Neural Word Embeddings

Roberto Basili, Danilo Croce

Machine Learning, Web Mining & Retrieval 2020/2021

Outline

- Recurrent Networks for the estimation of neural language models
- The CBOW and Skip-gram model
- Computational Tricks
- Applications of word embeddings to Language Processing

Neural Networks

- Powerful and flexible Machine Learning algorithm
- They can learn highly non linear functions and learn complex concepts
  - difficult to train until 2006 with the Deep Learning movement
- One of the key elements of Deep Learning is the use of pre-training techniques

Pre-training

- NNs are known to model non-linear classification functions
- The main difficulty is that NN cost functions are not convex
  - high probability of stopping in a local minimum
- Pre-training is a technique to initialize the network parameters
  - in a way that they are nearer to the global minimum
  - or at least in a better region of the cost function surface

Pre-training

- Pre-training can be obtained through
  - Auto-Encoders
  - Restricted Boltzmann Machines
  - Training with other data (e.g. heuristically annotated data)
- In NLP, often a form of pre-training is obtained by adopting Word Embeddings
  - a *d*-dimensional space representing words
  - each word vector encodes in its dimensions useful information to drive the learning process

Word representations in NNs

- Word vectors are related also to fighting the "curse of dimensionality" of standard word representations
- In a BOW model, the greater the vocabulary size the more examples you need to learn all the relevant variations of each feature
- If we know, that two words are similar given a dense vector representation of them
  - we could not observe all the necessary variations of the data
  - but instead we could rely on the similarity to make similar inferences during training

Language Models

- A model of how the words behave and interact in a language when forming sentences
- Probabilistic Language Modeling for
  - compute the probability of a sentence

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

  - compute the probability of the upcoming word

$$P(w_4 \mid w_1, w_2, w_3)$$

- A model trained to output these quantities is a Language Model
  - In Machine Translation is adopted to rank different possible translations of a given sentence
  - In Speech Recognition is adopted to rank different transcription hypotheses

Language Models

- How to compute *P(W)*
  - Chain rule $$P(W) = P(w_1, w_2, w_3, \ldots, w_n) = \prod_i P(w_n | w_1, w_2, \ldots, w_{n-1})$$
- Ex.

  P(“*John kills Mary with a knife*”) =

  P(*John*) × P(“*kills*” |“*john*”) × P(“*Mary*”|” *kills*”, “*John*”) × P(“with”|”*Mary*”,” *kills*”, “*John*”) ….

- How to estimate these quantities?
  - count the occurrences of sequences of words
  - affected by the problem of “curse of dimensionality”
  - a sequence will be observed few times
- Traditional solution
  - adopt Markov assumption and count *n*-grams
  - P(“with”|”Mary”, “kills”, “John”) or with bi-grams P(“with”|”Mary”, “kills”,)

Neural Networks and LM

- How do LM relates to word representations?
- Parameters estimation can be done in a NN architecture
- which is expected to learn jointly:
  - the parameters of the probability function
  - a representation of the words
- the vectors representing words captures different aspects of the word meaning
  - making near in the space similar words
  - thus helping in fighting the "curse of dimensionality"

Why it should work?

- For example, given the two sentences
  - *The cat is walking in the bedroom*
  - *A dog was running in a room*
- If we know that the pairs *(cat, dog), (is,was) (walking,running), (bedroom, room)* are similar
- we could try to compute that the two sentences are similar
  - it means that we rely on the similarity of words and not on the occurrence of a specific pattern
  - this helps in fighting the curse of dimensionality

A neural probabilistic language model (Bengio et al, 2003)

- Training set is a sequence of words $w_1, \ldots, w_T$ in a vocabulary V
- The objective is to learn a mapping

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

- Decompose the function f in two components
  - A mapping C from any element i of V to a real vector $C(i) \in R^m$. It represents the *feature vectors* associated with each word in the vocabulary.
  - The probability function over words, expressed with C

The model

- A function g maps an input sequence, $(C(w_{t-n+1}),\cdots,C(w_{t-1}))$, to a conditional probability distribution over words in V for the next word w_t.

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$$

- The function g is realized through a neural network with parameters ω
- The matrix behind the C mapping is learnt during the training process
- The whole parameters set is thus (C, ω)

The model: training

- Training maximize the training corpus penalized log-likelihood

$$L = \frac{1}{T}\sum_t \log f(w_t, w_{t-1}, \dots, w_{t-n+1}; \theta) + R(\theta)$$

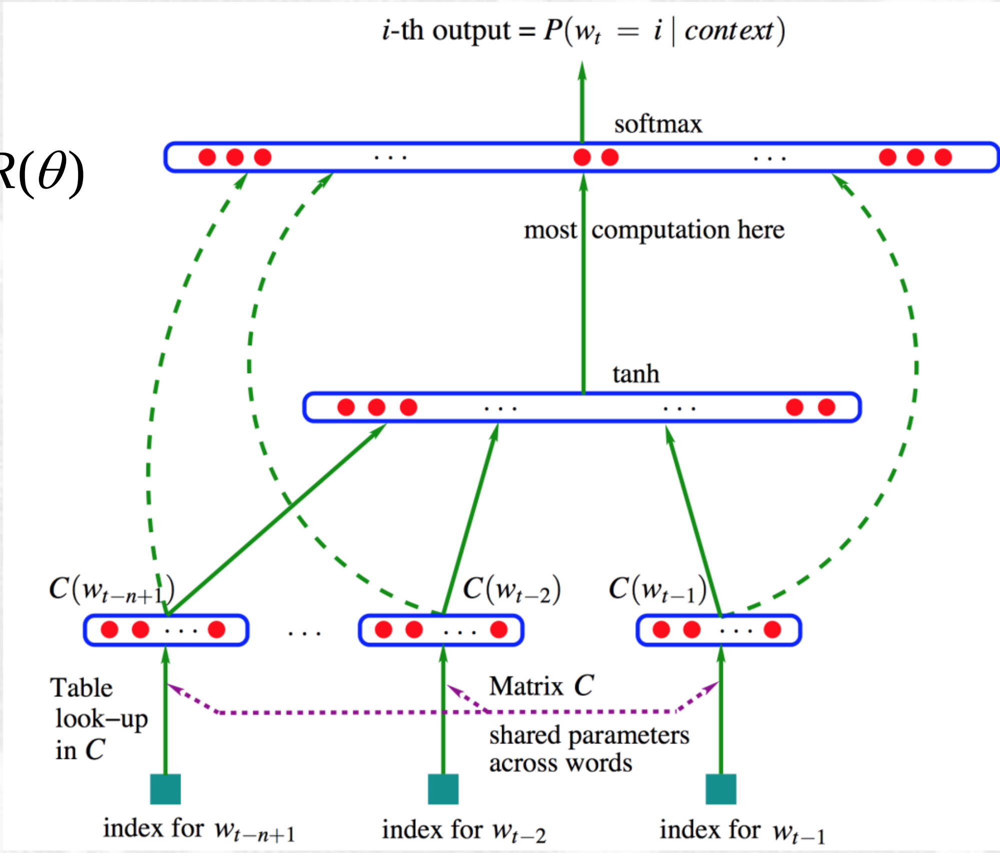
- How the probabilities in the output layer are computed?

