Natural Language Processing with PyTorch – Day 1

Yashesh A. Shroff, PhD

Dec 12, 2020

SFBay / AICamp NLP Bootcamp

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

Overview

Day 1:

1. Module 1 (20mins, Lecture): Foundations
    1. Fundamentals and application of Language Modeling Tools
    2. Classical vs DL NLP
    3. NLP Pipeline
2. Lab (20mins): NLTK from scratch
    1. Setting up your environment
    2. NLTK (tokenization)
3. Module 2 (30mins):
    1. Use NLP pipeline to process documents
    2. POS, Word embedding
4. Lab (30mins)

Break (15mins)

5. Module 3 Lecture (20mins): Key packages & libraries in NLP; dive into SpaCy
6. Lab (20mins): SpaCy
7. Lab: PyTorch (Build on Ravi's labs for PyTorch)

Transition to Ravi Ilango

8. Module 4 Lecture (30mins): TFIDF & Logistic Regression
9. Lab (30mins): Disaster Detection using TFIDF and

Day 2

1. Recap (15mins)
2. Module 5: Introduction to Transformers
    1. Theory
    2. Pre-trained models, such as BERT
3. Module 6: Text Classification
    1. Lab (20mins): Disaster Detection
    2. Lab (20mins): Headline Classifier

Break (15 mins)

    3. Lab (20mins): LSTM based sequence classifier
4. Module 7: Text summarization
    1. Lab (20mins): Text summarization with and without Transformers
5. Module 8: Training a chatbot
    1. Lab (20mins)
6. NLP in production
    1. Scheduler Overview
    2. Implementation walk-through

Desired background:
Python coding skills, intro to PyTorch framework is helpful, familiarity with NLP

A word about the training (setting expectations for the next 3 hours)

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”

