POLITECNICO MILANO 1863

Credits for images and examples to Elena Voita's

NLP Course | For You

https://lena-voita.github.io/nlp_course.html

Artificial Neural Networks and Deep Learning

- Seq2seq and Word Embedding-

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)

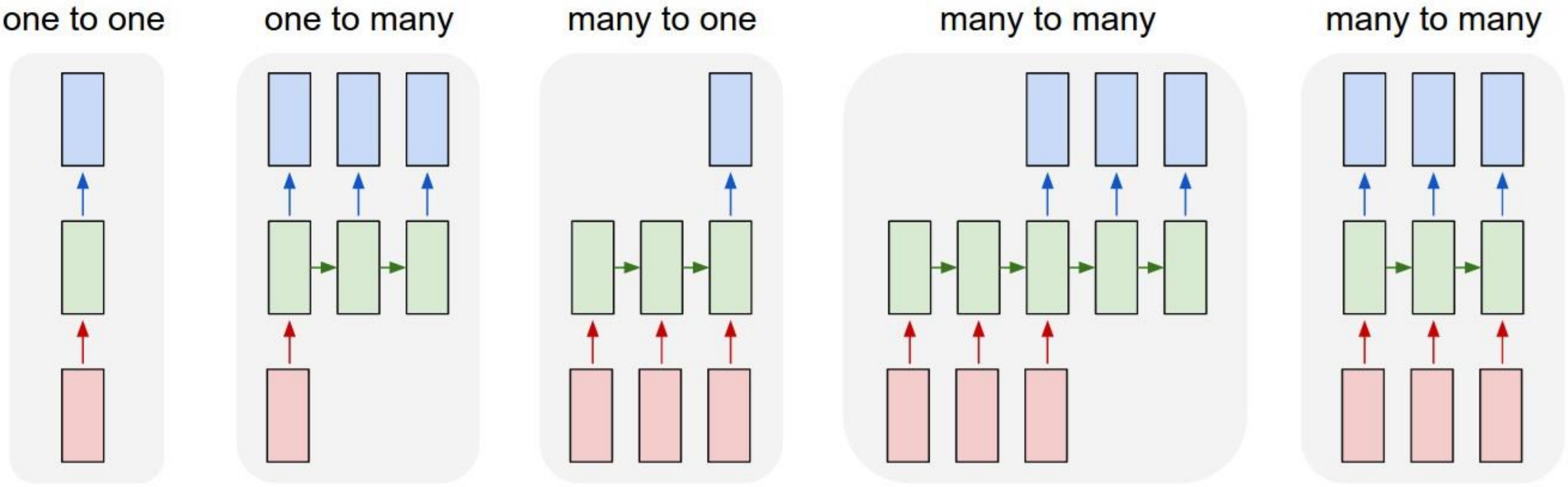
Artificial Intelligence and Robotics Laboratory

Politecnico di Milano

AIRLAB

ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

Sequential data problems



Fixed-sized input to fixed-sized output (e.g. image classification)

Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

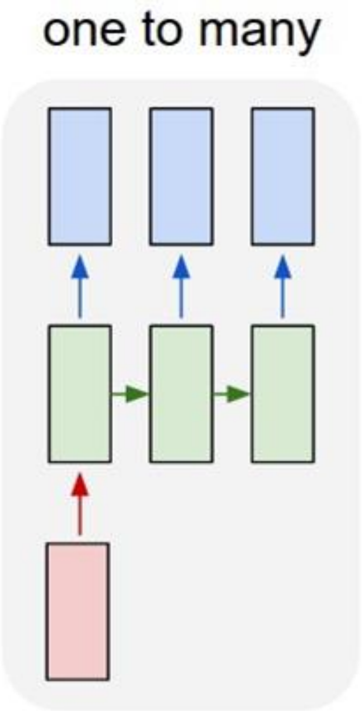
Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)

Synced sequence input and output (e.g. video classification where we wish to label each frame of the video)

The Unreasonable Effectiveness of Recurrent Neural Networks: [*http://karpathy.github.io/2015/05/21/rnn-effectiveness/*](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Sequence to Sequence Learning Examples (1/3)

Image Captioning: input a single image and get a series or sequence of words as output which describe it. The image has a fixed size, but the output has varying length.



A person riding a motorcycle on a dirt road.

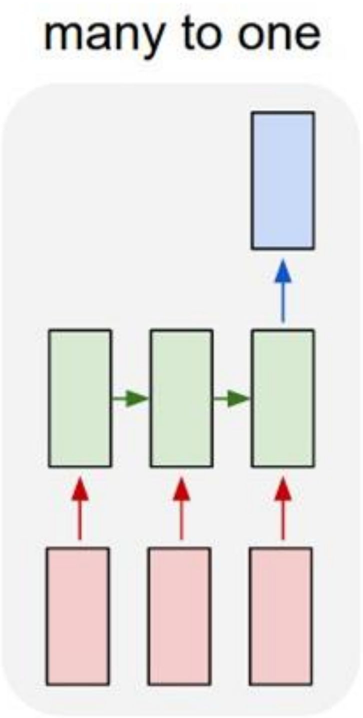
Two dogs play in the grass.

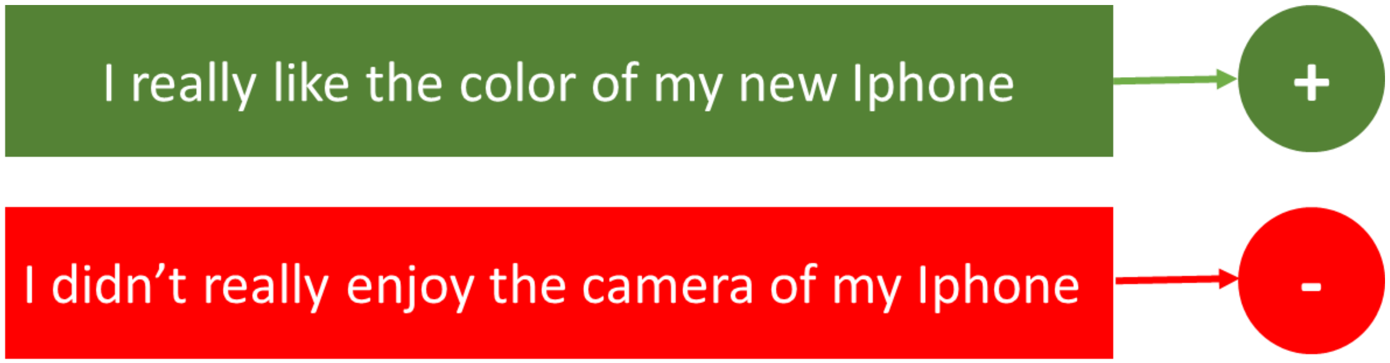
A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

Sequence to Sequence Learning Examples (2/3)

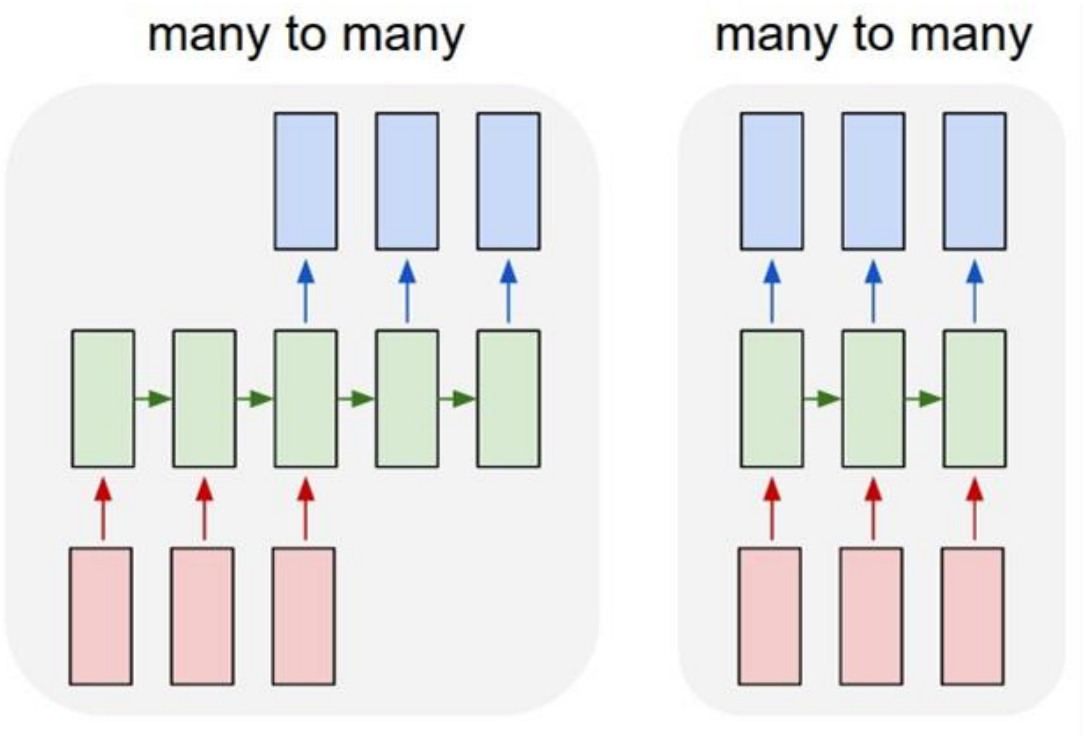
Sentiment Classification/Analysis: input a sequence of characters or words, e.g., a tweet, and classify the sequence into positive or negative sentiment. Input has varying lengths; output is of a fixed type and size.

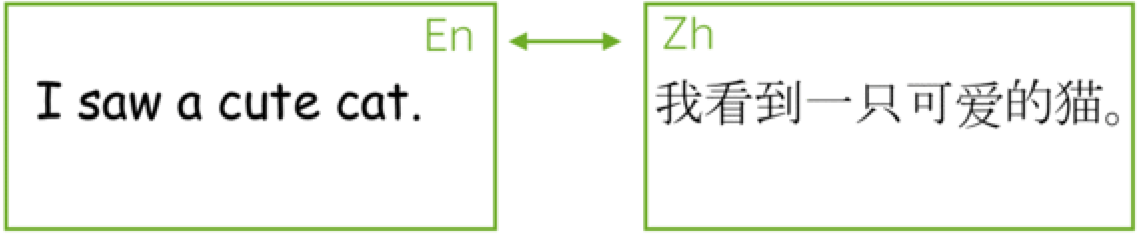




Sequence to Sequence Learning Examples (3/3)

__Language Translation__: having some text in a particular language, e.g., English, we wish to translate it in another, e.g., French. Each language has its own semantics and it has varying lengths for the same sentence.





How can we model language translation?

Conditional Language Models

Language model represents the probability of a sentence (sequence)

$$P(y_1, y_2, \dots, y_n) = \prod_{t=1}^{n} p(y_t | y_{<t})$$

Conditional language model conditions on a source sentence (sequence)

$$P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_m) = \prod_{t=1}^{n} p(y_t | y_{<t}, x_1, x_2, \dots, x_m)$$

In image captioning $x_1 x_2, \dots, x_m$ can be replaced by an image x

Sequence to Sequence Basics

Given an input sequence

$$x_1, x_2, \dots, x_m$$

and a target output sequence

$$y_1, y_2, \dots, y_n$$

En
I saw a cute cat.

Zh
我看到一只可爱的猫。

we aim the sequence which maximizes the conditional probability $\mathrm{P(y|x)}$

$$y^* = \operatorname*{argmax}_y P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_m)$$

in sequence-to-sequence modeling, we learn from data a model $\mathrm{P(y|x}, \theta)$ and our prediction now becomes

$$y' = \operatorname*{argmax}_y P(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_m, \theta)$$

Machine Translation Humans vs Machines

Human Translation

$$y^* = \arg\max_y p(y|x)$$

The "probability" is intuitive and is given by a human translator's expertise

Machine Translation Humans vs Machines

Human Translation

$$y^* = \arg\max_y p(y|x)$$

The "probability" is intuitive and is given by a human translator's expertise

Machine Translation

model parameters

$$y' = \arg\max_y p(y|x,\theta)$$

Questions we need to answer

- **modeling**

  How does the model for $p(y|x,\theta)$ look like?

