A wireframe model of a human head in profile, facing right, is superimposed on a blue background featuring a circuit board pattern. The wireframe is composed of white lines. The background has various labels like 'CM42', 'CM50', 'CM54', 'CM13', 'CM14', 'CM15', 'CM16', 'CM17', 'CM18', 'CM19', 'CM20', 'CM21', 'CM22', 'CM23', 'CM24', 'CM25', 'CM26', 'CM27', 'CM28', 'CM29', 'CM30', 'CM31', 'CM32', 'CM33', 'CM34', 'CM35', 'CM36', 'CM37', 'CM38', 'CM39', 'CM40', 'CM41', 'CM42', 'CM43', 'CM44', 'CM45', 'CM46', 'CM47', 'CM48', 'CM49', 'CM50', 'CM51', 'CM52', 'CM53', 'CM54', 'CM55', 'CM56', 'CM57', 'CM58', 'CM59', 'CM60', 'CM61', 'CM62', 'CM63', 'CM64', 'CM65', 'CM66', 'CM67', 'CM68', 'CM69', 'CM70', 'CM71', 'CM72', 'CM73', 'CM74', 'CM75', 'CM76', 'CM77', 'CM78', 'CM79', 'CM80', 'CM81', 'CM82', 'CM83', 'CM84', 'CM85', 'CM86', 'CM87', 'CM88', 'CM89', 'CM90', 'CM91', 'CM92', 'CM93', 'CM94', 'CM95', 'CM96', 'CM97', 'CM98', 'CM99', 'CM100'.

Natural Language Processing with PyTorch

Yashesh A. Shroff, PhD
REV. April 2021

Workshop Notes & Lab
References

Contact:
yshroff@gmail.com, @yashroff (twitter)

Session Outline

•Session 1:

- Module 1 (30mins, Lecture): Foundations
 - Fundamentals and application of Language Modeling Tools
 - Classical vs DL NLP
 - NLP Pipeline
- Lab (30mins): NLTK from scratch
 - Setting up your environment
 - NLTK (tokenization)
- Module 2 (30mins):
 - Use NLP pipeline to process documents
 - POS, Word embedding
- Lab (30mins)

Session 2:

- Module 3 Lecture (20mins): Key packages & libraries in NLP; dive into spaCy
- Lab (20mins): spaCy
- Lab: PyTorch

Session 3 & 4: Focus on use cases

- Module 4: Using RNN, LSTM with PyTorch
- Using Seq2Seq model for machine translation
- Lab: Seq2Seq model using PyTorch
- Text Classification
 - Lab: LSTM based text classifier
 - Lab: TFIDF and Logistic Regression based classifier

Learning Objective

- Foundations: Fundamentals and application of Language Modeling Tools
- Overview of Natural Language Processing Techniques & Transfer Learning
- Use NLP pipeline to process documents, Word Vectors
- Introduction to key packages and libraries
- Introduction to spaCy and PyTorch

Session Outline

Session 5:

•Learning Objective

- Deep dive into Transformer architecture

•Session Outline

- Module 5: Introduction to Transformers
 - Paper review (Attention is All you Need)
 - Transfer Learning Fundamentals
 - Pre-trained models, such as BERT, XLNet from Huggingface
- Lab(s): Solve NLP problems using PyTorch, pre-trained models

Session 6:

Learning Objective

- Question / Answering through developing a chatbot

•Session Outline

- Theory
- Stanford **Question Answering** Dataset (SQuAD)
- Lab: Develop a chatbot

<Capstone Project Assignment>

Session 7:

Learning Objective

- MLOps using a text classification model

•Session Outline

- Scheduler Overview
- Implementation walk-through

Session 8:

Capstone Project Presentations

- End to end including MLOps

A word about the training (setting expectations for the next 4 weeks)

What we cover:

- Deep Learning based Neural Machine Translation approach with some theoretical background and heavy labs usage
- Covers modern (last 2-4 years) development in NLP
- Gives a practitioner's perspective on how to build your NLP pipeline

What we do not cover much beyond foundational context:

- Statistical and probabilistic approach (minimal)
- Early Neural Machine Translation approaches (marginal)

“You shall know a word by the company it keeps”

J.R. Firth, 1957

Context is important if you want to understand the meaning of a word

Yashesh A. Shroff

Bit about me:

- Working at Intel as a Strategic Planner, responsible for driving ecosystem growth for AI, media, and graphics on discrete GPU platforms for the Data Center
- Prior roles in IOT, Mobile Client, and Intel manufacturing
- Academic background:
 - ~15 published papers, 5 patents
 - PhD from UC Berkeley (EECS)
 - MBA from Columbia Graduate School of Business (Corp Strategy)
 - Intensely passionate about programming & product development
- Contact:
 - Twitter: @yashroff, yashroff@gmail.com, <https://linkedin/yashroff>



Setting up your Environment

Most of the lab work will be in the Python Jupyter notebooks in the workshop Github repo:

- Jupyter (<https://jupyter.org/install>)
- PyTorch (<https://pytorch.org/get-started/locally/#start-locally>)
- spaCy (<https://spacy.io/usage>)
- Hugging face transformer (<https://huggingface.co/transformers/installation.html>)

Training GitHub Repo

Install git on your laptop:

- <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

And run the following command:

