An Introduction to Natural Language Processing and Web Mining

Guillaume Gravier

guillaume.gravier@irisa.fr

Who's who

- **Guillaume Gravier**, CNRS researcher

- **Christian Raymond**, INSA faculty
- **Morgane Casanova**, CNRS research engineer
- **Cyriel Mallart**, Université de Rennes 2 research engineer
- **Loïc Fosse**, Orange Labs PhD student with Aix-Marseille Université
- **Yasmine Hachani**, Inria PhD student with Université de Rennes

Outline of the course

Lectures (3 x 3h)

- Lecture #1: Introduction, representation of words and documents
- Lecture #2: Language modeling and tagging
- Lecture #3: Sequence to sequence models, transformers, standard tasks

Lab works / **projects** (4 x 3h)

- Lab #1: Playing with word and sentence embedding
- Lab #2: Classifying documents
- Lab #3 & #4: Scrapping data and developping an NLP app

Advanced lab works for SID and marketing students (4 x 3h) on training NLP models with tf/keras

Evaluation: project report from Lab #3 & #4 for all, plus interview and result for advanced lab works

Pillow books

Daniel Jurafsky, James H. Martin. Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition, 2nd edition, Prentice-Hall, 2009. Re-edited in 2023.

Yoav Goldberg. Neural Network Methods for Natural Language Processing, 2017.

Links to online version available on moodle.

Some definitions of 'natural language'

A few definitions from `dictionary.com`:

1. a body of words and the systems for their use common to a people who are of the same community or nation, the same geographical area, or the same cultural tradition
2. communication by voice in the distinctively human manner, using arbitrary sounds in conventional ways with conventional meanings
3. the system of linguistic signs or symbols considered in the abstract (opposed to speech)
4. any set or system of such symbols as used in a more or less uniform fashion by a number of people, who are thus enabled to communicate intelligibly with one another
5. any system of formalized symbols, signs, sounds, gestures, or the like used or conceived as a means of communicating thought, emotion, etc.: the language of mathematics; sign language. the means of communication used by animals

Some definitions of 'natural language processing'

1. NLP aims at creating computer programs able to automatically process natural languages [Bouillon 98]
   - NLP ... but also computational linguistics
   - rather text data than languages
2. All research and development aiming at modeling and reproducing, using machines, the human ability to produce and understand linguistic utterances for communication purposes [Yvon 07]
3. NLP is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data [wikipedia en]

Modeling of language competence and reproduction of language performance... but not necessarily in the same way as humans do

The ELIZA example – Joseph Weizenbaum, MIT, 1966

Automatic psychological analysis simulated:

[he says] I am depressed all the time

```
I am sorry to hear you are depressed
```

It's true that I am unhappy

```
Do you think coming here will help you not be unhappy
```

I need help, that seems certain

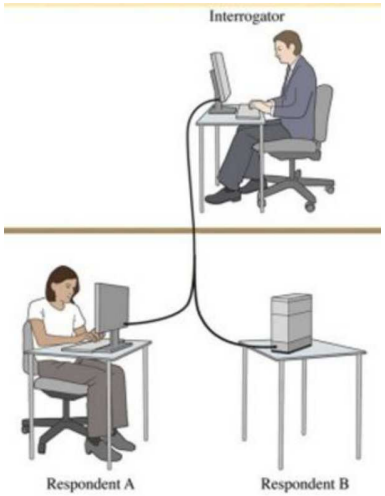
```
What would it mean if you got some help
```

Perhaps I could learn to get along with my mother

```
Tell me more about your family
```

ELIZA passes the Turing test but in fact, no comprehension: merely keyword spotting and basic rules, e.g.,

```
s/.* all .*/IN WHAT WAY?/1
s/.*I am (depressed).*/I AM SORRY TO HEAR YOU ARE \1/1
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/1
```



See demo at `http://www.manifestation.com/neurotoys/eliza.php3`

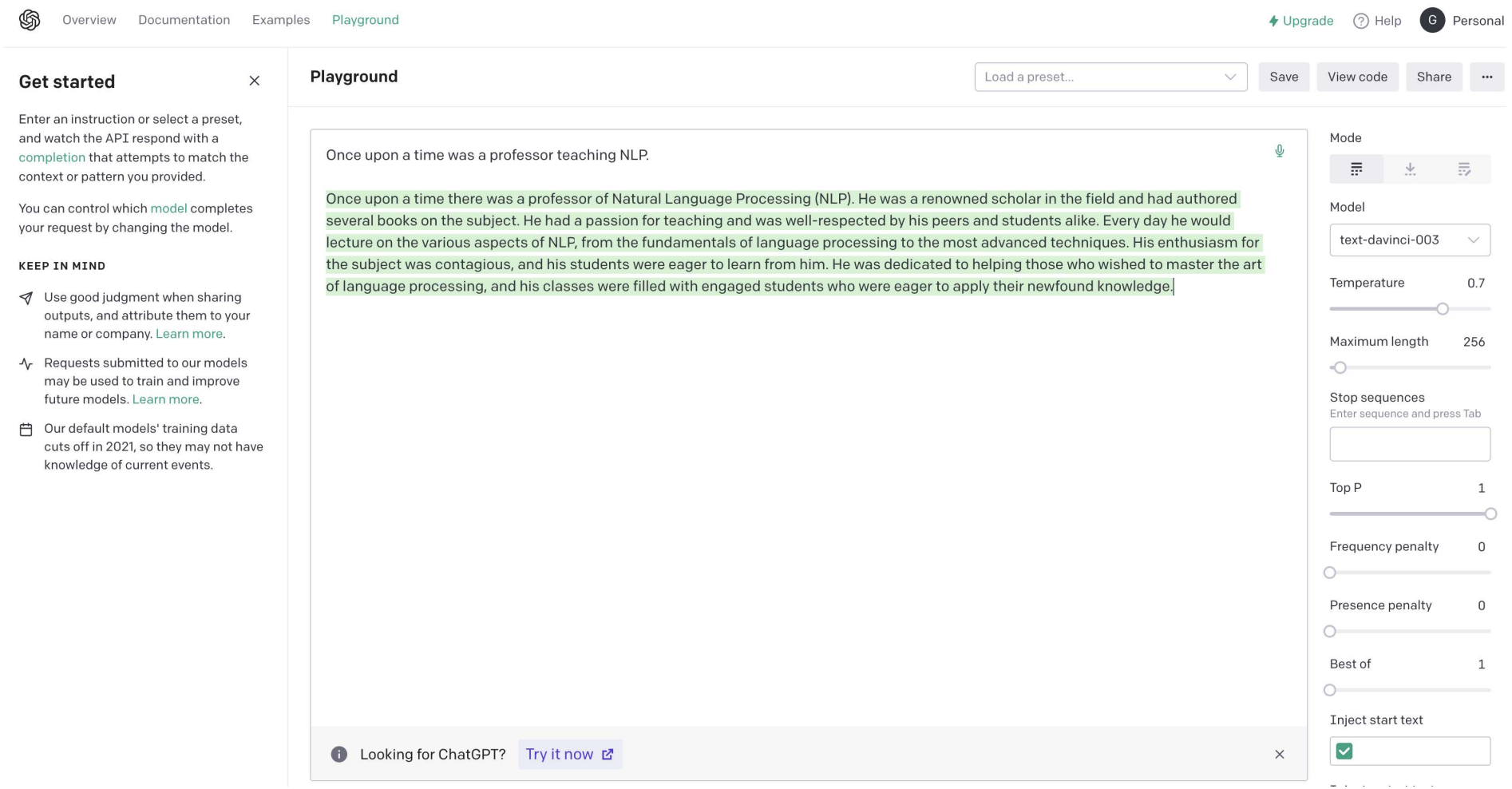
The Google captioning example – Oriol Vinyals et al., 2014

Automatic generation of captions from images ...

... with brute force machine learning behind the scene

The GPT-3 example – Tom Brown et al., 2020

Natural language generation from a prompt ...



here again with brute force machine learning behind the scene

Language is a complex multilevel process

Le président des antialcooliques mangeait une pomme avec un couteau.

(example borrowed from A. Vilnat)

Phonological: study of sounds in speech, organization of sounds

Graphical: segmentation of text into basic units (words?)

Lexical and morphological: identification of the lexical components (words?), of their properties

Syntactic: identification of higher level components and their organization (sentence level)

Semantic: constructing a representation of the meaning

Pragmatic: identifying the function of an utterance in its context

All levels are strongly intricated

- *J'ai lu un livre* $\longrightarrow$ gender determined by the determiner (syntactic)
- *Il ne s'agit pas de livres mais de lires* $\longrightarrow$ gender at the pragmatic level

Language is highly complex and ambiguous

- non structured data = no semantics brought by *a priori* structure
- graphical complexity and ambiguity
  - *etc*., SNCF, aujourd'hui, 1,23 %, 1939-45, Jean-Paul II, Washington, ...
  - park (v/n), rat (v/n/a) – in French, président (n/v), couvent (n/v)
- syntactic ambiguity
  - I saw the man with the binoculars. Look at that dog with one eye.
  - La belle ferme le voile. La petite brise la glace.
  - I saw a movie with Brad Pitt.
- semantic ambiguity
  - homonymy: park, bank, bright
  - polysemy: dish (plate *vs*. meal), glass (container *vs*. content)
- pragmatic ambiguity (co-references but not only)
  - A: Are you coming at Luke's party tonight?
  - B: I hear Ben will be there.

Also for different languages: tonal languages, aggutinative languages, etc.

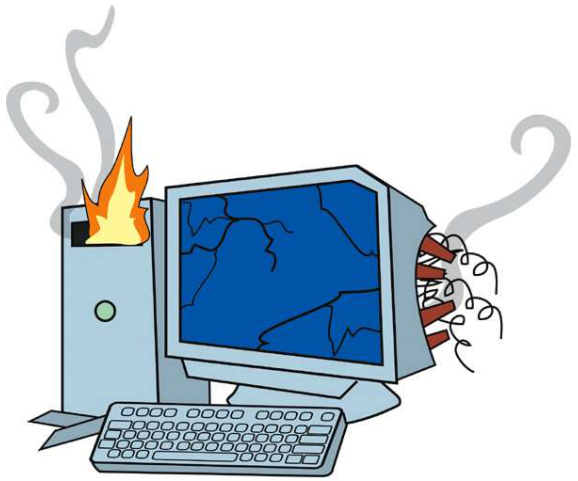
Language is highly complex and ambiguous (cont'd)

Lot of implicit knowledge and common sense in (human) language interpretation (that computers lack):

- "Obvious" interpretation (anaphora/coreference resolution)
  - She ordered a beer to the waitress. Then she left without paying.
  - The profressor sent the pupil to the princiapl becase
    - he was throwing pellets
    - he wanted peace
    - he wanted to see him

(example borrowed from F. Yvon)

- Metaphor, metonymy
  - you're a black sheep, you have her ear, the room applauded
- Common sense knowledge
  - I read an article on the accident in the newspaper.
  - I read an article on the accident in the metro.



Landmarks and trends: from symbols to machine learning

The *rationalism era* (1950-1980): linguistically-motivated models

- precise models targeting **exact meaning**
- Harris's distributional linguistics (1954), Chomsky's theories on syntax (1957)
- focus on machine translation from 1952 until ALPAC's report in 1966
- AI symbolic approaches in the 70s

  e.g., ELIZA (1966), SHRDLU dialogue (1972), Minsky's frame theory (1974)

The *statistical era* (1990-2010): corpus-based statistical models

- statistical models targeting **useful meaning**
- no longer trying to understand and be realistic
- heavy use of annotated corpora

  e.g., Brown corpus (1960), British national corpus (1994), Switchboard (1992)

The *machine learning* era (2010-20??): corpus-based deep learning

- strong trend towards deep learning in NLP
- from Bengio's work (2003) to *word2vec* (2013) and BERT (2017)
- a huge lot of neural architectures leveraging word and sentence embedding

Modern NLP frameworks (in python)

Powerful and flexible python frameworks available today, built on top of standard libraries like numpy/scipy, tensorflow and/or pyTorch, among which:

- gensim – `https://radimrehurek.com/gensim`
  - motly dedicated to word and document embedding
  - word2vec, fasttext, tf-idf, LSI, LDA, random projections

  spaCy – `https://spacy.io`
  - flexible framework designed for production use
  - standard pipelines for various languages
  - model training
  - compatibility with AWS

- NLTK – `http://www.nltk.org/`
  - NLP research full environment
  - copora and lexicons (including WordNet) plus standard processing tools
  - model training
  - industry-compatible with front-end to many open source tools

Modern NLP frameworks (in python and others) — cont'd

- Stanford CoreNLP – `https://stanfordnlp.github.io/CoreNLP/index.html`
  - wide range of JAVA tools and libraries
    $\Rightarrow$ tokenizer, tagger, parser, entity recognition and linking
  - python binding `https://stanfordnlp.github.io/stanfordnlp/`
  - many models and pipelines (53 languages)

```
> import stanfordnlp
> stanfordnlp.download('en')
> nlp = stanfordnlp.Pipeline()
> doc = nlp("Barack Obama was born in Hawaii.")
> for item in doc.sentences[0].tokens:
>     print(item)
```

- allenlp – `https://allennlp.org`
  - NLP research development and research environment
  - lots of pretrained models and easy training (interfacing pyTorch)

Modern NLP frameworks (in python and others) — cont'd

- pyTorch and TorchText – `https://pypi.org/project/torchtext`
- tensorflow and tensorflow_text – `https://www.tensorflow.org/text`
  - research environment for neural model training with pyTorch / tensorflow
  - provides easy access to and manipulation of corpora
- HuggingFace's transformers – `https://pypi.org/project/transformers`
  - large repository of pre-trained models for numerous tasks
  - library to fine-tune or retrain models

It's a mess out there! There are many many [...] many other resources on the web, not all are very good and well-maintained.

About web scrapping

About data scrapping

NLP requires text ... which in many cases is *scrapped* from the web or social networks.

Scrapping is nothing but easy because data is the new gold

- content is often protected and not easily parsable
- many APIs are proprietary, with limited sample access
- HTML pages are messy these days
  $\rightarrow$ not sufficient to grab <p> tags

So no single generic solution but some frameworks that can help significantly

Generic libraries to access HTML and XML data

- `urllib.request` – a practical library to grab raw web pages (DOM)
  - functions and classes which help in opening URLs (mostly HTTP) in a complex world

```
> import urllib.request

> with urllib.request.urlopen('http://lemonde.fr') as f:
...     print(f.read(1000).decode('utf-8'))
...
<!DOCTYPE html>
<html lang="fr" prefix="og: http://ogp.me/ns#">
<head> <meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

- `Requests` – `https://requests.readthedocs.io`

Generic libraries to access HTML and XML data

- Nice libraries to parse and search the HTML/XML code
  - `PyQuery` – `https://pythonhosted.org/pyquery`
  - `HTMLParser` – `https://docs.python.org/2/library/htmlparser.html`
  - `BeautifulSoup` – `https://www.crummy.com/software/BeautifulSoup`

```
> from bs4 import BeautifulSoup
> f = urllib.request.urlopen('http://lemonde.fr')
> dat = f.read()
> soup = BeautifulSoup(dat, 'html.parser')
> soup.find_all('p')
<p class="article__title">Le Mobile World Congress touché par le cor
<p class="article__title">Le coronavirus met « le fret est à fond de
<p class="article__title">« La discussion objective sur la stratégie
<p class="article__title">Comprendre la colonisation israélienne en
```

Messy example of HTML code

Specific libraries for web data

- `feedparser` – `https://pythonhosted.org/feedparser`

  python module for downloading and parsing syndicated feeds (RSS/atom)

```
> import feedparser as fp
> url = 'https://investir.lesechos.fr/RSS'
> fn = 'info-marches-investir-bourse-les-echos.xml'
> doc = fp.parse('/'.join([url,fn]))
> doc['feed']['title']
'Marchés, Indices, matières premières, analyses de séance, Taux'
> item = doc['entries'][0]
> item['published']
Fri, 07 Feb 2020T19:00:00 +0200
> item['title']
Rétro de la semaine : Le Cac 40 reprend les 6.000 points
> item['links'][0]['href']
'https://investir.lesechos.fr/marches/actualites/le-cac-40-reprend-les-6-000
```

Specific libraries for web data (cont'd)

- `newspaper` – `https://newspaper.readthedocs.io`

  python module for extracting and curating articles

```
> import newspaper as np

> article = np.Article(item['links'][0]['href'])
> article.download()
> article.parse()
> article.publish_date
datetime.datetime(2020, 2, 7, 19, 0, tzinfo=tzoffset(None, 3600))
> article.text
"Oubliée, la crise sanitaire du coronavirus. Ou presque. Après deux
semaines consécutives de baisse, le Cac 40 a rebondi de 3,85 % sur cinq
jours, ...
> article.top_image
'https://pics-investir.lesechos.com/images/favicon/apple-icon-57x57.png'
```

Specific libraries for web data (cont'd)

```
> lemonde = np.build('http://www.lemonde.fr')

> for article in lemonde.articles:
      print(article.url)
https://www.lemonde.fr/economie/article/2020/02/07/le-mobile-world-congress-
https://www.lemonde.fr/economie/article/2020/02/08/le-coronavirus-met-le-fre

> article = lemonde.articles[0]
> article.download()
> article.parse()
```

Example of specific libraries for social network

the Twitter API, tweepy or twython

```
import twython

twitter = twython.Twython(APPKEY, APPSECRET, OAUTHTOKEN, OAUTHSECRET)

query = 'wine OR beer -filter:retweets AND -filter:replies'
response = twitter.search(q=query, count=100)
statuses = response['statuses']
for status in statuses:
    print status['in_reply_to_status_id'], status.has_key('retweeted_status'
```

Facebook graph API and its python binding

```
import facebook
graph = facebook.GraphAPI(access_token='your_token', version='2.12')
post = graph.get_object(id='post_id', fields='message')
print(post['message'])
```

Wikipedia API, etc.

Basic concepts of NLP

Raw text is not what you want to play with

I got this as both a book and an audio file. I had waited to read it and was surprised by both the enthusiasm of the content and its author, but also by how he snuck in some odd biblically unsound thoughts (e.g., I gasped when he suggested Christ went to Hell...what Bible passage evidences this?).

I agree with how he suggests the enemy is out to deceive us and keep us asleep...but wonder if I go further how much more Eldredge will slip in of his own peculiar biblical misintrepretations. Where were his editors when this was being written? Why take sensible good sections and mar them with oddities?

I havent read all the reviews here but as one of those "conservatives" frequently mentioned in them I have to admit I may not even finish this book for fear of what else Eldredge has slipped up on.

I did appreciate his story about Daniel and the "delayed" angel...but am left wondering if I need go deeper into researching that as possible misintepretation too.

[extract from CLS corpus: review polarity classification task]

Raw text is really not what you want to play with

@fa6ami86 so happy that salman won. btw the 14sec clip is truely a teaser

@phantompoptart .....oops.... I guess I'm kinda out of it.... Blonde moment -blushes- epic fail

@bradleyjp decidedly undecided. Depends on the situation. When I'm out with the people I'll be in Chicago with? Maybe.

Just grabbed some bagels from Panera for everyone at wk. It's Brittany's last day

AHH i'm so HAPPY. I just found my ipod. God is sooo good to me!

Hoping for a better day today. Yesterday was fine until I passed out at my desk, fell off chair, and carpet burned my forehead.

@poepiandzegiant oops just saw you said hello! Hi there

att workkk dyinggg to get the fxckkkkk outt

[extract from carblaca/twitter-sentiment-analysis]

Tokenization: a crucial step at the graphical level

Tokenization is about breaking a text into *sentences* (or sort of sentences) and sentences into *tokens*

- based on punctuation marks and spaces
- plus a huge lot of rules and exceptions
  - j'ai $\longrightarrow$ j' + ai *vs*. aujourd'hui $\longrightarrow$ aujourd'hui
  - 31/12/2019 $\longrightarrow$ 31/12/2019 *vs*. 31 décembre 2019
  - what do we do with quotation marks???? parenthesis????
- might overlap with lexical and morphological anlalysis (but not necesssarily)
- painful and language-dependent but crucial step in NLP pipelines
- a few existing (free) tools available

Stanford	`https://nlp.stanford.edu/software/tokenizer.shtml`
NLTK	`http://www.nltk.org/api/nltk.tokenize.html`
spaCy	`https://spacy.io/api/tokenizer`

Tokenization at work with NLTK and spaCy

```
> from nltk.tokenize import word_tokenize, sent_tokenize
> s='A $2 example\nof a sentence. And a 2nd sentence.'
> word_tokenize(s)
['A', '$', '2', 'example', 'of', 'a', 'sentence', '.',
'And', 'a', '2nd', 'sentence', '.']
>  [word_tokenize(x) for x in sent_tokenize(s)]
[['A', '$', '2', 'example', 'of', 'a', 'sentence', '.'],
['And', 'a', '2nd', 'sentence', '.']]

