# Natural Language Processing with PyTorch – Day 1

Yashesh A. Shroff, PhD

Dec 12, 2020

SFBay / AICamp NLP Bootcamp

*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"

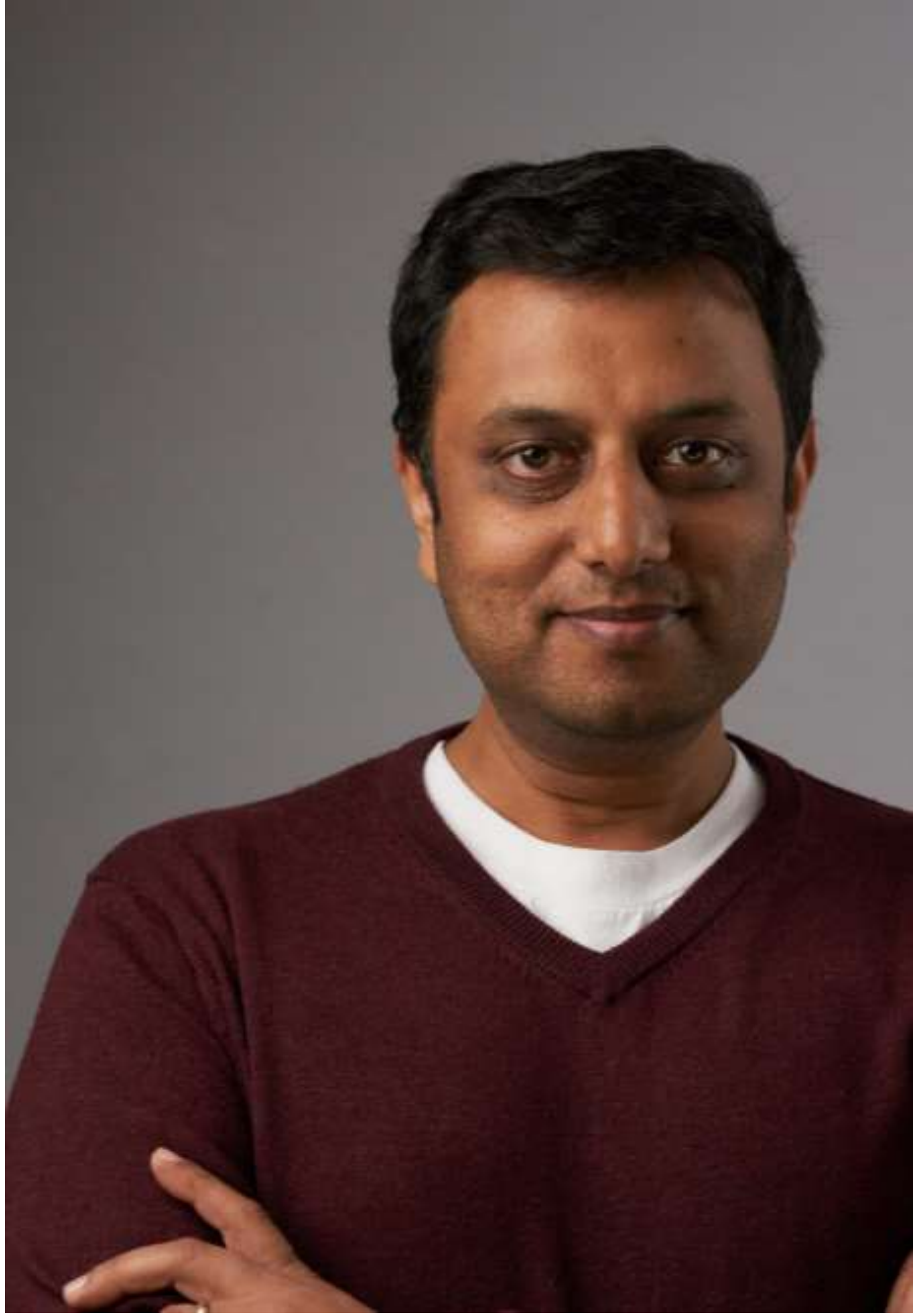
J.R. Firth, 1957

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, 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
```

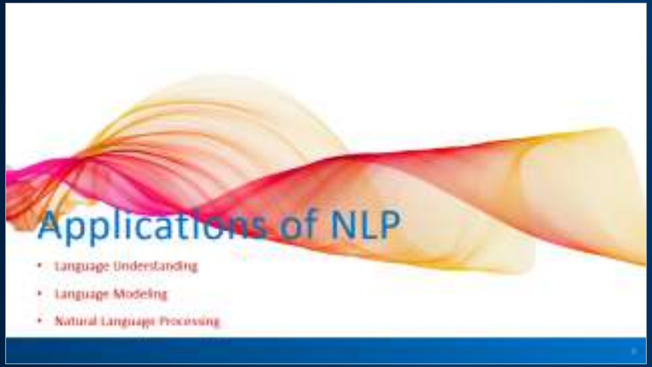
Use conda or pipenv to install the requirements dependencies in a virtual environment.

```
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)

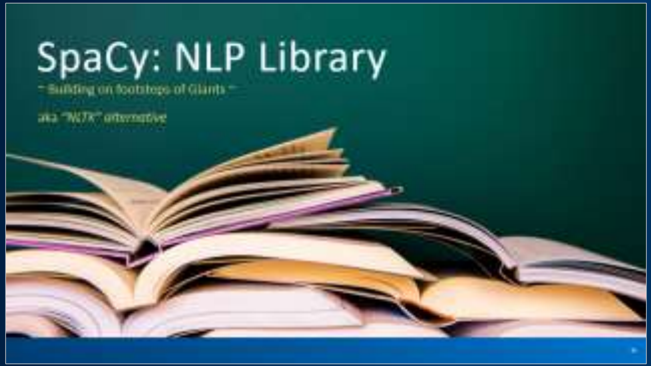
# Part 1: Foundations of NLP

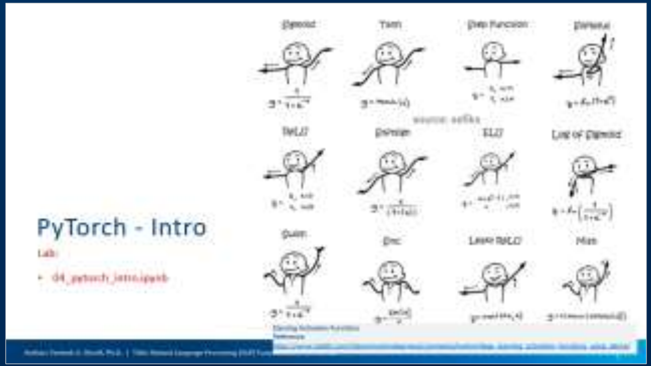







# Part 2: Practicum





# 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**

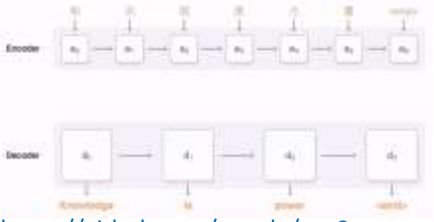
**Voicebots**: Voiq Sales & Marketing

**Text and auto-generation**: Gmail

**Sentiment analysis**: Social media (finance, reviews)

**Information extraction**: Unstructured (news, finance)

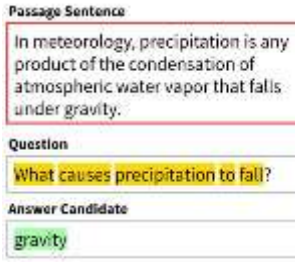
# NLP Tasks



https://github.com/google/seq2seq

## Machine Translation

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




## Question Answering

- Benchmarks:
  - https://paperswithcode.com/task/question-answering
- Knowledge-base answering
- Open-domain question answering
- Answer selection
- Community question answering

Text Classification Algorithms: A survey

## 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 |
| Text summarization | Natural Language Inference |
| Information Retrieval | Dependency Parsing |
| Dialog | Emotion Recognition |
| Semantic Textual Similarity | Reading comprehension |

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


# 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

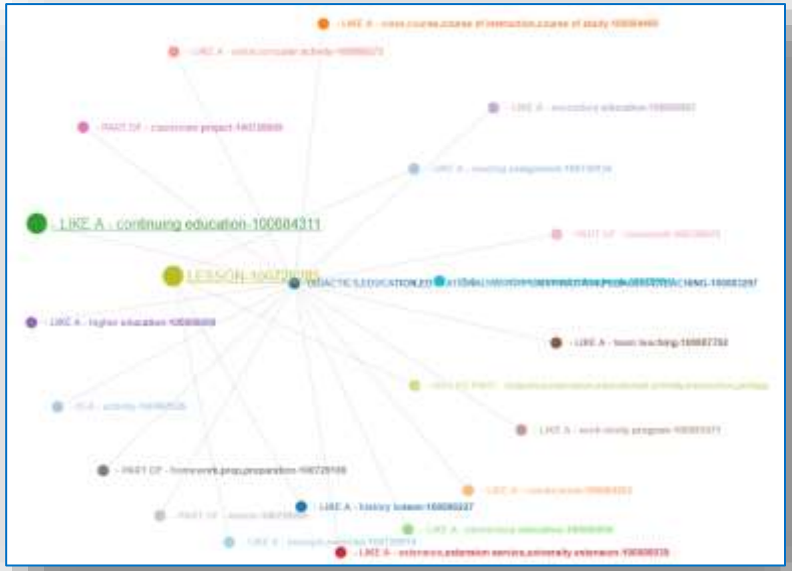
- word2vec: Mikolov et. al. (Google)
- GloVe: Pennington et al., Stanford CS. EMNLP 2014
- ElMo:
- ELMo (Embeddings from Language Models)
  - Memory augmented deep learning
- Survey paper (https://arxiv.org/abs/1708.02709)
  - Blog (https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d)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

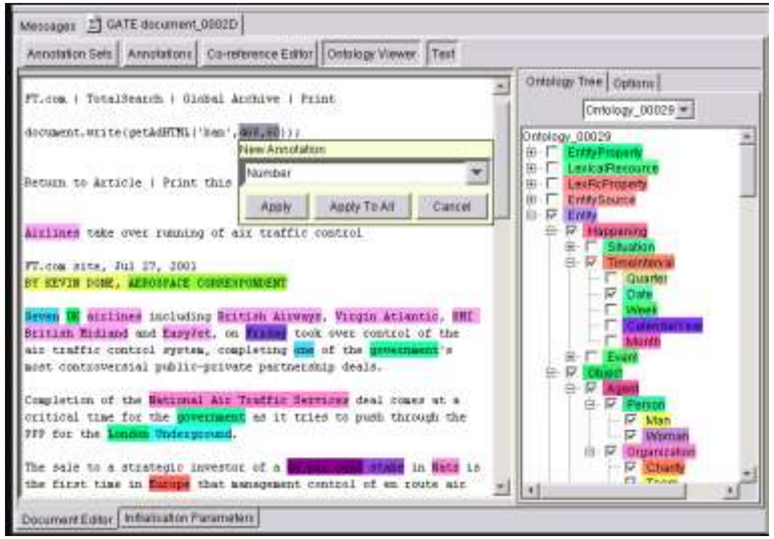
Ref: https://eigenfoo.xyz/transformers-in-nlp/

