Introduction to Natural Language Processing with Python spaCy

Daniel Kapitan | version February 2021

Jheronimus Academy of Data Science

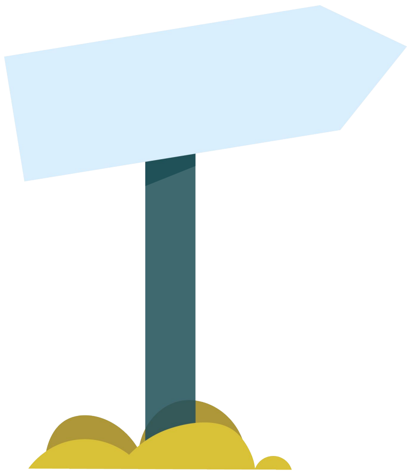
Attribution and copyright notice

This lecture is based on the following material:

- Artificial Intelligence: A Modern Approach (Fourth edition) by Stuart Russell and Peter Norvig
  - chapter 23: Natural Language Processing
  - chapter 24: Deep Learning For Natural Language Processing
- Text Mining with R, by Julia Silge and David Robinson
- Jay Alammar's excellent visual explanations of machine learning concepts
- Advanced NLP with spaCy
- pyLDAvis
- Several open access journal papers (referenced individually)

Learning objectives

- Understand the basic concepts of **probabilistic language models** and how these can be applied to different NLP tasks including sentiment analysis, genre classification and named-entity recognition
- Understand and know how to apply **n-grams** and **word embeddings** for feature extraction in a classification pipeline
- Have a conceptual understanding of **transformers** and **deep learning techniques** for NLP
- Understand and know how to use Python spaCy for NLP and develop reproducible text processing pipelines



Natural Language Processing (NLP)

How to acquire knowledge that is expressed in natural language?

Symbolic NLP

Given a collection of rules (e.g., a Chinese phrasebook, with questions and matching answers), the computer emulates natural language understanding (or other NLP tasks) by applying those rules to the data it is confronted with.

1950 – **1990**

Statistical NLP

Application of machine learning techniques to NLP.
Focus of this lecture.

1990 – **2010**

Neural NLP

Extension of statistical methods with representation learning and application of deep neural networks, including transformers

2010 – **present**

[Wikipedia: natural language processing]

Some common natural language processing tasks

(Many articles on Wikipedia are a good starting point)

Text classification	Information retrieval	Information extraction
spam filtering	recommender systems	Template-filling
topic modeling	search engine	named entitiy recognition (NER)
sentiment analysis	question answering	relationship extraction
	Summarization	ontology extraction

Challenges of probabilistic language models

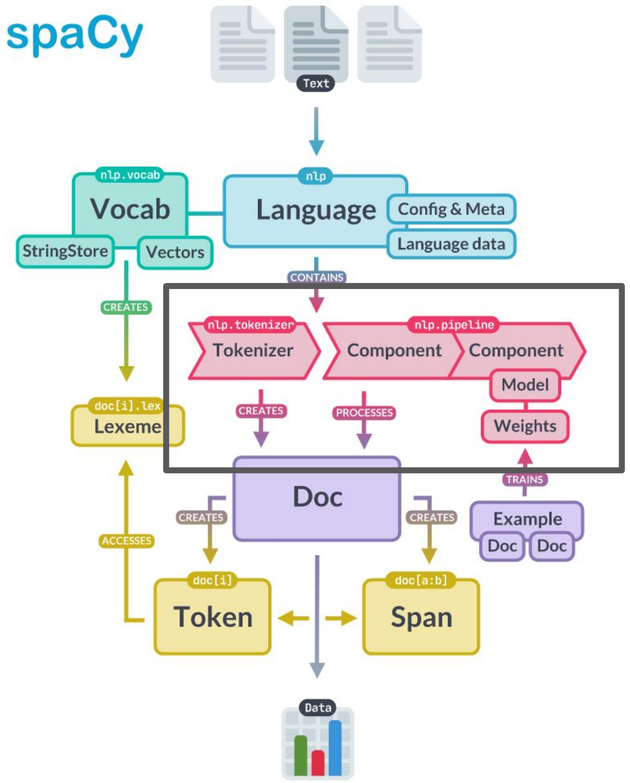
Challenges due to complexity of real natural language	Approach
Infinite amount of expressions	→ probabilities of a random word or sequence
Ambiguity	→ probabilities of meaning (see next slide)
Volume and velocity	→ approximations, for example assigning small probability for all out-of-vocabulary words

Ambiguity

Lexical	single word has different meanings (homonyms, test by antonyms): “rob the bank” or “walk along the bank”
Syntactic	how to interpret clauses in a sentence: “Lindsey told Jessica that she had cancer” “helicopter powered by human flies”
Semantic	occurs when a word, phrase or sentence, taken out of context, has more than one interpretation: “We saw her duck” “Milk drinkers are turning to powder”
Metonymy	figure of speech in which a thing or concept is referred to by the name of something closely associated with that thing or concept: “Chrysler announced a new model” (we know companies can’t talk)

Wikipedia: Ambiguity

Using spaCy as our practical guide to NLP



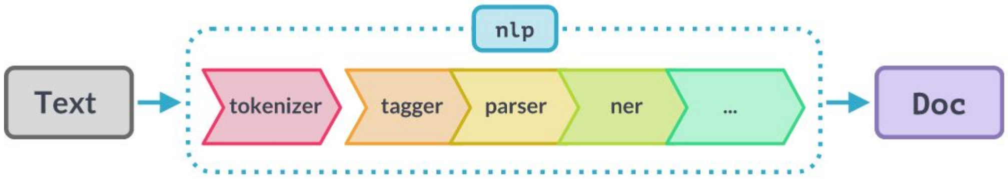
1. Reproducible NLP pipelines

2. Use different language models for **word embeddings** in your pipeline
   a. n-grams & TF-IDF
   b. word2vec, GloVe & fasttext
   c. transformers & BERT

[spaCy documentation: architecture]