- `git clone https://github.com/yasheshshroff/NLPworkshop.git`

Use conda or pipenv to install the requirements dependencies in a virtual environment.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
conda create -n pynlp python=3.6
source activate pynlp
conda install ipython
conda install -c conda-forge jupyterlab
conda install pytorch torchvision -c pytorch
pip install transformers
```

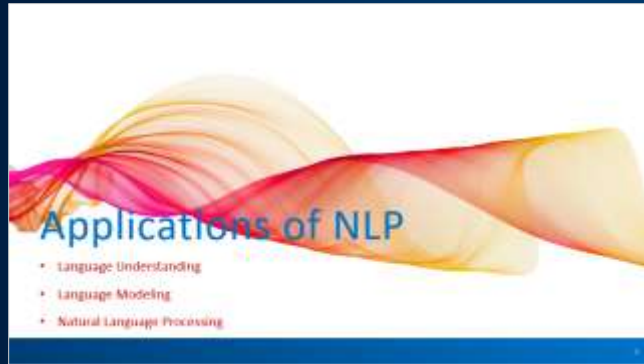
```
# Install spacy and download pretrained language
model
$ pip install -U spacy
$ pip install -U spacy-lookups-data # lang Lemmatization*
$ python -m spacy download en_core_web_sm
```

In Python:

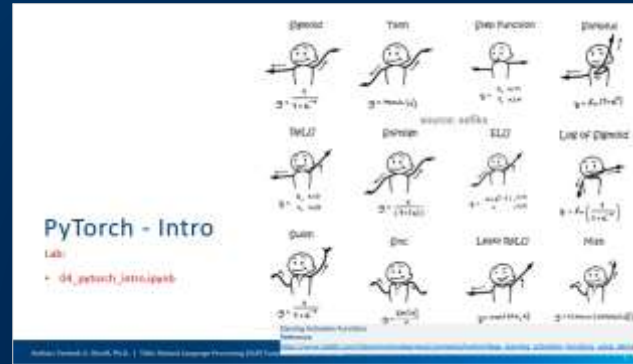
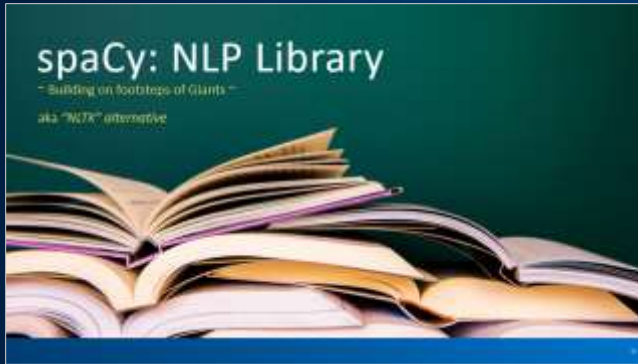
```
import spacy
nlp = spacy.load("en_core_web_sm")
```

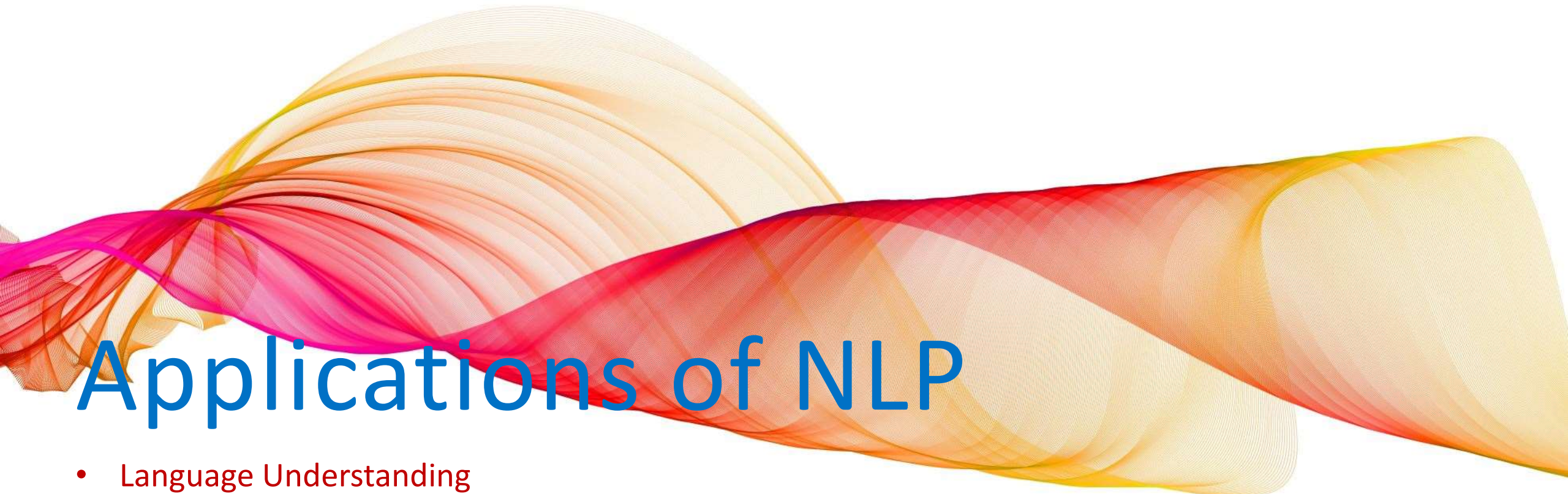
* Where Pretrained Language Model doesn't exist in spaCy (more compact distro)

Part 1: Foundations of NLP



Part 2: Practicum





Applications of NLP

- Language Understanding
- Language Modeling
- Natural Language Processing

Common Applications of Natural Language Processing

Machine Translation

Translating from one language to another

Speech Recognition

Question Answering

Understanding what the user wants

Text Summarization

Concise version of long text

Chatbots

Text2Speech, Speech2Text

Translation of text into spoken words and vice-versa

Voicebots

Text and auto-generation

Sentiment analysis

Information extraction

Common Applications of Natural Language Processing

Machine Translation: Google Translate

Speech Recognition: Siri, Alexa, Cortana

Question Answering: Google Assistant

Text Summarization: Legal, Healthcare

Chatbots: Helpdesk

Text2Speech, Speech2Text

Voicebots: Voic Sales & Marketing

Text and auto-generation: Gmail

Sentiment analysis: Social media (finance, reviews)

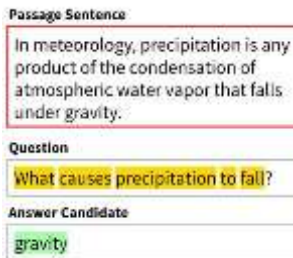
Information extraction: Unstructured (news, finance)

The diagram illustrates the Seq2Seq model architecture. It consists of two main parts: the Encoder and the Decoder.

Encoder: The input sequence $x_1, x_2, x_3, x_4, x_5, x_6, x_7$ is processed by the encoder to produce a sequence of hidden states $h_1, h_2, h_3, h_4, h_5, h_6, h_7$. The final hidden state h_7 is passed to the decoder.

Decoder: The decoder generates the output sequence y_1, y_2, y_3, y_4 based on the previous hidden state and the previous output token. The hidden states are labeled as d_1, d_2, d_3, d_4 .

<https://github.com/google/seq2seq>



Text Classification Algorithms: A survey



Machine Translation

- Benchmarks:
 - <https://paperswithcode.com/task/machine-translation>
- Legal document translation
- Unsupervised Machine Translation
- Low-Resource Neural Machine Translation
- Transliteration

Question Answering

Text Classification

- Benchmarks:
 - <https://paperswithcode.com/task/text-classification>
- Topic models
- Document classification
- Sentence classification
- Emotion Classification

Sentiment Analysis

- Benchmarks:
 - <https://paperswithcode.com/task/question-answering>
- Twitter sentiment analysis
- Aspect-Based sentiment analysis
- Multimodal sentiment analysis

& More...

Text Generation

NER

Natural Language Inference

Information Retrieval

Dependency Parsing

Dialog

Semantic
Textual
Similarity

Reading comprehension

741 benchmarks • 306 tasks •
100 datasets • 8368 papers with
code

The background of the slide is a blurred image of a financial market display. It features a line chart with multiple data series in white and blue, showing fluctuations over time. Surrounding the chart are various stock market data points, including indices like 'OMX18', 'OMX10', and 'OMX12', along with numerical values and 'Buy'/'Sell' indicators. The overall color scheme is dominated by blue and white, with some red and green accents from the market data.

Approaches to Natural Language Processing

From Heuristics to Deep Learning

A brief history of Machine Translation

Pre-2012: Statistical Machine Translation

- Language modeling, Probabilistic approach
- Con: Requires “high-resource” languages

Neural Machine Translation

- word2vec
- GloVe
- ELMo
- Transformer

Underlying common approaches

- Model, Training data, Training process

NMT: Key Papers

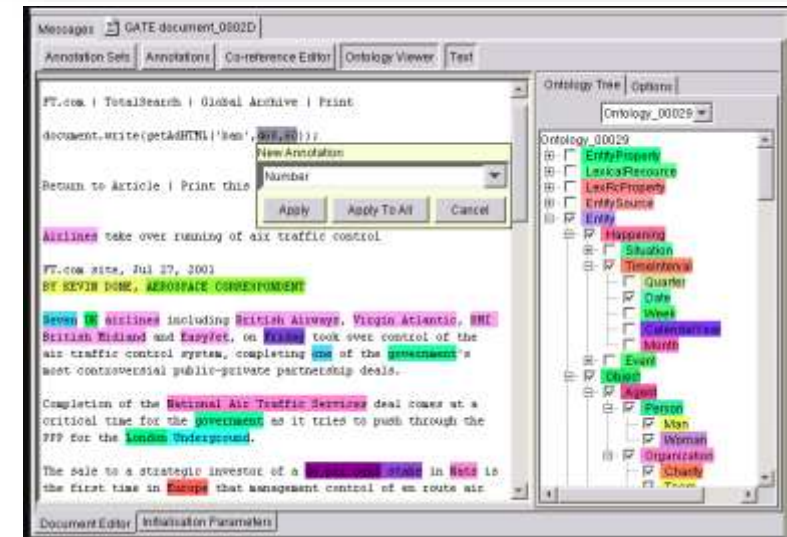
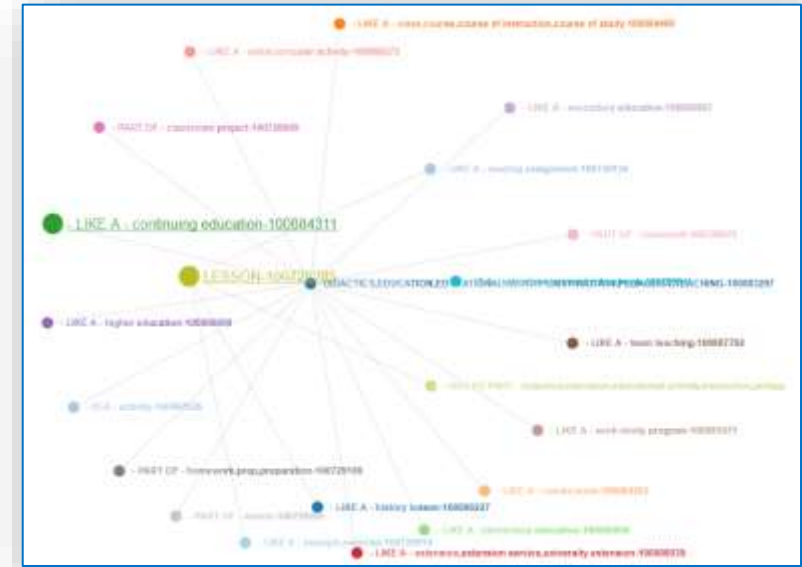
- word2vec: [Mikolov et. al. \(Google\)](#)
- GloVe: [Pennington et al., Stanford CS. EMNLP 2014](#)
- ELMo:
- ELMo (Embeddings from Language Models)