# 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-thesaurus.com/wordnet.php?link=100883297

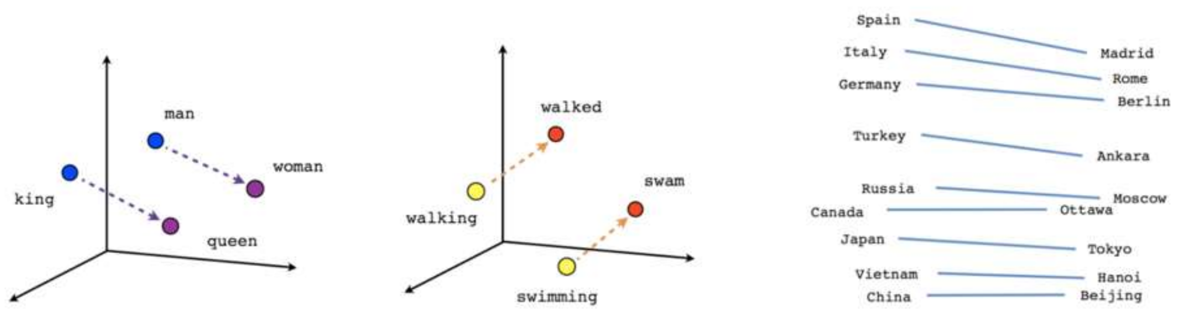
# Classical vs. DL NLP

## Classical:

- Task customization for NLP Applications

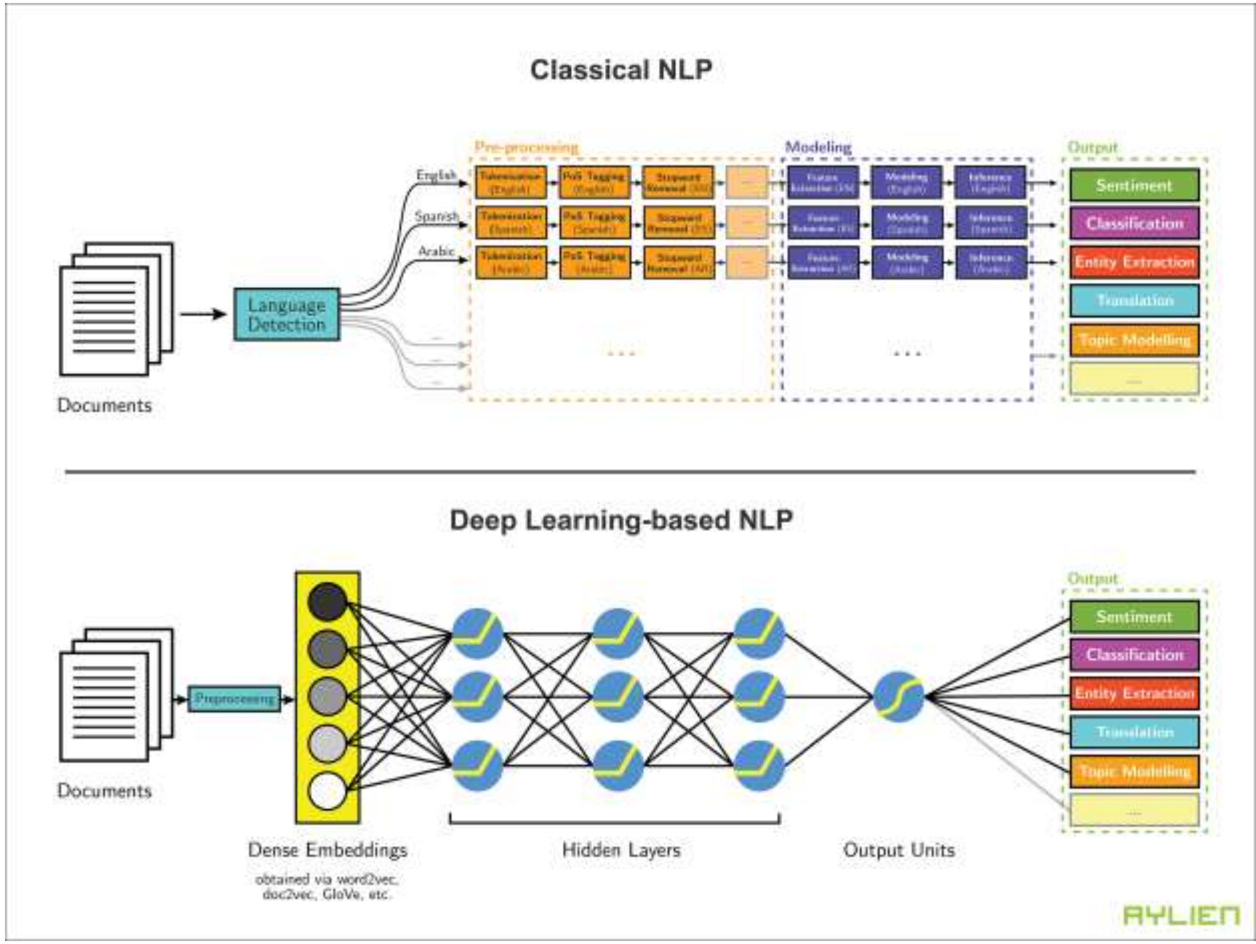
## DL Based NLP

- 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

Feature Extraction → Feature Representation → Evaluation

# Popular Machine Learning Algos for NLP

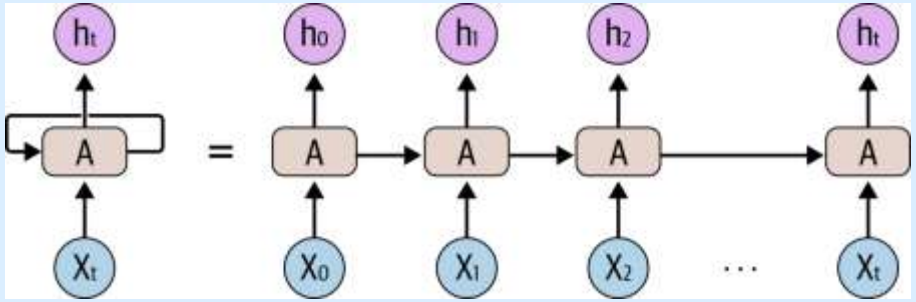
| Algorithm | Description |
|---|---|
| Naïve Bayes | Assumes feature independence (naïve)<br>Ex. Frequency of specific words for classification |
| Support Vector Machines | Leans optimal (linear or non-linear) decision boundaries between classes (sports vs political articles) |
| Hidden Markov Models | 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

