

Word Embedding

Ayoub Bagheri

Outline

- Word embedding
 - Skipgram learning
 - Pre-trained embeddings
- State-of-the-art

Word Embedding

Slides are partly based on the word embedding lecture by Dong Nguyen in the Applied Text Mining Utrecht summer school ([linkToRCourse](#), [linkToPythonCouse](#))

Word representations

How can we represent the meaning of words?

So, we can ask:

- How similar is cat to dog, or Paris to London?
- How similar is document A to document B?

Word as vectors

Can we represent words as vectors?

The vector representations should:

- capture semantics
 - similar words should be close to each other in the vector space
 - relation between two vectors should reflect the relationship between the two words
- be efficient (vectors with fewer dimensions are easier to work with)
- be interpretable

Word as vectors

How similar are the following two words? (not similar 0–10 very similar)

smart and **intelligent**:

easy and **big**:

easy and **difficult**:

hard and **difficult**:

Word as vectors

How similar are the following two words? (not similar 0–10 very similar)

smart and **intelligent**: **9.20**

easy and **big**: **1.12**

easy and **difficult**: **0.58**

hard and **difficult**: **8.77**

(SimLex-999 dataset, <https://fh295.github.io/simlex.html>)

Words as Vectors

One-hot encoding

Map each word to a unique identifier

e.g. cat (3) and dog (5).

- Vector representation: all zeros, except 1 at the ID

cat	0	0	1	0	0	0	0
dog	0	0	0	0	1	0	0
car	0	0	0	0	0	0	1

One-hot encoding

Map each word to a unique identifier

e.g. cat (3) and dog (5).

- Vector representation: all zeros, except 1 at the ID

cat	0	0	1	0	0	0	0
dog	0	0	0	0	1	0	0
car	0	0	0	0	0	0	1

What are limitations of one-hot encodings?

One-hot encoding

Map each word to a unique identifier

e.g. cat (3) and dog (5).

- Vector representation: all zeros, except 1 at the ID

cat	0	0	1	0	0	0	0
dog	0	0	0	0	1	0	0
car	0	0	0	0	0	0	1

Even related words
have distinct vectors!

High number of
dimensions

Distributional hypothesis: Words that occur in similar contexts tend to have similar meanings.

**You shall know a word by the company it keeps.
(Firth, J. R. 1957:11)**

Word vectors based on co-occurrences

documents as context
word-document matrix

	doc ₁	doc ₂	doc ₃	doc ₄	doc ₅	doc ₆	doc ₇
cat	5	2	0	1	4	0	0
dog	7	3	1	0	2	0	0
car	0	0	1	3	2	1	1

Word vectors based on co-occurrences

documents as context
word-document matrix

	doc ₁	doc ₂	doc ₃	doc ₄	doc ₅	doc ₆	doc ₇
cat	5	2	0	1	4	0	0
dog	7	3	1	0	2	0	0
car	0	0	1	3	2	1	1

neighboring words as context
word-word matrix

	cat	dog	car	bike	book	house	tree
cat	0	3	1	1	1	2	3
dog	3	0	2	1	1	3	1
car	0	0	1	3	2	1	1

Word vectors based on co-occurrences

There are many variants:

- Context (words, documents, which window size, etc.)
- Weighting (raw frequency, etc.)

Vectors are sparse: Many zero entries.

Therefore: Dimensionality reduction is often used (e.g., SVD)

These methods are sometimes called **count-based** methods as they work directly on **co-occurrence** counts.

