AI in Built Environment
DCP4300

Week 11: Natural Language Processing

Part A: Word Embeddings

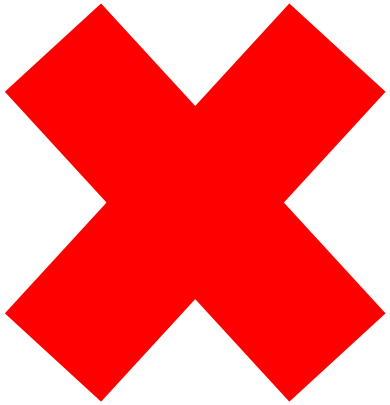
Dr. Chaofeng Wang
Jianhao Gao (TA)

University of Florida
College of Design Construction and Planning

What is a good way for computers to represent words ?

Remember one-hot encoding?

This is a puppy

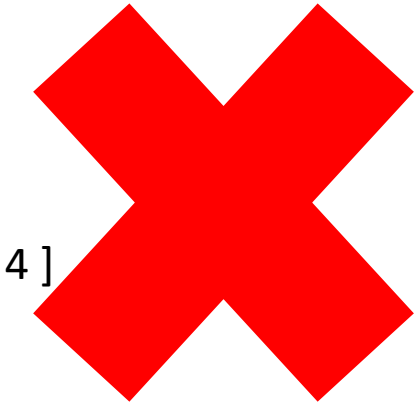


This	[1, 0, 0, 0]
is	[0, 1, 0, 0]
a	[0, 0, 1, 0]
puppy	[0, 0, 0, 1]

Resulting in a sparse vector, inefficient at storing and computing missing information: Words are equally apart from each other, relationship of words hard to be preserved

Encode each word with a unique number

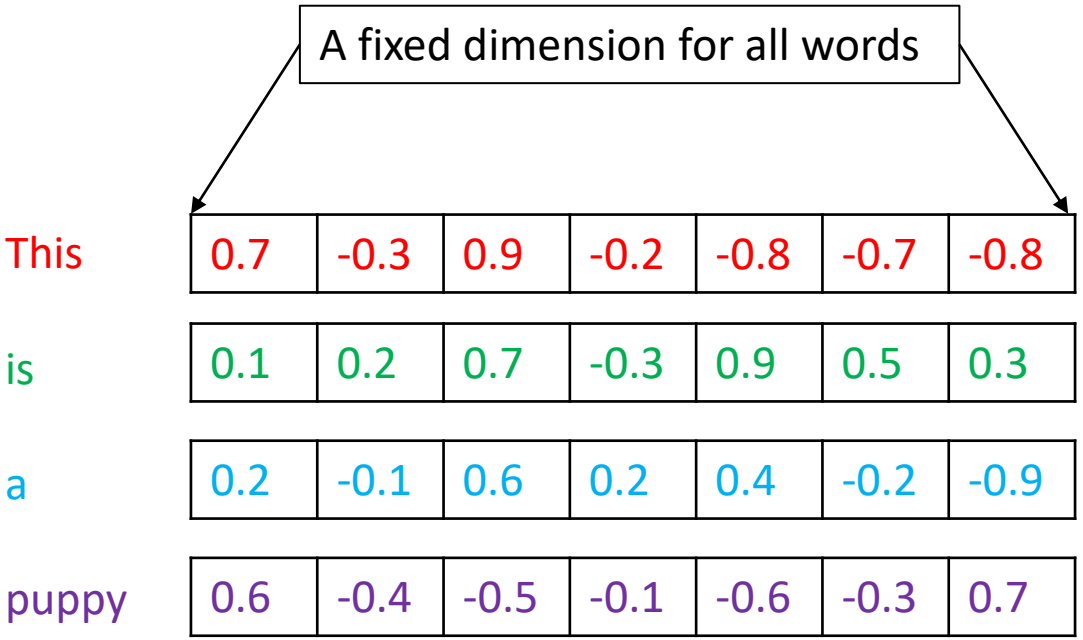
This is a puppy [1 2 3 4]



Computationally efficient
Missing information: relationship of words hard to be preserved