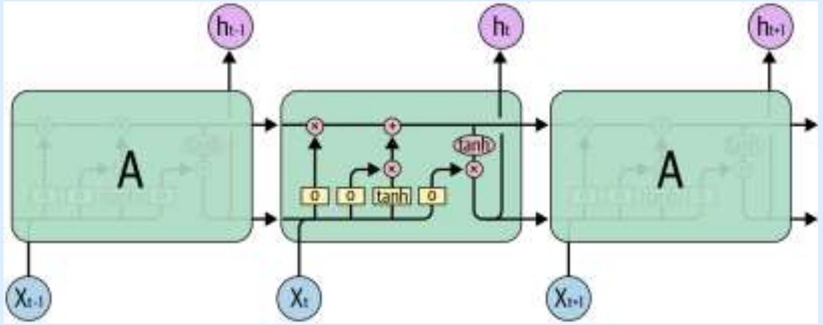
Recurrent Neural Networks

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



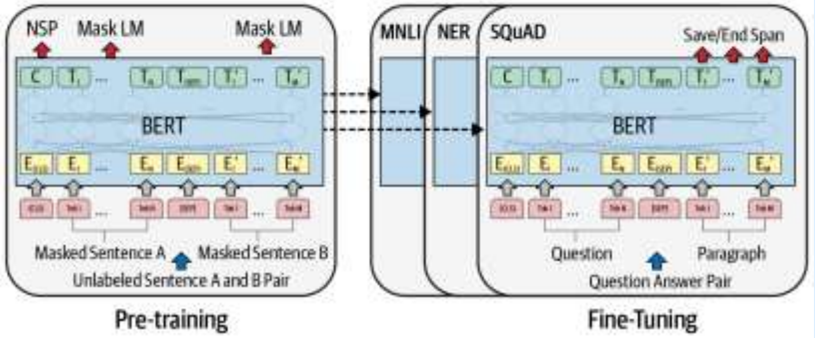

Long-Short Term Memory

- 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

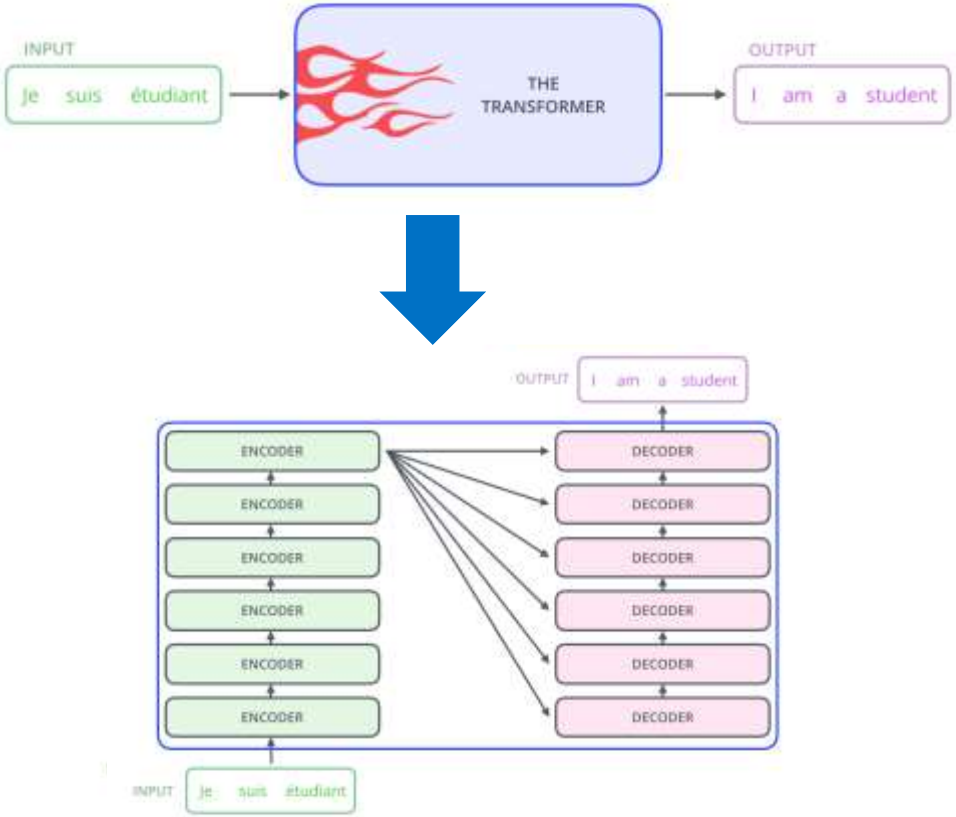


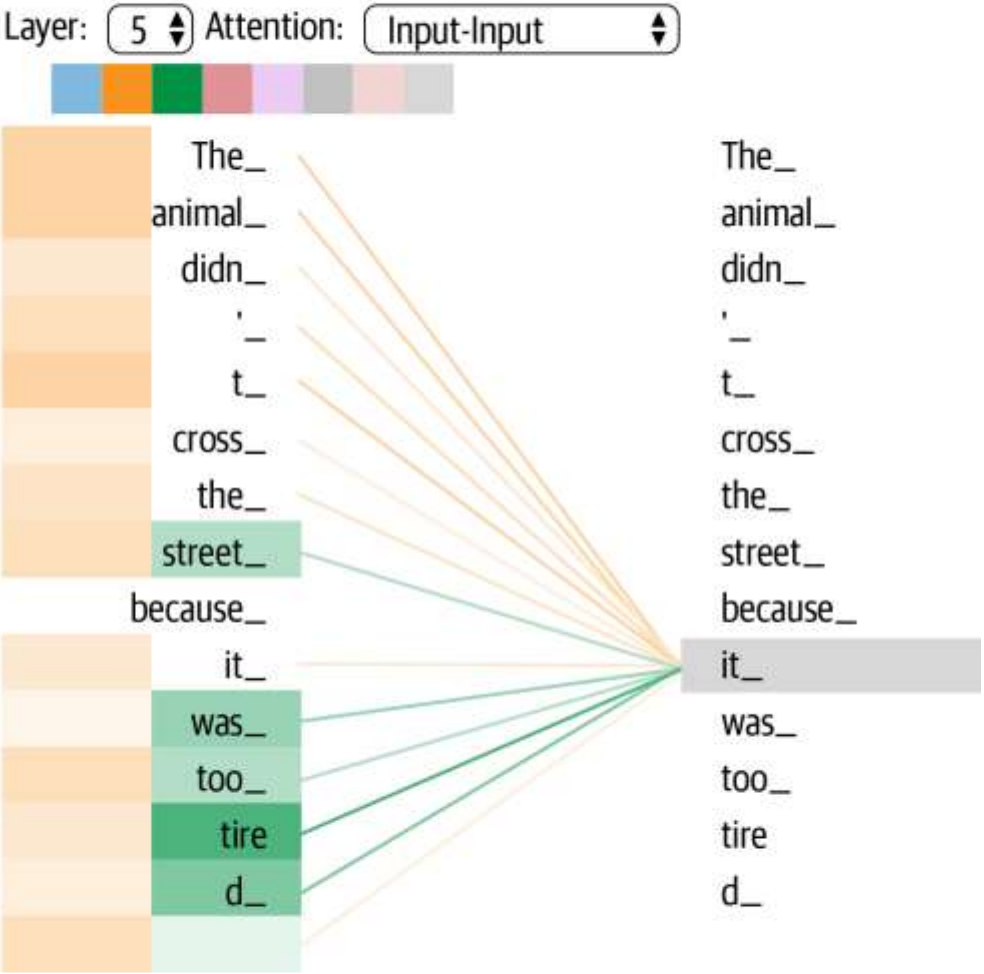



Christopher Colah: Understanding LSTM Networks
Jacob, et. Al "BERT Pre-Training Paper"

# Transformer (motivation)

## Self-Attention Mechanism



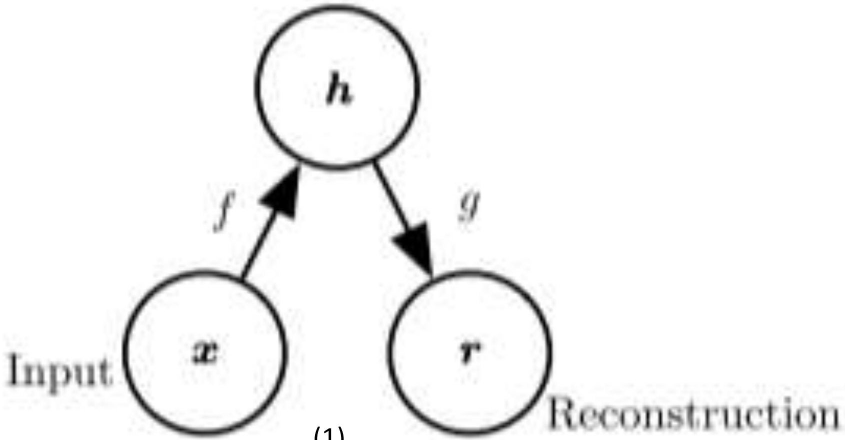




Jay Alammar: [The Illustrated Transformer]

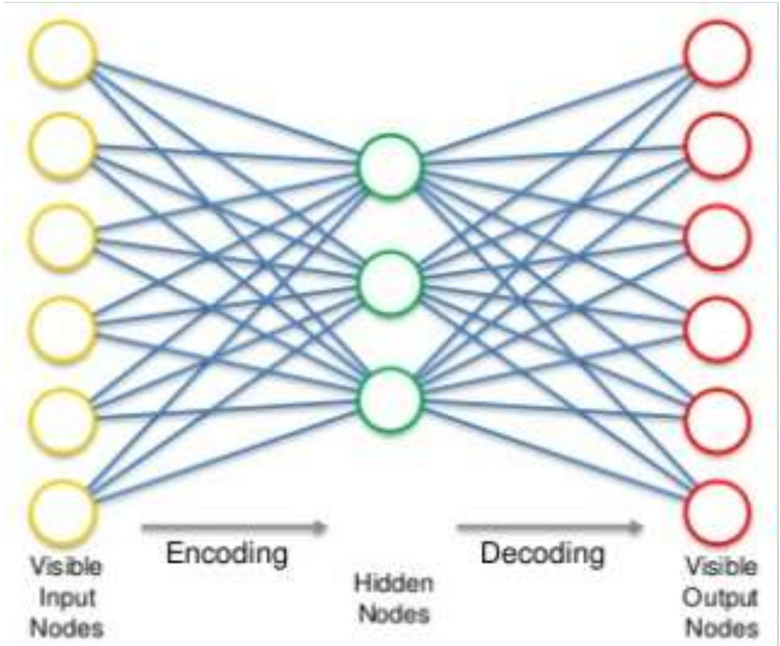
# Autoencoder

## 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

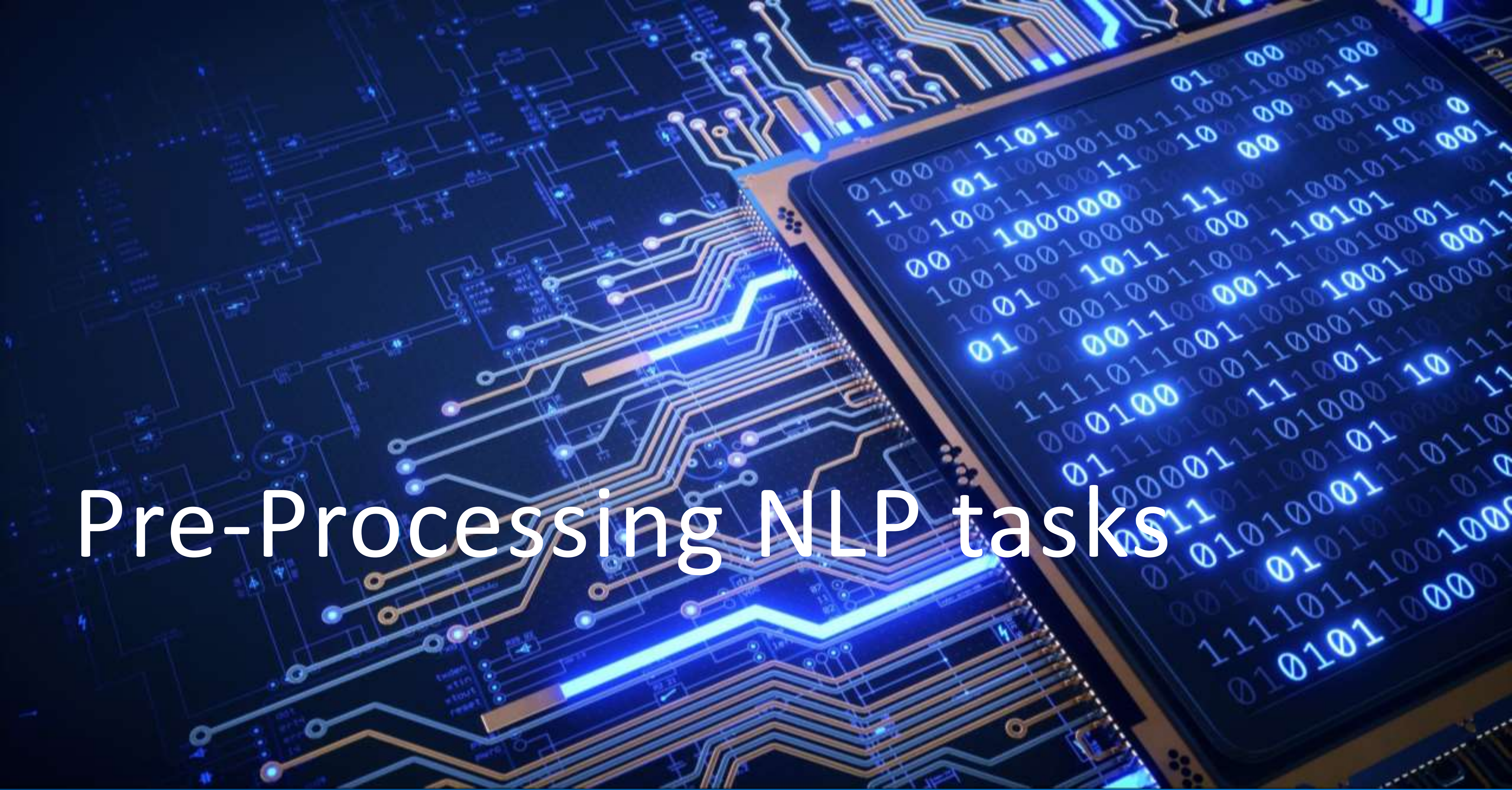



(1)




(2)

*Ref:*
*1) Ian Goodfellow, "[The Deep Learning Book]"*
*2) Kirill Eremenko, [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.

**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 |

# 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)
```

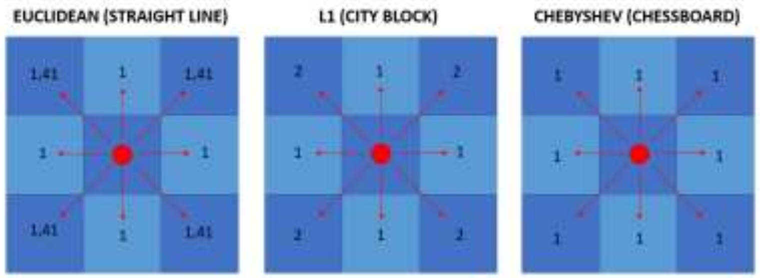
# Lab

Google Colab:

1. `01_NLP_basics.ipynb`

# Distance Similarity

# Measuring distances: Euclidean, L1, & L-Infinity




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

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

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

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

- Chebyshev Distance
  - Also known as L-Infinity or Chessboard distance
  - Can go one step in any direction

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

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

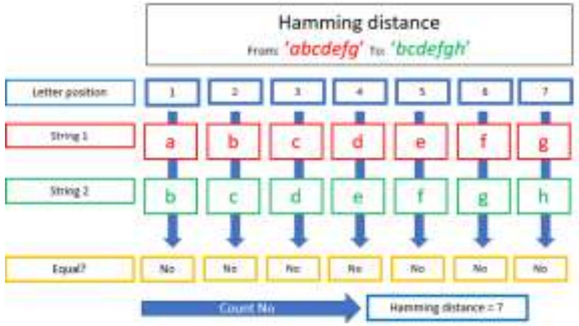
## Levenshtein Distance

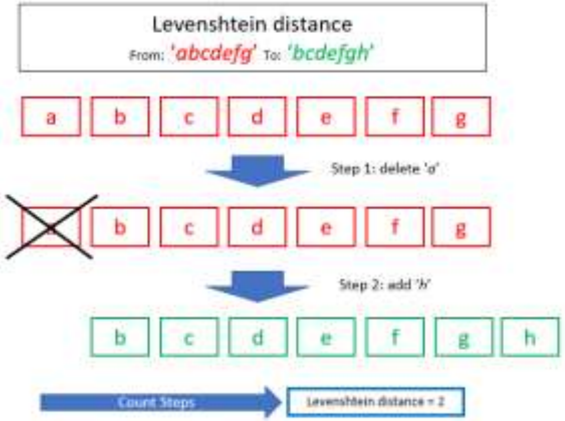
- Given by the number of ops required to convert one string to another
  - Inserting, Deleting, Substituting characters

## 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]:  from sklearn.metrics import pairwise

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

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

In [6]:  pairwise.cosine_similarity(matrix)
Out[6]:  array([[1.        , 0.81649658, 0.36927447],
                [0.81649658, 1.        , 0.45226702],
                [0.36927447, 0.45226702, 1.        ]])
```