> import spacy
> nlp = spacy.load('en_core_web_sm')
> doc = nlp(s)
> [token for token in doc]
[A, $, 2, example,
, of, a, sentence, ., And, a, 2nd, sentence, .]
```

Sub-word tokenization

Recent models makes use of sub-word tokens derived from large amount of data based on the frequency of occurrence of substrings, e.g.:

- byte pair encoding (BPE)
  $\longrightarrow$ used in GPT-3
- WordPiece
  $\longrightarrow$ used in BERT
- Unigram / SentencePiece

```
aaabdaaabac

AabdAabac
A=aa

ABdABac
A=aa B=ab
```

```
> tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
> s = "Using a Transformer network is simple"
> tokens = tokenizer.tokenize(s)
> print(tokens)
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

Morphology and lexicons

Lexicology = inventory and classification of the usages of “words”

- *lexical unit* $\simeq$ entry in a lexicon or dictionary
- existence of many *forms*: aimer, aime, aimé, aimions, aimât, *etc*.
- categories of lexical units: noun, verb, adjective, preposition, *etc*.

Morphology = process of the formation of words

- inflection: gender, plural, conjugation, *etc*. (no category change)
  - love $\rightarrow$ loved, loves, love’.
- derivation: new lexeme derived from existing ones (category change)
  - happy $\rightarrow$ happiness, happening
- composition: combination/concatenation of forms
  - can opener, neural network, pomme de terre

$\Rightarrow$ *morpheme* = the smallest meaningful morphological unit

- Example: antialcooliques = [anti] [[alcool] [ique]] [s]
- lexical/root morpheme = *lexeme*
- grammatical morphemes, in particular *affixes* such as prefixes, suffixes

word, lexeme, lemma, form and other fun things

The notion of *word* is fuzzy and ambiguous:

1. Surface: *His answer was only two-words long: certainly not*
2. Sign: *Am, are, is, was ... are some forms of the same word*

We should rather refer to the following concepts

token	a (normalized) graphical string, without meaning
(word)form	linguistic sign with a certain functioning autonomy and a certain internal cohesion
lexeme	cluster of wordforms distinguished by inflection (2) ○ described by its form (signifier) and meaning (signified) ○ comes with a part-of-speech (PoS) category: N, V, Adj
lemma	(arbitrary) canonical form used to represent a lexeme (e.g., aimer)
stem	morphological support of a lexeme, bearing the signified (e.g., aim-)

Lemmatization, stemming and POS tagging

- **Stemming**: wordform $\longrightarrow$ stem (usually morphological analysis)

```
> stemmer = nltk.stem.porter.PorterStemmer()
> stemmer.stem('candy')
'candi'
```

- **part-of-speech (POS) tagging**: wordform (token) $\longrightarrow$ POS tag

```
> nltk.tag.pos_tag(word_tokenize('The cat loves milk.'))
[('The', 'DT'), ('cat', 'NN'), ('loves', 'VBZ'), ...]
```

- **lemmatization**: wordform (token) $\longrightarrow$ lemma (requires POS tagging)

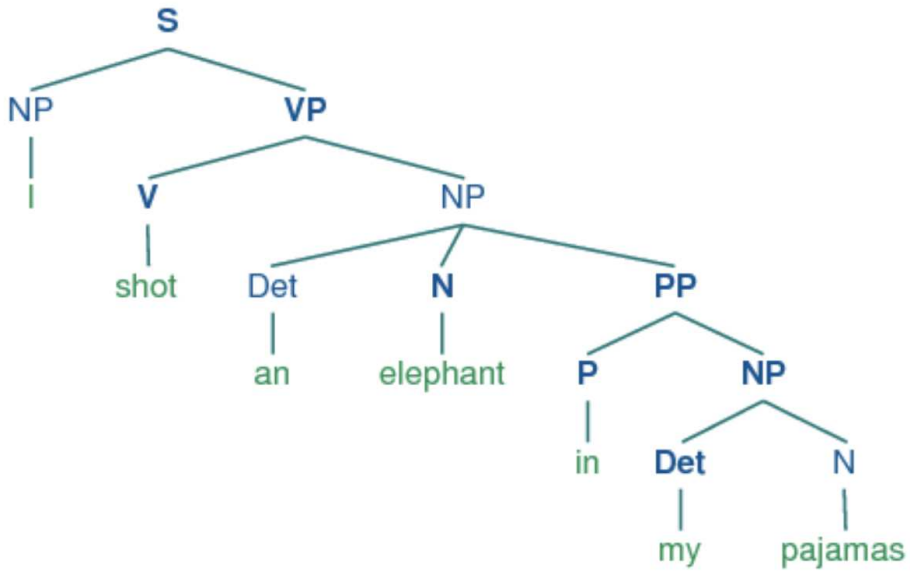
```
> from nltk.stem import WordNetLemmatizer
> lemmatizer = WordNetLemmatizer()
> lemmatizer.lemmatize('candies', pos='n')
> 'candy'
```

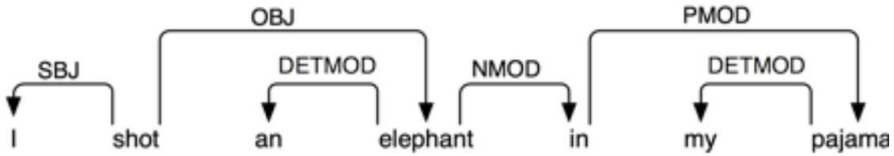
POS tagging and lemmatization at work

```
> import nltk
> tok = nltk.word_tokenize(s)
> tok
['The', 'cat', 'sleeps', 'on', 'the', 'mat', '.']
> tag = nltk.pos_tag(tok)
> tag
[('The', 'DT'), ('cat', 'NN'), ('sleeps', 'VBZ'), ('on', 'IN'),
('the', 'DT'), ('mat', 'NN'), ('.', '.')]

> nlp = spacy.load('fr_core_news_md')
> s='Les poules du couvent couvent.'
> doc = nlp(s)
> for token in doc:
...     print(token, token.pos_, token.lemma_)
Les DET le
poules NOUN poule
du DET de
couvent NOUN couvent
couvent ADV couvent
```

Representing the syntactic level

Parsing: unveil the sentence structure based on the the principles and constraints that govern the combination of words into grammatically correct sentences





constituency parse

phrase structure grammar

dependency parse

dependency grammar

No syntax model without naming Noam Chomsky

Formal (generative) grammars are widely used as syntactic models to describe language grammars and perform parsing:

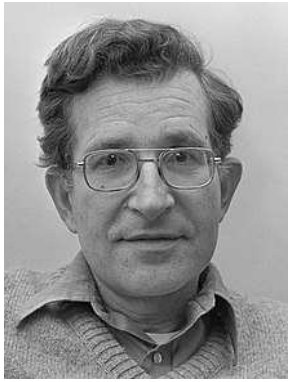
- terminal nodes V (= words)
- non terminal nodes N
- set of derivation rules of the form

$$(N \cup V)^* N (N \cup V)^* \rightarrow (N \cup V)^*$$

Comes in different types and flavors, in particular

- *context-free grammars* (type 2)
  $\rightarrow$ *left rule limited to a node from* N
- *regular grammars* (type 3)
  $\rightarrow$ *left rule:* N
  $\rightarrow$ *right rule:* $\emptyset$, *V, V N*

which can both be efficiently implemented



©Hans Peters / Anefo

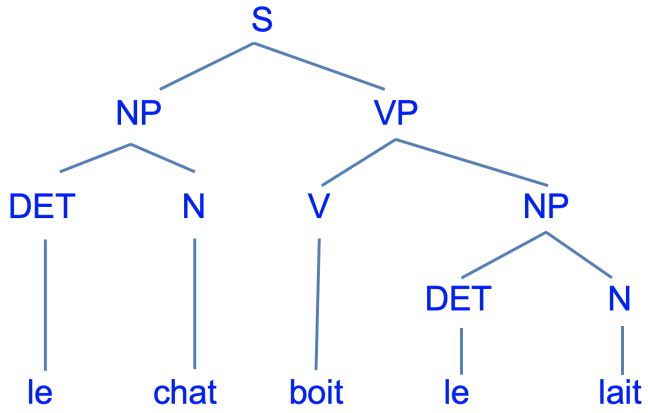
```
S -> NP VP
S -> VP NP
VP -> Verb
VP -> Verb NP
NP -> Noun
NP -> Det NP
...
Noun -> time
Noun -> arrow
Verb -> flies
...
```

ENSAI École nationale de la statistique et de l'analyse de l'information

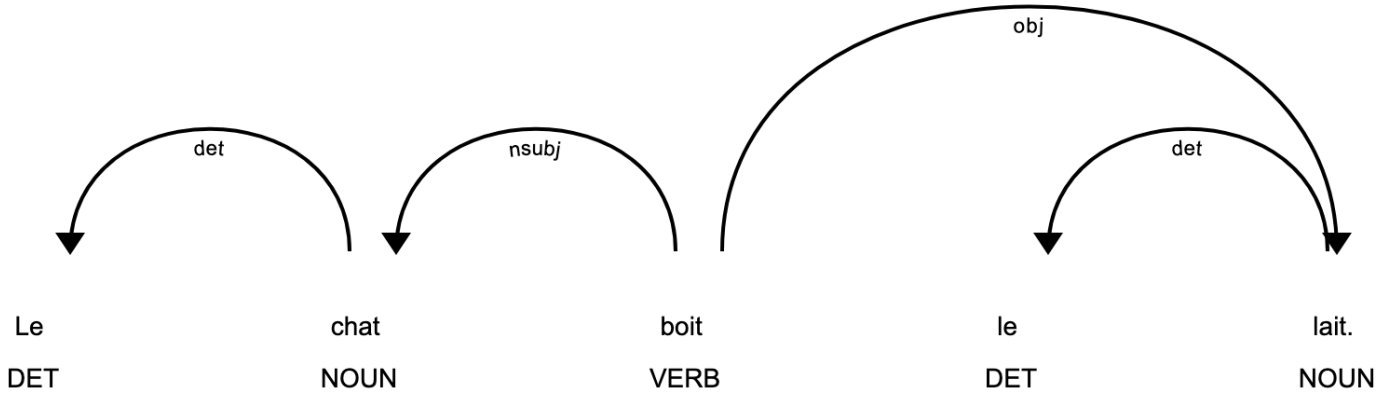
Grammars and parse trees

CFGs lead to constituency parse trees

S → NP VP
NP → Npr
NP → DET ADJ N
VP → V
VP → V NP
DET → la | le
N → chat | pomme | lait
Npr → Jean
V → mange | court | boit



from which one can infer dependency parse trees

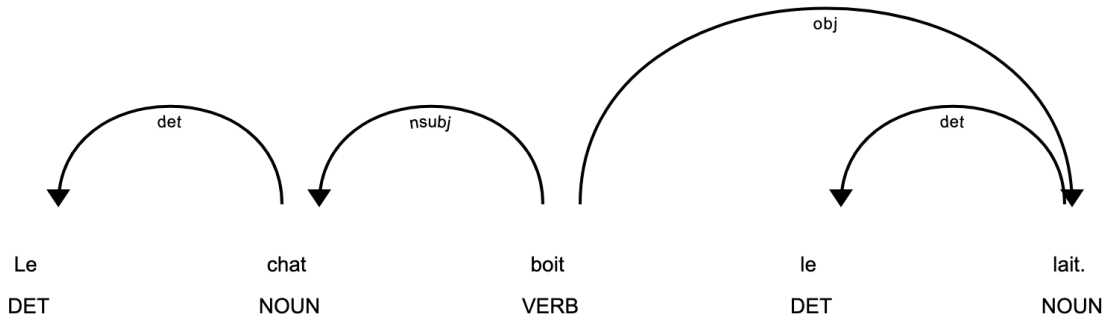


Note: this is not what is done in practice

Grammars and parse trees

Many tools for dependency parsing: MALT, Stanford CoreNLP, spaCy, etc.

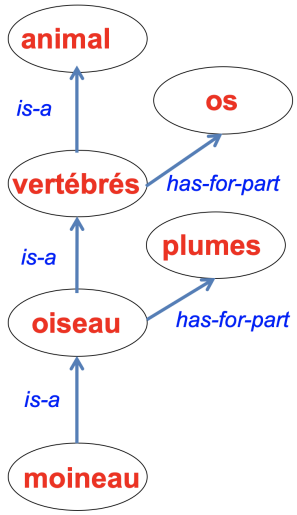
```
> import spacy
> process = spacy.load('fr_core_news_md')
> res = process('Le chat boit le lait.')
> for token in res:
>     print(token.text, token.pos_, token.lemma_, token.dep_)
Le        DET    le        det
chat      NOUN   chat      nsubj
boit      VERB   boire     ROOT
le        DET    le        det
lait      NOUN   lait      obj
.         PUNCT  .         punct
> spacy.displacy.render(xr, style="dep", jupyter=True)
```



About semantics

Representation of semantics in general goes (way) beyond NLP with no commonly adopted framework, among for instance

- precidate and description logics
- abstract meaning representation (AMR)
- semantic networks
- role semantics, frame semantics




```
      (w / want-01
:arg0 (b / boy)
:arg1 (g / go-01 :arg0 b))
```

$$\exists w, b, g : \text{instance}(w, \text{want\_01}) \wedge \text{instance}(g, \text{go\_01})$$
$$\wedge \text{instance}(w, \text{boy}) \wedge \text{arg0}(w, b)$$
$$\wedge \text{arg1}(w, g) \wedge \text{arg0}(g, b)$$

however many ways and resources for lexical semantics (i.e., reprensenting meaning of words)

- coocurrence and compositionality
- word nets

Newpaper quizz

Read in the newspaper Le Monde, Nov. 16 2023, article entitled “How to know if a text has been used by an AI” by David Larousserie:

These capacities are obtained by a rather brute force training procedure that consists in guessing the next word in a sentence taken from a very large corpus of texts, reaching thousands of billions of “tokens” (or semantic sub-units, such as syllables, prefixes, suffixes, etc.).

Correct or not?

Representation of words / tokens

Words = tokens = strings of characters

A natural solution is to represent the wordform as the corresponding string of characters

- very easy to implement
- rather efficient comparison with hash lookup tables (fixed vocabulary)
- symbolic representations are pratical for statistical models and meaning

but poor semantic information in the wordform representation

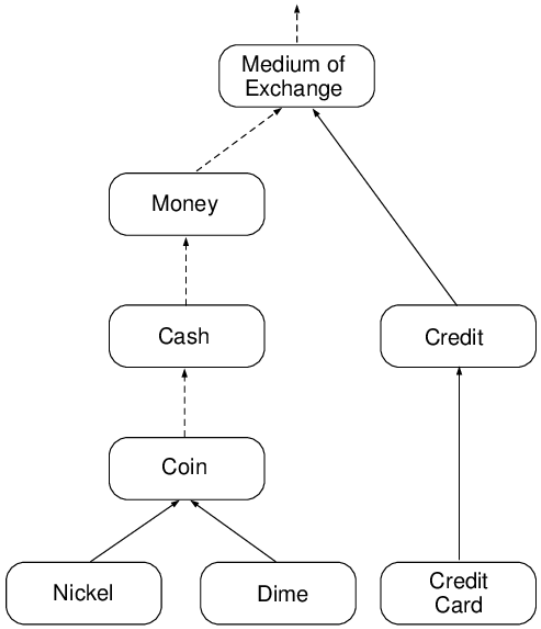
- hard to encode semantic relations with binary comparison
  - vélo $\neq$ bicyclette in the wordform domain
  - can maintain a matrix of semantic relations but costly
- hard to relate inflected forms
  - manger $\neq$ mangera $\neq$ mangerait in the wordform domain
  - can work with lemma but would loose semantic

plus not too convenient for neural networks!

