inholland hogeschool

Deep Learning

Deep Learning for Text

Chapter 11.1-11.3

Vera Hollink
2025

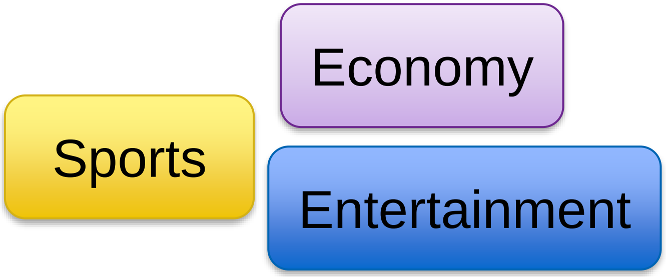
Natural Language Processing (NLP)

- Natural language = human language
  - Vocabulary changes
  - Grammar not well-defined

- NLP
  - do something useful with natural language
  - ≠ understanding

Some NLP Tasks

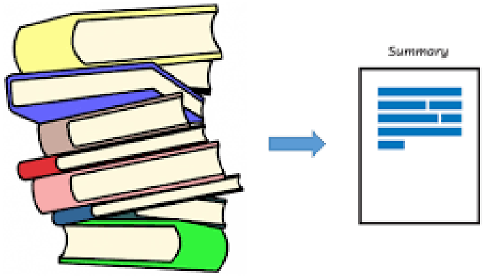
- Text classification



- Content filtering

- Sentiment analysis

- Translation
- Summarization



Text preprocessing

Text preprocessing

```
The quick brown fox jumps over the lazy dog.
```

1. Normalization/standardization

```
the quick brown fox jumps over the lazy dog
```

2. Tokenization

```
“the” “quick” “brown” “fox” “jumps” “over” “the”
“lazy” “dog”
```

3. Indexing

```
17 321 490 21 339 3021 17 591 111
```

4. Encoding

```
[[0,0,0,0,1,0], [[0,1,0,0,0,0], [[1,0,0,0,0,0], …]
```

Normalization/standardization

- Lowercase

  `The` ➔ `the`
- Remove punctuation

  `. ? " …`
- Convert special characters

  `résumé` ➔ `resume`
- Stemming

  `foxes => fox`

  `approximation` ➔ `approximat`

Disadvantage: information is lost

Advantage: less training data needed

Tokenization

the quick brown fox jumps over the lazy dog

- Words

“the” “quick” “brown” “fox” “jumps” “over” “the” “lazy” “dog”

- N-grams: sequences of N words

2-grams:

“the quick” “quick brown” “brown fox” “fox jumps” “jumps over” “over the” “the lazy” “lazy dog”

3-grams:

“the quick brown” “quick brown fox” “brown fox jumps” “fox jumps over” “jumps over the” “over the lazy” “the lazy dog”

- Characters

“t” “h” “e” “q” “u” “i” “c” “k” “b” …

Indexing

- Assign number to each token

  `the` ➔ `17`

  `quick` ➔ `321`

- Use only N most frequent tokens (e.g. 10000)

  other tokens get index 1

- Create vectors

  `[17 321 490 21 339 3021 17 591 111]`

- If fixed length needed, pad with 0's:

  Length 12:

  `[17 321 490 21 339 3021 17 591 111 0 0 0]`

Text preprocessing in Keras

Preprocessing module

```
from tensorflow.keras.layers import TextVectorization
dataset = ['The brown dog jumps.','Dog jumps over the fox.']
text_vectorization = TextVectorization(output_mode='int')
text_vectorization.adapt(dataset)
text = 'The quick brown dog jumps.'
encoded_text = text_vectorization(text)
print(encoded_text)
```

Default: lowercase, remove punctuaction, split on whitespace There are many alternative options

1 for unknown word

Apply to new data

Create vocabulary: give each token a number

```
tf.Tensor([2 1 7 4 3], shape=(5,), dtype=int64)
```