 - Memory augmented deep learning
- Survey paper (<https://arxiv.org/abs/1708.02709>)
 - Blog (<https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d>)
- [Vaswani et al., Google Brain. December 2017.](#)
 - [The Illustrated Transformer blog post](#)
 - [The Annotated Transformer blog post](#)

Ref: <https://eigenfoo.xyz/transformers-in-nlp/>

Heuristics based approach to NLP

Rules based AI systems requiring domain expertise. Applied as:

- Dictionary & thesaurus-based sentiment analysis with counts)
- Knowledge-based relationship between words and concepts
 - Wordnet – mapping of terms for similarity
- Regex: `^([a-zA-Z0-9_-\.\.]+)@([a-zA-Z0-9_-\.\.]+\.[a-zA-Z]{2,5})$`
 - Key sub-strings, such as product ID
- Context-Free Grammar (formal): GATE / JAPE



Reference: <https://www.visual-the-saurus.com/wordnet.php?link=100883297>

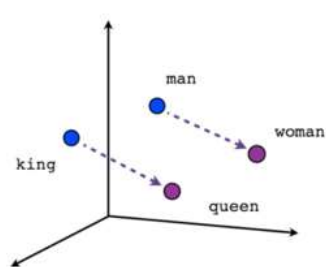
Classical vs. DL NLP

Classical:

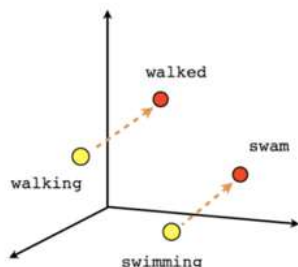
- Task customization for NLP Applications

DL Based NLP

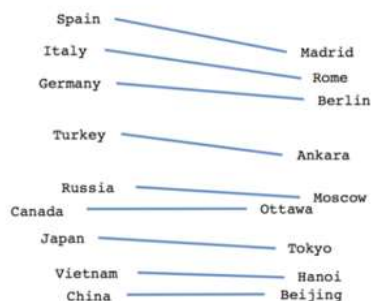
- Compressed representation
- Word Embeddings



Male-Female

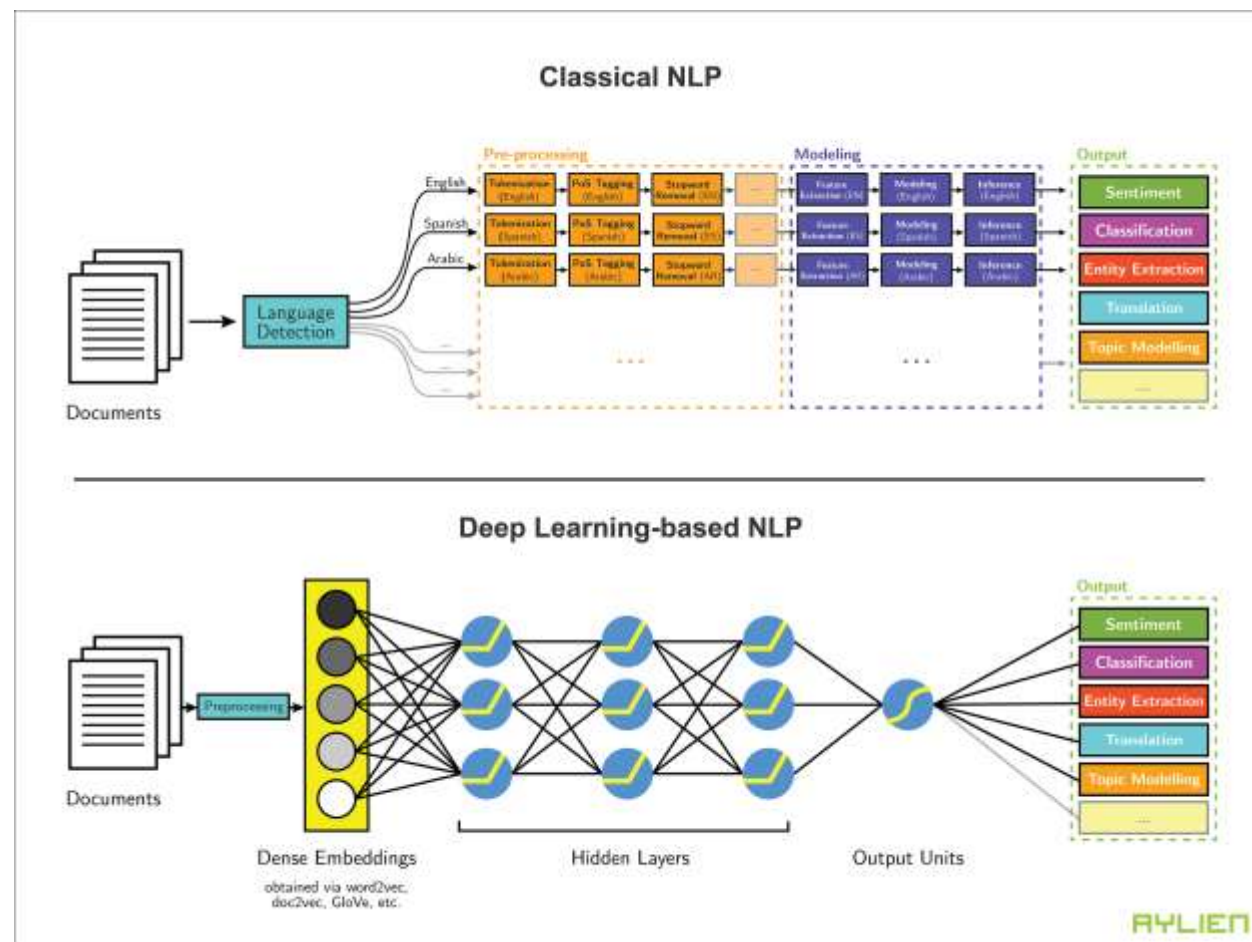


Verb tense



Country-Capital

Reference: <https://arxiv.org/abs/1301.3781>
(Efficient Estimation of Word Representations in Vector Space)



Reference: <https://aylien.com/blog/leveraging-deep-learning-for-multilingual>

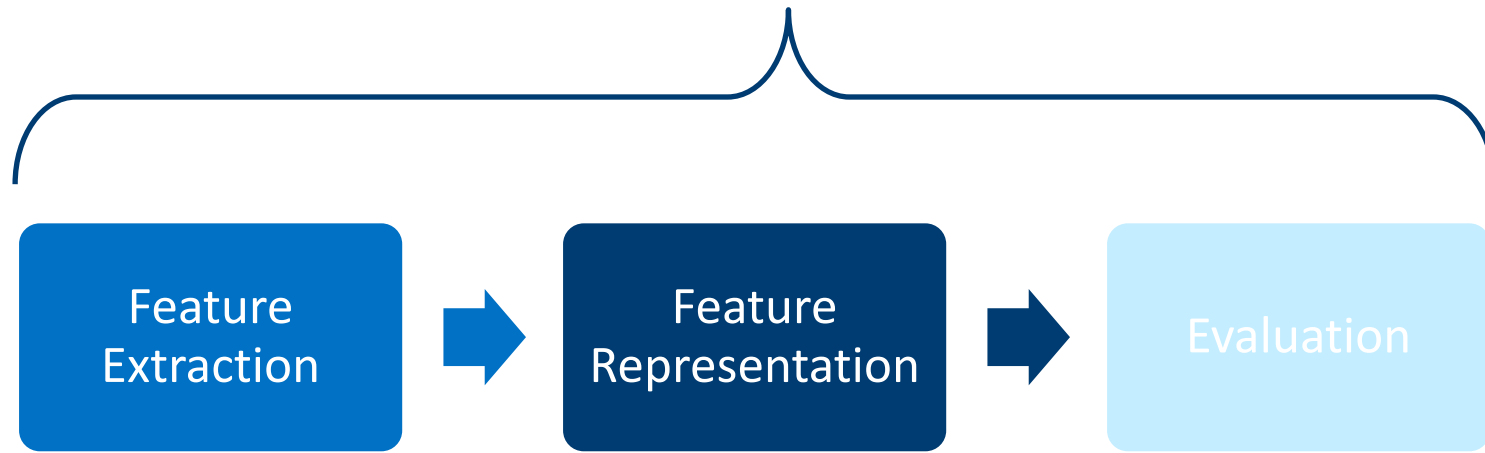
Machine Learning based NLP

Supervised

- Text classification
- Regression

Unsupervised

- Document topic modeling



Popular Machine Learning Algos for NLP

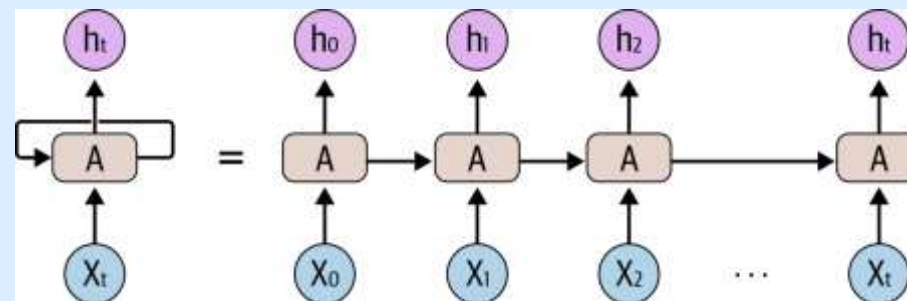
Algorithm	Description
Naïve Bayes	Assumes feature independence (naïve) Ex. Frequency of specific words for classification
Support Vector Machines	Leans optimal (linear or non-linear) decision boundaries between classes (sports vs political articles)
Hidden Markov Models	Models unobserved hidden states that generate observed data, for example, for parts-of-speech tagging*
Conditional Random Fields	Sequential, context-based information management, works better than HMM in a closed domain [1 , 2]

* POS is covered next as a topic

Deep Learning in NLP

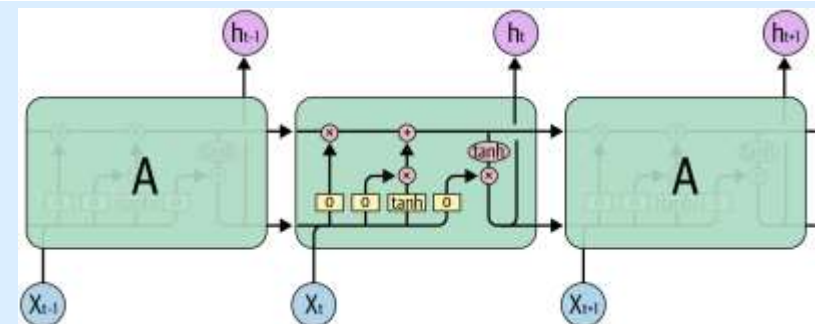
Recurrent Neural Networks

- Progressively reads input and generates output
- Capability to 'remember' short texts



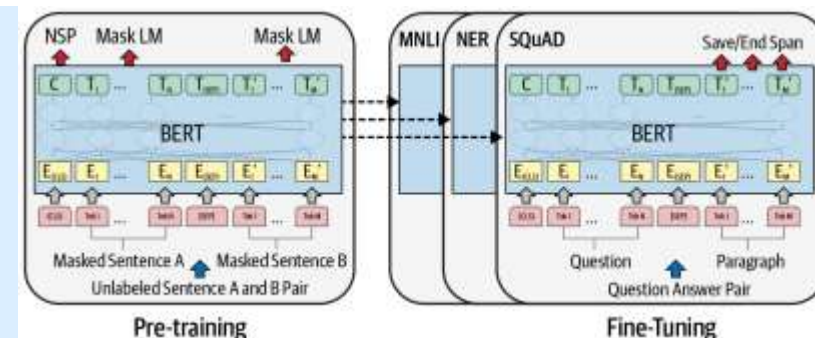
Long-Short Term Memory

- Improves upon RNN with longer text memory
- Ability to let go of certain context



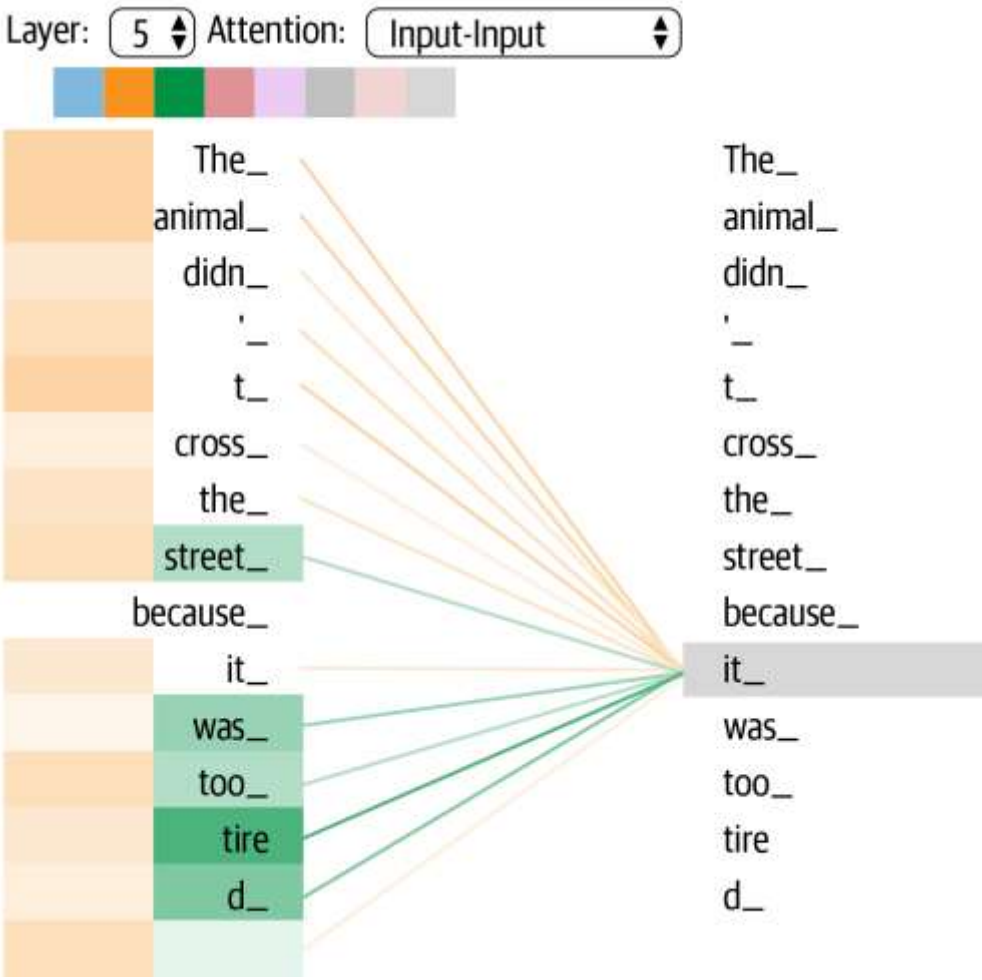
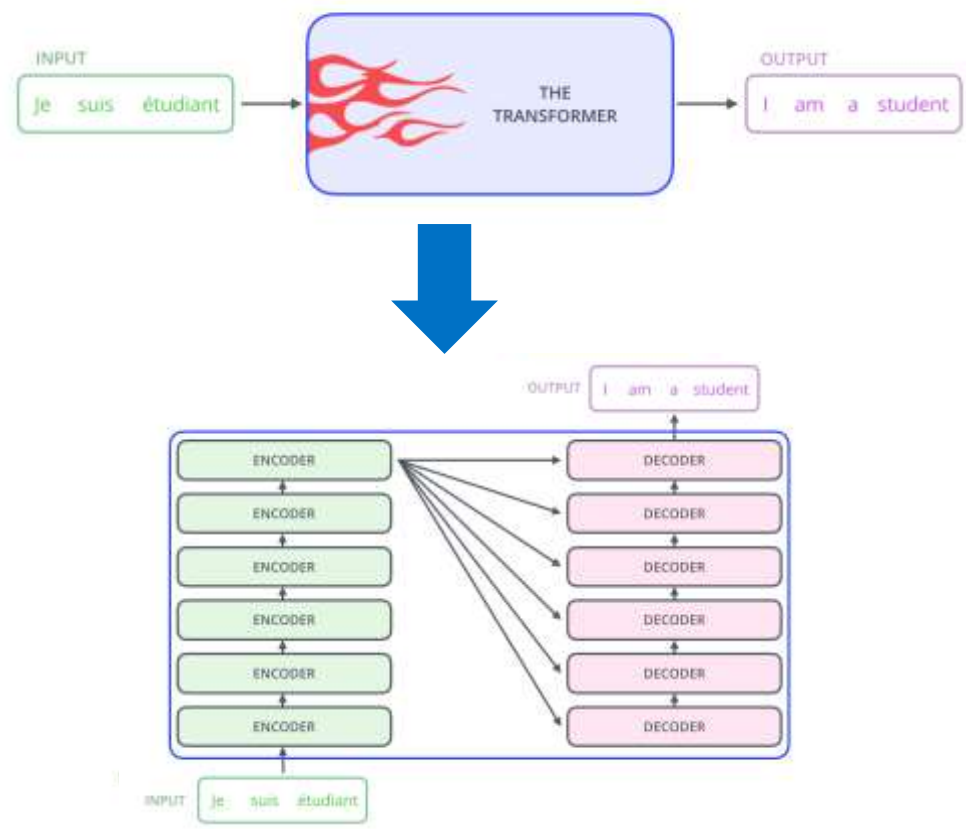
Transformers

- Language modeling with context 'around' a word
- Transfer learning applies to downstream tasks



Transformer (motivation)

Self-Attention Mechanism

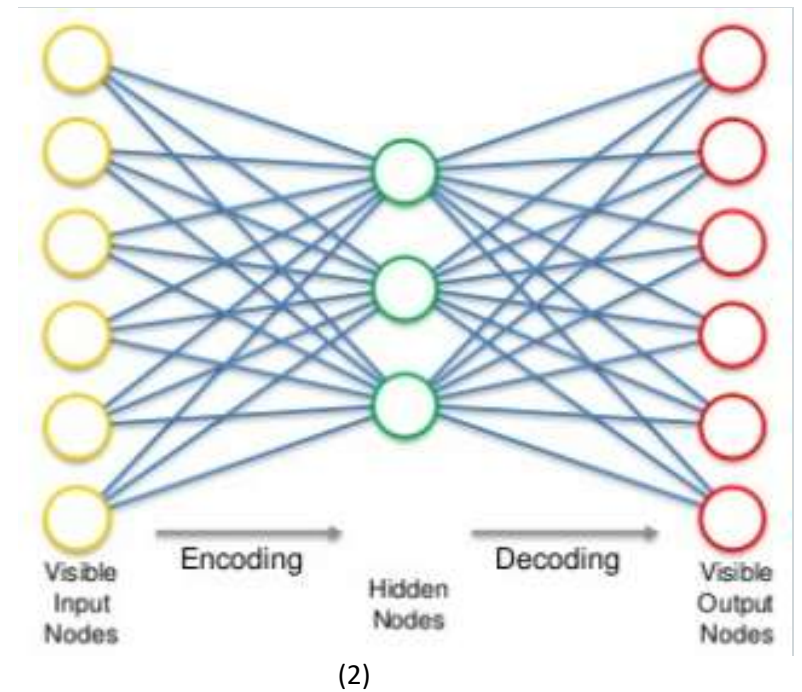
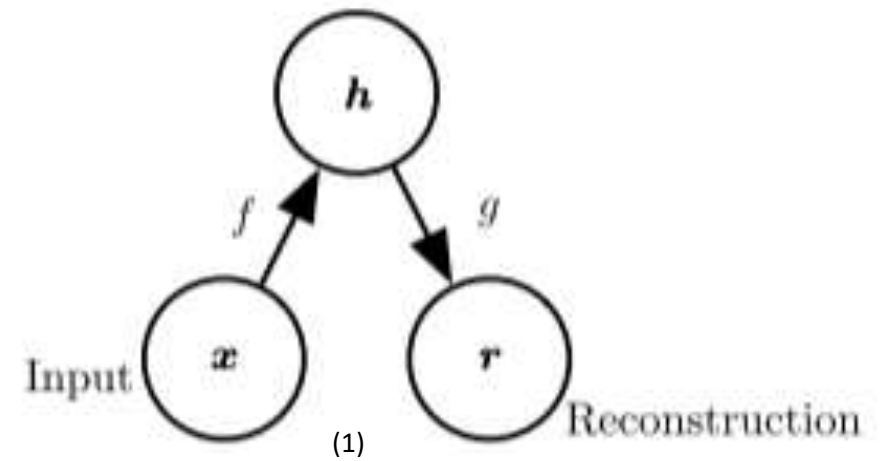


Jay Alammar: [The Illustrated Transformer](#)

Autoencoder

Learning Compressed Vector Representation

- Unsupervised learning
- Mapping a function of input to the output
- Reconstruct back to the output
- Example: Vector representation of text
 - Post training: collect the vector representation as a dense vector of the input text



Ref:

1) Ian Goodfellow, [“The Deep Learning Book”](#)

2) Kirill Ermenko, [Auto Encoder](#)



Pre-Processing NLP tasks

NLP Preprocessing Tasks

Tokenization

- Splitting text into meaningful units (words, symbols)

POS tagging

- Words->Tokens (verbs, nouns, prepositions)

Dependency Parsing

- Labeling relationship between tokens

Chunking

- Combine related tokens ("San Francisco")

Lemmatization

- Convert to base form of words (slept -> sleep)

Stemming

- Reduce word to its stem (dance -> danc)

Named Entity Recognition

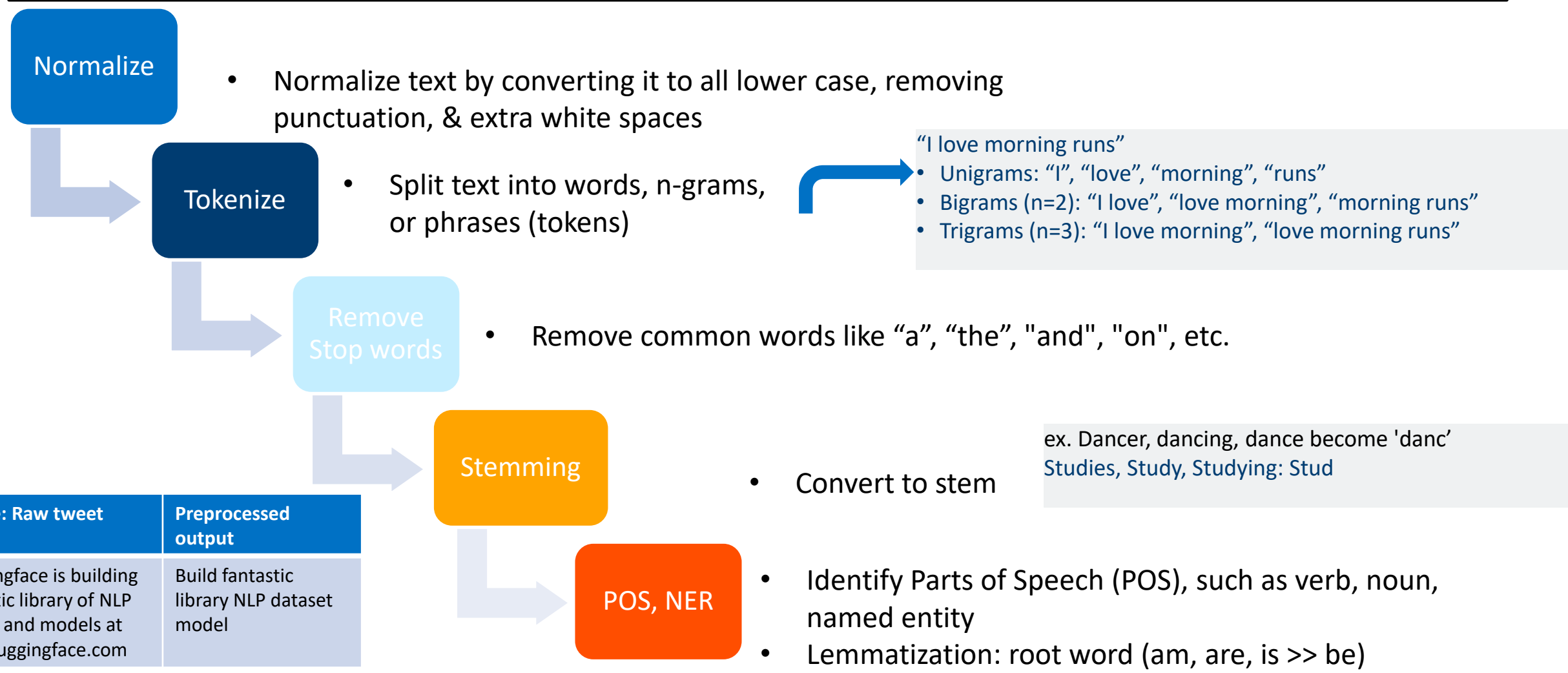
- Assigning labels to known objects: Person, Org, Date

Entity Linking

- Disambiguating entities across texts

NLP Tasks: Working through examples

Start with clean text, without immaterial items, such as HTML tags from web scraped corpus.



Top NLP Packages

NLTK

- Preprocessing: Tokenizing, POS-tagging, Lemmatizing, Stemming
- Cons: Slow, not optimized

Gensim

- Specialized, optimized library for topic-modeling and document similarity

spaCy

- "Industry-ready" NLP modules.
- Optimized algorithms for tokenization, POS tagging
- Text parsing, similarity calculation with word vectors

Huggingface – Transformers / Datasets (Day 2)

Starting from scratch

Normalization: convert every letter to a common case so each word is represented by a unique token

