VL Deep Learning for Natural Language Processing

09. Word Embeddings III

Prof. Dr. Ralf Krestel

AG Information Profiling and Retrieval

What is a „Word“

- Representation of a word different in different languages
- Word boundries marked by space character (or not)
  - 我想吃一个汉堡
  - I want to eat a hamburger
- Clitics, pronouns, agreement?
  - Separated
    - **Je vous aime** **Il y a beaucoup d'argent**
  - Joined
    - اهانلفق = ف+ لقا +نا +ها = so+said+we+it
- Composita
  - Separated
    - life insurance company employee
  - Joined
    - Lebensversicherungsgesellschaftsangestellter

Why is a Model on Word Level not Enough?

- Unknown word (**out-of-vocabulary**, OOV)
  - Spelling mistakes:
  - Rich morphology: Composita
  - Informal spelling: Veeeeery Slooooow
  - Transliterations: Christopher ↦Kryštof

Rare words

- Differences in writing systems (Where is the meaning?)

– Phonemic	jiyawu ngabulu	Wambaya
– Fossilized phonemic	thorough failure	English
– Syllabic/moraic	ᑐᖑᓯᐊᖓᓯᖅ	Inuktitut
– Ideographic	去年太空船二号坠毁	Chinese
– Combination (syllabic+ideographic)	インド洋の島	Japanese

Lerning Goals for this Chapter

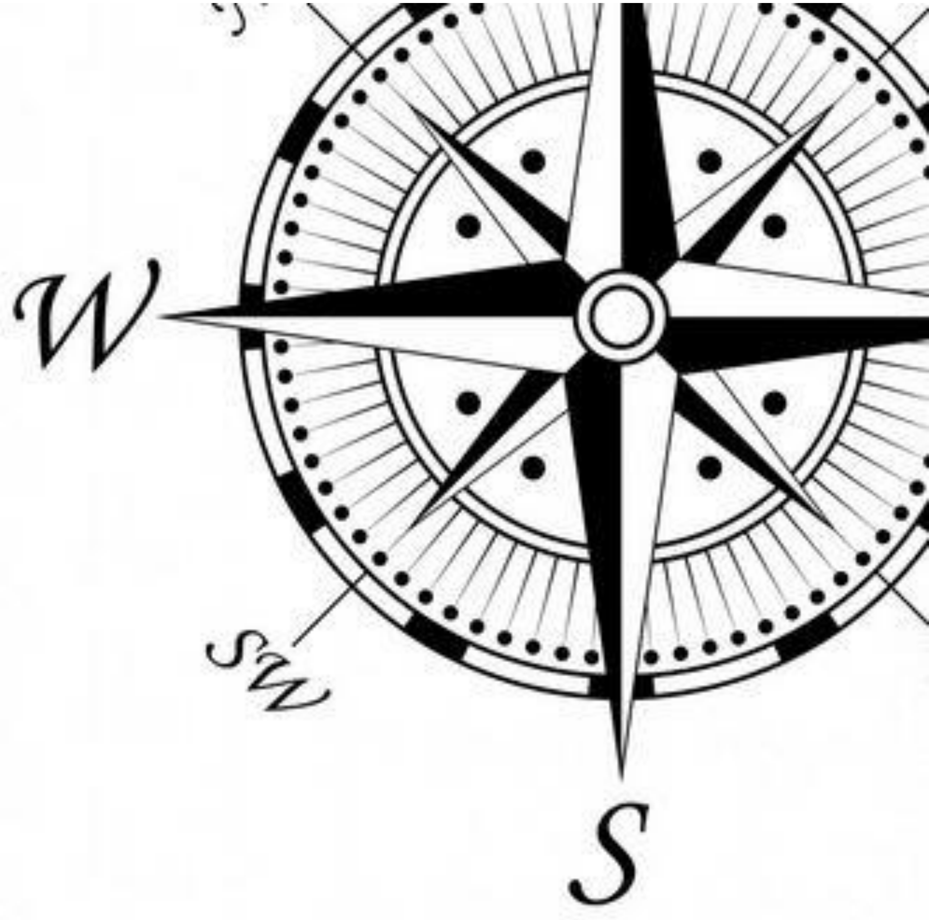
- Be able to explain the diffences between embedding models
  - Character level
  - Subword level
  - Word level

- Understand document/paragraph/sentence embeddings
  - Word movers distance
  - Doc2vec

- Partly based on Chris Manning's lecture 2019
  - https://www.youtube.com/watch?v=9oTHFx0Gg3Q&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=12