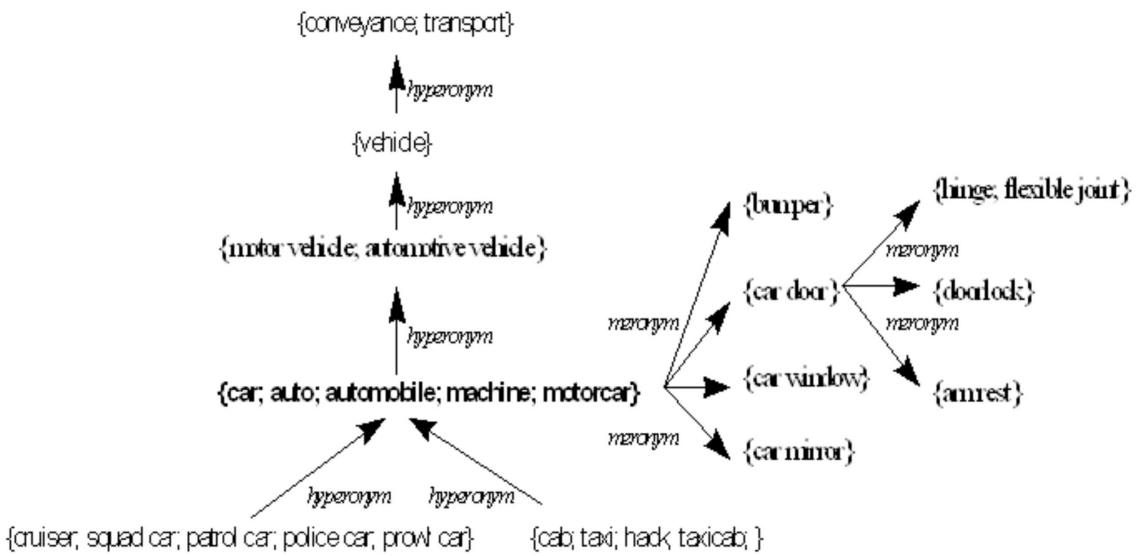
Lexicons and semantic networks

Existence of rich lexicons organized as frames or semantic network, e.g.,

- *FrameNet* – `https://framenet.icsi.berkeley.edu/fndrupal`
  - 13k word senses, 1.2k frames
  - mostly verbs with semantic roles
  - small French version available (ASFALDA)
- *WordNet* – `https://wordnet.princeton.edu`
  - 100k+ nouns, 20k+ adjectives, 10k+ verbs
  - lemmas grouped into sets of cognitive synonyms (synsets)
  - synsets interlinked with semantic and lexical relations
  - Wordnet Libre du Français (WOLF)
- *SentiWordNet* – `https://github.com/aesuli/sentiwordnet`
  - annotation of WordNet synsets with sentiment cues
  - positivity, negativity, objectivity



Lexicons and semantic networks: a quick zoom on WordNet



(a) WordNet structure

WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: eat Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (frequency) <lexical filename > (gloss)

Verb

- (61)<verb.consumption>S: (v) **eat** (take in solid food)
  - *direct troponym* / *full troponym*
  - *verb group*
    - <verb.consumption>S: (v) **eat** (eat a meal; take a meal)
    - <verb.consumption>S: (v) feed, **eat** (take in food; used of animals only)
  - *entailment*
  - *direct hypernym* / *inherited hypernym* / *sister term*
  - *derivationally related form*
    - <noun.person> W: (n) eater [Related to: eat] (someone who consumes food for nourishment)
    - <noun.food> W: (n) eater [Related to: eat] (any green goods that are good to eat)
    - <noun.act> W: (n) eating [Related to: eat] (the act of consuming food)
  - *sentence frame*
- (13)<verb.consumption>S: (v) **eat** (eat a meal; take a meal)
- (4)<verb.consumption>S: (v) feed, **eat** (take in food; used of animals only)
- <verb.emotion>S: (v) **eat**, eat on (worry or cause anxiety in a persistent way)
- <verb.consumption>S: (v) consume, eat up, use up, **eat**, deplete, exhaust, run through, wipe out (use up (resources or materials))
- <verb.change>S: (v) corrode, **eat**, rust (cause to deteriorate due to the action of water, air, or an acid)

(a) WordNet browser

WordNet similarity

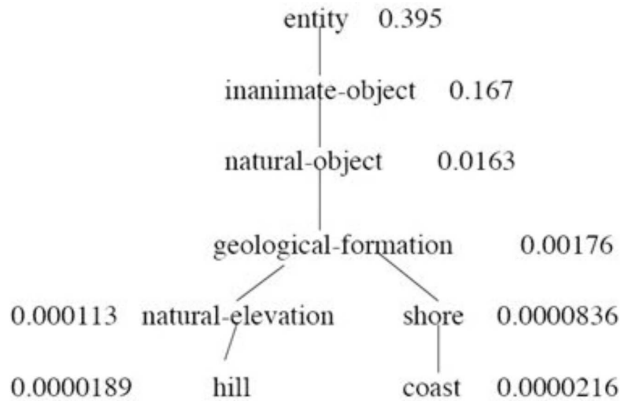
Exploit hierarchical structure of WordNet, with *least common subsumer* (LCS) of the two synsets to determine semantic proximity between two synsets.

$$\text{simpath}(c_1, c_2) = \frac{1}{1 + \text{shortest\_path\_length}(c_1, c_2)}$$

Generalize to words

$$\text{simword}(w_1, w_2) = \max_{c_1 \in S_1, c_2 \in S_2} \text{simpath}(c_1, c_2)$$

with $S_i = \text{senses}(w_i)$.



Many many many variants, e.g., with probabilities associated to the specificity of the concepts (P(entity) = 1) or ratio between LCS depth and synset depth

$$\text{Wu Palmer}(c_1, c_2) = 2\frac{\text{depth}(\text{lcs}(c_1, c_2))}{\text{depth}(c_1) + \text{depth}(c_2)}$$

Pedersen *et al.*, 2004. WordNet:: Similarity-Measuring the Relatedness of Concepts.

Playing with WordNet in practice

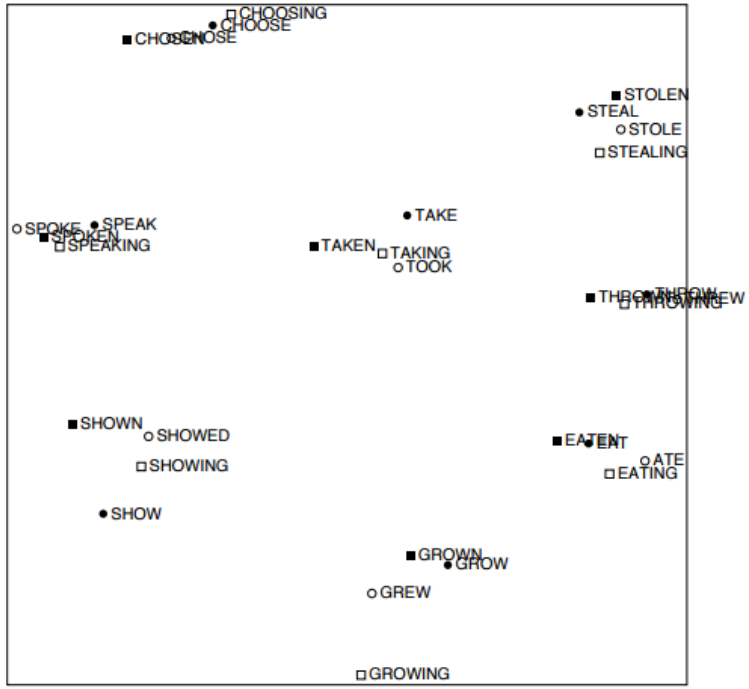
```
> from nltk.corpus import wordnet as wn

> wn.synsets('rat')
[Synset('rat.n.01'), Synset('scab.n.01'), Synset('rotter.n.01'),
Synset('informer.n.01'), Synset('rat.n.05'), Synset('rat.v.01'),
Synset('rat.v.02'), Synset('fink.v.01'), Synset('rat.v.04'),
Synset('rat.v.05'), Synset('denounce.v.04')]
> rat = wn.synset('rat.n.01')
> rat.hypernyms()
[Synset('rodent.n.01')]
> rat.hyponyms()
[Synset('bandicoot_rat.n.01'), Synset('black_rat.n.01'),
Synset('brown_rat.n.01'), Synset('jerboa_rat.n.01'),
Synset('pocket_rat.n.01'), Synset('rice_rat.n.01')]
> rat.lowest_common_hypernyms(wn.synset('man.n.01'))
[Synset('organism.n.01')]
> rat.wup_similarity(wn.synset('man.n.01')))
0.54
```

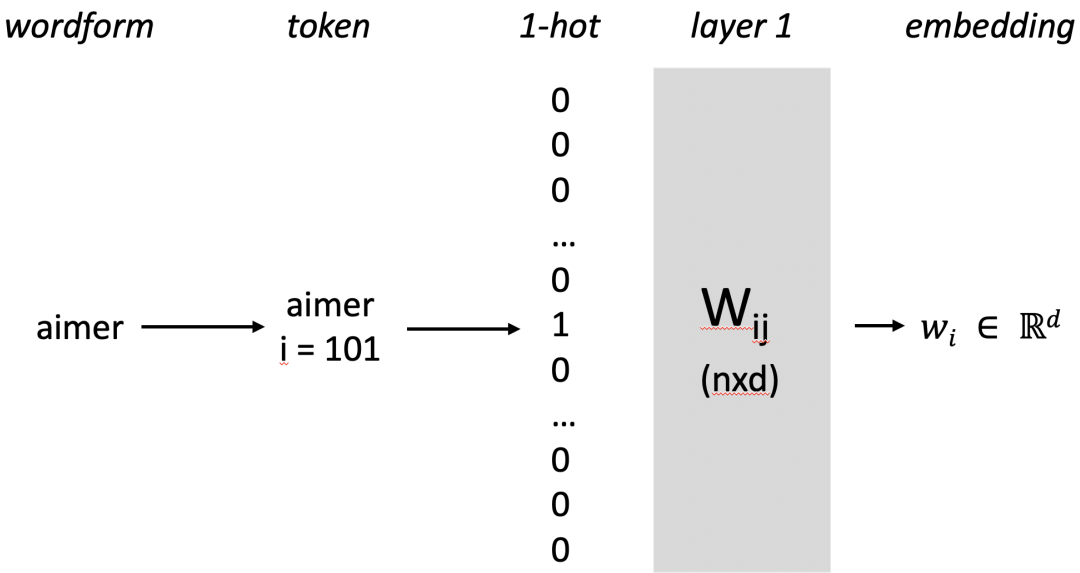
https://www.nltk.org/howto/wordnet.html

Distributed or embedded representations: What the heck?

The idea is to represent words (better said wordforms or tokens) in a Euclidean space with nice (semantic) properties

This is also known as *embedding*, which can be seen as the first layer of a neural network





- What properties for w_i?
- How to obtain w_i?
- How to evaluate w_i?

The seminal idea of distributional semantics

[...] the parts of a language do not occur arbitrarily to each other: each element occurs in certain positions relative to certain other elements. – Zellig S. Harris, Distributional structure, in Word, 10(23):146–162, 1954

You should know a word by the company it keeps – J. R. Firth, A synopsis of linguistic theory 1930-1955 in Studies, in Linguistic Analysis, 1–32, 1957

In practice, words are represented by the context (typically nouns and verbs, possibly lemmatized) they appear in, e.g.,

The dog barked in the park.

The owner of the dog put him on the leash since he barked.

...

	leash	walk	run	owner	leg	bark
dog	3	5	1	5	4	2
cat	0	3	3	1	5	0
lion	0	3	2	0	1	0
light	0	0	0	0	0	0
bark	1	0	0	2	1	0
car	0	0	4	3	0	0

A direct approach: the co-occurrence matrix

count co-occurrences of words within a context window

reduce dimension somehow (and avoid sparsity)

Corpus *I like deep learning. I like NLP. I enjoy flying.*

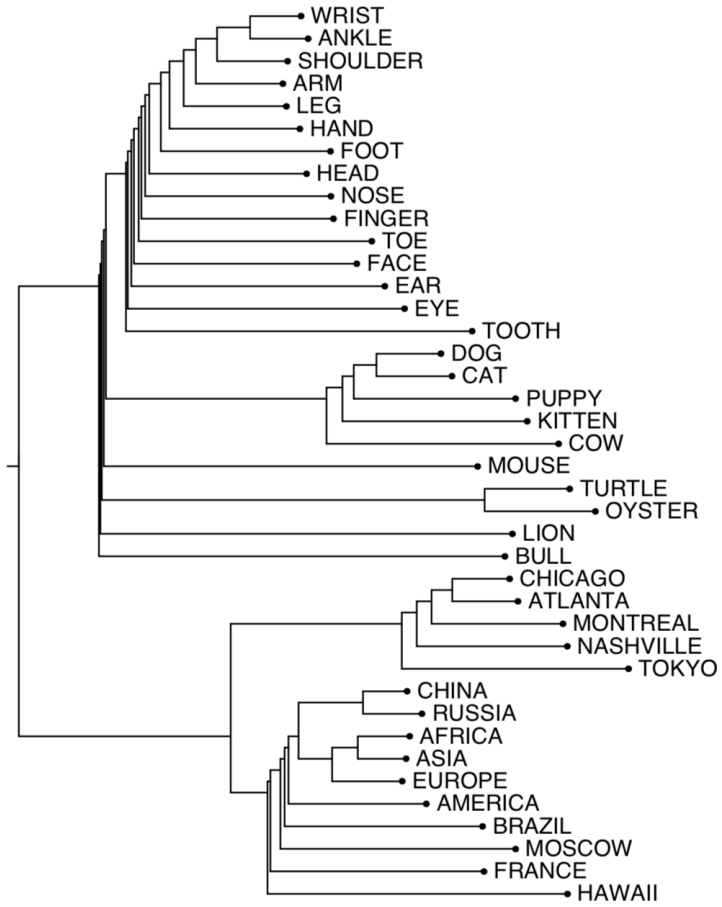
Window size $\pm$ 1 token

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

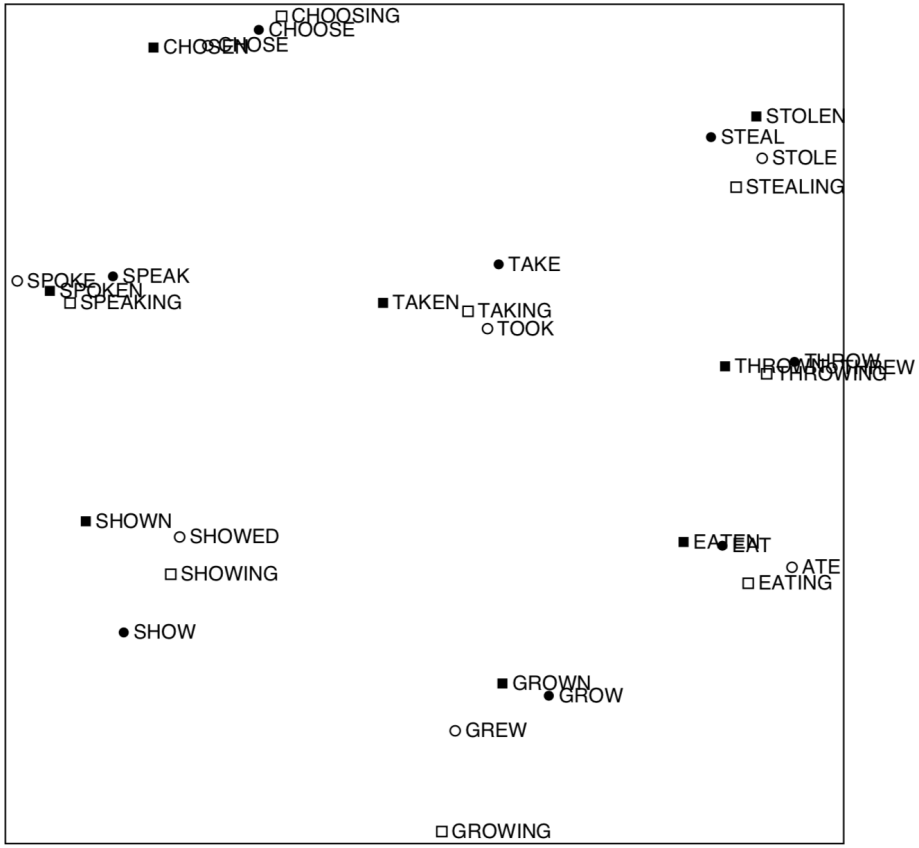
In practice, use singular value decomposition for dimenstionality reduction and limit sparcity.

The direct approach illustrated

Semantic



Syntax



Rohde et al., 2005. An improved model of semantics similarity based on lexical co-occurrence.

The many variants of co-occurrence counting

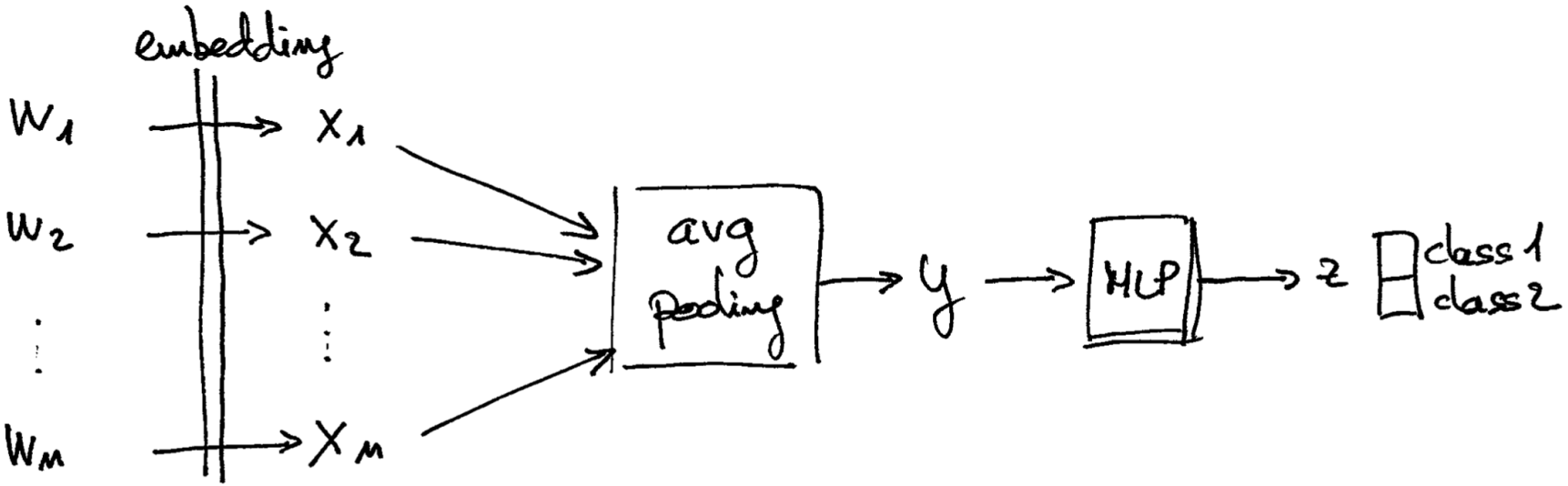
Many variants of the co-occurrence scheme are possible

- weighting w.r.t. the location in the window
- downplay function words
  $\rightarrow$ discard function words, cap counts e.g. at 100, etc.
- use (positive) correlation coefficients instead of counts

Yet, **not too practical** because

- the size of the matrix and the correspinding computational cost
- the cost of adding new words

The notion of word embedding



$x_i = W \cdot 1hot(w_i) \in \mathbb{R}^d$ $\quad y = \frac{1}{m}\sum x_i$ $\quad z = softmax(tanh(Ay+b)) \in \mathbb{R}^2$

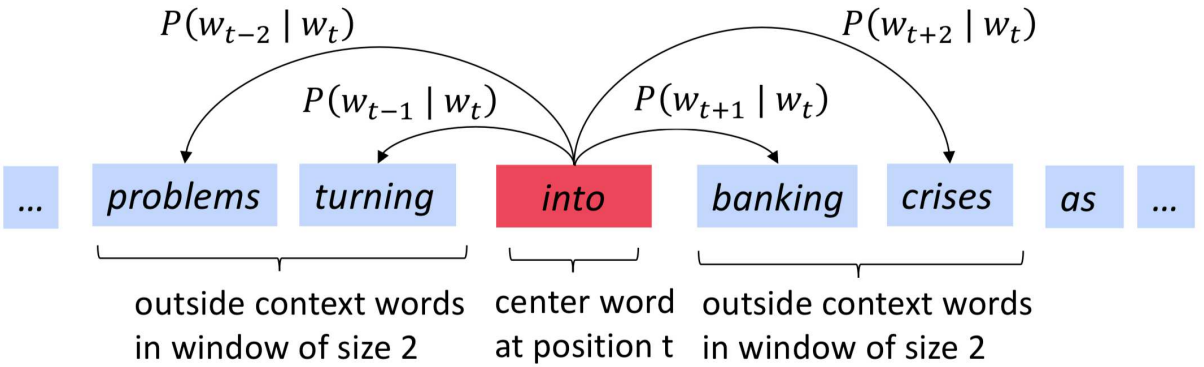
That's embedding but is it distributional semantics?