$$P(w_t \mid w_{t-1}, \dots, w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

- Where

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

$$x = (C(w_{t-1}), C(w_{t-2}), \dots, C(w_{t-n+1}))$$

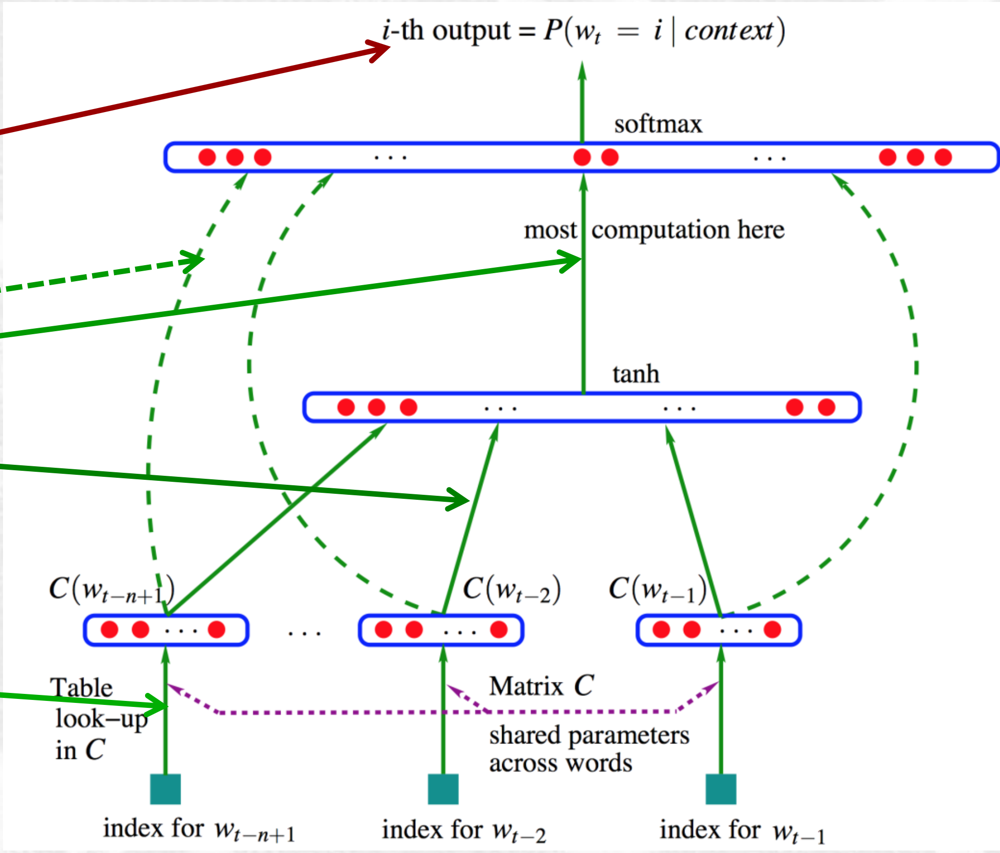


The model: details

$$P(w_t \mid w_{t-1},\dots,w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

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

$$x = (C(w_{t-1}), C(w_{t-2}), \dots, C(w_{t-n+1}))$$



- The whole set of learned parameters are then

$$\theta = (b, d, W, U, H, C)$$

What about co-occurrences?

- In previous lessons we studied co-occurrence based models
- We have seen that co-occurrences modeling works very well to generalize the meaning of words in compact vector representations

A co-occurrence matrix

	and::CC R	and::CC L	a::DT R	a::DT L	verb::N R	verb::N L	be::V R	be::V L	class::N R	of::IN R	class::N L	of::IN L	lexicon::N R	verbnet::N L	vn::N R	vn::N L	syntactic::J R	syntactic::J L
and::CC:	0	0	0	0	0	0	0	0	0	0,142	0	0,142	0	0	0	0	0	0,253
a::DT:	0	0	0	0	0	0	0	0,155	0,155	0	0	0,210	0	0	0	0	0,210	0
verb::N:	0	0	0	0	0	0	0	0	0,244	0	0	0	0,302	0	0	0	0	0
be::V:	0	0	0,174	0	0	0	0	0	0	0	0	0	0	0,255	0	0,255	0	0
of::IN:	0,147	0,147	0,219	0	0	0	0	0	0,180	0	0	0	0	0	0	0	0,237	0
class::N:	0	0	0,000	0,184	0	0,271	0	0	0	0	0	0,205	0	0	0,271	0	0	0
the::DT:	0	0	0	0	0	0	0	0,214	0	0	0	0	0	0	0	0	0	0
to::TO:	0	0	0	0	0	0	0	0	0	0	0	0,200	0	0	0	0	0,256	0
in::IN:	0	0	0,295	0	0	0,320	0,320	0	0	0	0,320	0	0	0	0,397	0	0	0
xtag::N:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
lexicon::N:	0	0	0	0	0	0,331	0	0	0	0	0	0	0	0	0	0	0	0
syntactic::J:	0,344	0	0	0,289	0	0	0	0	0	0	0	0,313	0	0	0	0	0	0
with::IN:	0	0	0,259	0	0	0,280	0	0	0	0	0	0	0	0	0	0	0	0
semantic::J:	0	0,304	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,343

What about co-occurrences?