1. Reproducible NLP pipelines

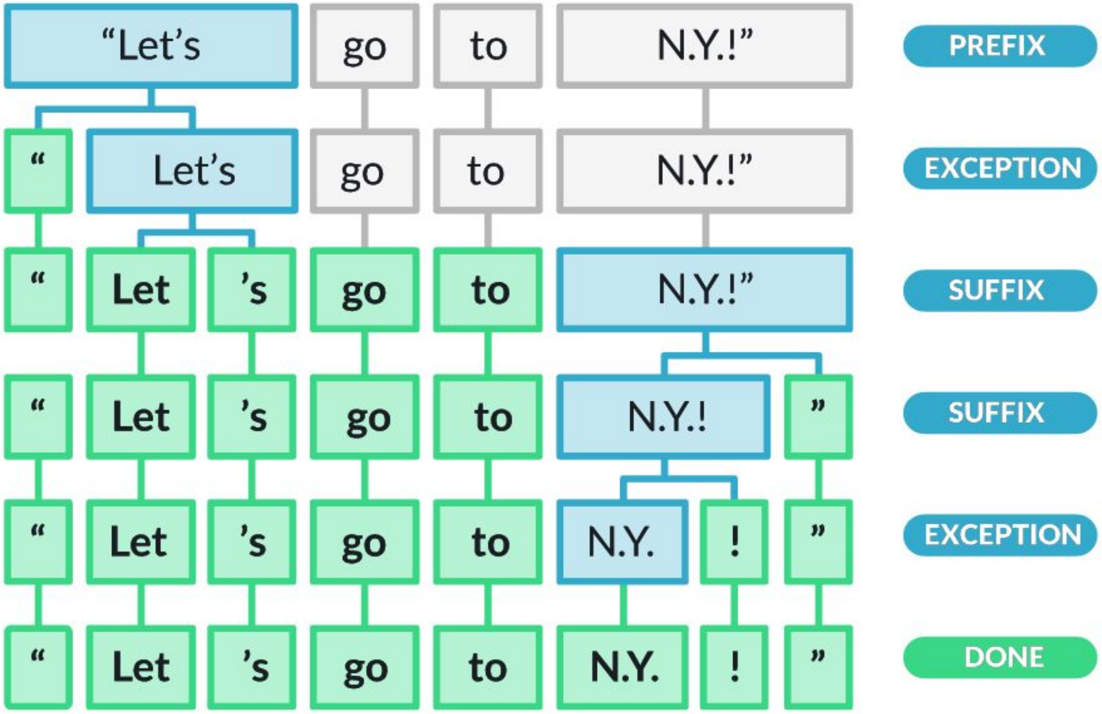
1. Reproducible NLP pipelines



Component	Creates	Description
Tokenizer	Doc	Segment text into tokens
Tagger	Token.tag	Assign part of speech tag
DependencyParser	Token.head, Token.dep, Doc.sents, Doc.noun_chunks	Assign dependency labels
EntityRecognizer	Doc.ents, Token.ent_iob, Token.ent_type	Detect and label named entities
Lemmatizer	Token.lemma	Assign base forms
TextCategorizer	Doc.cats	Assign document labels

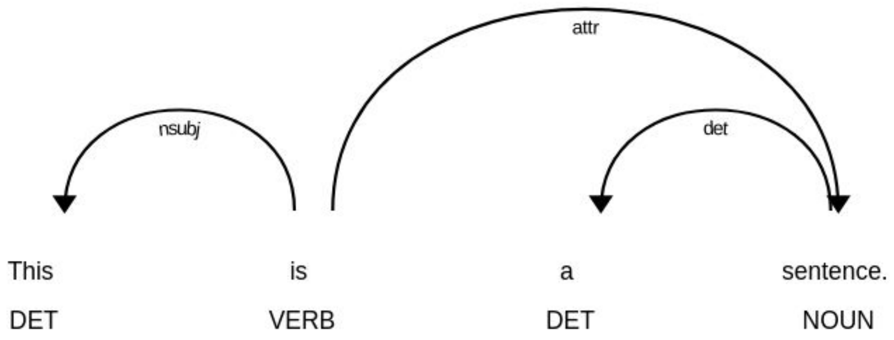
spaCy documentation: pipelines

Tokenization



[spaCy documentation: tokenization]

Tagger



Open class words	Closed class words	Other
ADJ: adjective	ADP: adposition	PUNCT: punctuation
ADV: adverb	AUX: auxilary verb	SYM: symbol
INTJ: interjection	CONJ: coordinating conjunction	X: other
NOUN	DET: determiner	
PROPN: proper noun	NUM: numeral	
VERB	PART: particle	
	PRON: pronoun	
	SCONJ: subordinating conjunction	

- spaCy documentation: part-of-speech tagging
- Universal POS tags

Named entity recognition

Named entity

A named entity is a "real-world object" that's assigned a name – for example, a person, a country, a product or a book title. spaCy can recognize various types of named entities in a document, by asking the model for a prediction.

[spaCy documentation: named entities]

Stemming and lemmatization

Stemming

Stemming is the process of **reducing inflection in words** to their root forms such as mapping a group of words to the same stem **even if the stem itself is not a valid word in the language**.

Lemmatization

Lemmatization, unlike stemming, **reduces the inflected words properly ensuring that the root word belongs to the language**. In lemmatization root word is called lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.

2a. n-grams & TF-IDF

n-grams are sequences of n words

Original text	**To Sherlock Holmes she is always the woman. I have seldom heard him mention her under any other name.**
Unigrams (1-grams)	{to, sherlock, holmes, she, is, always, the, woman, I , have …}
Bigrams (2-grams)	{to sherlock, sherlock holmes, holmes she, she is, is always …}
Trigrams (3-grams)	{to sherlock holmes, sherlock holmes she, holmes she is, …}
Quadgrams (4-grams)	{to sherlock holmes she, sherlock holmes she is, holmes she is always …}

bag-of-words is basically counting unigrams

unigrams and bag of words

[Force of LSTM and GRU – Blog]

document term matrix

term	doc_1	doc_2	…
it	6	…	…
I	5	…	…
the	4	…	…
to	3	…	…
and	3	…	…
seen	2	…	…
…	…	…	…

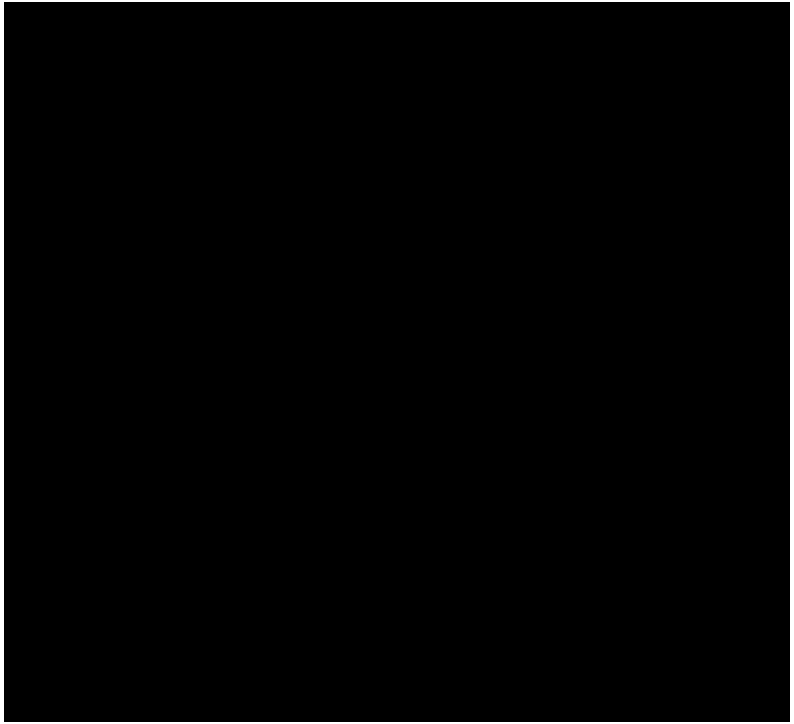
Why do n-grams models work at all?