The *word2vec* principle

Directly learn word vectors without looking at co-occurrences!

The model comes in two flavors:

1. vectors to “predict” center word from surrounding words (cbow)
2. vectors to “predict” surrounding words from center word (skip-gram)



For the skip-gram model, we thus seek vectors that will maximize

$$J(\theta) = \sum_{t} \sum_{j \in [-m,m], j \neq 0} \ln p(w_{t+j}|w_t)$$

Towards the skip-gram maths

If we denote

- u_o = vector representing the context/output word o
- v_c = vector representing the center word c

we define the probability of the context knowing the word as a logistic regression (or softmax function), i.e.

$$p(o|c) = \frac{\exp(u'_o v_c)}{\sum_{w \in \mathcal{V}} \exp(u'_w v_c)}$$

and hence

$$J(\theta) = \sum_{t,j} \left(u'_{w_{t+j}} v_{w_t} - \ln \left(\sum_v u'_v v_{w_t} \right) \right)$$

This is a mess to optimize!!!!

The skip-gram model approximation

- Approximate the objective function and use negative sampling, i.e.

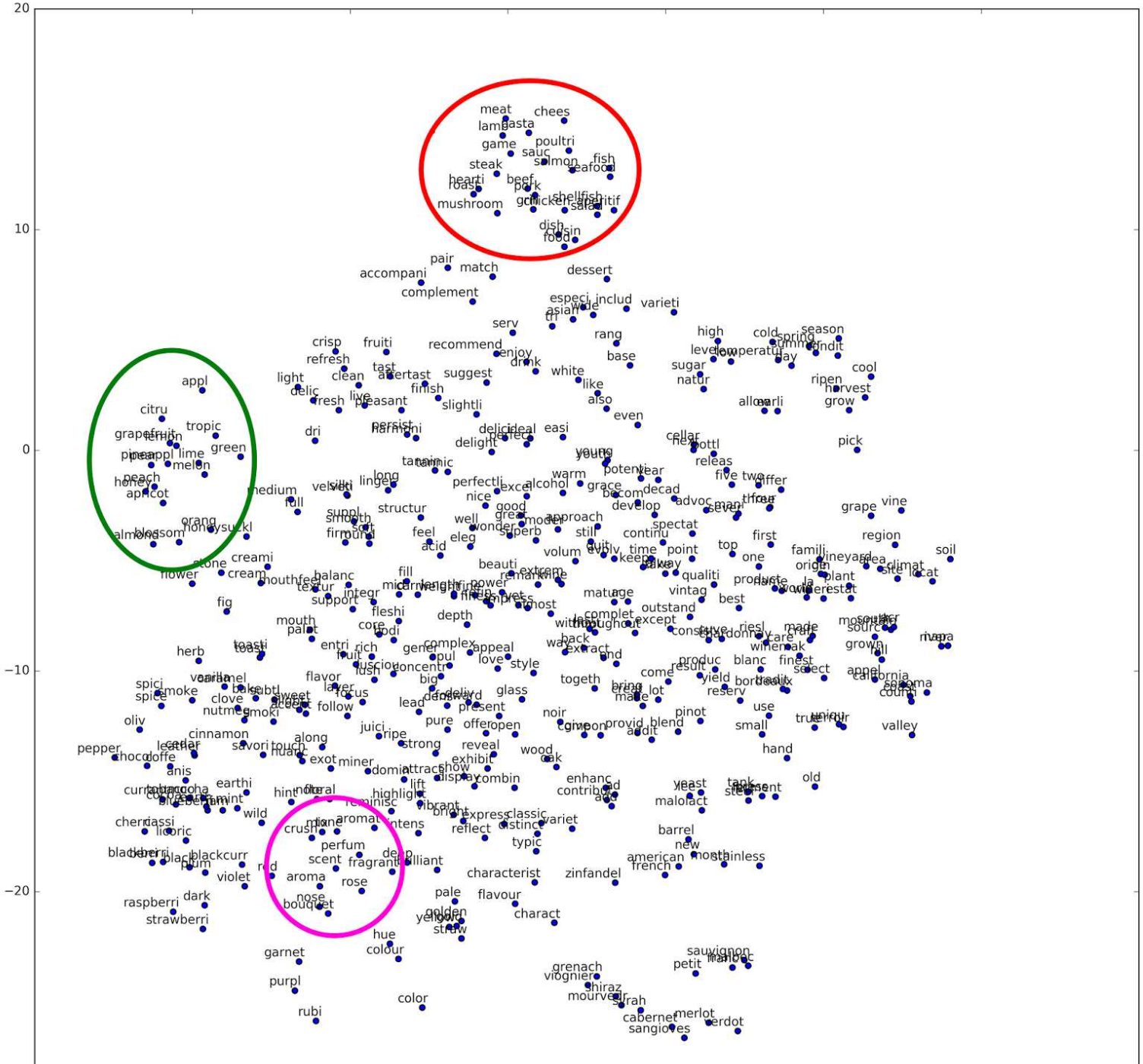
$$J(o, c, \theta) = \sigma(u'_o v_c) + \sum_{w \sim P[w]} \sigma(-u'_w v_c)$$

  with a sum over a small number of negative samples

- Minimax principle
  - maximize value for co-occurring words $\rightarrow \sigma(u'_o v_c)$
  - minimize value for non co-occuring ones $\rightarrow \sigma(-u'_w v_c)$
  - note that $\sigma(-x) = 1 - \sigma(x)$

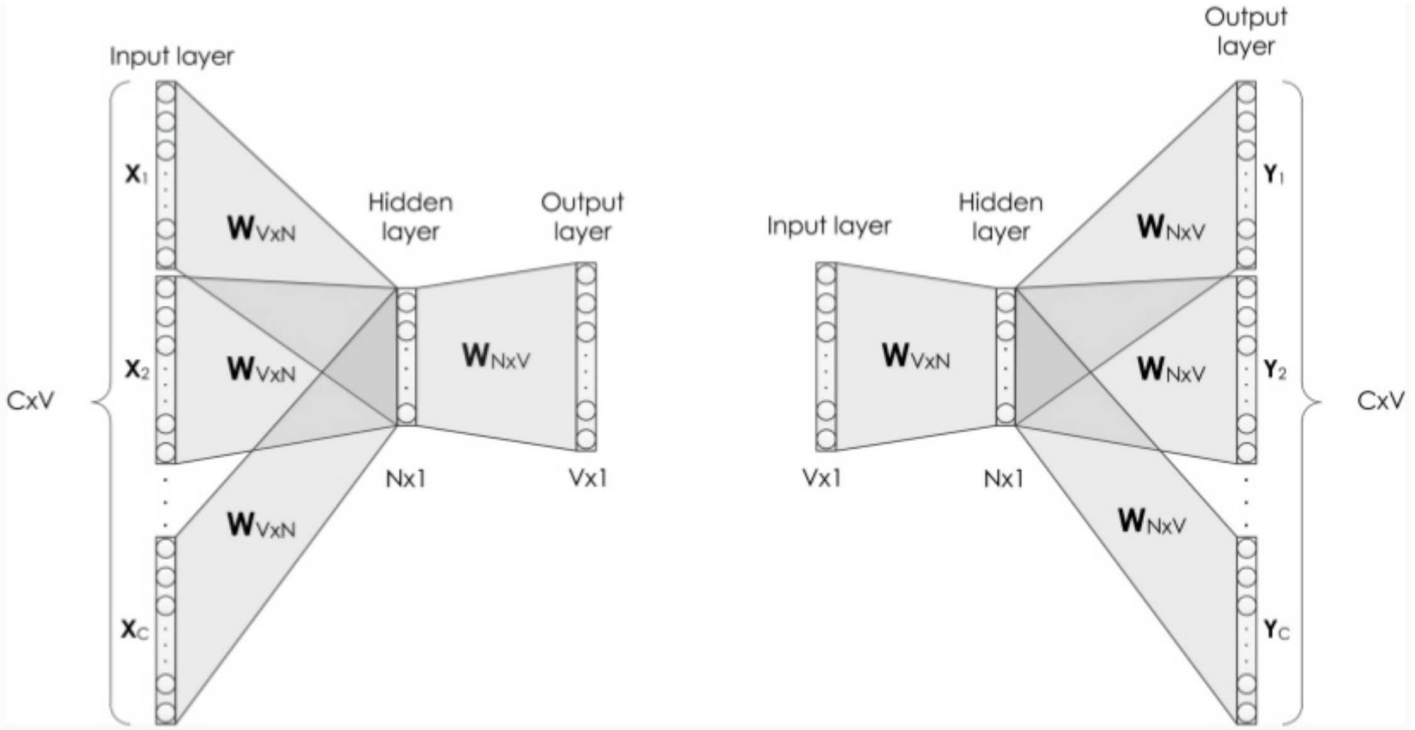
In the end (after training properly), we have $u'_a v_b \simeq \ln P[a|b]$ for two arbitrary words a and b.

Output of skip-gram



What relation with neural networks?

cbow (fast, small data, syntactic) vs. skip-gram (accurate, large data, semantic)



Global vectors

Mikolov's *word2vec* (and other related methods not presented here) are well-behaved

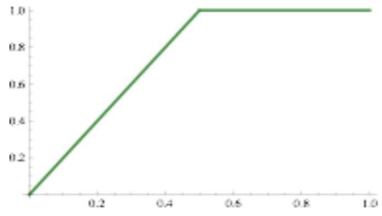
- generalizes pretty well to various tasks
- captures complex linguistic patterns

but also fail in some aspects

- not using global statistics on corpus
- training can be slow and requires very large amounts of data

GloVE combines "the best of both world" with

$$J(\theta) = \frac{1}{2} \sum_{i,j \in \mathcal{V} \times \mathcal{V}} f(P_{ij})(u_i' v_j - \ln P_{ij})^2$$

$f \sim$ 

Pros: fast and efficient, scalable, need for less data

Pennington et al., 2014. Glove: Global vectors for word representation.

Global vectors illustrated

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Pennington et al., 2014. Glove: Global vectors for word representation.

Similarity evaluation

- Use dot product (cosine similarity) to predict similarity between two words
- Correlates with human judgement

love	sex	6.77
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	keyboard	7.62
computer	internet	7.58
...		

`http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/`

See François Torregrossa *et al*. A survey on training and evaluation of word embeddings. International Journal of Data Science and Analytics, 2021.

fasttext: word embedding through n-gram embedding

$\Rightarrow$ embed subword units rather than words

- decompose word as bag of n-gram with $n \in [3, 6]$
  - where = (<wh + whe + her + ere + re> + <whe + ...)
- word embeding is the sum of subword embeddings
- similarity between w and c is given by

$$s(w, c) = \sum_{g \in \mathcal{G}_w} z_g v_c$$

- optimization of subword embeddings similar to word2vec

Bojanowski et al., 2017. Enriching word vectors with subword information.

Model resources on the web

- The word embedding repository `http://vectors.nlpl.eu/repository`
- Fasttext repository `https://fasttext.cc/docs/en/english-vectors.html`
- GloVe models `https://nlp.stanford.edu/projects/glove`
- Pretrained models in NLP pipelines, e.g., gensim model zoo

Practical examples

```
import gensim.downloader
> print(list(gensim.downloader.info()['models'].keys()))
['fasttext-wiki-news-subwords-300',
 'conceptnet-numberbatch-17-06-300',
 'word2vec-ruscorpora-300',
 'word2vec-google-news-300',
...

> glove_vectors = gensim.downloader.load('glove-twitter-25')
> glove_vectors.most_similar('twitter')
[('facebook', 0.948005199432373),
 ('tweet', 0.9403423070907593),
 ('fb', 0.9342358708381653),
...
```

Representation and classification of documents

Representing documents: what for?

Documents can be (almost) everything ... that contains text

- book, chapter, paragraph, etc.
- newspaper/web article
- tweet, blog or facebook post

Most document representations seek to representing a documents as a fixed-dimension *feature vector* further used for, e.g.,

- topic classification
- polarity and sentiment detection
- comparison of documents (information retrieval)

- often based on *the bag hypothesis*
  = order of words does not matter
- might implement selection of relevant terms

A naive Bayes approach to document classification

Simplify the maximum a posteriori rule $p(c|d) = p(d|c)p(c)$ considering each $w \in d = \{w_1, \ldots, w_{n_d}\}$ independently, i.e.,

$$p(d|c) = \prod_{i=1}^{n_d} p(w_i|c)$$

T. Bayes (c. 1702–1761)

Estimating conditional word occurrence probabilities $p(w|c)$ a from large corpora $D = \cup D_c$, e.g.,

$$p(w|c) = \frac{\sum_{d \in D_c} \delta(w,d)}{\sum_{v \in V} \sum_{d \in D_c} \delta(v,d)} \quad \text{or} \quad p(w|c) = \frac{\sum_{d \in D_c} n(w,d)}{\sum_{d \in D_c} n_d}$$

or variants of this (e.g., Laplace, Dirichlet)

Often assuming a prior $\mathbf{p}|\alpha \rightsquigarrow \mathrm{Dir}(\alpha)$ so that posterior probability $\mathbf{p}|\alpha, X$ is also a Dirichlet distribution (facilitate estimation)

ENSAI École nationale de la statistique et de l'analyse de l'information

The naive Bayes approach illustrated

1. class = love, content = {aimer: 5, manger: 0, Paul: 1, Virignie: 1, je: 5}
2. class = love, content = {aimer: 3, manger: 0, Paul: 0, Virignie: 0, je: 4}
3. class = food, content = {aimer: 0, manger: 2, Paul: 0, Virignie: 1, je: 5}
4. class = food, content = {aimer: 2, manger: 2, Paul: 0, Virignie: 0, je: 3}

For class 'love', we have:

$$P[\text{aimer}] = \frac{\sum_{d \in D_c} \delta(\text{aimer}, d)}{\sum_{v \in V} \sum_{d \in D_c} \delta(v, d)} = \frac{2}{6} \quad \text{or} \quad P[\text{aimer}] = \frac{\sum_{d \in D_c} n(\text{aimer}, d)}{\sum_{d \in D_c} n_d} = \frac{8}{19}$$

and for class 'food'

$$P[\text{aimer}] = \frac{\sum_{d \in D_c} \delta(\text{aimer}, d)}{\sum_{v \in V} \sum_{d \in D_c} \delta(v, d)} = \frac{1}{6} \quad \text{or} \quad P[\text{aimer}] = \frac{\sum_{d \in D_c} n(\text{aimer}, d)}{\sum_{d \in D_c} n_d} = \frac{2}{15}$$

The naive Bayes approach illustrated

1. class = love, content = {aimer: 5, manger: 0, Paul: 1, Virignie: 1, je: 5}
2. class = love, content = {aimer: 3, manger: 0, Paul: 0, Virignie: 0, je: 4}
3. class = food, content = {aimer: 0, manger: 2, Paul: 0, Virignie: 1, je: 5}
4. class = food, content = {aimer: 2, manger: 2, Paul: 0, Virignie: 0, je: 3}

With the first estimator, we get in the end

class	P[aimer]	P[manger]	P[Paul]	P[Virginie]	P[je]
love	2/6	0/6	1/6	1/6	2/6
food	1/6	2/6	0/6	1/6	2/6

Assuming equal class prior, classify new document d = {aimer: 2, manger: 0, Paul: 0, Virignie: 1, je: 1} according to

$$P[d|\text{class=love}] = 0.5 * (2 * 2/6) * 1/6 * 2/6 \sim .0185$$

$$P[d|\text{class=food}] = 0.5 * (2 * 1/6) * 1/6 * 2/6 \sim .0093$$

Naive Bayes and regularization (aka smoothing)

Now classifying d = {aimer: 0, manger: 10, Paul: 1, Virignie: 0, je: 0} :

$$P[d|\text{class=love}] = 0.5 * (10 * 0) * 1/6 = 0$$
$$P[d|\text{class=food}] = 0.5 * (10 * 2/6) * 0 = 0$$

Need for smoothed probability estimates to avoid 0s, e.g,

$$p(w|c) = \frac{1 + \sum_{d \in D_c} n(w,d)}{|V| + \sum_{d \in D_c} n_d} \quad \text{or} \quad p(w|c) = \frac{\lambda P[w] + \sum_{d \in D_c} n(w,d)}{\lambda + \sum_{d \in D_c} n_d}$$

with $P[W] \rightsquigarrow \text{Dir}(\alpha)$.