- We have seen that co-occurrences modeling works very well to generalize the meaning of words in compact vector representations
- Can we think a NN modeling how the language works and jointly accounting for the co-occurrences?
  - YES

CBOW and Skip-gram (Mikolov et al, 2013)

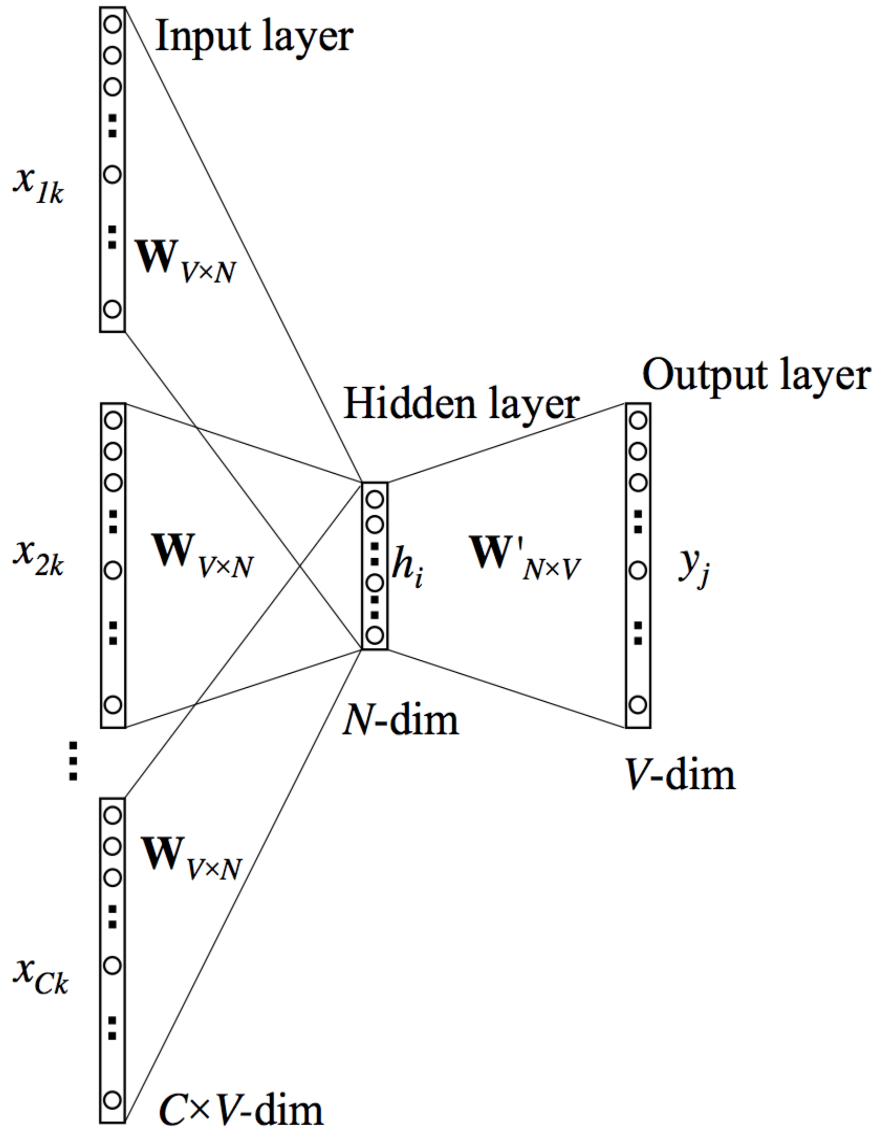
- Mikolov and colleagues proposed two NN based models that accounts for co-occurrences
  - in the learning of word vectors
- CBOW (Contextual Bag-Of-Word)
  - model the co-occurrences in the input to a neural network
- Skip-gram
  - model the co-occurrences in the output of a neural network

CBOW

- Contextual Bag-of-Words model
- Given a context predict the word within that context
- Each word is represented with a distributed representation
  - a *d*-dimensional vector
- The learning process makes similar the representations of similar words
- How?

