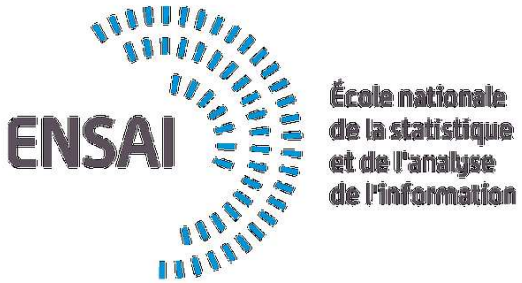
Machine Learning for Natural Language Processing

Guillaume Gravier

guillaume.gravier@irisa.fr



Outline of the course

Lectures (6 x 3h)

- Lecture #1: Introduction and representation of words
  - Notions: morphology, tokens, lemmas, POS, word net, word embedding
  - Hands-on: manipulate basic pipelines and visualize word embeddings
- Lecture #2: Representation of documents
  - Notions: vocabulary, Zipf's curse, bag of words, Bayes, RNN, BERT
  - Hands-on: basic neural network classifiers
- Lecture #3: Language models
  - Notions: ngrams, LSTM, bi-LSTM, language generation
  - Hands-on: train a small LM and generate text
- Lecture #4: Transformers and large language models
  - Notions: encoder/decoder, transformers, fine-tuning
  - Hands-on: visualize embeddings, fine-tune a LLM

A quick glance at RNN tagging

Tagging solves (almost) everything

Many NLP tasks can be cast as a *tagging* task, i.e., assigning a unique tag to each token of an utterance.

- sentence boundary detection: predict of word ends sentence

The	cat	is	black	The	dog	is	brown
0	0	0	1	0	0	0	1

- part-of-speech tagging

The	cat	drinks	milk
DET	NOUN	VERB	NOUN

- dependency parsing

The	cat	drinks	milk
DET	NSUBJ	ROOT	OBJ

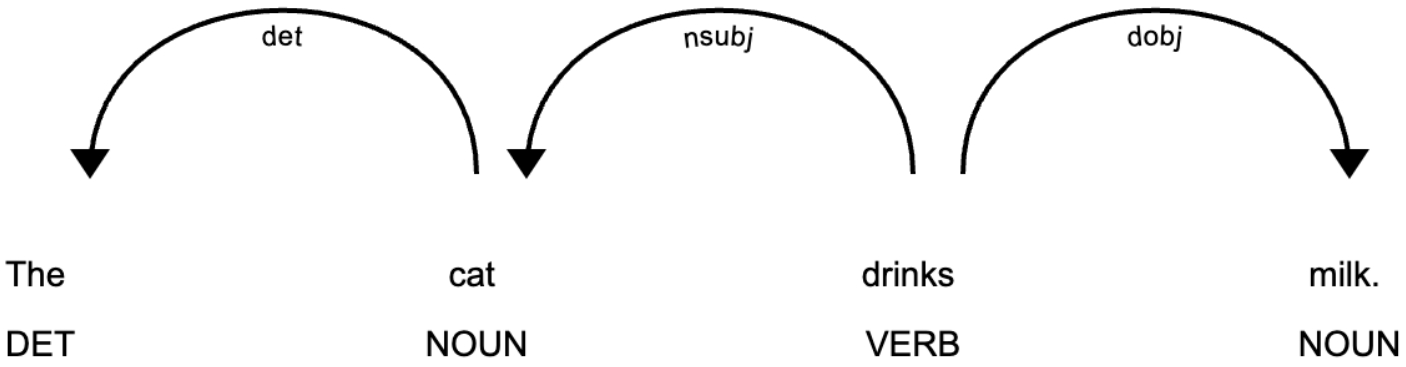
Tagging solves (almost) everything illustrated

```
import spacy
process = spacy.load('en_core_web_md')

for token in process('The cat drinks milk.'):
    print('{:8} {:6} {:8} {:6}'.format(token.text, token.pos_, token.tag_, token.dep_))


spacy.displacy.render(xr, style="dep", jupyter=True)
```

```
The      DET    DT       det
cat      NOUN   NN       nsubj
drinks   VERB   VBZ      ROOT
milk     NOUN   NN       dobj
.        PUNCT  .        punct
```



Tagging solves (almost) everything: the IOB trick

Can also tag spans and perform semantic role labeling

- entity detection: predict if within an entity or not

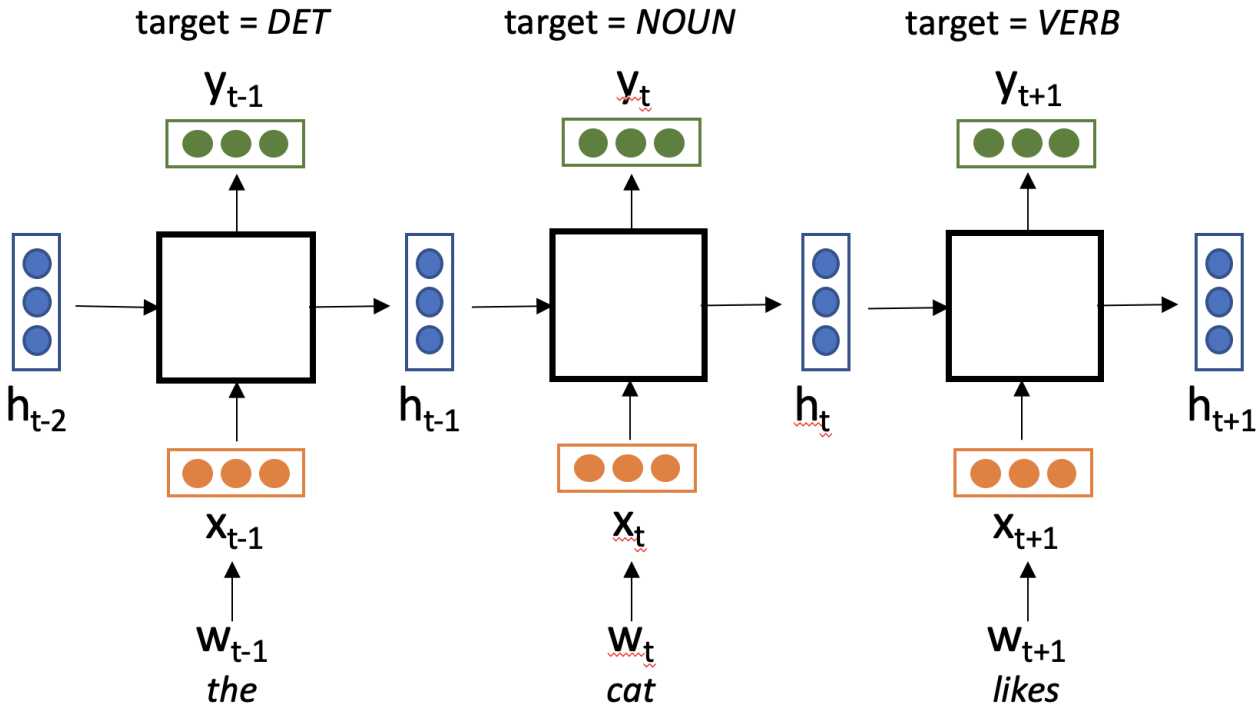
À	Marseille	Jean-Claude	Gaudin	est	élu	depuis	1995
O	B	B	I	O	O	O	B
O	B-LOC	B-NP	I	O	O	O	B-TIME

  - The BIO trick relies on three special tags
    - B begining of event
    - I inside event
    - O outside event
  - generalizes to variety of tasks, e.g., chunking, slot filling, parsing