- **learning**

  How to find θ?

- **search**

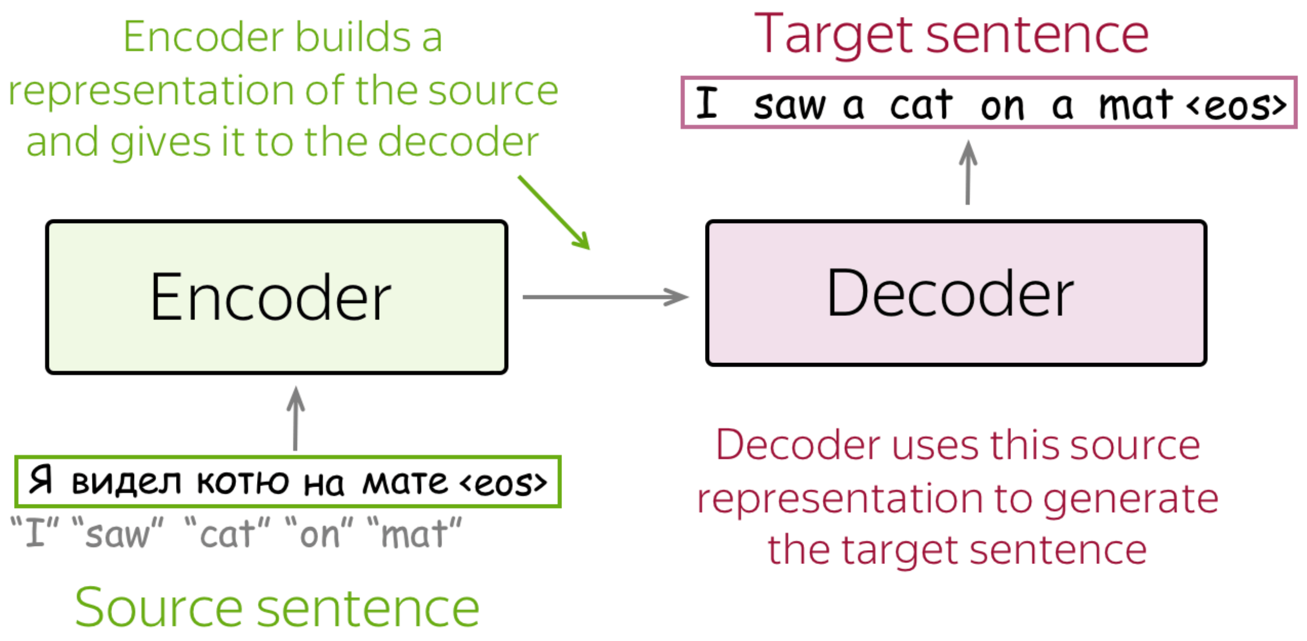
  How to find the argmax?

We will just mention about these!

The Encoder-Decoder Framework

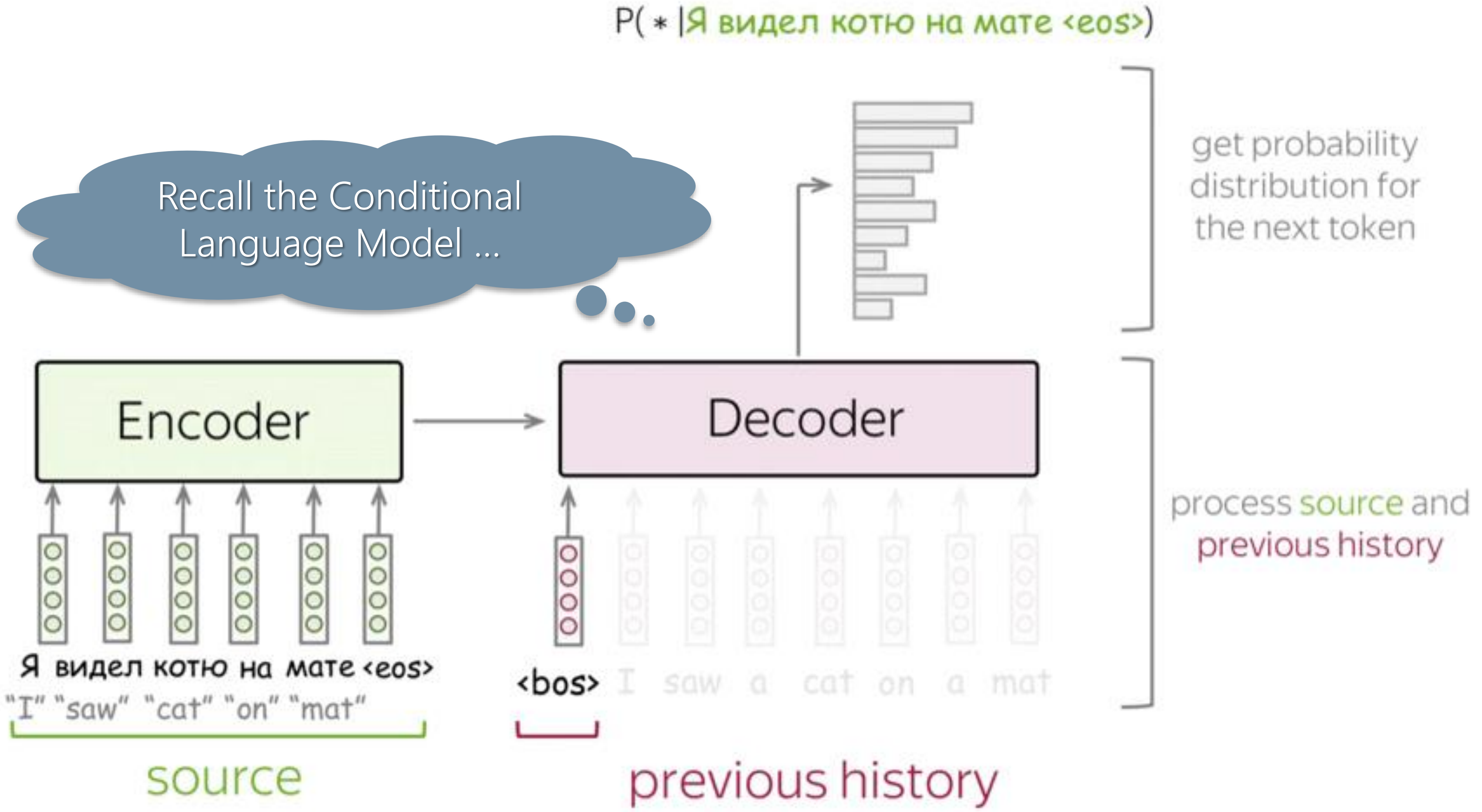
True for most of deep learning models …

Sequence-to-sequence models as encoder-decoder architectures



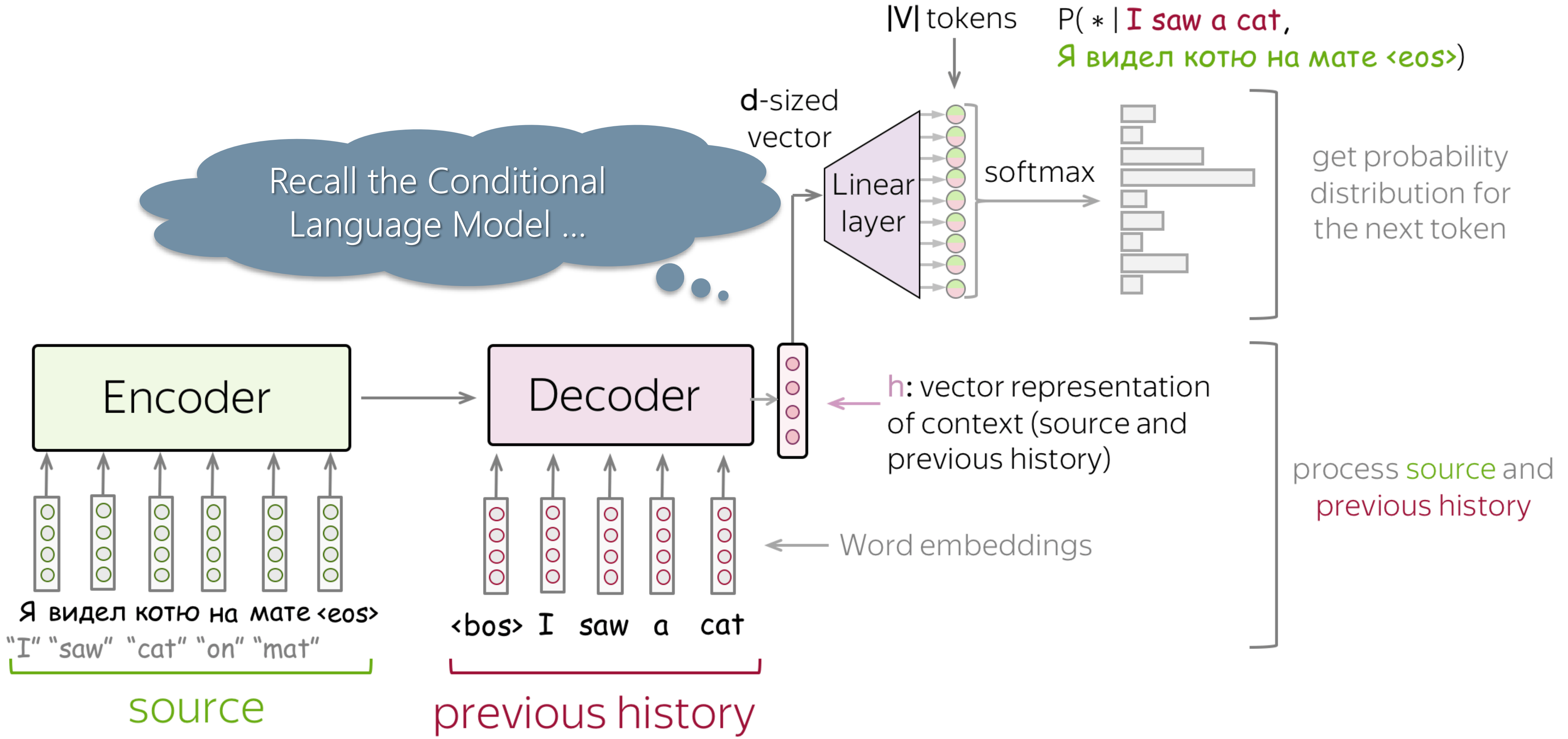
The Encoder-Decoder Framework

Sequence-to-sequence models as encoder-decoder architectures

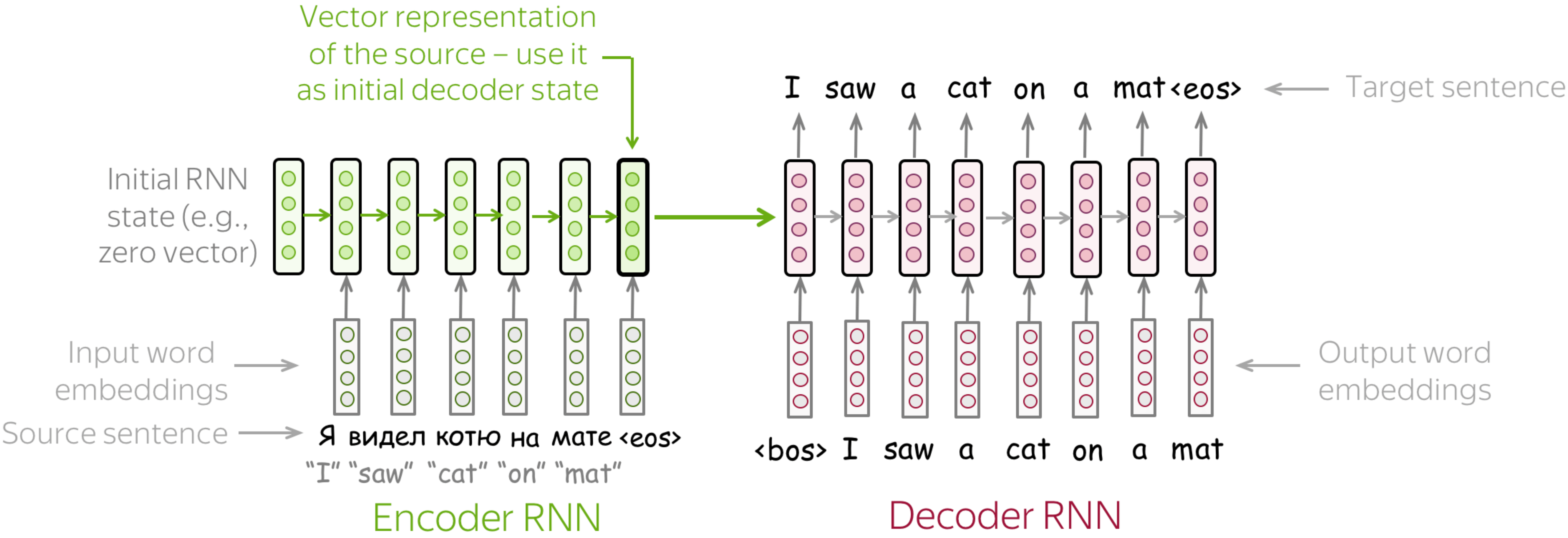


The Encoder-Decoder Framework

Sequence-to-sequence models as encoder-decoder architectures



Encoding/Decoding with Recurrent Neural Networks (LSTM)