Topics Today

1. **Sub-Word Models**
2. Document Embeddings



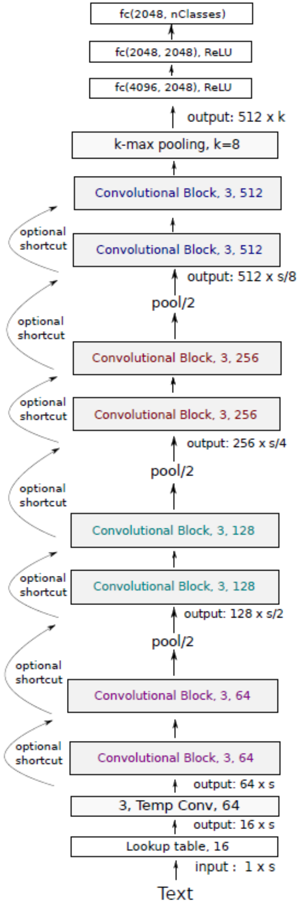
Character Level Models

1. Word embeddings can be composed from character embeddings
   - Enables embeddings for unknown words
   - Similar spelling -> similar embedding
   - Solves the OOV problem
   - Combination of character and word level
2. Written language as a sequence of individual characters
   - No explicit representations for words
   - **Pure character-based models**

- Both methods are very successful
  - Surprising, since traditionally, phonemes/characters are not semantic units, but, DL models group them (morphemes)
    - Unfortunately = $\left[\left[un\ [[fortun(e)]_{ROOT}\ ate]_{STEM}\right]_{STEM}\ ly\right]_{WORD}$

Pure Character Level Models

- Pure character level model based on CNNs
- Idea:
  - Learn hierarchical representations of sentences
  - Task-based (text classification)
  - Input are characters
  - Deeper layers form syllables, words, phrases, sentences
  - Up to 29 layers



Conneau, Schwenk, Lecun, Barrault. *Very deep convolutional networks for text classification*. In EACL 2017

Sub-Word Embeddings

- Instead of words, also other units could be considered
  - Sentences bear meaning
    - *John likes strawberries*
  - Documents bear meaning
  - …

  - Syllables (subwords) bear meaning
    - In particular in morphological rich languages
    - *expensive* vs. *inexpensive*

Out-of-Vocabulary (OOV) words

- Advantage of subword embeddings
  - Words can be embedded (represented as vectors) even if they do not occur very frequently (or not at all) in the training data

Byte Pair Encoding (BPE)

- Originally a compression algorithm:
  - Most frequent byte pair↦a new byte
  - Replace bytes with character n-grams

- A word segmentation algorithm:
  - Though done as bottom up clustering
  - Start with a unigram vocabulary of all (Unicode) characters in data
  - Most frequent n-gram-pairs ↦a new n-gram

- Implementierung
  - *https://github.com/rsennrich/subword-nmt*

Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units." *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016.

Byte Pair Encoding

- *Dictionary (# occurences, word)*
  5 l o w
  2 l o w e r
  6 n e w e s t
  3 w i d e s t

- Start with all characters in vocabulary
- Add a pair (e,s) with frequency 9
- Add a pair (es,t) with frequency 9
- Add a pair (l,o) with frequency 7

- Vocabulary
  – l, o, w, e, r, n, w, s, t, i, d

- New encodings:
  – l, o, w, e, r, n, w, s, t, i, d
  – l, o, w, e, r, n, w, s, t, i, d, es
  – l, o, w, e, r, n, w, s, t, i, d, es, est
  – l, o, w, e, r, n, w, s, t, i, d, es, est, lo

Byte Pair Encoding

- Given a target vocabulary size
  - stop when you reach it
- Do deterministic longest piece segmentation of words
- Segmentation is only within words identified by some prior tokenizer (commonly Moses tokenizer for MT)
- Automatically decides vocab for system
- No longer strongly “word” based in conventional way
- Top places in WMT 2016!
  - Also widely used in WMT 2018

Wordpiece/Sentencepiece Models I

- Google NMT (GNMT) uses a variant of this
  - V1: wordpiece model
  - V2: sentencepiece model
  - Rather than char *n*-gram count, uses a greedy approximation to maximizing language model log likelihood to choose the pieces
  - Add *n*-gram that maximally reduces perplexity