```
text = text.lower()
text = re.sub(r"[^a-zA-Z0-9]", " ", text)
```

Token: Implies symbol, splitting each sentence into words

```
text = text.split()
```

```
from nltk.tokenize import
word_tokenize
words = word_tokenize(text)
```

NLTK: Split text into sentences

```
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
```

Stop-word removal

Stop-word removal

```
from nltk.corpus import stopwords
print(stopwords.words("english"))
words = [w for w in words if not in stopwords.words("english")]
```

Parts of speech tagging

```
from nltk import pos_tag
sentence = word_tokenize("Start practicing with small code.")
pos_text = pos_tag(sentence)
```

Name Entity Recognition (NER) to label names (used for indexing and searching for news articles)

```
from nltk import ne_chunk
ne_chunk(pos_text)
```

Normalizing word variations

1. Stemming: reducing words to their stem or root

```
from nltk.stem.porter import PorterStemmer
stemmed = [PorterStemmer().stem(w) for w in words]
print(stopwords.words("english"))
words = [w for w in words if not in stopwords.words("english")]
```

2. Lemmization

```
from nltk.stem.wordnet import WordNetLemmatizer
lemmed = [WordNetLemmatizer().lemmatize(w) for w in words]
lemmed = [WordNetLemmatizer().lemmatize(w, pos='v') for w in lemmmed]
```

Name Entity Recognition (NER) to label names (used for indexing and searching for news articles)