Greedy Decoding vs Beam Search

Once we have trained the mode, i.e., learned $\boldsymbol{\theta}$, we predict a sequence $y = y_1, y_2, \dots, y\_n$ given $x = x_1, x_2, \dots, x_m$ by selecting y' as

$$y' = \underset{y}{\operatorname{argmax}} \prod_{t=1}^{n} P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta)$$

To compute the argmax over all possible sequences we can use:

- *Greedy Decoding*: at each step, pick the most probable token

$$y' = \underset{y}{\operatorname{argmax}} \prod_{t=1}^{n} P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta) \approx \prod_{t=1}^{n} \underset{y_t}{\operatorname{argmax}} P(y_t | y_{<t}, x_1, x_2, \dots, x_m, \theta)$$

but this does not guarantee to reach the best sequence and it does not allow to backtrack from errors in early stages of classification.

Greedy Decoding vs Beam Search

Once we have trained the mode, i.e., learned $\boldsymbol{\theta}$, we predict a sequence $\boldsymbol{y} = \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}\_\boldsymbol{n}$ given $\boldsymbol{x} = \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m$ by selecting $\boldsymbol{y}'$ as

$$y' = \underset{y}{\operatorname{argmax}} \prod_{t=1}^{n} P(y_t | y_{<t}, x_1, x_2, \ldots, x_m, \theta)$$

To compute the argmax over all possible sequences we can use:

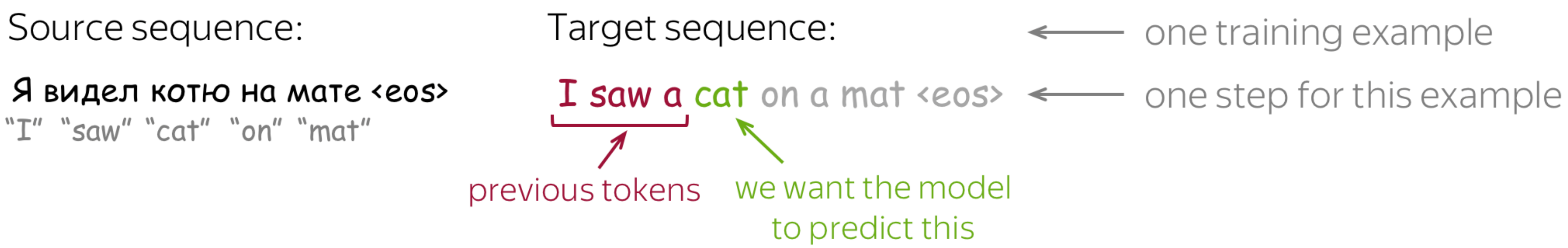
- <u>*Beam Search*</u>: *Keep track of several most probably hypotheses*

$\boxed{\text{<bos>}}$

Start with the begin of sentence token or with an empty sequence

Training Sequence to Sequence Models

Given a training sample $< x, y >$ with input sequence $x = x_1, x_2, \dots, x_m$ and target sequence $y = y_1, y_2, \dots, y_{n'}$



time t our model predicts

$$p_t = p(\cdot \mid y_1, y_2, \dots, y_{t-1}, x_1, x_2, \dots, x_m)$$

Using a one-hot vector for y_t we can use the cross-entropy as loss

$$loss_t(p_t, y_t) = -\sum_{i=1}^{|V|} y_t^{(i)} \log\left(p_t^{(i)}\right) = -y_t^T \log(p_t)$$

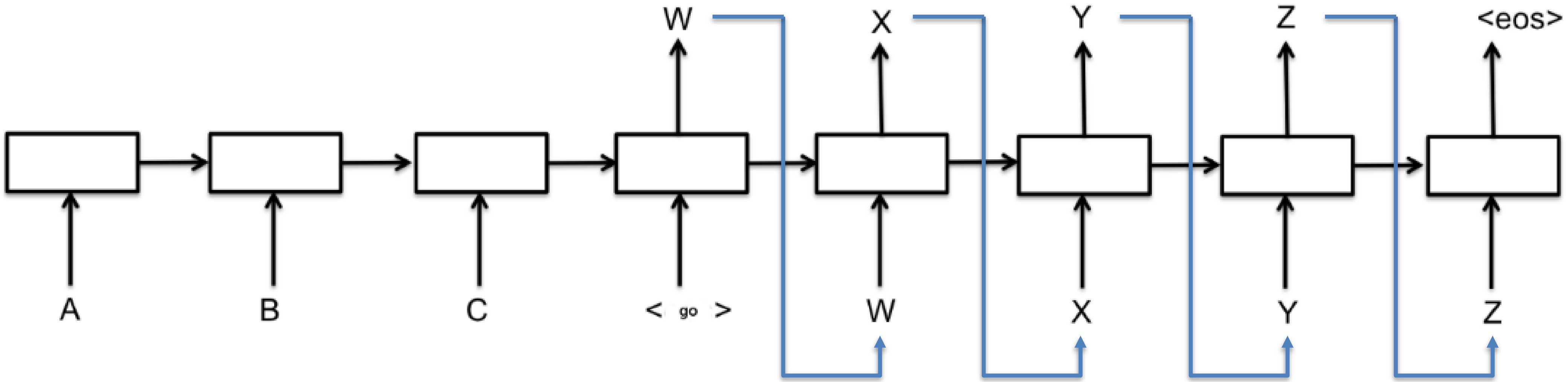
Training Sequence to Sequence Models

Over the entire sequence cross-entropy becomes $-\sum_{t=0}^{n} y_t^T \log(p_t)$

Training Sequence to Sequence Models

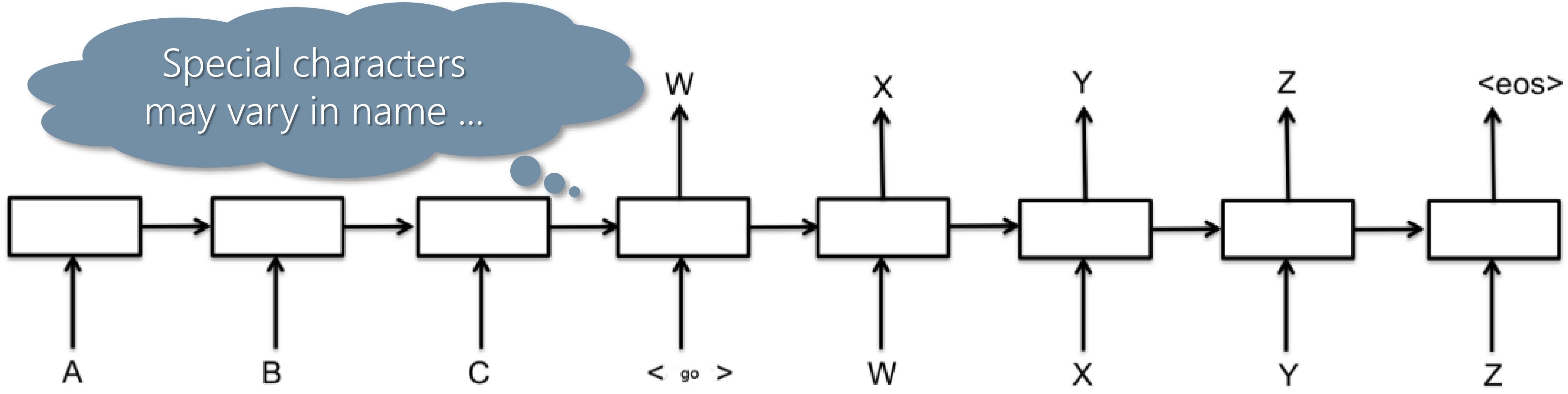
Seq2Seq model follows a classical encoder decoder architecture

- At trainng time the decoder **does not** feed the output of each time step to the next; the input to the decoder time steps are the target from the training
- At inference time the decoder feeds the output of each time step as an input to the next one



Sequence to sequence learning with Neural networks: https://arxiv.org/pdf/1409.3215.pdf

Special Characters



<PAD>: During training, examples are fed to the network in batches. The inputs in these batches need to be the same width. This is used to pad shorter inputs to the same width of the batch

<EOS>: Needed for batching on the decoder side. It tells the decoder where a sentence ends, and it allows the decoder to indicate the same thing in its outputs as well.

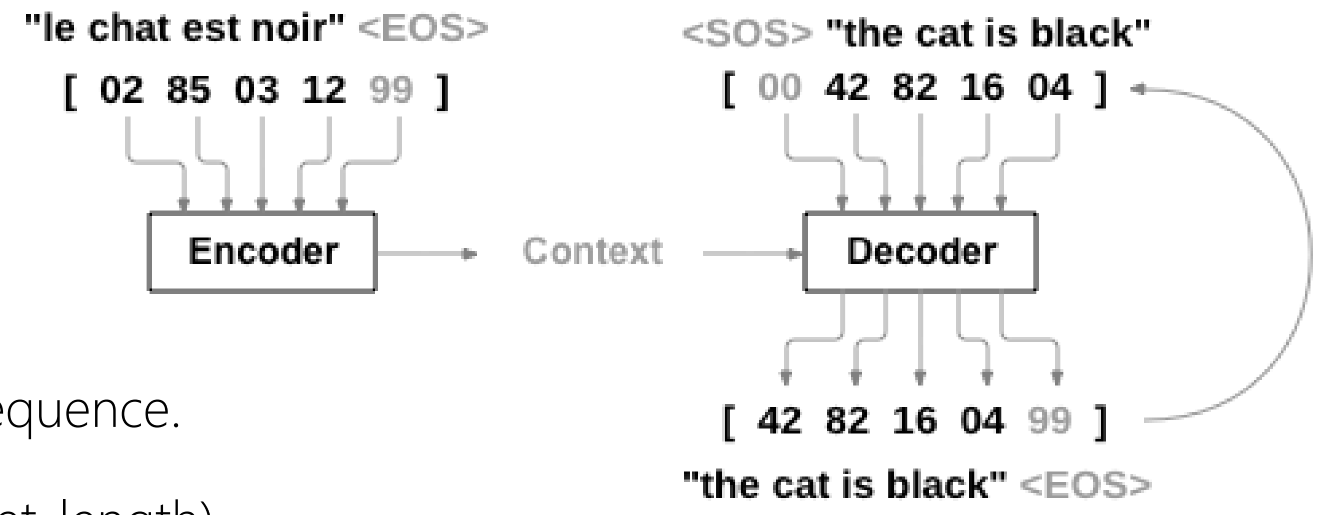
<UNK>: On real data, it can vastly improve the resource efficiency to ignore words that do not show up often enough in your vocabulary by replace those with this character.