- Wordpiece model tokenizes inside words
- Sentencepiece model works from raw text
  - Whitespace is retained as special token (_) and grouped normally
  - You can reverse things at end by joining pieces and recoding them to spaces
  - https://github.com/google/sentencepiece
  - https://arxiv.org/pdf/1804.10959.pdf

Wordpiece/Sentencepiece Models II

- BERT (later in lecture) uses a variant of the wordpiece model
  - (Relatively) common words are in the vocabulary:
    - *at, fairfax, 1910s*
  - Other words are built from wordpieces:
    - *hypatia = h ##yp ##ati ##a*
- If you're using BERT in an otherwise word based model, you have to deal with this

```
from tokenizers import BertWordPieceTokenizer
from transformers import BertTokenizer
```

https://pixy.org/src/425/4254306.png

fastText

- "Enriching Word Vectors with Subword Information"
  - Developed by Facebook
  - https://fasttext.cc

- Aim: a next generation efficient word2vec-like word representation library, but better for
  - rare words and
  - languages with lots of morphology

- An extension of the word2vec skip-gram model with character n-grams

Bojanowski, Piotr, et al. "Enriching word vectors with subword information." *Transactions of the Association for Computational Linguistics* 5 (2017): 135-146.

fastText

- A word is represented as a bag-of-character n-grams.
  - E.g. for n=3 the word *Fishing:*
    - $G_{fishing} = \_fi,\ fis,\ ish,\ shi,\ hin,\ ing,\ ng\_,\ \_fishing\_$

- A word vector is then represented as the sum of its n-grams.
  - In word2vec:

$$sim(w_u, w_v) = u^T v$$

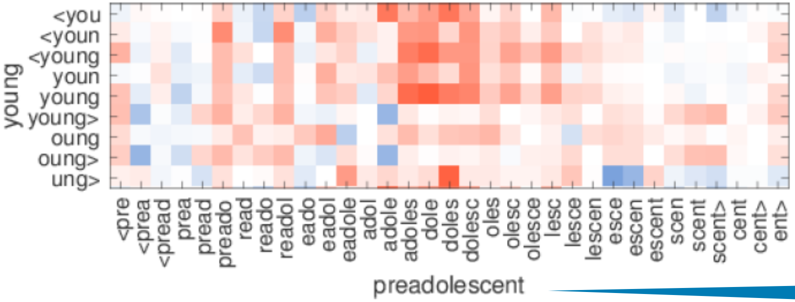
  - In fastText

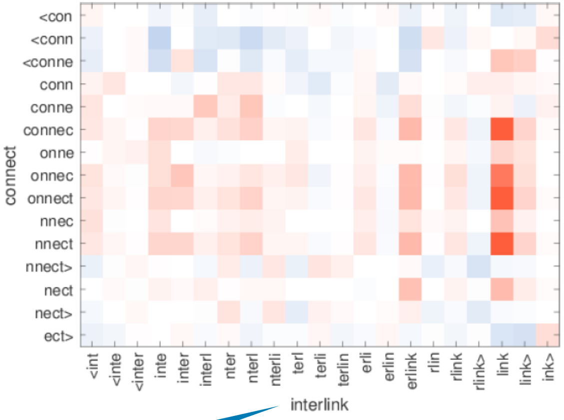
$$sim(w_u, w_v) = \sum_{g \in G_v} u^T v$$

fastText Results

Rare words

query	tiling	tech-rich	english-born	micromanaging	eateries	dendritic
sisg	tile flooring	tech-dominated tech-heavy	british-born polish-born	micromanage micromanaged	restaurants eaterie	dendrite dendrites
sg	bookcases built-ins	technology-heavy .ixic	most-capped ex-scotland	defang internalise	restaurants delis	epithelial p53





OOV

Example fasttext Keras Implementation

- https://keras.io/zh/examples/imdb_fasttext/

```
from __future__ import print_function
import numpy as np
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Embedding
from keras.layers import GlobalAveragePooling1D
from keras.datasets import imdb

def create_ngram_set(input_list, ngram_value=2):
        return set(zip(*[input_list[i:] for i in range(ngram_value)]))
```

Extract a set of n-grams from a list of integers.
>>> create_ngram_set([1,4,9,4,1,4], ngram_value=3)
[(1,4,9),(4,9,4),(9,4,1),(4,1,4)]