Ref: https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcdf850e510

# Sentiment Analysis

Text Classification

# Text Classification with Neural Networks

**Tokenize** — Tokenize the document into words

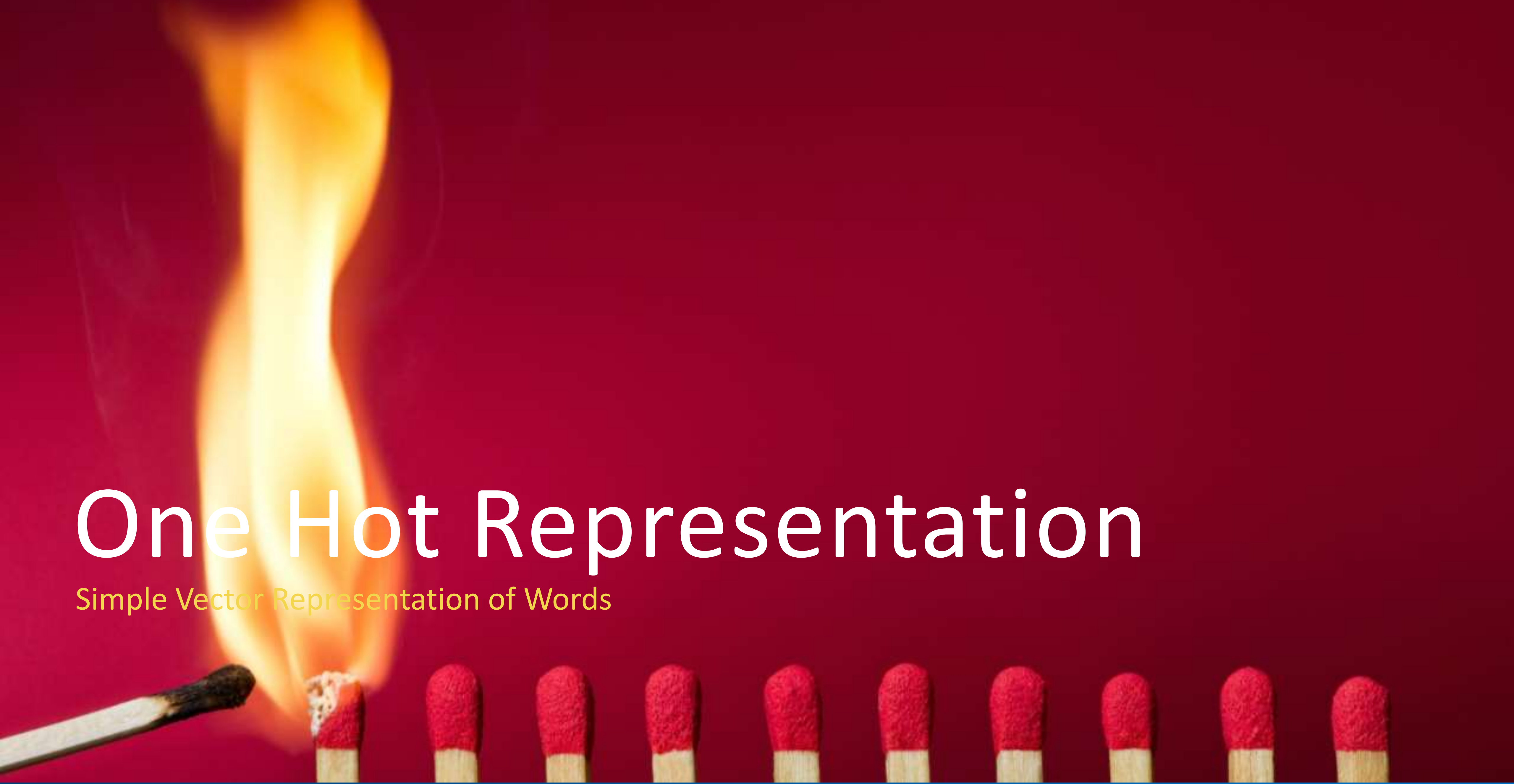
**Represent** — Represent word in numeric encoding

**Collect** — Collect into a Tensor to represent a document

One Hot Encoding

Frequency

Prediction



# One Hot Representation

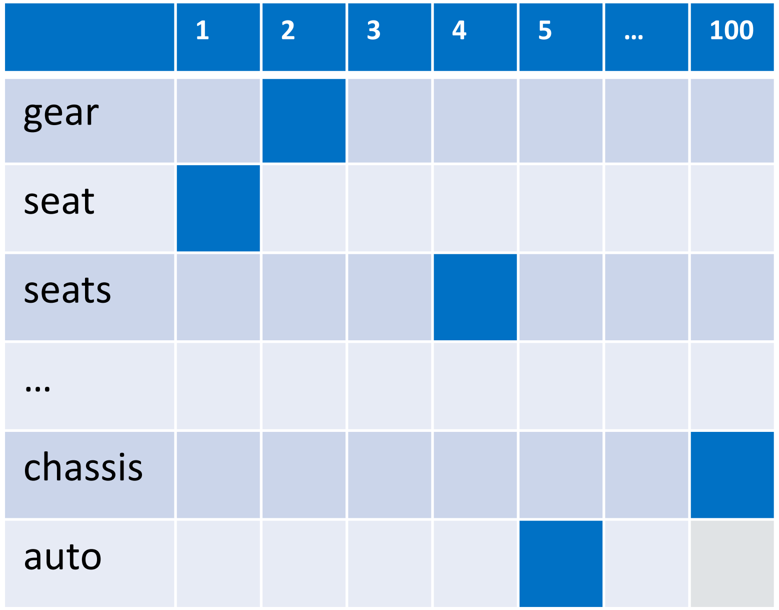
Simple Vector Representation of Words

# 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



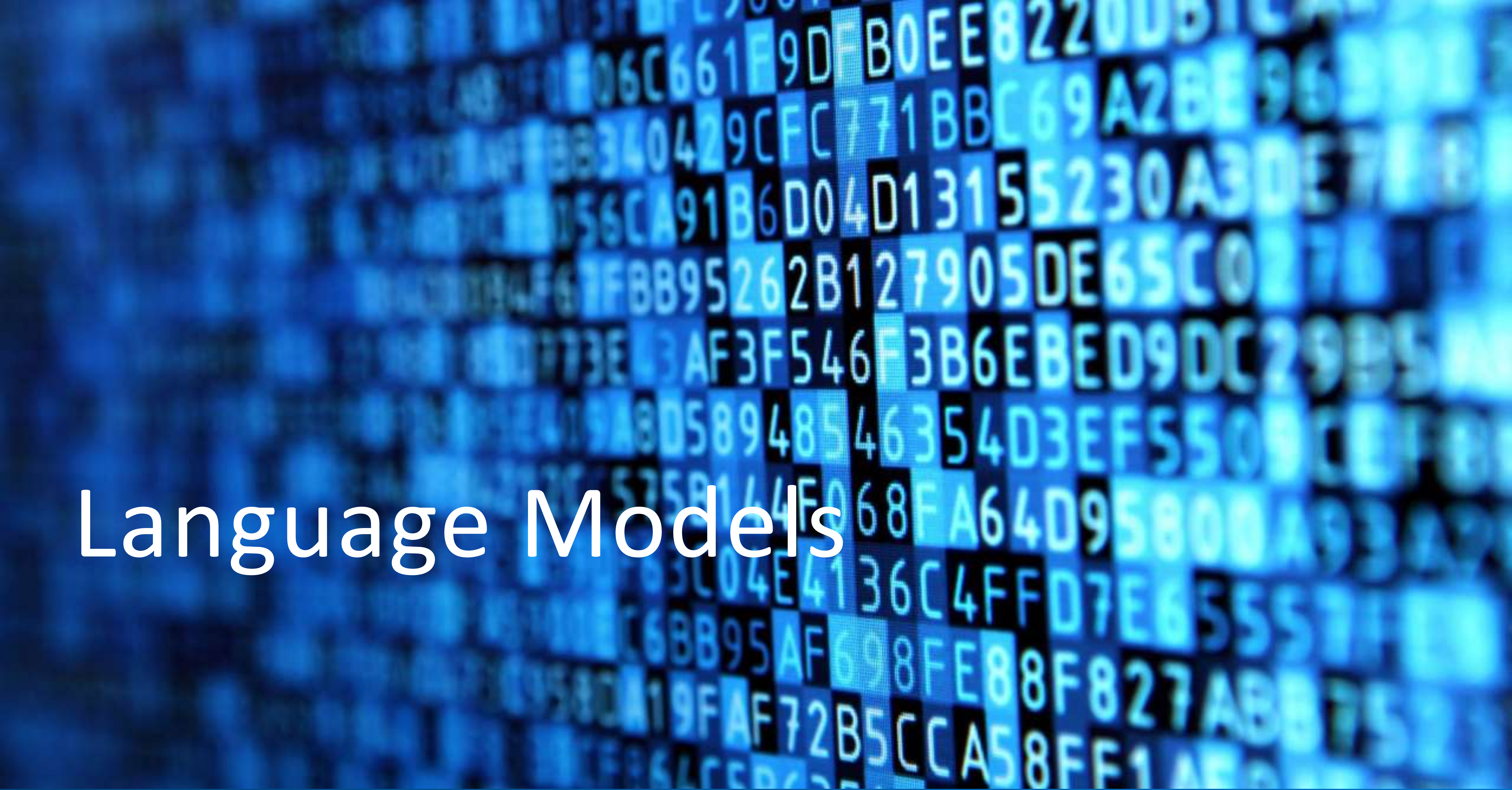

Challenges with this approach:

- 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`



# Language Models

# Neural Language Models

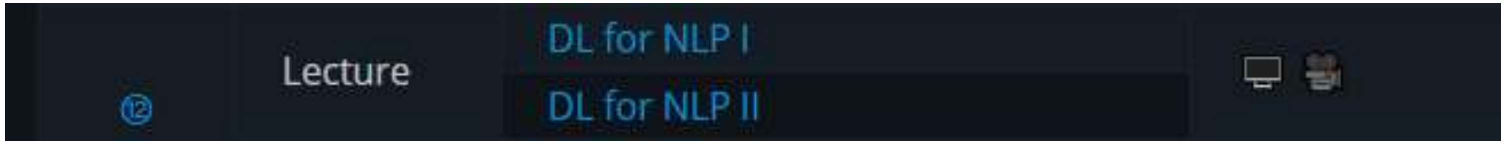
https://drive.google.com/file/d/149m3wRavTp4DQZ6RJTej8KP8gv4jnkPW/edit

Input Text Into Neural Network (somehow) -> NN maps all this context onto a vector -> this vector represents the next word -> get a big word embedding matrix which basically contains a vector for every possible word the model knows how to output.