Many variants such as subtractive and Good-Turing discounting

Why smoothing is so important? (because of Zipf)

Statistics on the newspaper Le Monde in 2003

r	n_r	token	r	n_r	token
1	227306	académisé	274928	1	pour
2	59053	gutturale	286277	1	une
3	28459	port-cros	287036	1	dans
4	17223	s'imputer	325378	1	a
5	11483	remariée	339432	1	un
6	8310	échangée	438658	1	du
7	6190	mastercard	494437	1	en
8	4901	délégitimer	591394	1	des
9	3744	teenage	638864	1	et
10	3072	diamonds	682522	1	à
11	2477	matta	684617	1	les
12	2022	cammas	836026	1	le
13	16462	collabos	1081822	1	la
14	7458	sidibe	1892396	1	de

[courtesy of François Yvon]

George K. Zipf

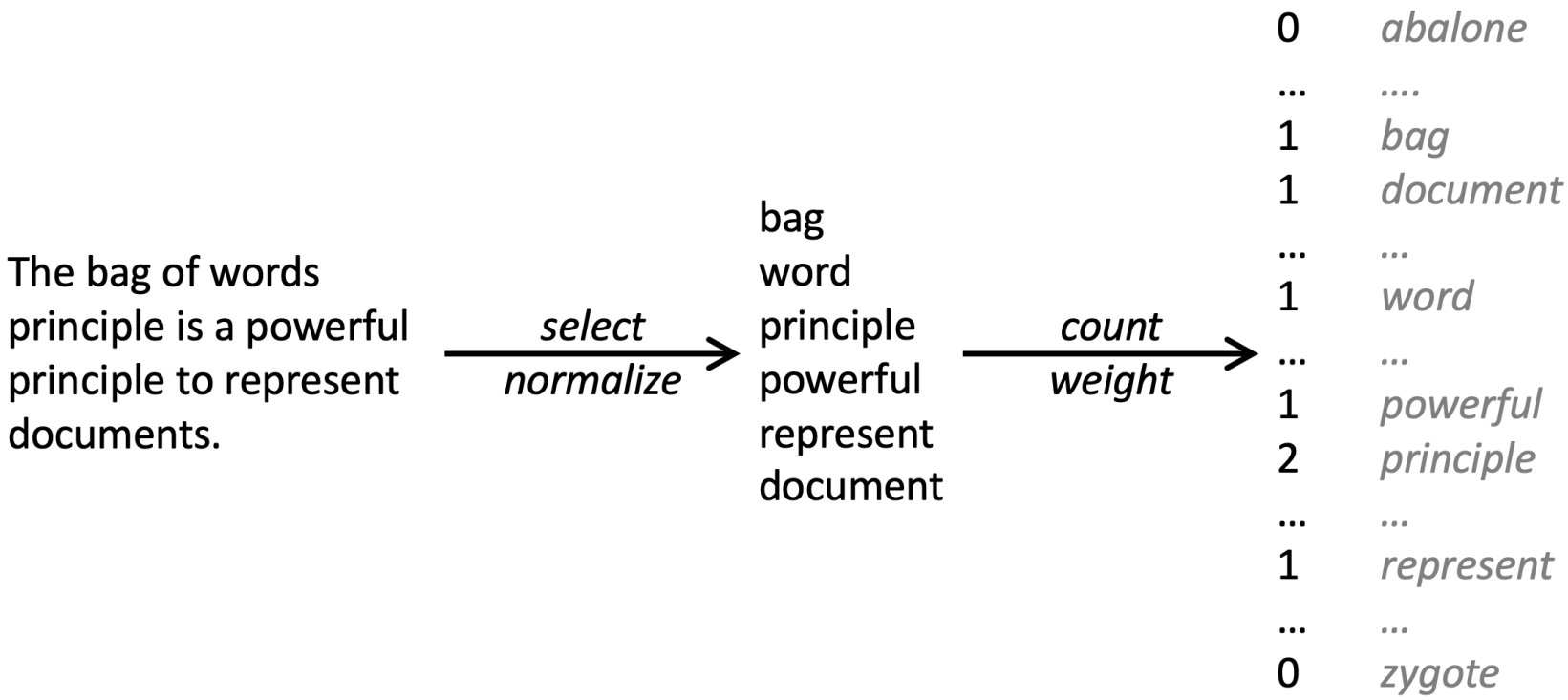
1902–1950

Frequent events are rare and rare events are frequent, which roughly translate to

$$\text{rank}(w)\text{freq}(w) = \text{cst}$$

Explicit bag-of-words: the vector space model

Assign a weight to each possible word in a fixed-size vocabulary according to its appearance in the document



BoW: choosing and weighting tokens/words

Step 1. Selection of terms for the vocabulary

- tokenization and normalization
- lemmatization, stemming ... or none
- selection of relevant terms
  - frequency, POS (NVA), stop lists
  - might be crucial (retrieval) ... or not (classification)

Step 2. Assignement of weights for each token

- binary indicator $\delta(w, d)$ $\longrightarrow$ aka 1-hot encoding
- number of occurrences $n(w, d)$ of word w in document d
- frequency of occurrence $n(w, d) / \sum_{v \in V} n(v, d)$

$\Rightarrow$ issue with frequent words, typically non-informative function words

tf-idf weighting for bag-of-words

Normalizing term frequency to downplay frequent function words that bear limited information in most cases

$$f(w,d) = \underbrace{\left(\frac{n(w,d)}{\sum_{v \in V} n(v,d)} \right)}_{\text{term frequency}} \underbrace{\log \left(\frac{\sum_{d' \in D} \delta(w,d')}{N} \right)^{-1}}_{\text{inverse document frequency}}$$

where D is a collection of N documents to compute prior probability of how likely w is to appear in a document

$\Rightarrow$ can be extended in a number of ways mixing local weight (term frequency), global weight (inverse document frequency) and possibly a normalization weight

tf-idf illustrated

1. class = love, content = {aimer: 5, manger: 0, Paul: 1, Virignie: 1, je: 5}
2. class = love, content = {aimer: 3, manger: 0, Paul: 0, Virignie: 0, je: 4}
3. class = food, content = {aimer: 0, manger: 2, Paul: 0, Virignie: 1, je: 5}
4. class = food, content = {aimer: 2, manger: 2, Paul: 0, Virignie: 0, je: 3}

- aimer appears in 3 documents out of |D|=4

$$\text{idf}(\text{aimer}) = \log \left(\frac{\sum_{d' \in D} \delta(\text{aimer}, d')}{|D|} \right)^{-1} = \log(4/3) \simeq 0.125$$

- aimer appears 5 times in document d_1

$$\text{tf}(\text{aimer}, d_1) = \frac{n(\text{aimer}, d_1)}{\sum_{v \in V} n(v, d_1)} = 5/12 \simeq 0.417$$

- weight of aimer in document $d_1 = \text{tf}(\text{aimer}, d_1)\text{idf}(\text{aimer}) \simeq 0.052$

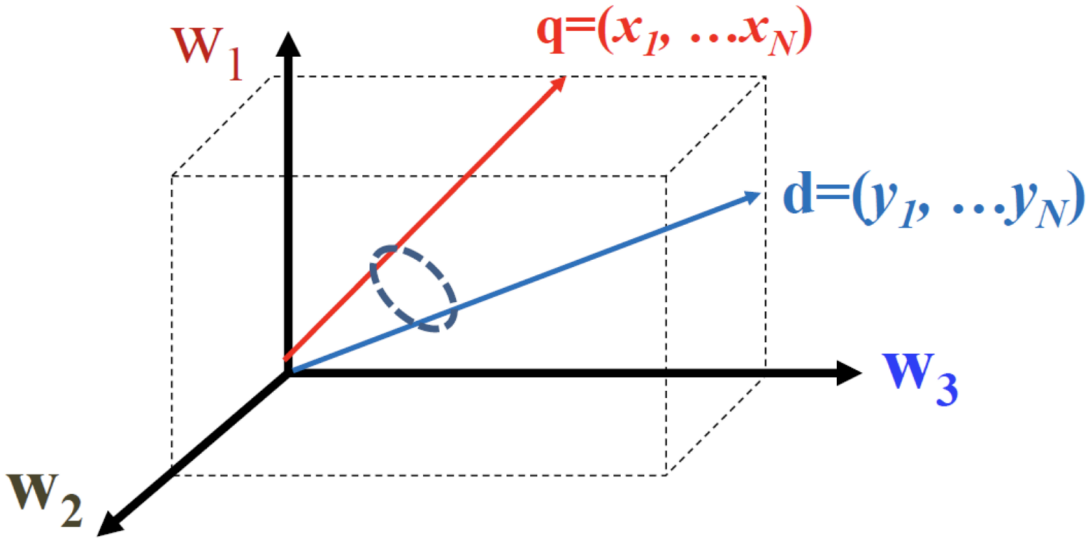
tf-idf illustrated

1. class = love, content = {aimer: 5, manger: 0, Paul: 1, Virignie: 1, je: 5}
2. class = love, content = {aimer: 3, manger: 0, Paul: 0, Virignie: 0, je: 4}
3. class = food, content = {aimer: 0, manger: 2, Paul: 0, Virignie: 1, je: 5}
4. class = food, content = {aimer: 2, manger: 2, Paul: 0, Virignie: 0, je: 3}

	idf	doc1	doc2	doc3	doc4
n_w		12	7	8	7
aimer	0.125	0.052	0.054	0	0.036
manger	0.301	0	0	0.077	0.089
Paul	0.602	0.050	0	0	0
Virginie	0.301	0.025	0	0.038	0
je	0	0	0	0	0

The vector space model (information retrieval)

Documents (and possibly queries in IR) are represented in a vector space over which we can define a metric



borrowed from Tonny Kwon's blog

dot product $x \cdot y = \sum_i x_i y_i$

ℓ^2 norm $||x - y|| = \sqrt{\sum_i (x_i - y_i)^2}$

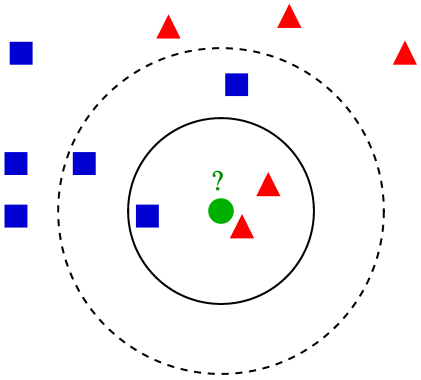
cosine $\text{cosine}(x, y) = \dfrac{x \cdot y}{||x||\ ||y||}$

Classification in the vector space model

All flavors of feature-based classifiers can be used with the bag-of-word representation, e.g.,

- k-nearest neighbors in the vector space
- logistic regression

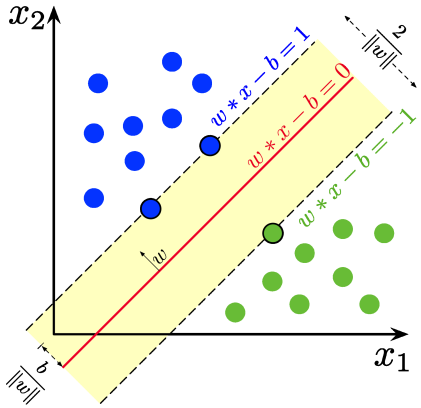
$$p(c|d) = \frac{1}{1 + \exp\left(\alpha_0 + \sum_{w \in d} \alpha_w f(w,d)\right)}$$



© Antti Ajanki

- support vector machines

$$\widehat{c} = \text{sign}\left(\sum_{w \in V} \alpha_w f(w,d) - \alpha_0\right)$$



© Larhmam

- feed-forward neural nets

Note: Not limited to binary classification obviously!

Latent variable variants of the BoW model

Some of the downsides of the BoW approach

- no ordering of words — that's the price to pay
- very sparse representation, high dimension
- distributional semantics is absent (cat $\neq$ kitty)
- cannot compare documents with no words in common

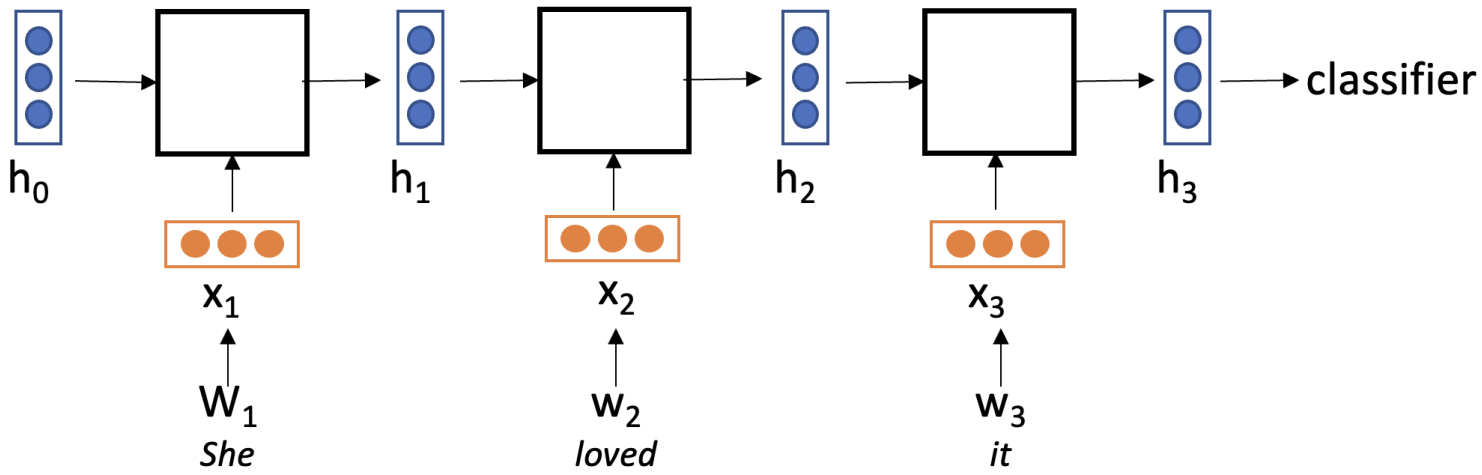
Seek small, compact and efficient representations that can be directly used rather than the BoW vector

Option 1: Latent semantic indexing with PCA/SVD

Option 2: Latent Dirichlet allocation

Embedding documents with recurrent neural networks

h_i **= summary of document up to** $w_i \Rightarrow h_n$ **= summary of document**



Example of an Elman recurrent network

Embedding layer $x_i = c(w_i)$

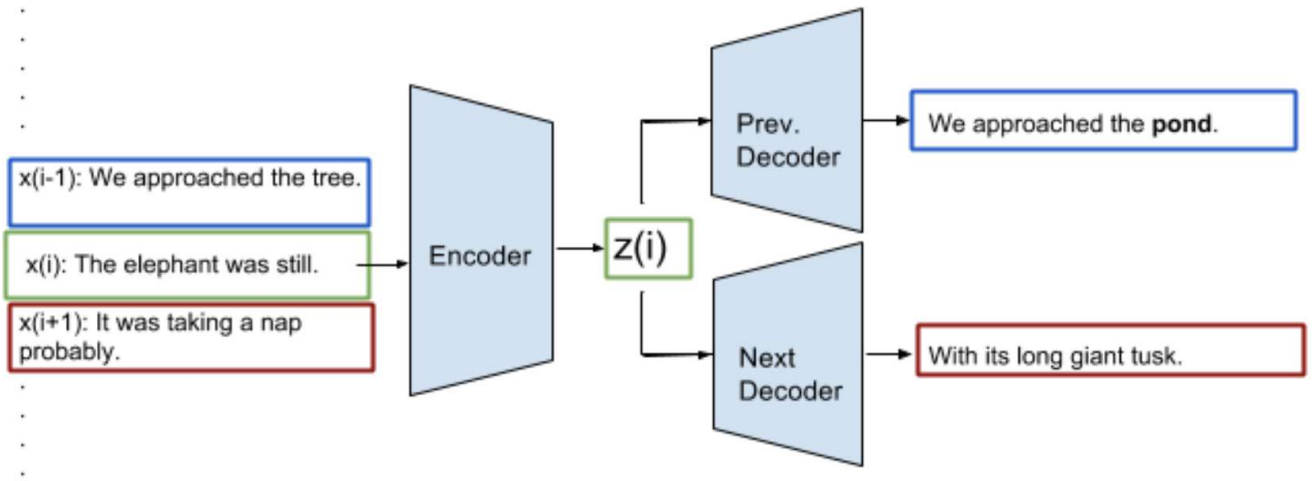
Merging layer $y_i = x_i + h_{i-1}$

State prediction $h_i = \sigma(U y_i)$ or $\sigma(U_c c_i + U_h h_{i-1}))$

Shades of RNN training for document embedding

The general idea of recurrent neural network for document embedding can be cast in a number of ways, e.g.,

- task oriented training
  $\rightarrow$ freezing or not the embedding layer
- auto-encoding and its many variants

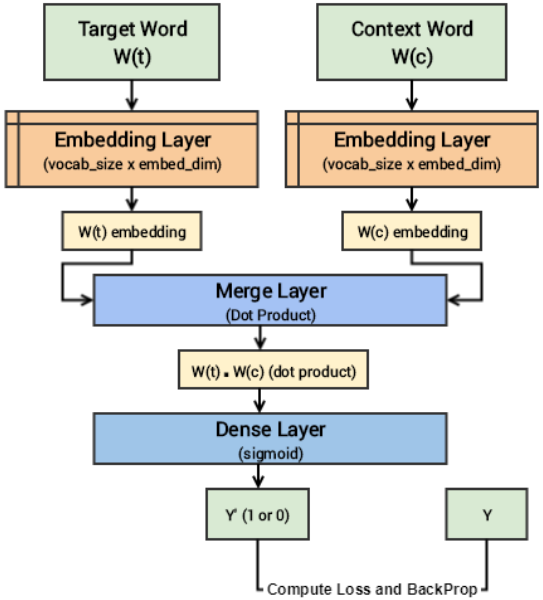


- pre-trained RNN language models (we'll see that next)
- etc.

[picture borrowed from Sanyam Agarwal's thoughts on skip-thought]

Classification with document embeddings

- document embedding = vector space
  $\Rightarrow$ any classifier does the job
- most often used with feed-forward nets
  - fixed pretrained embeddings
  - retrain embedding layer
- can also be used for ranking documents in IR with specific loss functions (e.g., triplet loss)



$$L(a,p,n) = \max\left(||f(a) - f(p)||^2 - ||f(a) - f(n)||^2 + \alpha, 0\right)$$

Language modeling

Language models: what for?