<SOS>/<GO>: This is the input to the first time step of the decoder to let the decoder know when to start generating output.

Sequence to sequence learning with Neural networks: https://arxiv.org/pdf/1409.3215.pdf

Dataset Batch Preparation

1. Sample batch_size pairs of (source_sequence, target_sequence).
2. Append <EOS> to the source_sequence
3. Prepend <SOS> to the target_sequence to obtain the target_input_sequence and append <EOS> to obtain target_output_sequence.
4. Pad up to the max_input_length (max_target_length) within the batch using the <PAD> token.
5. Encode tokens based of vocabulary (or embedding)
6. Replace out of vocabulary (OOV) tokens with <UNK>. Compute the length of each input and target sequence in the batch.




```
Vocabulary = {"<SOS>": 00,
              "<EOS>": 99,
              "<UNK>": 01,
              "<PAD>": 03,
              "the": 42,
              "is": 16,
               ...      }
```

Where do these numbers come from?

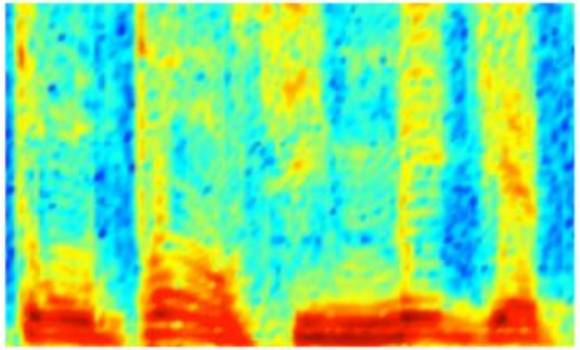
Word Embedding Motivation

Natural language processing treats words as discrete atomic symbols

- 'cat' is encoded as Id537
- 'dog' is encoded as Id143
- …

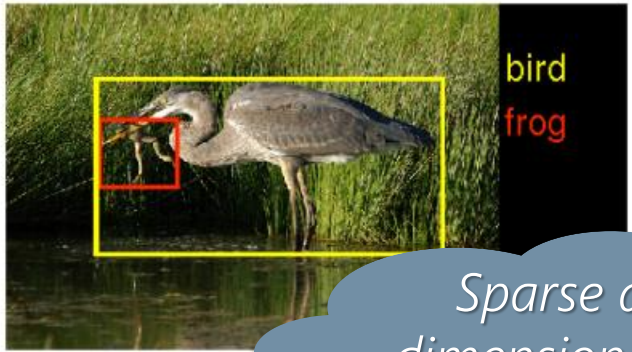
Items in a dictionary …

A document becomes a Bag of Words



Audio Spectrogram

DENSE



Sparse and high dimensional -> Curse of Dimensionality!

Image pixels

DENSE

0	0	0	0.2	0	0.7	0	0	0	...	...

Word, context, or document vectors

SPARSE

Encoding Text is a Serious Thing

Performance of real-world applications (e.g., chatbot, document classifiers, information retrieval systems) depends on input encoding:

Local representations

- N-grams —Language Model→
- Bag-of-words
- 1-of-N coding

Continuous representations

- Latent Semantic Analysis
- Latent Dirichlet Allocation
- Distributed Representations

Determine $P(s = w_1, \ldots, w_k)$ in some domain of interest

$$P(s_k) = \prod_i^k P(w_i | w_1, \ldots, w_{i-1})$$

In traditional n-gram language models "the probability of a word depends only on the context of n−1 previous words"

$$\hat{P}(s_k) = \prod_i^k P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$

Typical ML-smoothing learning process (e.g., Katz 1987):

- compute $\hat{P}(w_i | w_{i-n+1}, \ldots, w_{i-1}) = \frac{\#w_{i-n+1}, \ldots, w_{i-1}, w_i}{\#w_{i-n+1}, \ldots, w_{i-1}}$
- smooth to avoid zero probabilities

N-gram Language Model: Curse of Dimensionality

Let's assume a 10-gram LM on a corpus of 100.000 unique words

- The model lives in a 10D hypercube where each dimension has 100.000 slots
- Model training ↔ assigning a probability to each of the 100.000^{10} slots
- *Probability mass vanishes* → more data is needed to fill the huge space
- The more data, the more unique words! → Is not going to work …

In practice:

- Corpuses can have 10^6 unique words
- Contexts are typically limited to size 2 (trigram model),
  e.g., famous Katz (1987) smoothed trigram model
- With short context length a lot of information is not captured

N-gram Language Model: Word Similarity Ignorance

Let assume we observe the following similar sentences

- *Obama speaks to the media in Illinois*
- *The President addresses the press in Chicago*

With classic one-hot vector space representations

- speaks = [0 0 1 0 … 0 0 0 0]
- addresses = [0 0 0 0 … 0 0 1 0]
- obama = [0 0 0 0 … 0 1 0 0]
- president = [0 0 0 1 … 0 0 0 0]
- illinois = [1 0 0 0 … 0 0 0 0]
- chicago = [0 1 0 0 … 0 0 0 0]

speaks ⊥ addresses

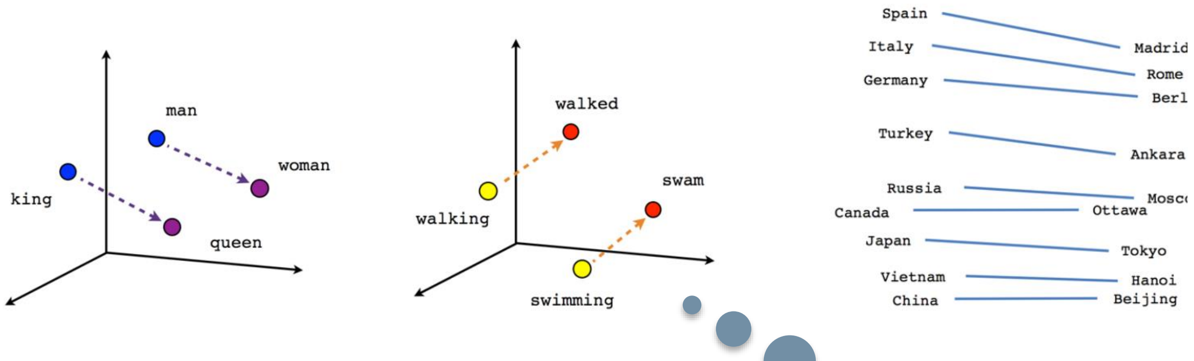
obama ⊥ president

illinois ⊥ chicago

Word pairs share no similarity, and we need word similarity to generalize

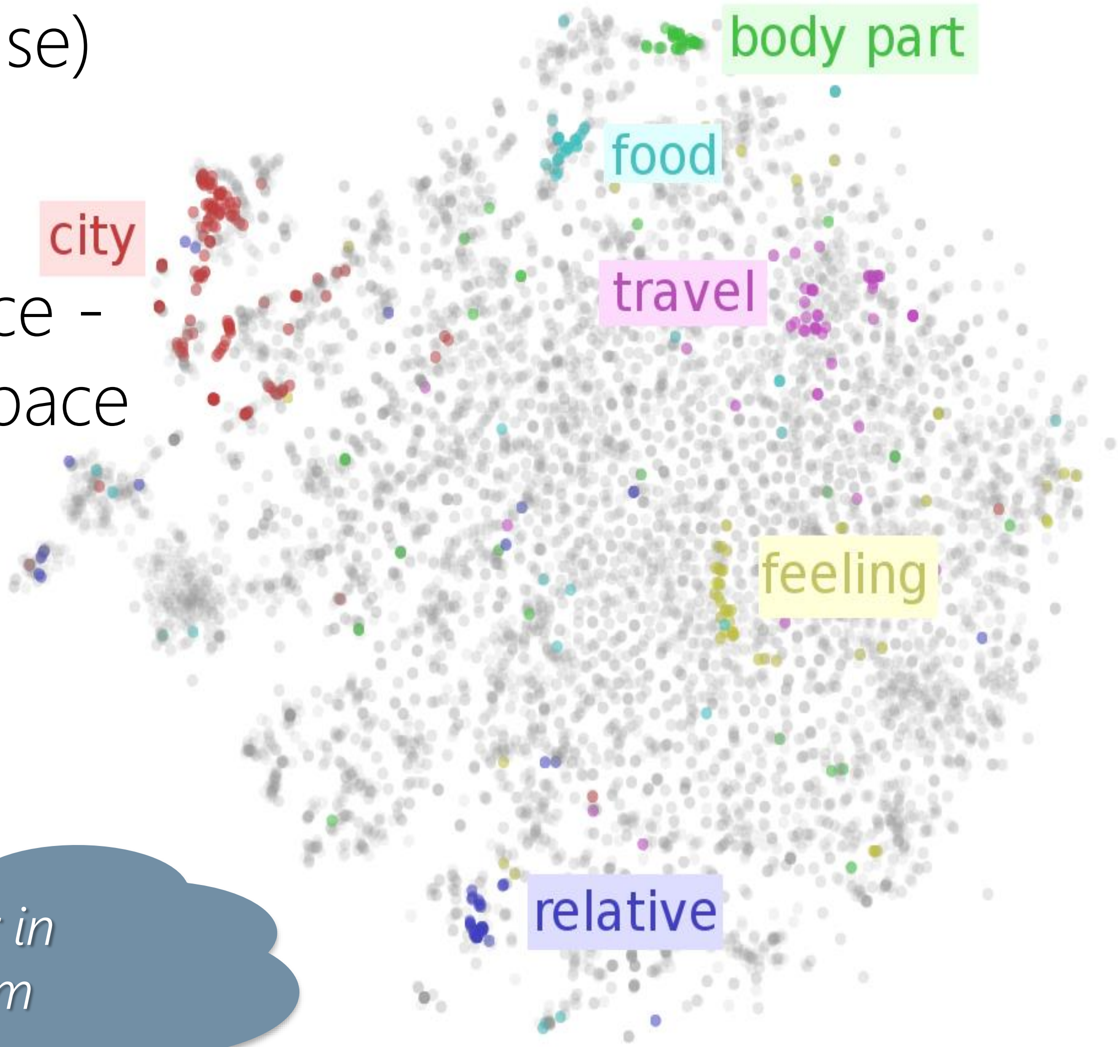
Embedding

Any technique mapping a word (or phrase) from it's original high-dimensional input space (the body of all words) to a lower-dimensional numerical vector space - so one *embeds* the word in a different space