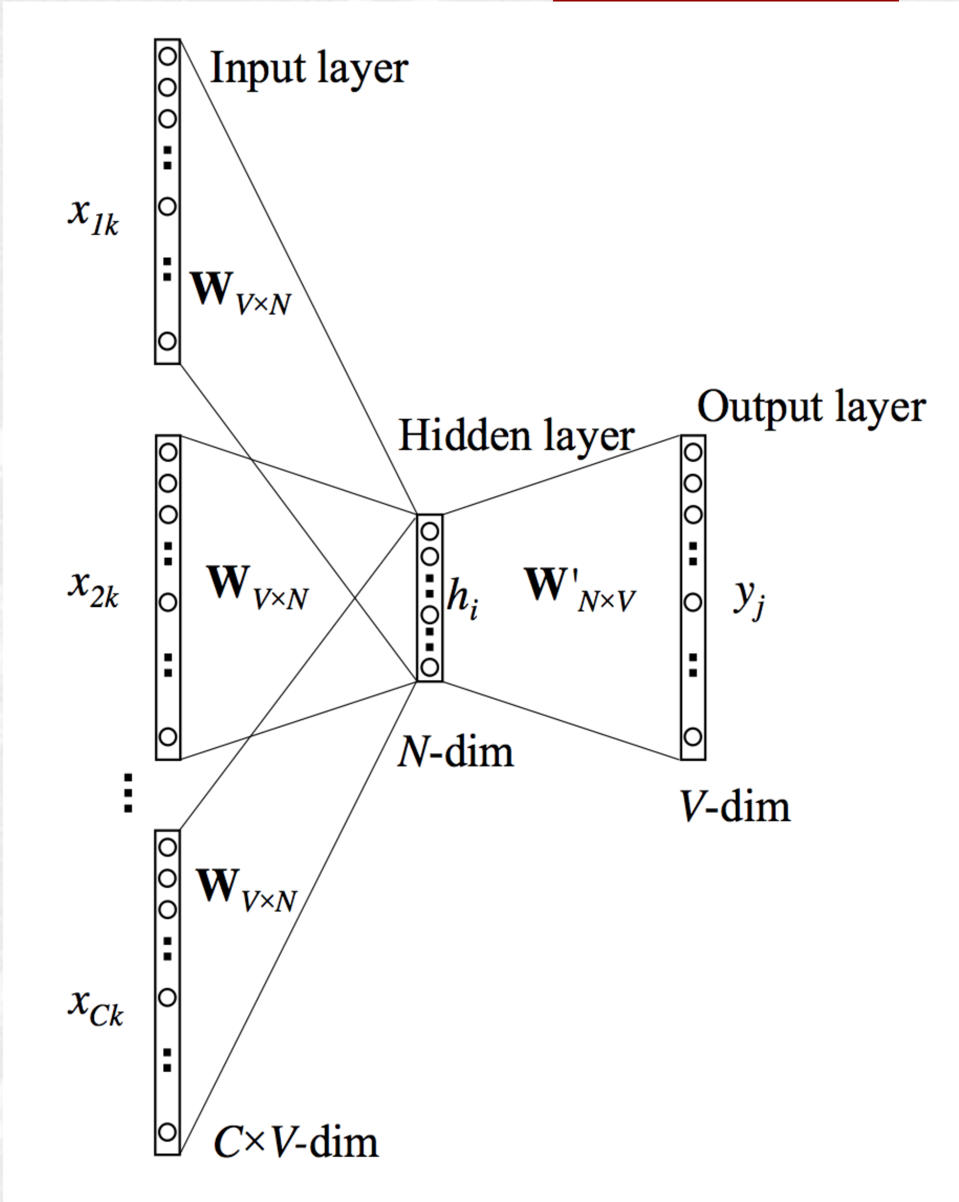
CBOW architecture

- $x_{1k}, \ldots, x_{Ck}$ is a context
  - each x_{ij} is mapped into a vector
  - the vectors are contained in the matrix W (as rows)
- h_i maps the input context into a hidden compact representation
  - in this case is the mean of the context vectors
- in the output layer the network is expected to compute a probability distribution
  - the probability of the correct word in a context should be higher



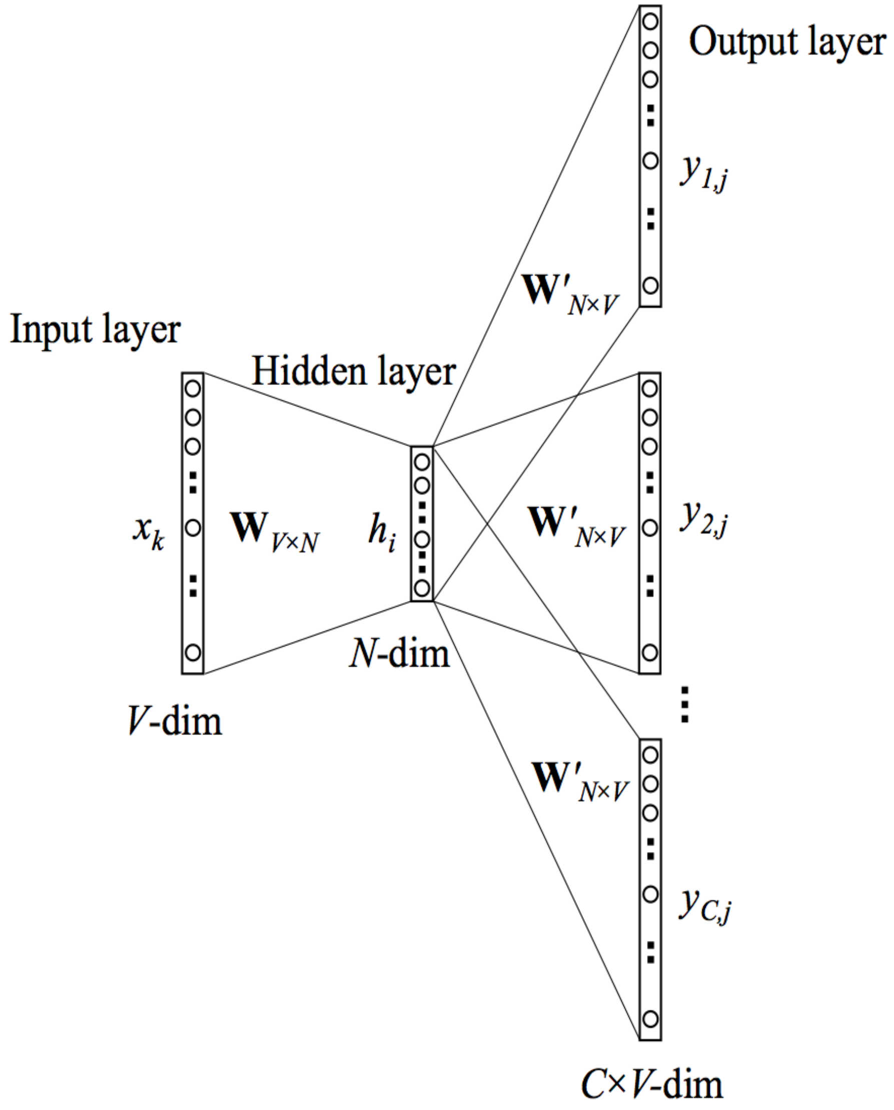
CBOW architecture

- The matrix containing the word vectors (W) are learned during the training of the network
- If two words share many contexts during training the representations of these words will be similar
  - as their similar contexts will be forced to reconstruct either one or the other
- The training process will be directed to optimizing the log-likelihood of recovering the correct y_j given its context.



Skip-gram

- The same principle as CBOW, but
- the input layer contains a word w_i
- in the output layer will be predicted the context words of w_i
- Again, the word vectors are learned during training
- The training process will maximize the log-likelihood of recovering the correct context given a target word
  - On the output layer, we are outputting C distributions
  - Each output is computed using the same hidden → output matrix



Skip-gram details

- After a forward step, in the output layer we want to obtain the probability distribution of the context words

$$p(w_{c,j} = w_{O,c} \mid w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'} \exp(u_{j'})}$$

  - $w_{c,j}$ is the j-th word on the c-th panel
  - $w_{O,c}$ is the actual c-th word in the context
  - w_I is the input word
  - $y_{c,j}$ is the output of the j-th unit on the c-th panel
  - $u_{c,j}$ is the net input of the j-th unit on the c-th panel

- The objective function is thus the probability of recovering all the context words given the target

$$E = -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,c} \mid w_I) = -\log \prod_c \frac{\exp(u_{c,j})}{\sum_{j'} \exp(u_{j'})}$$

Skip-gram and CBOW

- CBOW model averages over the context in the input; it “smooths” the original distributional statistics
  - it is a sort of regularization, as the model learns from a “corrupted” input
- The Skip-gram model does not; it needs more data but it doesn’t modify the input
  - given that you have enough data, the Skip-gram model generally learns better vectors
- Both learns word vectors as a supervised process
  - however the input are raw texts, i.e. there is no need of a real supervision!
- They can be implemented very efficiently, and can produce word vectors starting from corpora of million of words
  - a couple of optimization techniques makes the learning process very fast.