Markov assumption

$$P\left(\frac{I\ believe\ coding\ in\ Python\ is\ fun}{I\ believe\ coding\ in\ Python\ is}\right)$$

$$\approx P\left(\frac{Python\ is\ fun}{Python\ is}\right)$$

The effectiveness of n-grams: topic modeling



[Wikipedia: topic model]

The amazing effectiveness of character n-grams

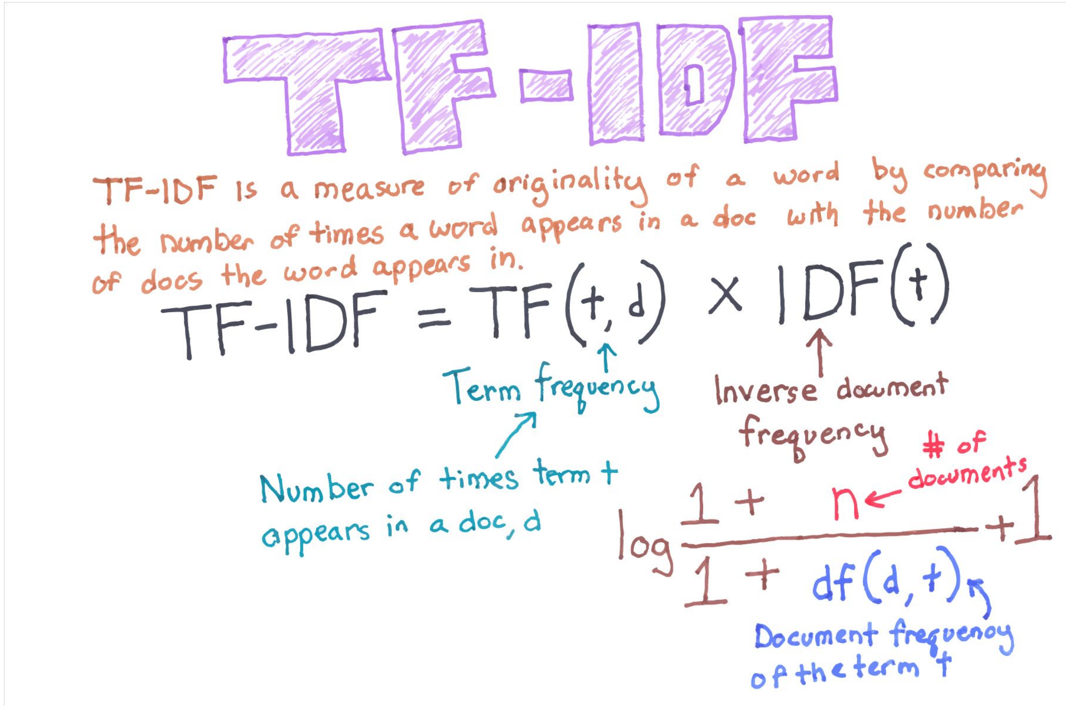
Language detection

- Character quadgrams can detect language with >99% accuracy
- Implemented in e.g. cld2 library
- See exercise: practice with the Universal Declaration of Human Rights

Spam detection

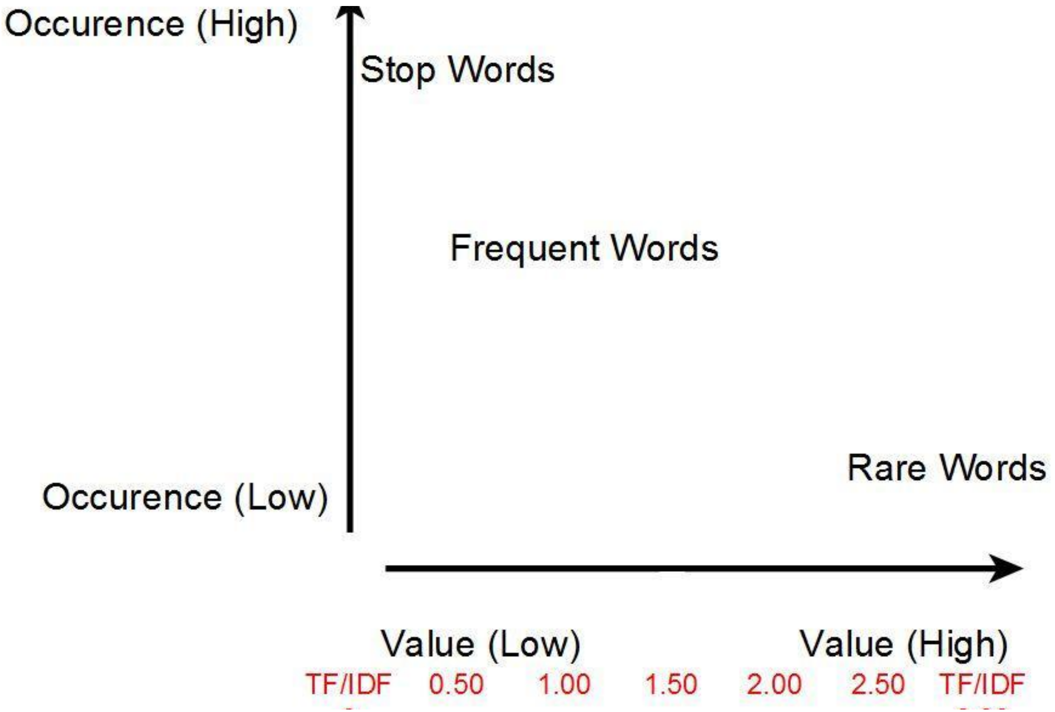
- Character quadgrams and pentagrams can detect spam with up to 95% precision and recall
- Algorithm is language *independent*
- See article Spam Detection Using Character N-Grams