Male-Female

Verb tense

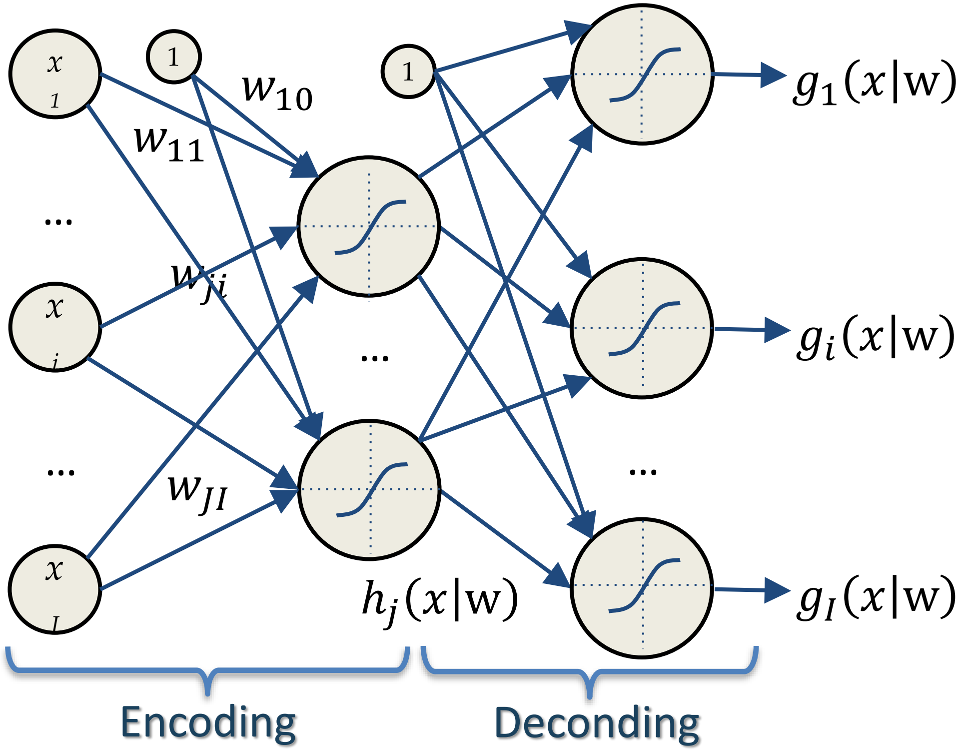


Closer points are closer in meaning and they form clusters …

Neural Autoencoder Recall

Network trained to output the input (i.e., to learn the identity function)

- Limited number of units in hidden layers (compressed representation)
- Constrain the representation to be sparse (sparse representation)



$$x \in \Re^I \xrightarrow{enc} h \in \Re^J \xrightarrow{dec} g \in \Re^I$$
$$J \ll I$$

$$E = \underbrace{\|g_i(x_i|\mathrm{w}) - x_i\|^2}_{\text{Reconstruction error}} + \underbrace{\lambda \sum_j \left| h_j\left(\sum_i w_{ji}^{(1)} x_i\right)\right|}_{\text{Sparsity term}}$$

$$g_i(x_i|\mathrm{w}) \sim x_i \qquad h_j(x_i|\mathrm{w}) \sim 0$$

Word Embedding: Distributed Representation

Each unique word $\boldsymbol{w}$ in a vocabulary V (typically $\|V\| > 10^6$) is mapped to a continuous m-dimensional space (typically $100 < m < 500$)

$$w \in V \xrightarrow{mapping\ C} \Re^m$$

$$\text{obama} = \underbrace{\overset{w_1}{[0\ 0}\ \ldots\ 0\ \overset{\text{obama}}{1}\ 0\ \ldots\ 0\ \overset{w_V}{0]}}_{\text{«one-hot» encoding}} \quad \Rightarrow \quad \text{obama} = \underbrace{\overset{f_1}{[0.12}\ \ldots\ \overset{f_m}{-0.25]}}_{\text{feature vector}}$$

Fighting the curse of dimensionality with:

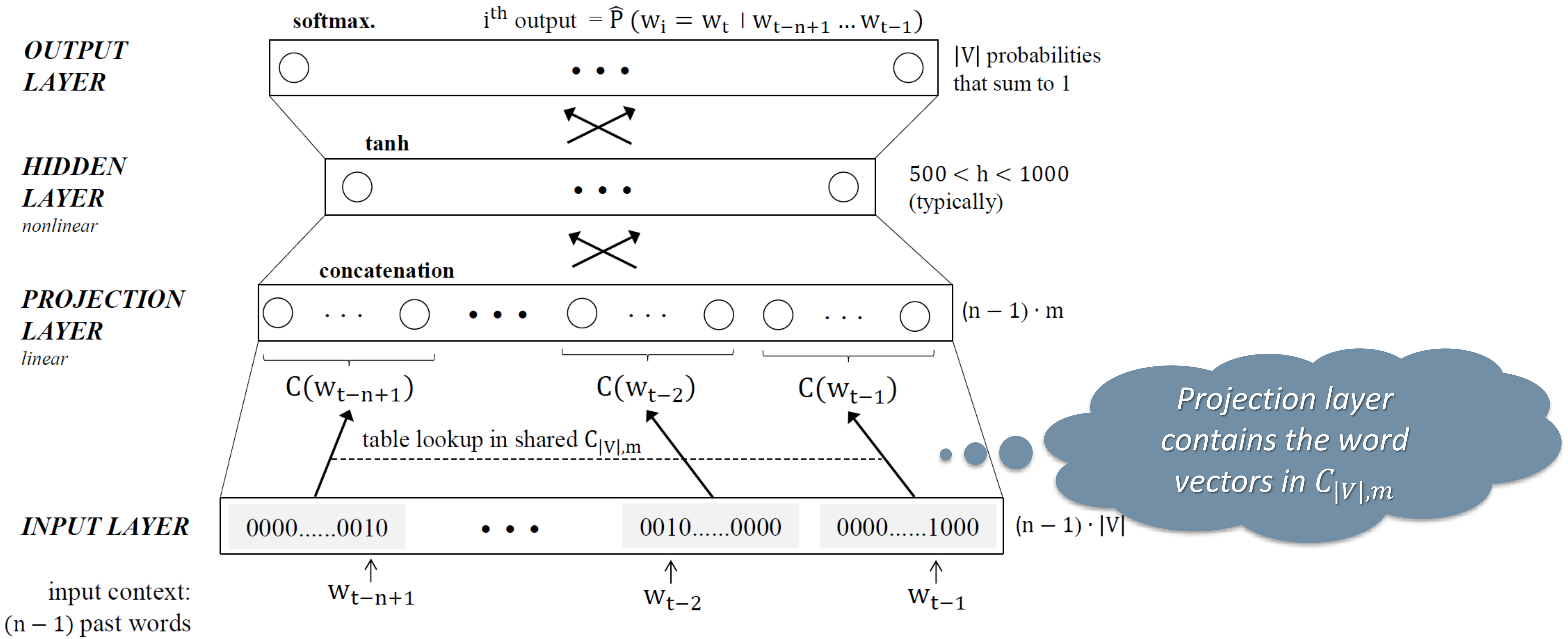
- Compression *(dimensionality reduction)*
- Smoothing *(discrete to continuous)*
- Densification *(sparse to dense)*

Similar words should end up to be close to each other in the feature space …

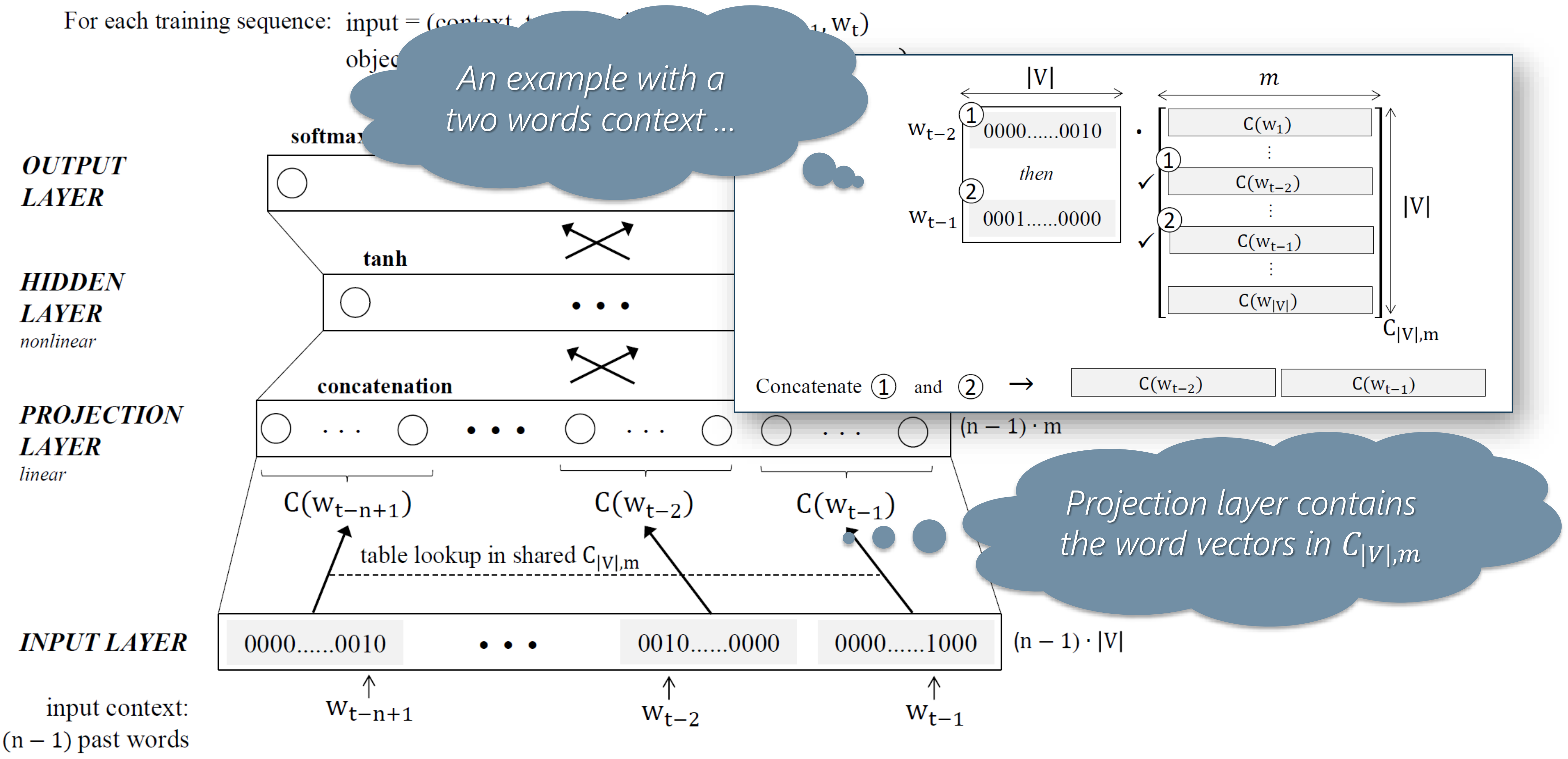
Neural Net Language Model (Bengio et al. 2003)

For each training sequence: input = (context, target) pair: $(w_{t-n+1} \dots w_{t-1}, w_t)$

objective: minimize $E = -\log \widehat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$