Speed optimizations

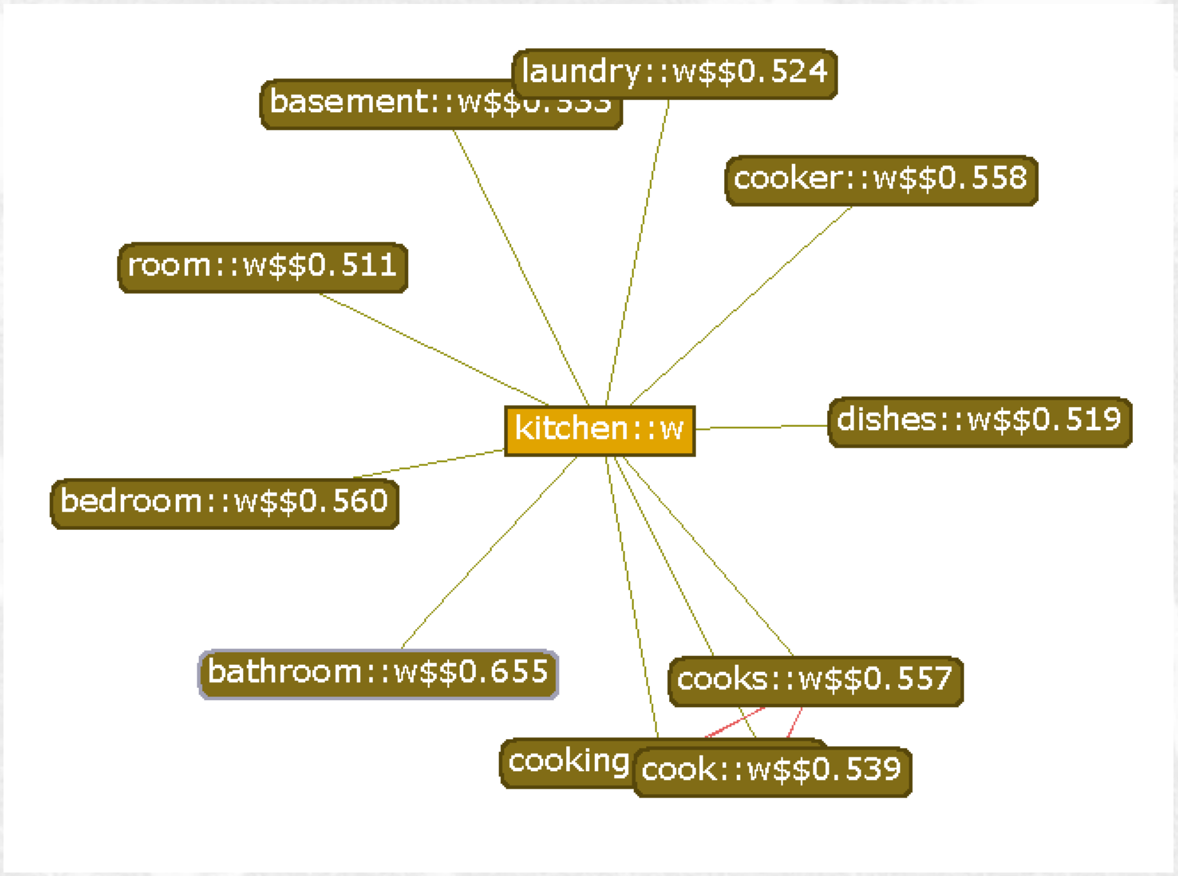
- Are meant to avoid the full computation/update of parameters at each iteration
- **Hierarchical Softmax**
  - it's a technique to avoid the full computation of the output layer (which can potentially contain millions of neurons)
- The hierarchical softmax uses a binary tree representation of the output layer
  - the words in the vocabulary are the leaves
  - for each leaf, there exists a unique path from the root to the unit
  - this path is used to estimate the probability of the word represented by the leaf unit

Speed optimizations

- **Negative sampling**
  - in the softmax operation we should compute the output vectors for all the words in the vocabulary (the denominator)
  - to avoid this computation just a sampling of the words are adopted
  - This sampling is “negative”, as the chosen words are selected from the words that should not be “similar”, i.e. they are not in the context of the target in the Skip-gram model

