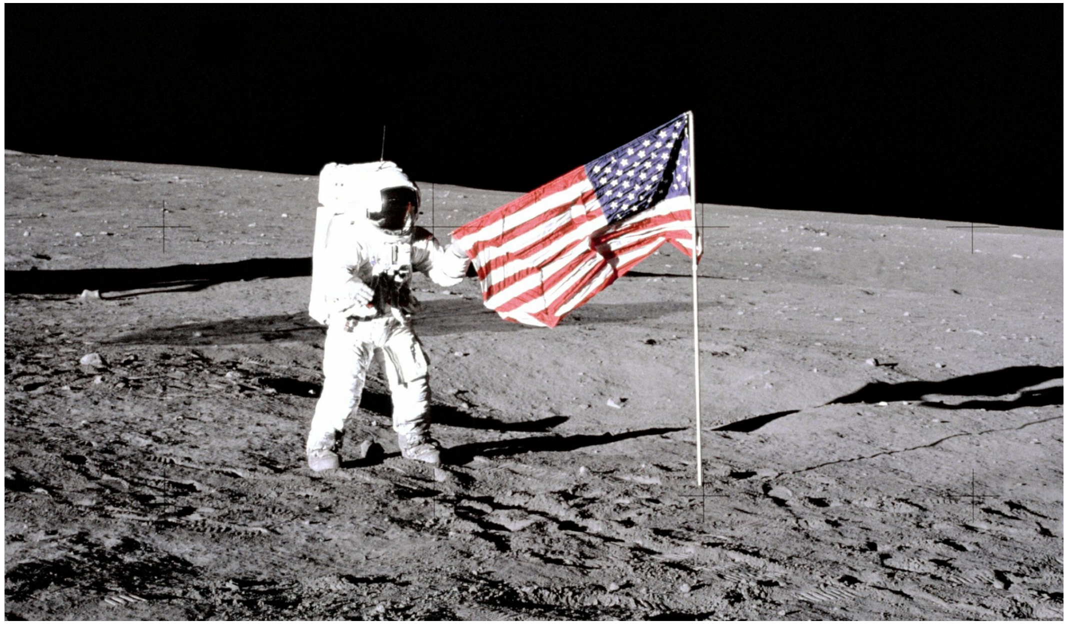
Ronan Collobert
NEC Labs America, Princeton, USA

Jason Weston
Google, New York, USA

Disclaimer: the characters and events depicted in this movie are fictitious. Any similarity to any person living or dead is merely coincidental.

A Brief History Of Machine Learning

As with the history of the world, machine learning has a history of



and

exploration

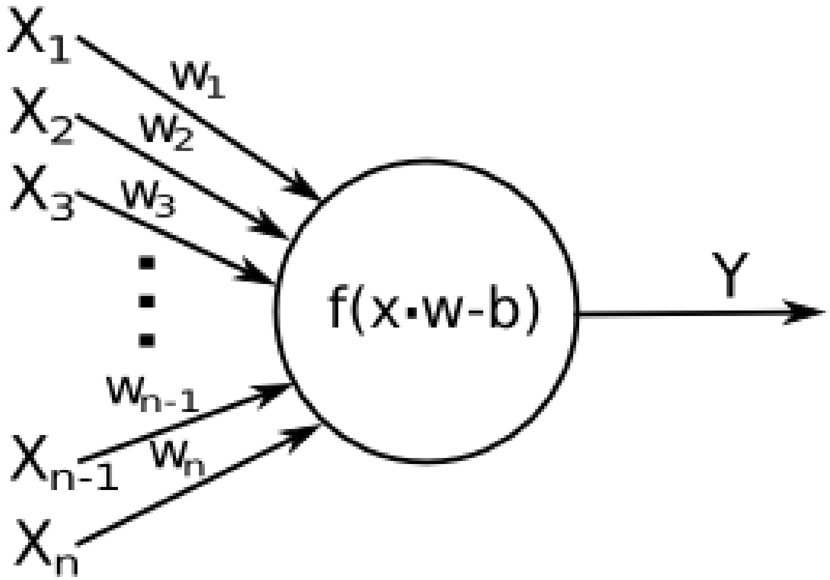
(finding new things)

exploitation

(of what you, or someone else, found)

(and sometimes wars because of it!)

In the beginning: discovery of the Perceptron



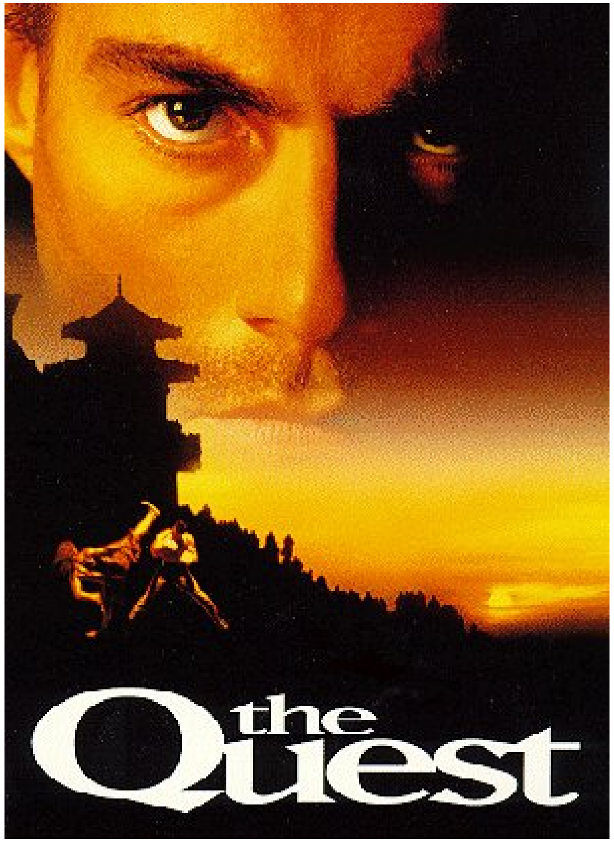
"It's cool, it's sexy." (Franky Rosenblatt 1957)

"It's linear. It sucks" (Minsky, Papert 1969)..

... and people believed Minksy, which made them sad ..

The Quest to Model Nonlinearities

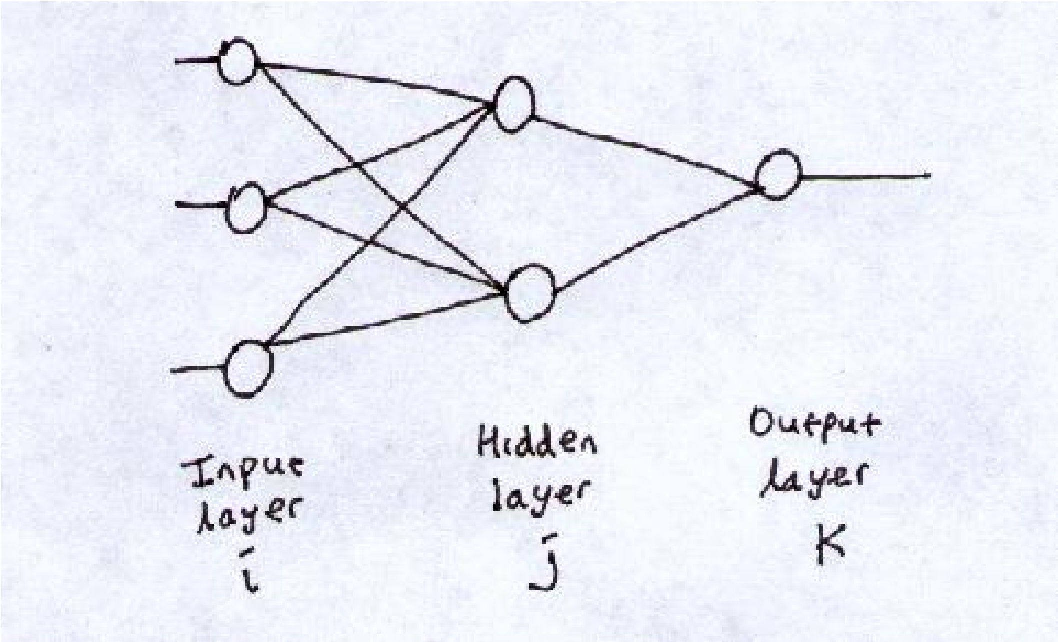
So they tried to make it nonlinear:



- Random projections to induce nonlinearities,
- Adding nonlinear features to the inputs, e.g. products of features,
- They even thought of kernels (Aizerman, Brav., Roz. 1964).

but they were still depressed.... until......

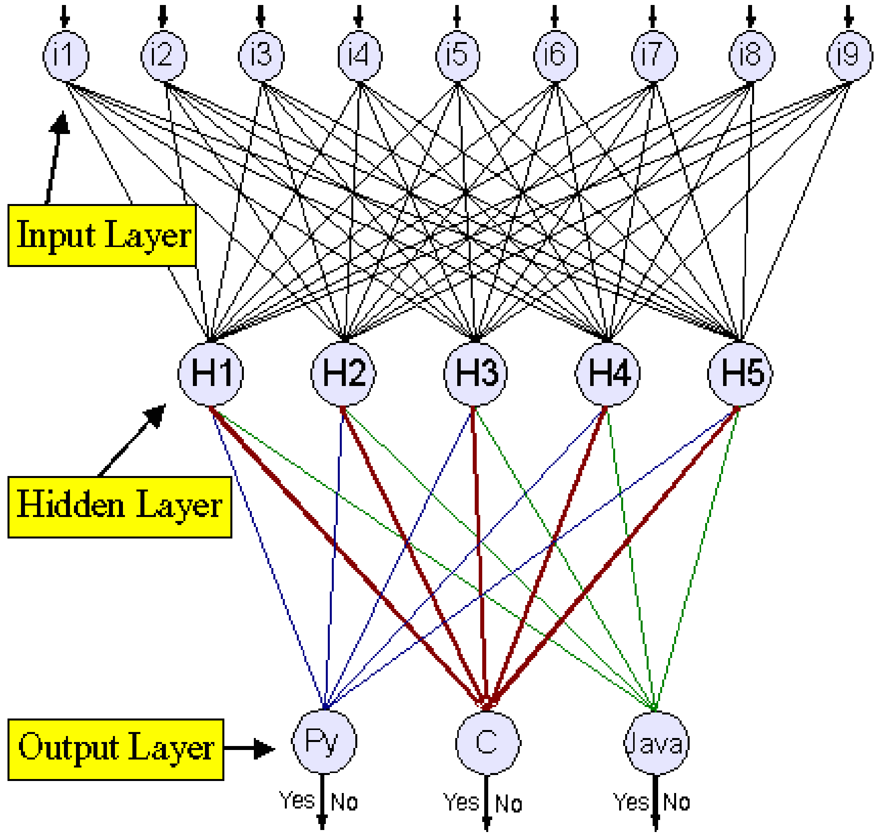
They Discovered Multi-Layered Perceptrons



(Backprop - Rumelhart, Hinton & Williams, 1986)

...**and they got excited..!** 

They were so excited they kept trying more and more things...



And more and more things...

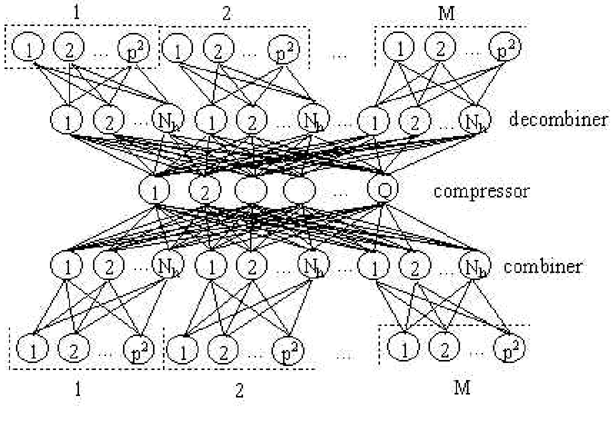
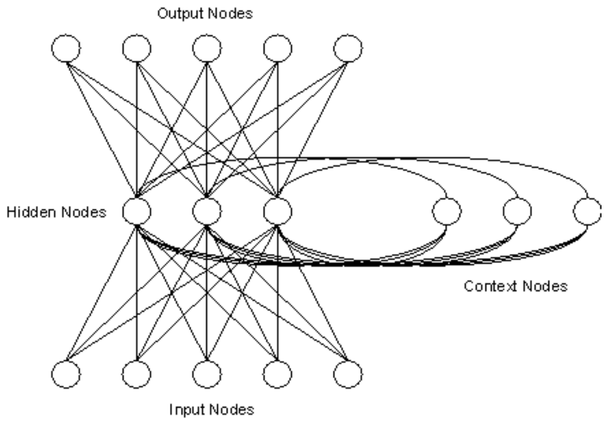
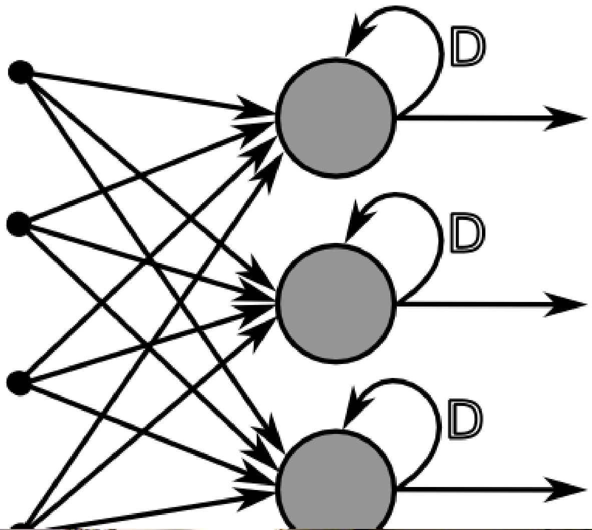
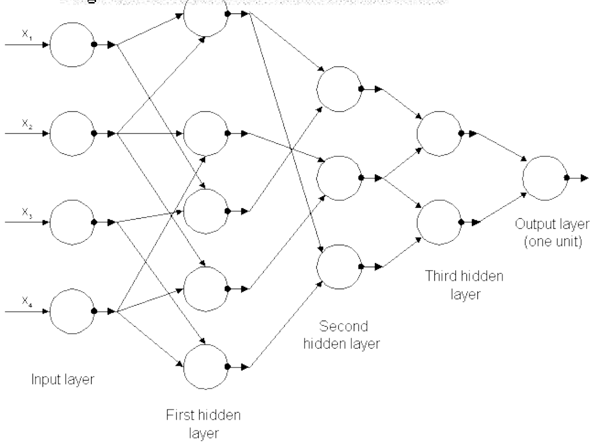
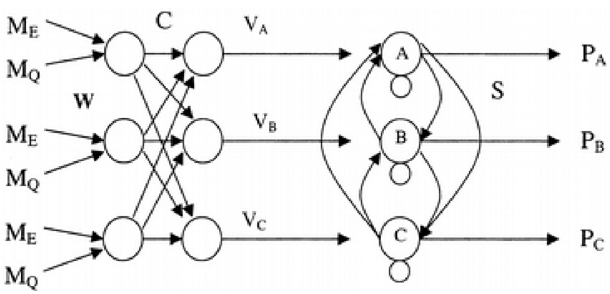


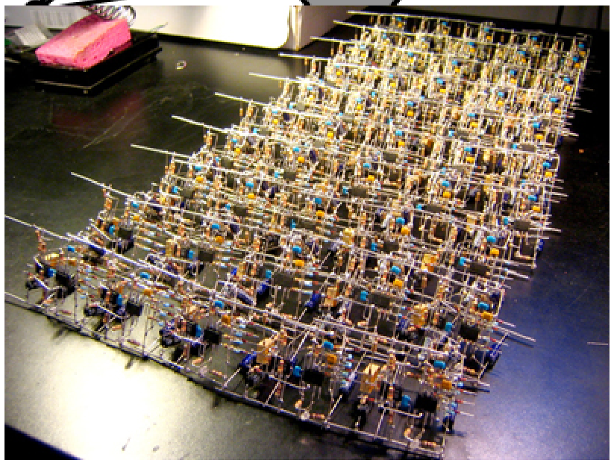
Figure 2 *Hierarchical neural network structure*







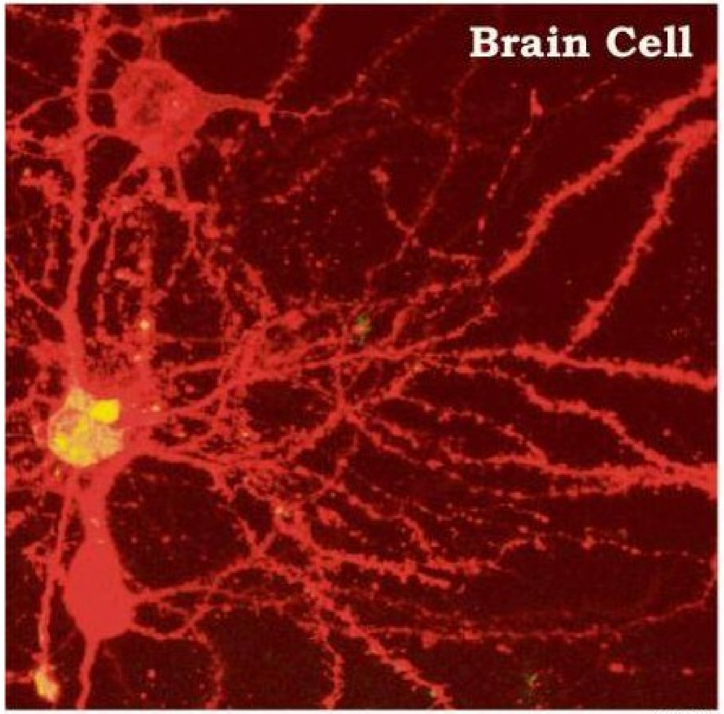




...until people got scared!

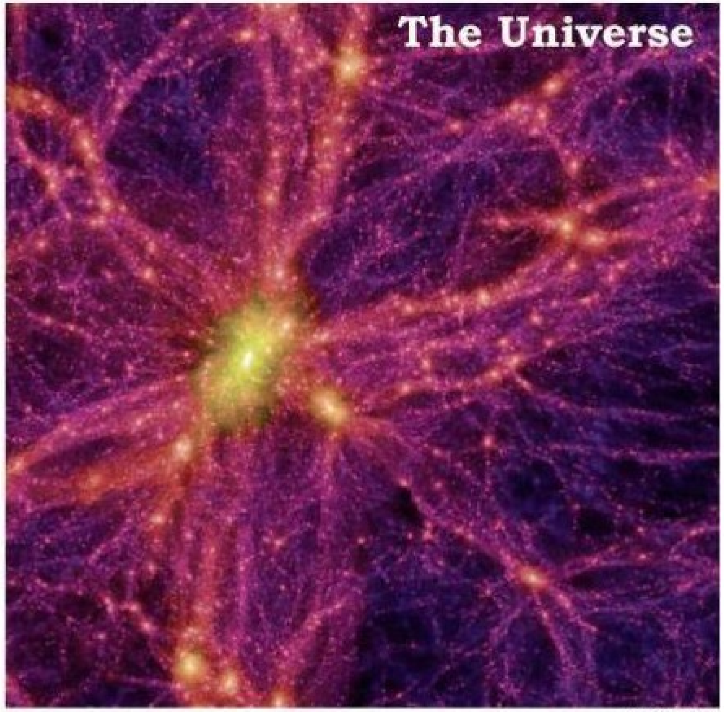
Even though they hadn't reached the complexity of the only known intelligent thing in the universe (the brain)

One is only micrometers wide. The other is billions of light-years across. One shows neurons in a mouse brain. The other is a simulated image of the universe. Together they suggest the surprisingly similar patterns found in vastly different natural phenomena. *DAVID CONSTANTINE*



Mark Miller

Mark Miller, a doctoral student at Brandeis University, is researching how particular types of neurons in the brain are connected to one another. By staining thin slices of a mouse's brain, he can identify the connections visually. The image above shows three neuron cells on the left (two red and one yellow) and their connections.