Example fasttext Keras Implementation

```
def add_ngram(sequences, token_indice, ngram_range=2):
    new_sequences = []
    for input_list in sequences:
        new_list = input_list[:]
        for ngram_value in range(2, ngram_range + 1):
            for i in range(len(new_list) - ngram_value + 1):
                ngram = tuple(new_list[i:i + ngram_value])
                if ngram in token_indice:
                    new_list.append(token_indice[ngram])
        new_sequences.append(new_list)
    return new_sequences
```

```
Augment the input list of list (sequences) by appending n-grams values.
   Example: adding tri-gram
   >>> sequences = [[1, 3, 4, 5], [1, 3, 7, 9, 2]]
   >>> token_indice = {(1,3): 1337, (9,2): 42, (4,5): 2017, (7,9,2): 2018}
   >>> add_ngram(sequences, token_indice, ngram_range=3)
   [[1, 3, 4, 5, 1337, 2017], [1, 3, 7, 9, 2, 1337, 42, 2018]]
```

Example fasttext Keras Implementation

```
# ngram_range = 2 will add bi-grams features
ngram_range = 1
max_features = 10000
maxlen = 400
batch_size = 32
embedding_dims = 50
epochs = 5

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
```

Example fasttext Keras Implementation

```
if ngram_range > 1:
    ngram_set = set()
    for input_list in x_train:
        for i in range(2, ngram_range + 1):
            set_of_ngram = create_ngram_set(input_list, ngram_value=i)
            ngram_set.update(set_of_ngram)

    start_index = max_features + 1
    token_indice = {v: k + start_index for k, v in enumerate(ngram_set)}
    indice_token = {token_indice[k]: k for k in token_indice}

    max_features = np.max(list(indice_token.keys())) + 1

    x_train = add_ngram(x_train, token_indice, ngram_range)
    x_test = add_ngram(x_test, token_indice, ngram_range)
```

Example fasttext Keras Implementation

```
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

model = Sequential()
model.add(Embedding(max_features, embedding_dims, input_length=maxlen))
model.add(GlobalAveragePooling1D())
model.add(Dense(1, activation='sigmoid'))
```

GlobalAveragePooling1D averages the embeddings of all words in the document

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

History = model.fit(partial_x_train, partial_y_train, batch_size=batch_size,
        epochs=epochs, validation_data=(x_val, y_val))