Define a probability distribution over sentences from a vocabulary V

$P[\text{This sentence is very likely}]\ ?$ $P[\text{Sentence this not very likely is}]\ ?$

Very useful for language identification, speech recognition and translation

- parce qu'il était vs. pace qu'il été vs. parce qu'il étais
- avocat $\longrightarrow$ lawyer vs. avocado

Also very useful for prediction (e.g., spell checking, language generation) if able to provide a distribution probability for the next word

$\rightarrow$ une baguette de ????? $\rightarrow$ Jean aime ????

probabilistic grammar	$\longrightarrow$	powerful but too complex and not robust
n-gram and neural models	$\longrightarrow$	basic but robust and efficient

$\Longrightarrow$ Often boils down to assigning a (non-zero) probability to a(ny) word occurrence given the occurrences of the previous and/or following words

First step: define the vocabulary (already discussed)

In practice, we first need to define the vocabulary V over which the probability distribution over sentences is defined, which is no trivial task – *cf*. thoughts on what's a word in the introduction part

- only consider the k most frequent words/tokens in a corpus $\mathcal{C}$
  $\longrightarrow$ approx. k^n parameters for a ngram LM!
- k typically in the range 64k - 200k in practice
- optimize tokenization to improve coverage
  $\longrightarrow$ URLs, numbers, units and other weird things
  $\longrightarrow$ hyphens (-) and apostrophe (') — prud'homme, d', l', s'
  $\longrightarrow$ locutions — ad_hoc, il_y_a, c'_est_à_dire, c'_est
  $\longrightarrow$ remove punctuation marks, quotes, etc.
- open *vs*. closed vocabulary LM: the <*unk*> trick
- sub-word tokens (e.g., BPE, WordPiece)

Why is sentence probability an issue?

$$
\begin{aligned}
P[\text{Jean aime Marie qui aime Paul}] \quad = \quad & P[\text{Jean}] \times \\
& P[\text{aime}|\text{Jean}] \times \\
& P[\text{Marie}|\text{Jean aime}] \times \\
& P[\text{qui}|\text{Jean aime Marie}] \times \\
& P[\text{aime}|\text{Jean aime Marie qui}] \times \\
& P[\text{Paul}|\text{Jean aime Marie qui aime}]
\end{aligned}
$$

Use approximations of the *history* to simplify probability/score computation

$$
P[w_i|\underbrace{w_{i-1}\dots w_1}_{\textit{history}}] \simeq P[w_i|\underbrace{f(w_{i-1}\dots w_1)}_{h(w_i)}]
$$

The ngram approximation

$$P[w_i|w_1, w_2, \ldots, w_{i-1}] \doteq P[w_i|\underbrace{w_{i-n+1}, \ldots, w_{i-1}}_{n-1 \text{ words}}]$$

$$P[\text{Jean aime Marie qui aime Paul}] =$$

unigram (n=1)

$P[\text{Jean}] \times$
$P[\text{aime}|\sout{\text{Jean}}] \times$
$P[\text{Marie}|\sout{\text{Jean aime}}] \times$
$P[\text{qui}|\sout{\text{Jean aime Marie}}] \times$
$P[\text{aime}|\sout{\text{Jean aime Marie qui}}] \times$
$P[\text{Paul}|\sout{\text{Jean aime Marie qui aime}}]$

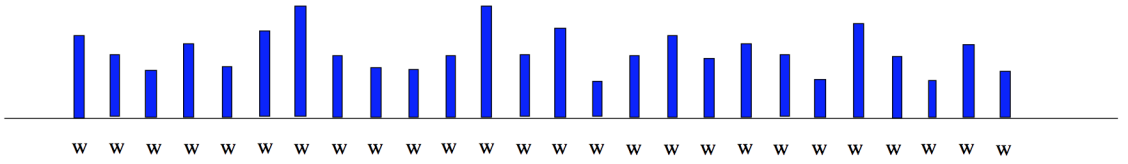
bigram (n=2)

$P[\text{Jean}] \times$
$P[\text{aime}|\text{Jean}] \times$
$P[\text{Marie}|\sout{\text{Jean}}\ \text{aime}] \times$
$P[\text{qui}|\sout{\text{Jean aime}}\ \text{Marie}] \times$
$P[\text{aime}|\sout{\text{Jean aime Marie}}\ \text{qui}] \times$
$P[\text{Paul}|\sout{\text{Jean aime Marie qui}}\ \text{aime}]$

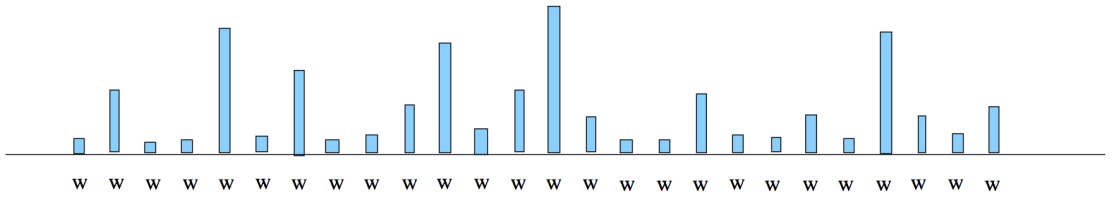
From the computer point of view: a set of tables...

Nothing but a *set of distribution tables over words in* V for each possible history.

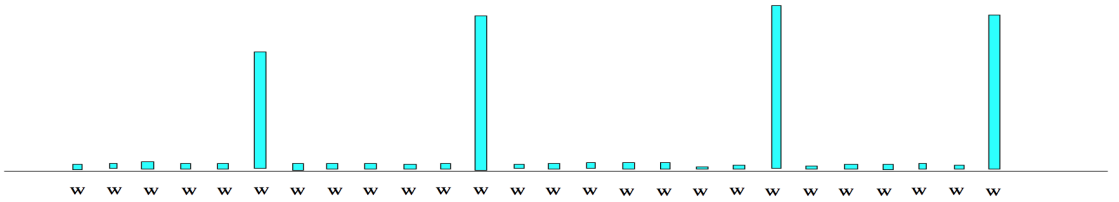
ϵ



Jean



Jean aime



... or a stochastic graph

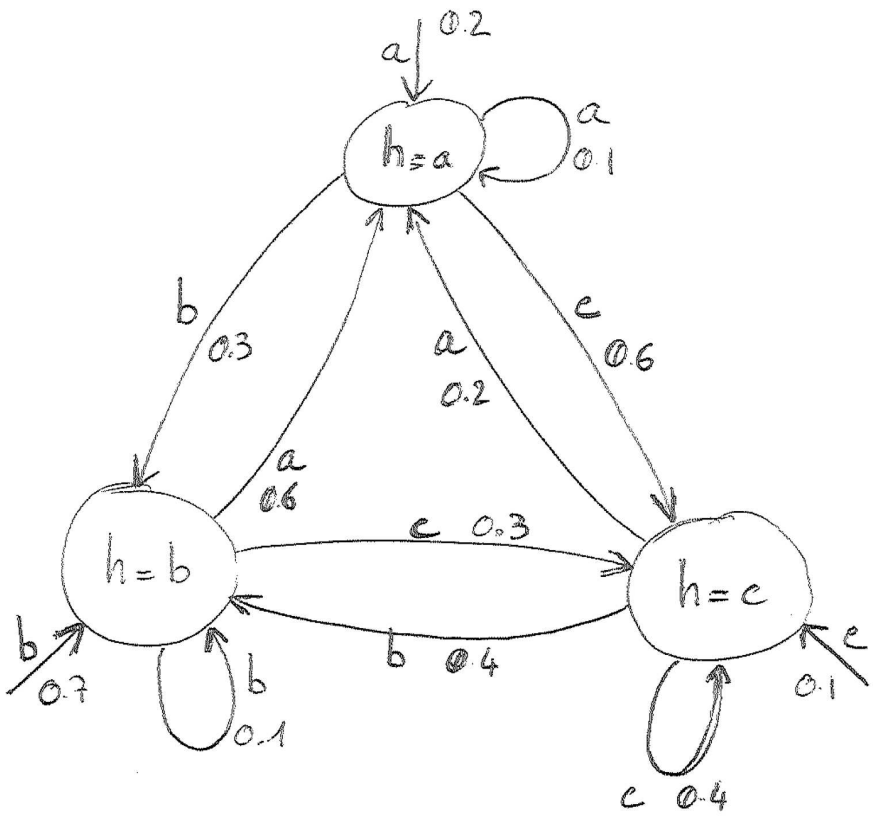
h	w	p
ϵ	a	0.2
ϵ	b	0.7
ϵ	c	0.1

h	w	p
a	a	0.1
a	b	0.3
a	c	0.6

h	w	p
b	a	0.6
b	b	0.1
b	c	0.3

h	w	p
c	a	0.2
c	b	0.4
c	c	0.4

table view



graph view

$$
\begin{aligned}
P[abaca] &= P[a] \times P[b|a] \times P[a|b] \times P[c|a] \times P[a|c] \\
&= 0.2 \times 0.3 \times 0.6 \times 0.6 \times 0.2 = 0.00432
\end{aligned}
$$

Text generation with ngram models

Sampling from the distributions $P[w|h]$, where each new sample changes the history.

For example, with a trigram model:

1. choose initial word w_1 according to the distribution $P[u|\epsilon]$
2. choose next word w_2 according to $P[u|w_1]$
3. for $i = 3$ to the number of words you want
4. choose w_i according to $P[w_i|w_{i-2}w_{i-1}]$
5. end for

Similar to a random walk in the LM graph ... but not very useful indeed

Markov approximation of language

- **Order 1**: REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE T
- **Order 2**: THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPETED

[courtesy of F. Coste]

Evaluation of language models: the perplexity

Perplexity is directly linked to likelihood of the samples. If we consider a sample text $w = \{w_1, \ldots, w_k\}$, we can measure its perplexity w.r.t. the bigram model distribution P_q as

$$\begin{aligned} P(q, w) &= 2^{-\frac{1}{k}\sum_{i=1}^{k} \log_2(P_q[w_i|w_{i-1}])} = 2^{-\frac{1}{k}\log_2(P_q(w_1, \ldots, w_k))} \\ &= \left(2^{\log_2(P_q(w_1, \ldots, w_k))}\right)^{\frac{-1}{k}} = \sqrt[k]{\frac{1}{P_q(w_1, \ldots, w_k)}} \end{aligned}$$

$\longrightarrow$ you can generalize that to a n-sample/a whole corpus

A few things to remember about perplexity

- the smaller, the better
  - $\rightarrow$ can in fact be interpreted as the average rank of the word actually following a given history h given guesses ranked according to $P_q[.|h]$
- always measured w.r.t. a given corpus of texts (so cannot be compared if different corpora are used)

Estimation: the theory

The problem: From a (large) corpus of sentences $\mathcal{C} = \{S_1, \ldots, S_n\}$, estimate the set of probabilities $P[w|h]\ \forall h, w$ so that they reflect the reality of the data

The solution: maximum likelihood estimation given by

$$P_{\text{ML}}[w|h] = \frac{C(hw)}{C(h)} = \frac{C(hw)}{\sum_{v \in V} C(hv)} = \frac{\text{\# times you see hw}}{\text{\# times you see h}}$$

The maths: for a bigram model, the solution comes from the maximization of

$$\ln P[\mathcal{C}] = \sum_{i=1}^{n} \ln P[S_i] = \sum_{i=1}^{n} \left(\ln \left(P[w_1^i | \epsilon] \right) + \sum_{k=2}^{n_i} \ln \left(P[w_k^i | w_{k-1}^i] \right) \right)$$

with the constraints that $\forall v \in V$ we have $\sum_{w \in V} P[w|v] = 1$.

Estimation: a toy example

[borrowed again from F. Yvon]

$$011011110111110$$
$$1101110011101011$$

0	9	1	23	$p(0) = 9/32, p(1) = 23/32$
0 1	7	1 1	15	$p(0\|0) = 1/8, p(1\|0) = 7/8$
0 0	1	1 0	7	$p(0\|1) = 7/22, p(1\|1) = 15/22$
0 1 1	6	1 1 0	6	$p(1\|01) = 6/7, p(0\|01) = 1/7$
0 1 0	1	1 1 1	8	...
0 0 1	1	1 0 1	5	
		1 0 0	1	

Estimation: Zipf still sucks

Taille du corpus		10^6	4×10^6	3.9×10^7
# Phrases		37 831	187 892	1 611 571
# Mots		892 333	4 472 827	38 532 518
Unigrammes	Total	892 333	4 472 827	38532518
	Distincts	17 189	19 725	19 981
	Singletons	2 465	235	0
Bigrammes	Total	892 333	4 472 827	38 532 518
	Distincts	285 692	875 497	3 500 633
	Singletons	199 493	565 549	2 046 462
Trigrammes	Total	854502	4283935	36920518
	Distincts	587 985	2 370 914	1 4039 536
	Singletons	510 043	1 963 267	10 987 166

The principle of discounting

borrow probability mass from observed events pretending they were observed less than they actually were, and redistribute the probability mass to unobserved events (as cleverly as possible)

- Laplace smoothing (aka add-one): $c^*(hw) = c(hw) + 1$

$$P[w|h] = \frac{c(hw)+1}{\sum_{v \in V}(c(hv)+1)} = \frac{c(hw)+1}{c(h)+|V|}$$

- absolute discounting $\rightarrow$ same as Laplace but soustractive

$$P[w|h] \propto \max(c(hw) - \delta, 0))/\sum_u c(uw)$$

- Kneser-Ney discounting
  $\rightarrow$ refinement of absolute discounting where δ depends on h

- Good-Turing discounting (unseen events get probability n_1/N)

$$c^*(hw) = (c(hw)+1)\frac{n_{c(hw)+1}}{n_{c(hw)}}$$, n_r = # ngrams occurring r times

Estimation: a toy example with Laplace smoothing

[adapted from F. Yvon]

$$011011110111110$$
$$1101110011101011$$

0	9=>10	1	23=>24	$p(0) = 10/34, p(1) = 23/32$
0 1	7=>8	1 1	15=>16	$p(0\|0) = 2/10, p(1\|0) = 8/10$
0 0	1=>2	1 0	7=>8	$p(0\|1) = 8/24, p(1\|1) = 16/24$
0 1 1	6=>7	1 1 0	6=>7	$p(1\|01) = 7/8, p(0\|01) = 2/8$
0 1 0	1=>2	1 1 1	8=>9	...
0 0 1	1=>2	1 0 1	5=>6	
		1 0 0	1=>2	

Interpolation and back-off

Notation: h_k: history limited to k previous tokens

- standard linear interpolation

$$P_{\mathsf{I}}[w|h_n] = \sum_{i=0}^{n} \lambda_i P_{\mathsf{ML}}[w|h_i]$$

- recursive linear interpolation

$$P_{\mathsf{I}}[w|h_i] = \lambda_i(wh_i) P_{\mathsf{ML}}[w|h_i] + (1 - \lambda_i(wh_i)) P_{\mathsf{I}}[w|h_{i-1}]$$

- back-off

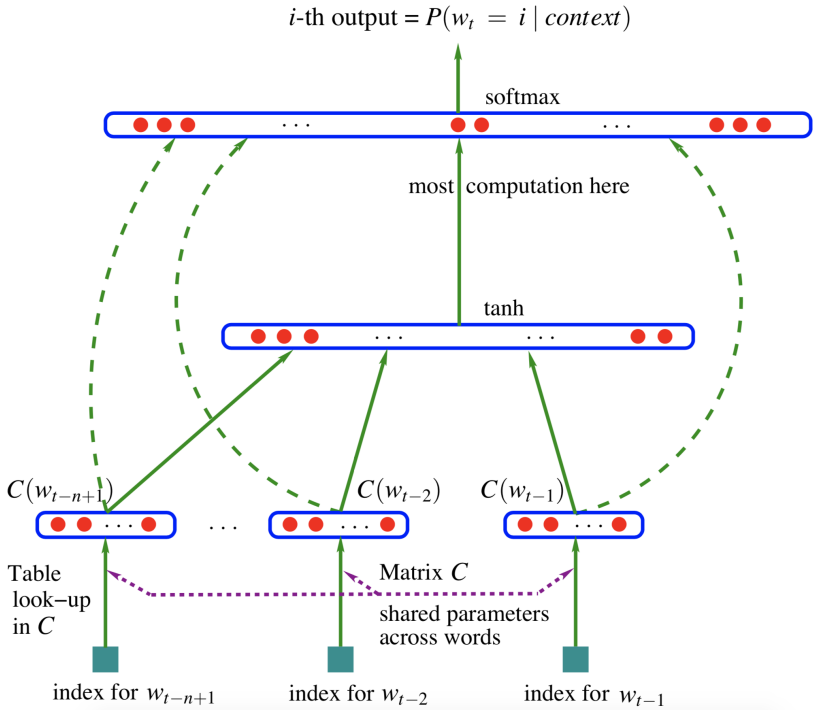
$$P_{\mathsf{bo}}[w|h] = \begin{cases} \frac{c^*(h_n w)}{C(h_n)} & \text{if} \quad c(h_n w) > 0 \\ \alpha(h_n) P_{\mathsf{bo}}[w|h_{n-1}] & \text{else} \end{cases}$$

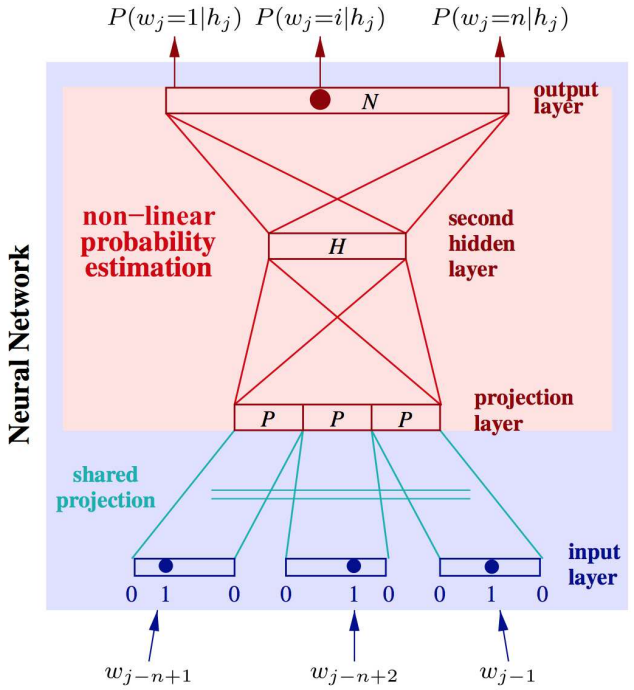
where $c^*(h_n w)$ is the biased count for $h_n w$ and $\alpha(h_n)$ s.t. $P_{\mathsf{bo}}[w|h]$ is a probability distribution.