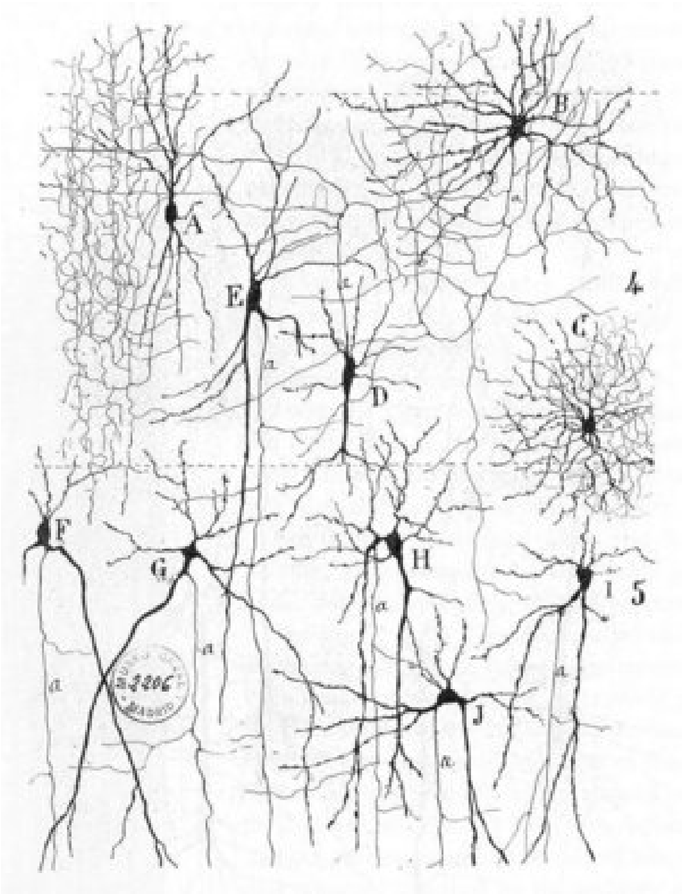
Virgo Consortium

An international group of astrophysicists used a computer simulation last year to recreate how the universe grew and evolved. The simulation image above is a snapshot of the present universe that features a large cluster of galaxies (bright yellow) surrounded by thousands of stars, galaxies and dark matter (web).

Source: Mark Miller, Brandeis University; Virgo Consortium for Cosmological Supercomputer Simulations; www.visualcomplexity.com

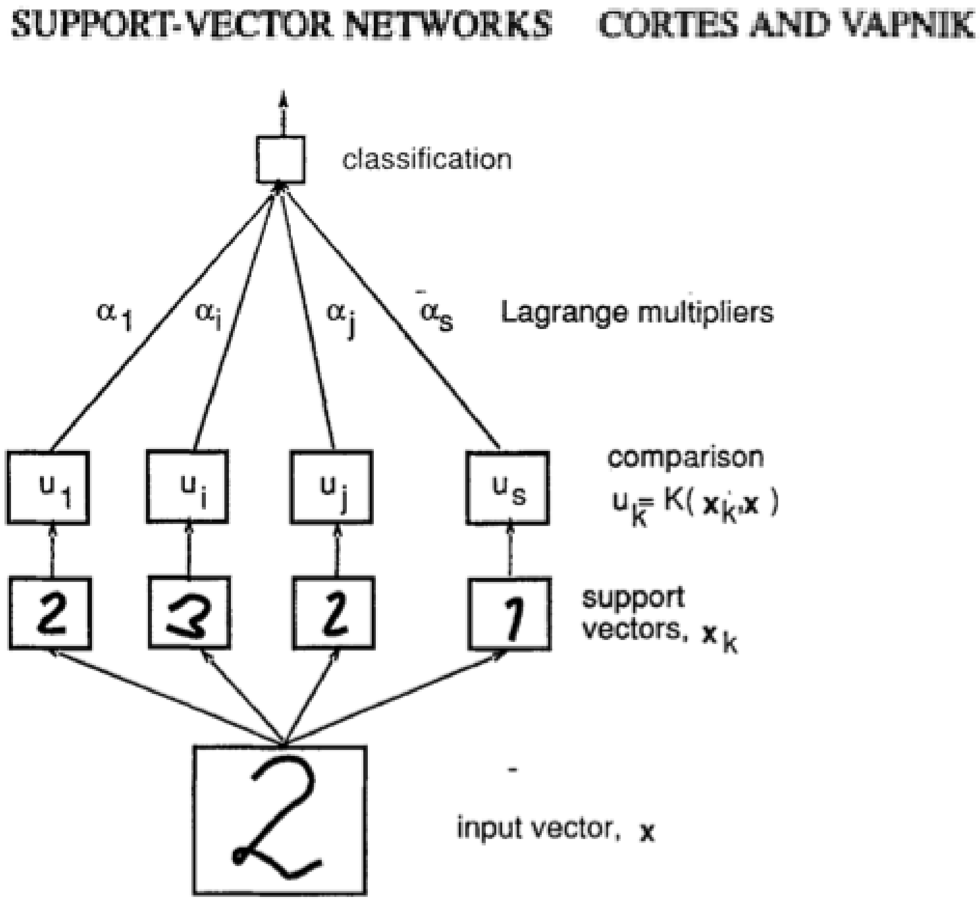
The New York Times

..and the universe they were trying to model itself seemed just as complex,



They decided what they were doing was too complex...

So they found something less complex... someone came up with a new Perceptron network!

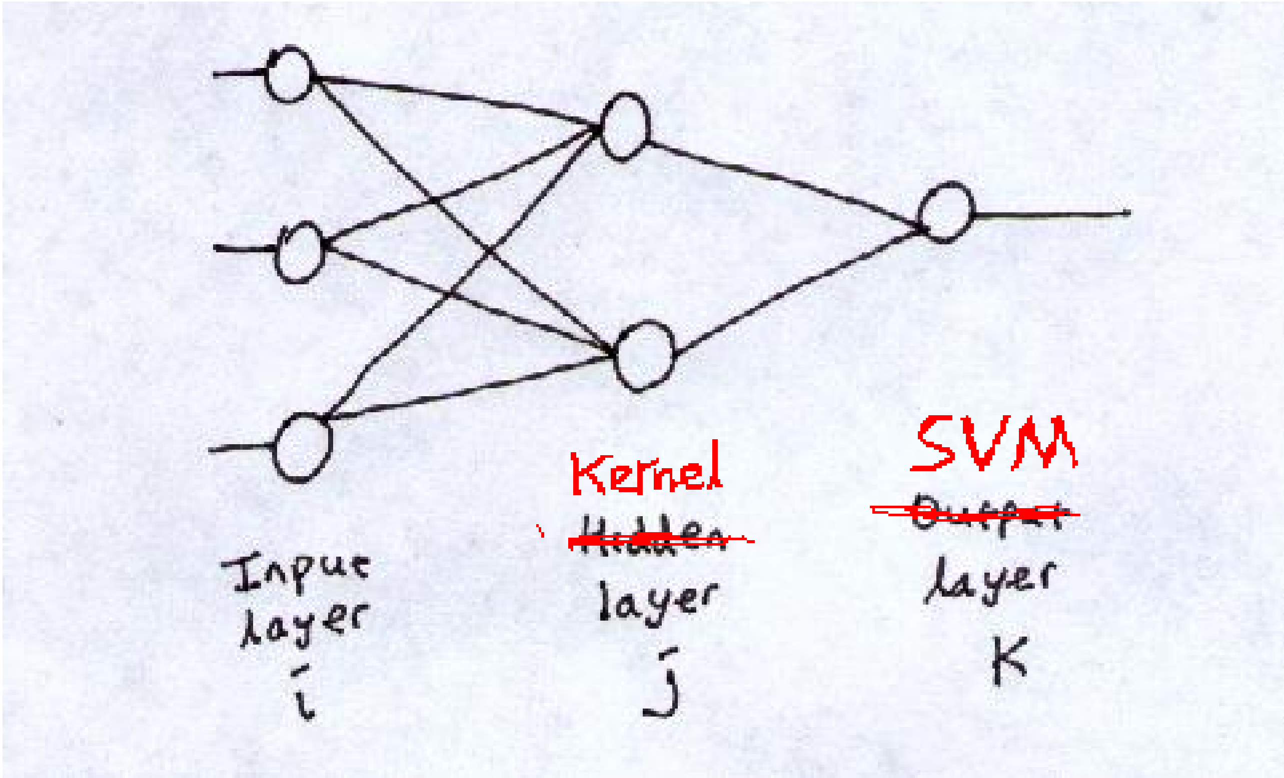


"It's cool. It's sexy" (Vlad Vapnik, 1992)
"Isn't it a linear model?" (Yann LeCun, 1992)

Life was Convex

... and life was good.
People published papers about it.
But it didn't do everything they wanted...

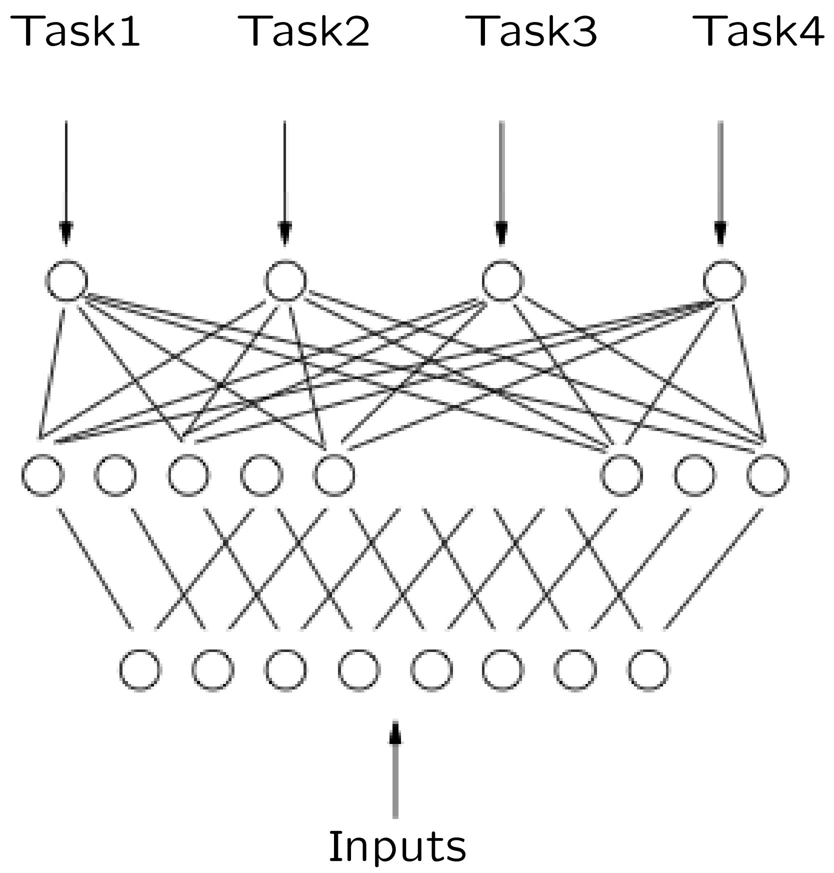
Learning Representations



Learning the kernel = multi-layer again!

Neural nets are an elegant model for learning representations.

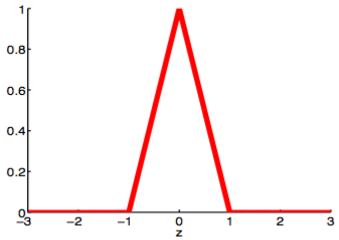
Multi-tasking: sharing features

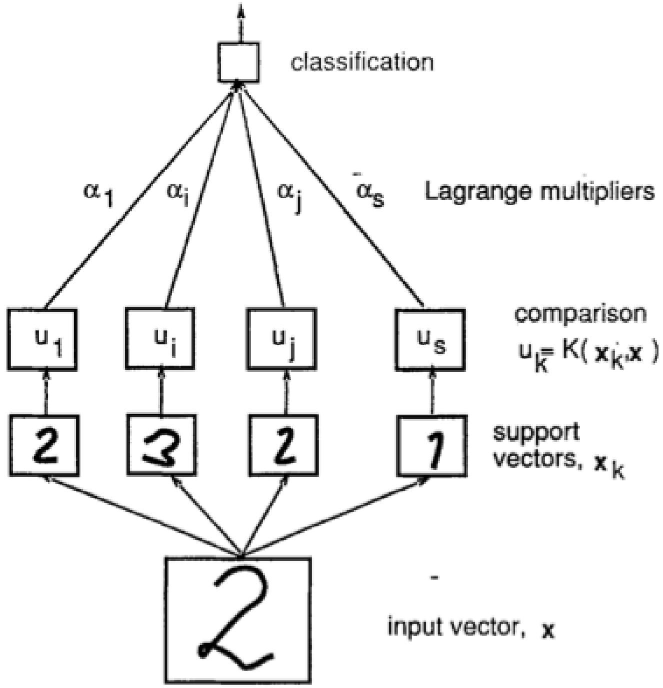


Non-convex even for linear models! (Ando & Zhang, 2005)

Nevertheless, Neural nets are an elegant model for multi-tasking.

Semi-supervised learning: Transductive SVM

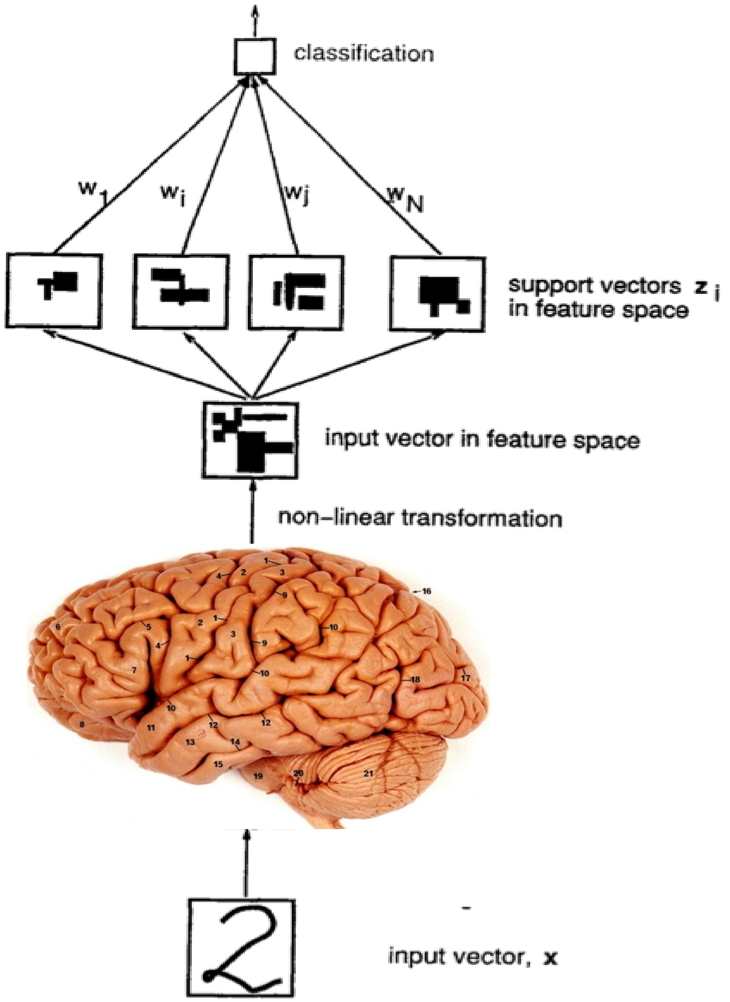




The loss was non-convex! (& convex relaxations = slow)

Semi-supervision for Neural nets is no problem, don't worry.

Feature Engineering

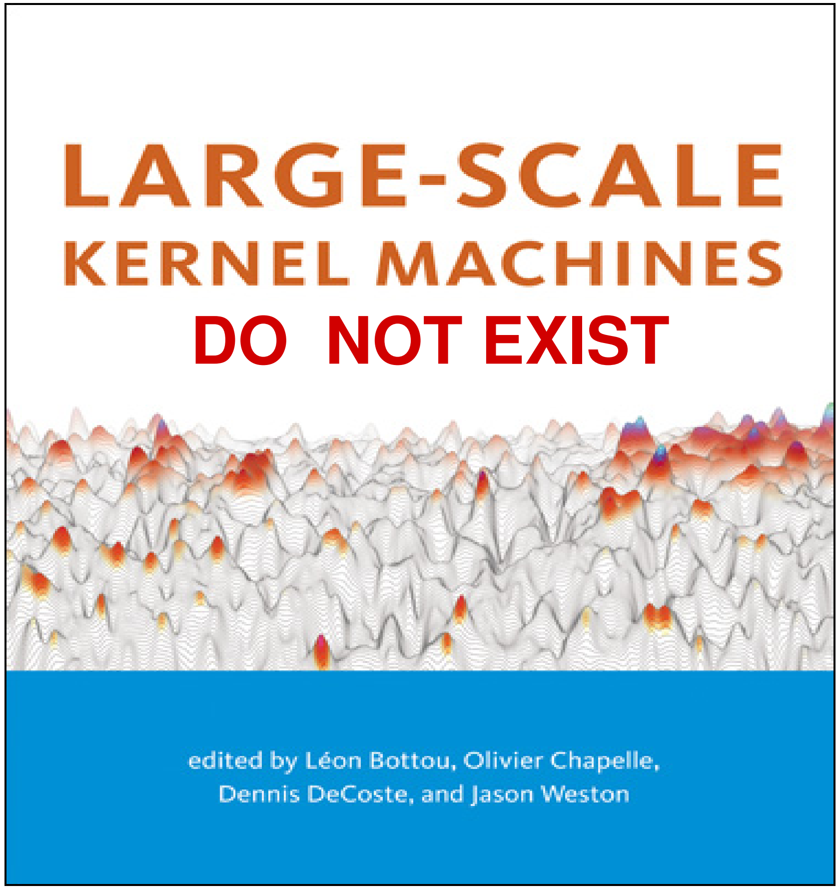


Multi-layer: 1^{st} layer = human brain = <u>nonconvex!!</u>

The first layers of a neural net use machine learning not human learning, which is what we're supposed to be doing.