```
from nltk import ne_chunk
ne_chunk(pos_text)
```

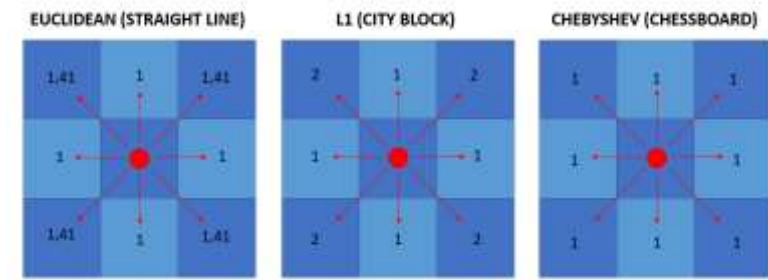

Lab

Google Colab:

1. 01_NLP_basics.ipynb

Distance Similarity

Measuring distances: Euclidean, L1, & L-Infinity



- Euclidean Distance:
 - Computing the diagonal between the two points
 - Pythagoras theorem

$$dist(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

- L1 Distance
 - Also known as "Cityblock distance"
 - Measures distance only along straight lines

$$dist(A, B) = |x_A - x_B| + |y_A - y_B|$$

- Chebyshev Distance
 - Also known as L-Infinity or Chessboard distance

$$dist(A, B) = \max(|x_A - x_B|, |y_A - y_B|)$$

Ref: <https://towardsdatascience.com/3-distances-that-every-data-scientist-should-know-59d864e5030a>

Distance between texts

Hamming Distance

- Compares every letter of two strings based on position

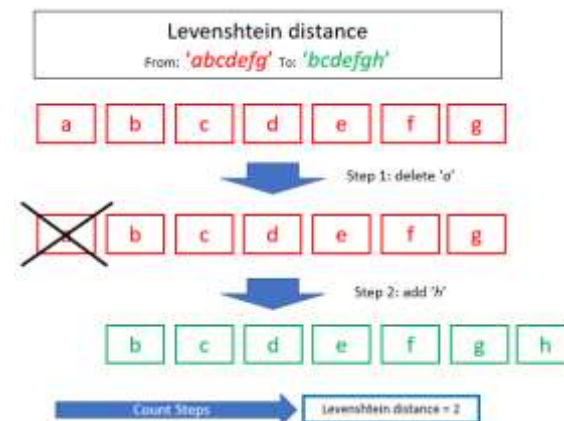
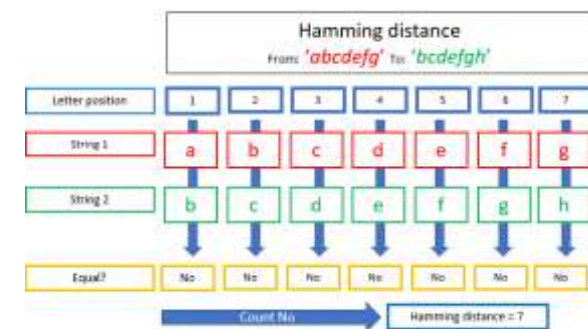
Levenshtein Distance

- Given by the number of ops required to convert one string to another
 - Inserting, Deleting, Substituting characters

Cosine Distance

- Applies to vector representation of documents
 - Uses a word count vectorizer

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$



	1	2	3
i	1	1	1
love	1	1	0
going	1	1	1
to	1	1	1
the	1	0	0
movies	1	0	0
work	0	1	1
why	0	0	1
is	0	0	1
it	0	0	1
always	0	0	1
raining	0	0	1
when	0	0	1
am	0	0	1

```
In [3]: 1 from sklearn.metrics import pairwise

In [4]: 1 vector_1 = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
        2 vector_2 = [1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
        3 vector_3 = [1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]

In [5]: 1 matrix = [vector_1, vector_2, vector_3]

In [6]: 1 pairwise.cosine_similarity(matrix)