Text preprocessing

`The quick brown fox jumps over the lazy dog.`

1. Normalization/standardization

`the quick brown fox jumps over the lazy dog`

2. Tokenization

`“the” “quick” “brown” “fox” “jumps” “over” “the” “lazy” “dog”`

3. Indexing

`17 321 490 21 339 3021 17 591 111`

4. **Encoding**

`[[0,0,0,0,1,0], [[0,1,0,0,0,0], [[1,0,0,0,0,0], …]`

Bag-of-word representations

1. Multi-hot encoding

Doc id	dog	fox	jump	over	…
1	1	0	1	0	…
2	1	1	0	0	…
3	1	0	0	0	…
4	0	1	0	0	…
5	0	1	1	1	…
…	…	…	…	…	…

```
text_vectorization = TextVectorization(max_tokens = 10000,
    output_mode='multi-hot')
```

Bag-of-word representations

2. Frequency encoding

Doc id	dog	fox	jump	over	…
1	2	0	1	0	…
2	1	2	0	0	…
3	4	0	0	0	…
4	0	1	0	0	…
5	0	6	2	1	…
…	…	…	…	…	…

```
text_vectorization = TextVectorization(max_tokens = 10000,
    output_mode='count')
```

Bag-of-word representations

3. Tf.idf encoding

Value = importance score of the token

Doc id	dog	fox	jump	over	…
1	2.6	0	0.2	0	…
2	0.9	2	0	0	…
3	3.2	0	0	0	…
4	0	0.9	0	0	…
5	0	2.1	1.9	0.1	…
…	…	…	…	…	...

```
text_vectorization = TextVectorization(max_tokens = 10000,
    output_mode='tf_idf')
```

Bag-of-word representations

N-gram encoding (multi-hot)

Doc id	dog_jump	jump_over	over_the	…
1	1	0	0	…
2	0	0	0	…
3	0	0	0	…
4	0	0	0	…
5	0	1	1	…
…	…	…	…	…

```
text_vectorization = TextVectorization(n_grams=2,
    max_tokens = 10000, output_mode='multi-hot')
```

count or tf_idf
also possible

Exercise: text preprocessing and encoding

Two documents

```
“Are oranges always orange?”
“The cat ate the oranges.”
```

- Apply stemming and other normalization techniques
- Apply tokenization
- Encode using
  1. Bag-of-words with multi-hot encoding
  2. Bag-of-words with frequency encoding
  3. Bag-of-words with 2-gram multi-hot encoding

Exercise: text preprocessing and encoding

Two documents

document 1: `“Are oranges always orange?”`

document 2: `“The cat ate the oranges.”`

- Apply stemming and other normalization techniques
- Apply tokenization

```
[are, orange, always, orange]
[the, cat, ate, the, orange]
```

Exercise: text preprocessing and encoding

Bag-of-words with multi-hot encoding

Doc id	are	orange	always	the	cat	ate
1	1	1	1	0	0	0
2	0	1	0	1	1	1

Bag-of-words with frequency encoding

Doc id	are	orange	always	the	cat	ate
1	1	2	1	0	0	0
2	0	1	0	2	1	1

Exercise: text preprocessing and encoding

Bag-of-words with 2-gram multi-hot encoding

```
[are_orange, orange_always, always_orange]
[the_cat, cat_ate, ate_the, the_orange]
```

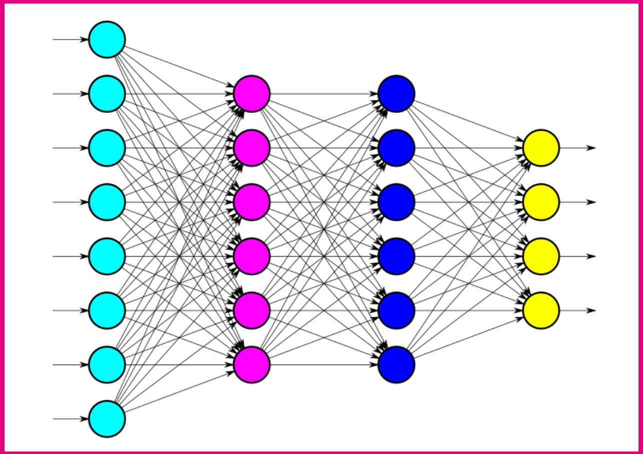
Doc id	are_orange	orange_always	always_orange	the_cat	cat_ate	ate_the	the_orange
1	1	1	1	0	0	0	0
2	0	0	0	1	1	1	1