Scalability

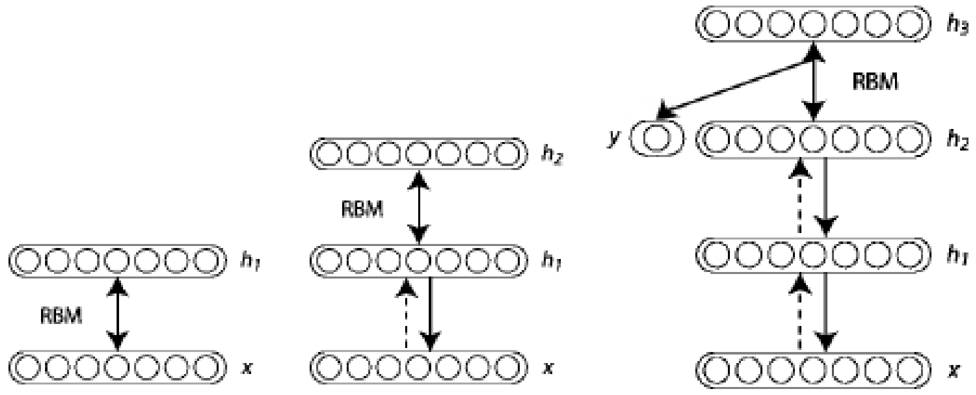
SVMs are slow, even though books were devoted to making them fast (Bottou, Chapelle, Descoste, Weston 2007). Problem: too many SVs!



Solutions:

- Using stochastic gradient descent **like NNs** (Bottou, NIPS 2008)
- Learning which SVs to use – non-convex, **like a 2-layer NN.**
- Using linear SVMs (very popular) – **back to the Perceptron!**

IDEA! Rebrand “Neural Nets” → “Deep Nets”



(a) Train RBM for $\mathbf{x}$ (b) Train RBM for $\mathbf{h}^1$ (c) Train RBM for $\mathbf{h}^2$ and y

(and add some semi-supervision to improve their performance)

 “It’s cool!” (Geoff Hinton, this morning after breakfast)

 “It’s sexy!” (Yann L. and Yoshua B., just before lunch)

 “Haven’t we been here before?” (Everyone else, 2009)

...BUT, still, some were  *excited* enough to come to this tutorial!

But seriously, putting it all together:

- NNs are flexible:
  - Different module (layers), losses, regularizers, . . .
  - Multi-tasking
  - Semi-supervised learning
  - Learning hidden representations
- NNs are scalable

The ideal tool for NLP!

All hail NNs!

This Talk: The Big Picture

The Goal:

- We want to have a conversation with our computer

  (not easy)

- Convert a piece of English into a computer-friendly data structure

  = find hidden representations

- Use NLP tasks to measure if the computer "understands"

Learn NLP from "scratch"
(i.e. minimal feature engineering)

The Plan:

Part I Brainwashing: Neural Networks are Awesome!
Part II Labeling: Hidden Representations for Tagging
Part III Retrieval: Hidden Representations for Semantic Search
Part IV Situated Learning: Hidden Representations for Grounding

Part II
NLP Labeling

Ronan Collobert — ronan@collobert.com

Jason Weston — jaseweston@gmail.com

Léon Bottou, Koray Kavukcuoglu, Pavel Kuksa

NEC Laboratories America, Google Labs

Natural Language Processing Tasks

- 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 ($\approx$ up to 1M labeled words)
  - ⋆ NER: Reuters ($\approx$ 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

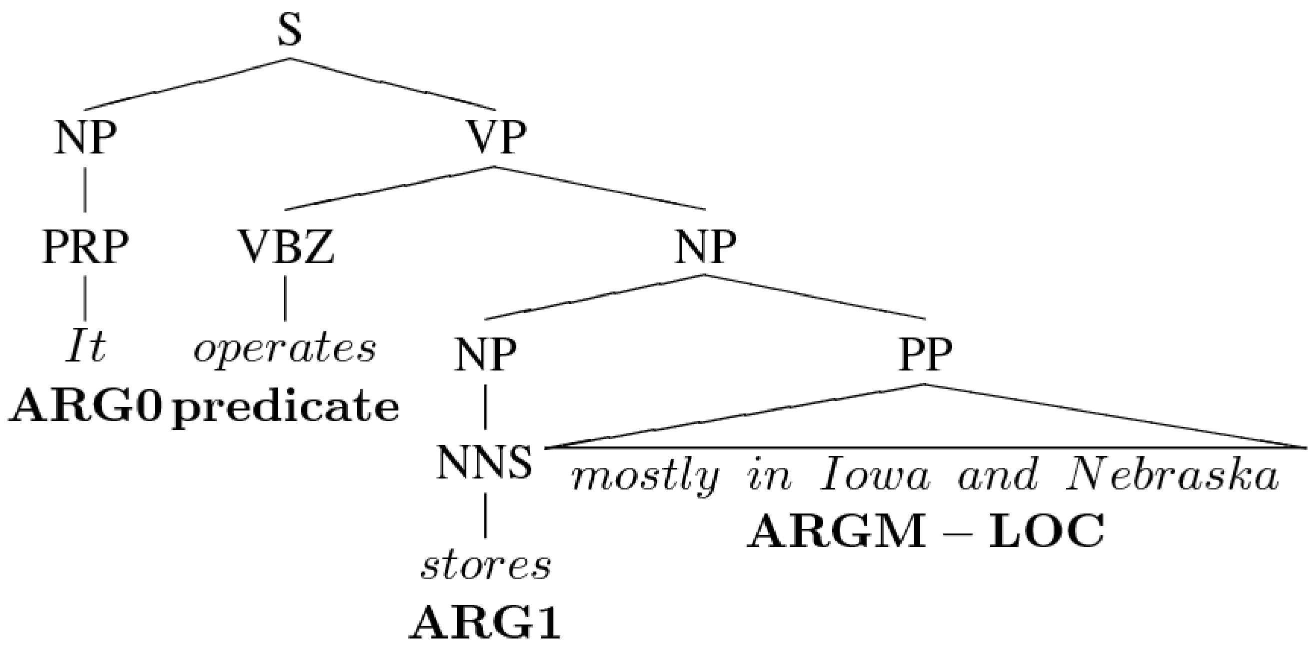
NLP: Large Scale Engineering (1/2)

- 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 shallow classifier like SVM

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



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

Goals

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

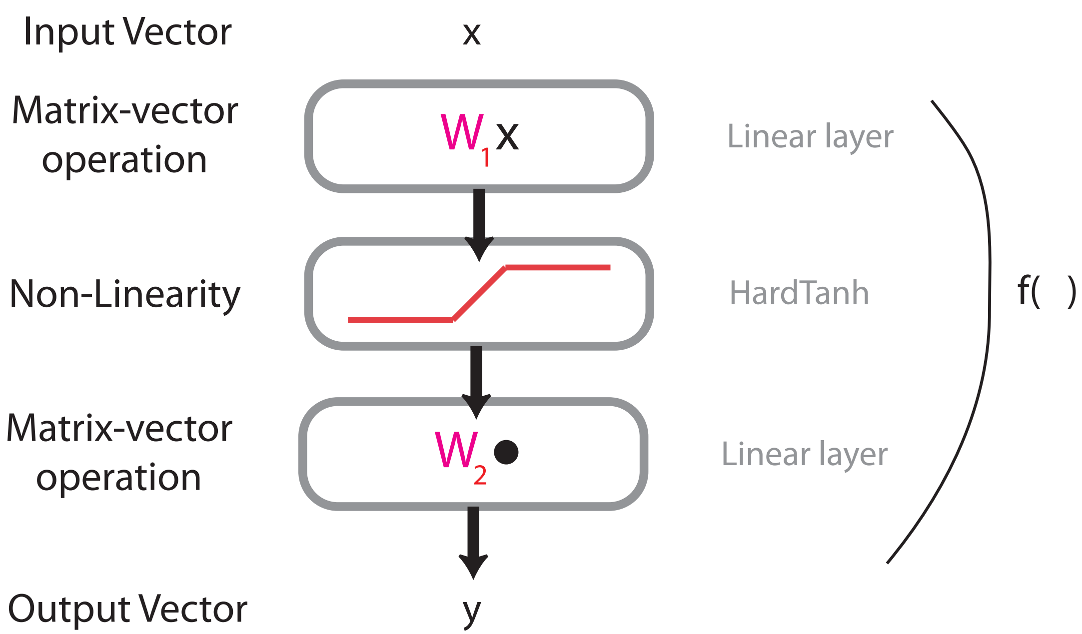
Means

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

The Networks

Neural Networks

- Stack several layers together

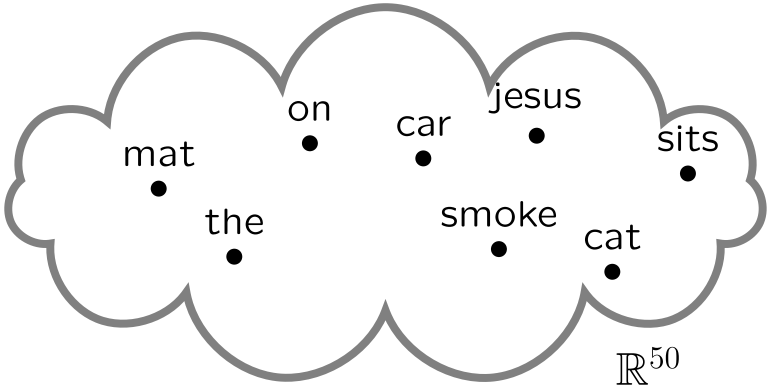


- Increasing level of abstraction at each layer
- Requires simpler features than "shallow" classifiers
- The "weights" W_i are trained by gradient descent
- How can we feed words?

Words into Vectors

Idea

- Words are embed 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)

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

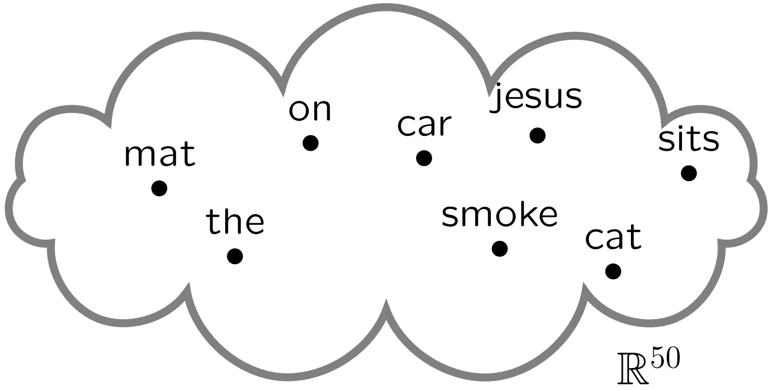
Remarks

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

Words into Vectors

Idea

- Words are embed in a vector space



- Embeddings are trained

Implementation

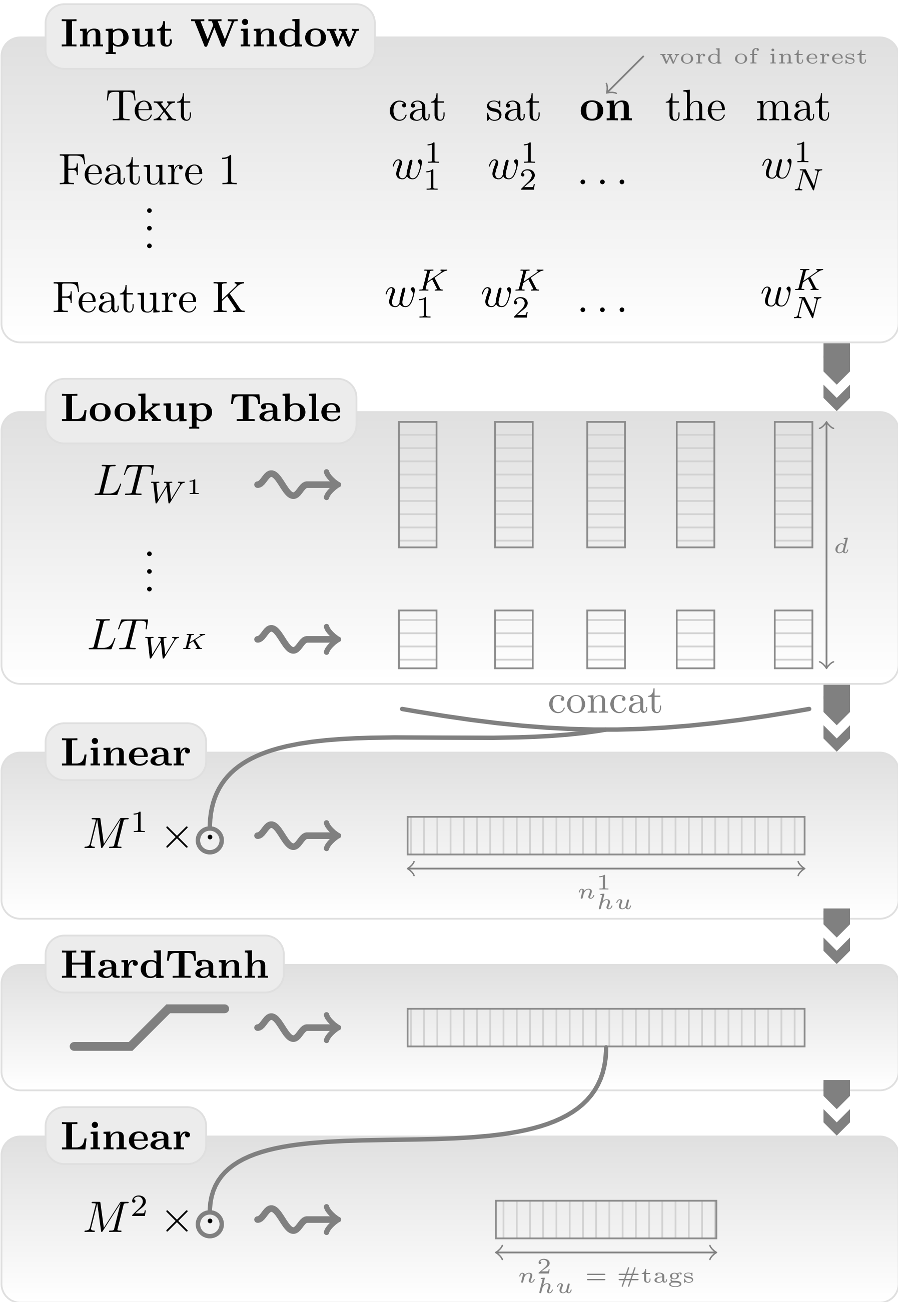
- A word w is an index in a dictionary $\mathcal{D} \in \mathbb{N}$
- Use a lookup-table ($W \sim$ feature size $\times$ dictionary size)

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

Remarks

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

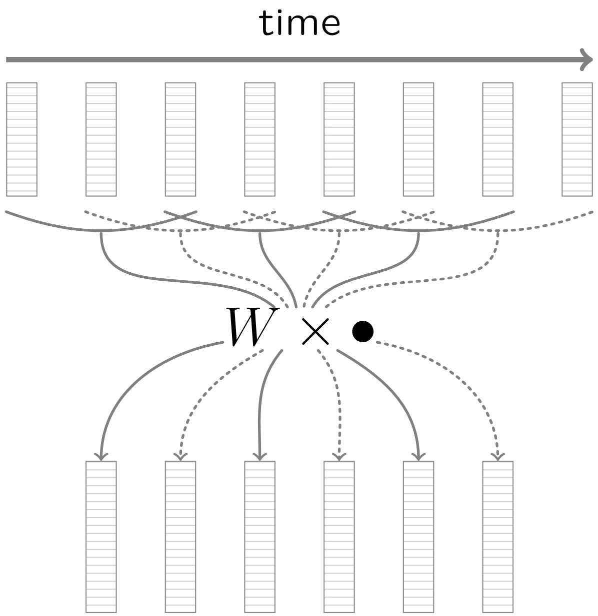
Window Approach



- 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?

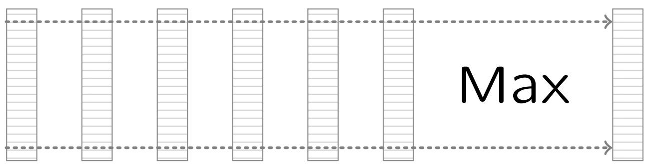
  E.g. in SRL, the verb of interest might be outside the window!

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

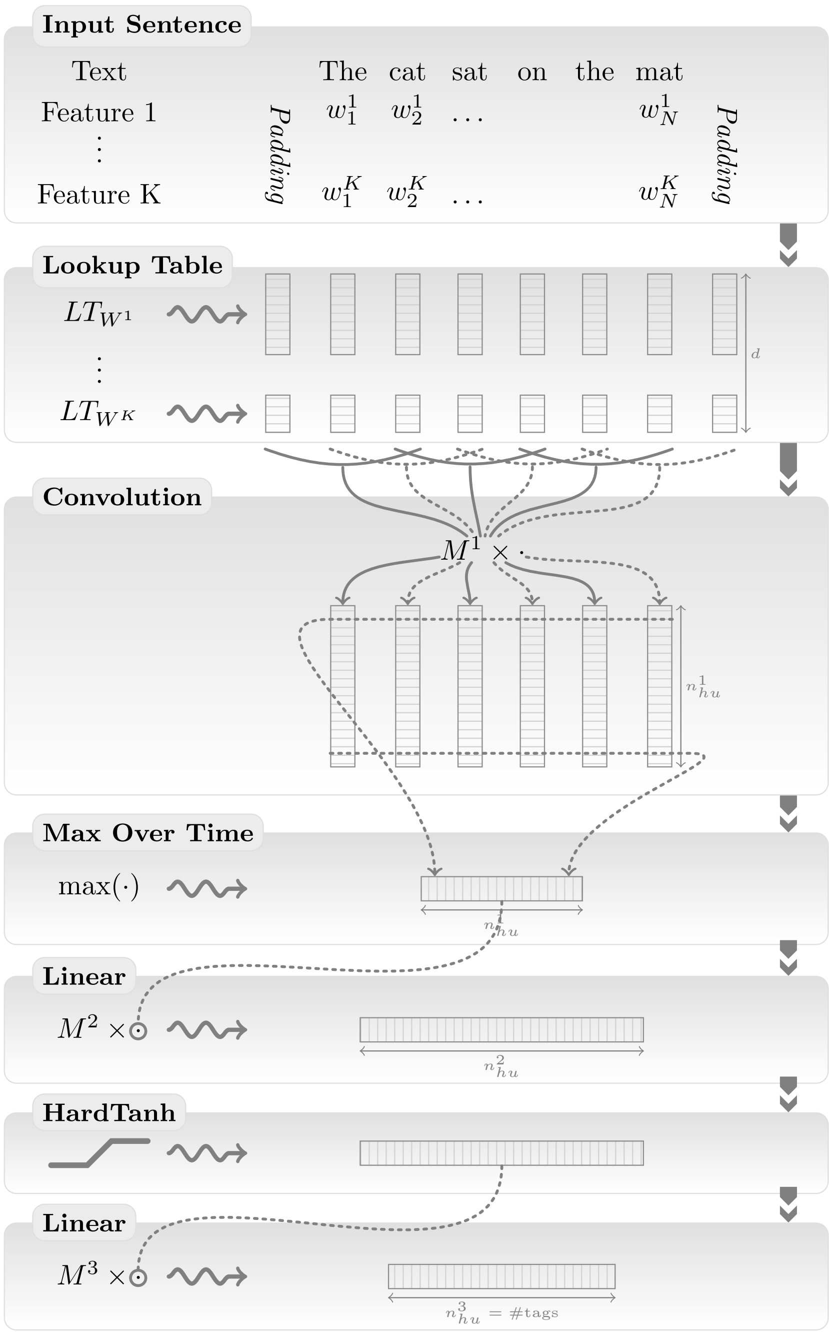


See (Bottou, 1989) or (LeCun, 1989).