Directly estimating probabilities with neural networks

Directly compute the n-gram probability with a feed-forward neural network train to yield

$$P[w_i|w_{i-1}\dots w_{i-n+1}] \doteq f(w_i,\dots,w_{i-n+1})$$





Yoshua Bengio et al. A Neural Probabilistic Language Model, Journal of Machine Learning Research, 2003

Holger Schwenk. Continuous space language models. Computer, Speech and Language, 21:492-518, 2007

Dissecting the seminal model of Bengio et al. 2003

Decomposition of $f(w_i, \ldots, w_{i-n+1})$ in two parts

1. an embedding $c()$ of words mapping token i to $c(i) \in \mathbb{R}^d$
2. a MLP with softmax loss to estimate

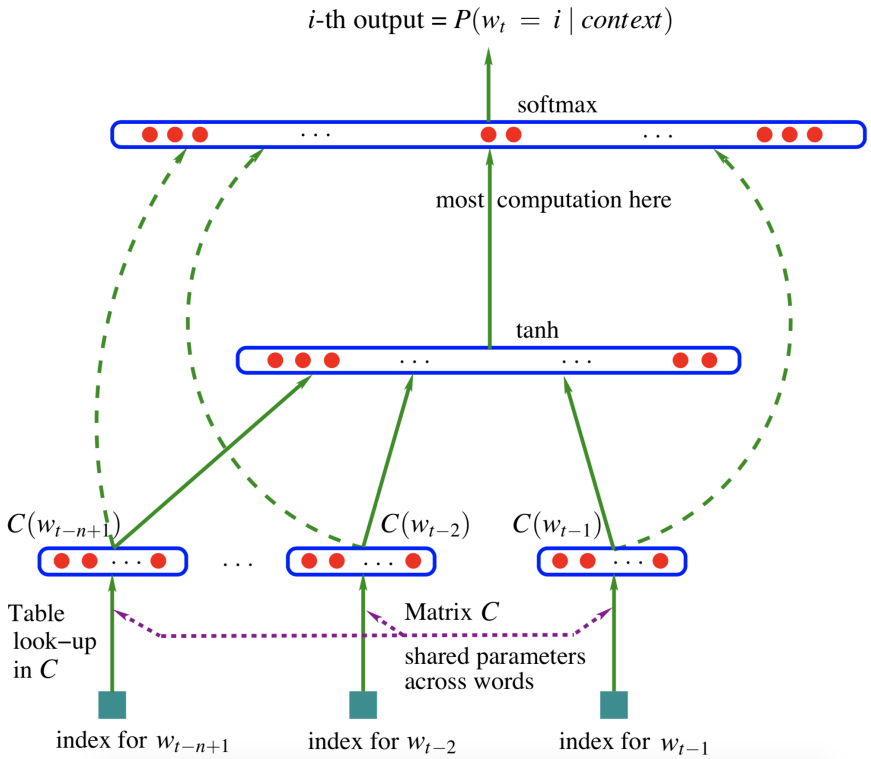
$$P[w_i | w_{i-1} \ldots w_{i-n+1}] = \frac{\exp(y_{w_i})}{\sum_k \exp(y_k)}$$

with

$$y = U \tanh(d + Hx)$$

or

$$y = b + Wx + U \tanh(d + Hx)$$

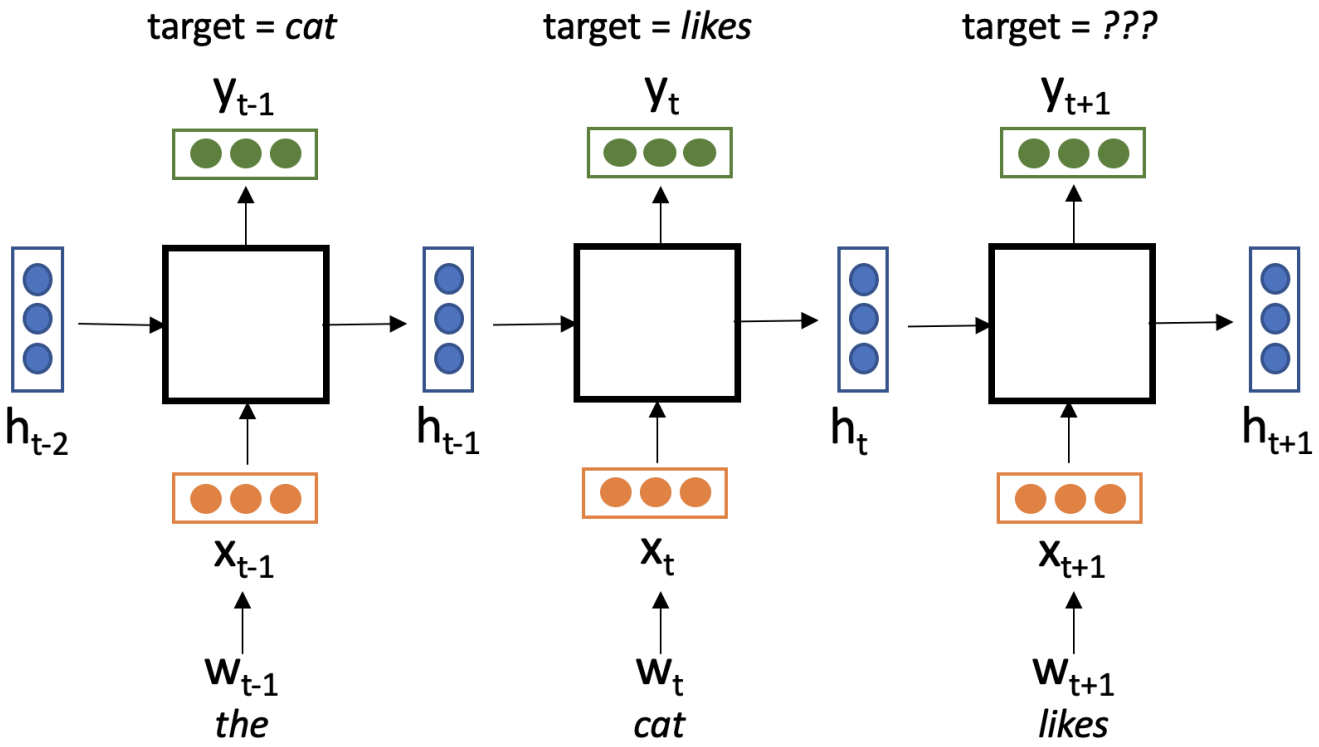


Objective function: $J = \frac{1}{T} \sum_t \log f(w_t^i, \ldots, w_{t-n+1}^i; \Theta) + R(\Theta)$

Modeling sentences with recurrent networks

Use the hidden state vector h_{i-1} as (a summary of) the history of word w_i to predict

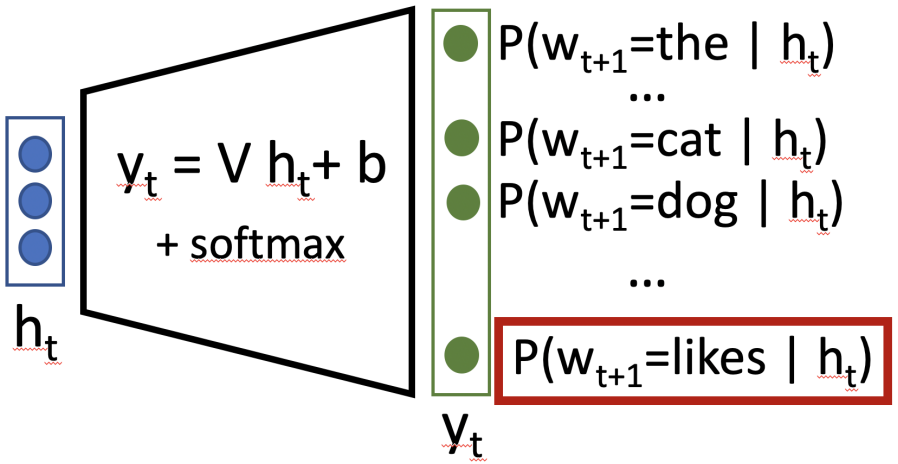
$$P[w_i | \underbrace{w_{i-1} \ldots w_1}_{\text{full history}}] \doteq f(w_i, h_{i-1})$$



Much less parameters than in the feed-forward approach!

Modeling sentences with recurrent networks

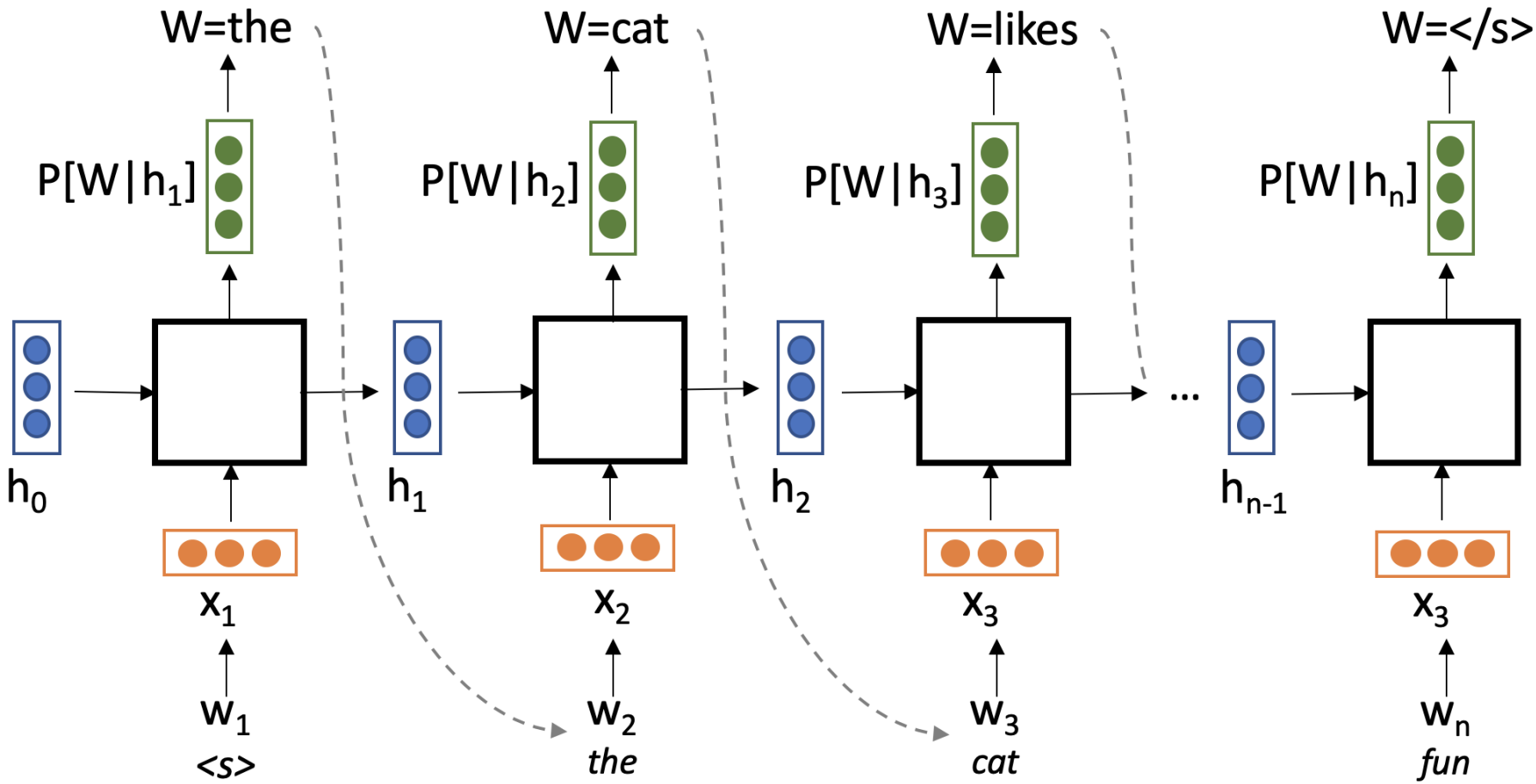
Predict a probability distribution over the vocabulary at each time step based on a feed-forward and softmax projection, where y_t is trained to hold probabilities $P[W_{t+1}|h_t]$ for all posible values of W_{t+1}



Parameters trained in a self-supervised manner with large amounts of texts so as to maximize the log-likelihood

$$J(\Theta) = \sum_t \log\left(y_t(\text{index}(w_{t+1}))\right)$$

Sampling from RNNs is trivial



Yet no control on semantics, cannot draw from $P[W_1, \ldots, W_n|X]$

Beam search decoding for better control

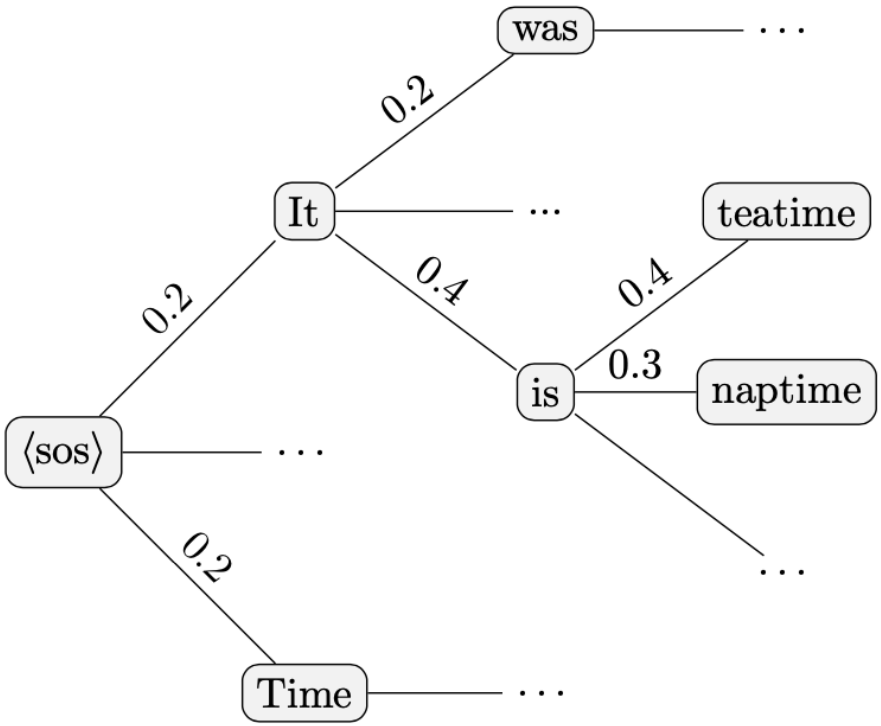


Figure 4: Toy example of beam search procedure with beam width $k = 2$. The search has been run for 3 steps and no hypothesis has terminated with ⟨eos⟩ yet. Edges are annotated with probabilities of tokens. Only the tokens after pruning to the top k hypotheses are shown.

borrowed from Ziang Xie's 2018 practical guide on neural text generation

For (gory) details on beam search decoding for text generation, see Sina Zarrieß et al., 2021. Decoding Methods in Neural Language Generation: A Survey

Wandering through side paths: tagging

Many NLP tasks can be cast as a *tagging* task, i.e., assigning a unique tag to each token in a sentence:

- part-of-speech tagging: predict morpho-syntactic tag

Le	chat	est	noir
DET	NMS	V3S	AMS

- entity detection: predict if within an entity or not

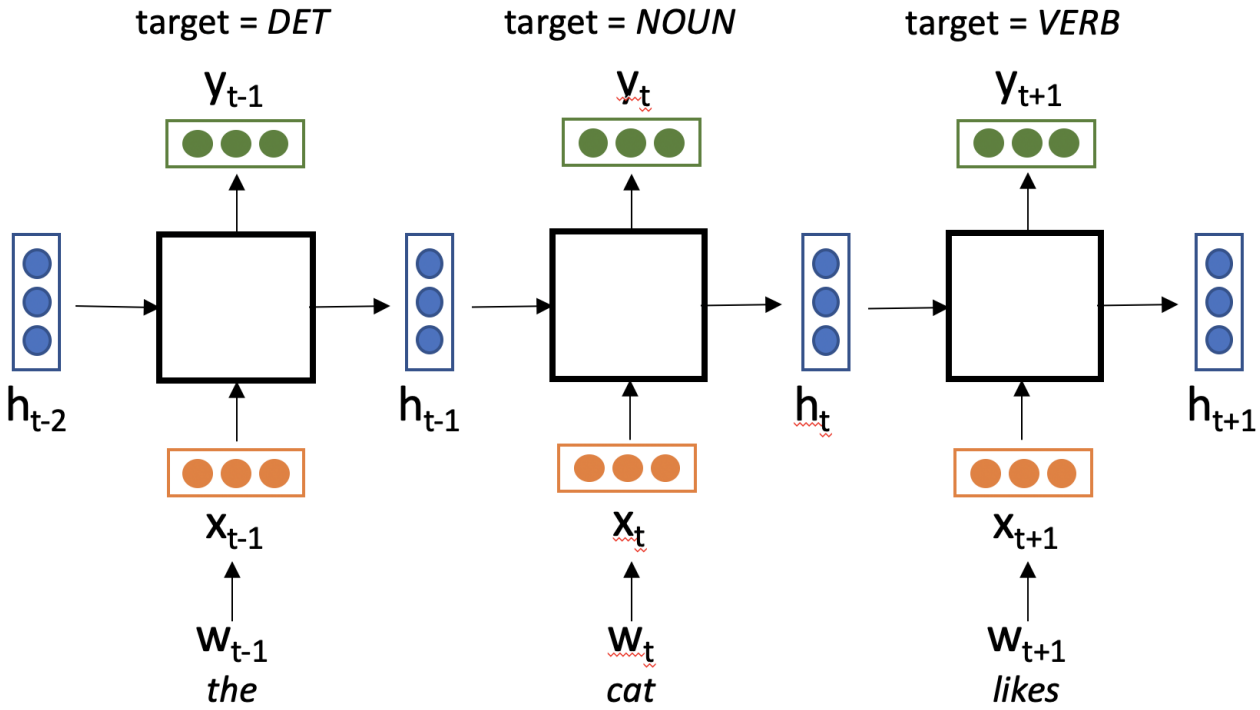
À	Marseille	Jean-Claude	Gaudin	est	élu	depuis	1995
O	B-LOC	B-NP	I	O	O	O	B-TIME

- language understanding (aka slot filling)

Je	veux	un	resto	italien	à	Rennes
nil	nil	nil	quoi_1	quoi_2	nil	où
O	O	O	B-WHAT	I-WHAT	O	B-WHERE

Recurrent neural network tagging

Predict at each position t the distribution probability over the set of tags from the hidden state h_t



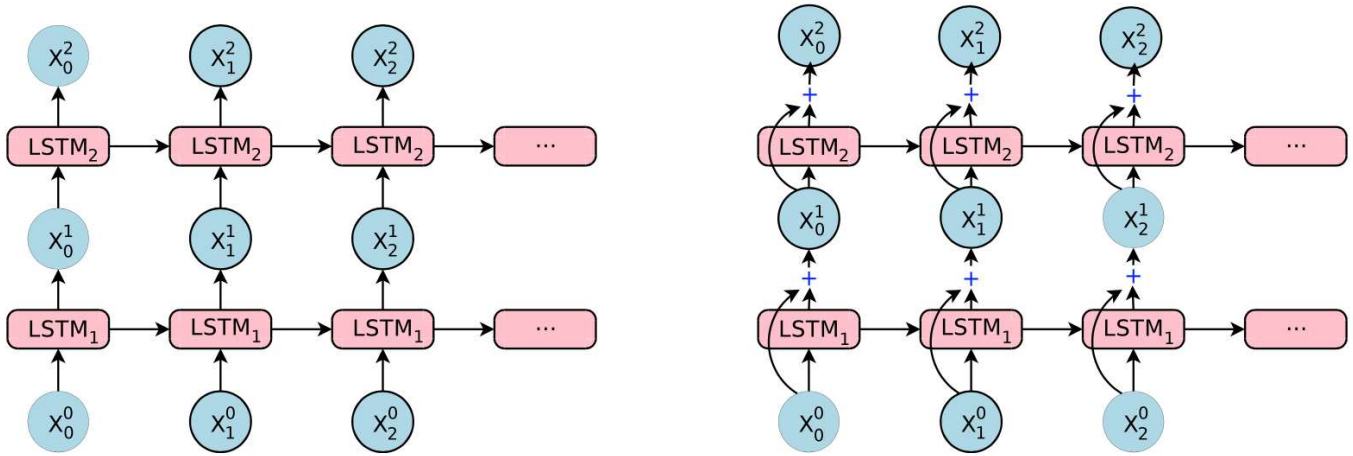
Decision at step t: $\hat{c}_t = \arg\max_i y_t(i)$