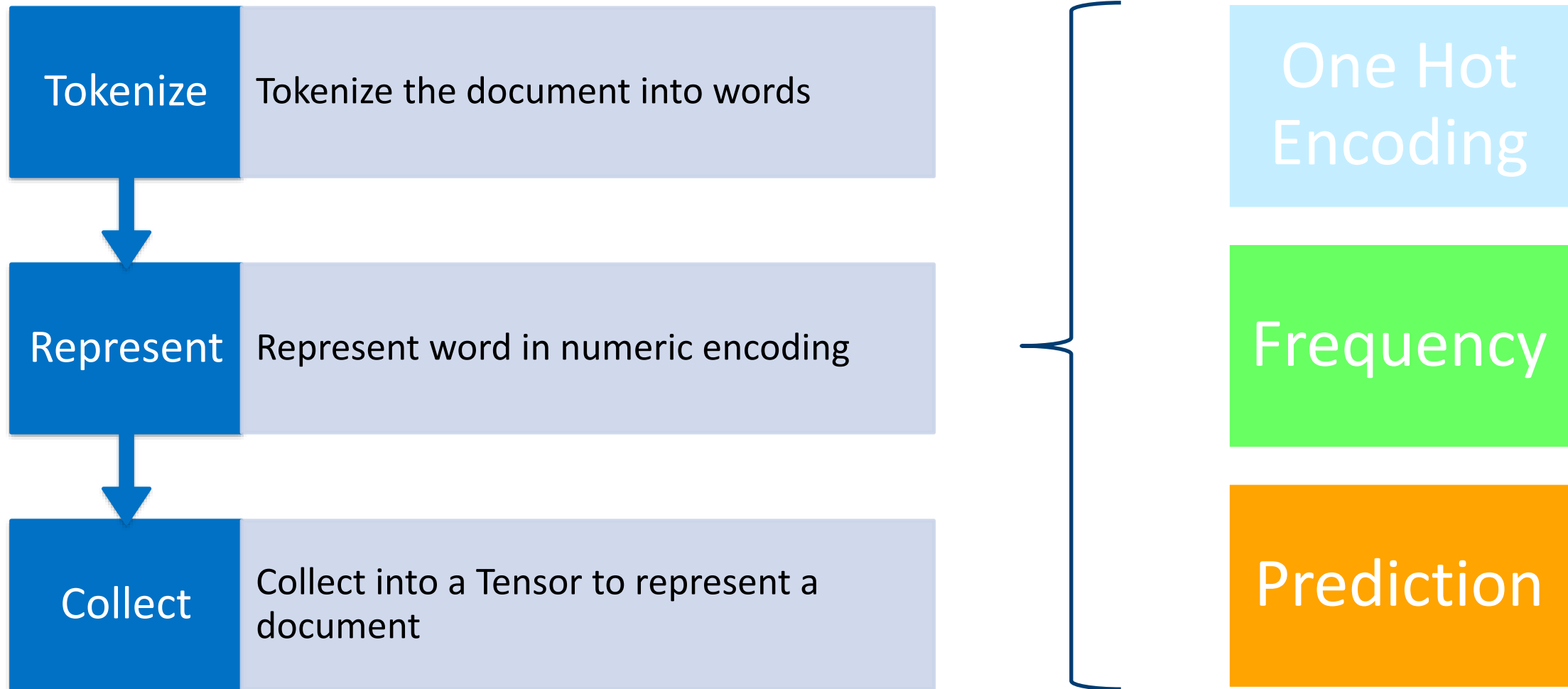
Out[6]: array([[1.          , 0.81649658, 0.36927447],
               [0.81649658, 1.          , 0.45226702],
               [0.36927447, 0.45226702, 1.          ]])
```




Sentiment Analysis

Text Classification

Text Classification with Neural Networks



A row of matches is shown against a dark red background. The match on the far left is lit, with a bright yellow and orange flame rising from its tip. The other matches are unlit, showing their red heads and wooden stems. The flame is the central focus, symbolizing the 'hot' in 'One Hot Representation'.

One Hot Representation

Simple Vector Representation of Words

One Hot Representation: Vector Representation of Words

Fundamental Idea

- Assume we have a toy 100-word vocabulary
- Associate to each word an index value between 1 to 100
- Each word is represented as a 100-dimension array-like representation
- All dimensions are zero, except for one corresponding to the word

Vocabulary

seat: 1
gear: 2
car: 3
seats: 4
auto: 5
engine: 6
belt: 7
...
chassis: 100

	1	2	3	4	5	...	100
gear							
seat							
seats							
...							
chassis							
auto							

Challenges with this approach:

- Curse of dimensionality: Memory capacity issues
 - The size of the matrix is proportionate to vocab size (there are roughly 1 million words in the English language)
- Lack of **meaning** representation or word **similarity**
 - Hard to extract meaning. All words are equally apart
 - “seat” and “seats” vs “car” and “auto” (former resolved with stemming and lemmatization)

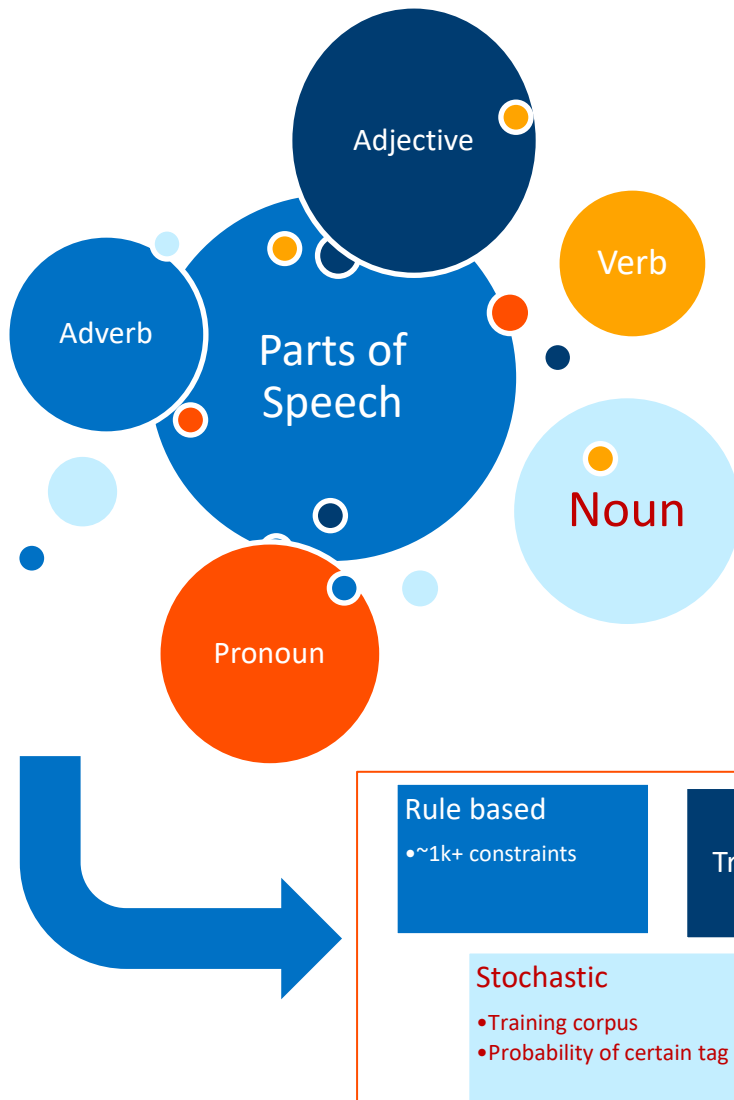
Lab

Google Colab:

- `02_inefficient.ipynb`

POS, Word Embedding

Parts of Speech Tagging



One tag for each part of speech

- Choose a courser tagset (~6 is useful)
- Finely grained tagsets exist (ex. Upenn Tree Bank II)

Sentence: "Flies like a flower"

- **flies**: Noun or Verb?
- **like**: preposition, adverb, conjunction, noun or verb?
- **a**: article, noun, or preposition
- **flower**: noun or verb?

<https://parts-of-speech.info/>

"The blue house at the end of the street is mine."

The blue house at the end of the street is mine

Adjective	Number
Adverb	Preposition
Conjunction	Pronoun
Determiner	Verb
Noun	

Word Embeddings

Techniques to convert text data to vectors

Frequency based

- Count Vector
- TF-IDF
- Co-occurrence Vector

- Count based feature engineering strategies (bag of words models)
- Effective for extracting features
- Not structured
 - Misses semantics, structure, sequence & nearby word context
- 3 main methods covered in this lecture. There are more...

Prediction based Word2Vec

- CBOW
- Skip-Gram

- Capture meaning of the word
- Semantic relationship with other adjacent words
 - Deep Learning based model computes distributed & dense vector representation of words
- Lower dimensionality than bag of words model approach
- **Alternative:** GloVe



Word Embedding

Frequency based

Document 1: "This is about cars"
Document 2: "This is about kids"

TF-IDF vectorization

Term	Count		TF-IDF
	Doc1	Doc2	Doc 1 example
This	2	1	$2/8 * \log(2/2) = 0$
is	3	2	$3/8 * \log(2/2) = 0$
about	1	2	$1/8 * \log(2/2) = 0$
Kids	0	4	
cars	2	0	$2/8 * \log(2/1) = 0.075$
Terms	8	9	

Co-Occurrence Vector

"He is not lazy. He is intelligent. He is smart"

	He	is	not	lazy	intelligent	smart
He	0	1	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	3	0
lazy	1	2	1	0	4	0
intelligent	2	2	0	0	3	0
smart	1	1	0	0	3	0

He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart

Count Vector

Doc 1	"The athletes were playing"
Doc 2	"Ronaldo was playing well"

	The	Athlete	was	playing	Ronaldo	well
Doc 1	1	1	1	1	0	0
Doc 2	0	0	1	1	1	1

- Real-world corpus can be millions of documents & 100s M unique words resulting in a very sparse matrix.
- Pick top 10k words as an alternative.

$$TF = \frac{\text{\# times term } T \text{ appears in the document}}{\text{\# of terms in the document, } m}$$

$$IDF = \left(\frac{\text{Number of documents, } N}{\text{Number of documents in which term } T \text{ appears, } n} \right) = \log \left(\frac{N}{n} \right)$$

} Calculate
 $TF \times IDF$

- Term frequency across corpus accounted, but penalizes common words
- Words appearing only in a subset of document are weighed favorably

$$\begin{pmatrix} \hat{X} \\ x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}_{m \times n} \approx \underbrace{\begin{pmatrix} U \\ u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mr} \end{pmatrix}}_{m \times r} \underbrace{\begin{pmatrix} S \\ s & 0 & \cdots \\ 0 & \ddots & \\ & & s_{rr} \end{pmatrix}}_{r \times r} \underbrace{\begin{pmatrix} V^T \\ v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{r1} & \cdots & v_{rn} \end{pmatrix}}_{r \times n}$$

Word-vector representation Context

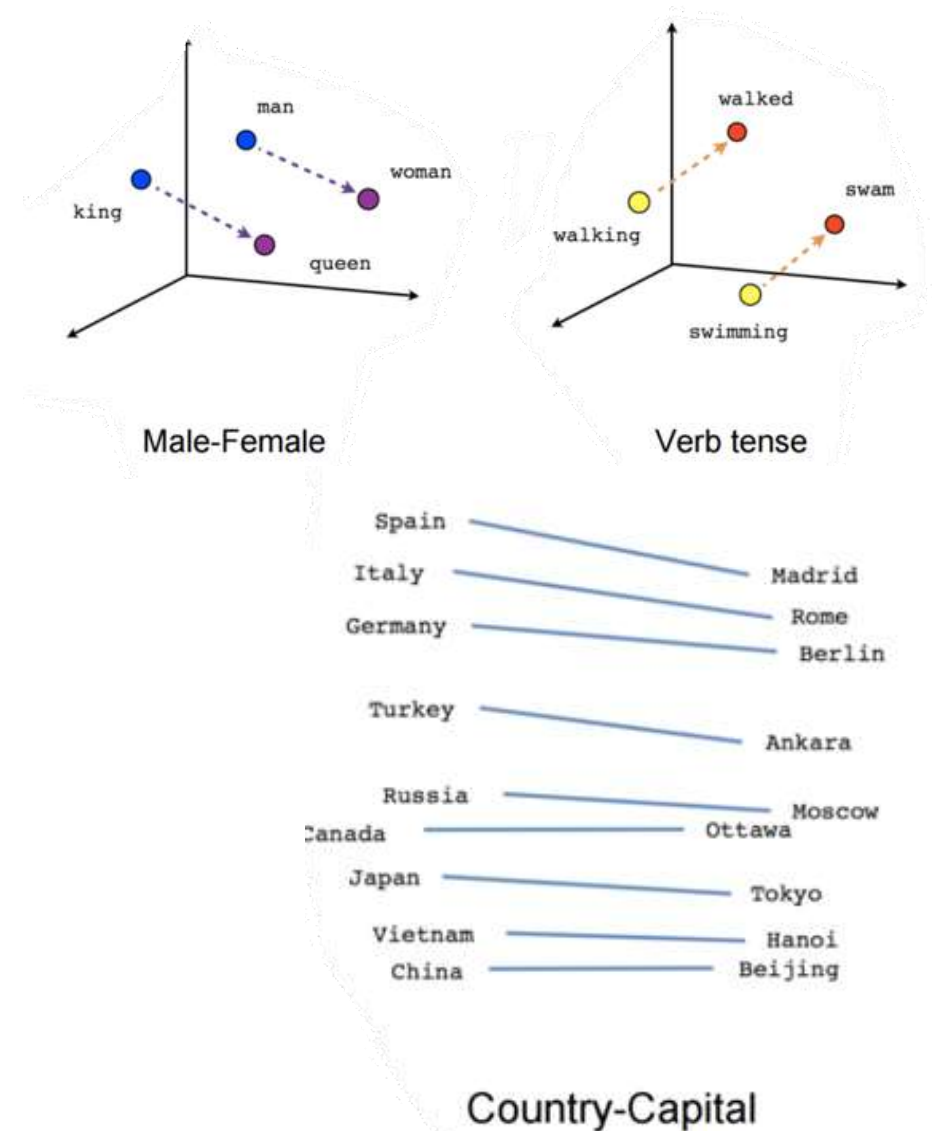
m : # of terms
 n : m minus stop words
• Uses SVD decomposition and PCA to reduce dimensionality

- Similar words tend to occur together: "Airbus is a plane", "Boeing is a plane"
- Calculates the # of times words appear together in a context window

Prediction based Word Embedding

Key Idea: Words share context

- Embedding of a word in the corpus (numeric representation) is a function of its related words – words that share the same context
- Examples: “word” => (embeddings)
 - “car” => (“road”, “traffic”, “accident”)
 - “language” => (“words”, “vocabulary”, “meaning”)
 - “San Francisco” => (“New York”, “London”, “Paris”)



Reference: <https://arxiv.org/abs/1301.3781>

(Efficient Estimation of Word Representations in Vector Space)

Learning Outcomes for Session 2

Diving into Word2Vec

- 15min: CBOW & Skip-Gram
- 15min: Word2Vec lab with Gensim

spaCy library

- 30min: What it is, why it's important, key features, and when it's useful
- 30min: Hands-On: spaCy foundations, diving deep, and pipelines

PyTorch

- 10min: Intro - exercises
- 20min: Backpropagation – Autograd



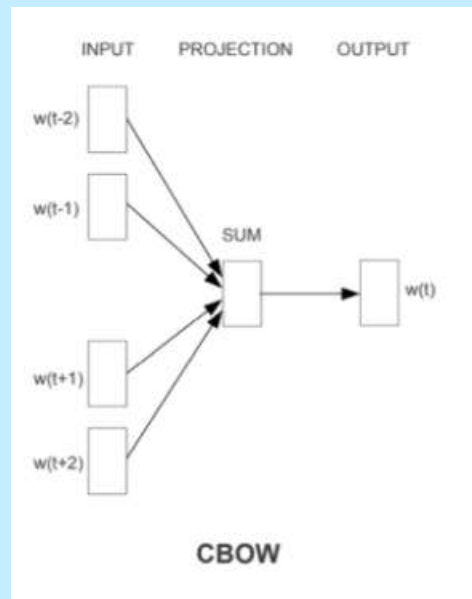
Word Vectors

Moving beyond OHE

Word Embedding

Prediction based Word2Vec

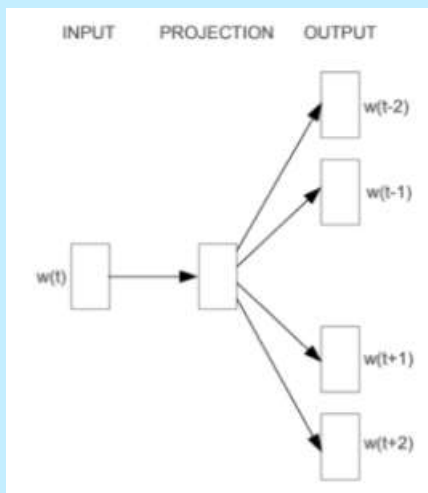
CBOW



- The distributed representation of the surrounding words are combined to predict the word in the middle
- Input word is OHE vector of size V and hidden layer is of size N
- Pairs of context window & target window
- Using context window of 2, let's parse:
 - "The quick brown fox jumps over the lazy dog"
 - "quick __ fox": ([quick, fox], brown)
 - "the __ brown": ([the, brown], quick)
- Tip: Use a framework to implement (ex. Gensim)

<https://arxiv.org/pdf/1301.3781.pdf>

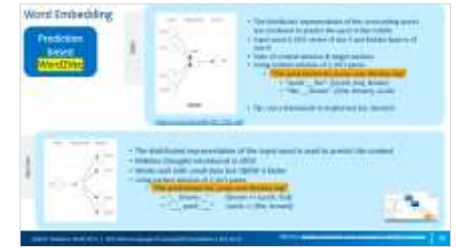
Skip-Gram



- The distributed representation of the input word is used to predict the context
- Mikolov (Google) introduced in 2013
- Works well with small data but CBOW is faster
- Using context window of 2, let's parse:
 - "The quick brown fox jumps over the lazy dog"
 - "__ brown __" (brown => [quick, fox])
 - "__ quick __" (quick => [the, brown])



Representing words by their context

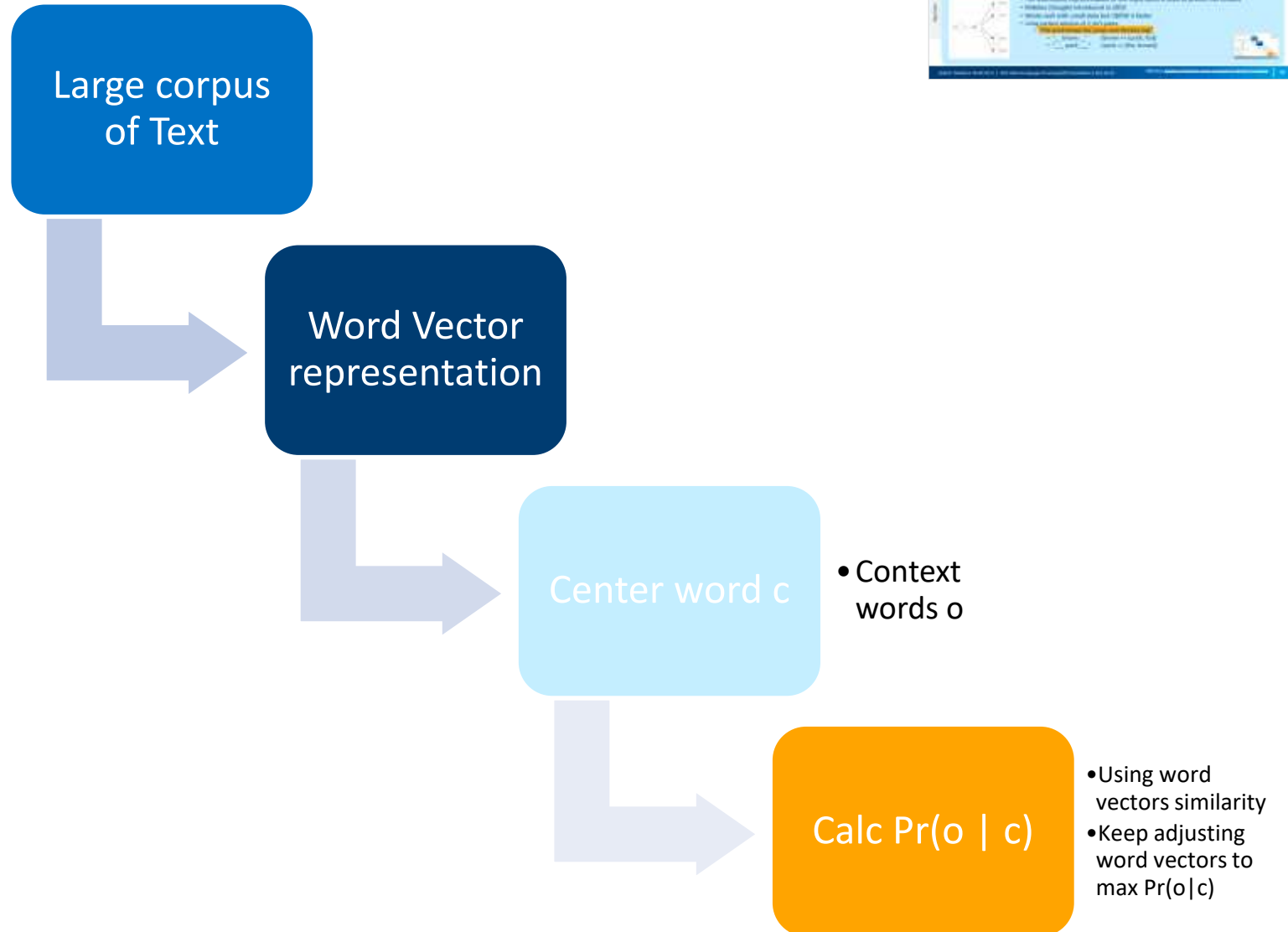


Recap

- We saw the challenges with One Hot Encoding
- We want to build a dense vector for each word

$$- \text{banking} = \begin{pmatrix} 0.182 \\ 0.232 \\ 0.725 \\ 0.375 \\ 0.982 \\ 0.245 \end{pmatrix}$$

- Encoding Similarity in the vectors
- **Distributed** representation (these are all the same):
 - Word Vectors
 - Word Embeddings
 - Word Representation



Skip-Gram Objective Function

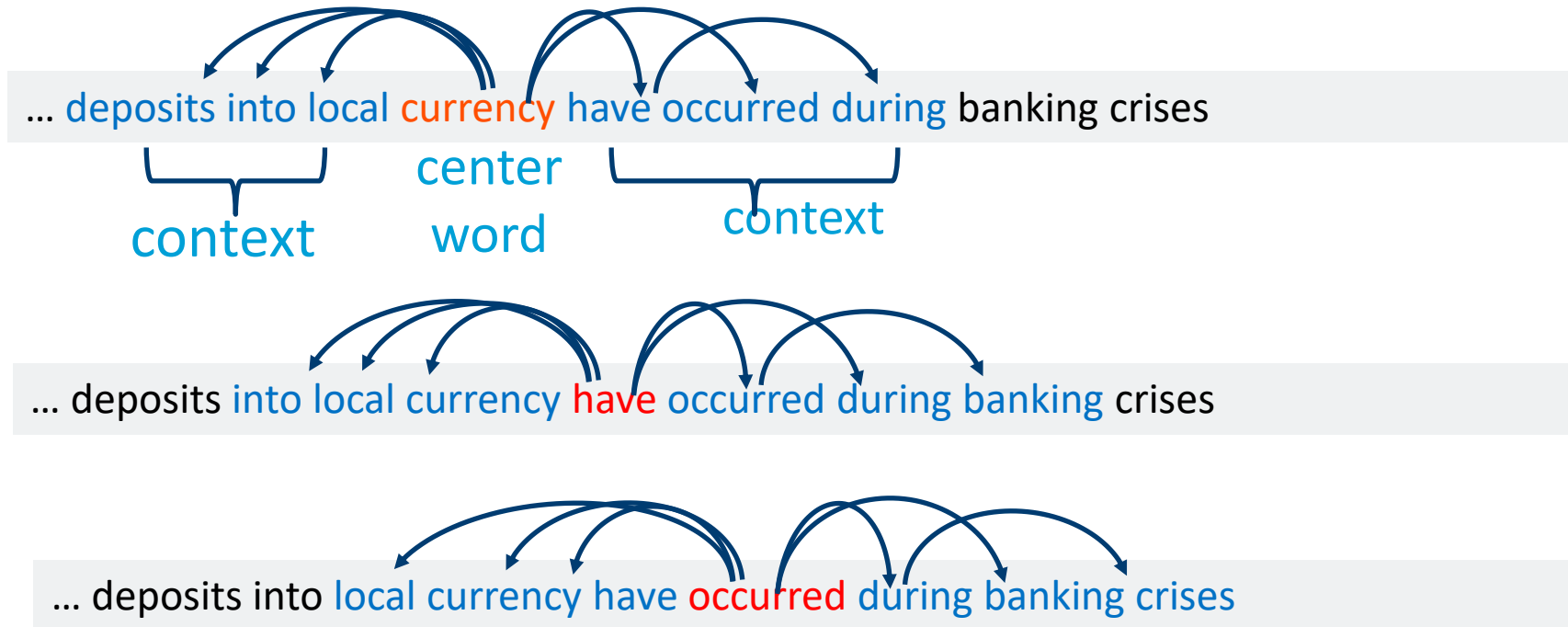
$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log P(\mathbf{w}_{t+j} | \mathbf{w}_j)$$

c is the size of the training context

Processing windows for Word2Vec Computing

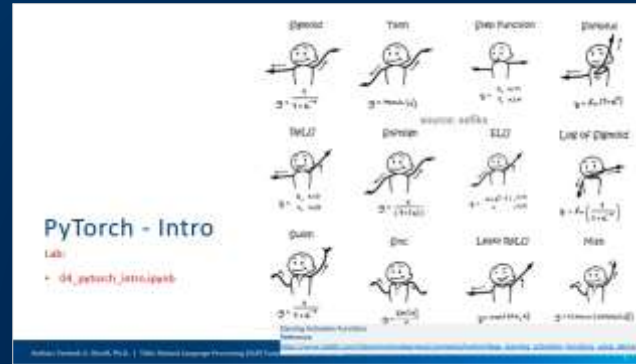
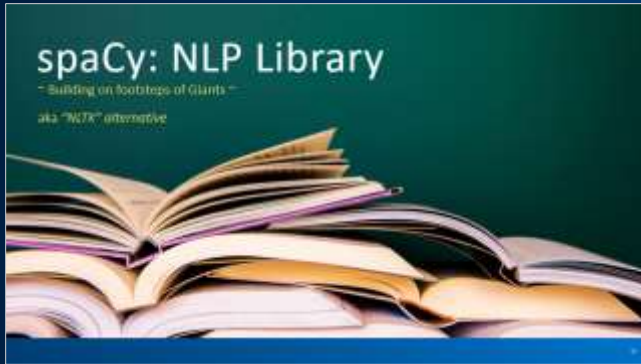
$$-3 \leq j \leq 3$$

$$P_r(w_{t+j}|w_t)$$



- Word2Vec Papers:
 - Efficient Estimation of Word Representations in Vector Space: <https://arxiv.org/abs/1301.3781>

Part 2: Practicum



spaCy: NLP Library

~ Building on footsteps of Giants ~

aka *“NLTK” alternative*



What is spaCy & Why Use it?

spaCy is *fast, accurate, with integrated word vectors*.

- Batteries included: Use the built-in tokenizer. Can add special tokens
- Pipeline approach: Part-of-speech tagging, and parsing requires a model

But what about Huggingface Transformers?

- We will cover Transformers in a later session – both are valuable, depending on your use case. spaCy 3.0 now has Transformer support, while Huggingface has more support for data pre-processing

What about NLTK?

- A very useful library for everything, but it misses the ‘glue’ that spaCy and Huggingface provide. Taking NLTK into production is more of a challenge, but it’s a very good first step to *learn* about the pre-processing steps

- ✓ Support for **70+ languages**
- ✓ **58 trained pipelines** for 18 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

Getting started with spaCy