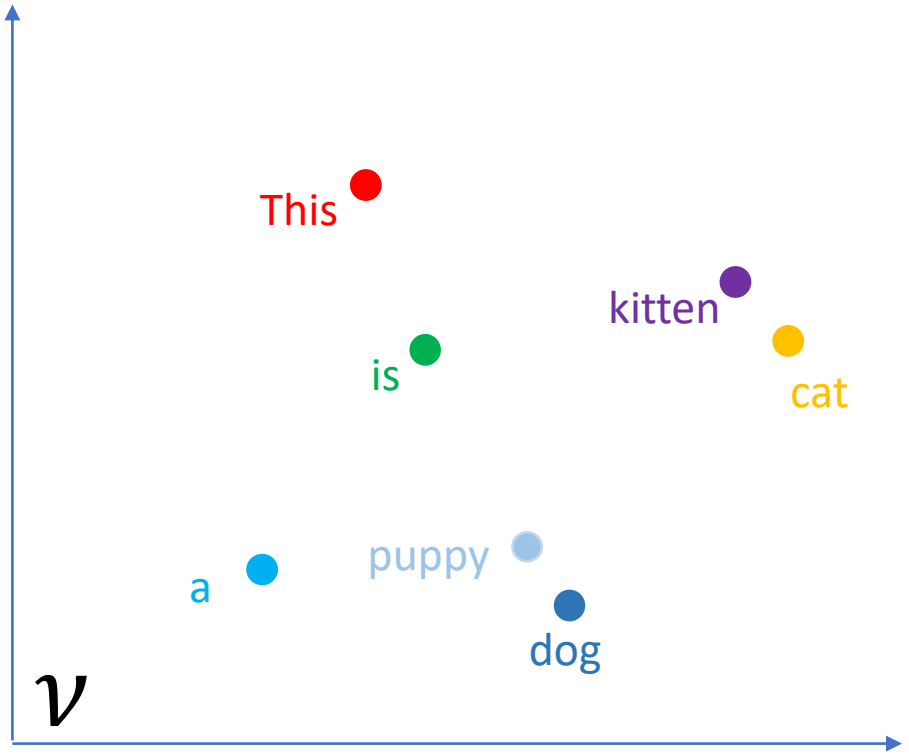
Word Vector Representations

Converting words into fixed-length **vectors**.
So that text can be manipulated computationally.



word embedding

Word Embedding



Learn the embedding using a neural net

Input is a context word -> predict the target word

Objective:
To maximize the probability of predicting
target words given the context word.

Sentence:

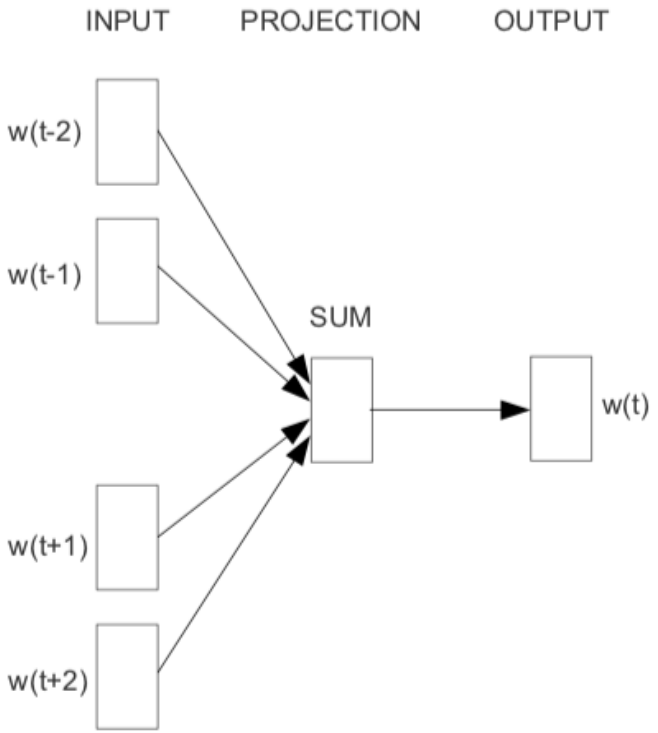
I would like to have an apple for breakfast.

Word2Vec

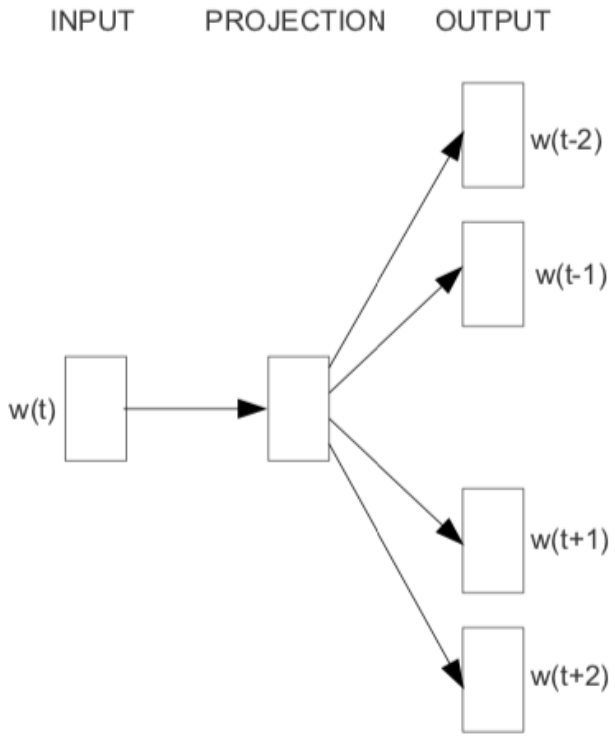
A family of algorithms that can learn embeddings from data:

Continuous Bag-of-Words (CBOW)

Skip-Gram



CBOW



Skip-gram

Learn more about Word2Vec
https://www.tensorflow.org/tutorials/text/word2vec

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. https://arxiv.org/abs/1301.3781

Word2Vec: CBOW

Target word

I would like to have an apple.