results = model.evaluate(x_test, y_test)
```

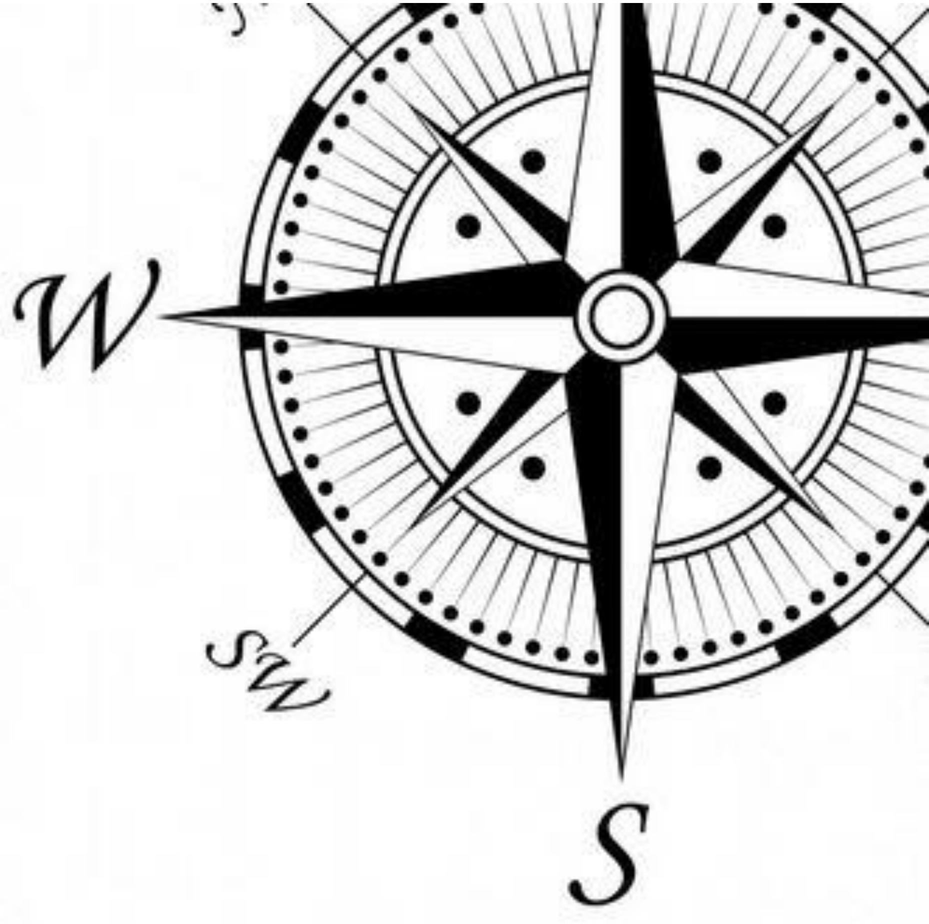
Exercise

- Implement the previous example using fasttext idea of n-gram representations instead of word2vec/GloVe word representations only.
- Compare the results
  - fasttext vs. word2vec
  - flatten() vs. GlobalAveragePooling1D()

Start 10 9 8 7 6 5 4 3 2 1 End

Topics Today

1. Sub-Word Models
2. **Document Embeddings**



Representation of a Document

- Document ≈ long text, paragraph, sentence
- Many options:
  - Bag-of-words
  - TF-IDF
  - N-grams
- Problem:
  - Same content ≠ same words

Obama speaks to the media in Illinois

The President greets the press in Chicago

- Solution: methods, that capture semantics:
  - E.g. topic models
  - **Something with word embeddings**

Kusner, Matt, et al. "From word embeddings to document distances." *International conference on machine learning*. 2015.

Document Embeddings

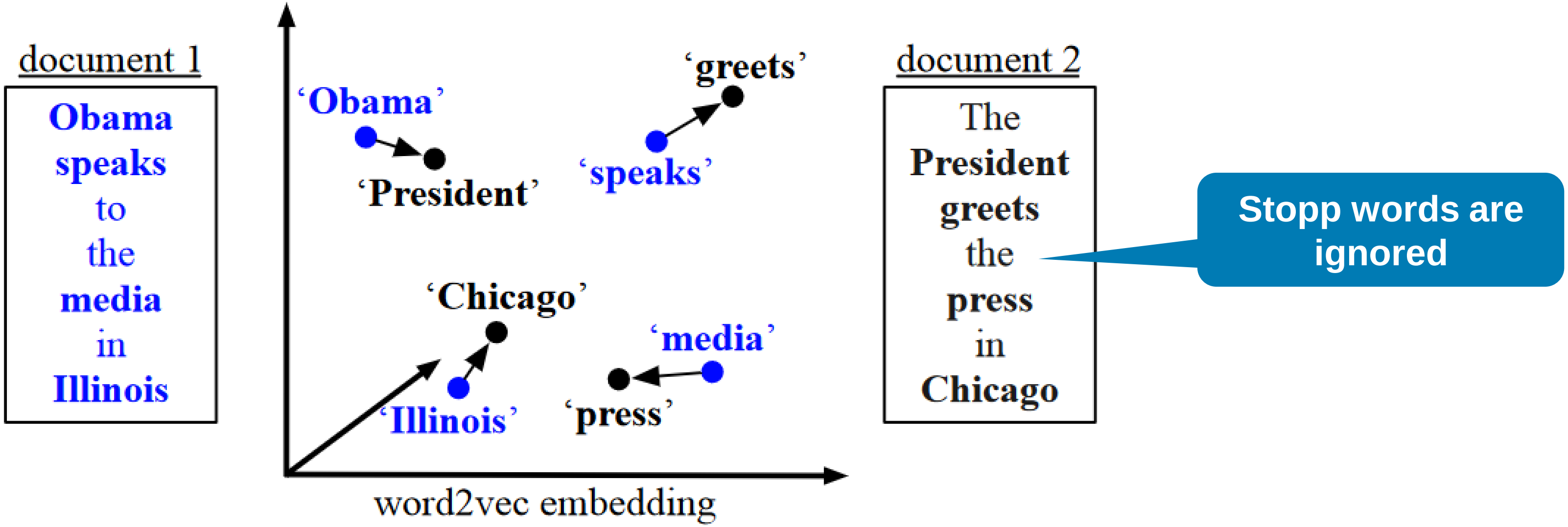
- Most simple appraoch (usually works quite well!)
  - (Weighted) **Average** of individual word embeddings

- More sophisticated:
  - **Word movers distance**
  - „Bag-of-word-embeddings“
  - Word vectors are mapped between documents

- Data-driven:
  - Direct learning of vectors for groups of words or whole documents
  - Most popular: **doc2vec**
  - Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." *International conference on machine learning*. 2014.

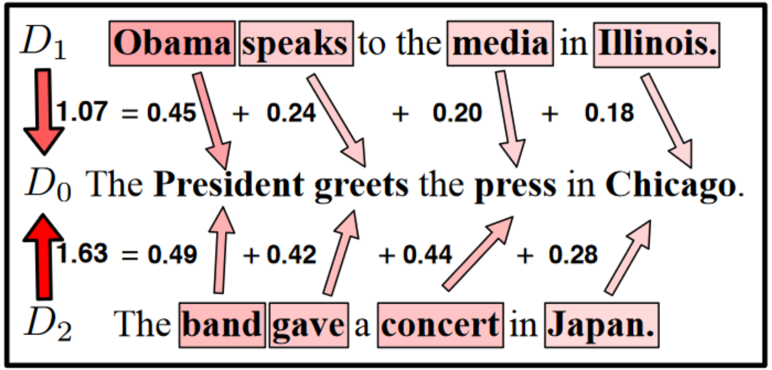
Word Movers Distance I

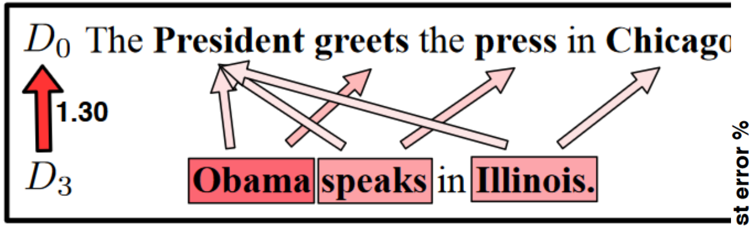
- Simlarity of documents
  - Sum of the minimal distances in the embedding space to move from one document to the other



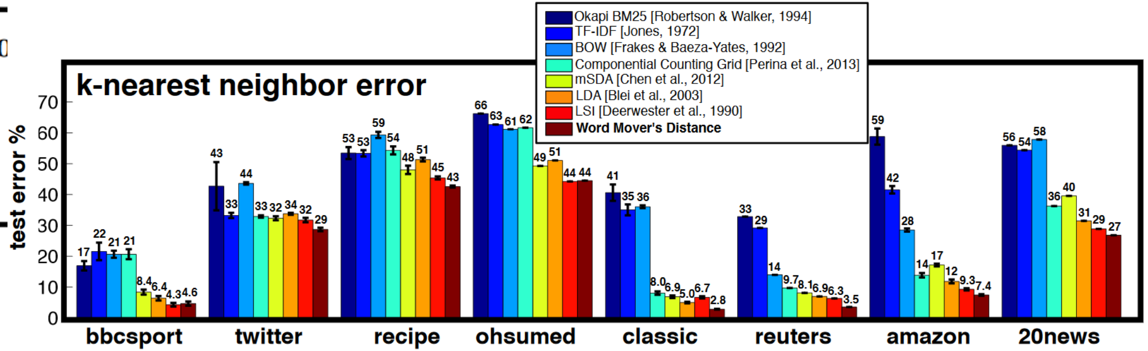
Kusner, Matt, et al. "From word embeddings to document distances." *International conference on machine learning*. 2015.

Word Movers Distance II



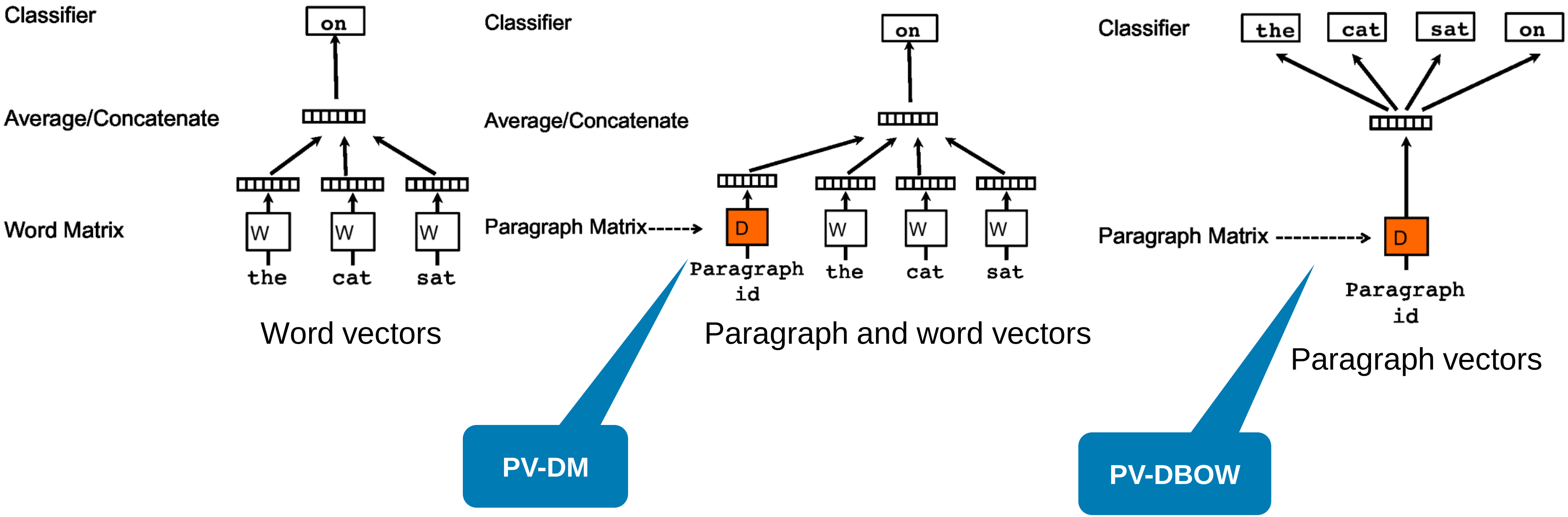


NAME	n	BOW DIM.	UNIQUE WORDS (AVG)	$\lvert\mathcal{Y}\rvert$
BBCSPORT	517	13243	117	5
TWITTER	2176	6344	9.9	3
RECIPE	3059	5708	48.5	15
OHSUMED	3999	31789	59.2	10
CLASSIC	4965	24277	38.6	4
REUTERS	5485	22425	37.1	8
AMAZON	5600	42063	45.0	4
20NEWS	11293	29671	72	20