```
python -m spacy download 'en_core_web_sm'
```

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

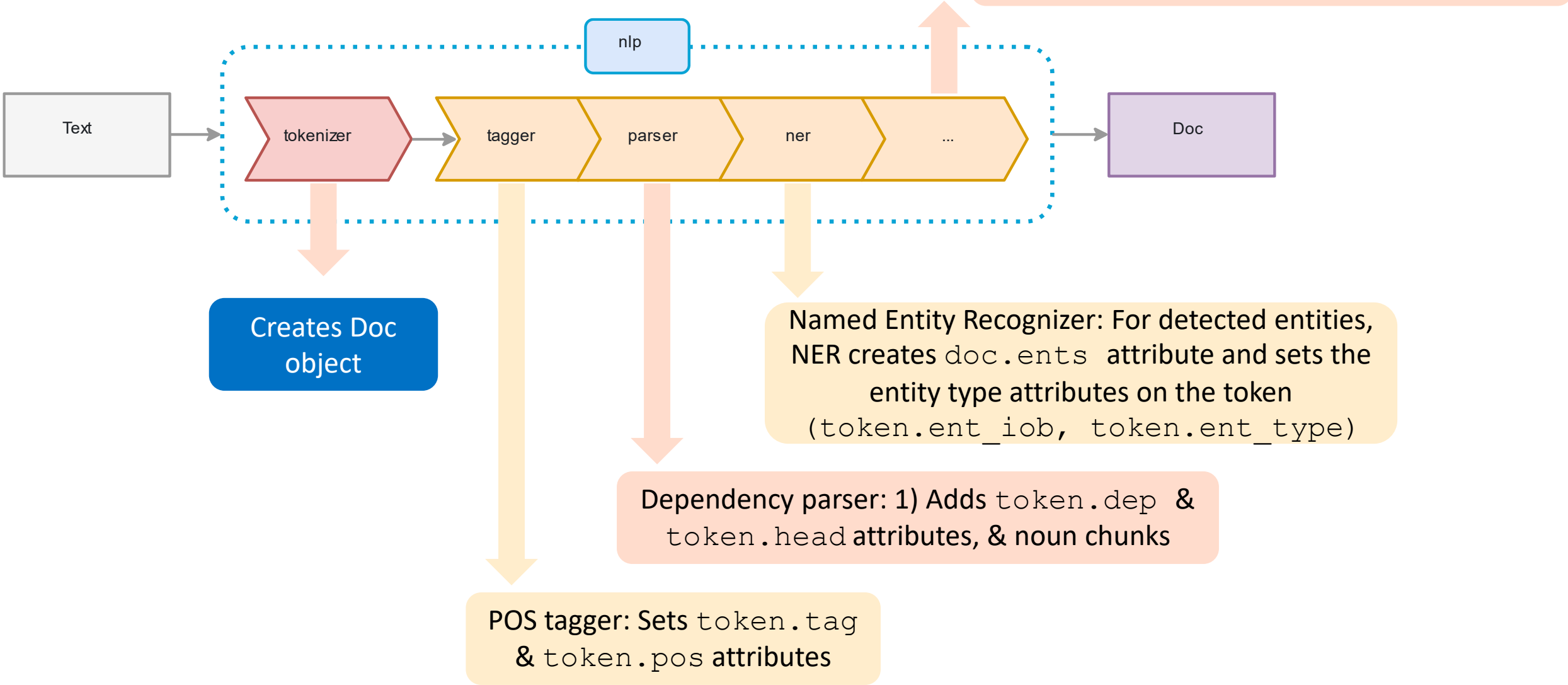
```
# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
       "Google in 2007, few people outside of the company took him "
       "seriously. "I can tell you very senior CEOs of major American "
       "car companies would shake my hand and turn away because 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_)
```

spaCy: <https://spacy.io/>

spaCy Pipelines



*Not part of any pre-trained models

Spacy Models

meta.json