Je	veux	un	resto	italien	à	Rennes
O	O	O	B-WHAT	I-WHAT	O	B-WHERE

Recurrent neural network tagging

Predict at each position t the distribution probability over the set of tags from the hidden state h_t

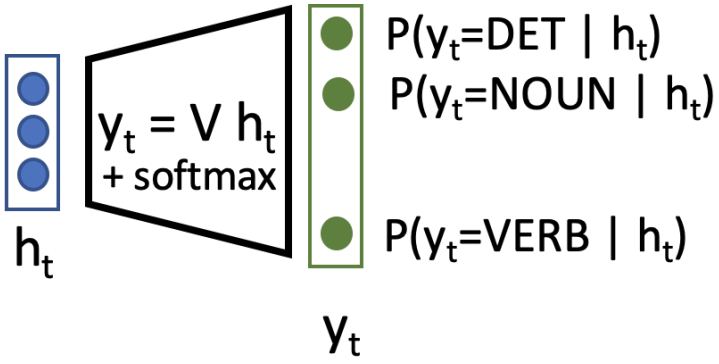


Embedding layer $x_t = C(w_t)$

Merging layer $\tilde{x}_t = c_t + h_{t-1}$

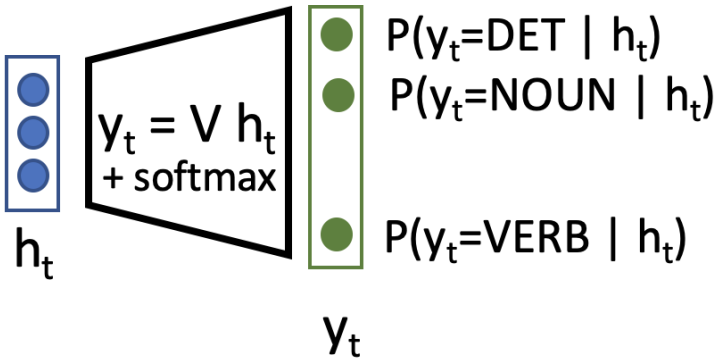
State prediction $h_t = \sigma(U\tilde{x}_t)$

Output prediction $\widehat{y}_t = \text{softmax}(Vh_t)$



Recurrent neural network tagging

Embedding layer	$x_t = C(w_t)$
Merging layer	$\tilde{x}_t = c_t + h_{t-1}$
State prediction	$h_t = \sigma(U\tilde{x}_t)$
Output prediction	$\widehat{y}_t = \text{softmax}(Vh_t)$



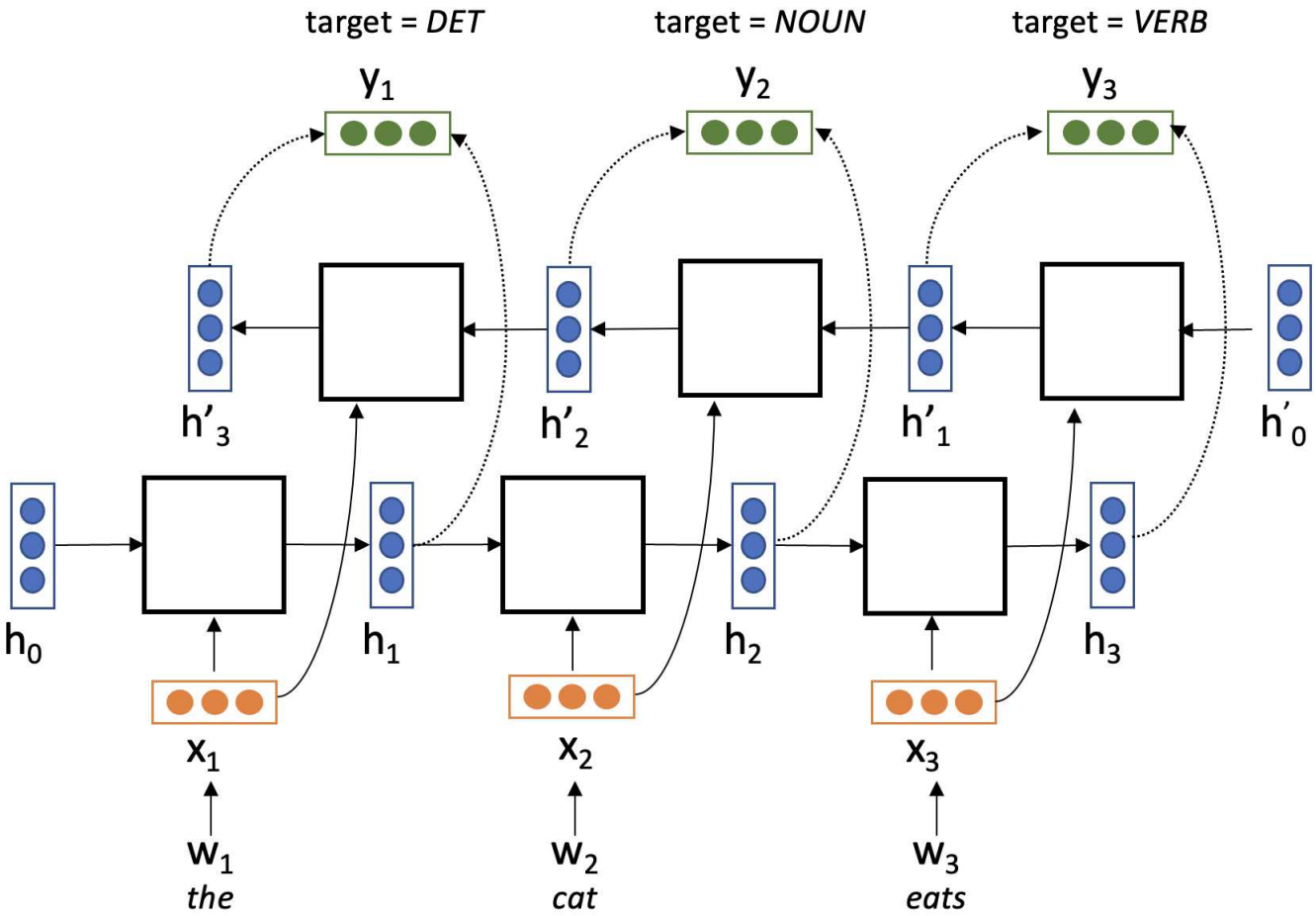
Decision at step t: $\hat{c}_t = \arg\max_i \widehat{y}_t(i)$

Objective function for training is categorical cross-entropy

$$J(\theta = \{C, U, V\}) = -\sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{j=1}^{|C|} \delta_{(y_t^{(i)}=j)} \ln(\widehat{y}_t^{(i)}(i))$$