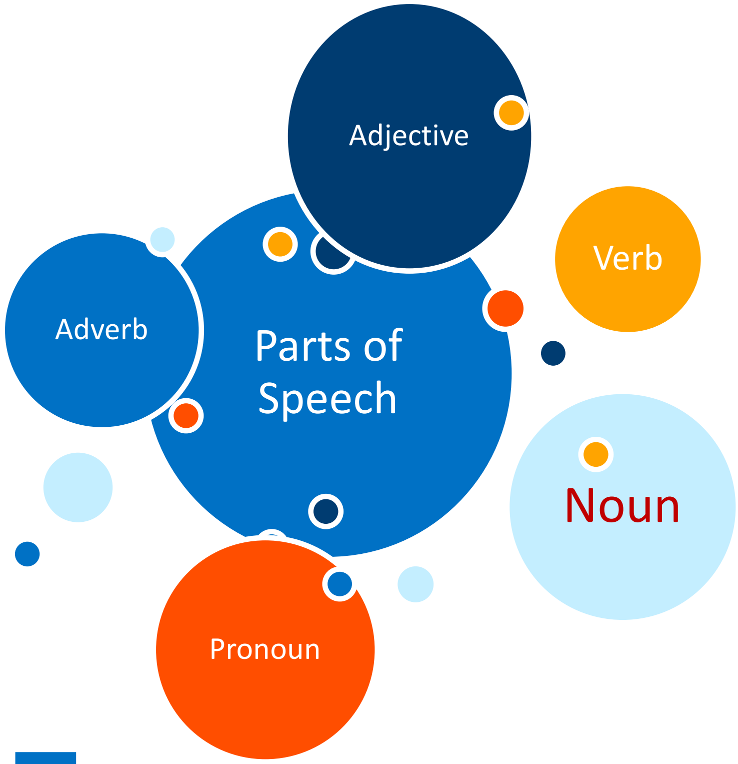
Then, all we need to do is compute the similarity by doing a dot product between the context vector and each of these word vectors and we'll get a likelihood of predicting the next word. Next, we train this model by maximum likelihood in the 'obvious way'.

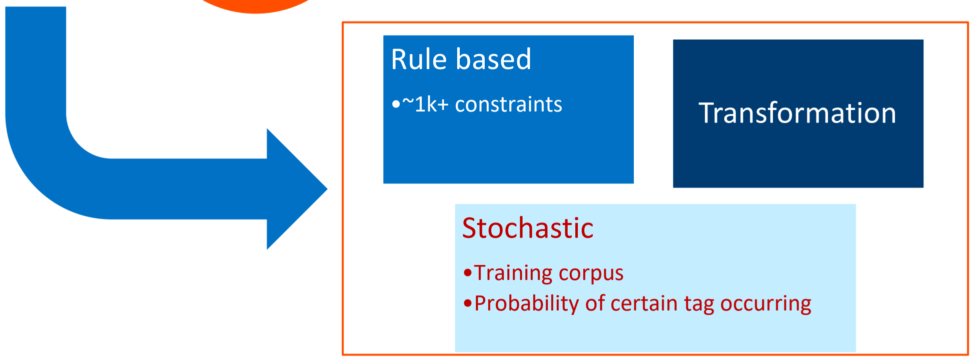
We often don't deal with words directly, we deal with sub-words or characters...All the skill is in building the encoder.

The first try was with convolutional models. Interpret a phrase the same way, regardless fo the order. Each word gets the same context vector.



# Parts of Speech Tagging



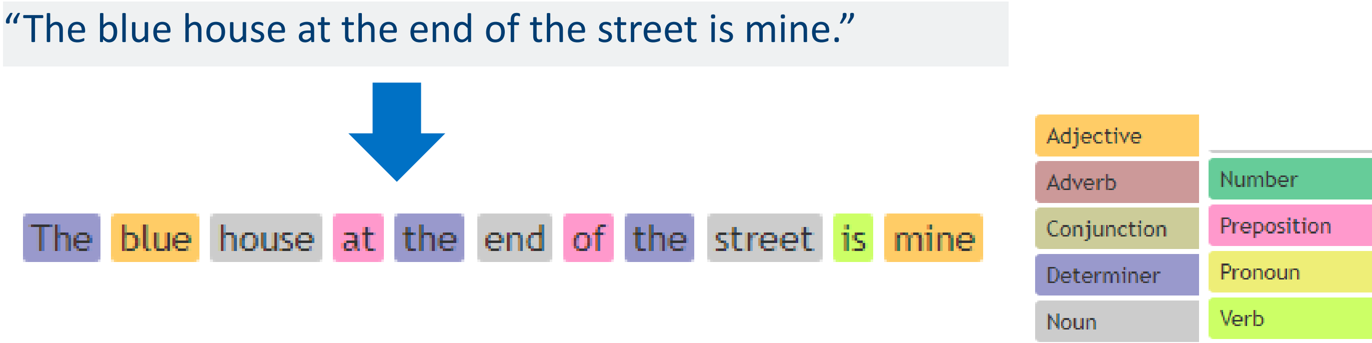

## One tag for each part of speech

- 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/



# 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

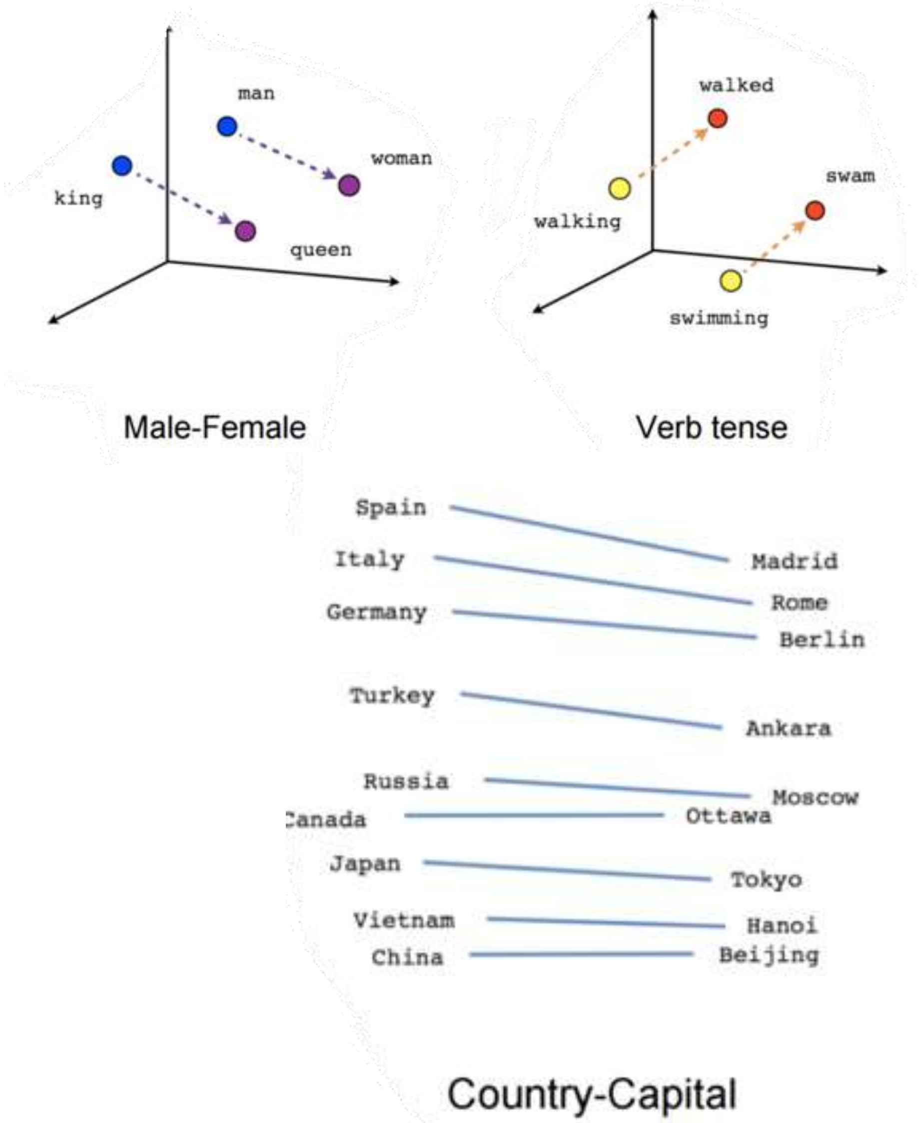
- Uses SVD decomposition and PCA to reduce dimensionality

- 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”)

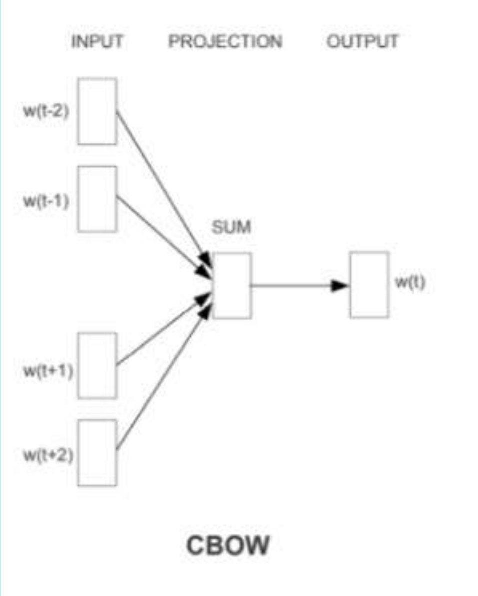



Reference: https://arxiv.org/abs/1301.3781
(Efficient Estimation of Word Representations in Vector Space)

# Word Embedding

Prediction based Word2Vec

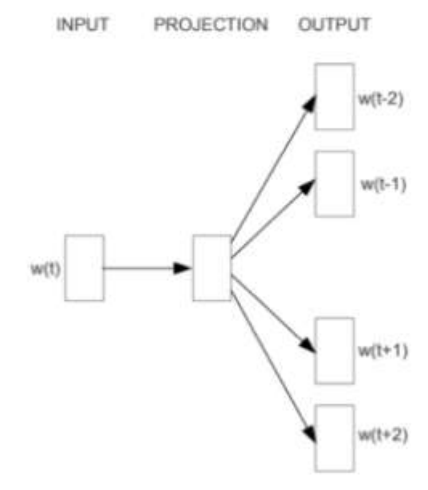
CBOW




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

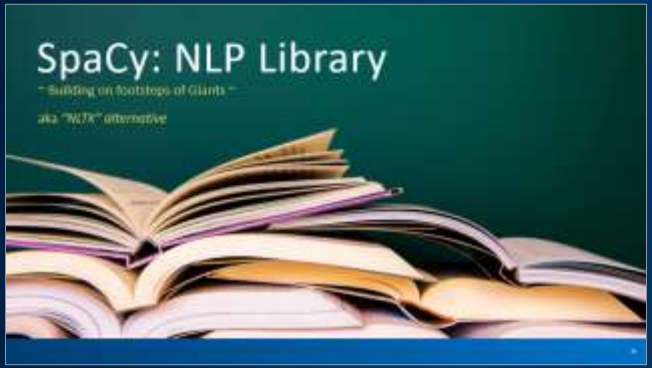
- The distributed representation of the surrounding words are combined to predict the word in the middle
- Input word is OHE vector of size V and hidden layer is of size N
- Pairs of context window & target window
- Using context window of 2, let's parse:
  - "The quick brown fox jumps over the lazy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

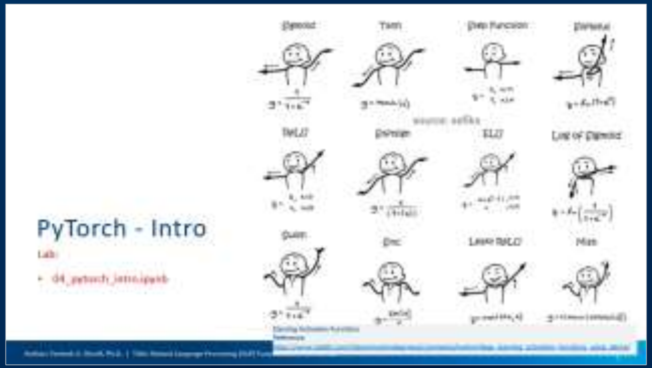
Skip-Gram




- 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])