```
{
  "lang": "en",
  "name": "core_web_sm",
  "pipeline": ["tagger", "parser", "ner"]
}
```

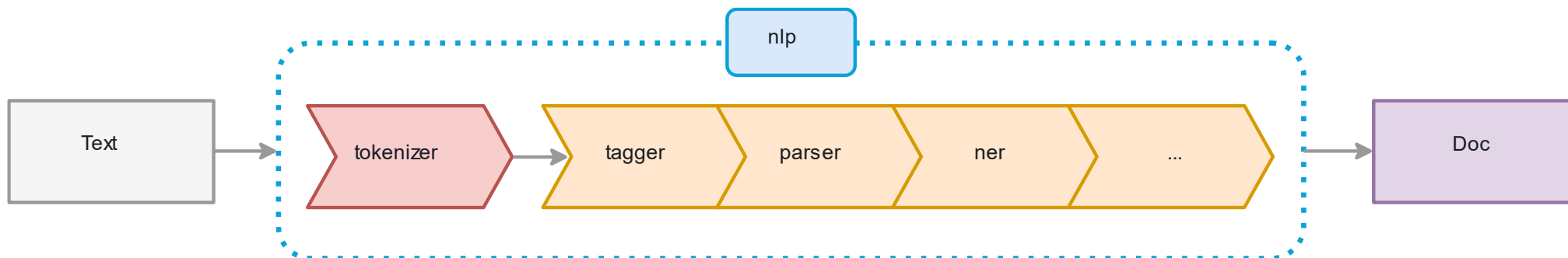
Model	Size	Type
en_core_web_sm	11 MB	Small: Multi-task CNN trained on OntoNotes .
en_core_web_md	48 MB	Medium: Multi-task CNN trained on OntoNotes , with GloVe vectors trained on Common Crawl – 20k unique vectors for 685k keys
en_core_web_lg	746MB	Large: Multi-task CNN trained on OntoNotes , with GloVe vectors trained on Common Crawl – 685k unique vectors & keys

```
nlp.pipe_names
['tok2vec', 'tagger', 'parser', 'ner', 'attribute_ruler', 'lemmatizer']
```

Functions applied to the Doc & set attributes

```
nlp.pipeline
[('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x7f8329aaf810>),
 ('tagger', <spacy.pipeline.tagger.Tagger at 0x7f8329ab0bd0>),
 ('parser', <spacy.pipeline.dep_parser.DependencyParser at 0x7f832b2f5820>),
 ('ner', <spacy.pipeline.ner.EntityRecognizer at 0x7f8327284ac0>),
 ('attribute_ruler',
  <spacy.pipeline.attributeruler.AttributeRuler at 0x7f83294bca80>),
 ('lemmatizer',
  <spacy.lang.en.lemmatizer.EnglishLemmatizer at 0x7f832d2ed740>)]
```

spaCy Custom Components



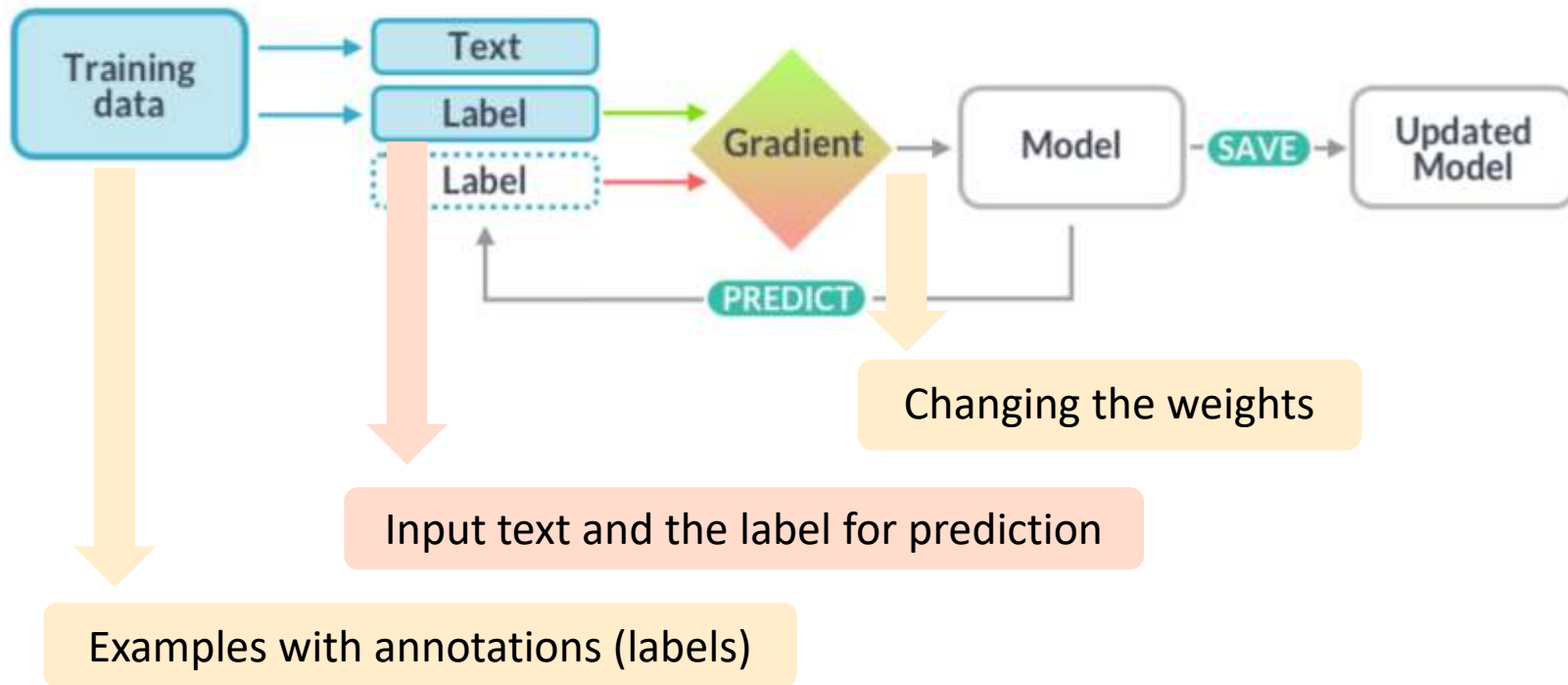
Custom components are executed when `nlp("text")` is called

```
nlp = spacy.load("en_core_web_sm")
def my_component(doc):
    print("Doc length:", len(doc))
    return doc

nlp.add_pipe(my_component, first=True)
print("Pipeline:", nlp.pipe_names)
# Output
# Pipeline: ['my_component', 'tagger', 'parser', 'ner']
```

- `nlp.add_pipe(component, last=True)`
- `nlp.add_pipe(component, first=True)`
- `nlp.add_pipe(component, before="ner")`
- `nlp.add_pipe(component, after="tagger")`

Training



Parts of the pipeline can be disabled during training

Training examples:

```
training_data = [  
    ("iPhone X is coming", {"entities": [(0, 8, "GADGET")]}),  
    ("I need a new phone! Any tips?", {"entities": []})  
]
```


Universal Parts of Speech Tagging

spaCy Documentation:

- The individual mapping is specific to the training corpus and can be defined in the respective language data's `tag_map.py`.

Reference:

- <https://spacy.io/api/annotation>



Universal Part-of-speech Tags ⓘ		
spaCy maps all language-specific part-of-speech tags to a small, fixed set of word type tags following the Universal Dependencies scheme . The universal tags don't code for any morphological features and only cover the word type. They're available as the <code>Token.pos</code> and <code>Token.pos_</code> attributes.		
POS	DESCRIPTION	EXAMPLES
ADJ	adjective	big, old, green, incomprehensible, first
ADP	adposition	in, to, during
ADV	adverb	very, tomorrow, down, where, there
AUX	auxiliary	is, has (done), will (do), should (do)
CONJ	conjunction	and, or, but
CCONJ	coordinating conjunction	and, or, but
DET	determiner	a, an, the
INTJ	interjection	psst, ouch, bravo, hello
NOUN	noun	girl, cat, tree, air, beauty
NUM	numeral	1, 2017, one, seventy-seven, IV, MMXIV
PART	particle	's, not,
PRON	pronoun	I, you, he, she, myself, themselves, somebody
PROPN	proper noun	Mary, John, London, NATO, HBO
PUNCT	punctuation	., (,), ?
SCONJ	subordinating conjunction	if, while, that
SYM	symbol	\$, %, \$, ©, +, -, ×, ÷, =, :, ☹️
VERB	verb	run, runs, running, eat, ate, eating
X	other	sfpksdpsxmsa
SPACE	space	