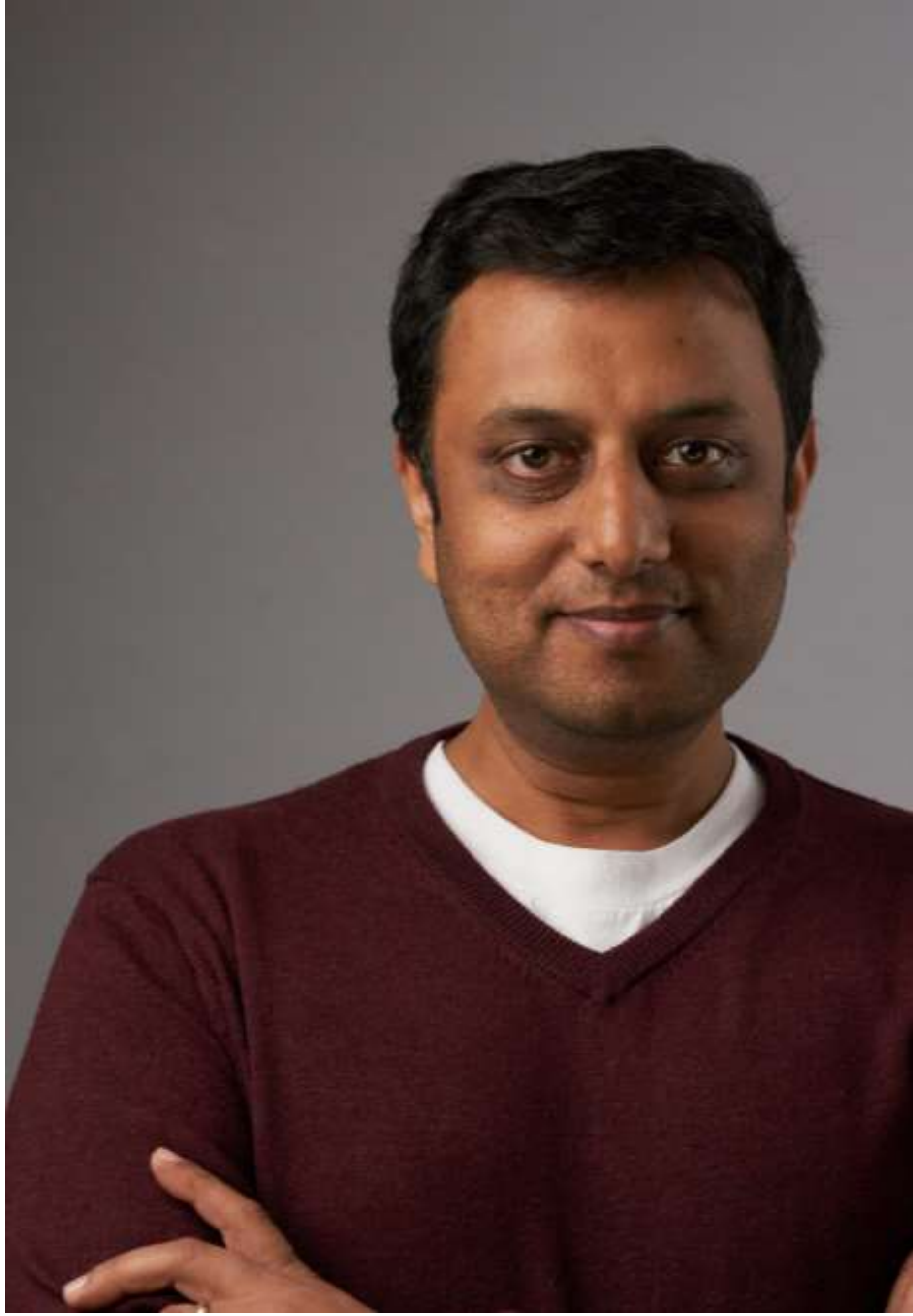
J.R. Firth, 1957

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, yshroff@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/ravi-ilango/acm-dec-2020-nlp
```

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)

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/

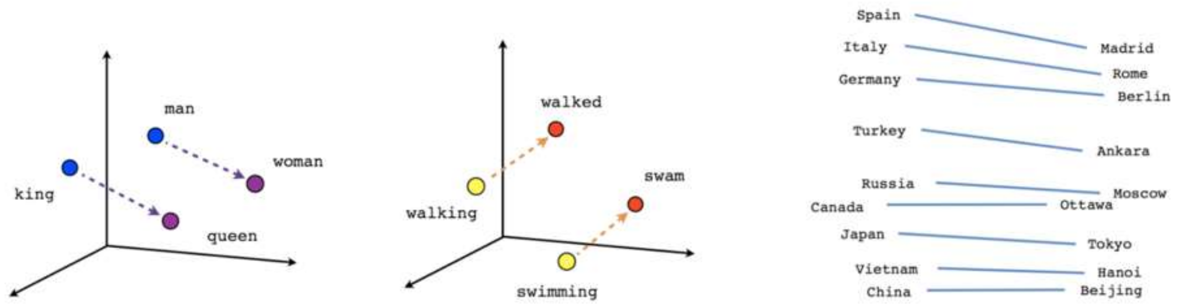
Classical vs. DL NLP

Classical:

- Task customization for NLP Applications

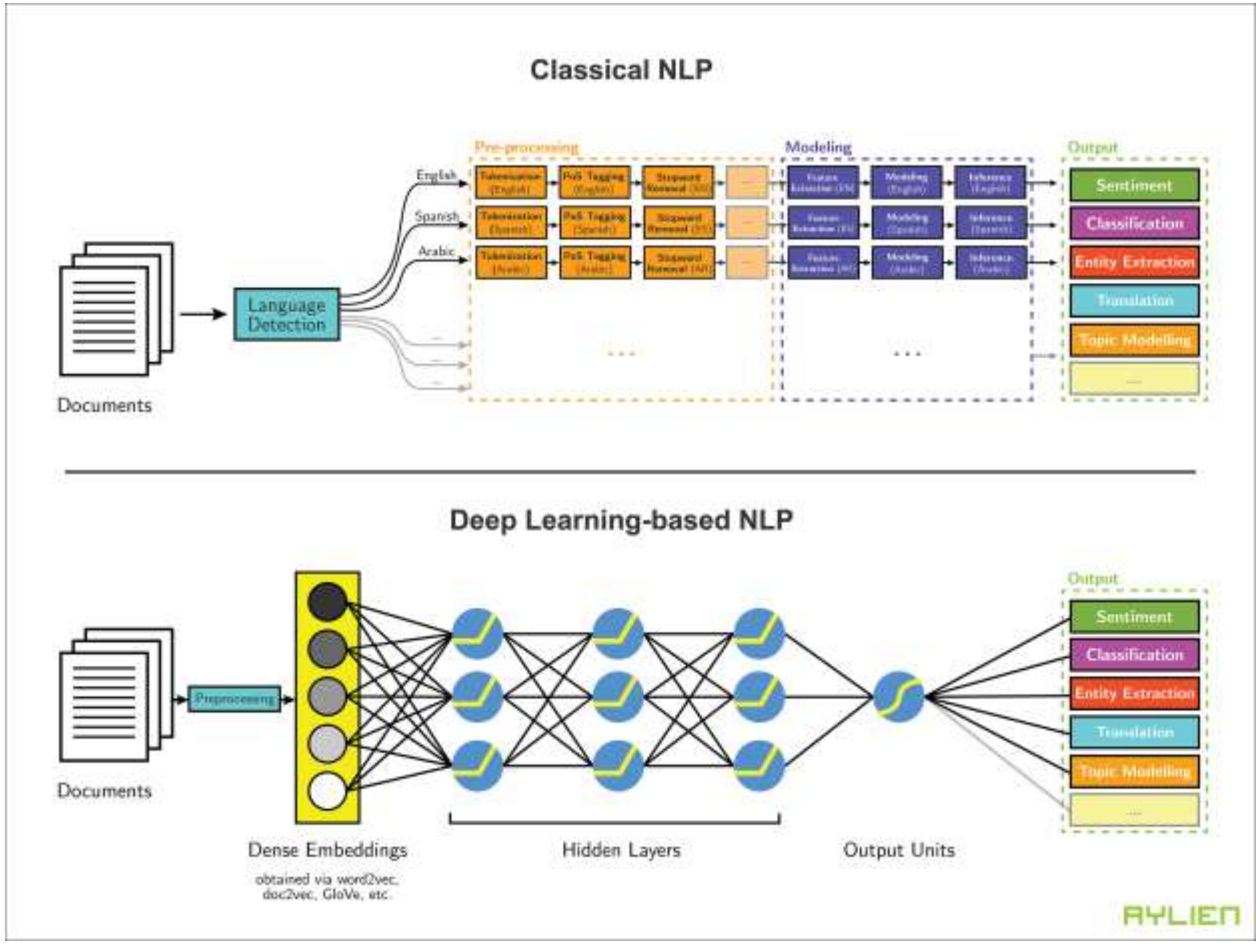
DL Based NLP

- 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

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: Voiq Sales & Marketing

Text and auto-generation: Gmail

Sentiment analysis: Social media (finance, reviews)

Information extraction: Unstructured (news, finance)

NLP 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.

Normalize

- Normalize text by converting it to all lower case, removing punctuation, & extra white spaces

Tokenize

- Split text into words, n-grams, or phrases (tokens)

"I love morning runs"
- Unigrams: "I", "love", "morning", "runs"
- Bigrams (n=2): "I love", "love morning", "morning runs"
- Trigrams (n=3): "I love morning", "love morning runs"

Remove Stop words

- Remove common words like "a", "the", "and", "on", etc.

Stemming

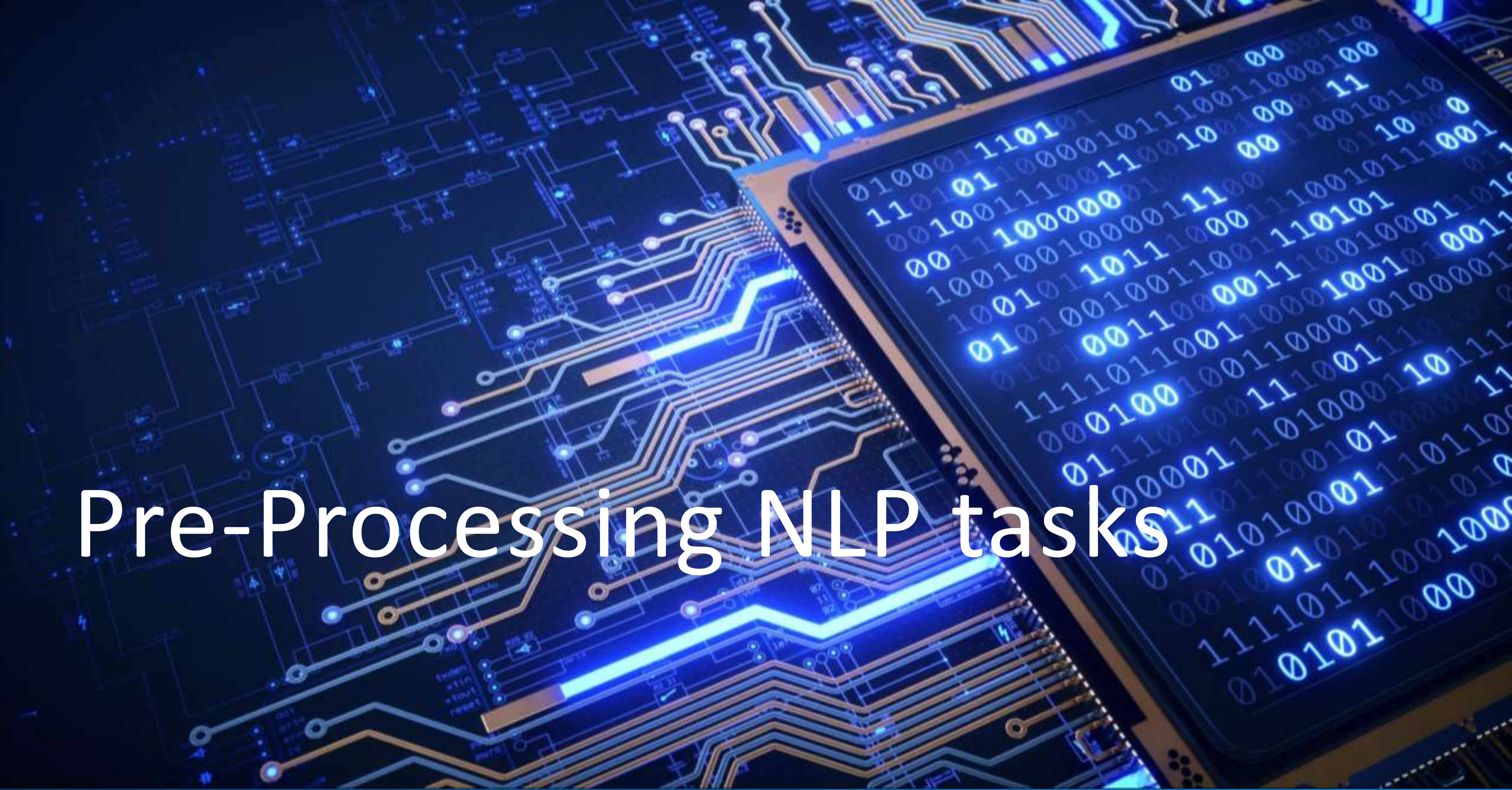
- Convert to stem

ex. Dancer, dancing, dance become 'danc'
Studies, Study, Studying: Stud

POS, NER

- Identify Parts of Speech (POS), such as verb, noun, named entity
- Lemmatization: root word (am, are, is >> be)

Example: Raw tweet	Preprocessed output
@huggingface is building a fantastic library of NLP datasets and models at http://huggingface.com	Build fantastic library NLP dataset model



Pre-Processing NLP tasks

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 lemmed]
```