# Part 2: Practicum



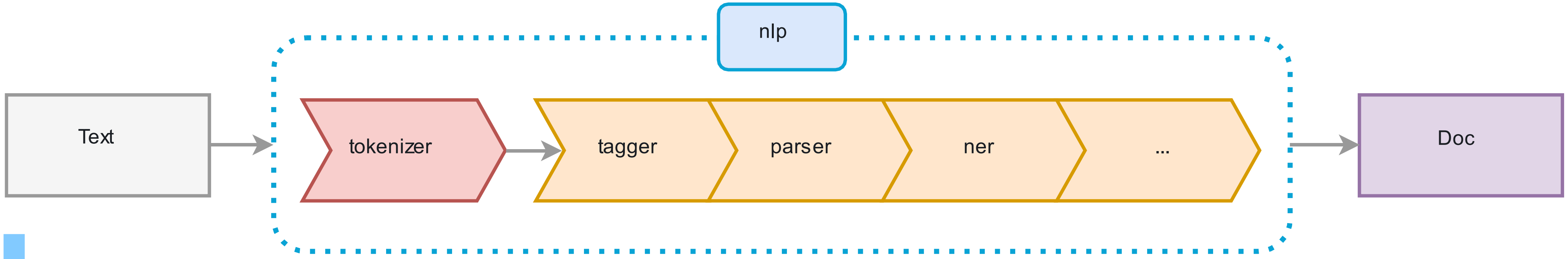


# SpaCy: NLP Library

~ 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](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

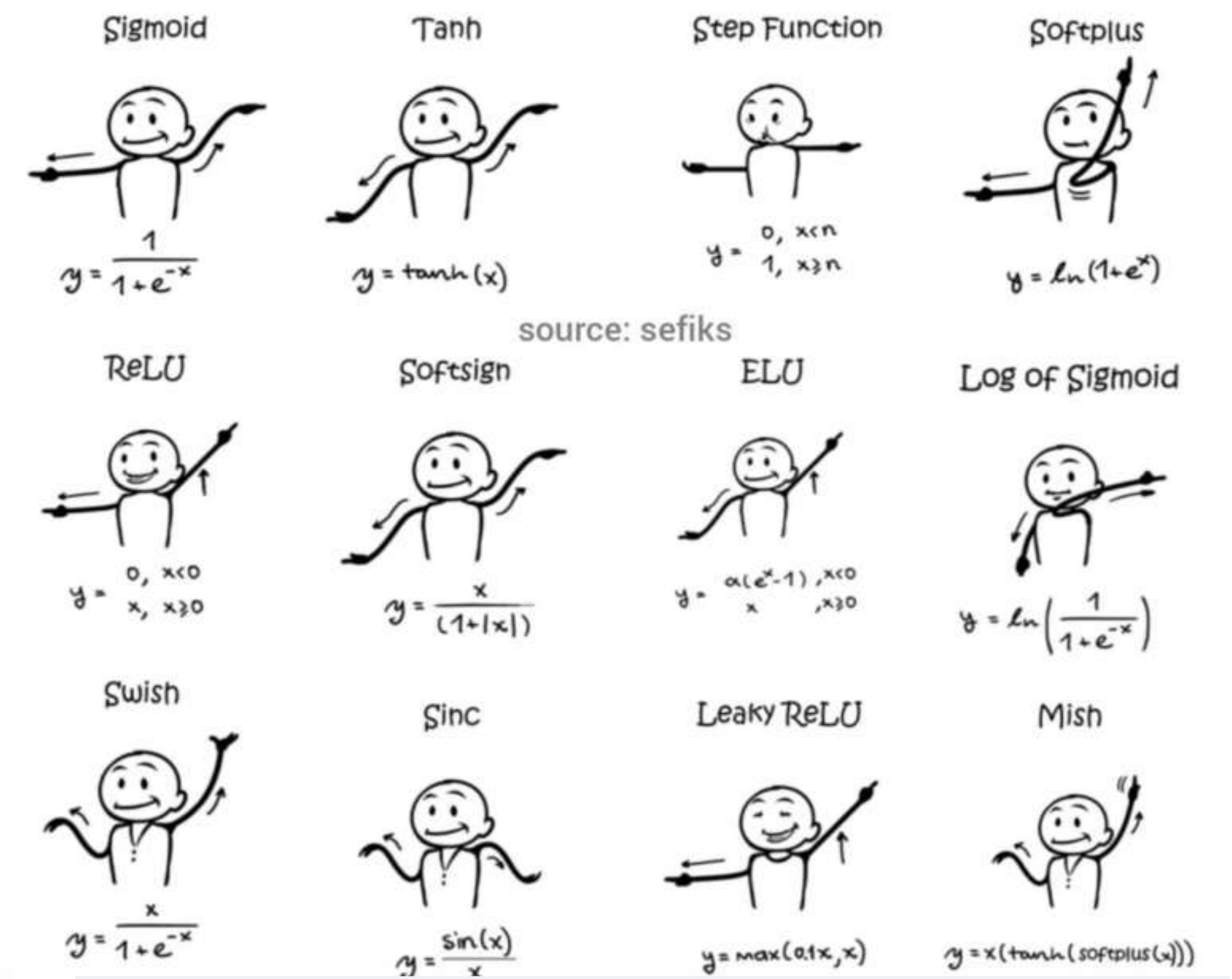
Lab:

- 03_SpaCy.ipynb

# PyTorch - Intro

Lab:

- 04_pytorch_intro.ipynb




Dancing Activation Functions
Reference:
https://www.reddit.com/r/learnmachinelearning/comments/lvehmi/deep_learning_activation_functions_using_dance/

# Deep Learning Frameworks

## Top Frameworks

- PyTorch ⇐ Facebook
- Tensorflow/Keras ⇐ Google
- MXNet ⇐ Amazon
- Caffe ⇐ BAIR (now part of PyTorch)
- PaddlePaddle ⇐ Baidu

## About PyTorch

- A deep learning framework originally built on Lua programming language and converted to Python
- Utilizes GPU as a replacement for Numpy (CPU)
- Imperative programming model (dynamic graph, generated at each step)
- Utilizes `tensor` as core data structure (similar to Numpy `ndarrays`)

# Fundamentals of PyTorch

- Imperative Programming ➔ Computations are performed on the fly. This means code debugging is easier
- Graphs are not compiled ➔ Neural network is generated at runtime. TensorFlow uses a static graph representation
- Tensors and Numpy Arrays occupy the **same** memory space. Zero cost of conversion
- Building a Neural Net
  - Forward pass
    - Activations $z = w * x + b$
    - Affine transformations $a = sigmoid(z), a = \tanh(z)\,, a = ReLU(z), \ldots$
  - Loss calculation
    - $loss = MSE(y_{pred}, y_{actual}), MAE(\ldots)$
  - Back Prop

# PyTorch Fundamentals

```
# 2D tensors
x = torch.tensor([[3.0, 8.0], [2.3, 1.4]])
print(m)

# 3D tensors
y = torch.tensor([[[3., 2.], [2., 1.]],
  [[2., 3.], [2., 0.]]])

print(x.shape)
print(y.shape)

# Indexing into the tensors
print(z[2])
print(z[1:3])

print(x[1][0]) # 2D
print(y[1][0][0]) # 3D
```

```
# Create a numpy array

x = np.array([[1, 2, 3], [3, 4, 5]])

Convert to torch tensor
y = torch.from_numpy(x)

# Convert torch to numpy
z = y.numpy()
```

```
t1 = torch.tensor([[1, 2, 3], [2, 3, 4]])
t2 = torch.tensor([[1, 2, 3], [2, 3, 4]])
print(t1 + t2) # normal addition works
print(torch.add(t1, t2)) # addition
print(torch.sub(t1, t2)) # subtraction
print(torch.mm(t1, t2)) # multiplication
print(t1/t2) # Division

a = torch.rand(3)
torch.sqrt(a)
tensor([nan, 1.02, 0.2, 0.33])
```

# PyTorch Modules

**Loading Dataset**

- **torch.utils.data.Dataset**
- **torch.utils.data.DataLoader**

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

**Defining the Neural Network**

- **torch.nn**
- **torch.optim** (update weight & biases)
- **torch.autograd** (backward pass to compute gradients)

torch.nn

- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- Non-linear Activations (other)
- Normalization Layers
- Recurrent Layers
- Transformer Layers
- Linear Layers
- Dropout Layers
- Sparse Layers
- Distance Functions
- Loss Functions
- Vision Layers
- DataParallel Layers (multi-GPU, distributed)
- Utilities
- Quantized Functions

**Saving & Conversion**

- **torch.save**
- **Convert to ONNX**

## Other modules

- torchtext
- torchvision
- torchaudio
- torchserve

**torchtext:** Primarily for NLP tasks. Contains several modules for text preprocessing for sentiment analysis, Question Answering, and others.

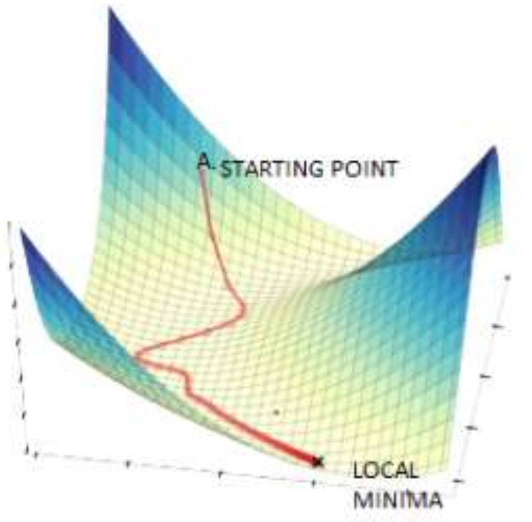
**torchvision:** Image data and image transformation library used for computer vision. Used for MNIST, COCO, CIFAR, and others.

**torchaudio:** Audio preprocessing and production deployment library with datasets of Cornell BirdCall Identification, UrbanSound8k, and others.

**torchserve:** Deploying model to production

https://pytorch.org/docs/stable/data.html
https://pytorch.org/docs/stable/nn.html

# PyTorch Training using Autograd




- Cost or loss function between actual or predicted values: MSE, Absolute error, etc.
- Given a function, $f(x_1, x_2, x_3)$, gradient is given by:
  - $\nabla f(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \dots) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \dots)$
- A gradient is a vector of partial derivatives. For a Neural Network with one neuron, this is:
  - $Gradient(\theta) = \nabla\theta(W_1, b_1) = \left(\frac{\partial\theta}{\partial W_1}, \frac{\partial\theta}{\partial b_1}\right)$
- With millions of neurons, this becomes:
  - $\nabla\theta\left(W_1, b_1, \dots W_{10{,}000}, b_{10{,}000}\right) = \left(\frac{\partial\theta}{\partial W_1}, \frac{\partial\theta}{\partial b_1}, \dots, \frac{\partial\theta}{\partial W_{10{,}000}}, \frac{\partial\theta}{\partial b_{10{,}000}}\right)$
  - PyTorch provides sophisticated methods for calculating & optimizing the loss function

Image credit: https://www.datasciencecentral.com/profiles/blogs/alternatives-to-the-gradient-descent-algorithm

# Calculating Gradients

$$\frac{\partial y}{\partial x} = \frac{(f(x + \partial x) - f(x))}{\partial x}$$

## Methods for calculating gradients

- Symbolic differentiation: conceptually simple, but hard to implement
- Numeric differentiation: Easy to implement but hard to scale
- Automatic differentiation: conceptually simple, but easy to implement

## Autograd is the PyTorch package to calculate gradient for model parameters

## Back propagation is implemented using a technique called reverse auto differentiation

- Weight parameters at time t+1 are calculated based on prior time-step weights minus the learning rate time the gradient at time *t.*
  - $W^{t+1} = W^t - \eta \times Gradient(\theta)^t$
  - This moves each parameter value in the direction of reducing gradient
- Every optimization algorithm implements weight update differently
  - PyTorch provides different options & you can write yours as well!