Kusner, Matt, et al. "From word embeddings to document distances." *International conference on machine learning*. 2015.

Doc2Vec I



Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." *International conference on machine learning*. 2014.

Doc2Vec II

- Both paragraph vector models have two modi:
  - Training:
    - The model is trained with training data
    - Embedding vectors and softmax weight are fitted
  - Testing/in production:
    - New, unseen documents arrive
    - Only embedding vectors are learned; softmax-weights remain unchanged
- Testing is faster than training
  - But still not cheap (learning word embeddings)

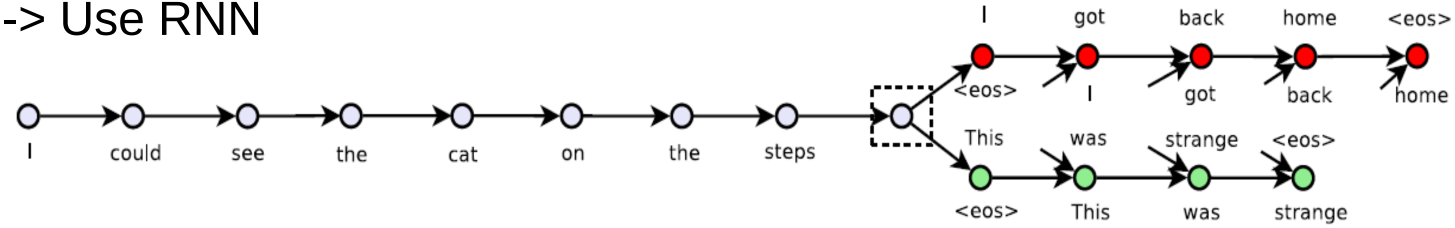
Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

Concatenated PV-DM ∘ PV-DBOW

Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." *International conference on machi* *ing*. 2014.

Skip-Thought Vectors

- The models we discussed so far embed texts as **the sum of their words** (lexical semantics).
- Clearly there is a lot missing from these representations:
  - “man bites dog” = “dog bites man”
  - “the quick, brown fox jumps over the lazy dog”
    = “the lazy fox over the brown dog jumps quick”
- How can we model text structure as well as word meanings?
- -> Use RNN



Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. *NIPS.*

Results: Similar Sentences

Query and nearest sentence
he ran his hand inside his coat , double-checking that the unopened letter was still there . he slipped his hand between his coat and his shirt , where the folded copies lay in a brown envelope .
im sure youll have a glamorous evening , she said , giving an exaggerated wink . im really glad you came to the party tonight , he said , turning to her .
although she could tell he had n't been too invested in any of their other chitchat , he seemed genuinely curious about this . although he had n't been following her career with a microscope , he 'd definitely taken notice of her appearances .
an annoying buzz started to ring in my ears , becoming louder and louder as my vision began to swim . a weighty pressure landed on my lungs and my vision blurred at the edges , threatening my consciousness altogether .