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



Outputs a fixed-sized feature vector


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} = #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 ascent (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

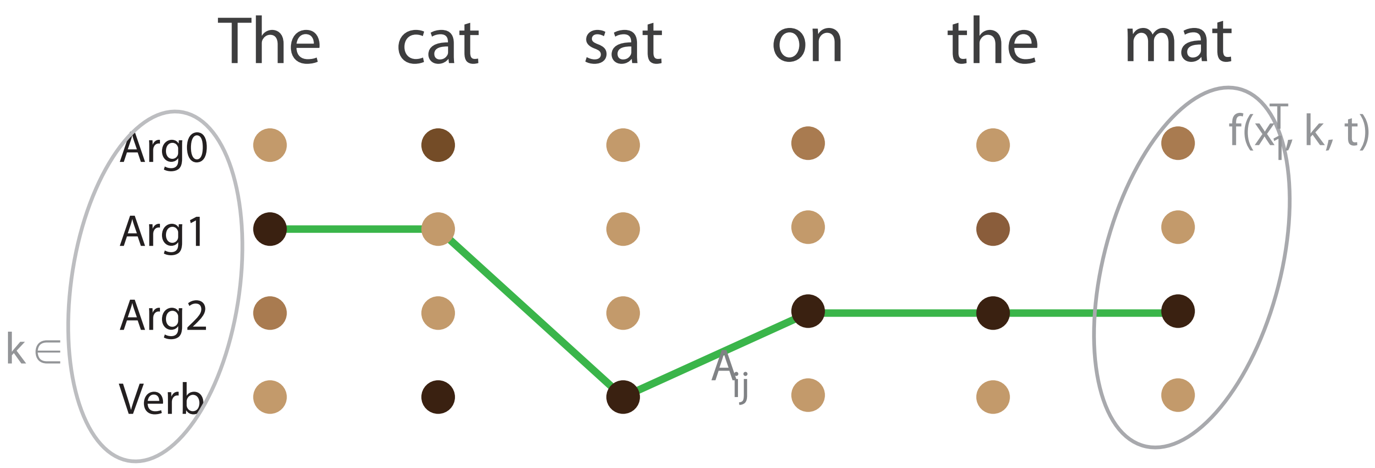
$$\operatorname*{logadd}_i z_i = \log(\sum_i e^{z_i})$$

- 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 $f([\boldsymbol{x}]_1^T, k, t, \boldsymbol{\theta})$
- A_{kl} transition score to jump from tag k to tag l



- Sentence score for a tag path $[i]_1^T$

$$s([\boldsymbol{x}]_1^T, [i]_1^T, \tilde{\boldsymbol{\theta}}) = \sum_{t=1}^{T} \left(A_{[i]_{t-1}[i]_t} + f([\boldsymbol{x}]_1^T, [i]_t, t, \boldsymbol{\theta}) \right)$$

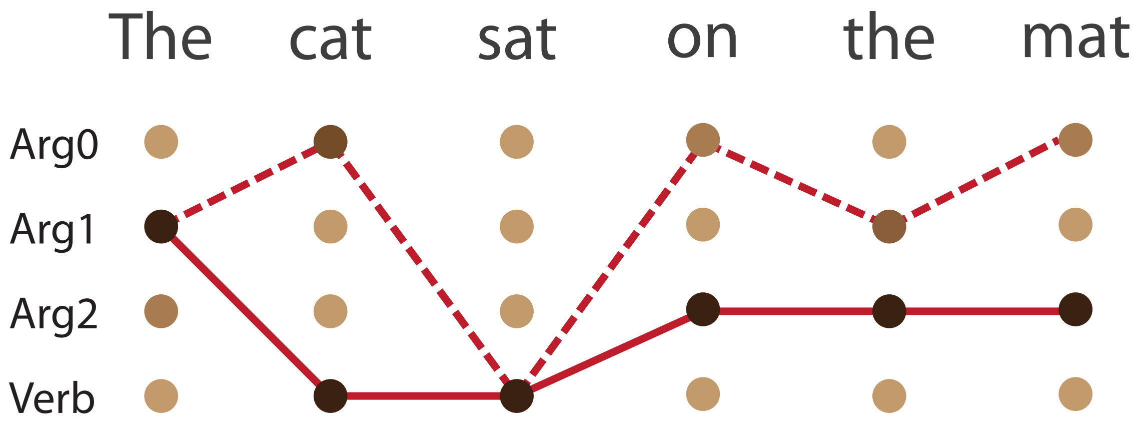
- Conditional likelihood by normalizing w.r.t all possible paths:

$$\log p([y]_1^T \,|\, [\boldsymbol{x}]_1^T, \tilde{\boldsymbol{\theta}}) = s([\boldsymbol{x}]_1^T, [y]_1^T, \tilde{\boldsymbol{\theta}}) - \underset{\forall [j]_1^T}{\text{logadd}}\, s([\boldsymbol{x}]_1^T, [j]_1^T, \tilde{\boldsymbol{\theta}})$$

- How to efficiently compute the normalization?

Sentence Tag Likelihood (STL) (1/2)

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



- Sentence score for a tag path $[i]_1^T$

$$s([\boldsymbol{x}]_1^T, [i]_1^T, \tilde{\boldsymbol{\theta}}) = \sum_{t=1}^{T} \left(A_{[i]_{t-1}[i]_t} + f([\boldsymbol{x}]_1^T, [i]_t, t, \boldsymbol{\theta}) \right)$$

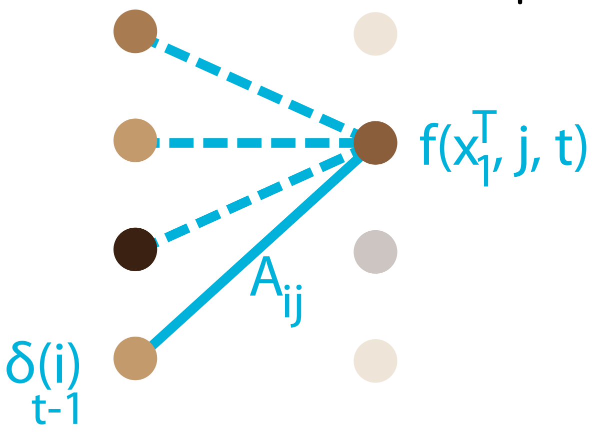
- Conditional likelihood by normalizing w.r.t all possible paths:

$$\log p([y]_1^T \,|\, [\boldsymbol{x}]_1^T, \tilde{\boldsymbol{\theta}}) = s([\boldsymbol{x}]_1^T, [y]_1^T, \tilde{\boldsymbol{\theta}}) - \operatorname*{logadd}_{\forall [j]_1^T} s([\boldsymbol{x}]_1^T, [j]_1^T, \tilde{\boldsymbol{\theta}})$$

- How to efficiently compute the normalization?

Sentence Tag Likelihood (STL) (2/2)

- Normalization computed with recursive Forward algorithm:



$$\delta_t(j) = \text{logAdd}_i \left[\delta_{t-1}(i) + A_{i,j} + f_\theta(j, x_1^T, t)\right]$$

Termination:

$$\underset{\forall [j]_1^T}{\text{logadd}}\, s([\boldsymbol{x}]_1^T,\, [j]_1^T,\, \tilde{\boldsymbol{\theta}}) = \text{logAdd}_i\, \delta_T(i)$$

- Simply backpropagate through this recursion with chain rule

- Non-linear CRFs: Graph Transformer Networks (Bottou, 1997)

- 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!

Lots Of Unlabeled Data

Ranking Language Model

- 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
- Entropy criterion largely determined by 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	*74.15*

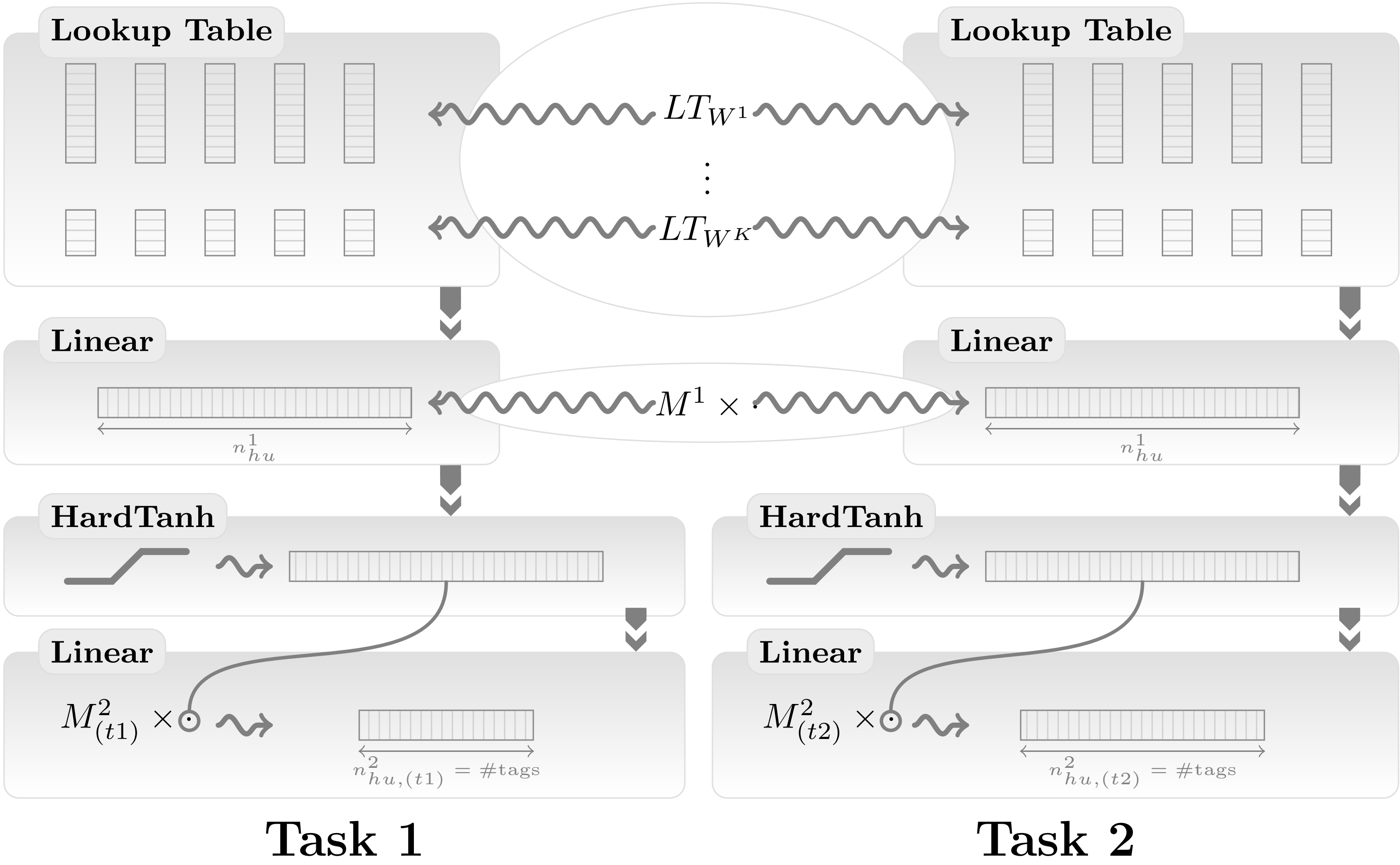
- Huge boost from language models
- Training set word coverage:

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

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

The Temptation

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	*74.15*
NN+STC+LM2+Suffix2	97.29	–	–	–
NN+STC+LM2+Gazetteer	–	–	89.59	–
NN+STC+LM2+POS	–	94.32	88.67	–
NN+STC+LM2+CHUNK	–	–	–	*74.72*

Variance

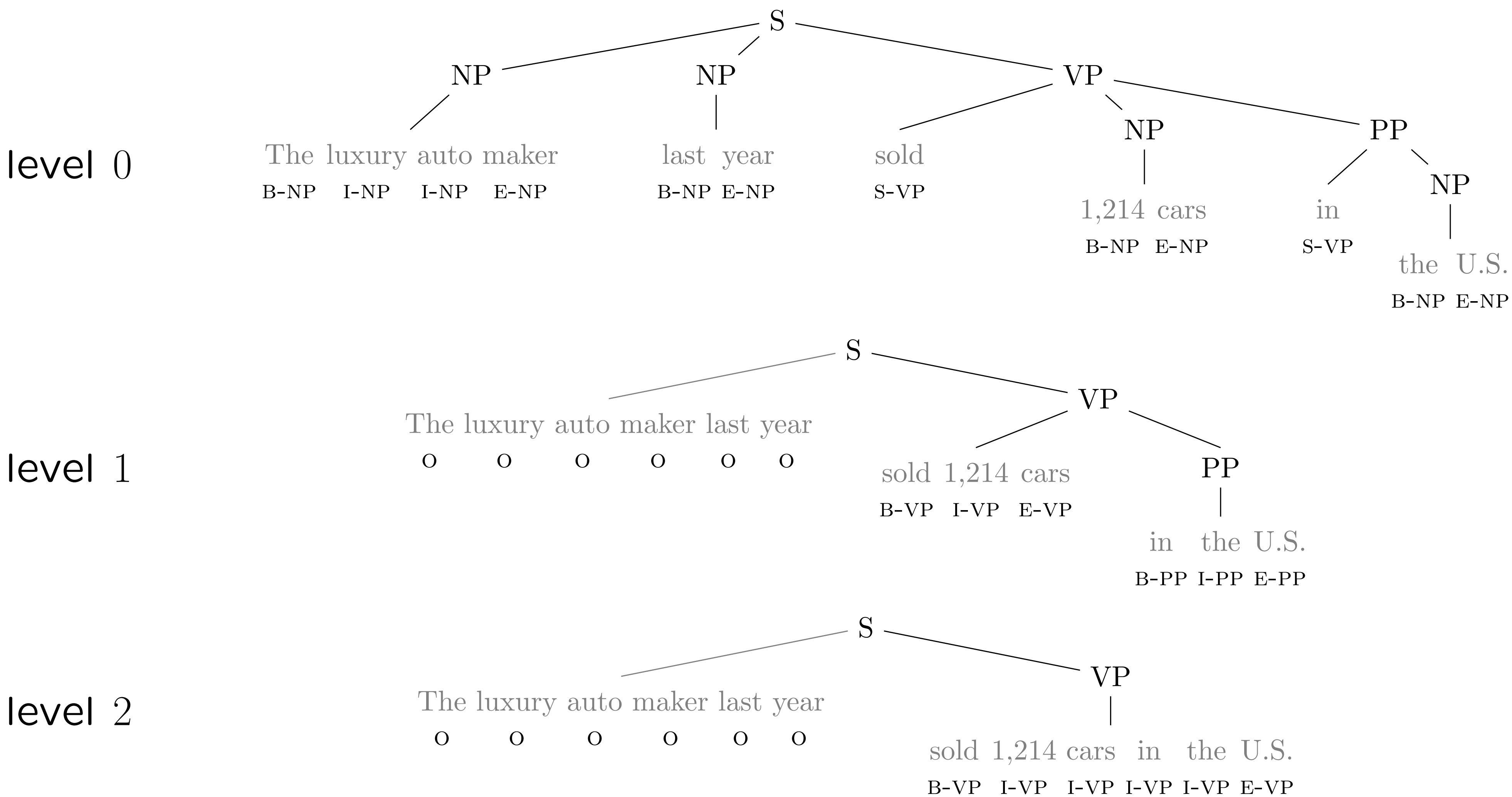
- Train 10 networks

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 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	*74.15*
NN+STC+LM2+CHUNK	*74.72*
NN+STC+LM2+Charniak (level 0 only)	*75.62*
NN+STC+LM2+Charniak (levels 0 & 1)	*75.86*
NN+STC+LM2+Charniak (levels 0 to 2)	*76.03*
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

Achievements

- "All purpose" neural network architecture for NLP tagging
- Limit task-specific engineering
- Rely on very large unlabeled datasets
- We do not plan to stop here

Critics

- 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 (Bottou, 1997)

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

Deep Learning for NLP: Parts 3 & 4

Ronan Collobert
NEC Labs America, Princeton, USA

Jason Weston
Google, New York, USA

Part 3

“Semantic Search”

Learning Hidden Representations for Retrieval

Collaborators: B. Bai, D. Grangier, K. Sadamasa, Y. Qi, C. Cortes, M. Mohri

Document Ranking: Our Goal

We want to learn to match a query (text) to a target (text).

Most supervised ranking methods use hand-coded features.

Methods like LSI that learn from words are unsupervised.

In this work we use supervised learning from text only:

Learn hidden representations of text for learning to rank from words.

Outperforms existing methods (on words) like TFIDF, LSI or a (supervised) margin ranking perceptron baseline.

Basic Bag-o'-words

Bag-of-words + cosine similarity:

- Each doc. $\{d_t\}_{t=1}^{N} \subset \mathbb{R}^{\mathcal{D}}$ is a *normalized* bag-of-words.
- Similarity with query q is: $f(q, d) = q^{\top} d$

Doesn't deal with synonyms: bag vectors can be orthogonal

No machine learning at all

Latent semantic indexing (LSI)

Learn a linear embedding $\phi(d_i) = Ud_i$ via a reconstruction objective.

- Rank with: $f(q,d) \;=\; q^\top U^\top U d \;=\; \phi(q)^\top \phi(d_i)$ [1].

Uses "synonyms": *low-dimensional latent "concepts"*.

Unsupervised machine learning: useful for goal?

---

[1] $f(q,d) = q^\top(U^\top U + \alpha I)d$ gives better results.
Also, usually normalize this $\rightarrow$ cosine similarity.

(Polynomial) Supervised Semantic Indexing (SSI)

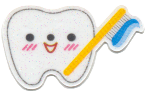
- Define document-query similarity function: $f(q,d) = w^\top \phi^k([q,d])$, where $\Phi^k(x_1,\ldots,x_{\mathcal{D}})$ considers all possible k-degree terms:

$$\Phi^k(x_1,\ldots,x_{\mathcal{D}}) = \langle x_{i_1}\ldots x_{i_k} : 1 \le i_1 \ldots i_k \le \mathcal{D}\rangle.$$

We consider:

- $f^2(q,d) \quad = \quad \sum_{i,j=1}^{\mathcal{D}} W_{ij} q_i d_j = \quad q^\top W d \qquad (1)$

- $f^3(q,d) \quad = \quad \sum_{i,j,k=1}^{\mathcal{D}} W_{ijk} q_i d_j d_k + f^2(q,d). \qquad (2)$



Supervised machine learning: targeted for goal, uses synonyms

Too Big/Slow?!

SSI: why is this a good model?