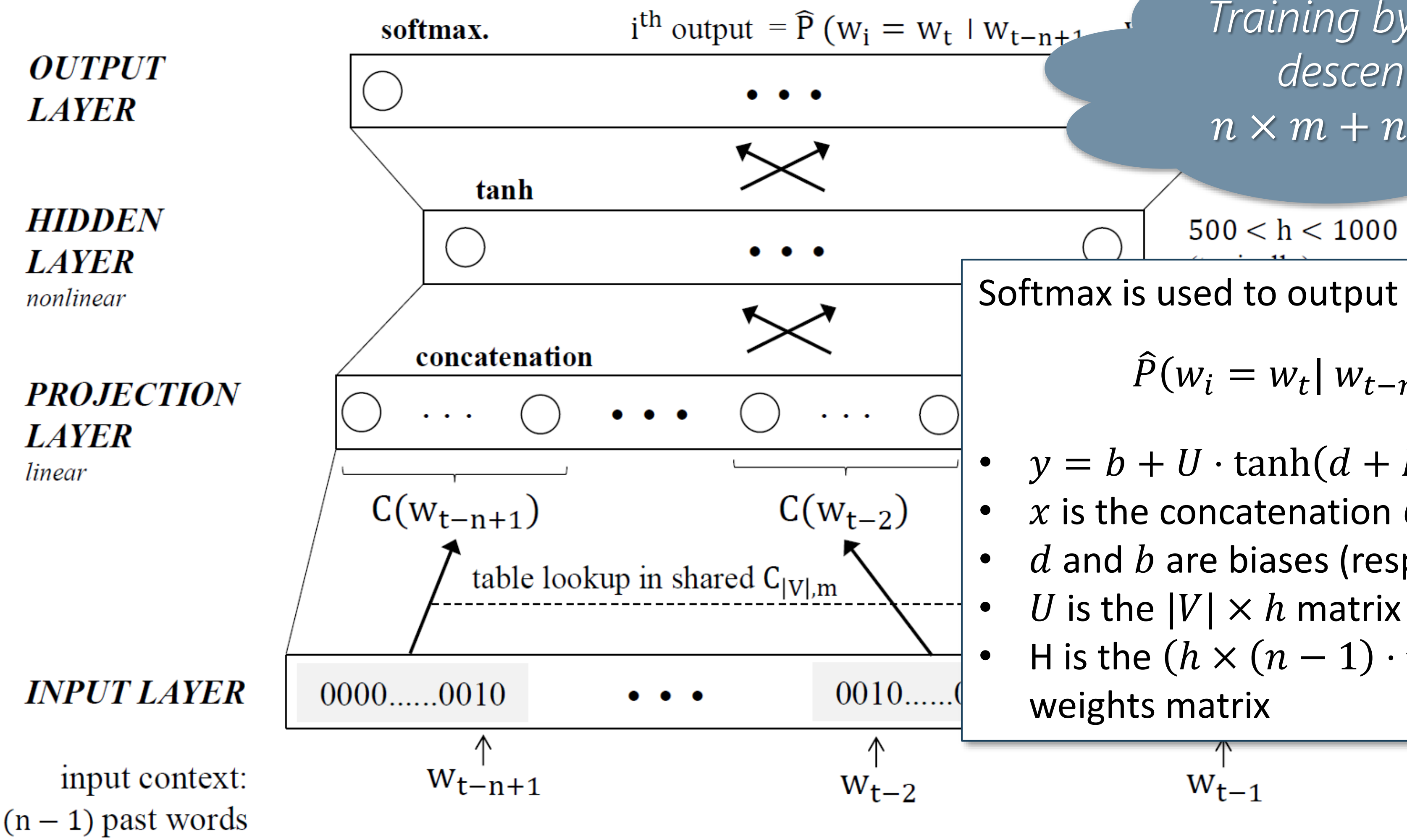


Neural Net Language Model (Bengio et al. 2003)



Neural Net Language Model (Bengio et al. 2003)

For each training sequence: input = (context, target) pair: $(w_{t-n+1} \dots w_{t-1}, w_t)$
objective: minimize $E = -\log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$



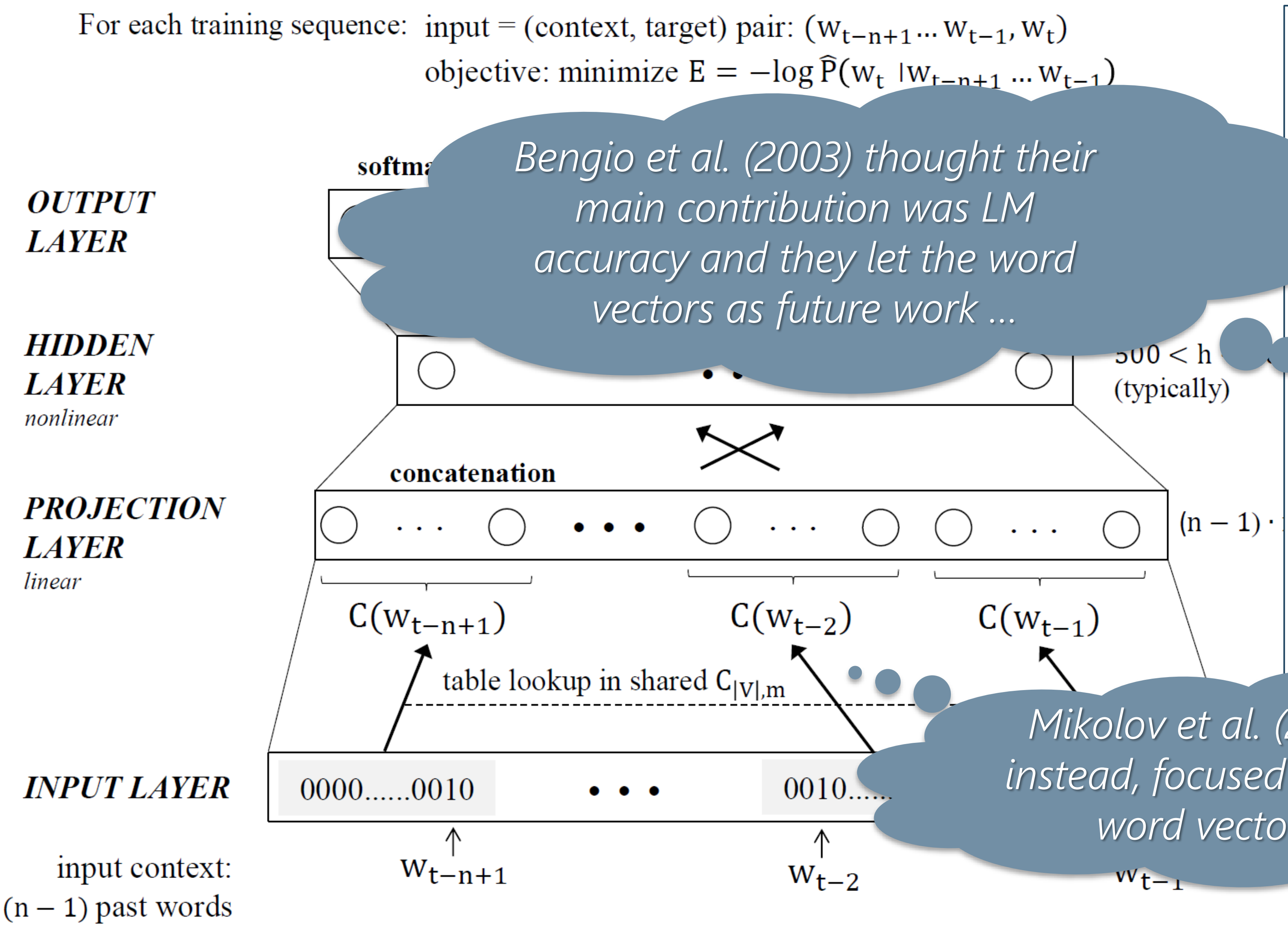
Training by stochastic gradient descent has complexity
$n \times m + n \times m \times h + \boldsymbol{h} \times |\boldsymbol{V}|$

Softmax is used to output a multinomial distribution

$$\hat{P}(w_i = w_t \mid w_{t-n+1}, \dots, w_{t-1}) = \frac{e^{y_{w_i}}}{\sum_{i'}^{|V|} e^{y_{w_{i'}}}}$$

- $y = b + U \cdot \tanh(d + H \cdot x)$
- x is the concatenation $C(w)$ of the context weight vectors
- d and b are biases (respectively h and $|V|$ elements)
- U is the $|V| \times h$ matrix with hidden-to-output weights
- H is the $(h \times (n-1) \cdot m)$ projection-to-hidden weights matrix

Neural Net Language Model (Bengio et al. 2003)



Bengio et al. (2003) thought their main contribution was LM accuracy and they let the word vectors as future work …

Tested on Brown (1.2M words, V≅ 16K, 200K test set) and AP News (14M words, V≅ 150K reduced to 18K, 1M
est set)

- Brown: h=100, n=5, m=30
- AP News: h=60, n=6, m=100
- **3 week** training using **40 cores**
- 24% (Brown) and 8% (AP News) relative improvement wrt traditional smoothed n-gram in terms of test set perplexity

Due to **complexity**, NNLM can't be applied to large data sets and it shows
on rare words

Mikolov et al. (2013), instead, focused on the word vectors

Google's word2vec (Mikolov et al. 2013a)

Idea: achieve better performance allowing a simpler (shallower) model to be trained on much larger amounts of data

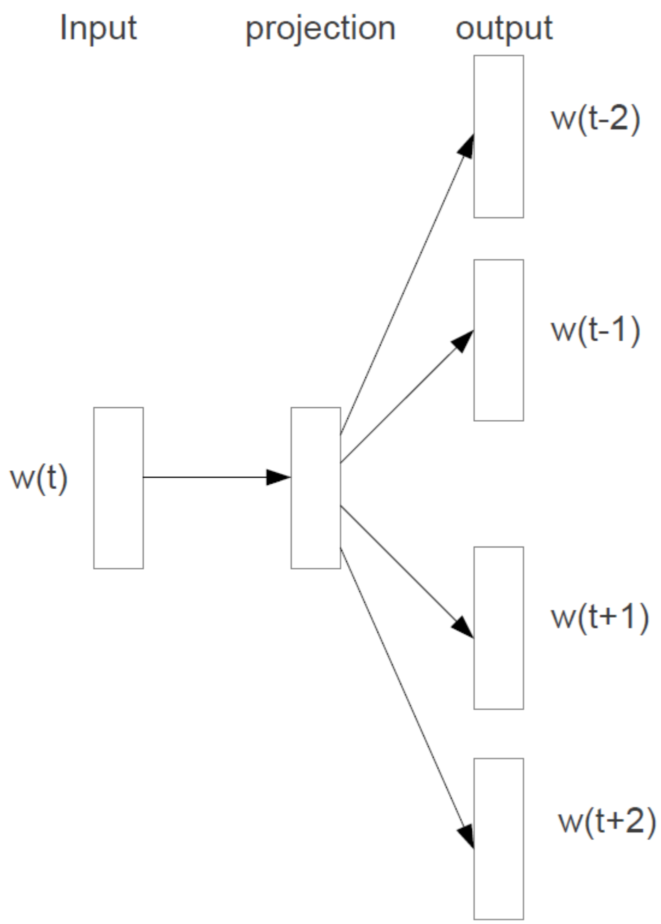
- No hidden layer (leads to 1000X speed up)
- Projection layer is shared (not just the weight m
- Context contain words both from history and futur

«You shall know a word
by the company it keeps»
John R. Firth, 1957:11.

…Pelé has called **Neymar** an excellent player…
…At the age of just 22 years, **Neymar** had scored 40 goals in 58 internationals…
…occasionally as an attacking midfielder, **Neymar** was called a true phenomenon…