if he had a weapon , he could maybe take out their last imp , and then beat up errol and vanessa . if he could ram them from behind , send them sailing over the far side of the levee , he had a chance of stopping them .
then , with a stroke of luck , they saw the pair head together towards the portaloos . then , from out back of the house , they heard a horse scream probably in answer to a pair of sharp spurs digging deep into its flanks .
" i 'll take care of it , " goodman said , taking the phonebook . " i 'll do that , " julia said , coming in .
he finished rolling up scrolls and , placing them to one side , began the more urgent task of finding ale and tankards . he righted the table , set the candle on a piece of broken plate , and reached for his flint , steel , and tinder .

Approximately two weeks of training on a billion-word Books corpus

Hard to evaluate!

Semantic Relatedness Evaluation

- SICK semantic relatedness task: score sentences for semantic similarity from 1 to 5 (average of 10 human ratings)

- Sentence A: A man is jumping into an empty pool
  Sentence B: There is no biker jumping in the air
  Relatedness score: 1.6
- Sentence A: Two children are lying in the snow and are making snow angels
  Sentence B: Two angels are making snow on the lying children
  Relatedness score: 2.9
- Sentence A: The young boys are playing outdoors and the man is smiling nearby
  Sentence B: There is no boy playing outdoors and there is no man smiling
  Relatedness score: 3.6
- Sentence A: A person in a black jacket is doing tricks on a motorbike
  Sentence B: A man in a black jacket is doing tricks on a motorbike
  Relatedness score: 4.9

Note: a separate model is trained to predict the scores from pairs of embedded sentences.

Semantic Entailment Evaluation

- SICK semantic entailment task: score sentences for relations: ENTAILMENT, CONTRADICTION, NEUTRAL:

- Sentence A: Two teams are competing in a football match
  Sentence B: Two groups of people are playing football
  Entailment judgment: ENTAILMENT
-
- Sentence A: The brown horse is near a red barrel at the rodeo