Classical bag-of-words doesnt work when there are few matching terms:

q=(kitten, vet, nyc)

d=(cat, veterinarian, new, york)

- Our method $q^\top W d$ learns that e.g. kitten and cat are highly related.
- E.g. if i is the index of kitten and j is the index of cat, then $W_{ij} > 0$ after training.

Usefulness of degree 3 model::
Poly degree 2:
Weights for word pairs: e.g. "jagger" $\in q$ & "stones" $\in d$.
Poly degree 3:
Weights for word triples: e.g. "jagger" $\in q$ & "stones" , "gem" $\in d$.

SSI: Why the Basic Model Sucks

- Even for degree 2, W is big : 3.4Gb if $\mathcal{D} = 30000$, 14.5Tb if $\mathcal{D} = 2.5M$.

- Slow: $q^\top W d$ computation has mn computations $q_j W_{ij} d_i$, where q and d have m and n nonzero terms.

- Or one computes $v = q^\top W$ once, and then vd for each document. Classical speed where query has $\mathcal{D}$ terms, assuming W is dense $\rightarrow$ still slow.

SSI Improved model: Low Rank W

For degree 2, Constrain W:

$$W = U^\top V + I.$$

- U and V are $N \times \mathcal{D}$ matrices → smaller
- Low dimensional "latent concept" space like LSI (same speed).
- Differences: supervised, asymmetric, learns with I.

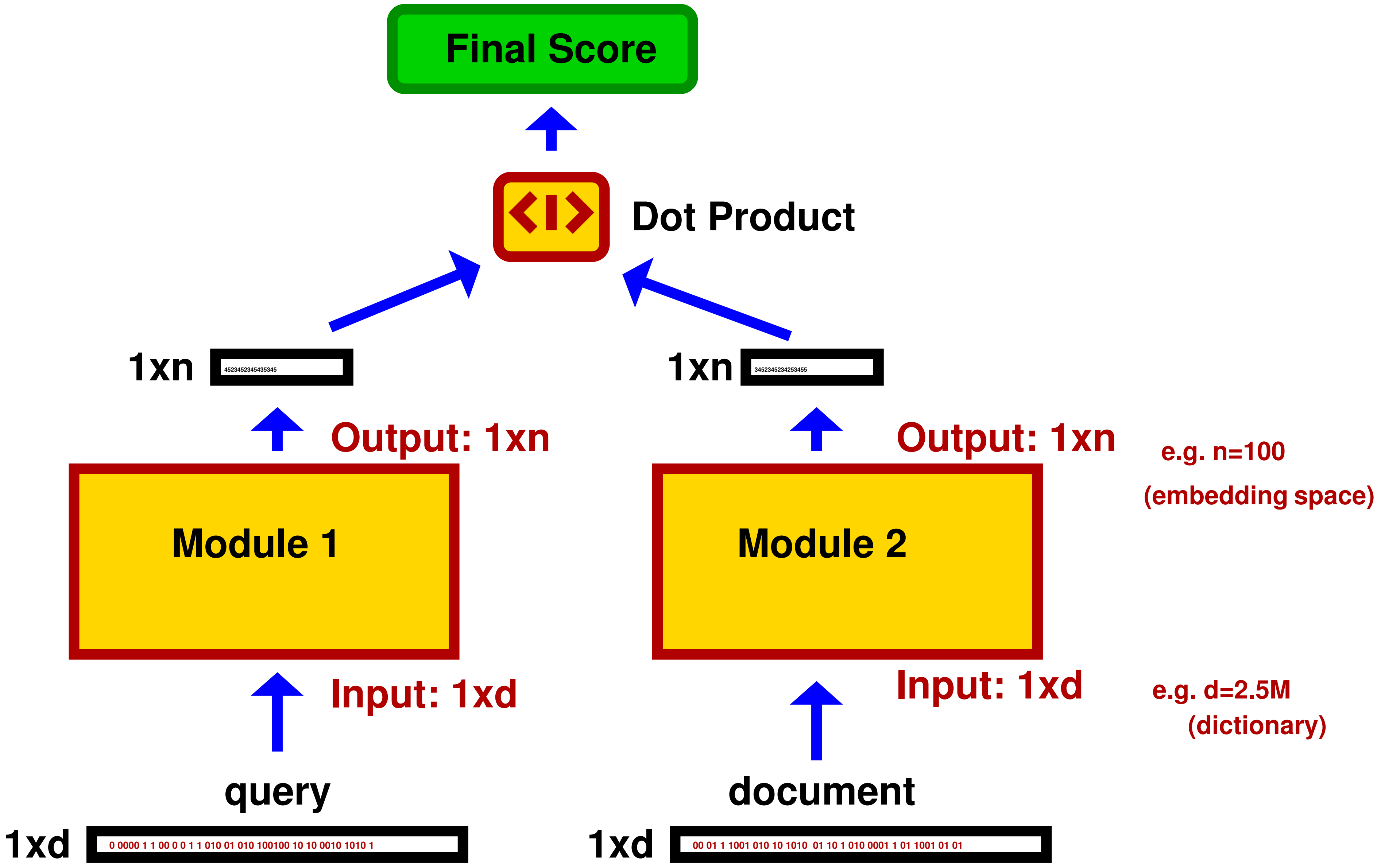
• For $k = 2$, replace W with $\overline{W} = (U^\top V) + I$:

$$f_{LR}^2(q,d) \;=\; q^\top (U^\top V + I) d, \;=\; \sum_{i=1}^N (Uq)_i (Vd)_i + q^\top d.$$

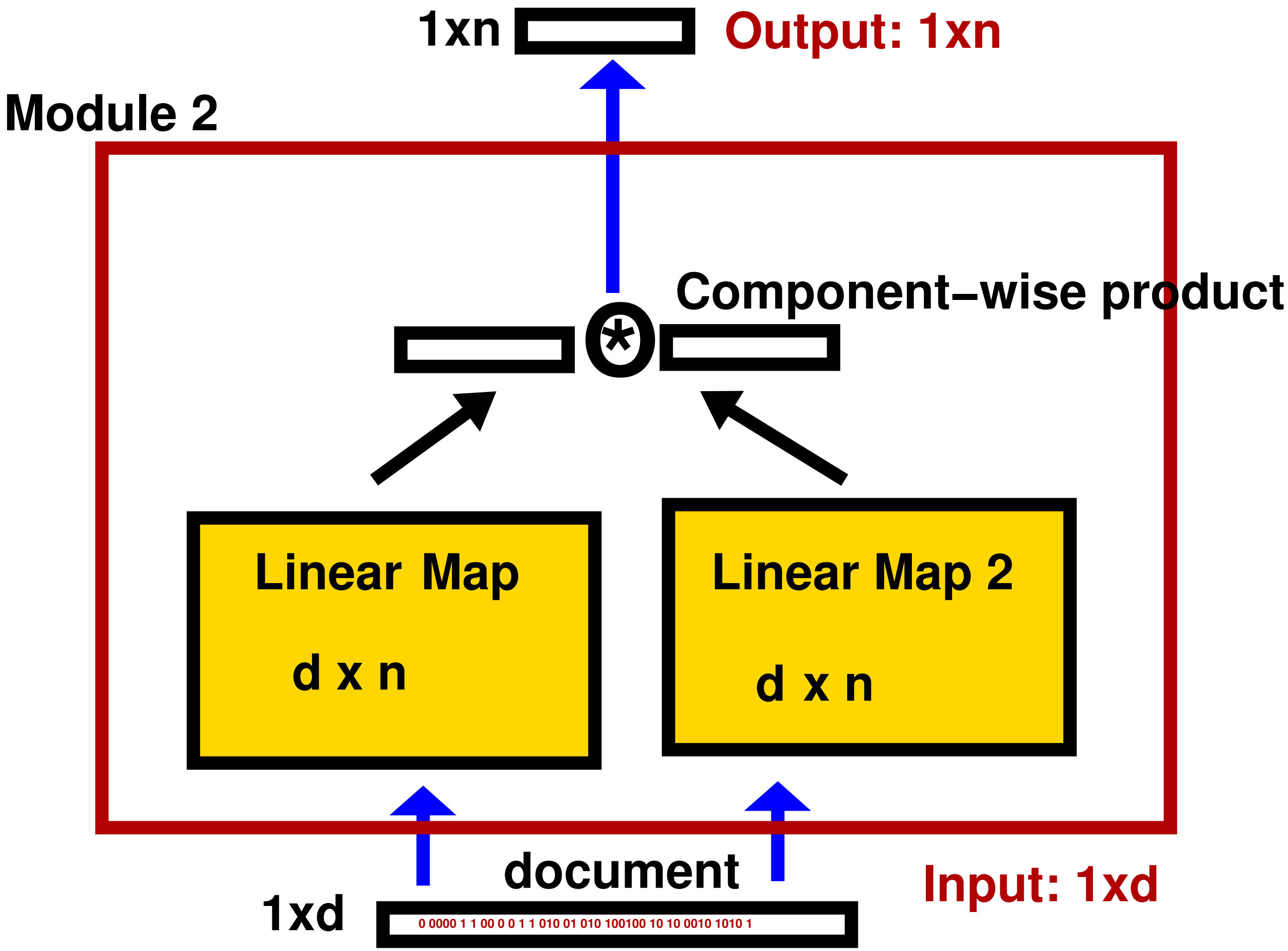
• For $k = 3$, approximate W_{ijk} with $\overline{W}_{ijk} = \sum_l U_{li} V_{lj} Y_{lk}$:

$$f_{LR}^3(q,d) = \sum_{i=1}^N (Uq)_i (Vd)_i (Yd)_i + f_{LR}^2(q,d).$$

Neural Network Models for Retrieval



Doc. Embedding for Polynomial Degree 3



SSI: Training

Training Loss

- Ranking loss from preference triplets (q, d^+, d^-), "*for query q, d^+ should appear above d^-*":
- $L(W;\mathcal{R}) = \sum_{(q,d^+,d^-)\in\mathcal{R}} \max(0, 1 - f_W(q,d^+) + f_W(q,d^-))$

Learning Algorithm Stochastic Gradient Descent: **Fast & scalable**.

Iterate	Sample a triplet (q, d^+, d^-), Update $W \leftarrow W - \lambda\frac{\partial}{\partial W}\max(0, 1 - f_W(q,d^+) + f_W(q,d^-))$.

Prior Work: Summary of learning to Rank

- SVM [Joachims, 2002] and NN ranking methods [Burges, 2005] .
  Use hand-coded features: title, body, URL, search rankings,. . . (don't use words)
  (e.g. Burges uses 569 features in all).

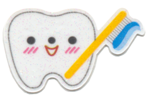
- In contrast we use only the words and try to find their hidden representation.

- Several works on optimizing different loss functions (MAP, ROC, NDCG): [Cao, 2008], [Yu, 2007], [Qin, 2006],. . . .

- [Grangier & Bengio, '06] used similar methods to basic SSI for retrieving images.

- [Goel, Langord & Strehl, '08] used Hash Kernels (Vowpal Wabbit) for advert placement.

- Main difference: i) we use low rank & ii) polynomial degree 3 features.



We could also add features + new loss to our method ..

Experimental Comparison

- **Wikipedia**
  - 1,828,645 documents. 24,667,286 links.
  - Split into 70% train, 30% test.
- Pick random doc. as query, then rank other docs.
- Docs that are linked to it should be highly ranked.
- **Two setups:**

(i) whole document is used as query;

(ii) 5,10 or 20 words are picked to mimic keyword search.

Experiments: Doc-Doc Ranking

$\mathcal{D} = 30000$

Algorithm	Params	Rank-Loss	MAP	P10
TFIDF	0	1.62%	0.342±0.01	0.170±0.007
Query Expansion	2	1.62%	0.330	0.160
LSI	$200\mathcal{D}$	4.79%	0.161	0.101
αLSI + $(1-\alpha)$TFIDF	$200\mathcal{D}+1$	1.28%	0.346	0.170
Marg. Rank Perceptron	$\mathcal{D}^2$	0.41%	0.477	0.212
SSI: poly $(k = 2)$	$400\mathcal{D}$	**0.30%**	**0.517**	**0.229**
SSI: poly $(k = 3)$	$600\mathcal{D}$	**0.14%**	**0.539**	**0.236**

NOTE:Best possible P10= 0.31 – on average every query has only about 3 links.

Experiments: Doc-Doc Ranking

$\mathcal{D} = 2.5M$

Algorithm	Rank-Loss	MAP	P10
TFIDF	0.842%	0.432±0.012	0.193
Query Expansion	0.842%	0.432	0.1933
αLSI + $(1-\alpha)$TFIDF	0.721%	0.433	0.193
Hash Kernels + αI	0.322%	0.492	0.215
SSI: poly $(k = 2)$	0.158%	0.547±0.012	0.239±0.008
SSI: poly $(k = 3)$	**0.099%**	**0.590±0.012**	**0.249±0.008**

Experiments: Query-Document Ranking

k-keywords based retrieval ($\mathcal{D} = 30000$):

Algorithm	Params	Rank ($k=5$)	MAP ($k=5$)	P@10 ($k=5$)
TFIDF	0	21.6%	0.047	0.023
αLSI + $(1-\alpha)$TFIDF	200$\mathcal{D}$+1	14.2%	0.049	0.023
SSI: poly $(k=2)$	400$\mathcal{D}$	**4.37%**	**0.166**	**0.083**

Algorithm	Params	Rank ($k=10$)	MAP ($k=10$)	P@10 ($k=10$)
TFIDF	0	14.0%	0.083	0.035
αLSI + $(1-\alpha)$TFIDF	200$\mathcal{D}$+1	9.73%	0.089	0.037
SSI: poly $(k=2)$	400$\mathcal{D}$	**2.91%**	**0.229**	**0.100**

Algorithm	Params	Rank ($k=20$)	MAP ($k=20$)	P@10 ($k=20$)
TFIDF	0	9.14%	0.128	0.054
αLSI + $(1-\alpha)$TFIDF	200$\mathcal{D}$+1	6.36%	0.133	0.059
SSI: poly $(k=2)$	400$\mathcal{D}$	**1.80%**	**0.302**	**0.130**

Experiments: Cross-Language Retrieval

Query: in Japanese Target Doc: in English – *use links from Wikipedia as before.*

Algorithm	Rank-Loss	MAP	P10
TFIDF_{EngEng}(Google translated queries)	4.78%	0.319±0.009	0.259±0.008
$\alpha\text{LSI}_{EngEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	3.71%	0.300±0.008	0.253±0.008
$\alpha\text{CL-LSI}_{JapEng}+(1-\alpha)\text{TFIDF}_{EngEng}$	3.31%	0.275±0.009	0.212±0.008
SSI_{EngEng} (Google Translated)	1.72%	0.399±0.009	0.325±0.009
SSI_{JapEng}	0.96%	0.438±0.009	0.351±0.009

What's Inside W?

We can look at the matrix W we learn and see the synonyms it learns (large values of Wij):

kitten	cat	cats	animals	species	dogs
vet	veterinarian	veterinary	medicine	animals	animal
ibm	computer	company	technology	software	data
nyc	york	new	manhattan	city	brooklyn
c++	programming	windows	mac	unix	linux
xbox	console	game	games	microsoft	windows
beatles	mccartney	lennon	song	band	harrison
britney	spears	album	music	pop	her

Summary

- Powerful: supervised method for document ranking.
- Efficient low-rank models $\longrightarrow$ learn hidden representations.
- Nonlinearities improve accuracy.

Part 4

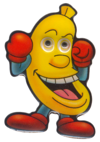
Situated Learning: Hidden Representations for Grounding Language

The Concept Labeling Task

Collaborators: Antoine Bordes, Nicolas Usunier

Connecting NLP with a world: Why?

- Existing NLP: Much (not all) solves syntactic or semantic sub-tasks:
  E.g. POS, chunking, parsing, SRL, MT, summarization . . .
  They don't use "situated" learning.



We understand language because it has a deep connection to the world it is used in/for → *strong prior knowledge*

"John saw Bill in the park with his telescope."
"He passed the exam."
"John went to the bank."

World knowledge we might already have:
Bill owns a telescope.
Fred took an exam last week.
John is in the countryside (not the city).

How can a computer do that?

Learning Speech in a Situated Environment?

The Learning Signal : text adventure game

Planet Earth = tricky:
vision, speech, motor control + *language understanding.*

Multi-user game (e.g. on the internet) = easier.
Simplest version = text adventure game. Good test-bed for ML?

Represent atomic actions as concepts (`get`, `move`, `give`, `shoot`, `...`).
Represent physical objects as concepts (`character1`, `key1`, `key2`, `...`).

(Can consider this signal as a pre-processed version of a visual signal.)

The Concept Labeling Task

Definition:

Map any natural language sentence $x \in \mathcal{X}$ to its labeling in terms of *concepts* $y \in \mathcal{Y}$, where y is a sequence of concepts.

One is given training data triples $\{\mathbf{x}_i, \mathbf{y}_i, \mathbf{u}_i\}_{i=1,\ldots,m} \in \mathcal{X} \times \mathcal{Y} \times \mathcal{U}$ where $\mathbf{u}_i$ is the current state the world.

Universe = set of concepts and their relations to other concepts, $\mathcal{U} = (\mathcal{C}, \mathcal{R}_1, \ldots, \mathcal{R}_n)$, where n is the number of types of relation and $\mathcal{R}_i \subset \mathcal{C}^2$, $\forall i = 1, \ldots, n$.

$\rightarrow$ **Learning** to perform this task is appropriate because:

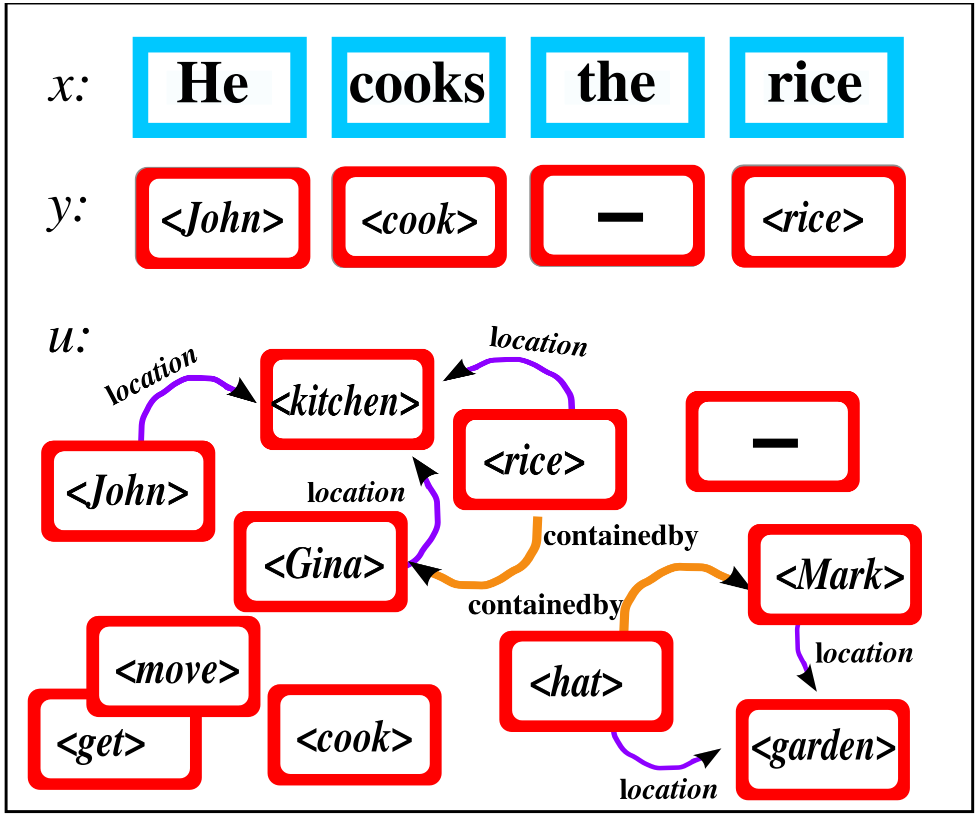
- possibly very *complex rules* to learn,
- rule-based systems might scale badly with large problems,
- *flexibility* from one domain to another.