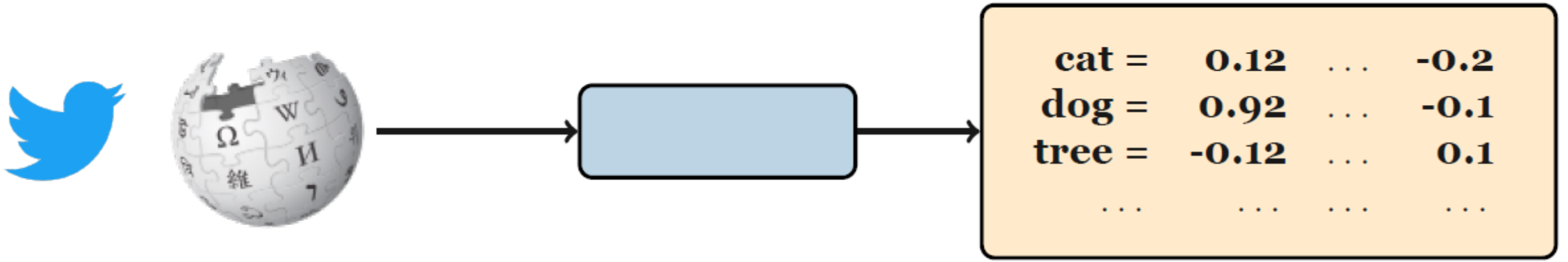
Word embeddings

- Vectors are short; typically 50-1024 dimensions 😊
- Vectors are dense (mostly non-zero values)
- Very effective for many NLP tasks 😊
- Individual dimensions are less interpretable 😞

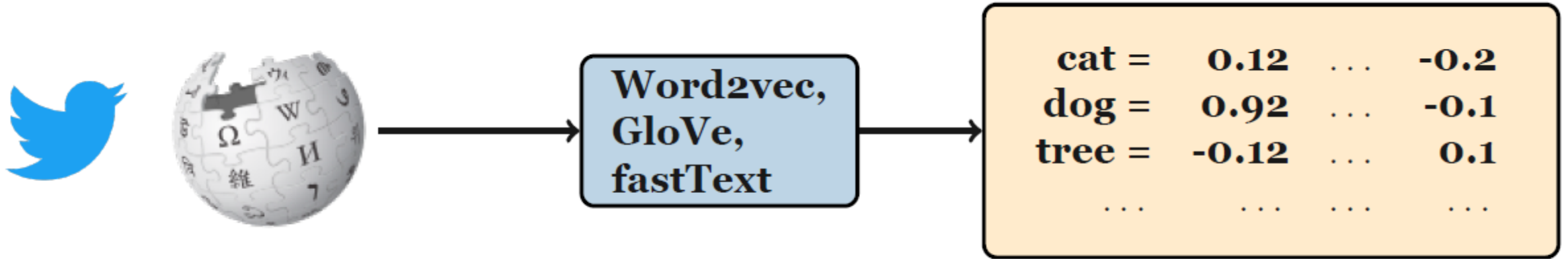
cat	0.52	0.48	-0.01	...	0.28
dog	0.32	0.42	-0.09	...	0.78

How do we learn word embeddings?

Learning word embeddings



Learning word embeddings



Training data for word embeddings

- Use **text itself** as training data for the model!
 - A form of self-supervision.
- Train a **classifier** (neural network, logistic regression, or SVM, etc.) to predict the next word given previous words.

Exercise: Word prediction task

Yesterday I went to the ?

A new study has highlighted the positive ?

Which word comes next?

Word2Vec

- Popular embedding method
- Very fast to train
- Idea: **predict** rather than **count**
- <https://projector.tensorflow.org/>