Context word

like

to

an

apple

One context word:
one hot embedding of
dimension 7

W: matrix of (7, 3)

average

Latent space: dimension of 3

W': matrix of (3, 7)

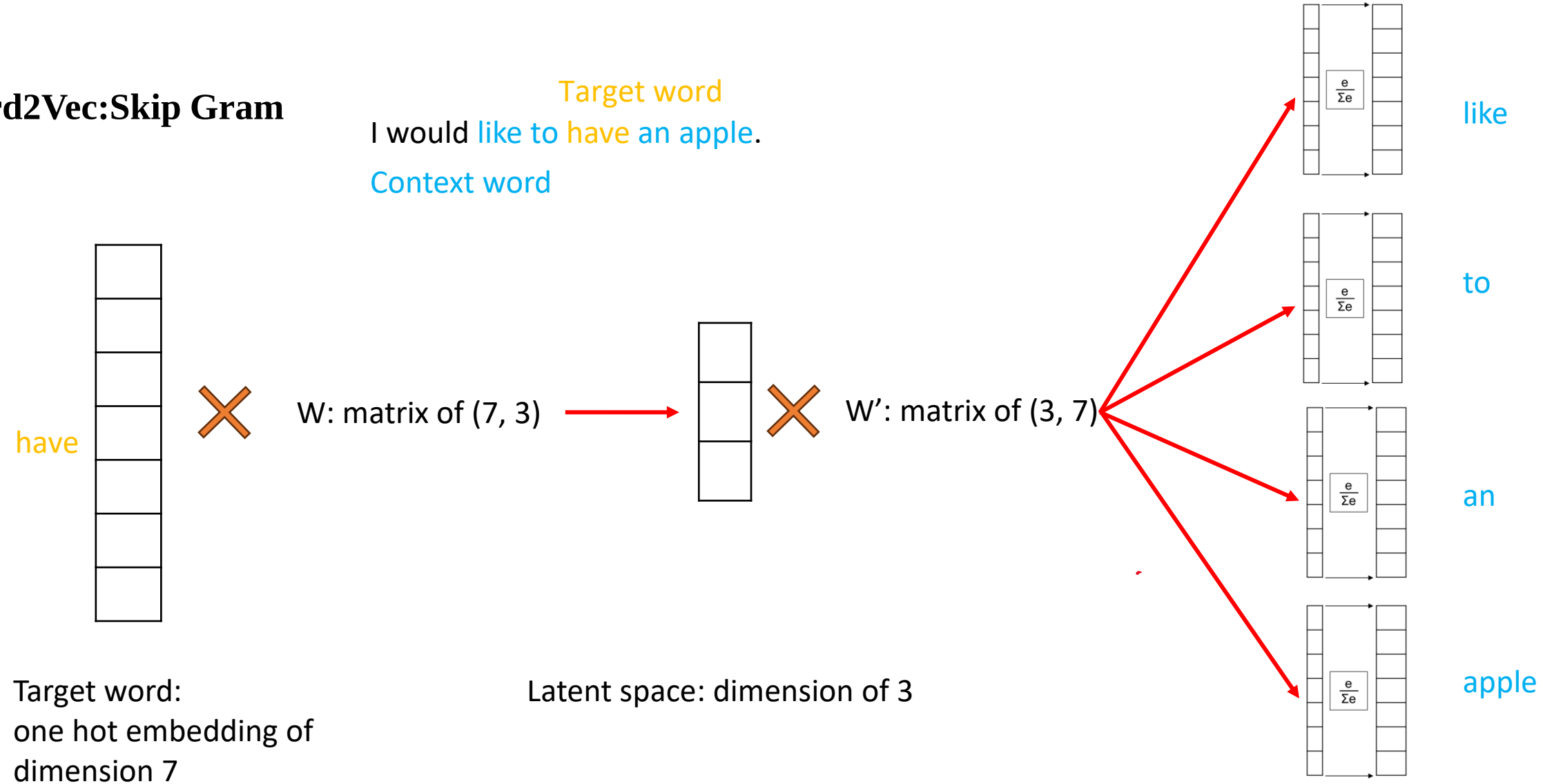
$\frac{e}{\Sigma e}$

Word2Vec:Skip Gram

Target word

I would like to have an apple.