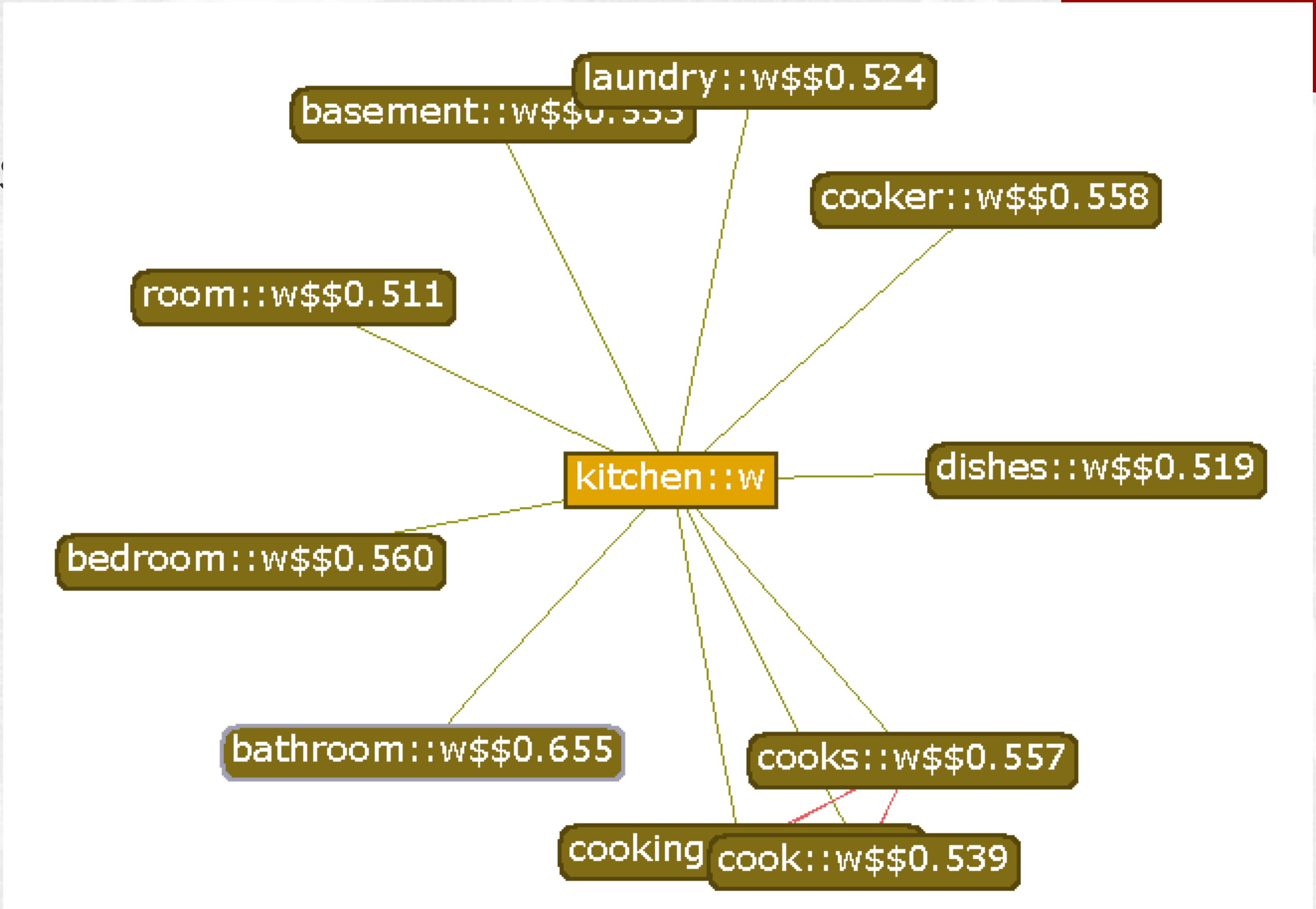
What does Skip-gram or CBOW learns?

- Semantically related words



What does Skip-gram or CBOW learns?

- S



Word Embedding Semantics

(slide from cs224n-2017-lecture3 by Socher)