More refined counting of n-grams: TF-IDF



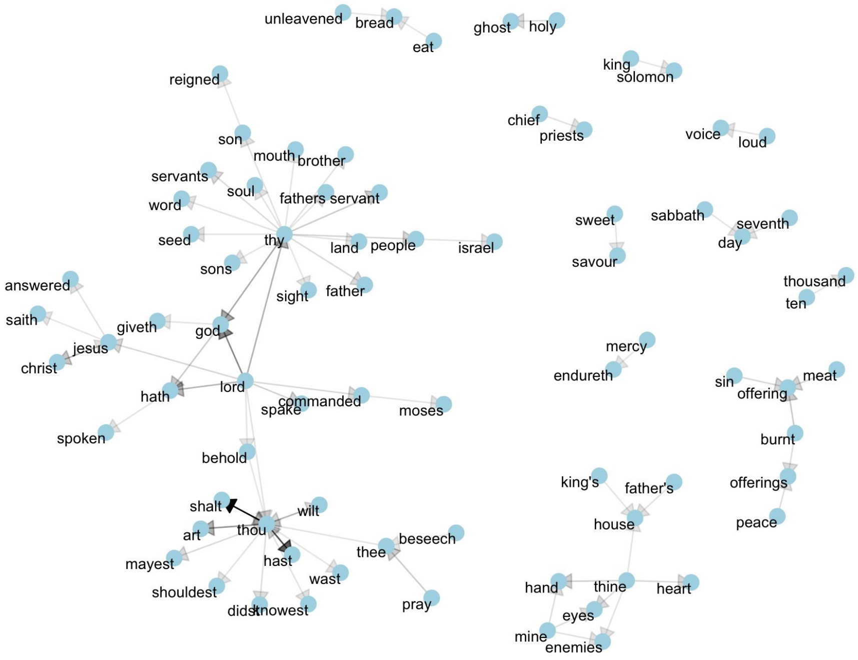
Creating TF-IDF from scratch in Python

More refined counting of n-grams: TF-IDF



For more details, see [Wikipedia: TF-IDF]

Visualizing n-grams



[Tidy Textmining 4.1.5]

Using n-grams & TF-IDF in practce

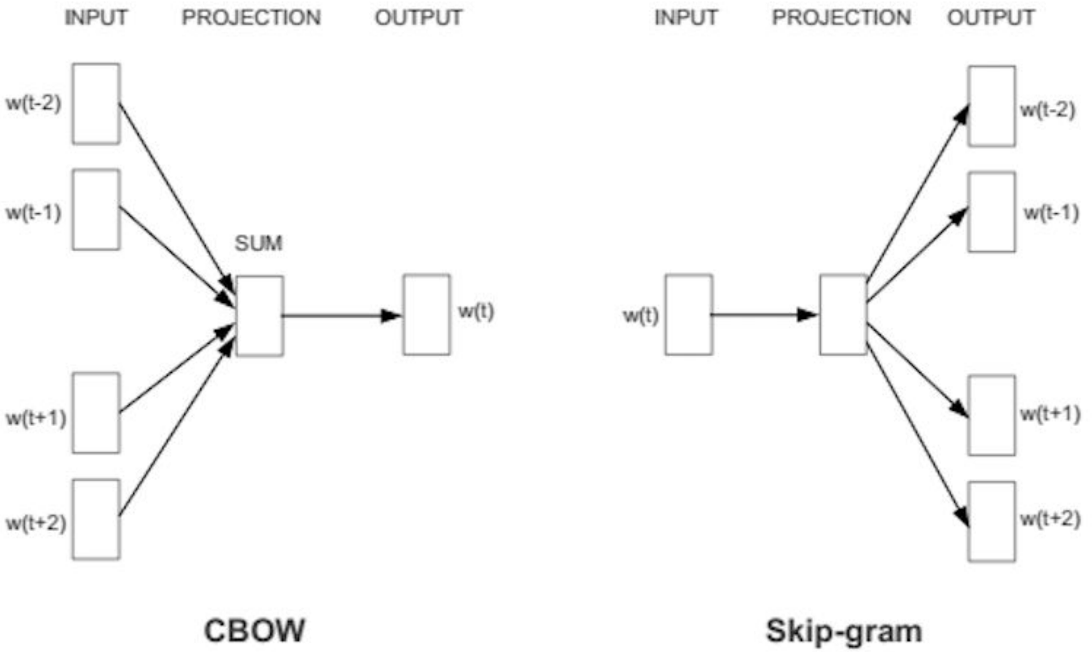
Advantages	Disadvantages
Easy to integrate in pipeline	Document term matrix is very sparse and can get large for n > 1 (using lemmatization helps)
Easy to customize parsing for domain specific texts	No contextual information in available in model
Bi-/tri-/quad grams are surprisingly effective	Can’t deal with out-of-vocabulary words (using lemmatization helps)

2b. word2vec, GloVe & fasttext

Word embeddings

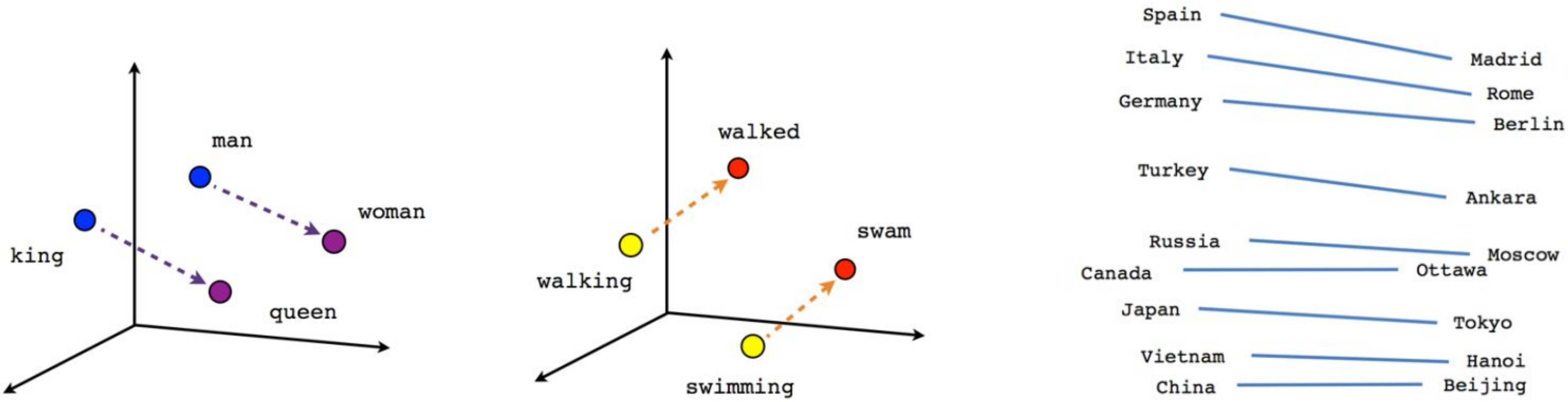
- How to create vector representations of words without the sparsity of n-grams?
- First solution: word2vec
  - Developed by four Google engineer (now at Facebook) in 2013 (original paper on Arxiv)
  - Has lead to significant breakthroughs in deep learning e.g. for translations
  - Google's pre-trained word2vec model:
    - 300-dimensional vector space
    - with 10^{11} words
    - trained on $3\text{x}10^{6}$ sentences
- Many variations since then (see overview article or this blogpost)
  - GloVe (2014): looks at global word co-occurence
  - FastText (2017): also looks at sub-word

word2vec: shallow neural network to predict neighbouring words



word2vec explained by Julian Gilyadov

The surprising properties of word embeddings



Male-Female

Verb tense

Country-Capital

Using word2vec in practice

Advantages	Disadvantages
Many pre-trained models available	Can’t deal with multiple meanings of the same word
Easy to integrate in pipeline	Extra care is needed when meanings of words change over time or when using in specific domain
	Can’t deal with out-of-vocabulary words