Word2Vec

The domestic **cat** is a small, typically furry carnivorous mammal

w_{-2} w_{-1} w_0 w_1 w_2 w_3 w_4 w_5

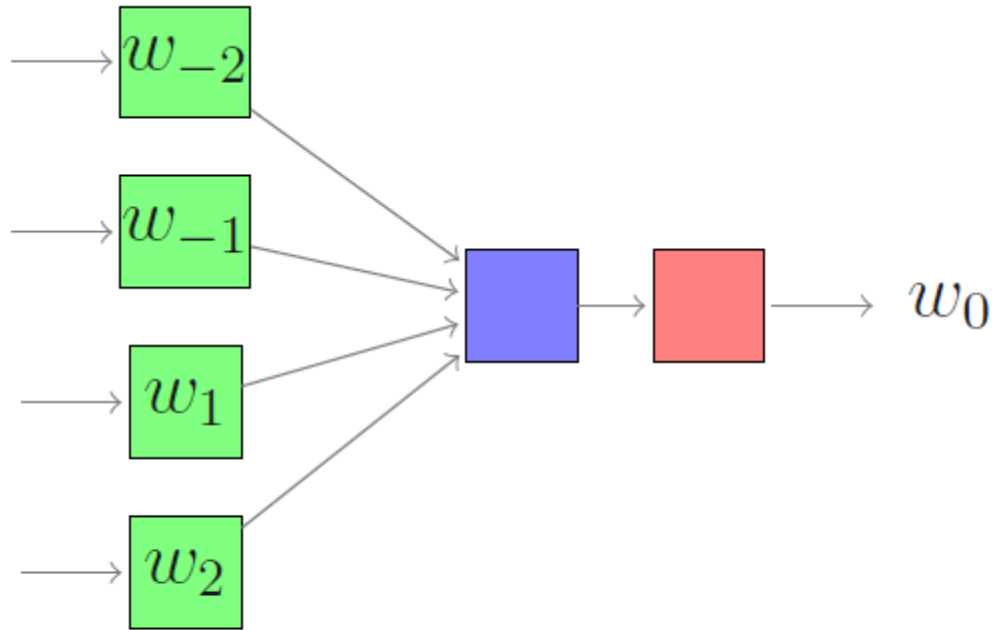
We have **target** words (cat) and **context** words (here: window size = 5).

Word2Vec

- Instead of **counting** how often each word w occurs near a target word
 - Train a classifier on a binary **prediction** task:
 - Is w likely to show up near target?
- We don't actually care about this task
 - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**
 - A word c that occurs near target in the corpus as the gold "correct answer" for supervised learning
 - **No need for human labels**
 - Bengio et al. (2003); Collobert et al. (2011)

Word2Vec algorithms

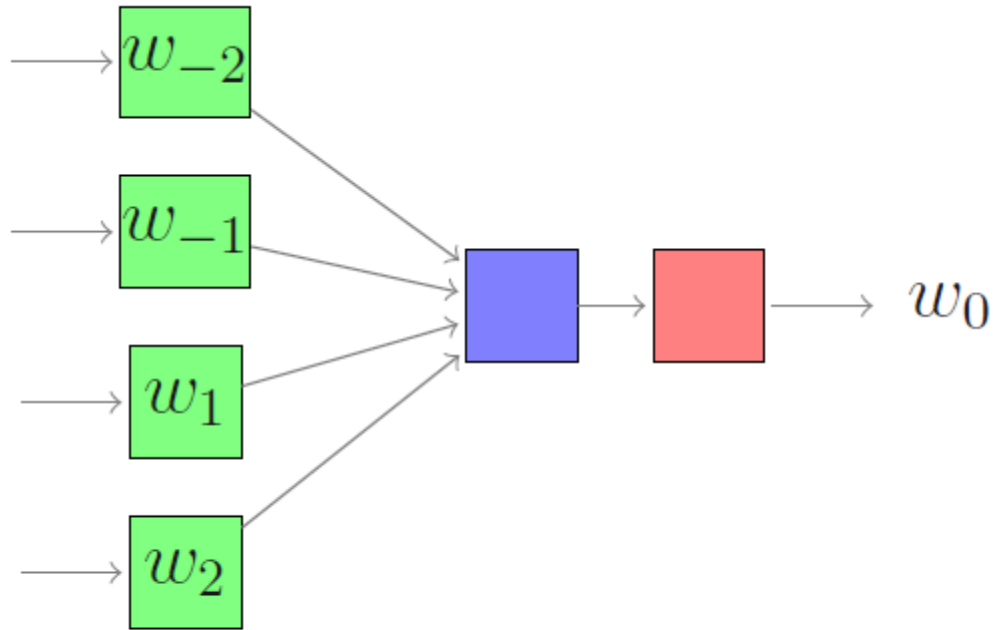
Continuous Bag-Of-Words (CBOW)