Example of Concept Labeling

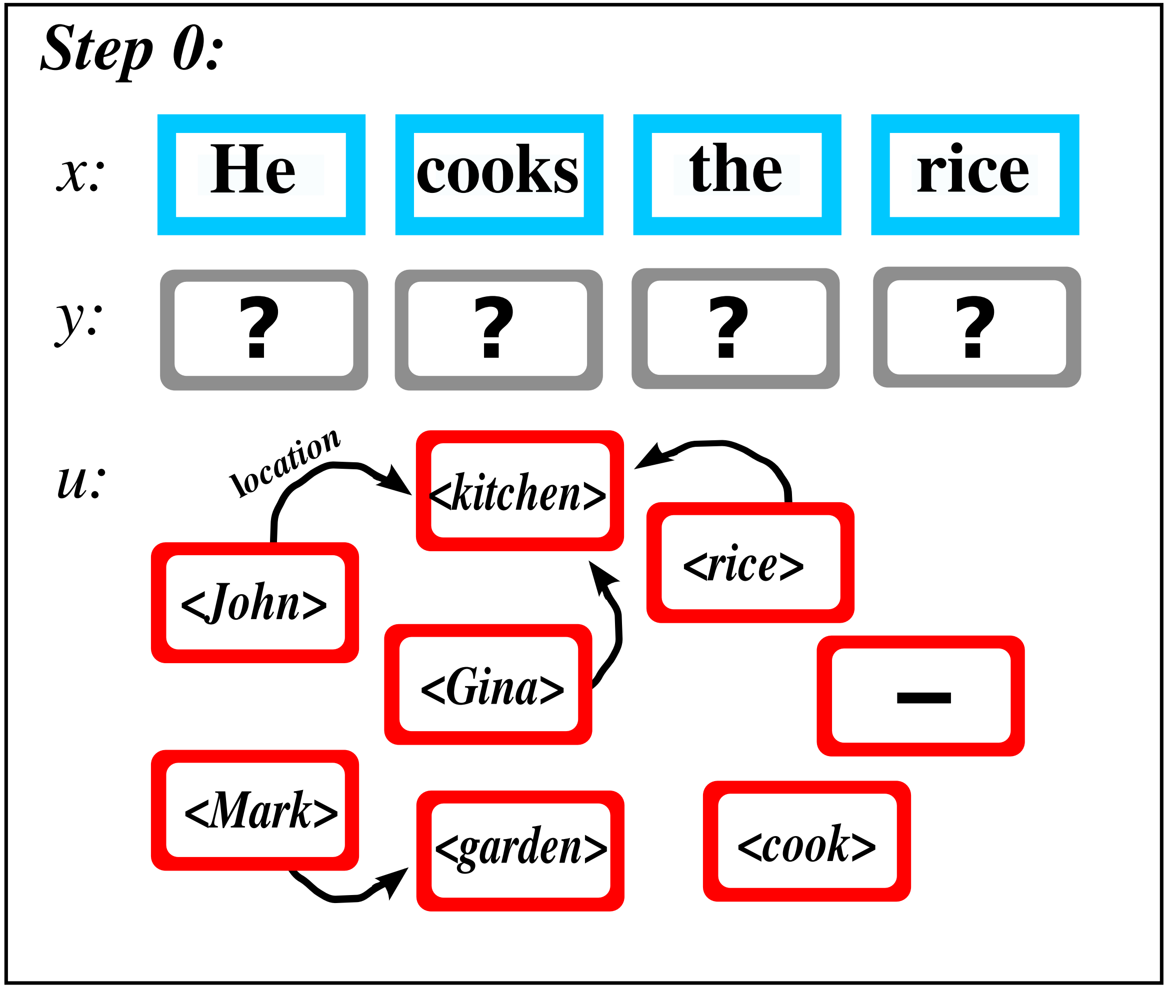
Define two relations:

- *location*$(c) = c'$ with $c, c' \in \mathcal{C}$,
- *containedby*$(c) = c'$ with $c, c' \in \mathcal{C}$.

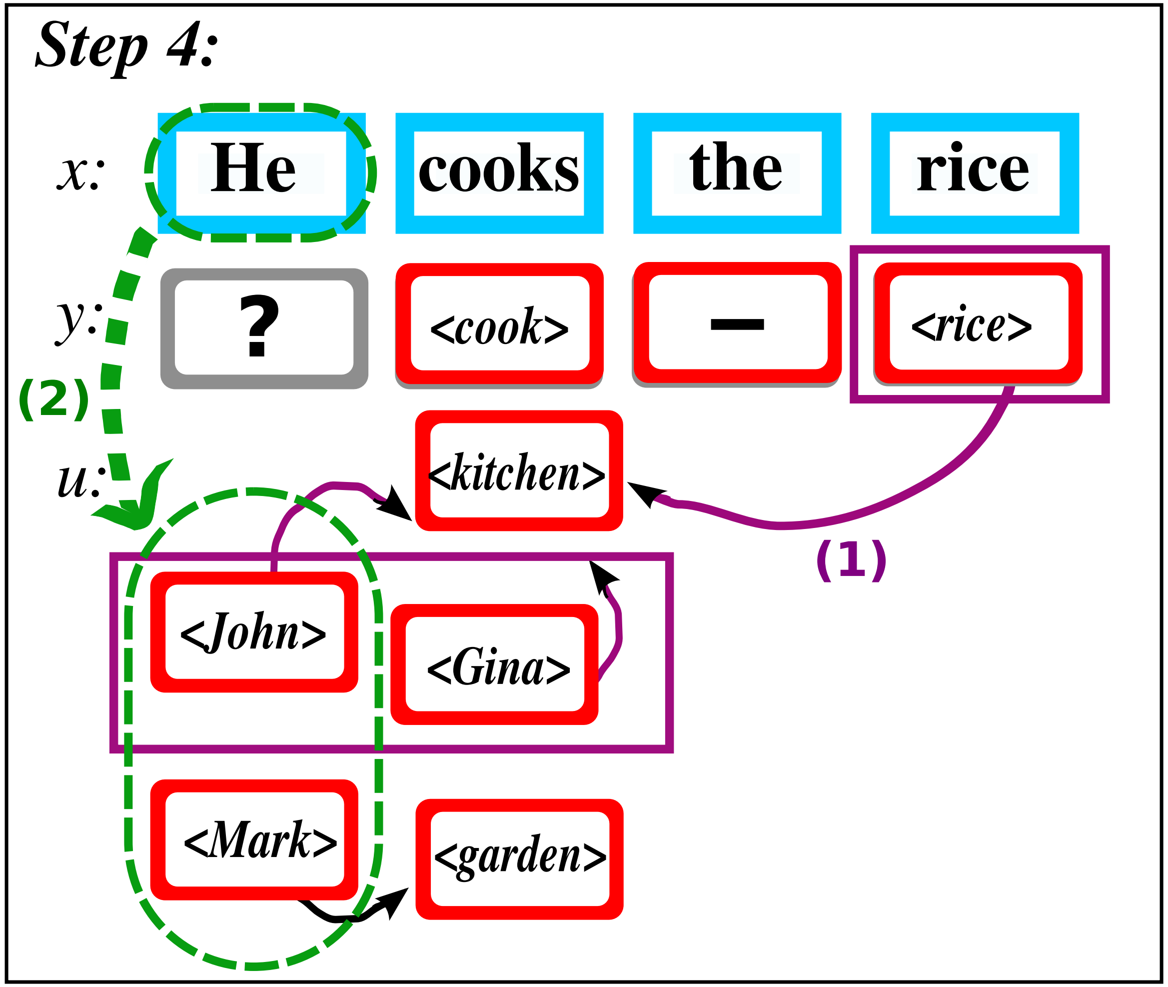
A training triple $(\mathbf{x}, \mathbf{y}, \mathbf{u}) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{U}$:



Disambiguation Example



Disambiguation Example



Label "He" requires two rules which are never explicitly given.

Ambiguities we will handle

He picked up the hat **there**.

The **milk** on the table.

The **one** on the table.

She left the kitchen.

The adult left the kitchen.

Mark drinks the **orange**.

...

(e.g. for sentence (2) there may be several milk cartons that exist...)

Concept Labeling Is Challenging

- Solving ambiguities requires to use rules based on linguistic information and available universe knowledge.

- **But**, these rules are never made explicit in training.

$\rightarrow$ A concept labeling algorithm has to learn them.

- No engineered features for describing words/concepts are given.

$\rightarrow$ A concept labeling algorithm has to discover them from raw data.

Learning Algorithm : Basic Argmax

We could do this:

$$y = f(x, u) = \text{argmax}_{y'} \;\; g(x, y', u),$$

$g(\cdot)$ should be large if concepts y' are consistent with both the sentence x and the current state of the universe u.

However. . . could be slow.

Simulation : algorithm

Model a world + Generate training data for our learning task:

1. **Generate a new event**, $(v, a) = \mathit{event}(u)$.

   – Generates verb+ set of args – a *coherent* action given the universe.

   E.g. actors change location and pick up, exchange & drop objects...

2. **Generate a training triple**, i.e. (x,y)=*generate*(v, a).

   – Returns a sentence and concept labeling pair given a verb + args.

   This sentence should describe the event.

3. **Update the universe**, i.e. $u := \mathit{exec}(v)(a, u)$.

Labeled Data generated by the Simulation

Simulation of a house with 58 concepts: 15 verbs, 10 actors, 15 small objects, 6 rooms and 12 pieces of furniture. . .

. . .

x:	the father gets some yoghurt from the sideboard
y:	- *<father>* *<get>* - *<yoghurt>* - - *<sideboard>*
x:	he sits on the chair
y:	*<brother>* *<sit>* - - *<chair>*
x:	she goes from the bedroom to the kitchen
y:	*<mother>* *<move>* - - *<bedroom>* - - *<kitchen>*
x:	the brother gives the toy to her
y:	- *<brother>* *<give>* - *<toy>* - *<sister>*

. . .

$\rightarrow$ Generate a dataset of 50,000 training triples and 20,000 testing triples ($\approx$55% ambiguous), without any human annotation.

Experimental Results using an SVM

Method	Features	Train Err	Test Err
SVM$_{struct}$	x	42.26%	42.61%
SVM$_{struct}$	$x + u$ (loc, contain)	18.68%	23.57%

- No feature engineering: used raw words (and concept relations) as input.

$\rightarrow$ Using world knowledge leads to better generalization.

- Can we learn a hidden representation and do better?

Neural Network Scoring Function

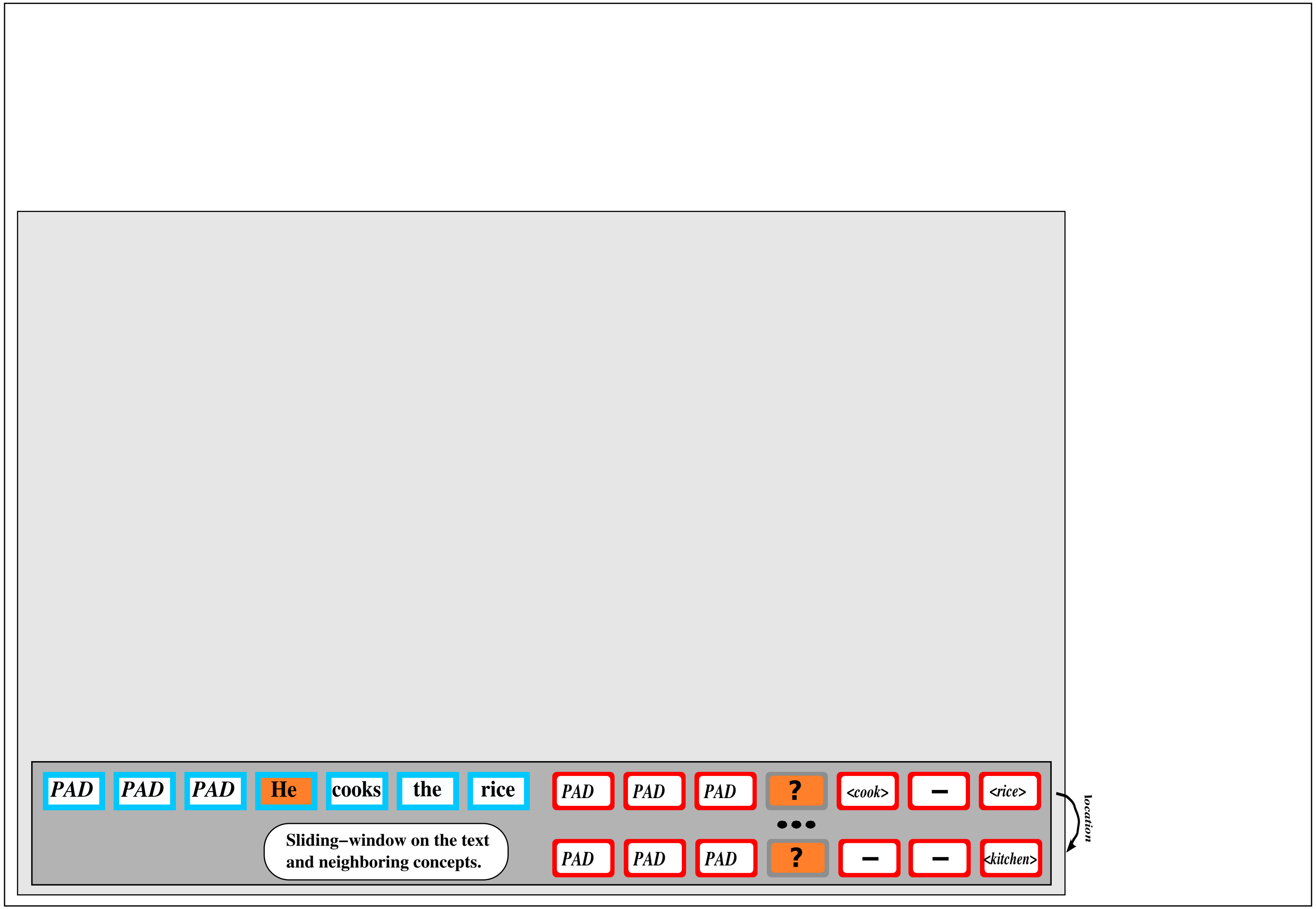
Our score combines two functions $g_i(\cdot)$ and $h(\cdot) \in \mathbb{R}^N$ which are neural networks.

$$g(x, y, u) = \sum_{i=1}^{|x|} g_i(x, y_{-i}, u)^\top h(y_i, u)$$

- $g_i(x, y_{-i}, u)$ is a sliding-window on the text *and* neighboring concepts centered around i^{th} word $\rightarrow$ embeds to N dim-space.
- $h(y_i, u)$ embeds the i^{th} concept to N dim-space.
- Dot-product: confidence that i^{th} word labeled with concept y_i.

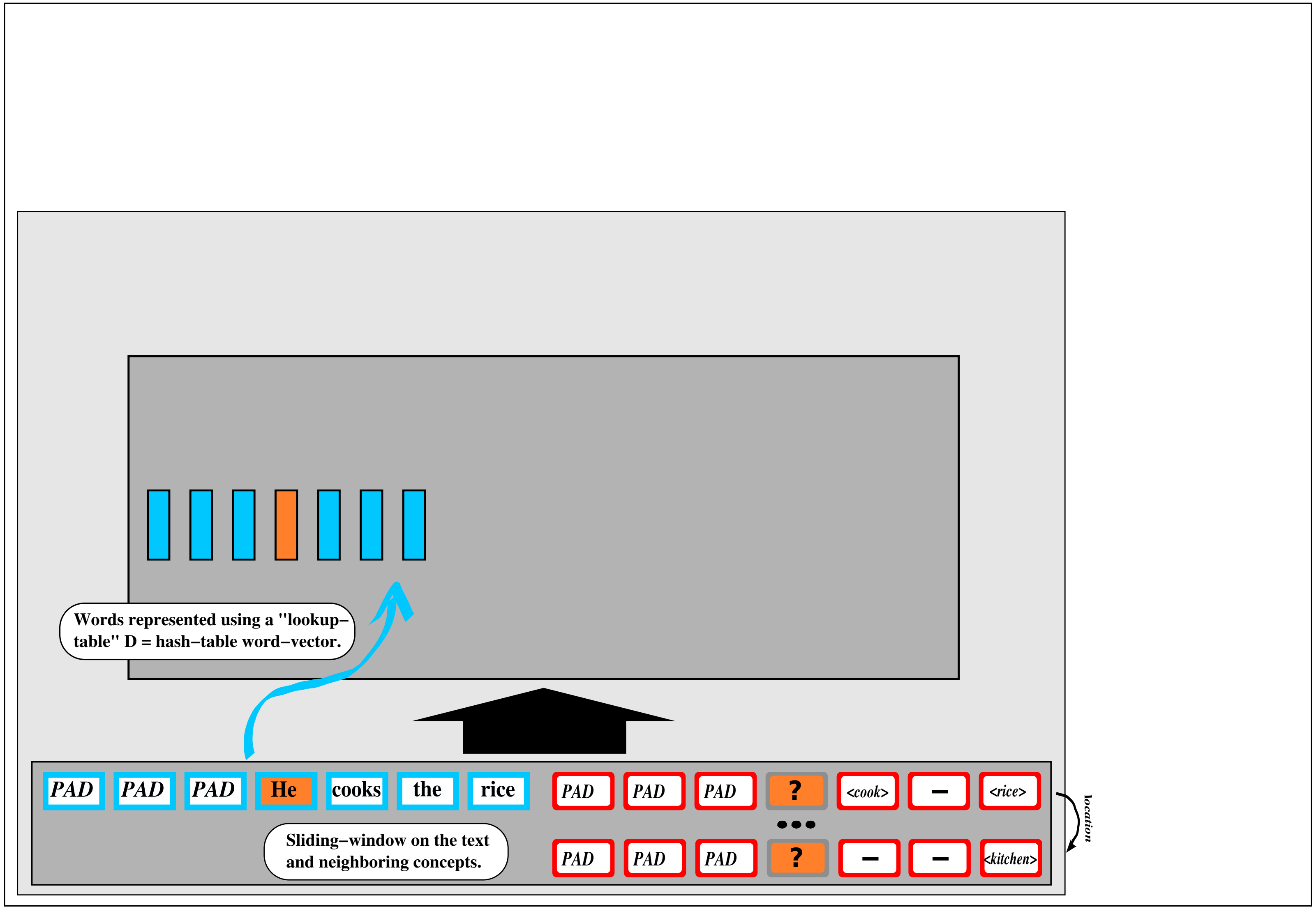
Scoring Illustration

Step 0: Set the sliding-window around the 1^{st} word.