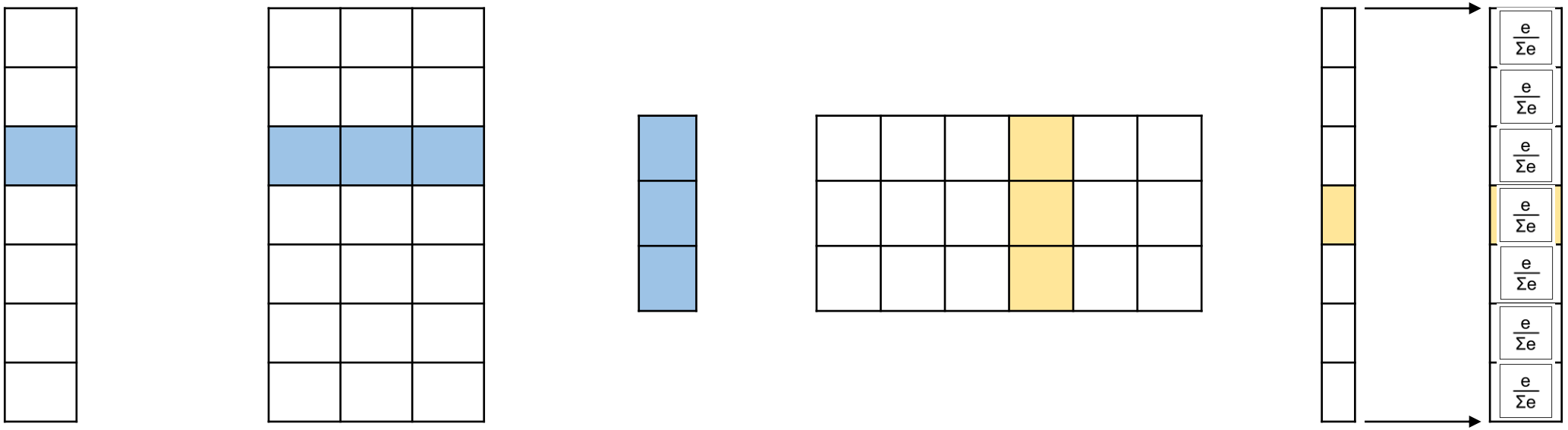
Context word



Target word:
one hot embedding of
dimension 7

Latent space: dimension of 3

Softmax is time consuming



$z_t = e'_t * e_c$

Softmax is time consuming

One solution is: Negative-Sampling

I would like to have an apple for breakfast.

New problem: For a given context, predict if a random word is its target or not.

	Context	Word	Target or not (window size 2)
Step 1: positive	apple	an	1
Step 2: negative	apple	would	0
	apple	like	0
	apple	to	0

Softmax is time consuming

One solution is: Negative-Sampling

Context	Target	Target or not (window size 2)
apple	an	1
apple	would	0
apple	like	0
apple	to	0

Softmax: $$P(t|c) = \frac{\exp(e_t' \cdot e_c)}{\sum_{i=1}^{n} \exp(e_i' \cdot e_c)}$$

Replace with logistic regression: $$P(y = 1|c, t) = \sigma(e_t' \cdot e_c)$$

Faster

Demo: Word2vec – skip-gram with negative sampling

https://www.tensorflow.org/tutorials/text/word2vec

Tokenization

Cutting a text/document into pieces (maybe throwing away certain characters such as punctuation)

Input: Today is a good day  , isn't it?

Tokenization 1:

Today	is	a	good	day	isn't	it

Tokenization 2:

Today	is	a	good	day		isn't	it	?

Tools

NLTK
https://www.nltk.org/

A platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning

Tokenize and tag some text:

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Identify named entities:

```
>>> entities = nltk.chunk.ne_chunk(tagged)
>>> entities
Tree('S', [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'),
           ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'),
      Tree('PERSON', [('Arthur', 'NNP')]),
           ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'),
           ('very', 'RB'), ('good', 'JJ'), ('.', '.')])
```

Tools

spaCy
https://spacy.io/

```
# pip install -U spacy
# python -m spacy download en_core_web_sm
import spacy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. “I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn’t "
        "worth talking to,” said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```

- Support for **64+ languages**
- **64 trained pipelines** for 19 languages
- Multi-task learning with pretrained **transformers** like BERT
- Pretrained **word vectors**
- State-of-the-art speed
- Production-ready **training system**
- Linguistically-motivated **tokenization**
- Components for **named entity** recognition, part-of-speech tagging, dependency parsing, sentence segmentation, **text classification**, lemmatization, morphological analysis, entity linking and more
- Easily extensible with **custom components** and attributes
- Support for custom models in **PyTorch**, **TensorFlow** and other frameworks
- Built in **visualizers** for syntax and NER
- Easy **model packaging**, deployment and workflow management
- Robust, rigorously evaluated accuracy