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`

Sentiment Analysis

Text Classification

Text Classification with Neural Networks

Tokenize — Tokenize the document into words

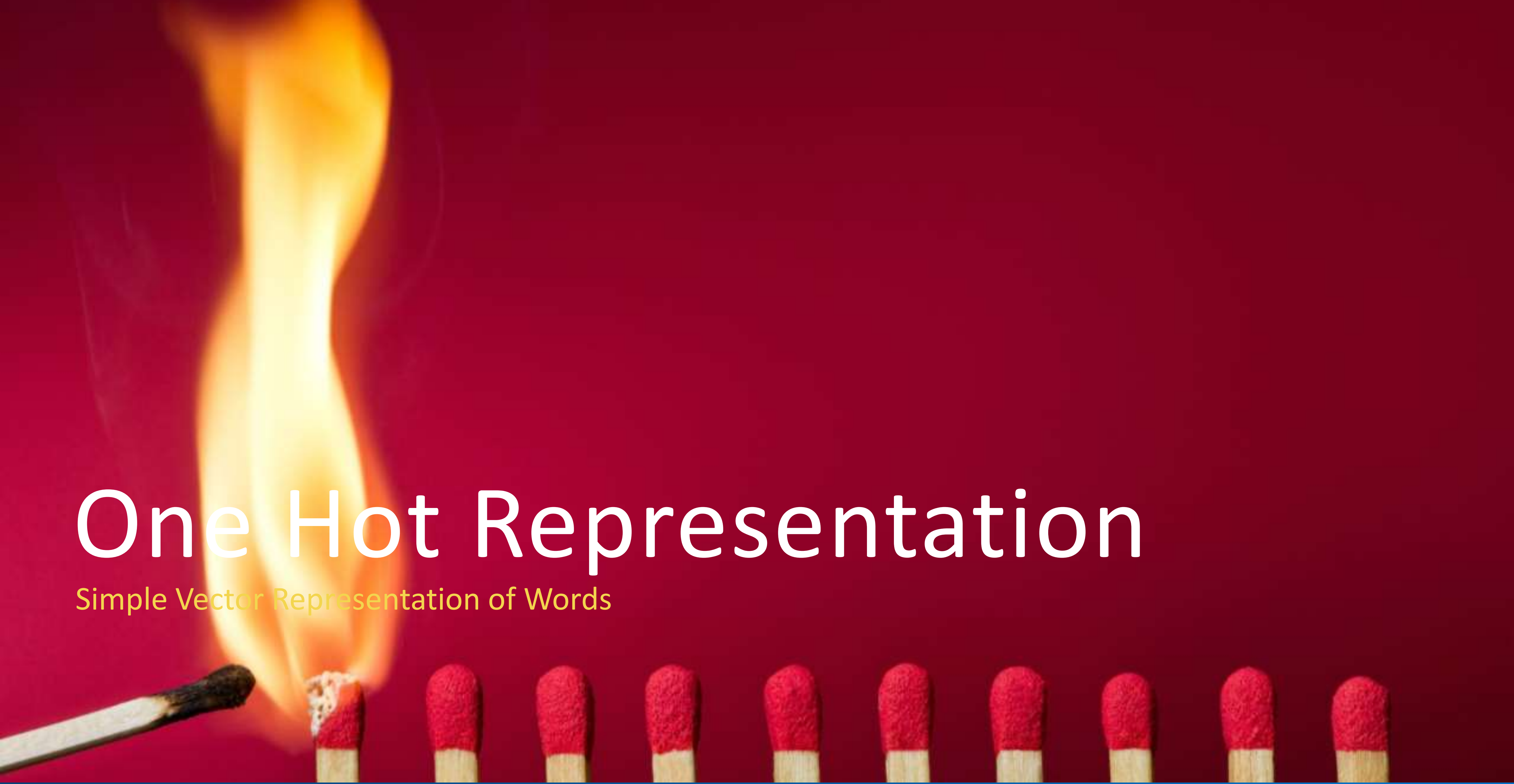
Represent — Represent word in numeric encoding