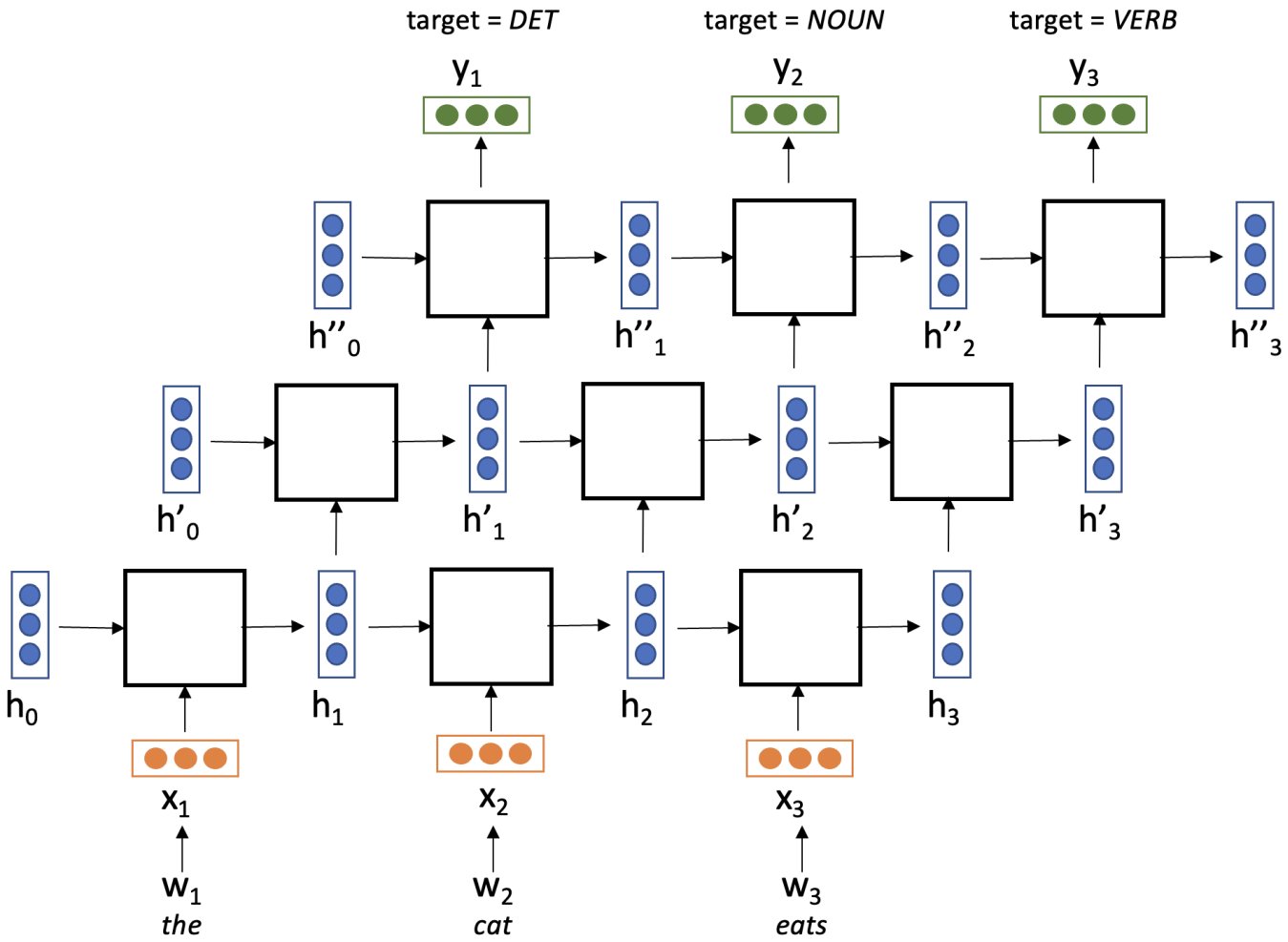
where $y_t^{(i)}$ is the actual label (among $|C|$ labels) for the t-th token of training sample i

Bidirectional RNN tagging



- h_i: summary of sequence up to position i
- h'_i: summary of from position i
- $h_i + h'_i$: representation of w_i in the context of the sentence

Hierarchical RNN tagging



Language models

Language models: what for?

Define a probability distribution over sentences from a vocabulary V

$P[\text{This sentence is very likely}]\ ?$ $P[\text{Sentence this not very likely is}]\ ?$

Very useful for language identification, speech recognition and translation

- parce qu'il était vs. pace qu'il été vs. parce qu'il étais
- avocat $\longrightarrow$ lawyer vs. avocado

Also very useful for prediction (e.g., spell checking, language generation) if able to provide a distribution probability for the next word

$\rightarrow$ une baguette de ????? $\rightarrow$ Jean aime ????

probabilistic grammar	$\longrightarrow$	powerful but too complex and not robust
n-gram and neural models	$\longrightarrow$	basic but robust and efficient

$\Longrightarrow$ Often boils down to assigning a (non-zero) probability to a(ny) word occurrence given the occurrences of the previous and/or following words

Why is sentence probability an issue?

$$
\begin{aligned}
P[\text{Mark loves Mary who loves Paul}] \quad = \quad & P[\text{Mark}] \times \\
& P[\text{loves|Mark}] \times \\
& P[\text{Mary|Mark loves}] \times \\
& P[\text{who|Mark loves Mary}] \times \\
& P[\text{loves|Mark loves Mary who}] \times \\
& P[\text{Paul|Mark loves Mary who loves}]
\end{aligned}
$$

Use approximations of the *history* to simplify probability/score computation

$$
P[w_i|\underbrace{w_{i-1}\dots w_1}_{\textit{history}}] \simeq P[w_i|\underbrace{f(w_{i-1}\dots w_1)}_{h(w_i)}]
$$

The ngram approximation

$$P[w_i|w_1, w_2, \ldots, w_{i-1}] \doteq P[w_i|\underbrace{w_{i-n+1}, \ldots, w_{i-1}}_{n-1 \text{ words}}]$$

$$P[\text{Mark loves Mary who loves Paul}] =$$

bigram (n=2)

$P[\text{Mark}] \times$

$P[\text{loves}|\text{Mark}] \times$

$P[\text{Mary}|\sout{\text{Mark}}\ \text{loves}] \times$

$P[\text{who}|\sout{\text{Mark loves}}\ \text{Mary}] \times$

$P[\text{loves}|\sout{\text{Mark loves Mary}}\ \text{who}] \times$

$P[\text{Paul}|\sout{\text{Mark loves Mary who}}\ \text{loves}]$

trigram (n=3)

$P[\text{Mark}] \times$

$P[\text{loves}|\text{Mark}] \times$

$P[\text{Mary}|\text{Mark loves}] \times$

$P[\text{who}|\sout{\text{Mark}}\ \text{loves Mary}] \times$

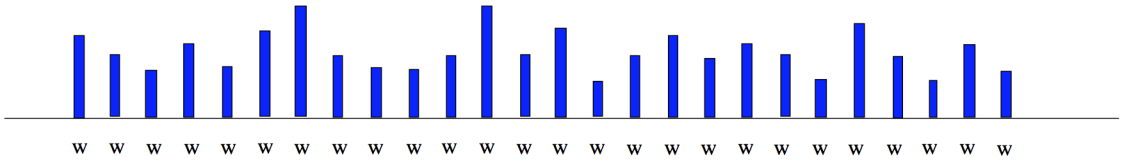
$P[\text{loves}|\sout{\text{Mark loves}}\ \text{Mary who}] \times$

$P[\text{Paul}|\sout{\text{Mark loves Mary}}\ \text{who loves}]$

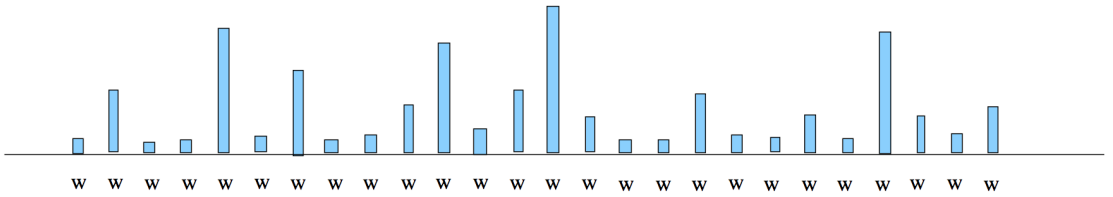
From the computer point of view: a set of tables...