# Symbolic differentiation of the loss Function

$$\frac{d}{dw}\left[(xw-y)^2\right]$$
$$= 2(xw-y)\cdot\frac{d}{dw}[xw-y]$$
$$= 2(xw-y)\left(x\cdot\frac{d}{dw}[w]+\frac{d}{dw}[-y]\right)$$
$$= 2(xw-y)(x\cdot 1+0)$$
$$= 2x(xw-y)$$

https://www.derivative-calculator.net/

## Find w that minimizes the loss

$$loss(w) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}-y)^2$$

$$\frac{\partial loss}{\partial w} = \frac{\partial (xw-y)^2}{\partial w}$$

$$\frac{\partial loss}{\partial w} = 2x * (wx-y)$$

## PyTorch:

$$\frac{argmin}{w} \; loss(w)$$

$$w_{t+1} = w_t - \eta\frac{\partial loss}{\partial w}$$

$$w_{t+1} = w_t - \eta * 2x\,(xw-y)$$

labs/04c_pytorch_symbolic_loss.ipynb

# Reverse mode autodifferentiation

Forward pass to calculate the loss (y_pred – y_actual)

Reverse pass to update the parameter values (weights)

# Implementing the symbolic differentiation

Symbolic Differentiation Lab: 04c_pytorch_symbolic_loss.ipynb

# Attention

An attention unit takes all sub-regions and their context as input and outputs a weighted average of the regions, based on probabilities. Context is everything in this case

Context, C, comes from RNN and input regions Y come from the Conv NN.

# Using `torchtext.<>` API

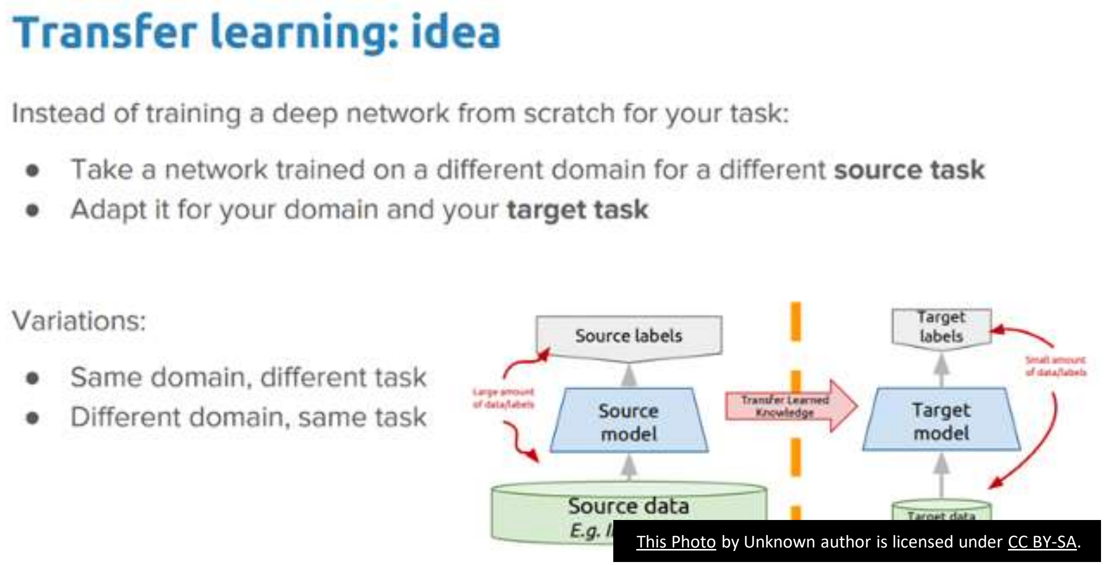
| .data | .datasets | .vocab |
|---|---|---|
| • Fields<br>• Iterators<br>• Pipelines | • Sentiment analysis<br>• Sequence tagging<br>• Question classification | GLoVe<br>CharNGram |

* https://torchtext.readthedocs.io/en/latest/data.html

# Model Pre-training

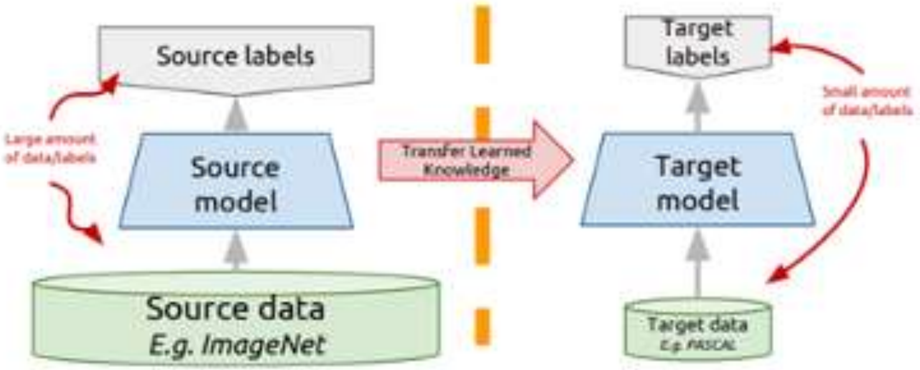
From ULMFiT to Transformers

# Transfer Learning






# Transfer learning concept




## Conventional approach:

- Training a deep learning model from scratch using your data
- Challenge: high compute, high data requirements)

## Transfer learning approach:

- Start with a network trained on a different domain and source task
- Adapt it for your domain and target task (smaller cClick to add text ts)
  - Can also apply for the same domain but different task or
    - For example: Imagenet dataset trained computer vision model applied to transfer learning for detecting species of butterflies or types of leaves
  - Different domain and same task
    - For example: Image segmentation in self driving for detecting pedestrians applied to image segmentation task in healthcare for detecting tumor

# Transfer Learning: History & State of the Art

Pre-2018:

2018: Jeremy Howard released ULMFiT – an approach to solve NLP problems, took away 90% of the developer pain in running new models

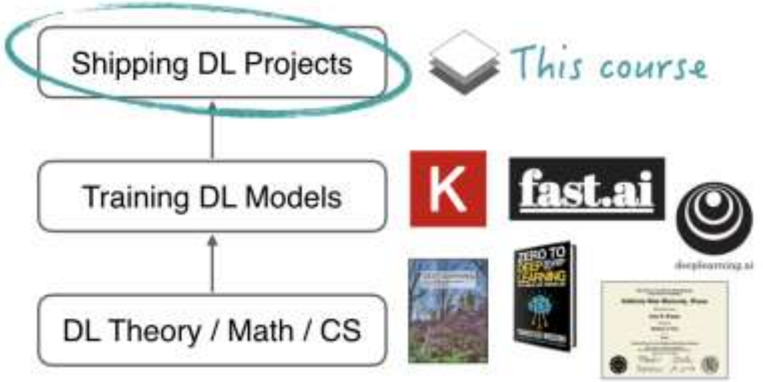
- AWD-LSTM neural network pre-trained on Wikitext-103

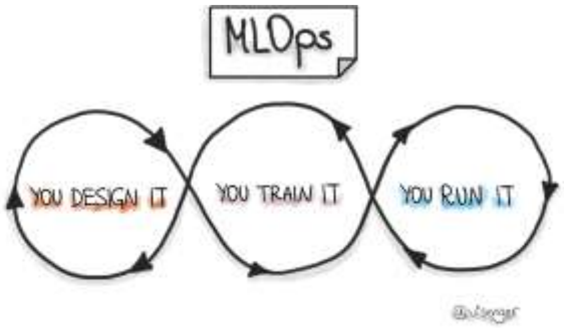
Post 2018: Attention is all you Need became popular, proposing the 'Transformer' architecture. Huggingface has a "Transformer" library that was used as the architecture for BERT, Transformer-XL, XLNet and (facebook) RoBERTa and XLM and OpenAI (GPT, GPT-2)

# Operationalizing Machine Learning Pipelines

# Resources for ML in Production

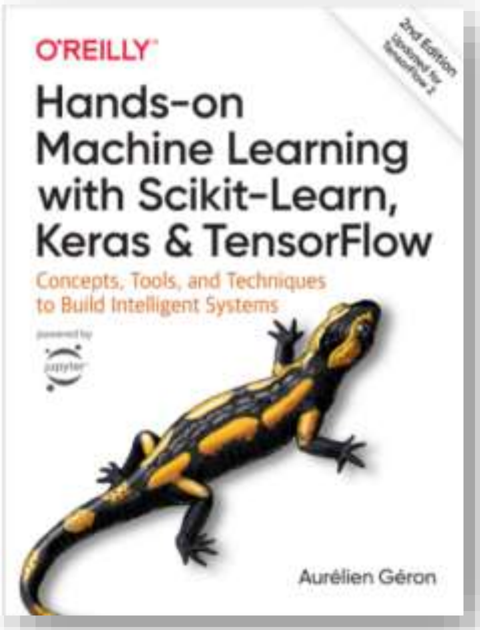
1. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition**
2. **Dive into Deep Learning (https://d2l.ai/:** *Aston Zhang, Zack C. Lipton, Mu Li, and Alex J. Smola*
3. **Full Stack Deep Learning (https://course.fullstackdeeplearning.com/)**
4. **Designing Data-Intensive Applications (***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***)**

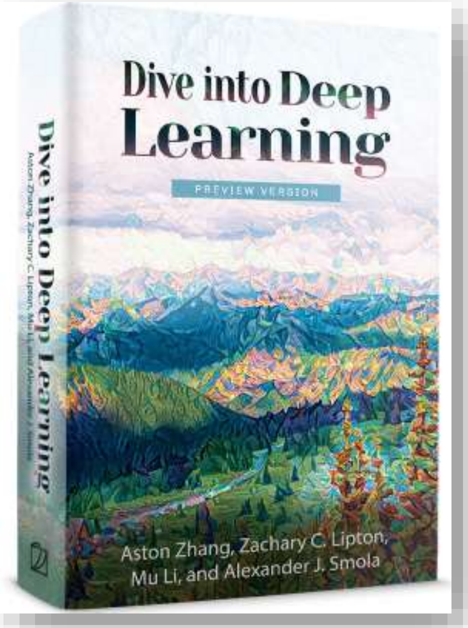


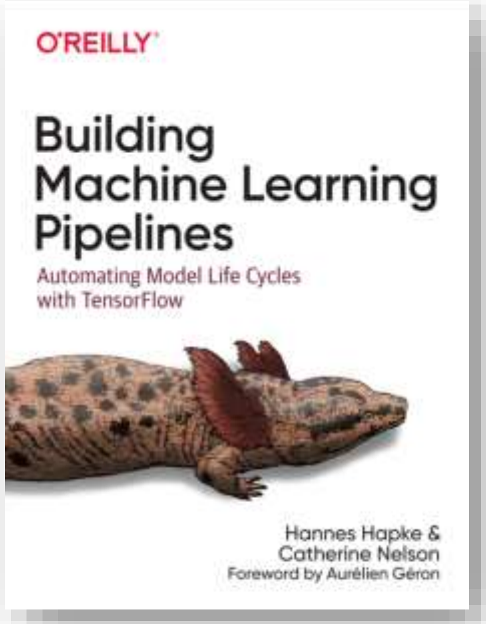


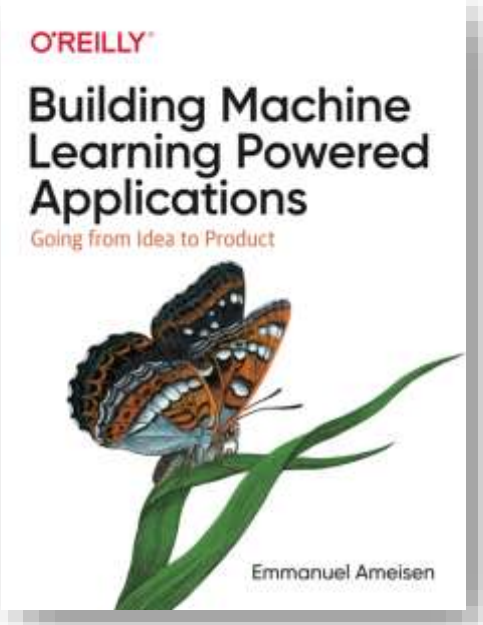
Credit: https://elvissaravia.substack.com/p/my-recommendations-to-learn-machine