Deep Learning models for text

1. Dense models
2. Recurrent Neural Networks
3. Transformers (next week)

Deep Learning models for text

1. **Dense models**
2. Recurrent Neural Networks
3. Transformers (next week)



Dense model

- Input: document encoded as bag-of-words

  Multi-hot: [0,0,**1**,0,0,0,0,0,0,**1**,0,0,0,0,0,**1**,0…]

  Frequency: [0,0,**2**,0,0,0,0,0,0,**1**,0,0,0,0,0,**3**,0…]

  tf_idf: [0,0,**2.4**,0,0,0,0,0,0,**0.9**,0,0,0,0,0,**1.3**,0…]

- Dense model:

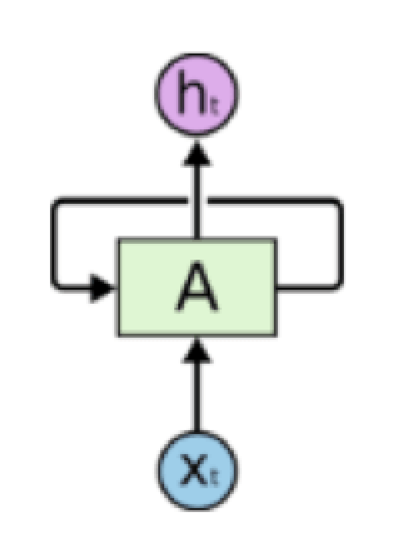
Number of words in the vocabulary

```
model = Sequential()
model.add(Dense(50, input_shape=(n_words,), activation='relu'))
model.add(Dense(1, activation='sigmoid’))
```

- Disadvantage: word order is lost
  - Partly solved with N-grams, but very short sequences

Deep Learning models for text

1. Dense models
2. **Recurrent Neural Networks**
3. Transformers (next week)



RNN

- Sequence processing: (bidirectional) LSTM

```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
```

- One-hot encoding:
  - Each word is a vector with exactly one 1
    [0, 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, …]
  - A sample (document) is a 2-dimensional vector
    [[0, 0, 0, 0, 0, 0, **1**, 0, 0, , …], [0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, …], [0, 0, **1**, 0, 0, 0, 0, 0, 0, …], …]
- A batch has a number of sequences with the same length
  - Use cut-off and padding with 0's
  - One batch is for instance 256x20000x600 ➔ training is slow

samples x max_tokens x sentence_length

RNNs and Convnets for text

- (bidirectional) LSTM
- (bidirectional) GRU
- 1D Convolutional Network

- Disadvantage:
  - Slow because input is huge
    samples x max_tokens x sentence_length

One-hot embedding vs word embeddings

- One-hot embedding:

  Dog

  `[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, …]`

  Cat

  `[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, …]`

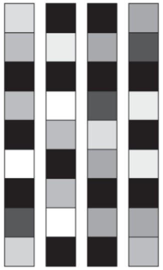
- Word embeddings:

  Dog

  `[0.12, 0.30, 0.20, 0.24]`

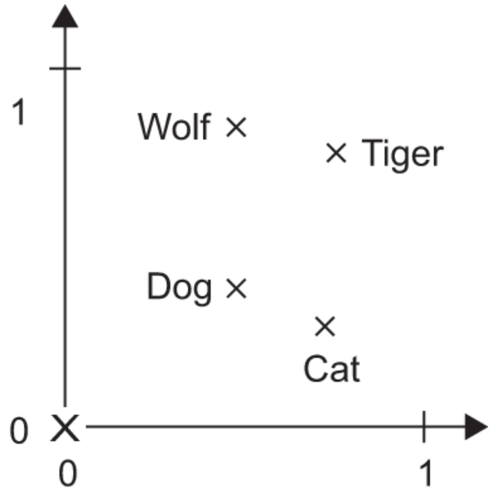
  Cat

  `[0.73, 0.13, 0.40, 0.44]`

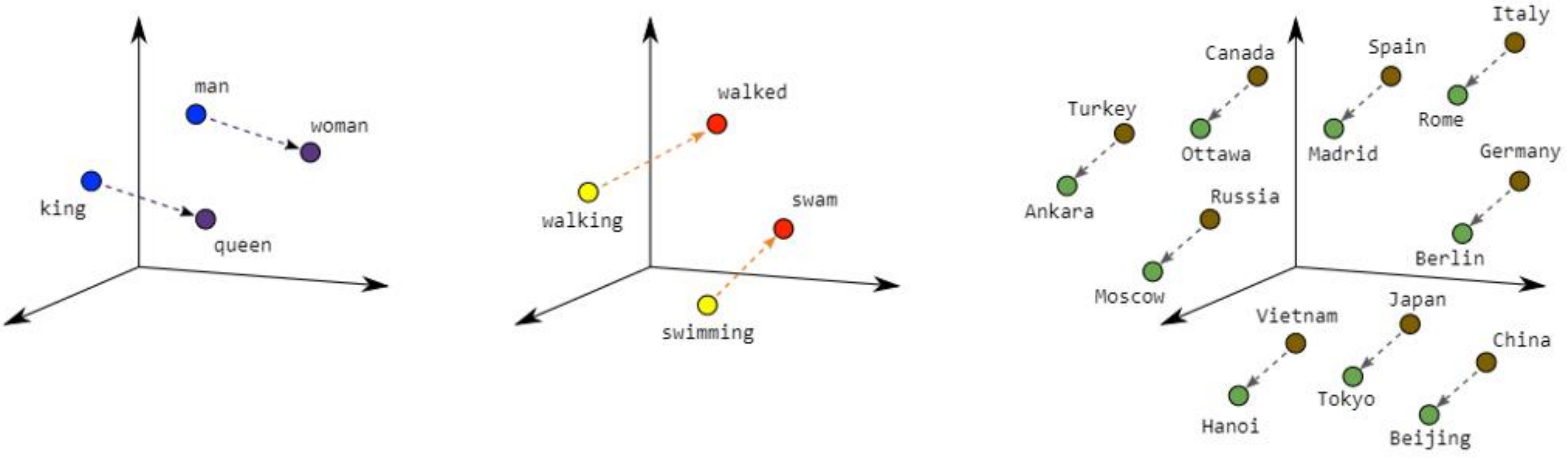


Advantages word embeddings

- Smaller representation
  - E.g. 20000 x 600 becomes 256 x 600
- Vector distances can represent meaning
  - Similar words can have similar vectors
    e.g. vector `cactus` closer to vector `aloe` than to vector `cat`
- Meaningful dimensions
  - Gender, singular/plural, …
  - Emerging, not hard-coded



Advantages word embeddings



Male-Female Verb Tense Country-Capital

Learning word embeddings

- Transform word indexes to word vector

[17 321 490 21 339 3021 17 591 111 0 0 0] ➔

[[0.72, 0.34, 0.1, ..], [0.32, 0.70, …], …]

- Initialize randomly, learn to structure space through backpropagation (more similar words get closer vectors)

Length of word vector

Number of words in sequences

Size of the vocabiary

```
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens,
   output_dim=256, input_length=600, mask_zero=True)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoi
model = keras.Model(inputs, outputs)
```

Make sure padding (trailing 0's) is not used in training.

Using pretrained word embeddings

- Meaning/embedding not specific for data set or task
- Use existing embedding
  1. Get embedding matrix:
     pretrained vector for each word
  2. Initialize embedding with matrix
  3. Fix embedding (layer not trainable)

```
embedding_layer = layers.Embedding(max_tokens, embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix), trainable=False, mask_zero=True
)
```

Popular pretrained word embeddings

word2vec:

- made by Google
- based on news data
- vector length 300

GloVe:

- made by Stanford,
- based on various sources (various versions available), such as Wikipedia and WWW crawl and Twitter,
- vector length 25-300

Using pretrained word embeddings

- Advantages?
  - Less training and data needed
  - Based on large corpus
- Disadvantages?
  - Not specialized for your task (e.g. sentiment analysis)
  - Not specialized for your texts (e.g. reviews)

Deep Learning models for text

1. Dense models
2. Recurrent Neural Networks
3. **Transformers (next week)**

Test your understanding!