one snowy ? she went

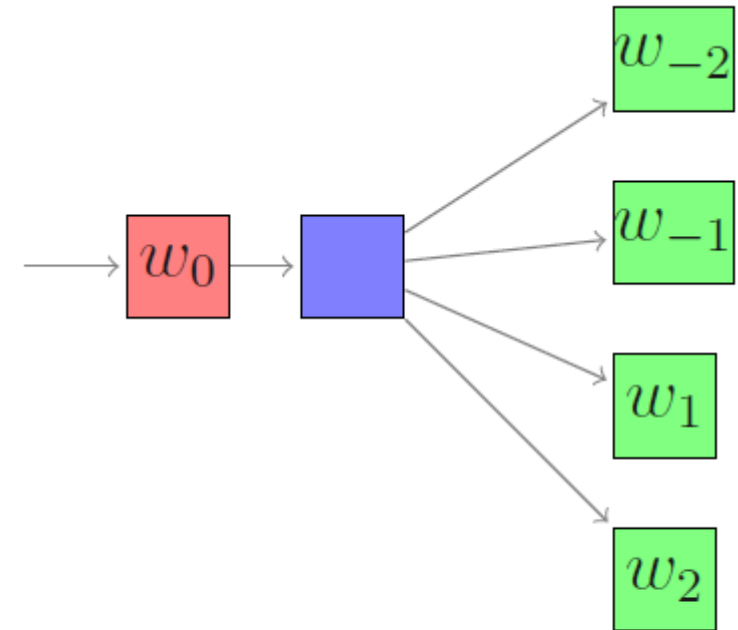
Word2Vec algorithms

Continuous Bag-Of-Words (CBOW)



one snowy ? she went

skipgram



? ? day ? ?

Skipgram overview

The domestic **cat** is a small, typically furry carnivorous mammal

1. Create examples

- Positive examples: Target word and neighboring context
- Negative examples: Target word and randomly sampled words from the lexicon (*negative sampling*)

2. Train a **logistic regression** model to distinguish between the positive and negative examples
3. The resulting **weights** are the embeddings!

word (w)	context (c)	label
cat	small	1
cat	furry	1
cat	car	0
...

Embedding vectors are essentially a byproduct!

Skipgram

The domestic **cat** is a small, typically furry carnivorous mammal

w_{-2} w_{-1} w_0 w_1 w_2 w_3 w_4 w_5

We have **target** words (cat) and **context** words (here: window size = 5).

The probability that c is a real context word, and the probability that c is not a real context word:

$$P(+ | w, c)$$

$$P(- | w, c) = 1 - P(+ | w, c)$$

Skipgram

Similarity is computed from dot product

- **Intuition:** A word c is likely to occur near the target w if its embedding is similar to the target embedding.

$$\approx w \cdot c$$

- Two vectors are similar if they have a high dot product
- Cosine similarity is just a normalized dot product

Turn this into a probability using the sigmoid function:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

How Skipgram classifier computes $P(+|w, c)$

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words. We'll assume independence and just multiply them:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(c_i \cdot w)$$

Word2vec: how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
- **Maximize** the similarity of the **target word, context word** pairs (w, c_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Loss function for one w with c_{pos} , $c_{neg1} \dots c_{negk}$