2c. transformers and BERT

A BERT's eye view of state-of-the-art NLP models

Bengio *et al.* (2003):
Proposed use neural networks for language models for i) distributed representation of words; and ii) probability function of word sequences

Mikolov *et al.* (2010):
demonstrated RNN

Mikolov *et al.* (2013):
word2vec

Sutskever *et al* (2014).:
sequence2sequence

Pennington *et al.* (2014): *GloVe*

Howard and Ruder (2018): *ULMFiT*
Easier to fine-tune pretrained language model without need for vast corpus with Universal Language Model Fine-Tuning

Peter *et al.* (2018): *ELMo*
Contextual word embeddings

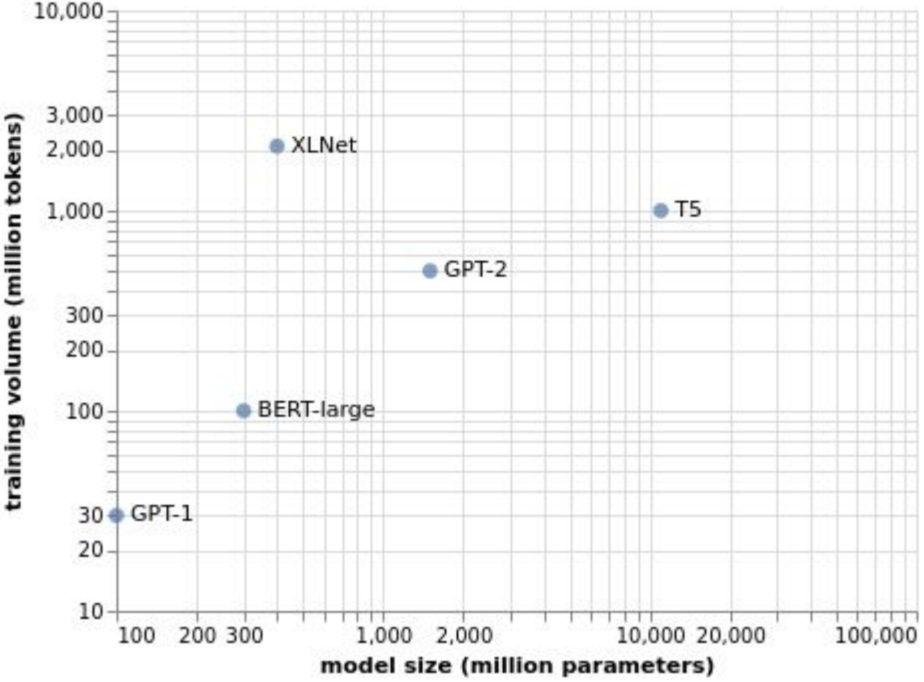
Devlin *et al.* (2018): *Bert*
Bidirectional Encoder Representations from Transformers

Radford *et al.* (2018):
GPT(-1)

Radford *et al.* (2019):
GPT-2

Radford *et al.* (2020):
GPT-3

A BERT's eye view of state-of-the-art NLP models

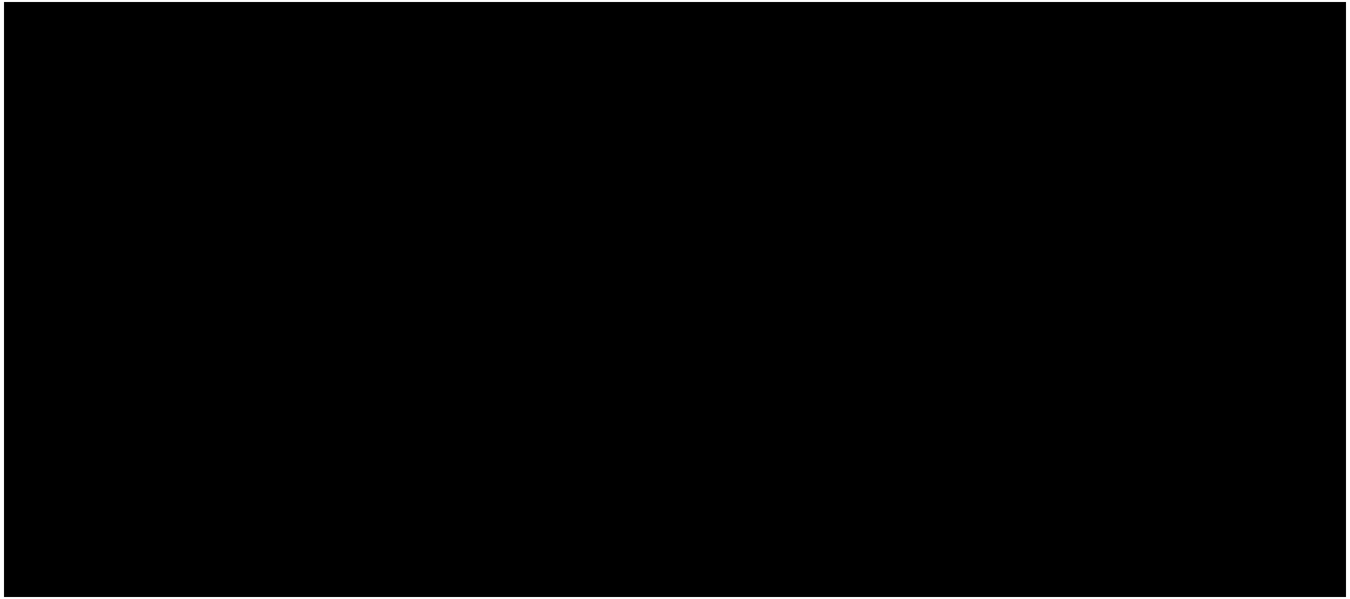


[Sharir et al. (2020), the cost of training NLP models]

Ball-park cost estimates for training these models:

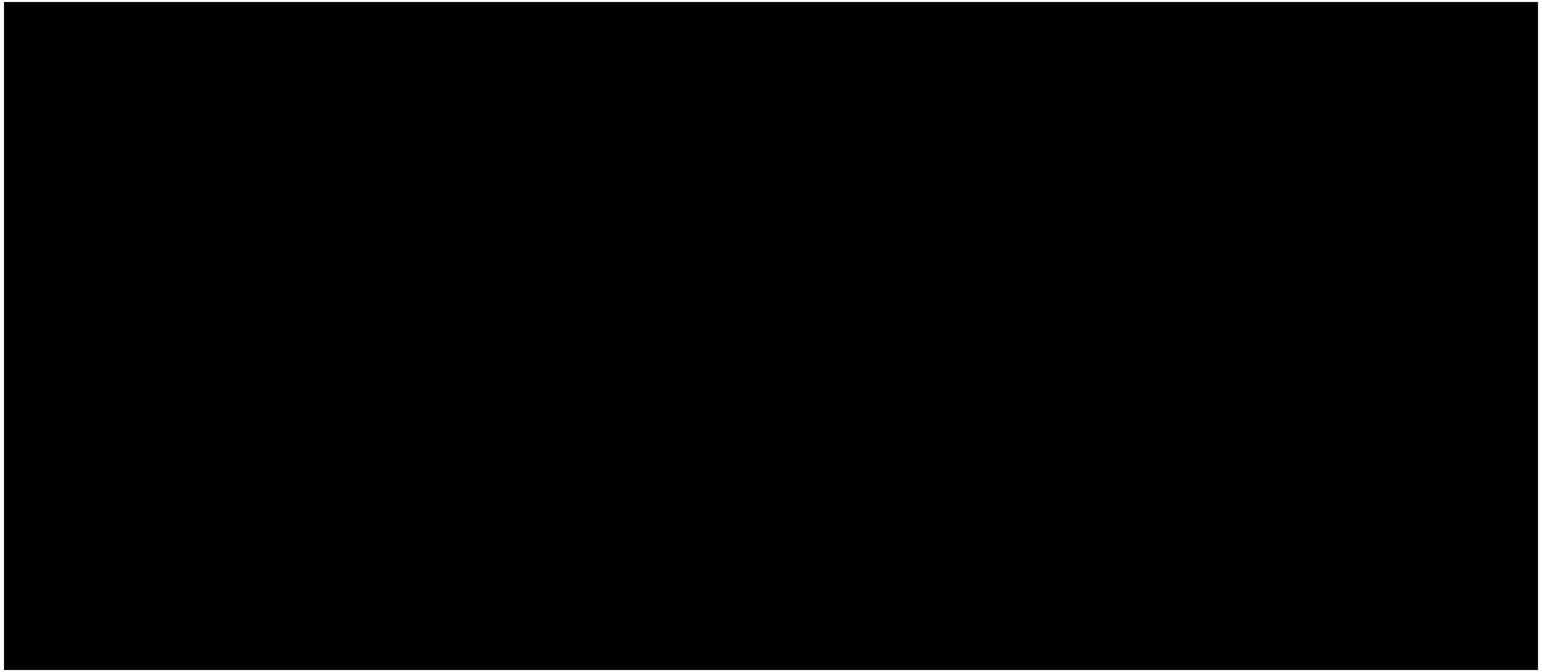
- **GPT-1**
  $2.5k - $50k
- **BERT-large, XLNet:**
  $10k - $200k
- **GPT-2:**
  $80k - $1.6m
- **T5:**
  $1.3 - $10m

sequence2sequence models



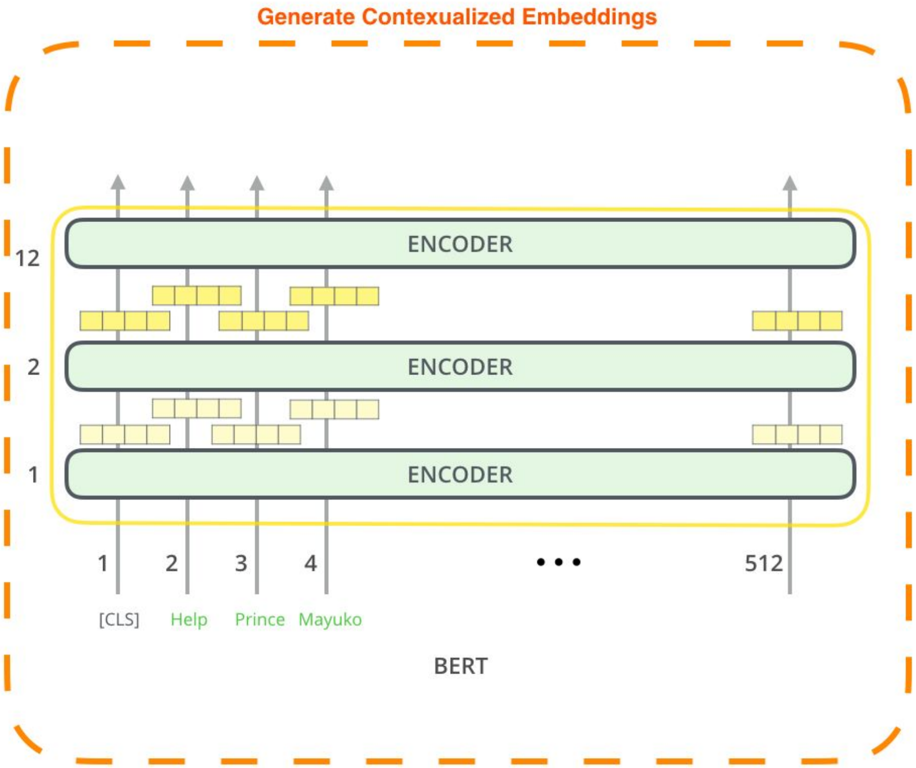
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

sequence2sequence models with attention

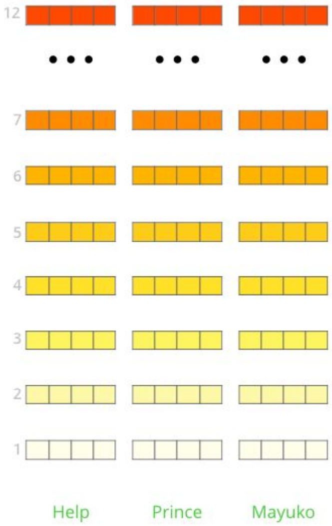


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

BERT for *contextualized* word embeddings (1 of 2)