Collect — Collect into a Tensor to represent a document

One Hot Encoding

Frequency

Prediction



One Hot Representation

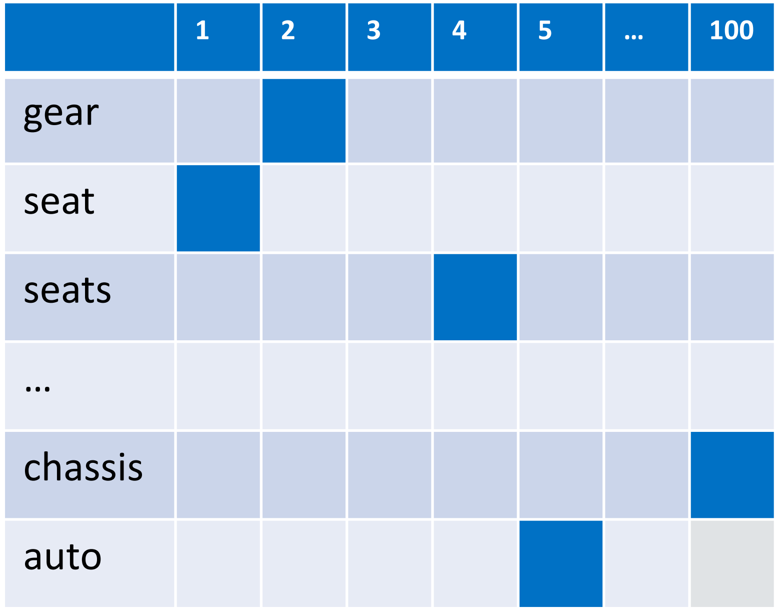
Simple Vector Representation of Words

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



Challenges with this approach:

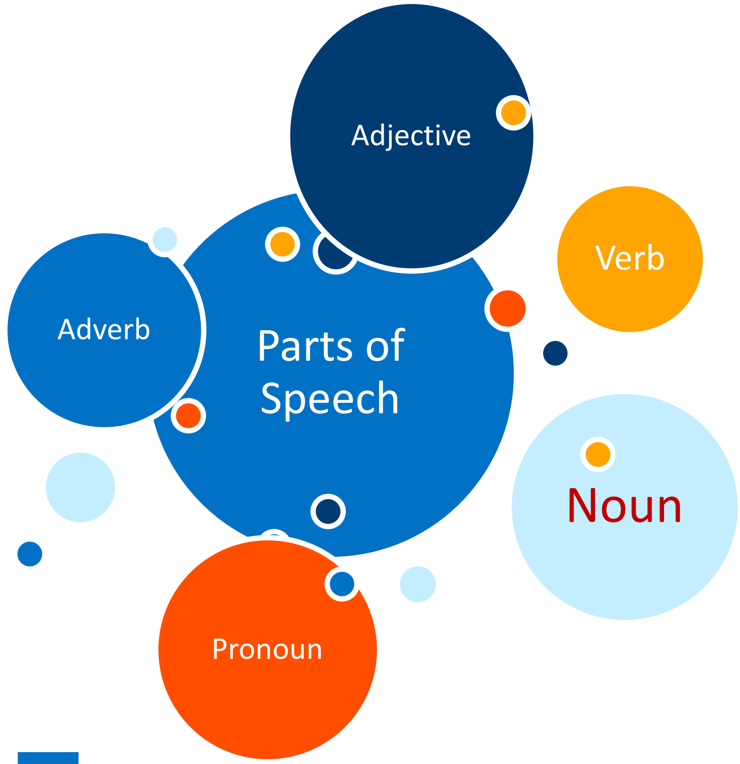
- Curse of dimensionality: Memory capacity issues
  - The size of the matrix is proportionate to vocab size (there are roughly 1 million words in the English language)
- Lack of **meaning** representation or word **similarity**
  - Hard to extract meaning. All words are equally apart
    - “seat” and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

Lab

Google Colab:

- `02_inefficient.ipynb`

Parts of Speech Tagging



Rule based

- ~1k+ constraints

Transformation

Stochastic

- Training corpus
- Probability of certain tag occurring

One tag for each part of speech

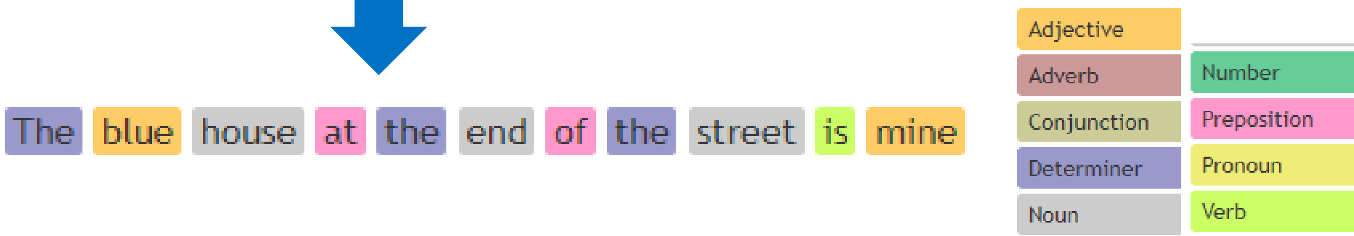
- Choose a courser tagset (~6 is useful)
- Finely grained tagsets exist (ex. Upenn Tree Bank II)

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.”



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

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-IDF vectorization

Document 1: “This is about cars”
Document 2: “This is about kids”

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	

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

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

Calculate $TF\ x\ IDF$

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

Co-Occurrence Vector

“He is not lazy. He is intelligent. He is smart”

	He	is	not	lazy	intelligent	smart
He	0	[illegible]	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	[illegible]	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	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

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

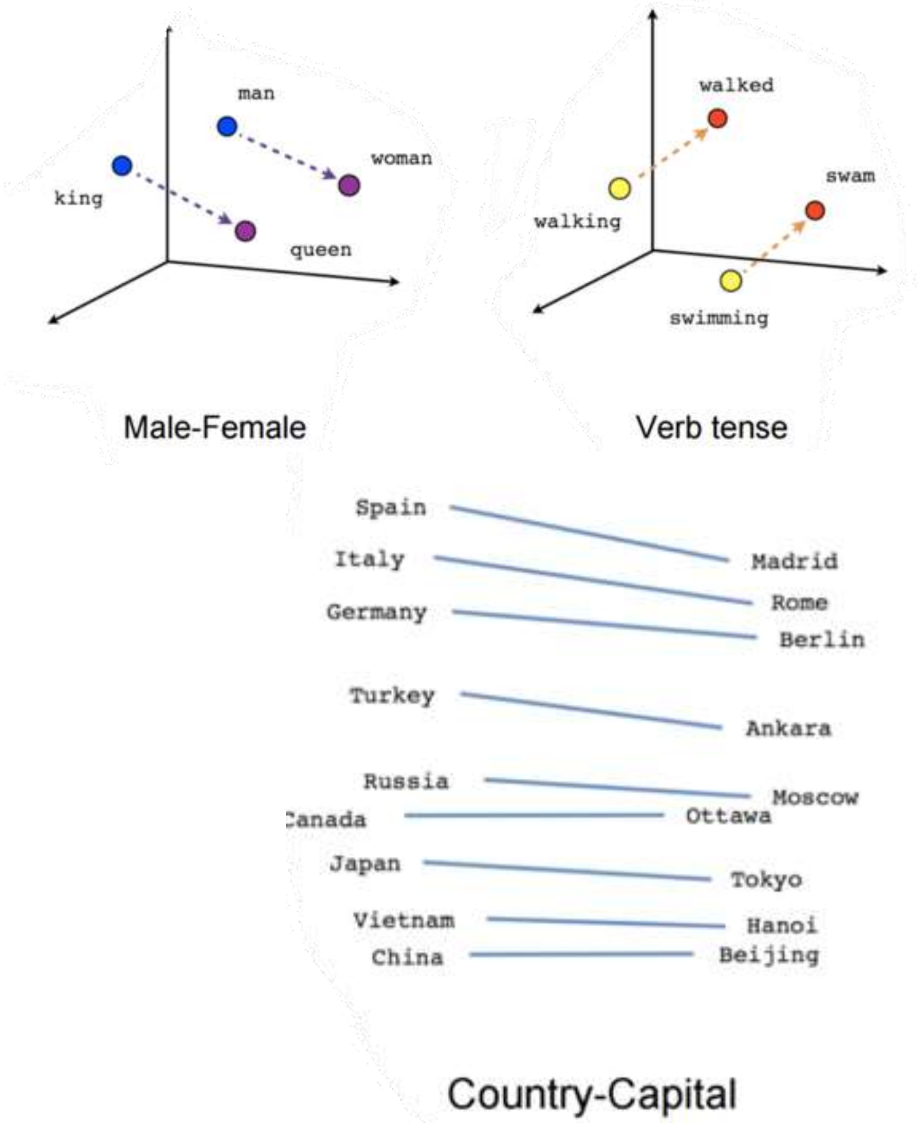
m: # of terms
n : m minus stop words
- Uses SVD decomposition and PCA to reduce dimensionality