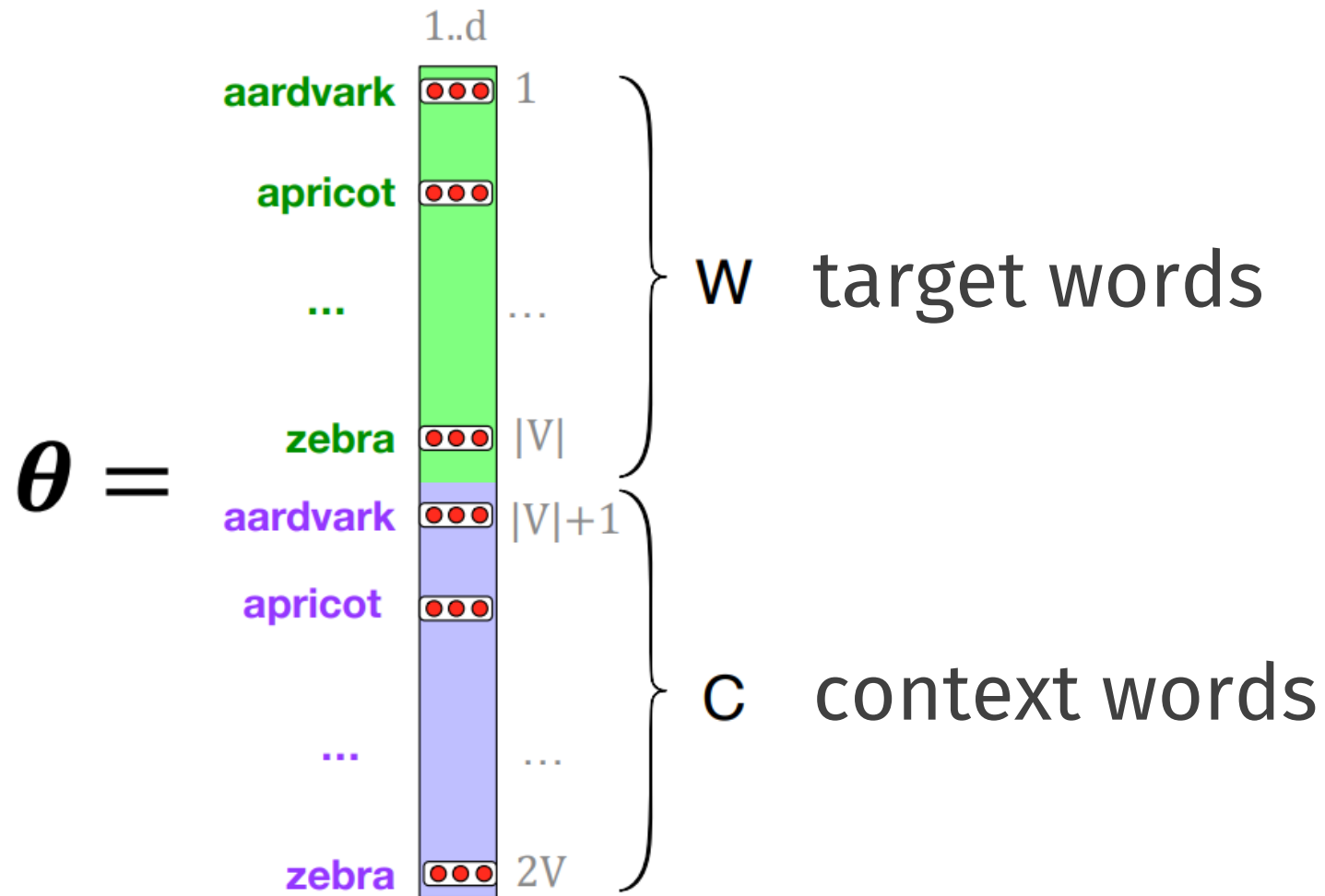
- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned}$$

Learning the classifier

- How to learn?
 - **Stochastic gradient descent!**

Skipgram embeddings



Learning the classifier

- How to learn?
 - **Stochastic gradient descent!**
- SGNS learns two sets of embeddings
 - Target embeddings matrix W
 - Context embedding matrix C
- It's common to just add them together, representing word i as the vector $W_i + C_i$

Skipgram

1. Treat the target word t and a neighboring context word c as positive examples.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skipgram classifier

- A probabilistic classifier, given
 - a test target word w
 - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).
- To compute this, we just need embeddings for all the words.

Pre-trained Embeddings

Pre-trained embeddings

- I want to build a system to **solve a task** (e.g., sentiment analysis)
 - Use pre-trained embeddings. Should I **fine-tune**?
 - Lots of data: yes
 - Just a small dataset: no
- **Analysis** (e.g., bias, semantic change)
 - Train embeddings from scratch

State-of-the-art

State-of-the-art

- Recurrent neural networks
 - LSTM
 - GRU
 - Bi-directional network
- Transformers
- Contextual embeddings

We will discuss the details during the summer text mining courses:

- [Click to go to the Intro to Text Mining with R course](#)
- [Click to go to the Applied Text Mining course \(Python\)](#)

Practical

Word embedding

Questions?