Nothing but a *set of distribution tables over words in* V for each possible history.

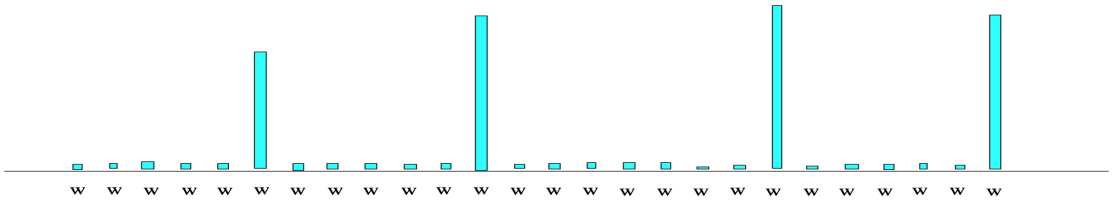
ϵ



Mark



Mark loves



... or a stochastic graph

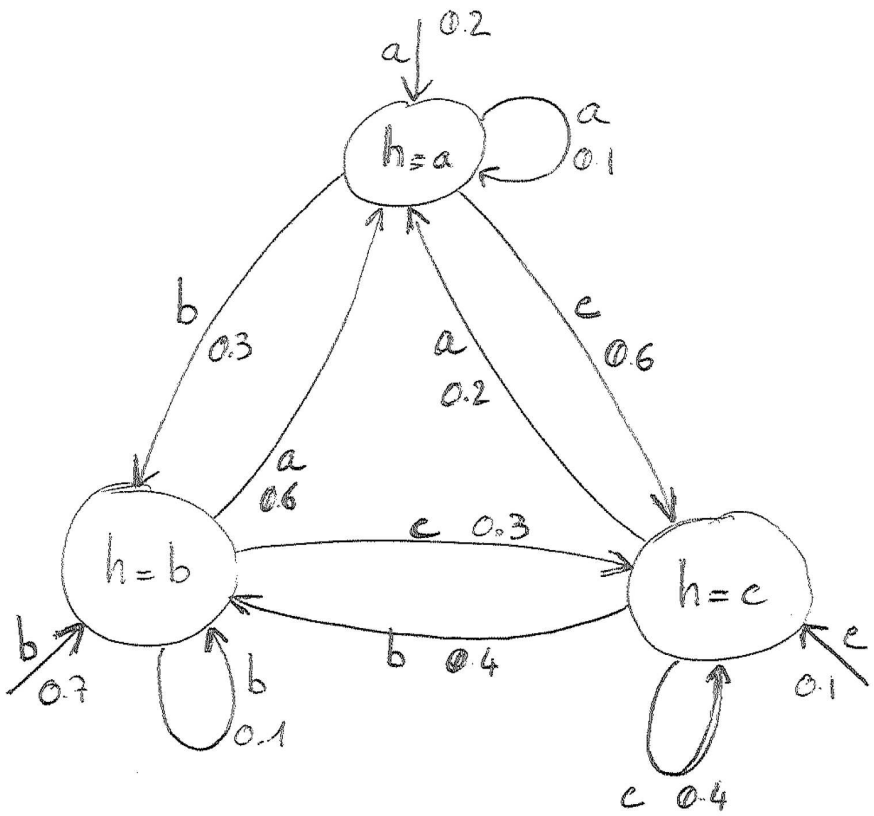
h	w	p
ϵ	a	0.2
ϵ	b	0.7
ϵ	c	0.1

h	w	p
a	a	0.1
a	b	0.3
a	c	0.6

h	w	p
b	a	0.6
b	b	0.1
b	c	0.3

h	w	p
c	a	0.2
c	b	0.4
c	c	0.4

table view



graph view

$$
\begin{aligned}
P[abaca] &= P[a] \times P[b|a] \times P[a|b] \times P[c|a] \times P[a|c] \\
&= 0.2 \times 0.3 \times 0.6 \times 0.6 \times 0.2 = 0.00432
\end{aligned}
$$

Text generation with ngram models

Sampling from the distributions $P[w|h]$, where each new sample changes the history.

For example, with a trigram model:

1. choose initial word w_1 according to the distribution $P[u|\epsilon]$
2. choose next word w_2 according to $P[u|w_1]$
3. for $i = 3$ to the number of words you want
4. choose w_i according to $P[w_i|w_{i-2}w_{i-1}]$
5. end for

Similar to a random walk in the LM graph ... but not very useful indeed

Markov approximation of language

- **Order 1**: REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE T
- **Order 2**: THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPETED

[courtesy of F. Coste]

Estimation: the theory

The problem: From a (large) corpus of sentences $\mathcal{C} = \{S_1, \ldots, S_n\}$, estimate the set of probabilities $P[w|h]\ \forall h, w$ so that they reflect the reality of the data

The solution: maximum likelihood estimation given by

$$P_{\text{ML}}[w|h] = \frac{C(hw)}{C(h)} = \frac{C(hw)}{\sum_{v \in V} C(hv)} = \frac{\text{\# times you see hw}}{\text{\# times you see h}}$$

The maths: for a bigram model, the solution comes from the maximization of

$$\ln P[\mathcal{C}] = \sum_{i=1}^{n} \ln P[S_i] = \sum_{i=1}^{n} \left(\ln \left(P[w_1^i | \epsilon] \right) + \sum_{k=2}^{n_i} \ln \left(P[w_k^i | w_{k-1}^i] \right) \right)$$

with the constraints that $\forall v \in V$ we have $\sum_{w \in V} P[w|v] = 1$.

Estimation: a toy example

[borrowed again from F. Yvon]

$$01101111010111 10$$
$$11011100111010111$$

0	9	1	23	$p(0) = 9/32, p(1) = 23/32$
0 1	7	1 1	15	$p(0\|0) = 1/8, p(1\|0) = 7/8$
0 0	1	1 0	7	$p(0\|1) = 7/22, p(1\|1) = 15/22$
0 1 1	6	1 1 0	6	$p(1\|01) = 6/7, p(0\|01) = 1/7$
0 1 0	1	1 1 1	8	...
0 0 1	1	1 0 1	5	
		1 0 0	1	

Estimation: Zipf still sucks

Taille du corpus		10^6	4×10^6	3.9×10^7
# Phrases		37831	187892	1611571
# Mots		892333	4472827	38532518
Unigrammes	Total	892333	4472827	38532518
	Distincts	17189	19725	19981
	Singletons	2465	235	0
Bigrammes	Total	892333	4472827	38532518
	Distincts	285692	875497	3500633
	Singletons	199493	565549	2046462
Trigrammes	Total	854502	4283935	36920518
	Distincts	587985	2370914	14039536
	Singletons	510043	1963267	10987166

The principle of discounting

borrow probability mass from observed events pretending they were observed less than they actually were, and redistribute the probability mass to unobserved events (as cleverly as possible)