  Sentence B: The brown horse is far from a red barrel at the rodeo
  Entailment judgment: CONTRADICTION
-
- Sentence A: A man in a black jacket is doing tricks on a motorbike
  Sentence B: A person is riding the bicycle on one wheel
  Entailment judgment: NEUTRAL

Hard to compare: Models trained with these specific objectives outperform general embedding models!

doc2vec Example

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
import numpy as np
sentences = ["I ate dinner.", "We had a three-course meal.", "Brad came to dinner with us.",
         "He loves fish tacos.", "In the end, we all felt like we ate too much.",
         "We all agreed; it was a magnificent evening."]
tokenized_sent = []
for s in sentences:
    tokenized_sent.append(word_tokenize(s.lower()))
tokenized_sent
def cosine(u, v):
    return np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
tagged_data = [TaggedDocument(d, [i]) for i, d in enumerate(tokenized_sent)]
model = Doc2Vec(tagged_data, vector_size = 20, window = 2, min_count = 1, epochs = 100)
test_doc = word_tokenize("I had pizza and pasta".lower())
test_doc_vector = model.infer_vector(test_doc)
model.docvecs.most_similar(positive = [test_doc_vector])
```

GENSIM
topic modelling for humans
https://radimrehurek.com/gensim/

Exercise

- Use average word2vec word vectors to represent sentences.
- Use doc2vec to compute sentence vectors.
  - Model available in Gensim
    - https://radimrehurek.com/gensim/models/doc2vec.html
- Compare most similar sentences for both representations

- More links:
  - https://ireneli.eu/2016/07/27/nlp-05-from-word2vec-to-doc2vec-a-simple-example-with-gensim/
  - https://www.analyticsvidhya.com/blog/2020/08/top-4-sentence-embedding-techniques-using-python/

Start 10 9 8 7 6 5 4 3 2 1 End

Lerning Goals for this Chapter

- Be able to explain the diffences between embedding models
  - Character level
  - Subword level
  - Word level

- Understand document/paragraph/sentence embeddings
  - Word movers distance
  - Doc2vec

- Partly based on Chris Manning‘s lecture 2019
  - https://www.youtube.com/watch?v=9oTHFx0Gg3Q&list=PLoROMvodv4rOhcuXMZkNm7j3fVwBBY42z&index=12