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

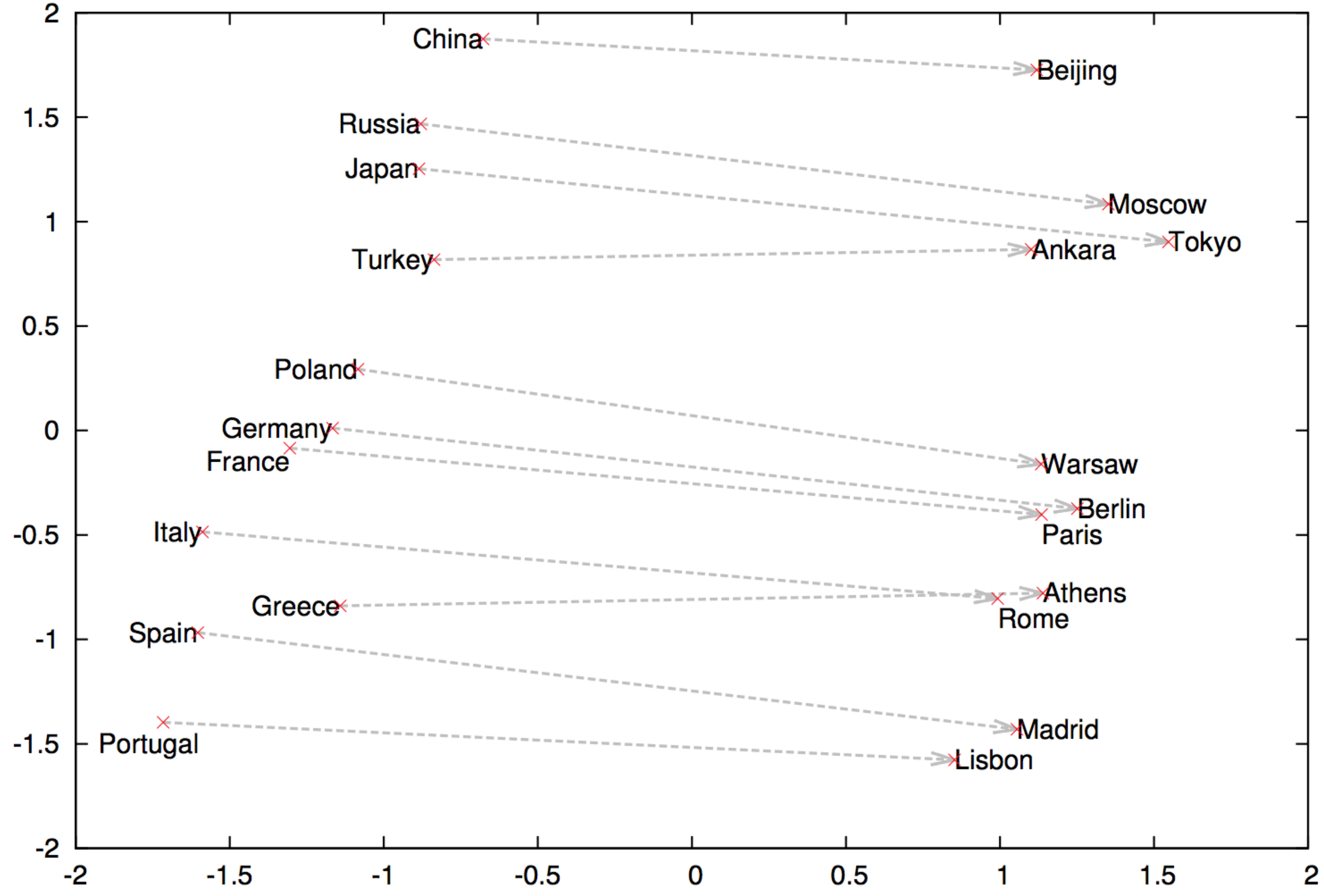
eleutherodactylus

rana eleutherodactylus

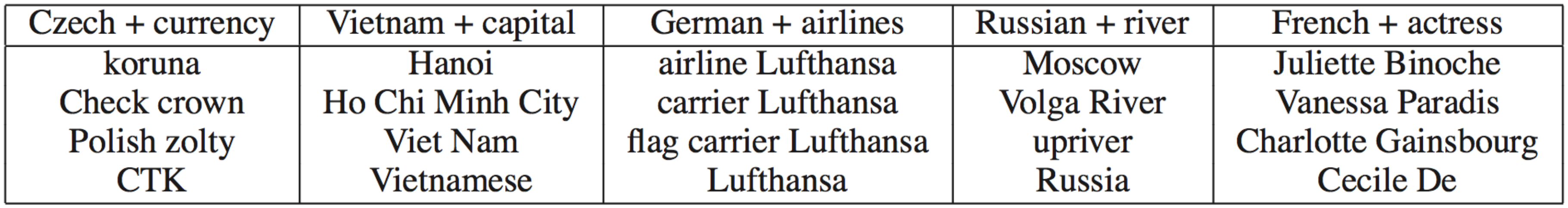
What does Skip-gram or CBOW learns?

- Other (meaningful) relationships

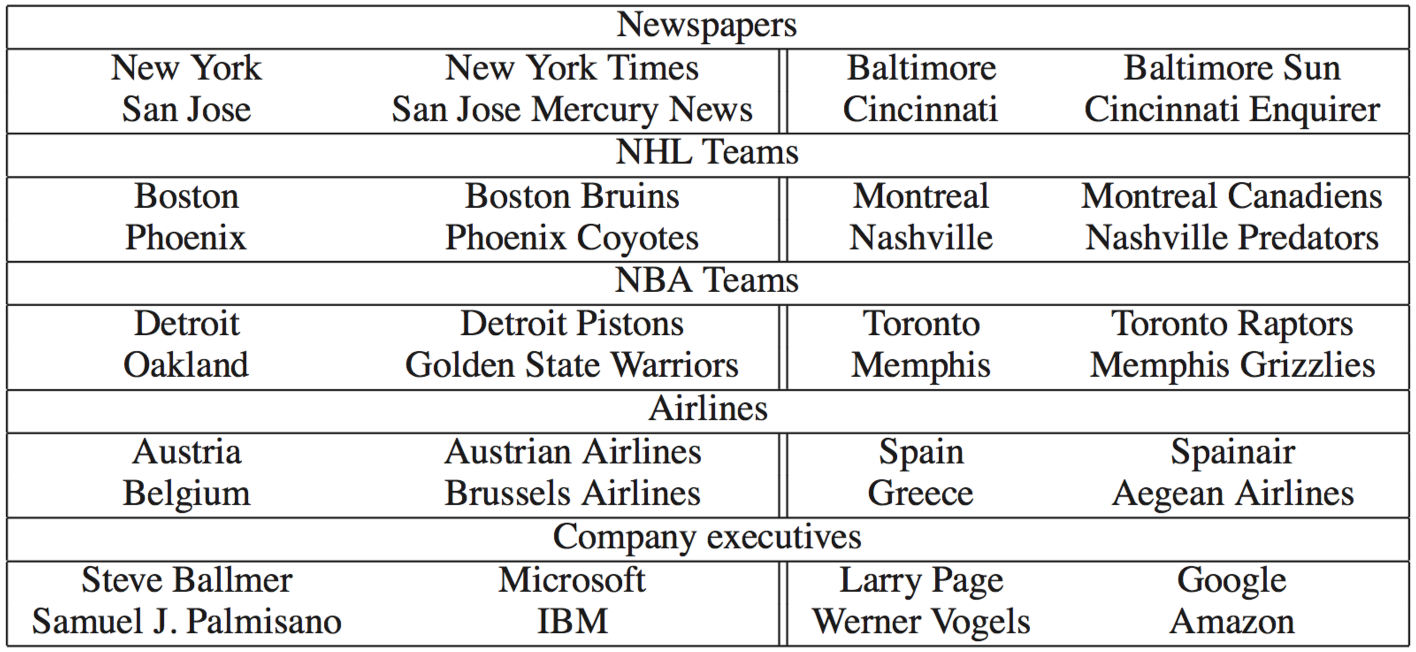
Country and Capital Vectors Projected by PCA



What does Skip-gram or CBOW learns?



Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

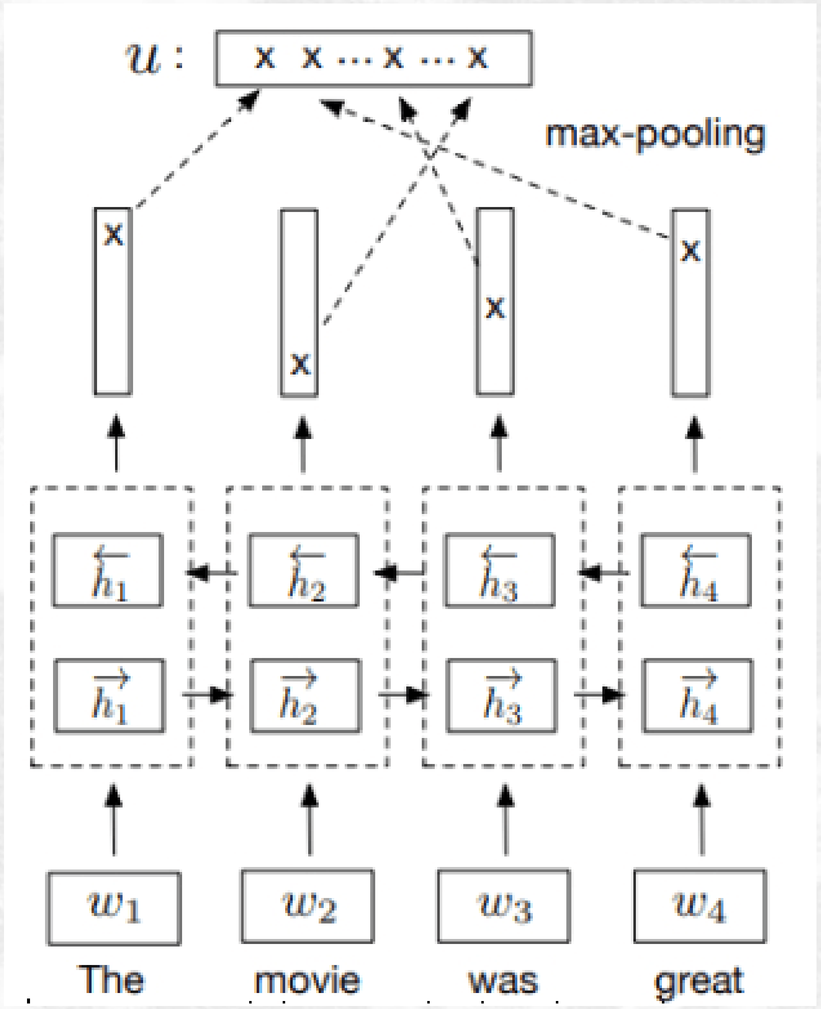


Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

What we haven't touched

- FastText: using subword information
  - https://www.aclweb.org/anthology/Q17-1010.pdf
  - https://github.com/facebookresearch/fastText
  - Embedding N-grams as features
  - Words as sequences of features
- Sentence embeddings:
  - Doc2Vec
    - Quoc Le and Tomas Mikolov: "Distributed Representations of Sentences and Documents", 2014; arXiv:1405.4053.
  - **InferSent**
    - Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault: "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data", 2017; arXiv:1705.02364.
- Language Independent embeddings
  - Neural embedding as a Multiple task learning
  - Subwords as core shared basis for multiple languages

Using word embeddings


u :
x x … x … x
max-pooling
x
x
x
x
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_4$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_4$
w_1 w_2 w_3 w_4
The movie was great

from (Conneau et al, 2017)

Recent Trends

- From word to sentence embeddings
  - Train NNs about the task of combining words to embed sentences
  - Character (instead of word) embeddings
- Contextual pretraining
  - Attempt to made embeddings better capturing differences in contextual use, aka senses
  - Multiple biLSTMs (ELMo, 2017)
- Adopting bidirectional transformers, BERT (2018)
  - Pretraining: Bidirectional Transformers for LM
  - Pretraining: Masking
  - Fine-tuning: Sentence prediction tasks

Differences in recent approaches

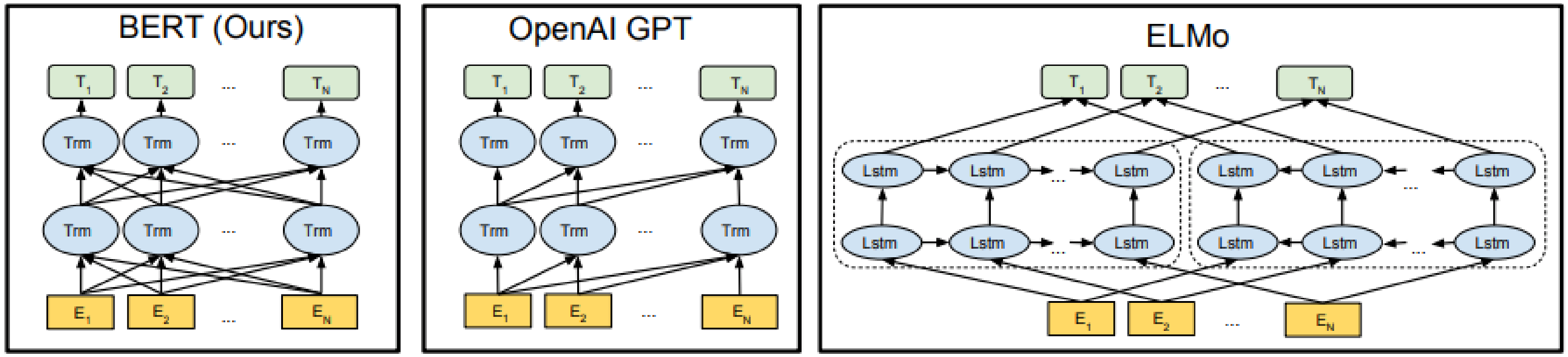


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

Summary

- Model language related problems with NN
  - fighting the curse of dimensionality with distributional representations of words
- Exploit the flexibility of Neural Networks for
  - transforming an unsupervised process into a supervised one
  - compute efficiently new representations
- The CBOW and Skip-gram models are not related to Deep Learning
  - they have nothing of a deep architecture
- However
  - they emerged in the Deep Learning "era"
  - they are adopted as a form of pre-training of Deep Architectures for NLP problems

References

- (Bengio et al, 2003): Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. J. Mach. Learn. Res. 3 (March 2003), 1137-1155.
- Mikolov, T.; Chen, K.; Corrado, G. & Dean, J. (2013), Efficient Estimation of Word Representations in Vector Space, CoRR abs/1301.3781.
- Tomas Mikolov, Wen-tau Yih, Geoffrey Zweig: Linguistic Regularities in Continuous Space Word Representations. HLT-NAACL 2013: 746-751
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, Jeffrey Dean: Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013: 3111-3119
- Word2Vec parameters learning explained

References (2)

CHARACTER EMBEDDINGS:

- **CNN character embedding layer** - Character-Aware Neural Language Models, Kim et al. 2015

SUBWORD EMBEDDINGS:

- **FastText** - Enriching Word Vectors with Subword Information, Bojanowski et al. 2017
- **WordPiece** - Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, Wu et al. 2016

CONTEXTUALIZED WORD EMBEDDINGS

- **ELMo** - Deep contextualized word representations, Peters et al. 2018
- **CoVe** - Learned in Translation: Contextualized Word Vectors, McCann et al. 2017

PRE-TRAINED DEEP LEARNING ARCHITECTURES:

- **BERT** - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. 2018
- **OpenAI GPT** – Improving language understanding with unsupervised learning, Radford et al. 2018
- **Transformer** - Attention Is All You Need, Vaswani et al. 2017