- Laplace smoothing (aka add-one): $c^*(hw) = c(hw) + 1$

$$P[w|h] = \frac{c(hw)+1}{\sum_{v \in V}(c(hv)+1)} = \frac{c(hw)+1}{c(h)+|V|}$$

- absolute discounting $\rightarrow$ same as Laplace but soustractive

$$P[w|h] \propto \max(c(hw) - \delta, 0))/\sum_u c(uw)$$

- Kneser-Ney discounting

  $\rightarrow$ refinement of absolute discounting where δ depends on h

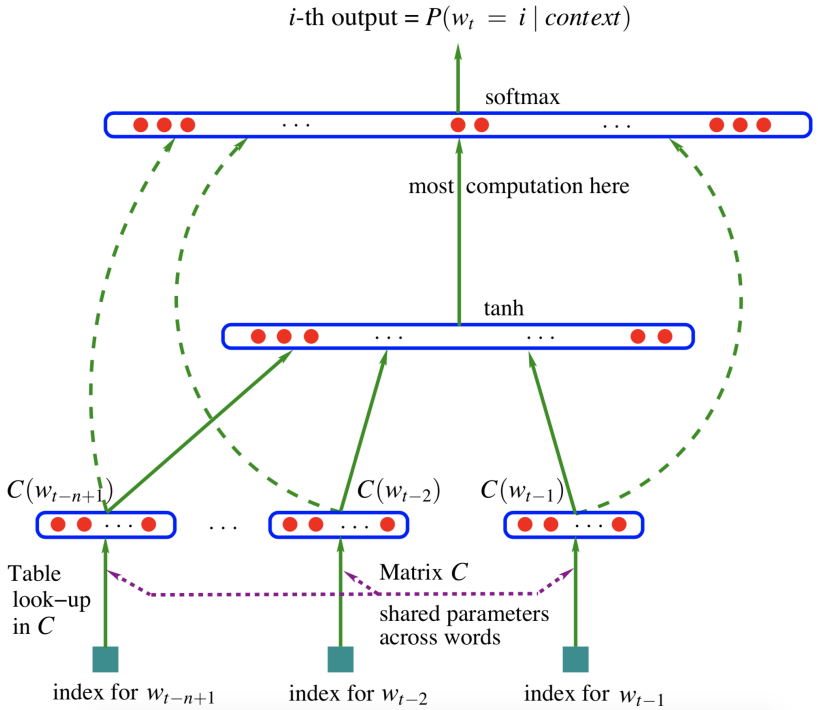
- Good-Turing discounting (unseen events get probability n_1/N)

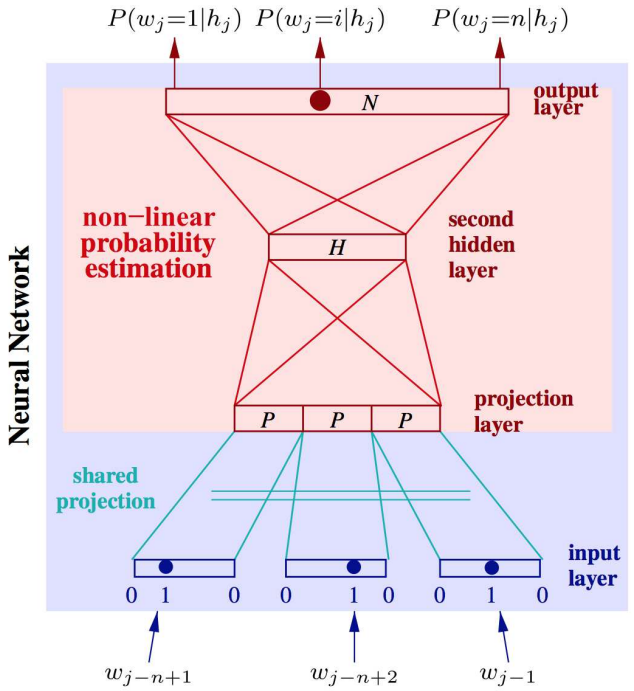
$$c^*(hw) = (c(hw)+1)\frac{n_{c(hw)+1}}{n_{c(hw)}}$$, n_r = # ngrams occurring r times

Directly estimating probabilities with neural networks

Directly compute the n-gram probability with a feed-forward neural network train to yield

$$P[w_i|w_{i-1}\dots w_{i-n+1}] \doteq f(w_i,\dots,w_{i-n+1})$$





Yoshua Bengio et al. A Neural Probabilistic Language Model, Journal of Machine Learning Research, 2003

Holger Schwenk. Continuous space language models. Computer, Speech and Language, 21:492-518, 2007

ENSAI École nationale de la statistique et de l'analyse de l'information

Dissecting the seminal model of Bengio et al. 2003

Decomposition of $f(w_i, \ldots, w_{i-n+1})$ in two parts

1. an embedding $c()$ of words mapping token i to $c(i) \in \mathbb{R}^d$, concatenated as x
2. a MLP with softmax to estimate

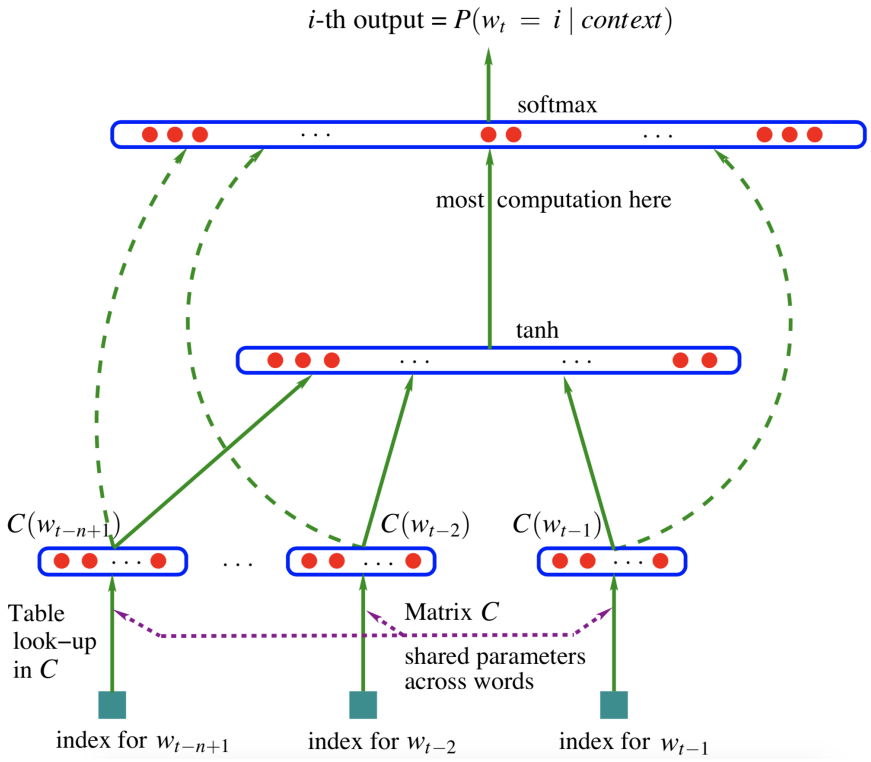
$$P[w_i | w_{i-1} \ldots w_{i-n+1}] = \frac{\exp(y_{w_i})}{\sum_k \exp(y_k)}$$