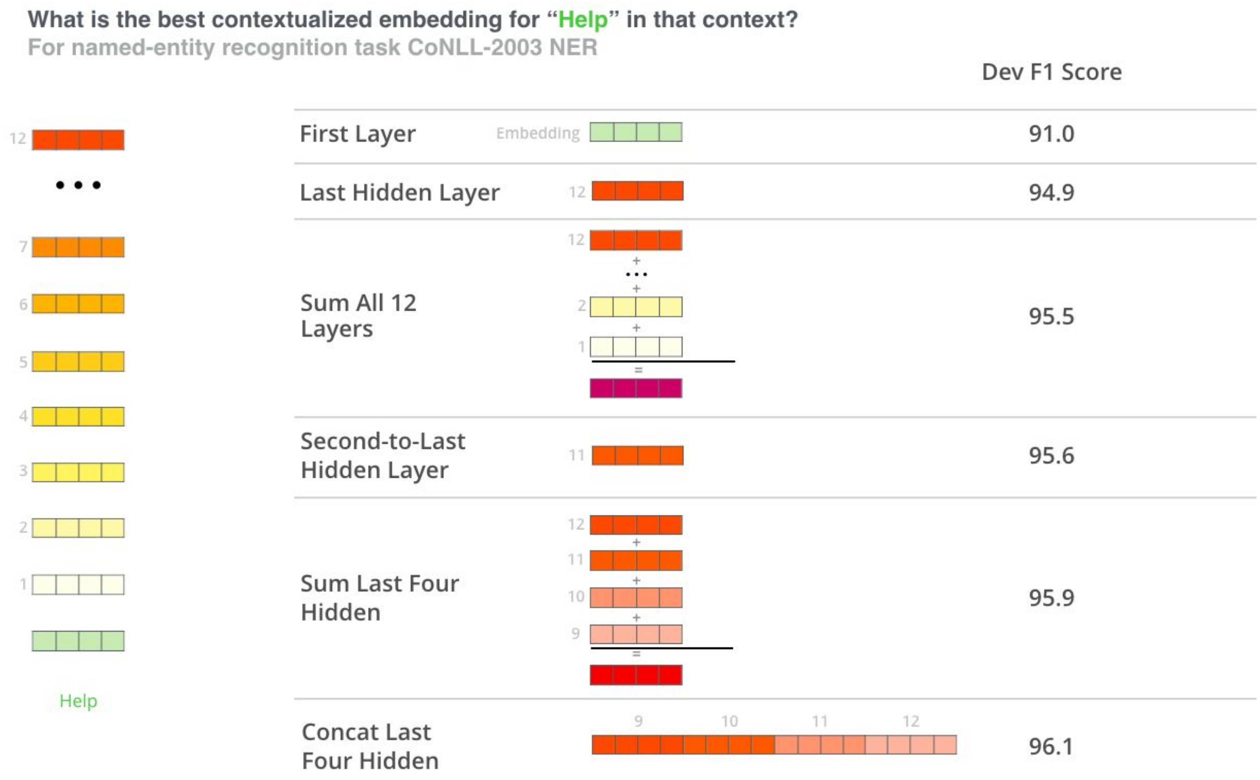
The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

http://jalammar.github.io/illustrated-bert/

BERT for *contextualized* word embeddings (2 of 2)

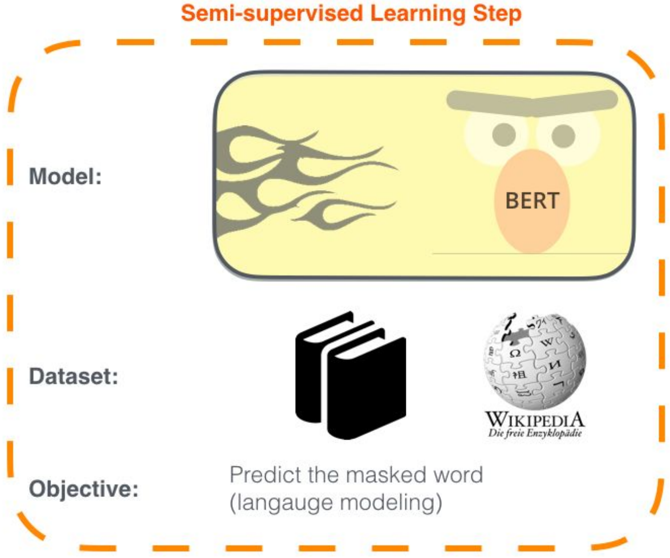


http://jalammar.github.io/illustrated-bert/

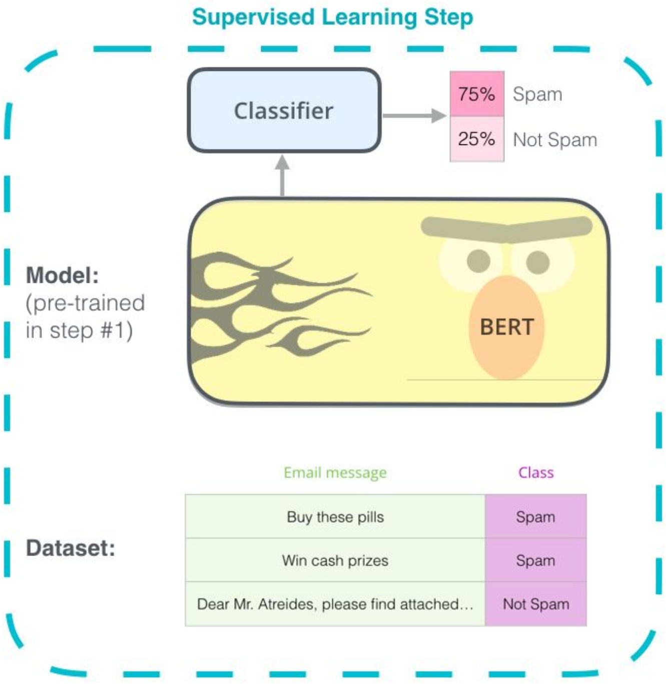
A BERT's eye view of state-of-the-art NLP models

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - Supervised training on a specific task with a labeled dataset.