These words will represent **Neymar**

Google word2vec Flavors

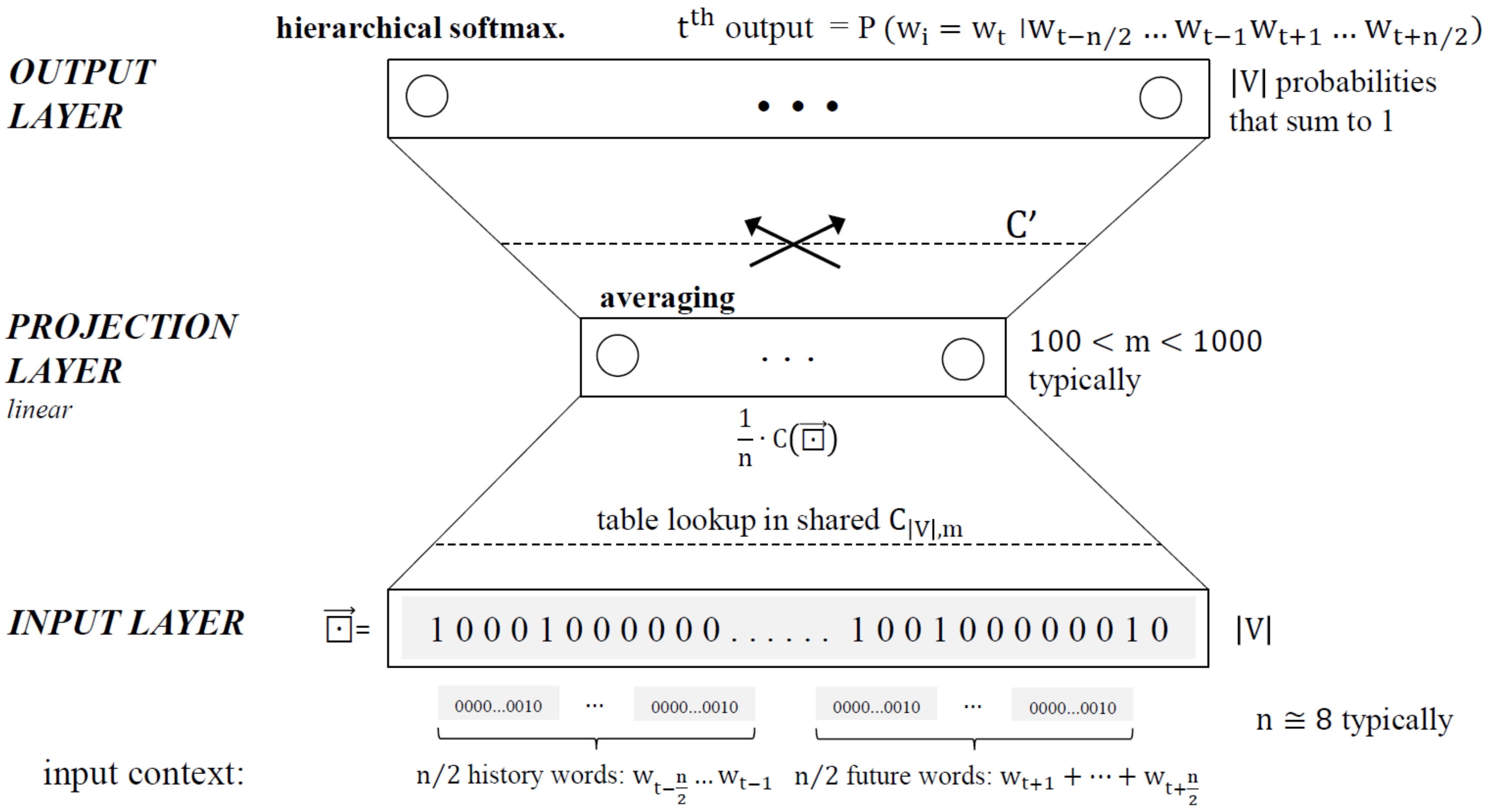


Skip-gram architecture

Word2vec's Continuous Bag-of-Words (CBOW)

For each training sequence: input = (context, target) pair: $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

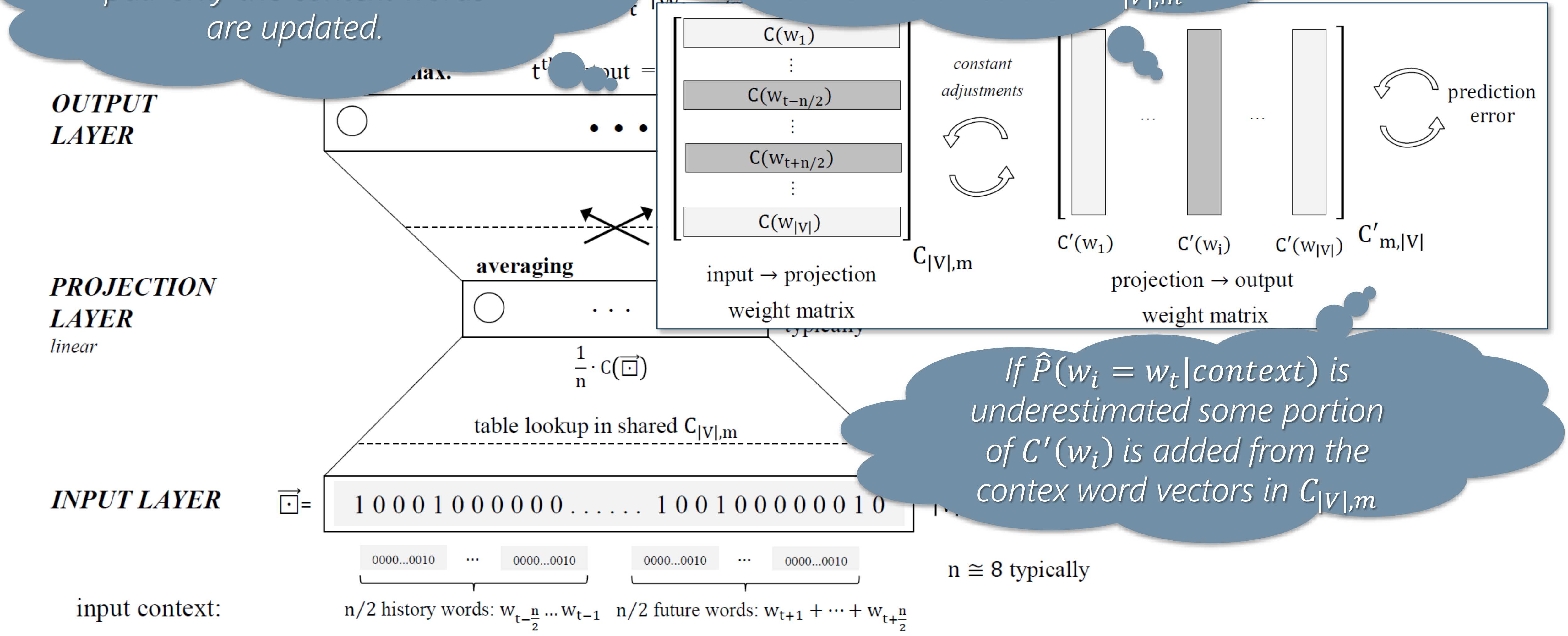
objective: minimize $E = -\log \hat{P}(w_t \mid w_{t-n/2} \dots w_{t-1} w_{t+1} \dots w_{t+n/2})$



Word2vec's Continuous Bag-of-W

For each <context, target> pair only the context words are updated.

If $\hat{P}(w_i = w_t|context)$ is overestimated some portion of $C'(w_i)$ is subtracted from the contex word vectors in $C_{|V|,m}$



If $\hat{P}(w_i = w_t|context)$ is underestimated some portion of $C'(w_i)$ is added from the contex word vectors in $C_{|V|,m}$

Word2vec facts

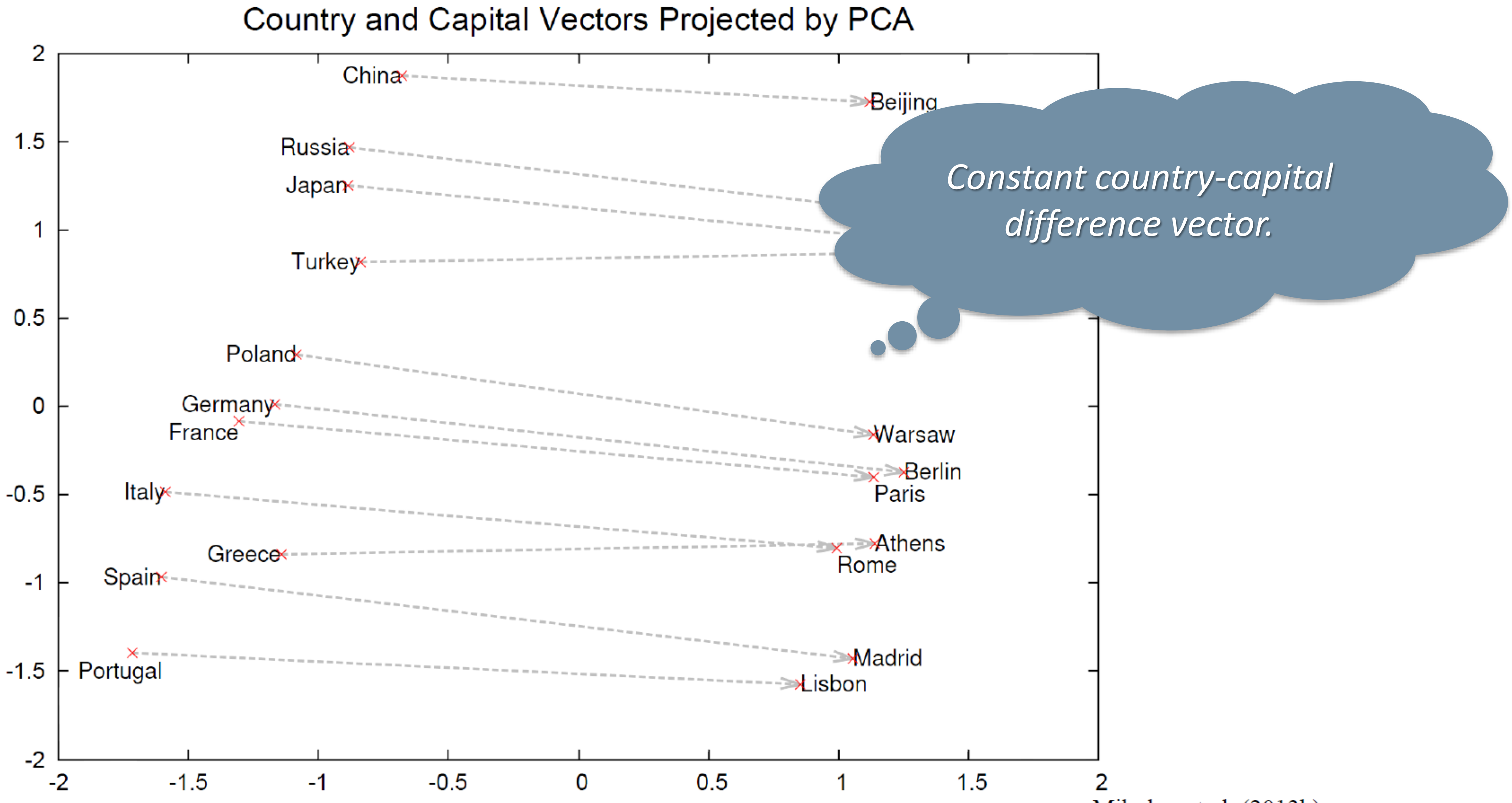
Word2vec shows significant improvements w.r.t. the NNML

- Complexity is $n \times m + m \times log|V|$ (Mikolov et al. 2013a)
- On Google news 6B words training corpus, with $|V| \sim 10^6$
  - CBOW with m=1000 took 2 days to train on 140 cores
  - Skip-gram with m=1000 took 2.5 days on 125 cores
  - NNLM (Bengio et al. 2003) took 14 days on 180 cores, for m=100 only!
- word2vec training speed ≅ 100K-5M words/s
- Best NNLM: 12.3% overall accuracy vs. Word2vec (with Skip-gram): 53.3%

Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

Adapted from Mikolov et al. (2013a)