with

$$y = U \tanh(Hx + d)$$

or

$$y = b + Wx + U \tanh(Hx + d)$$

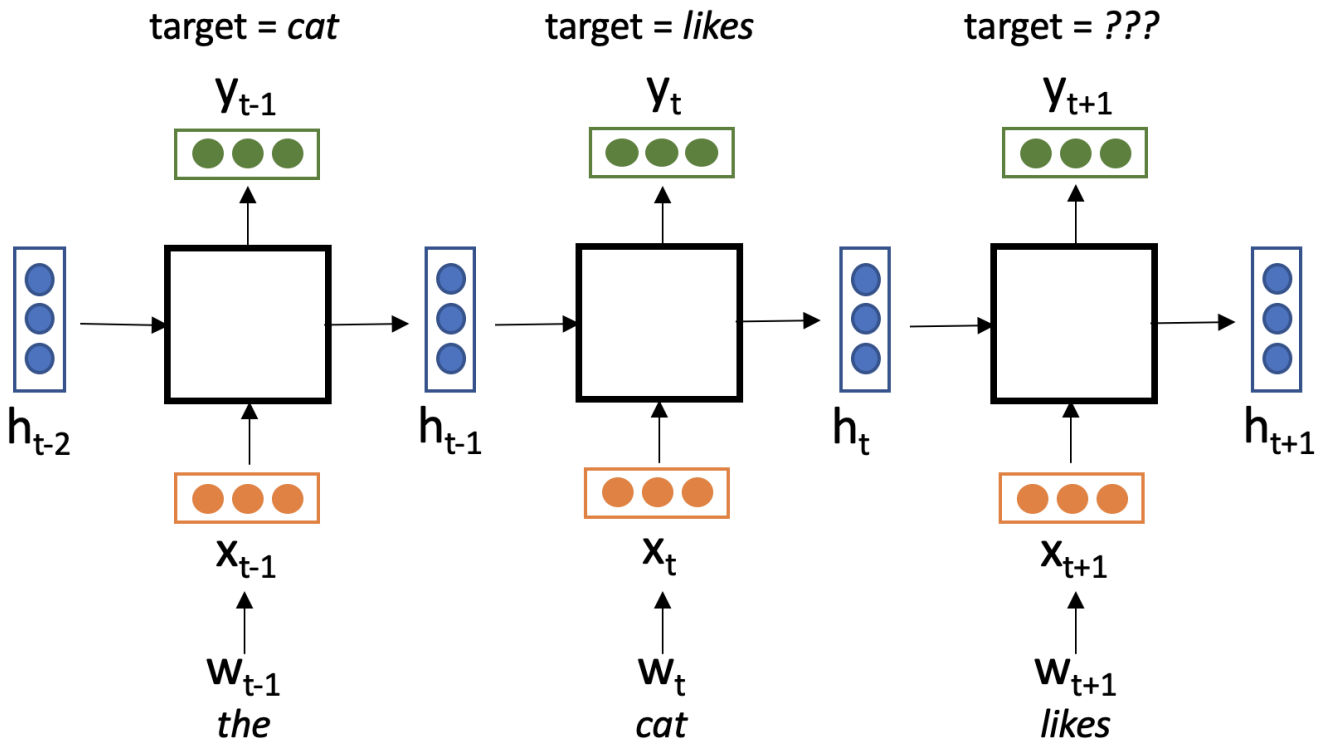


Objective function: $J \propto \sum_i, t \log f(w_t^i, \ldots, w_{t-n+1}^i; \Theta) + R(\Theta)$

Language modeling with recurrent networks

Use the hidden state vector h_{i-1} as (a summary of) the history of word w_i to predict

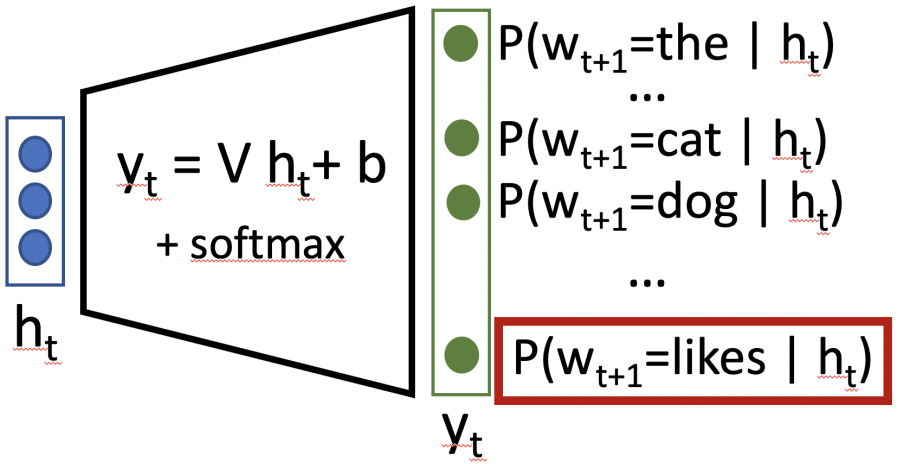
$$P[w_i | \underbrace{w_{i-1} \ldots w_1}_{\text{full history}}] \doteq f(w_i, h_{i-1})$$



Much less parameters than in the feed-forward approach!

Language modeling with recurrent networks

Predict a probability distribution over the vocabulary at each time step based on a feed-forward and softmax projection, where y_t is trained to hold probabilities $P[W_{t+1}|h_t]$ for all posible values of W_{t+1}

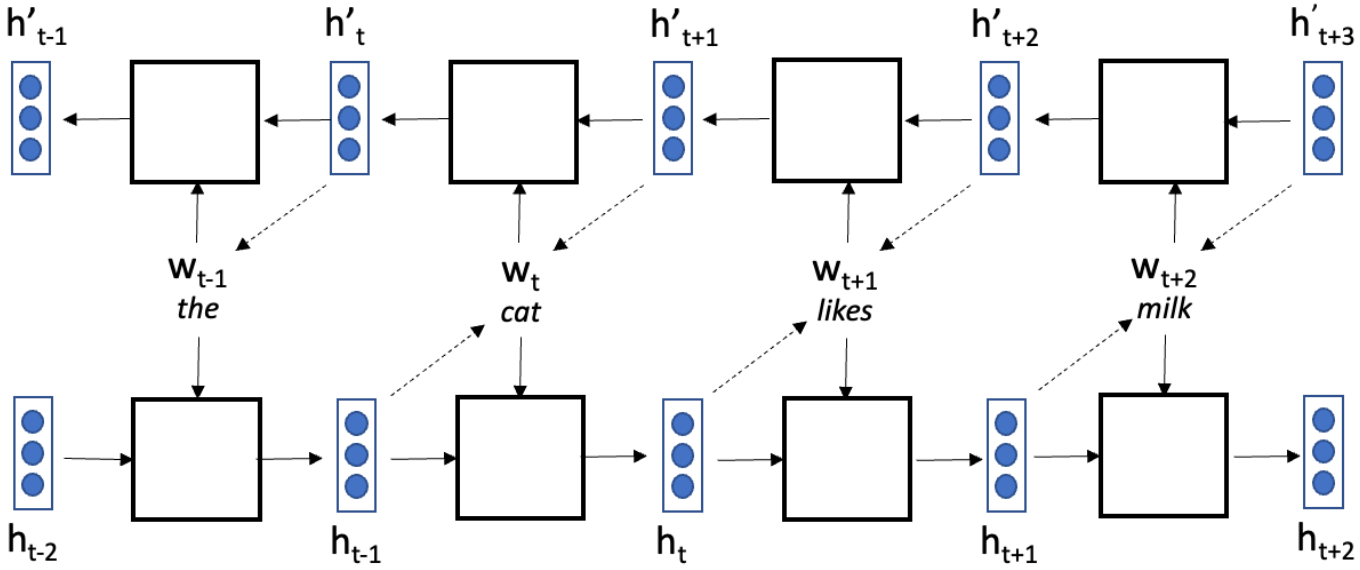


Parameters trained in a self-supervised manner with large amounts of texts so as to maximize the log-likelihood

$$J(\Theta) = \sum_t \log\left(y_t(\text{index}(w_{t+1}))\right)$$

A bidirectional variant of RNN-based language models

Model language as $P[w_1 \dots w_n] = P[w_n]P[w_{n-1}|w_n] \dots P[w_1|w_2 \dots w_n]$ and combine forward and backward RNN states to predict the probability distribution over words, thus optimizing $\prod_k P[w_k|w_1, \dots, w_{k-1}, w_{k+1}, w_n]$.