- 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”)

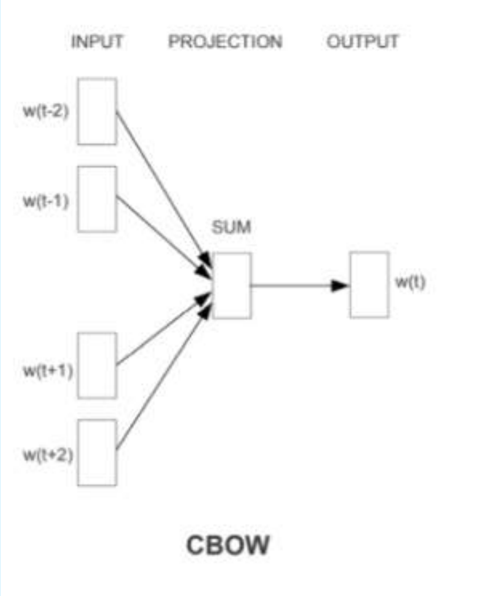


Reference: https://arxiv.org/abs/1301.3781
(Efficient Estimation of Word Representations in Vector Space)

Word Embedding

Prediction based Word2Vec

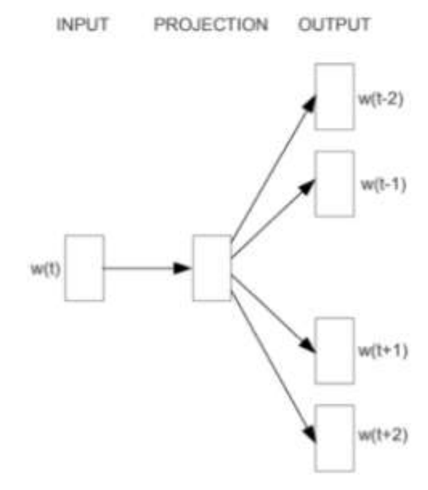
CBOW



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

- 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)

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])

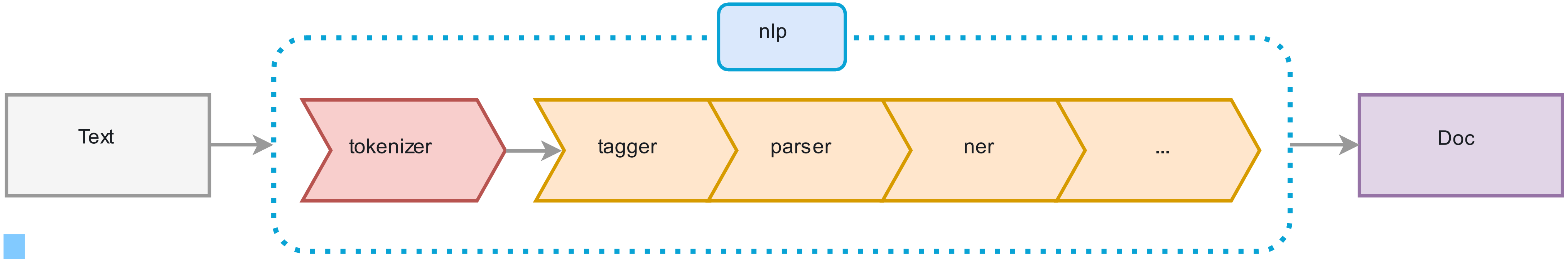
Reference: *Exploiting Similarities among Languages for Machine Translation*