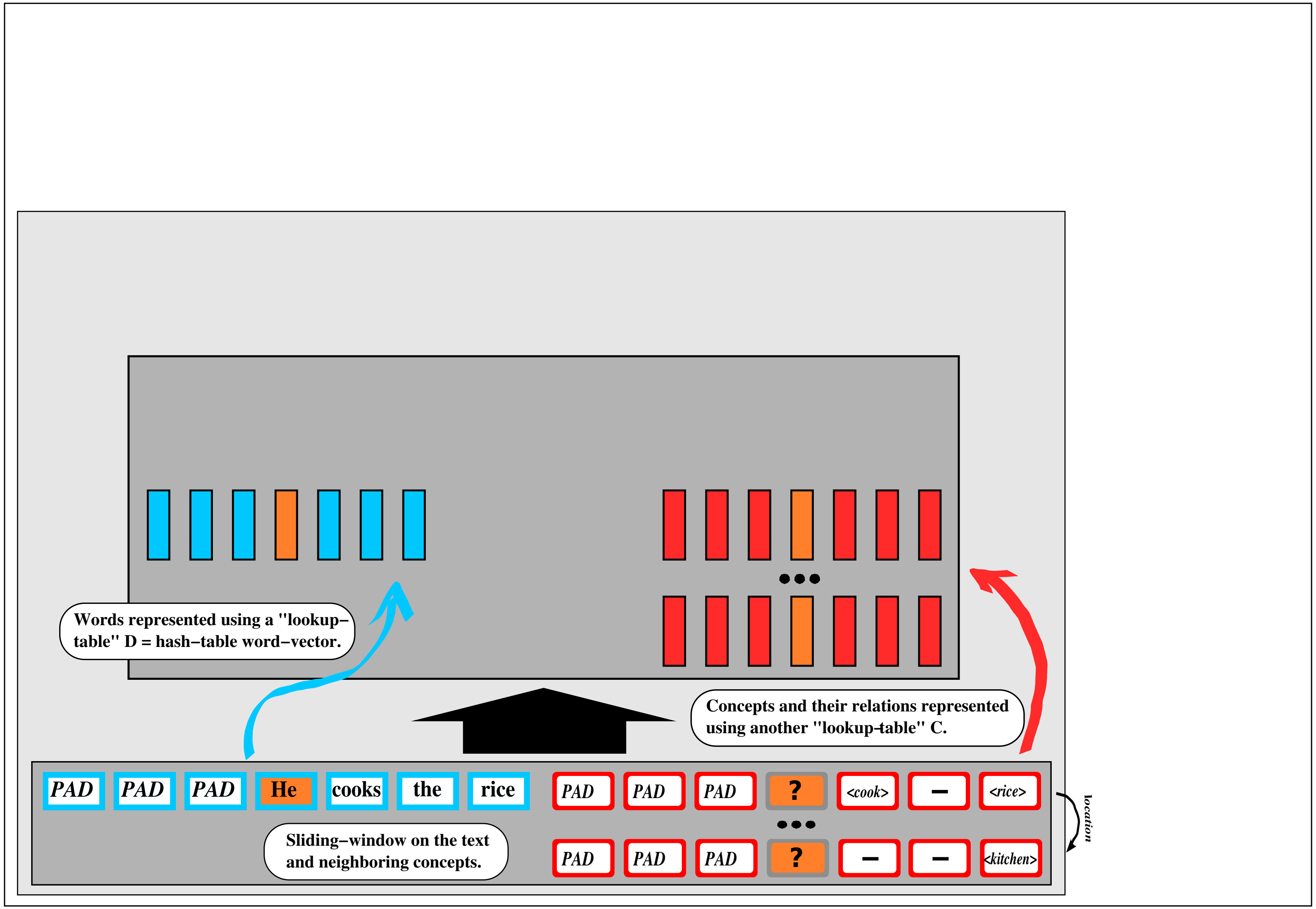
Scoring Illustration

Step 1: Retrieve words representations from the “lookup table”.



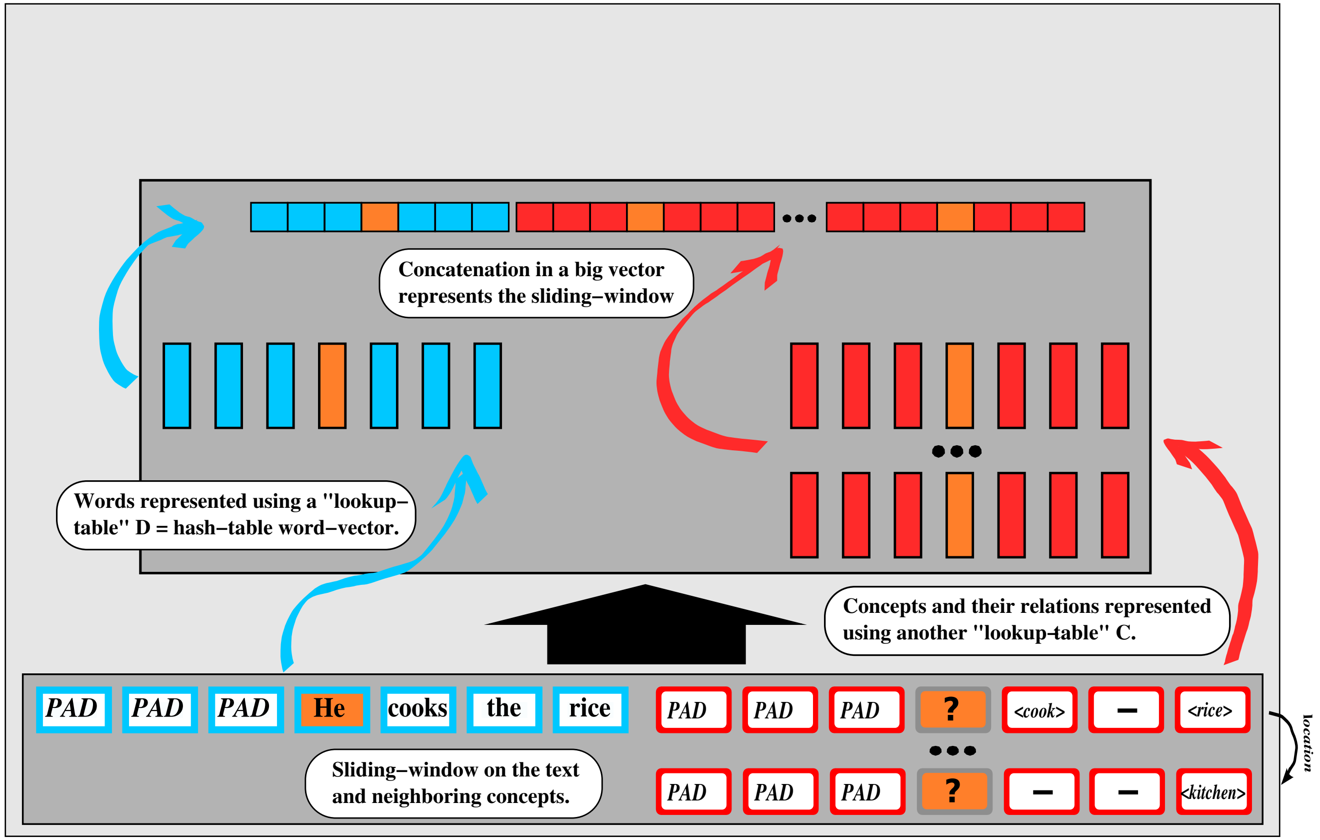
Scoring Illustration

Step 2: Similarly retrieve concepts representations.



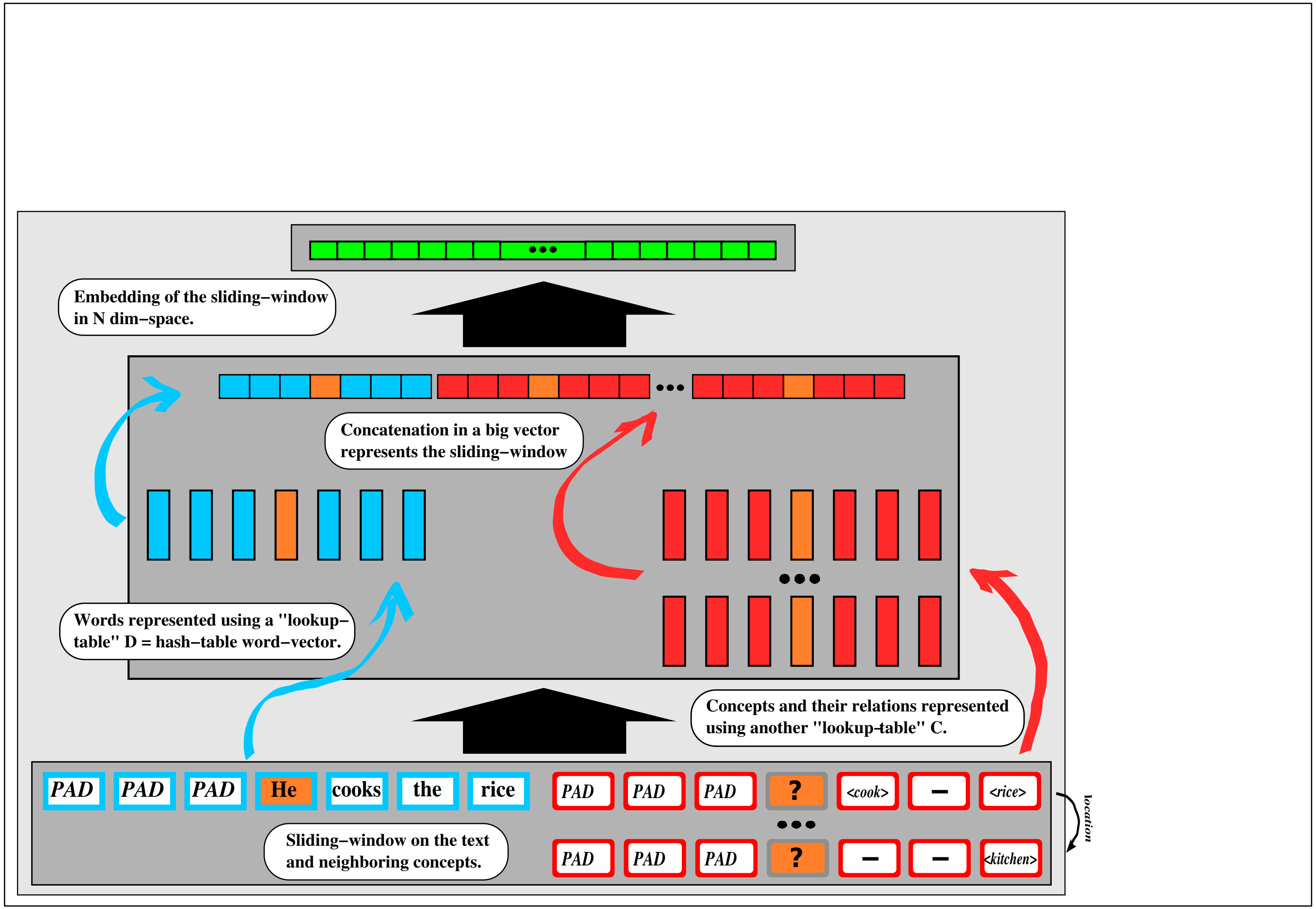
Scoring Illustration

Step 3: Concatenate vectors to obtain window representation.



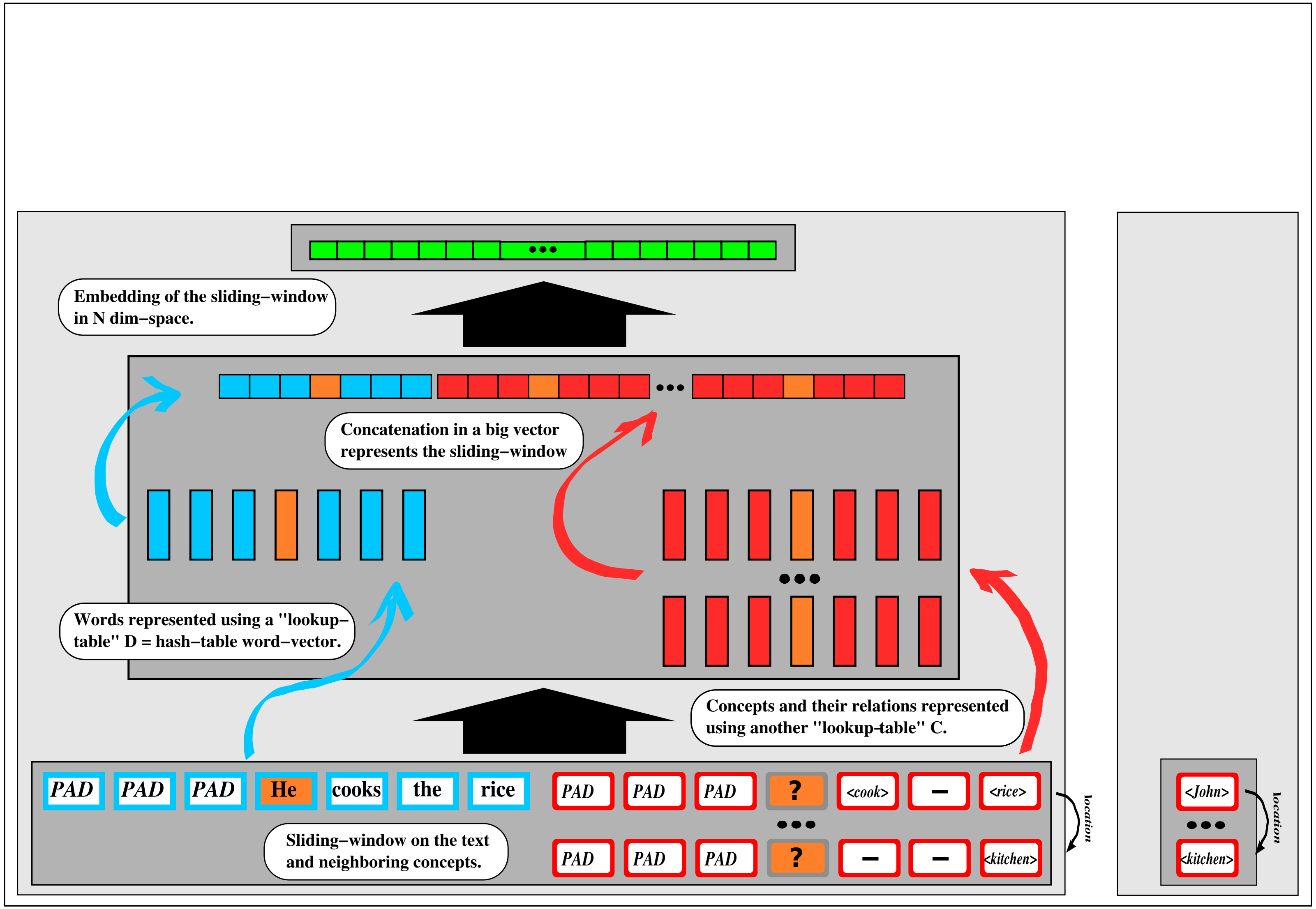
Scoring Illustration

Step 4: Compute $g_1(x, y_{-1}, u)$.



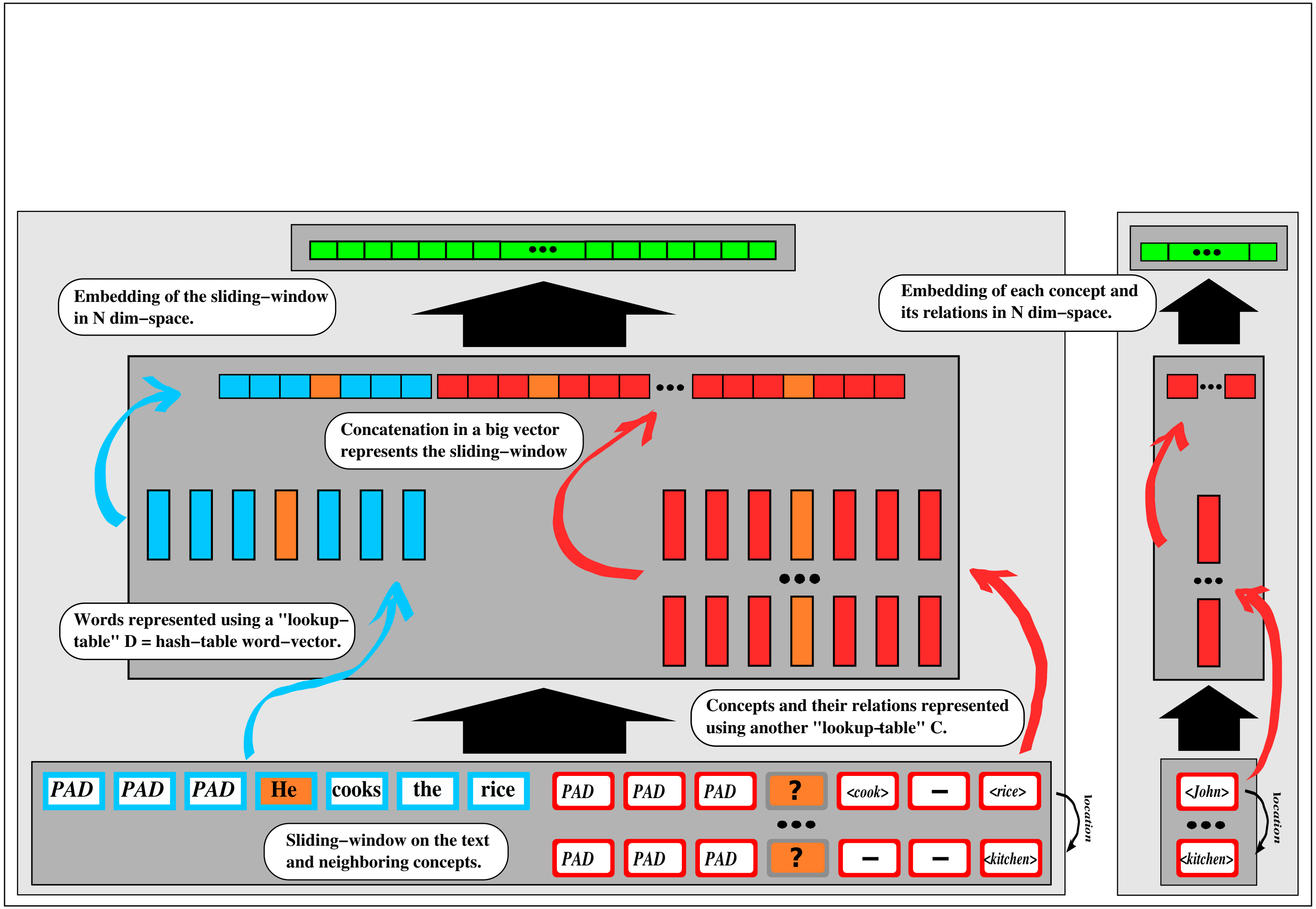
Scoring Illustration

Step 5: Get the concept <*John*> and its relations.