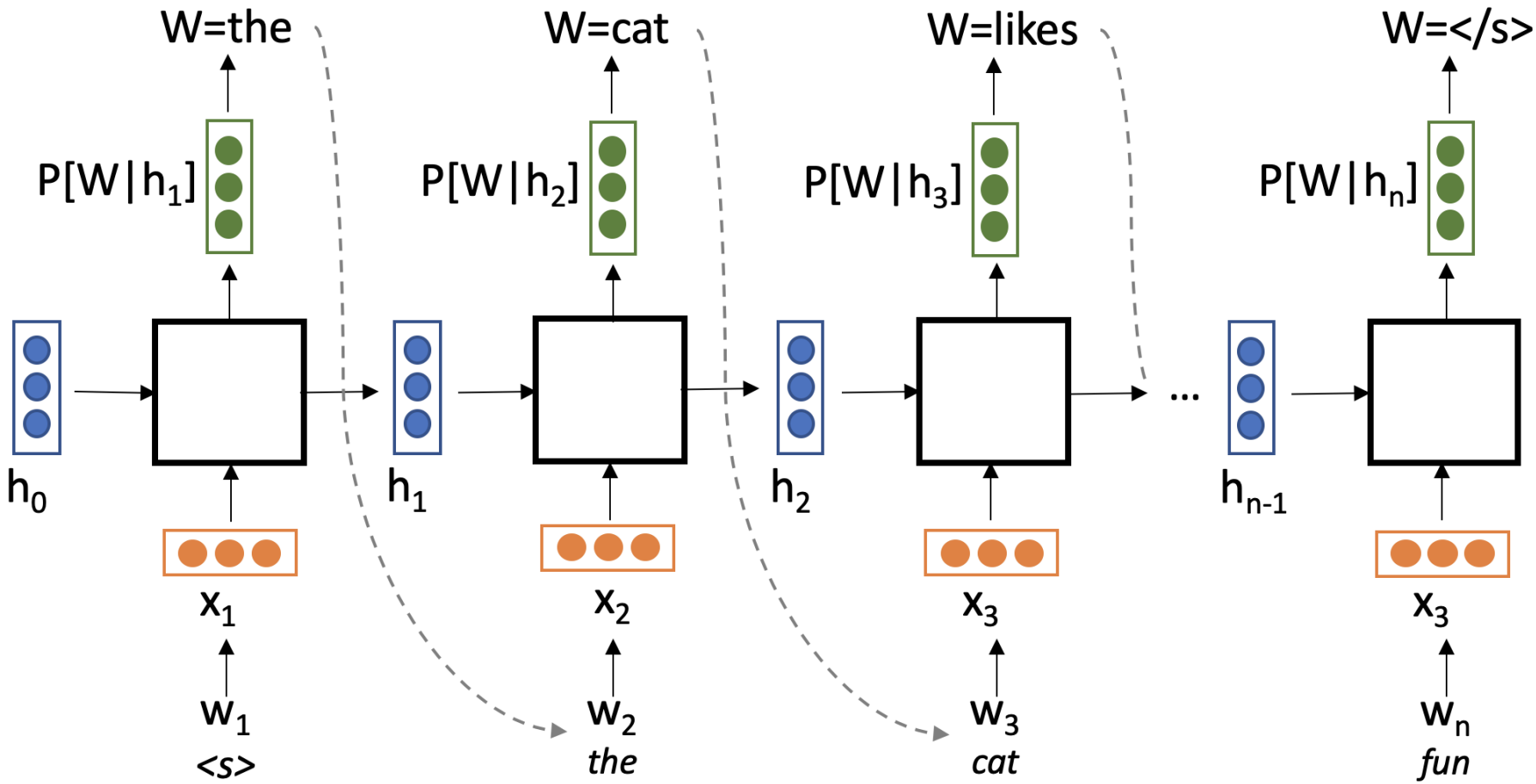


$h_t = f(h_{t-1}, w_t)$: encode $w_1, \dots, w_t$ $\longrightarrow$ predict w_{t+1}

$h'_t = f'(h'_{t+1}, w_t)$: encode $w_t, \dots, w_n$ $\longrightarrow$ predict w_{t-1}

$\Longrightarrow$ predict w_t from distribution $y_t = \text{softmax}(A[h_{t-1}h'_{t+1}] + b)$

Sampling from RNNs is trivial



Yet no control on semantics, cannot draw from $P[W_1, \ldots, W_n|X]$

Beam search decoding

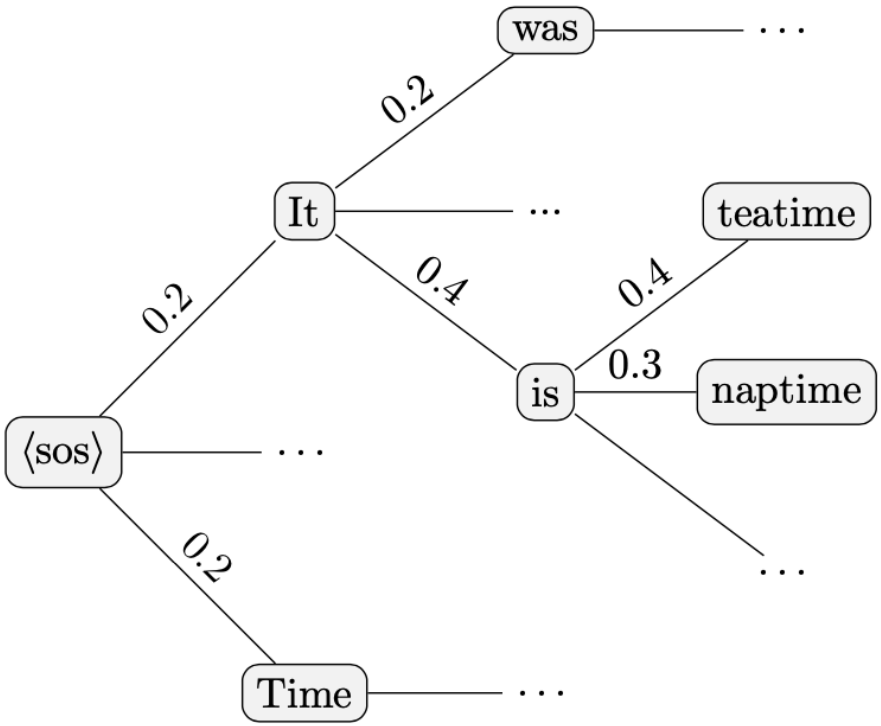


Figure 4: Toy example of beam search procedure with beam width $k = 2$. The search has been run for 3 steps and no hypothesis has terminated with ⟨eos⟩ yet. Edges are annotated with probabilities of tokens. Only the tokens after pruning to the top k hypotheses are shown.

borrowed from Ziang Xie's 2018 practical guide on neural text generation

For (gory) details on beam search decoding for text generation, see Sina Zarrieß et al., 2021. Decoding Methods in Neural Language Generation: A Survey

Conditioning language model on images

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A skateboarder does a trick on a ramp.

A dog is jumping to catch a frisbee.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

A little girl in a pink hat is blowing bubbles.

A refrigerator filled with lots of food and drinks.

A herd of elephants walking across a dry grass field.