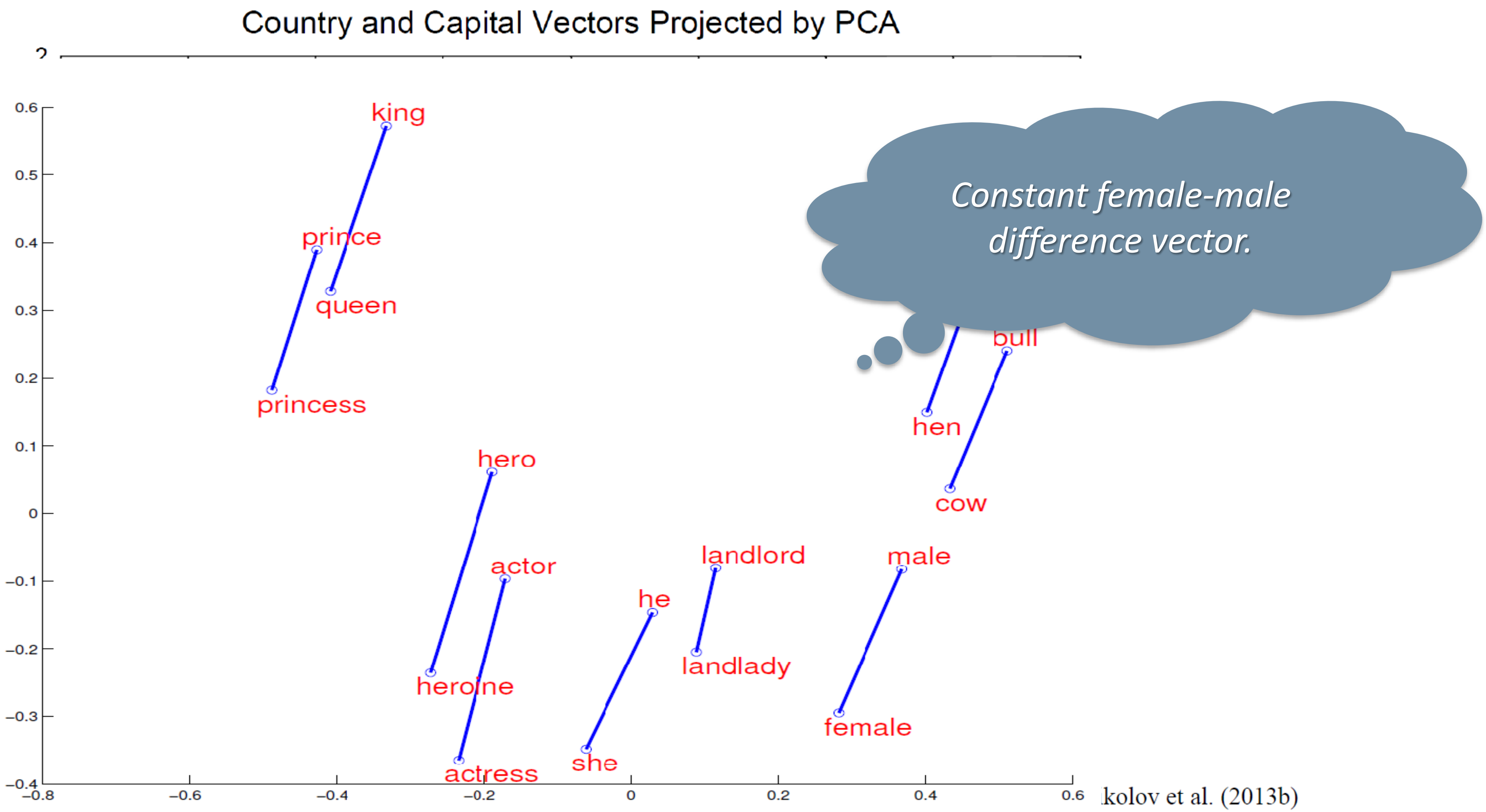
Regularities in word2vec Embedding Space



Mikolov et al. (2013b)

Picture taken from:
https://www.scribd.com/document/285890694/NIPS-DeepLearningWorkshop-NNforText

Regularities in word2vec Embedding Space

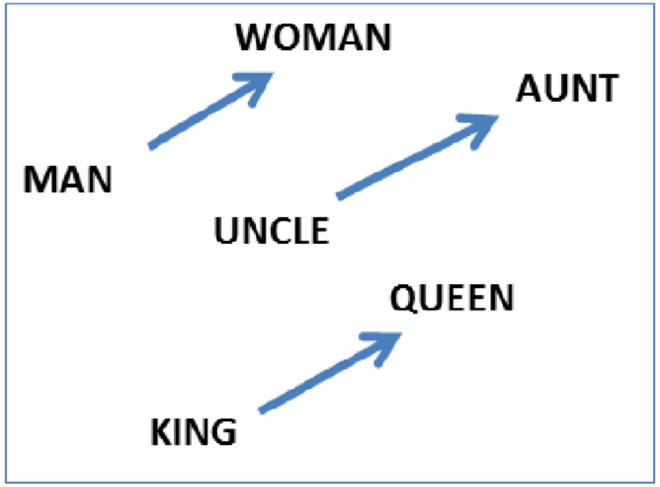


Picture taken from:
https://www.scribd.com/document/285890694/NIPS-DeepLearningWorkshop-NNforText

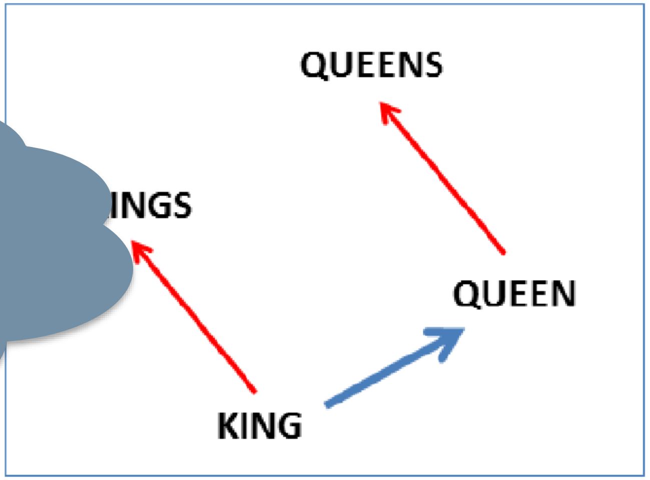
Regularities in word2vec Embedding Space

Vector operations are supported make «intuitive sense»:

- $w_{king} - w_{man} + w_{woman} \cong w_{queen}$
- $w_{paris} - w_{france} + w_{italy} \cong w_{rome}$
- $w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$
- $w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso}$
- $w_{his} - w_{he} + w_{she} \cong w_{her}$
- $w_{cu} - w_{copper} + w_{gold} \cong w_{au}$
- …



«You shall know a word by the company it keeps»
John R. Firth, 1957:11.



Picture taken from:
https://www.scribd.com/document/285890694/NIPS-DeepLearningWorkshop-NNforTex

Applications of word2vec in Information Retrieval

Query: “restaurants in mountain view that are not very good”

Phrases: “restaurants in (mountain view) that are (not very good)”

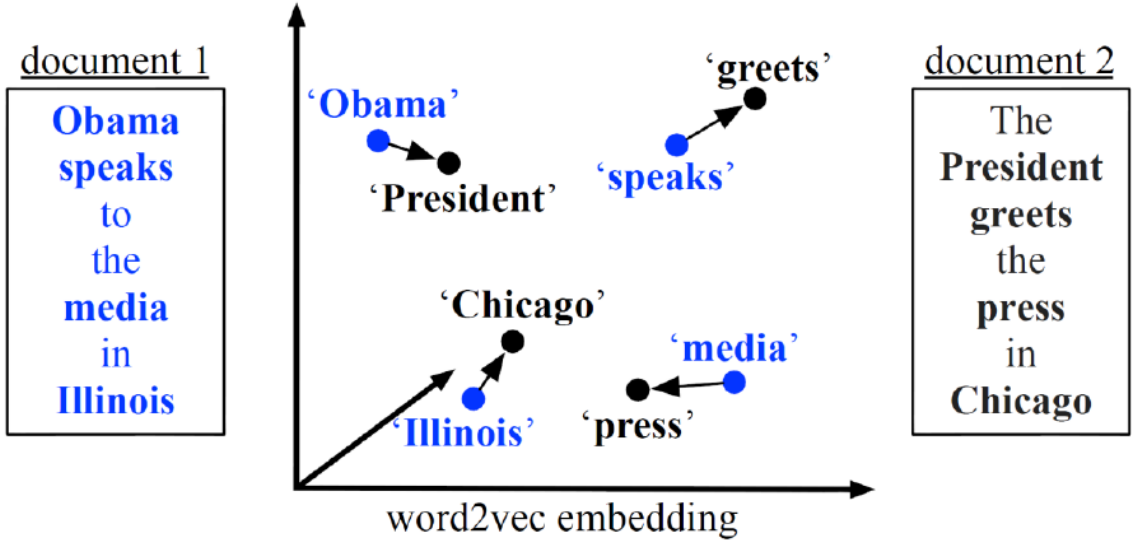
Vectors: “restaurants+in+(mountain view)+that+are+(not very good)”

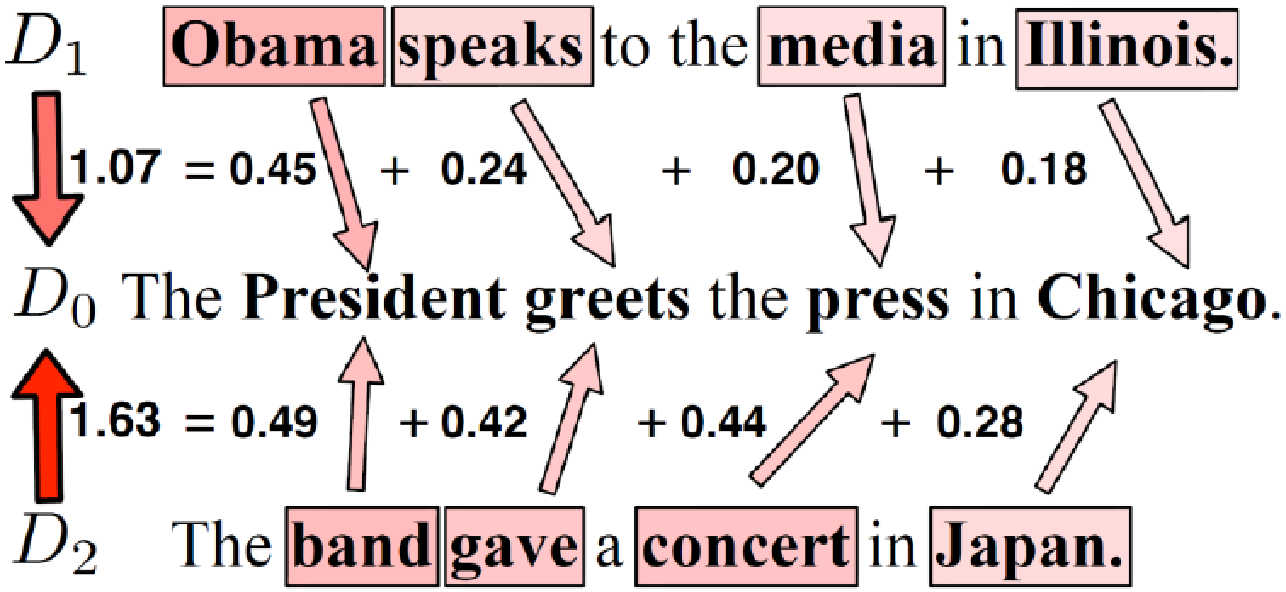
Expression	*Nearest tokens*
Czech + currency	koruna, Czech crown, Polish zloty, CTK
Vietnam + capital	Hanoi, Ho Chi Minh City, Viet Nam, Vietnamese
German + airlines	airline Lufthansa, carrier Lufthansa, flag carrier Lufthansa
Russian + river	Moscow, Volga River, upriver, Russia
French + actress	Juliette Binoche, Vanessa Paradis, Charlotte Gainsbourg

(Simple and efficient, but will not work for long sentences or documents)

Applications of word2vec in Document Classification/Similarity

With BoW D_1 and D_2 are equally similar to D_0.





Word embeddings allow to capture the «semantics» of the document ...

Applications of word2vec in Sentiment Analysis

«You shall know a word by the company it keeps»
John R. Firth, 1957:11.

No need for classifiers, just use cosine distanc

```
Enter word or sentence (EXIT to break): sad

Word: sad  Position in vocabulary: 4067

                                              Word       Cosine distance
------------------------------------------------------------------------
                                         saddening              0.727309
                                               Sad              0.661083
                                          saddened              0.660439
                                      heartbreaking              0.657351
                                      disheartening              0.650732
                                      Meny_Friedman              0.648706
                            parishioner_Pat_Patello              0.647586
                                         saddens_me              0.640712
                                        distressing              0.639909
                                  reminders_bobbing              0.635772
                                   Turkoman_Shiites              0.635577
                                           saddest              0.634551
                                        unfortunate              0.627209
                                              sorry              0.619405
                                        bittersweet              0.617521
                                             tragic              0.611279
                                          regretful              0.603472
```