SpaCy: NLP Library

~ Building on footsteps of Giants ~

aka *"NLTK" alternative*

SpaCy



Compared to NLTK, SpaCy is *fast, accurate, with integrated word vectors.*

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

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

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

doc = nlp(text)
```

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

SpaCy Models: https://spacy.io/models/en

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 `Token.pos` and `Token.pos_` 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	

Objective:

- Covered in lecture
  - Word–Embedding. Tokenization:
- NER: showing country
- POS
- Powered Regex with NER

SpaCy

Lab:

- 03_SpaCy.ipynb

PyTorch - Intro

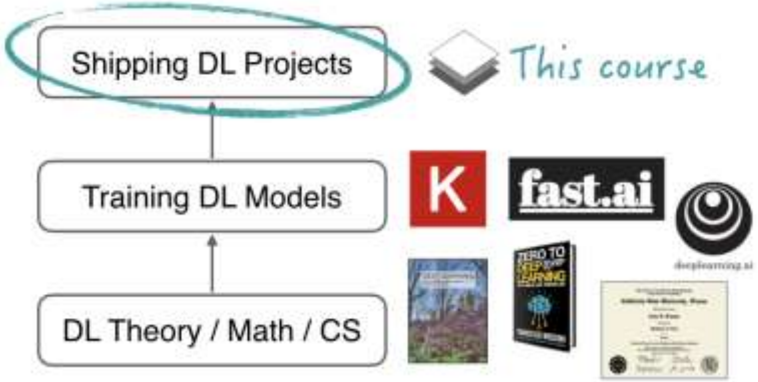
Lab:

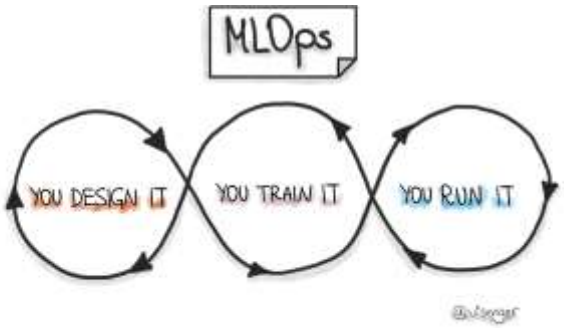
- 04_pytorch_intro.ipynb

Operationalizing Machine Learning Pipelines

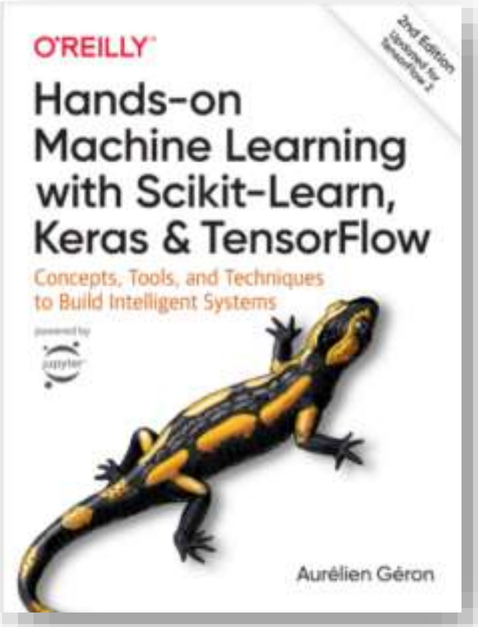
Resources for ML in Production

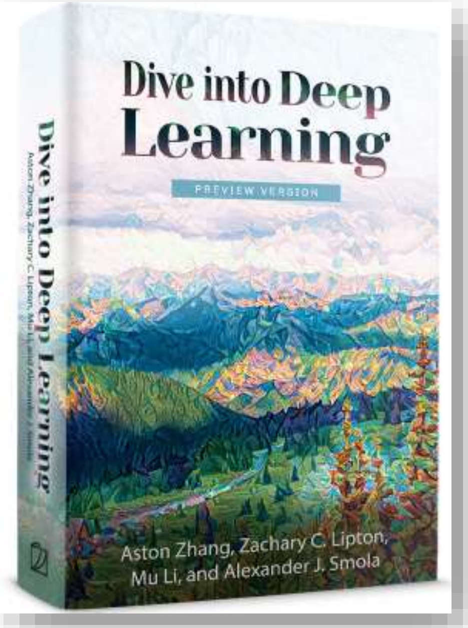
1. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition**
2. **Dive into Deep Learning (https://d2l.ai/:** *Aston Zhang, Zack C. Lipton, Mu Li, and Alex J. Smola*
3. **Full Stack Deep Learning (https://course.fullstackdeeplearning.com/)**
4. **Designing Data-Intensive Applications (***Martin Kleppmann***)**
5. **Building Machine Learning Pipelines (***Hannes Hapke and Catherine Nelson***)**
6. **Building Machine Learning Powered Applications (***Emmanuel Ameisen***)**
7. **Introducing MLOps: How to Scale Machine Learning in the Enterprise (***Clément Stenac, Léo Dreyfus-Schmidt, Kenji Lefèvre, Nicolas Omont, and Mark Treveil***)**
8. **Awesome MLOps (**https://github.com/visenger/awesome-mlops **)**
9. **Awesome production machine learning (https://github.com/EthicalML/awesome-production-machine-learning)**
10. **Kubeflow for Machine Learning (***Trevor Grant, Holden Karau, Boris Lublinsky, Richard Liu, Ilan Filonenko***)**



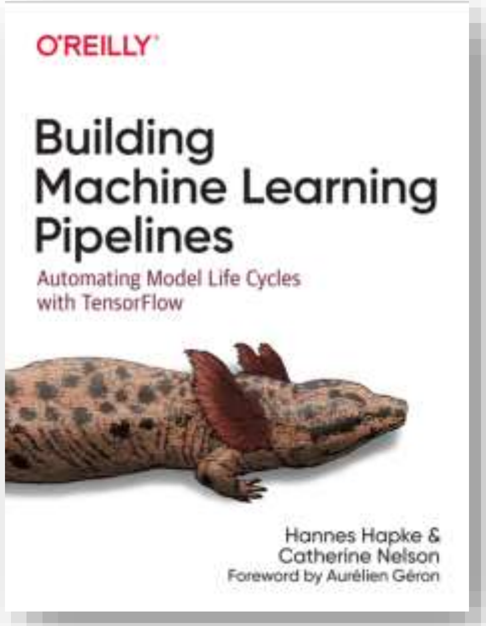


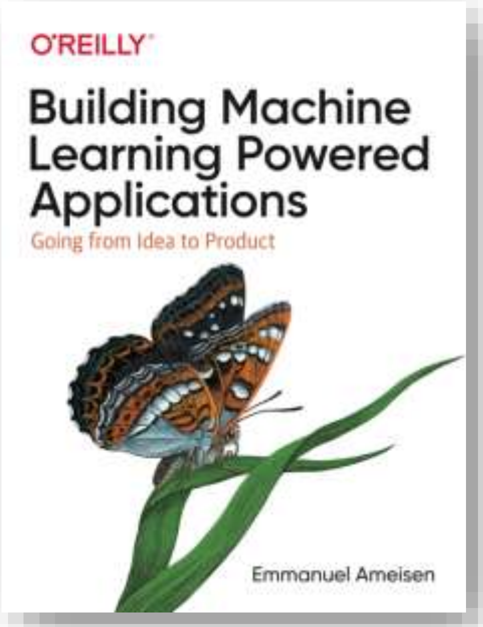
Credit: https://elvissaravia.substack.com/p/my-recommendations-to-learn-machine

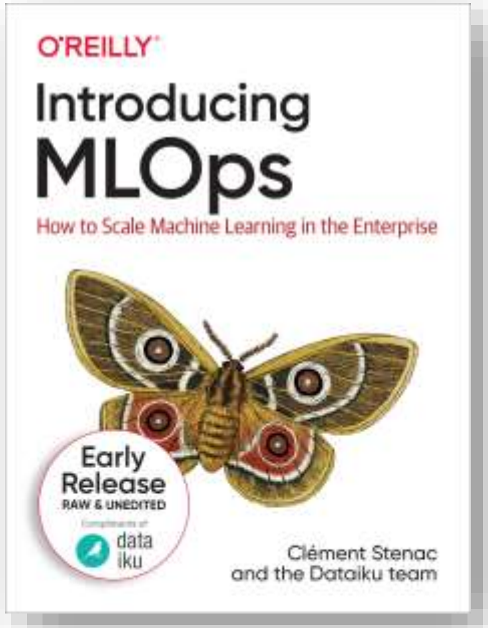

O'REILLY
Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
Aurélien Géron

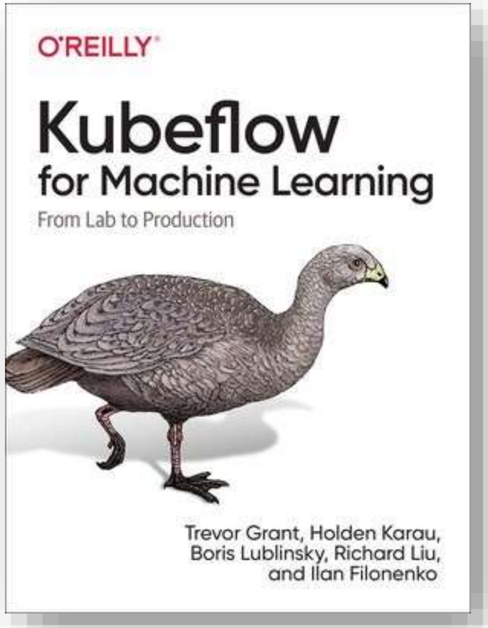

Dive into Deep Learning
PREVIEW VERSION
Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola


O'REILLY
Designing Data-Intensive Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS
Martin Kleppmann


O'REILLY
Building Machine Learning Pipelines
Automating Model Life Cycles with TensorFlow
Hannes Hapke & Catherine Nelson
Foreword by Aurélien Géron

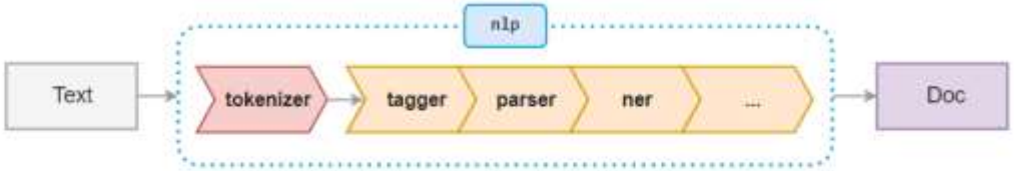

O'REILLY
Building Machine Learning Powered Applications
Going from Idea to Product
Emmanuel Ameisen


O'REILLY
Introducing MLOps
How to Scale Machine Learning in the Enterprise
Early Release
RAW & UNEDITED
data iku
Clément Stenac and the Dataiku team


O'REILLY
Kubeflow for Machine Learning
From Lab to Production
Trevor Grant, Holden Karau, Boris Lublinsky, Richard Liu, and Ilan Filonenko

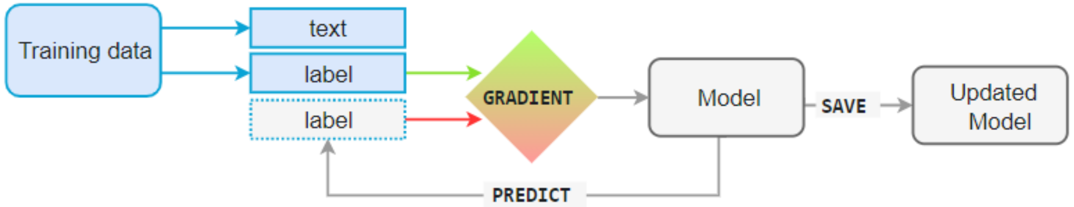
Language Processing Pipelines

- SpaCy’s `nlp` class first tokenizes the text
- Default pipeline: tagger, parser, NER
- Can add custom components at any point in the pipeline
- Finally, produce a `Doc` object



Training Models

- SpaCy’s `nlp` class first tokenizes the text
- Default pipeline: tagger, parser, NER
- Can add custom components at any point in the pipeline
- Finally, produce a `Doc` object



Is there a way to automate the flow?

Reference: spacy.io

Wrapping Up

One more thing…

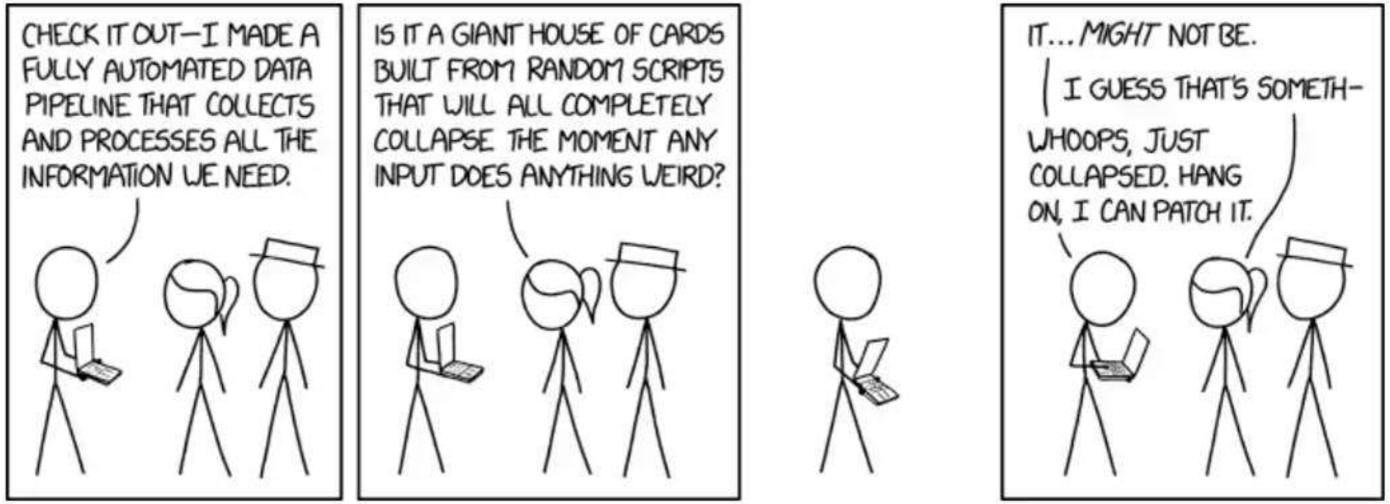


Image source: [xkcd: Data Pipeline](https://xkcd.com/2054/)

Creating NLP pipelines

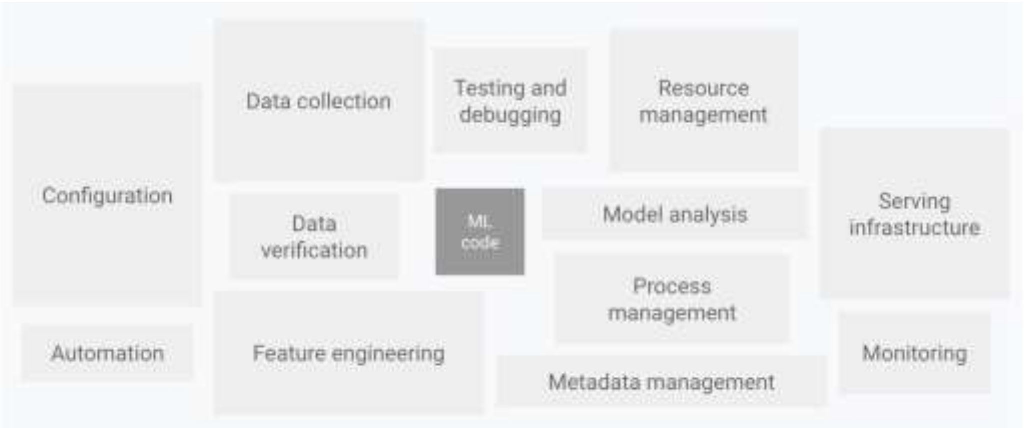
Apache Airflow DataRobot

Problem statement:

- Building a deep learning model is a small part of an end-to-end cycle of deploying an app
- Building an NLP pipeline is critical in managing model versions, dataset versions, and ensuring resiliency of the infrastructure

Directed Acyclic Graph, or DAG, to the rescue

- DAG is a data pipeline, an ETL process, or a workflow
- Each node or task of DAG includes an operator: Python, Bash, etc.
- When to use:
  - Going beyond cron jobs
  - Usually when business logic demands it



Source: https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Airflow installation

Setup:

```
pip3 install apache-airflow

# Set home env
export AIRFLOW_HOME=$(pwd)

# Initialize dB
airflow initdb
```

```
# Client
airflow scheduler

# in a different terminal, run:
airflow webserver
```

Simple DAG Script

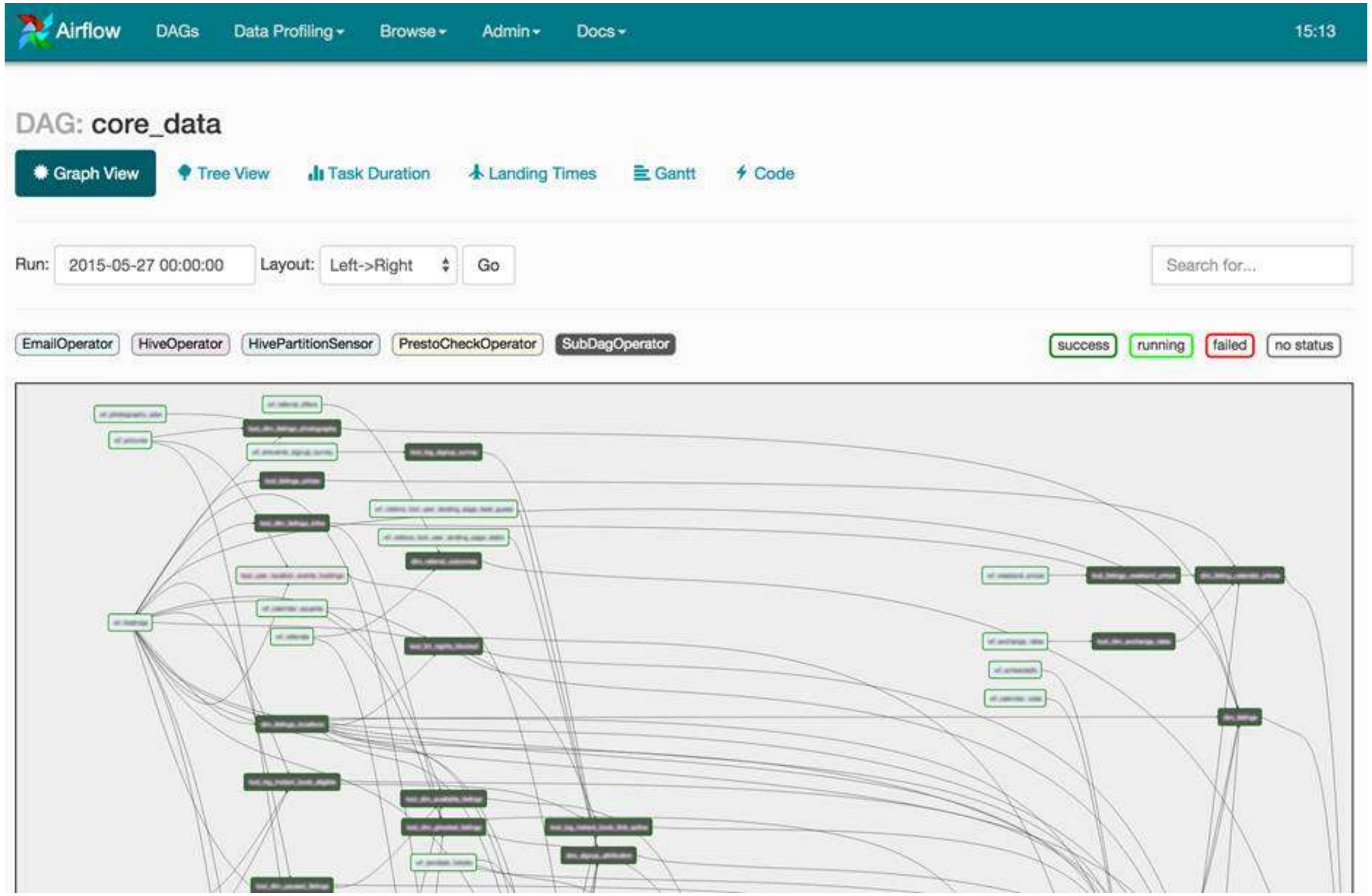
```
# Python standard modules
from datetime import datetime, timedelta
# Airflow modules
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
default_args = {
      'owner': 'airflow’,
      'depends_on_past': False,
      # Start on 27th of June, 2020
      'start_date': datetime(2020, 6, 27),
      'email': ['airflow@example.com’],
      'email_on_failure': False,
      'email_on_retry': False,
      # In case of errors, do one retry
      'retries': 1,
      # Do the retry with 30 seconds delay after the error
      'retry_delay': timedelta(seconds=30),
      # Run once every 15 minutes
      'schedule_interval': '*/15 * * * *’
}
```

```
# After defining the parameters, tell the DAG what to actually do
and # the dependencies for each task
with DAG(
      dag_id='simple_bash_dag’,
      default_args=default_args,
      schedule_interval=None,
      tags=['my_dags’],
) as dag:
      #Here we define our first task
      t1 = BashOperator(
      bash_command="touch ~/my_bash_file.txt",
      task_id="create_file")
      #Here we define our second task
      t2 = BashOperator(bash_command="mv ~/my_bash_file.txt
      ~/my_bash_file_changed.txt",
      task_id="change_file_name")
      # Configure T2 to be dependent on T1’s execution t1 >> t2
```

Ref: https://towardsdatascience.com/data-pipeline-orchestration-on-steroids-getting-started-with-apache-airflow-part-1-22b503036ee

How it looks in practice

- Data warehousing: Organize & clean input text
- A/B testing (trying out different models)
- Business Policy & governance compliance
- AWS – Managed Workflow for Apache Airflow



Goto:
https://airflow.apache.org/docs/stable/tutorial.html
https://aws.amazon.com/blogs/aws/introducing-amazon-managed-workflows-for-apache-airflow-mwaa/