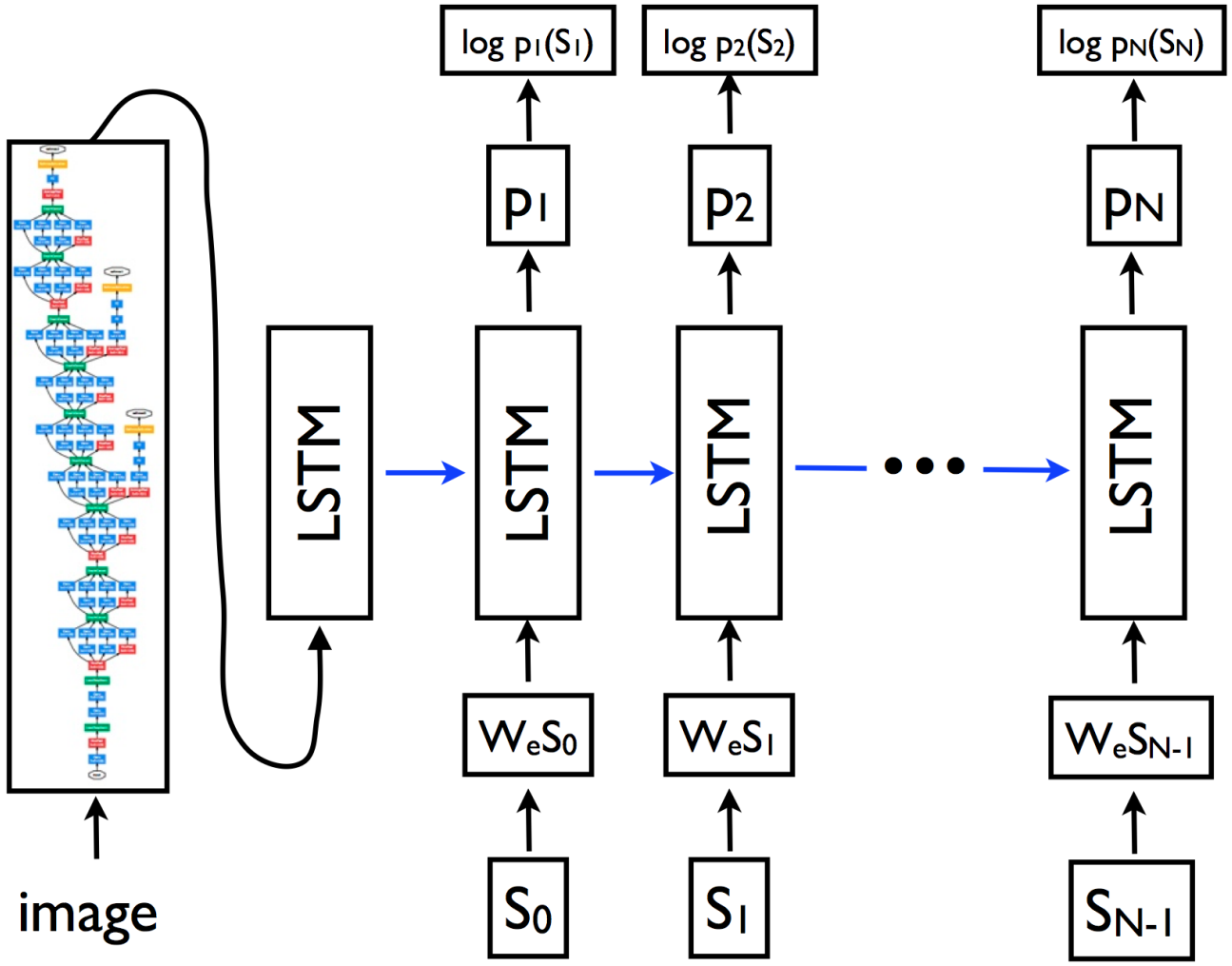
A close up of a cat laying on a couch.

A red motorcycle parked on the side of the road.

A yellow school bus parked in a parking lot.

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image

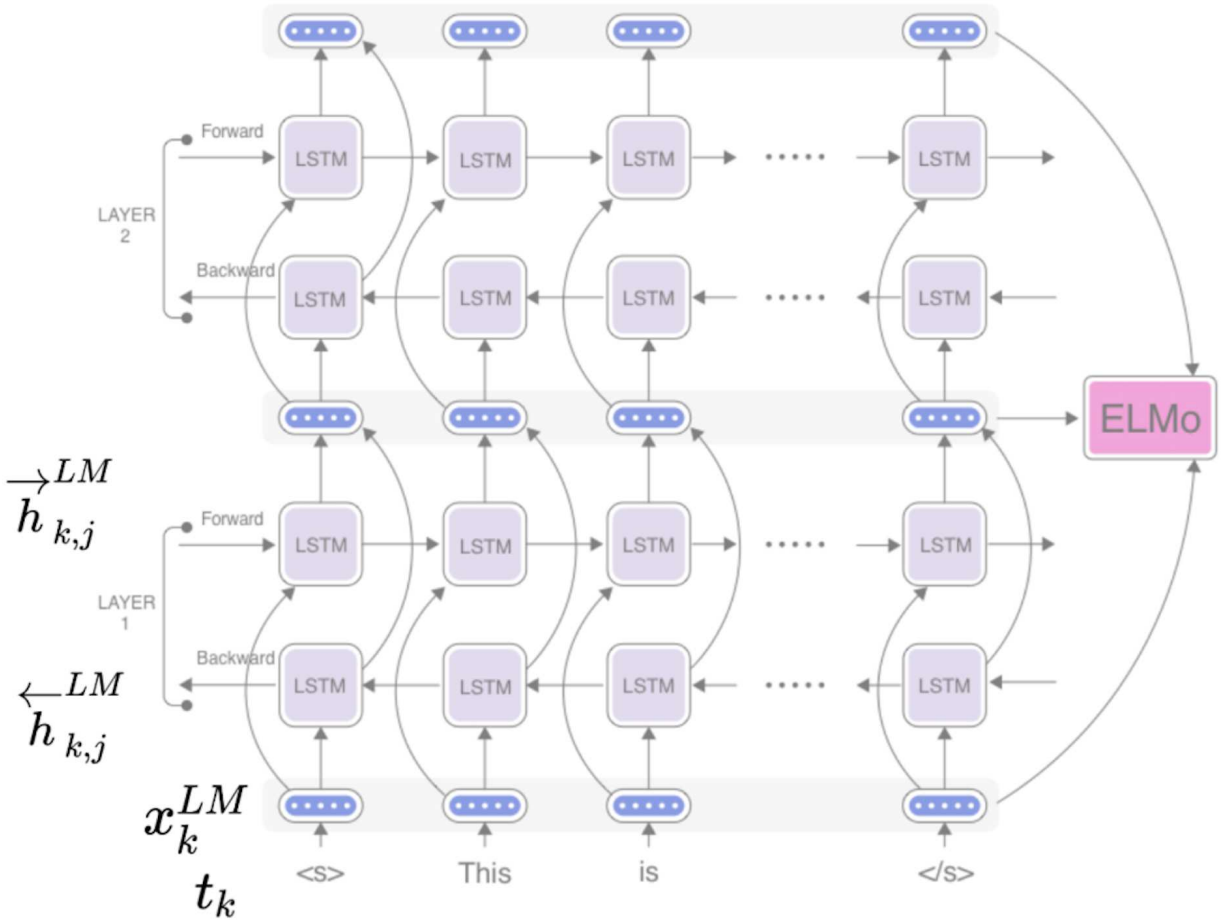
Conditioning language model on images



Training parameters of the RNN on images with human-generated captions so as to maximize the likelihood of the (training) caption texts.

LM pretraining for (contextual) word embeding

A key idea now widely used: pretrain a LM (here a 3-layer biLSTM LM) on large amounts of data to generate (deep) contextual representations of the onput tokens for further use in downstream tasks (here linear combination of the three embeddings from the pretrained model)



Peters et al. Deep Contextualized Word Representations. NAACL 2018


That's all Folks!