http://jalammar.github.io/illustrated-bert/

Some final points to ponder

Current state-of-the-art: GPT-3

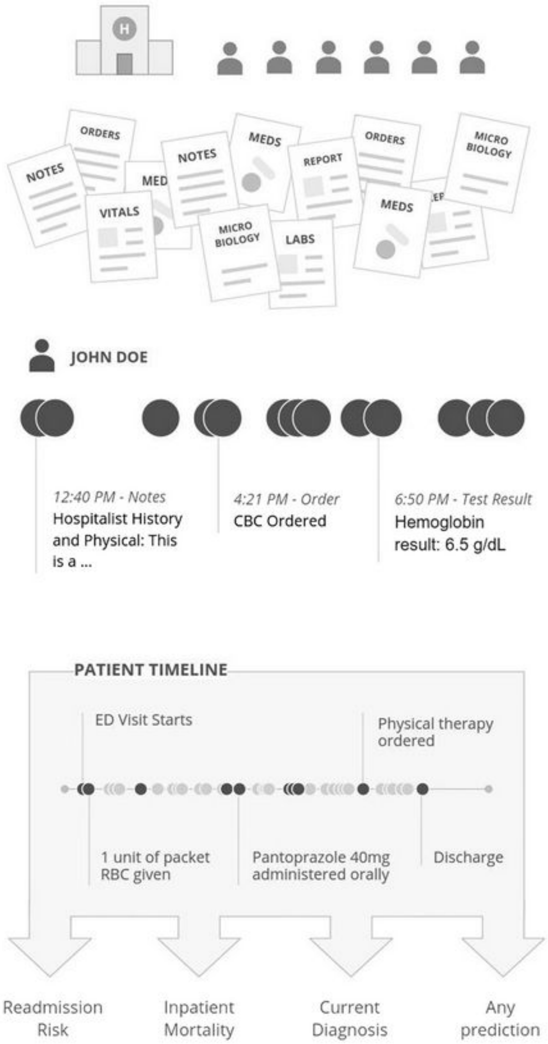
OpenAI, a non-profit artificial intelligence research company backed by Peter Thiel, Elon Musk, Reid Hoffman, Marc Benioff, Sam Altman and others, released its third generation of language prediction model (GPT-3) into the open-source wild. Language models allow computers to produce random-ish sentences of approximately the same length and grammatical structure as those in a given body of text.

In my early experiments with GPT-3 I found that GPT-3's predicted sentences, when published on the bitcointalk.org forum, attracted lots of positive attention from posters there, including suggestions that the system must have been intelligent (and/or sarcastic) and that it had found subtle patterns in their posts. I imagine that similar results can be obtained by republishing GPT-3's outputs to other message boards, blogs, and social media.

I predict that, unlike its two predecessors (PTB and OpenAI GPT-2), OpenAI GPT-3 will eventually be widely used to pretend the author of a text is a person of interest, with unpredictable and amusing effects on various communities. I further predict that this will spark a creative gold rush among talented amateurs to train similar models and adapt them to a variety of purposes, including: mock news, "researched journalism", advertising, politics, and propaganda (...)

https://maraoz.com/2020/07/18/openai-gpt3/

Scalable and accurate deep learning with electronic health records (2019)



1

Health systems collect and store electronic health records in various formats in databases.

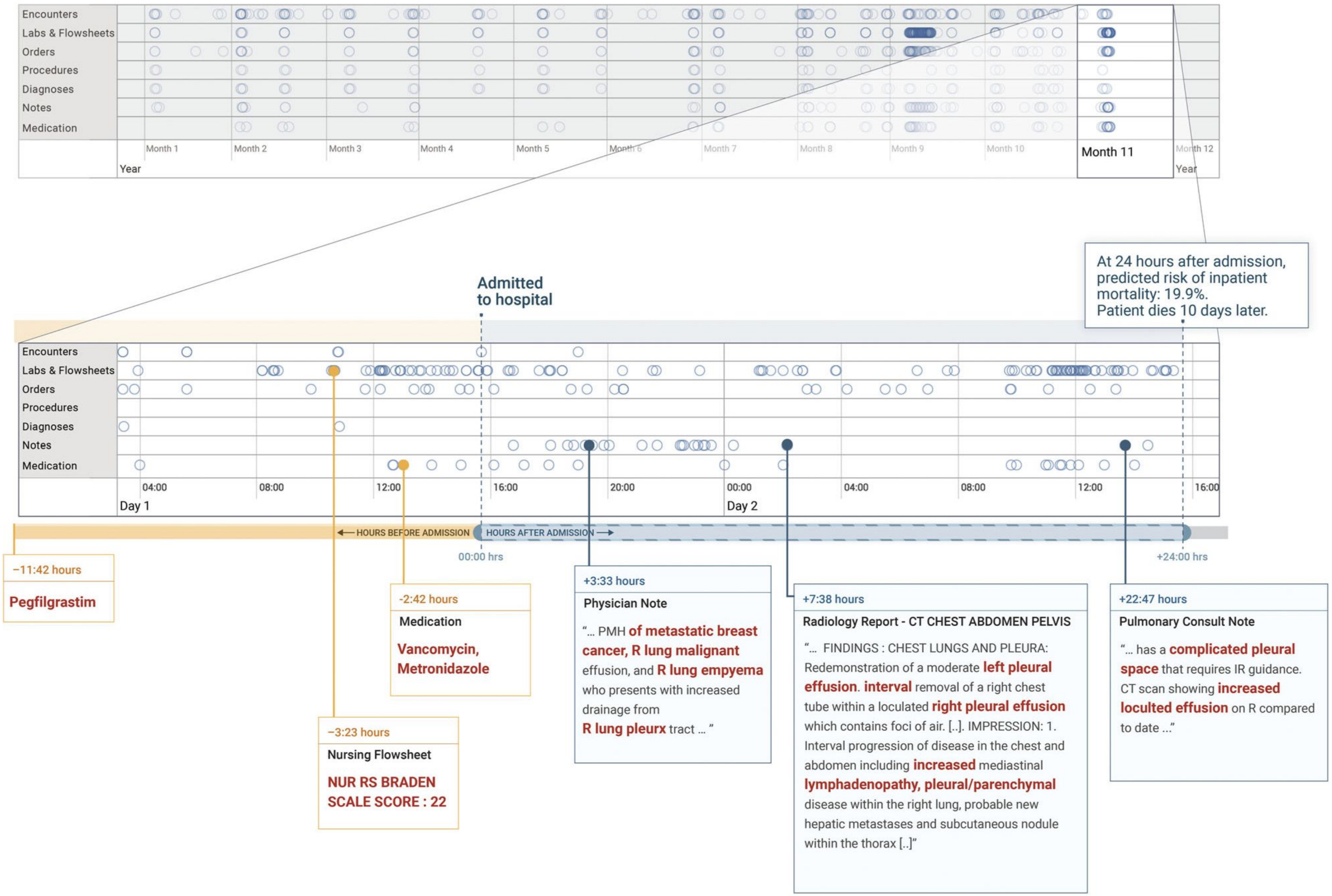
2

All available data for each patient is converted to events recorded in containers based on the Fast Healthcare Interoperability Resource (FHIR) specification.

3

The FHIR resources are placed in temporal order, depicting all events recorded in the EHR (i.e. timeline). The deep learning model uses this full history to make each prediction.

Patient Timeline



Keep It Simple, Stupid

- For most run-of-the-mill projects the large spaCy models should work just fine
- A pragmatic approach of stacking several simple models is often very effective
  - Combine text with tabular data
  - Try different approaches of tokenization, lemmatization
- As much as I like open source, be cautious about using standard lexicons
  - Data Science Lab sentiment lexicons are far from perfect, e.g.
    positive: *gedomineerd, Renaissance, nederlagen, koel*
    negative: *radicaal, concurrent, kever, grap*