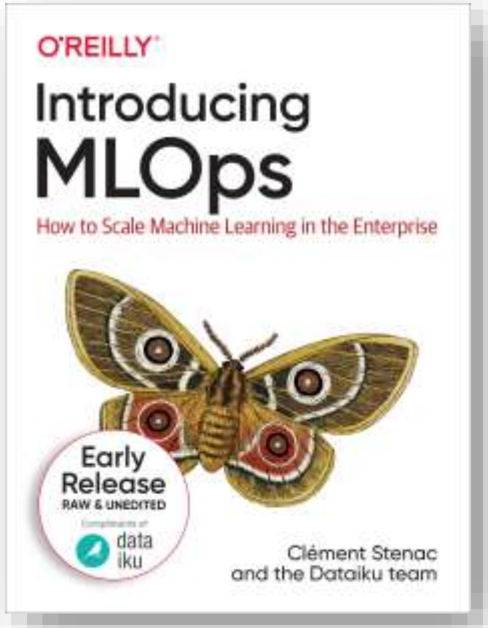

O'REILLY
2nd Edition
Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
Aurélien Géron

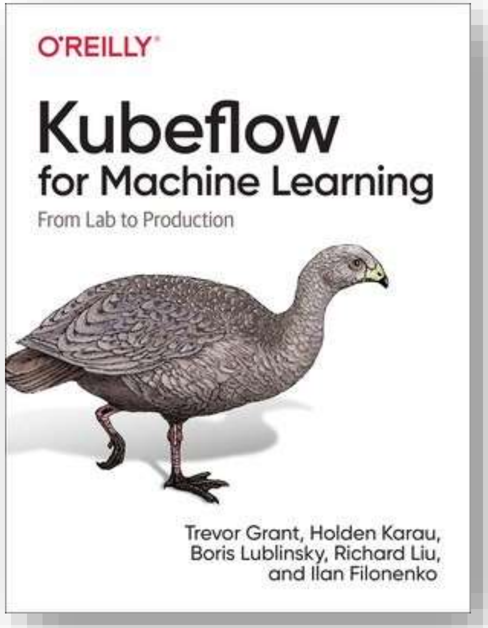

Dive into Deep Learning
PREVIEW VERSION
Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola


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

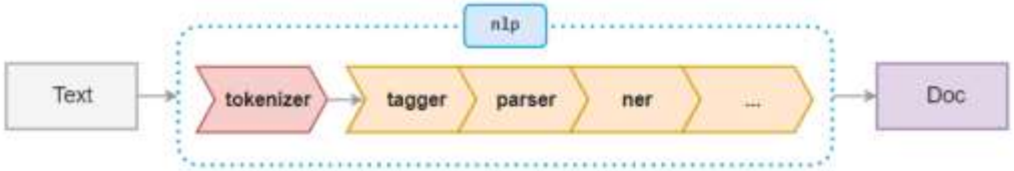

O'REILLY
Building Machine Learning Powered Applications
Going from Idea to Product
Emmanuel Ameisen


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

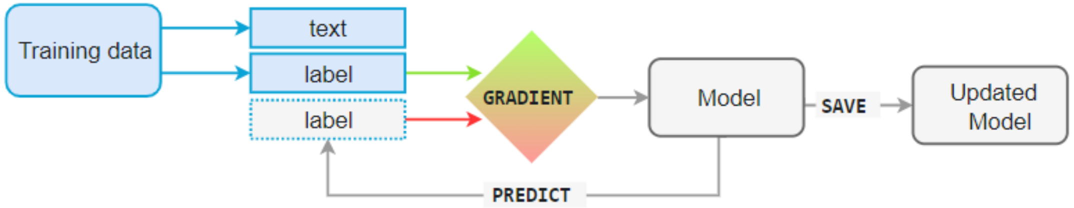
# Language Processing Pipelines

- 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…

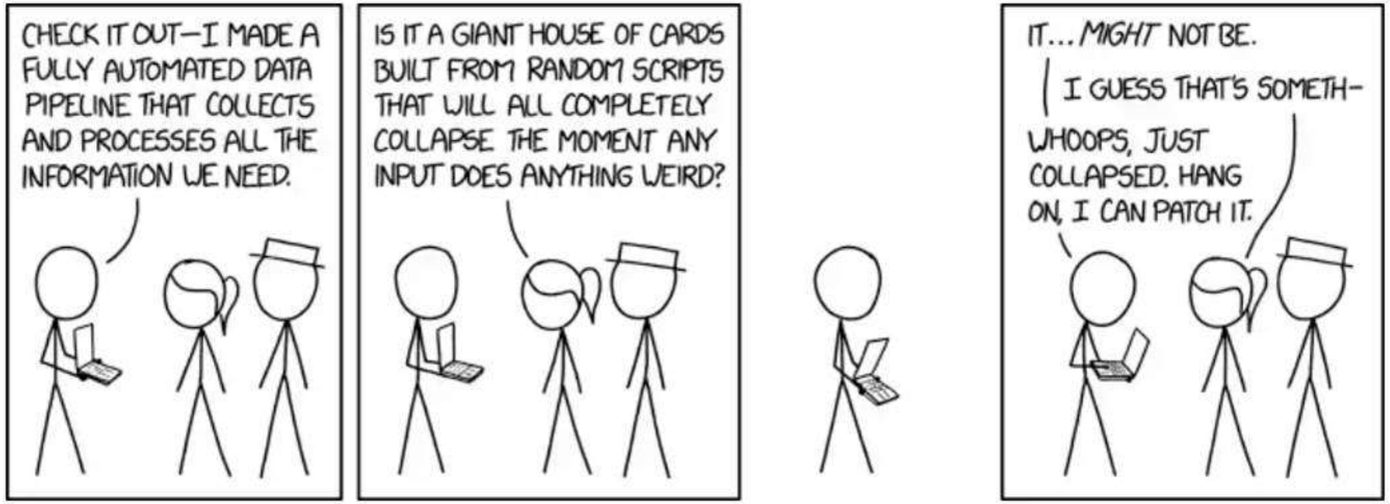



Image source: [xkcd: Data Pipeline](https://xkcd.com/2054/)

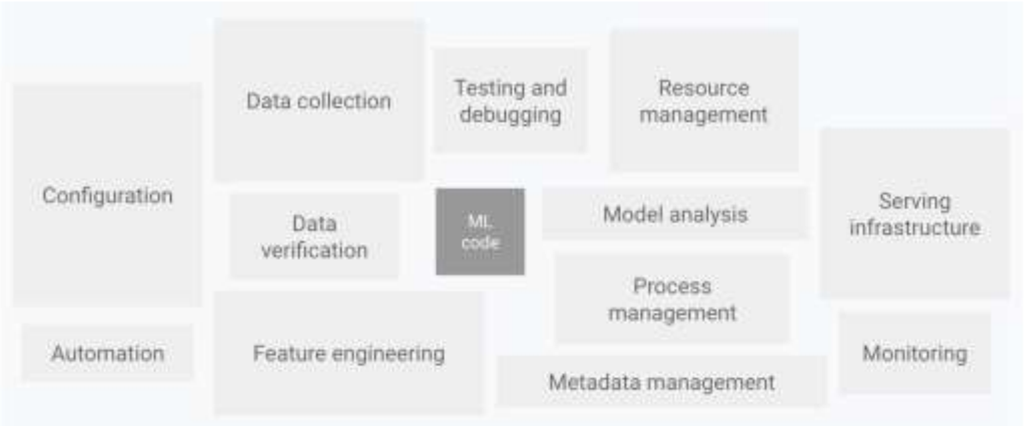
# Creating NLP pipelines

## 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

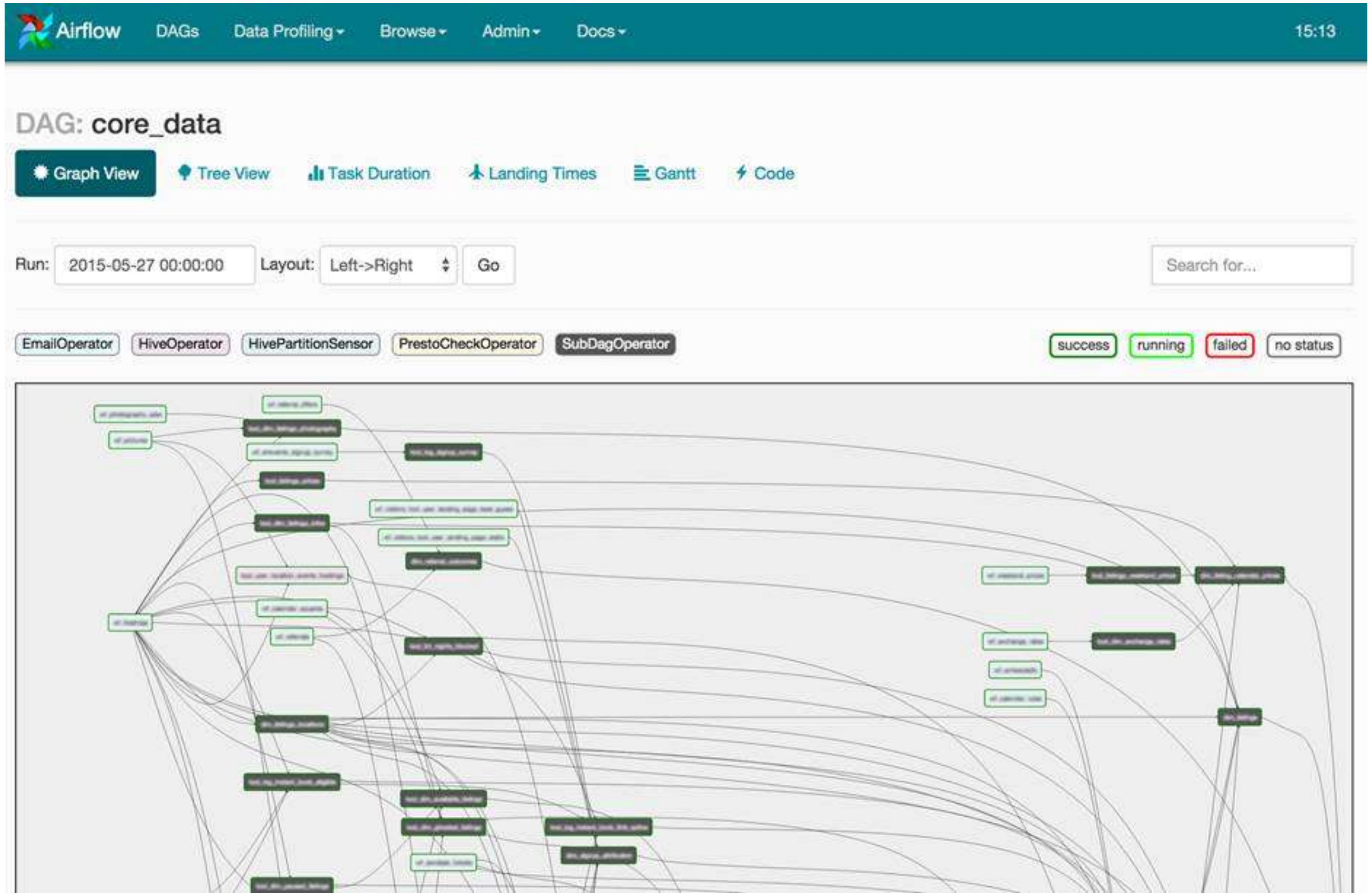
```
# Python standard modules
from datetime import datetime, timedelta
# Airflow modules
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
default_args = {
      'owner': 'airflow',
      'depends_on_past': False,
      # Start on 27th of June, 2020
      'start_date': 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/