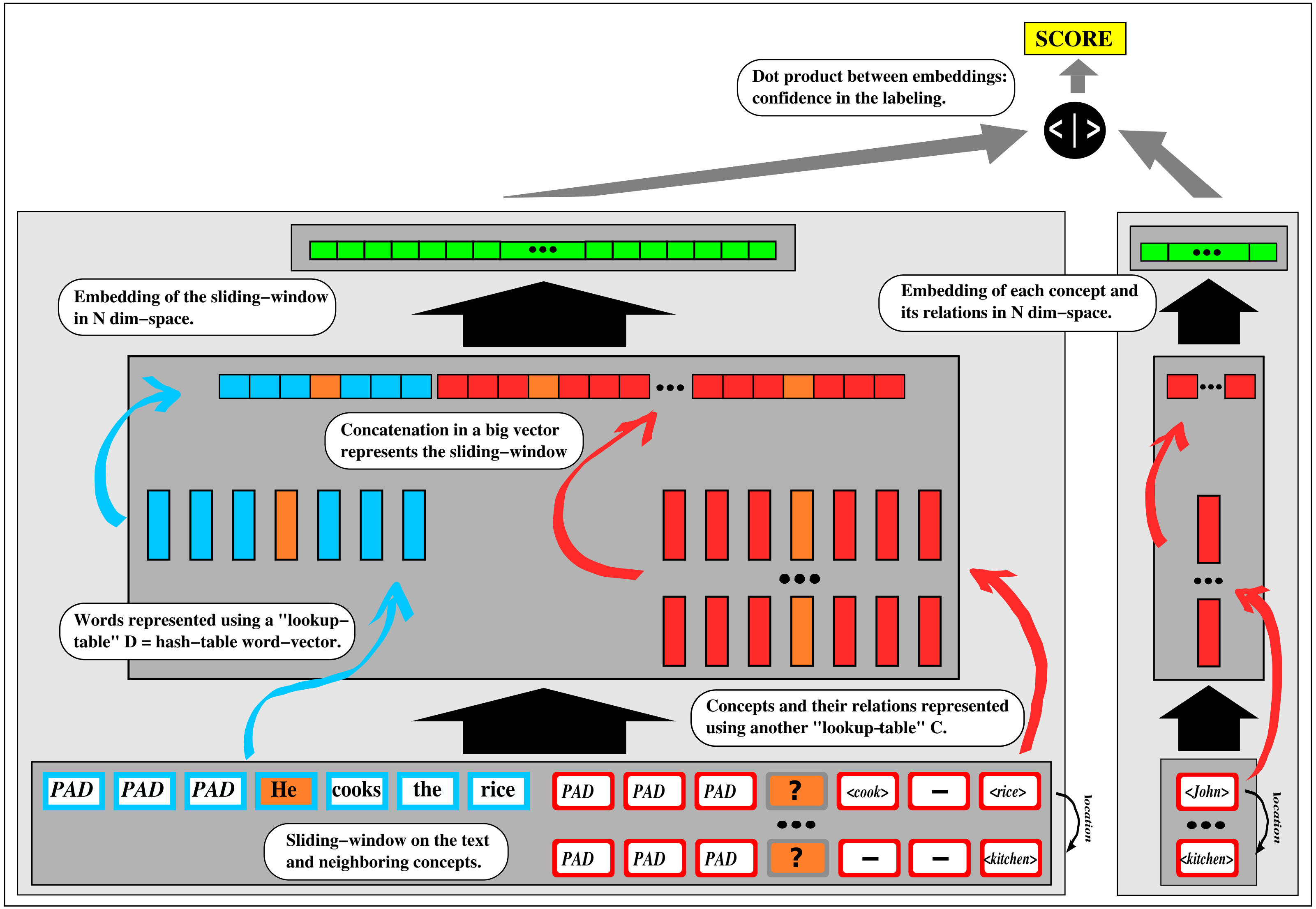
Scoring Illustration

Step 6: Compute $h(\text{<John>}, u)$.



Scoring Illustration

Step 7: Finally compute the score: $g_1(x, y_{-1}, u)^\top h(\text{<John>}, u)$.



Greedy “Order-free” Inference using LaSO

Adapted from LaSO (Learning As Search Optimization) [Daumé & al.,’05].

Inference algorithm:

1. For all the positions not yet labeled, predict the most likely concept.
2. Select the pair (position, concept) you are the most confident in. *(hopefully the least ambiguous)*
3. Remove this position from the set of available ones.
4. Collect all universe-based features of this concept to help label remaining ones.
5. Loop.

Experimental Results

Method	Features	Train Err	Test Err
SVM$_{struct}$	x	42.26%	42.61%
SVM$_{struct}$	$x + u$ (loc, contain)	18.68%	23.57%
NN$_{OF}$	x	32.50%	35.87%
NN$_{OF}$	$x + u$ (contain)	15.15%	17.04%
NN$_{OF}$	$x + u$ (loc)	5.07%	5.22%
NN$_{OF}$	$x + u$ (loc, contain)	**0.0%**	**0.11%**

- Different amounts of *universe* knowledge: no knowledge, knowledge about *containedby*, *location*, or both.

$\longrightarrow$ More world knowledge leads to better generalization.

$\longrightarrow$ Learning representations leads to better generalization.

Features Learnt By the Model

Our model *learns* representations of concepts embedding space.

Nearest neighbors in this space:

Query Concept	Most Similar Concepts
Gina	Francoise , Maggie
Mark	Harry, John
mother	sister, grandma
brother	friend, father
cat	hamster, dog
football	toy, videogame
chocolate	salad, milk
desk	bed, table
livingroom	kitchen, garden
get	sit, give

E.g. the model learns that female actors are similar, even though we have not given this information to the model.

Summary

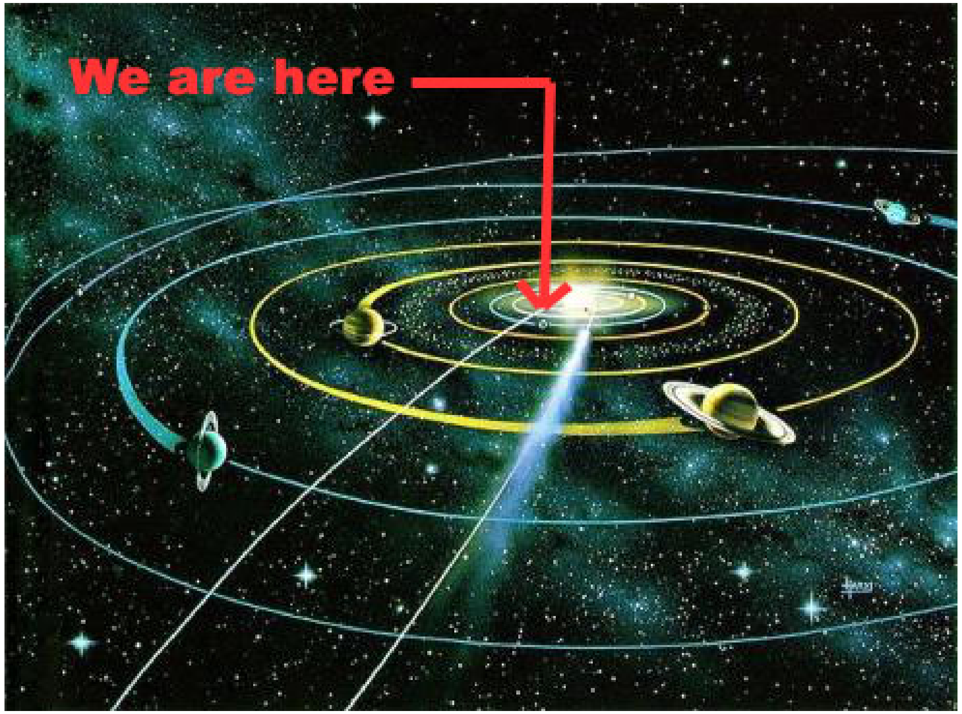
- *Simple*, but *general framework* for language grounding based on the task of *concept labeling*.

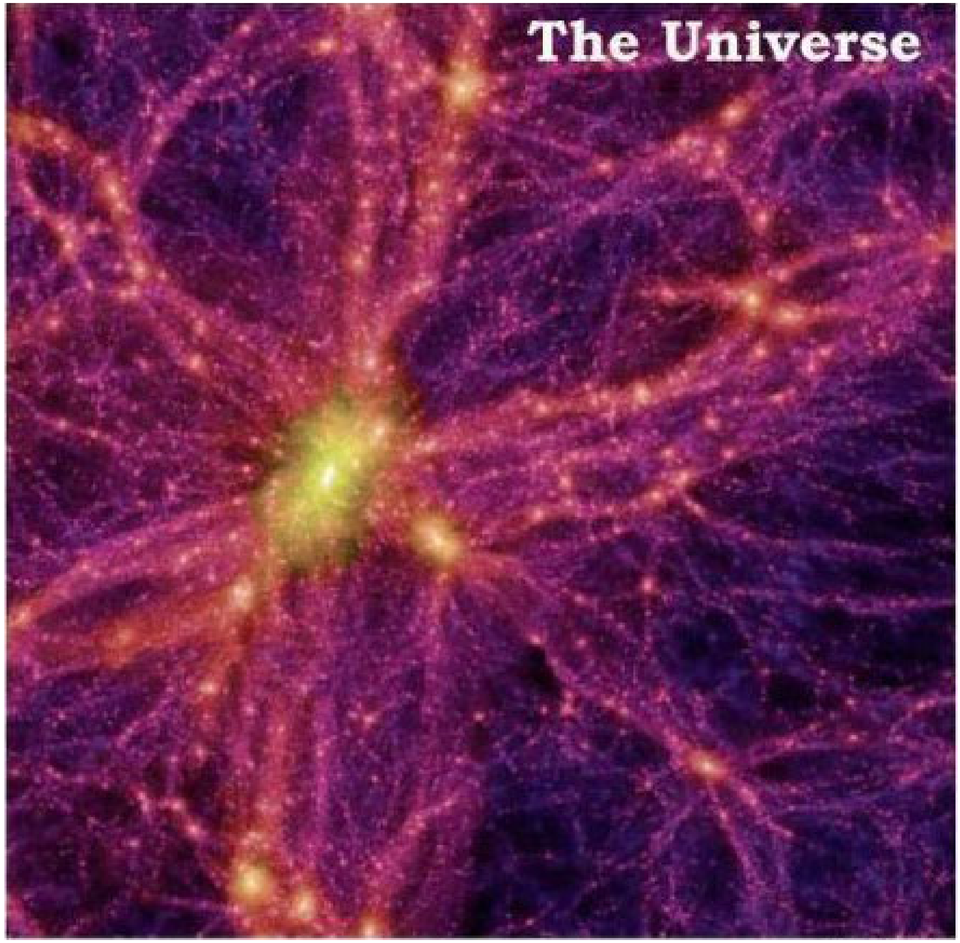
- *Scalable*, *flexible learning algorithm* that can learn without hand-crafted rules or features.

- Simulation validates our approach and shows that learning to disambiguate with world knowledge is possible.

AI goal: train learner living in a “computer game world” to learn language from scratch *from interaction alone* (communication, actions).

Final Conclusion





(Some of the) Previous Work

- Blocks world, KRL [Winograd, '72],[Bobrow & Winograd, '76]

- Ground language with visual reference, e.g. in blocks world [Winston '76],[Feldman et al. '96] or more recent works [Fleischman & Roy '07],[Barnard & Johnson '05],[Yu & Ballard '04],[Siskind'00].

- Map from sentence to meaning in formal language [Zettlemoyer & Collins, '05], [Wong & Mooney, '07], [Chen & Mooney '08]

  Example applications:
  (i) word-sense disambiguation (from images),
  (ii) generate Robocup commentaries from actions,
  (iii) convert questions to database queries.

Train the System

- Online training i.e. prediction and update for each example.
- At each greedy step, if a prediction $\hat{y}^t$ is incorrect, several updates are made to the model to satisfy:

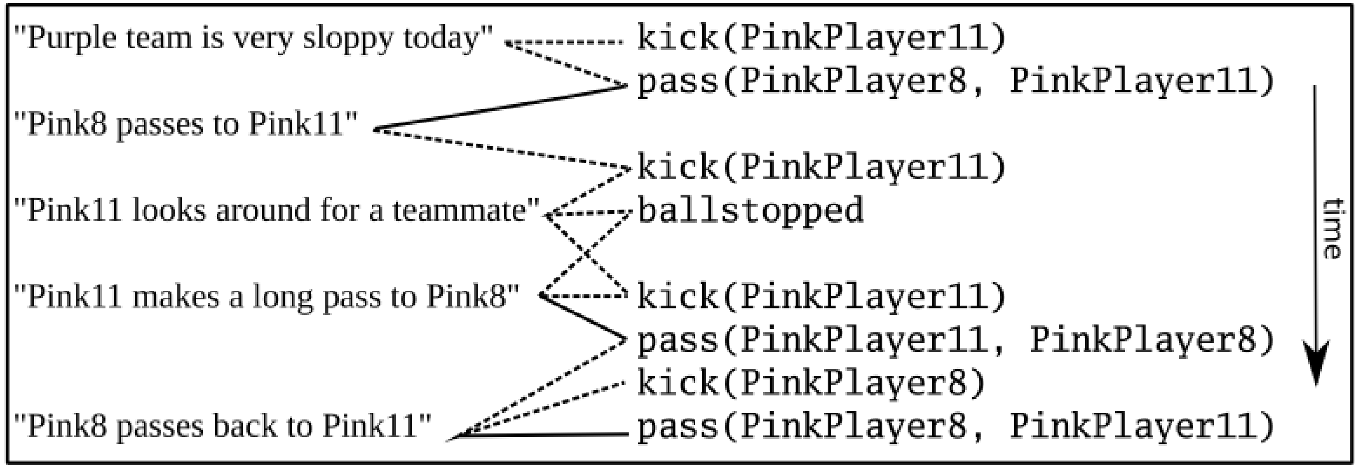
  For each correct labeling alternative $\hat{y}^{t-1}_{+y_i}$, $g(x, \hat{y}^{t-1}_{+y_i}, u) > g(x, \hat{y}^t, u)$.

- Intuitively, we want any incorrect partial prediction to be ranked below all correct partial labeling.
  $\rightarrow$ "Order-free" is not directly supervised.
- All updates performed with SGD + Backpropagation.

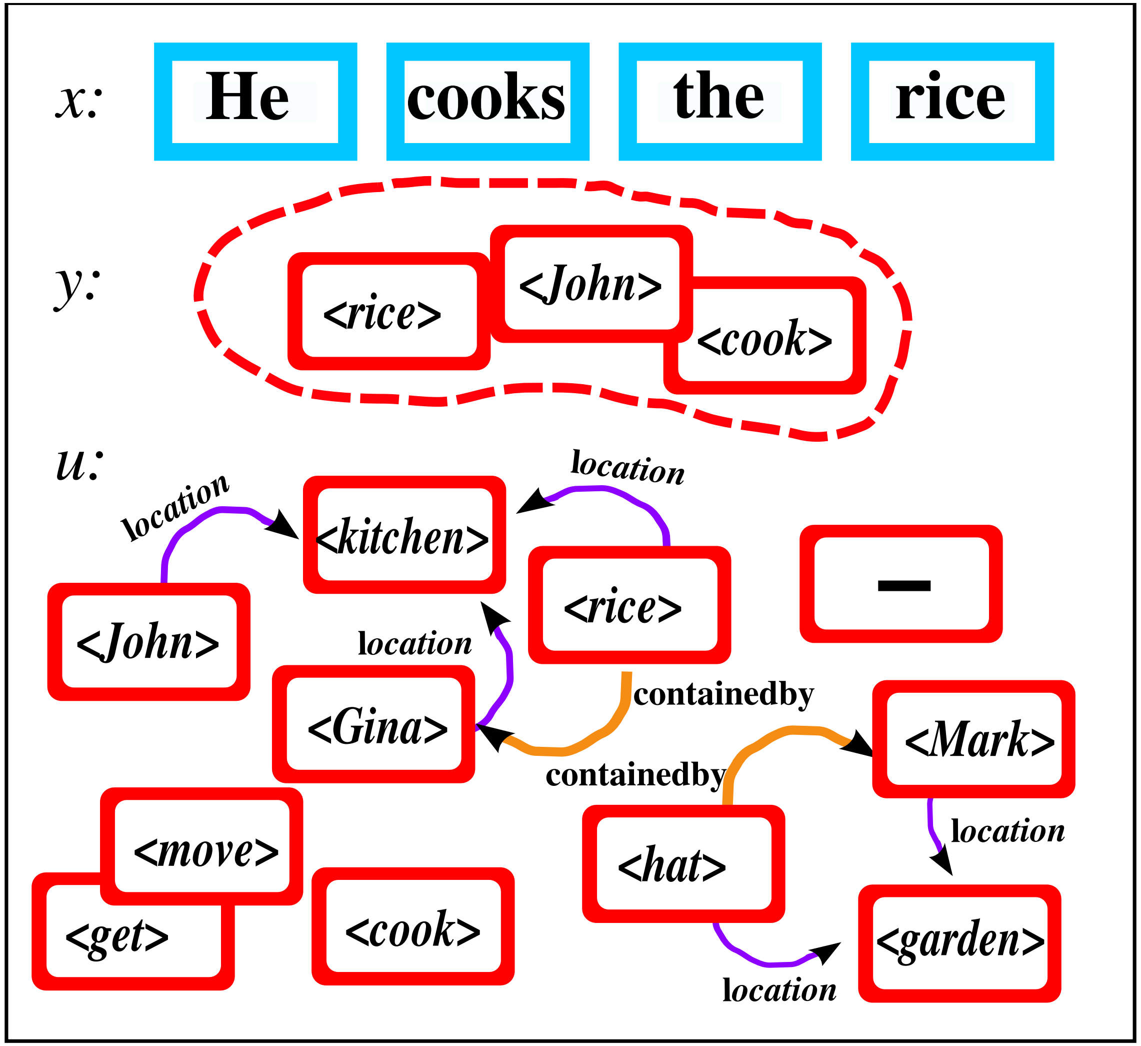
The Learning Signal: weak labeling scenario

Even more challenging setting: training data $\{\mathbf{x}_i, \mathbf{y}_i, \mathbf{u}_i\}_{i=1,\ldots,m}$ as before.
However, y is a set (bag) of concepts - **no alignment to sentence**.

This is more realistic:
A child sees actions and hears sentences $\rightarrow$ must learn correlation.



Extension: weak concept labeling



Extension: weak concept labeling

Solution: modified LaSo updates – rank anything in the "bag" higher than something not in the bag.

Results:

Method	Features	Train Err	Test Err
SVM_{struct}	$x + u$ (loc, contain)	18.68%	23.57%
NN_{OF}	$x + u$ (loc, contain)	**0.0%**	**0.11%**
NN_{WEAK}	$x + u$ (loc, contain)	**0.0%**	**0.17%**