Objective function for training is categorical cross-entropy

$$J(\theta) = -\sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{j=1}^{|C|} \hat{y}_t^{(i)}(j)\ln(y_t^{(i)}(i))$$

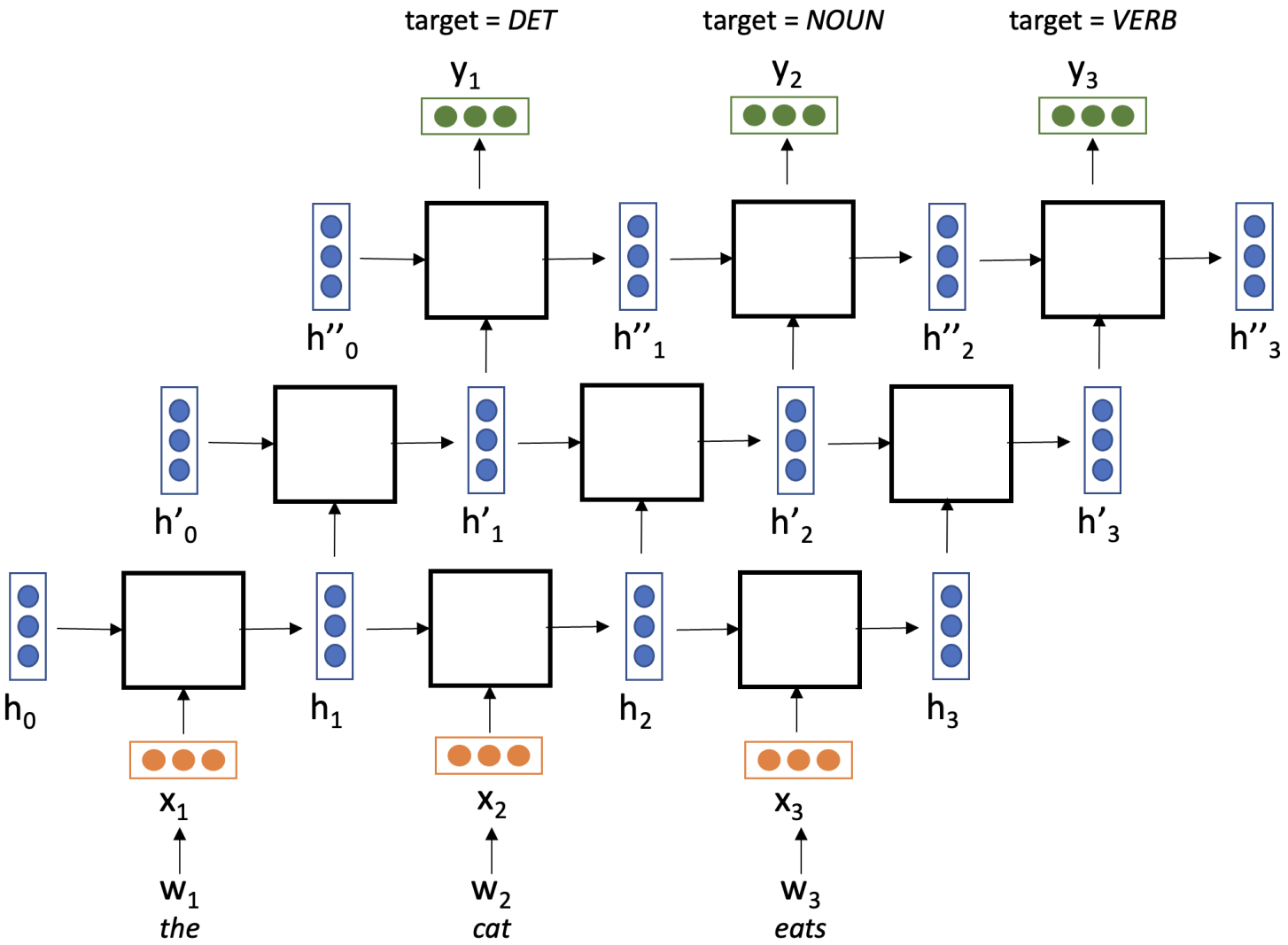
Variants of RNN models (cont'd)

- multi-layer LSTM
  - stack LSTM with one level feeding the other
  - combine (or not) LSTMs at prediction time

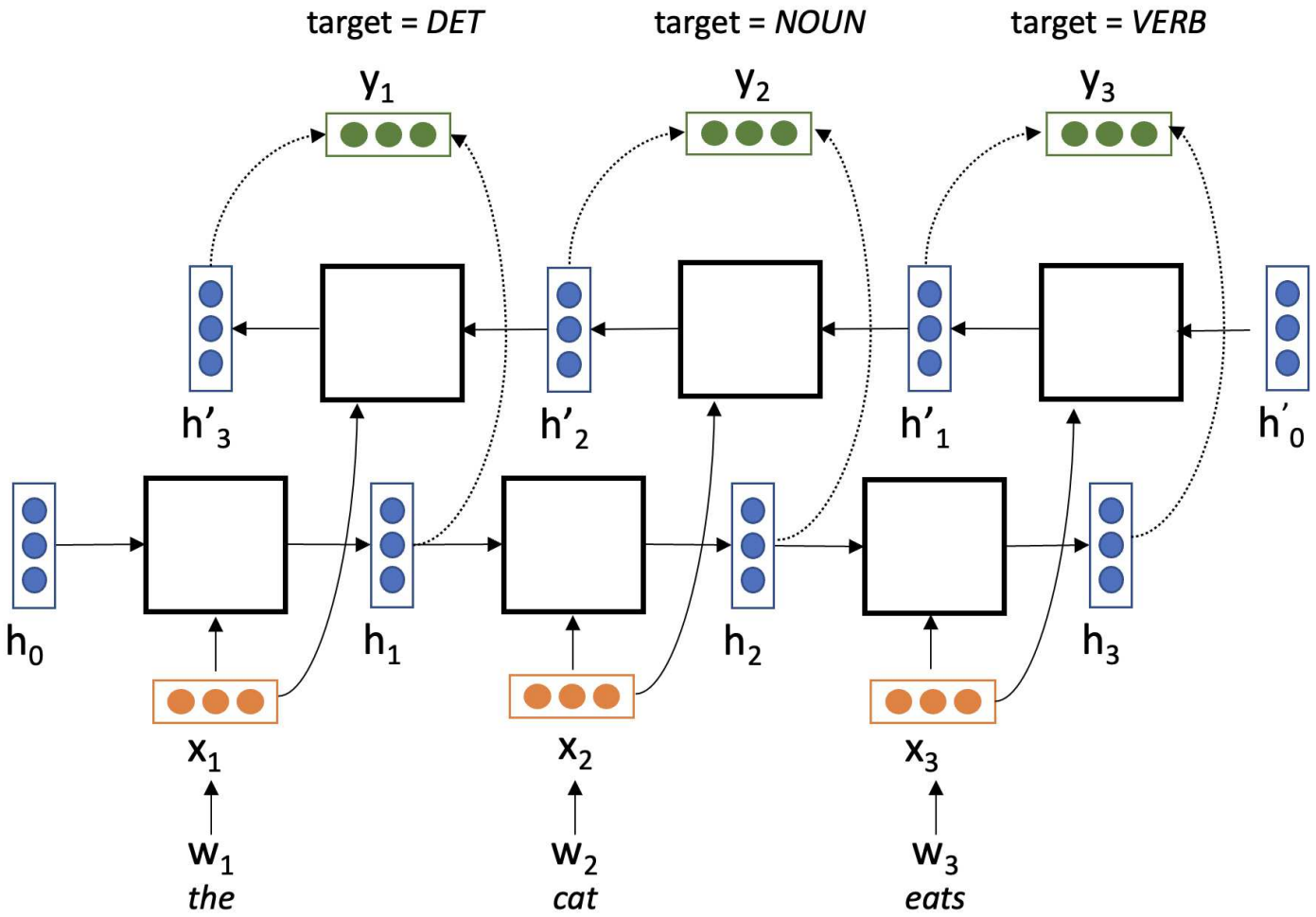


- multi-layer biLSTM
- LSTM / biLSTM with attention modeling
- etc.

Variants of RNN models: hierarchical models



Variants of RNN models: bi-directional models

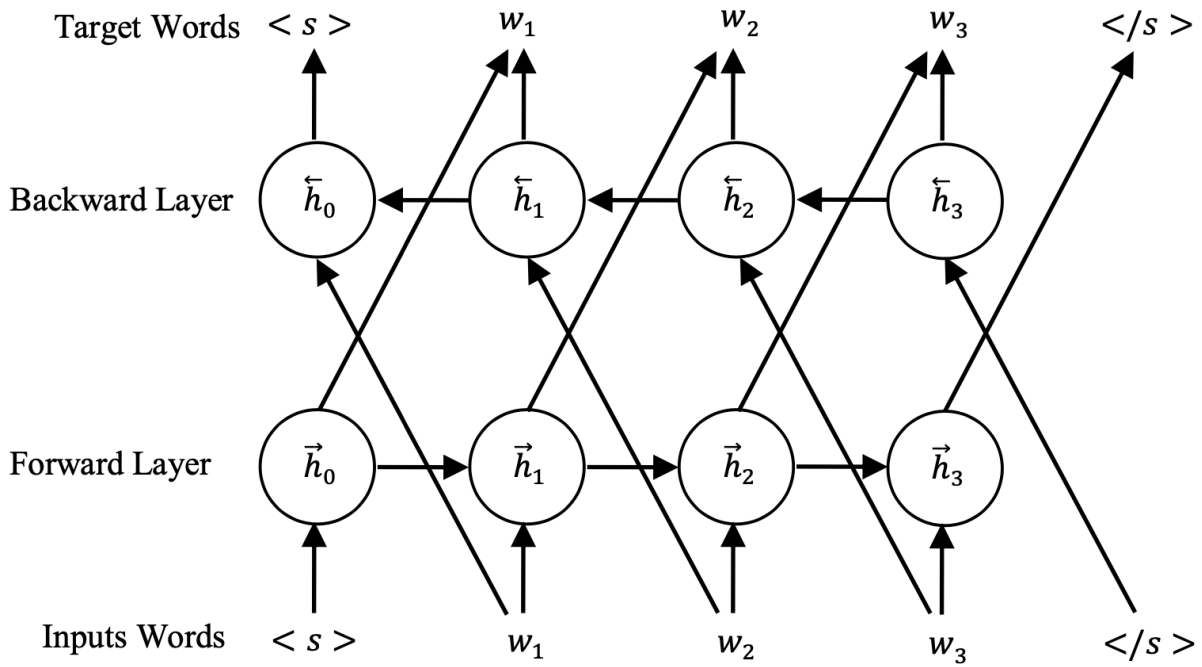


Bi-directional LSTM-based language models

Also consider language *backward*, i.e.,

$$P[w_1 \dots w_n] = P[w_n]P[w_{n-1}|w_n] \dots P[w_1|w_2 \dots w_n]$$

and combine forward and backward RNN states to predict the probability distribution over words, thus optimizing $\prod_k P[w_k|w_1, \dots, w_{k-1}, w_{k+1}, w_n]$



Mousa et al. Contextual Bidirectional Long Short-Term Memory Recurrent Neural Network Language Models: A Generative Approach to Sentiment Analysis. EACL 2017

Attention and transformers

Conditioning language model on images

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A skateboarder does a trick on a ramp.

A dog is jumping to catch a frisbee.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

A little girl in a pink hat is blowing bubbles.

A refrigerator filled with lots of food and drinks.

A herd of elephants walking across a dry grass field.

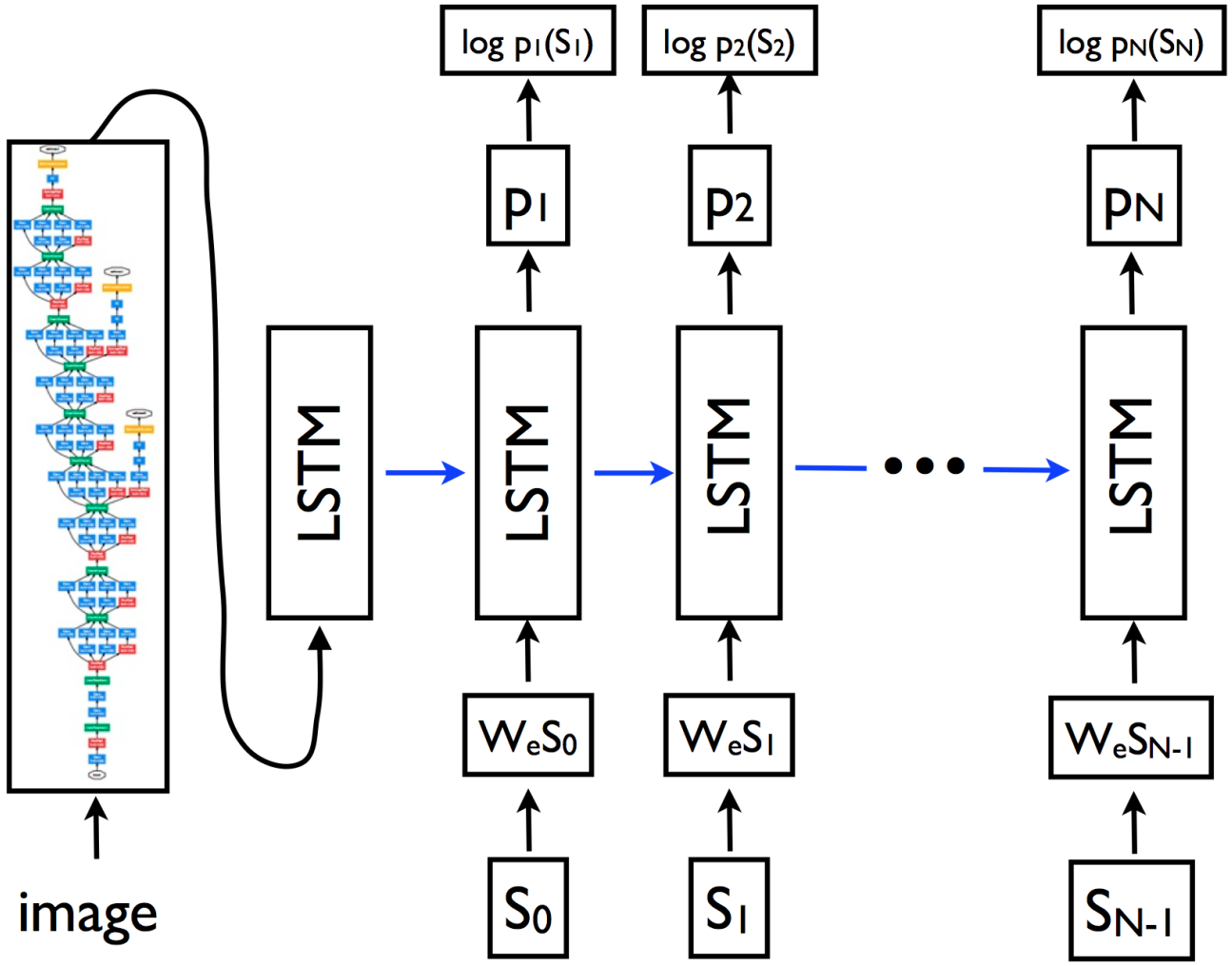
A close up of a cat laying on a couch.

A red motorcycle parked on the side of the road.

A yellow school bus parked in a parking lot.

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image

Conditioning language model on images

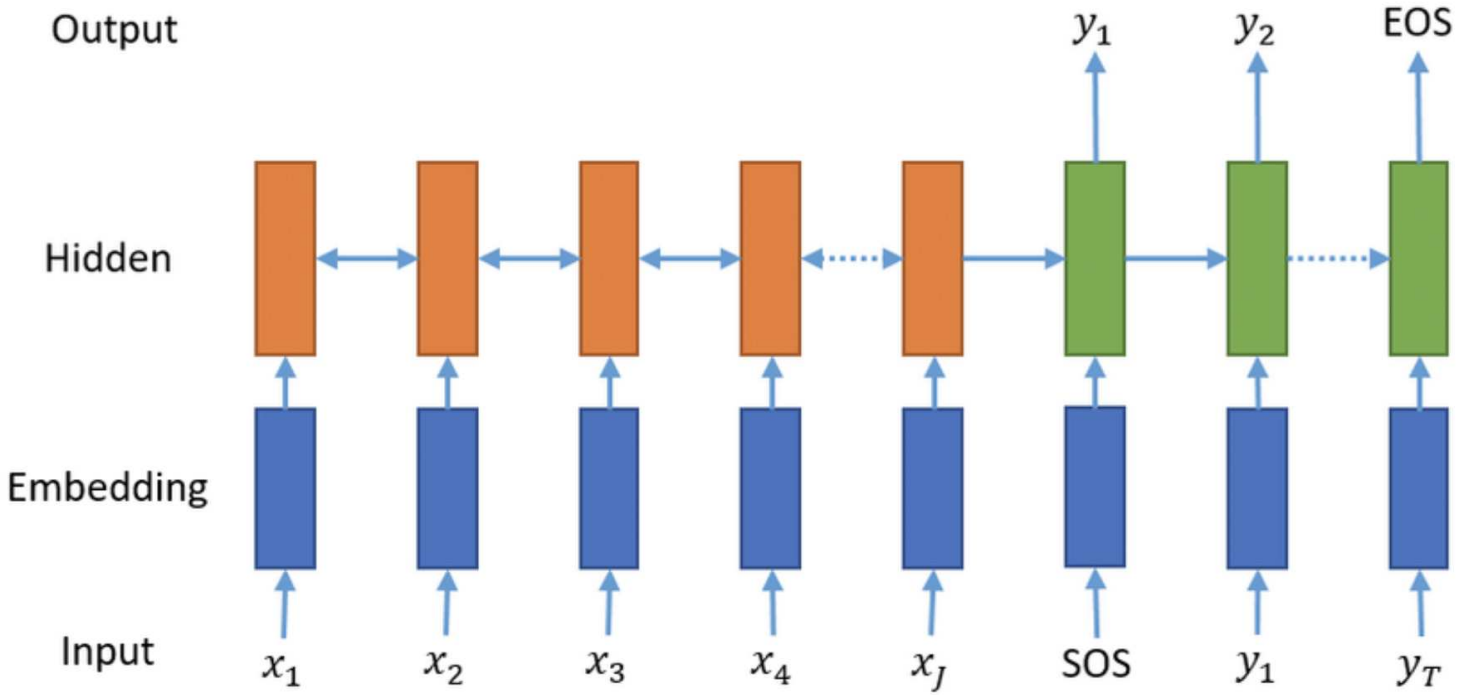


Training parameters of the RNN on images with human-generated captions so as to maximize the likelihood of the (training) caption texts.

Conditioning LMs on text (aka seq2seq)

Sequence to sequence encoder/decoder systems combine

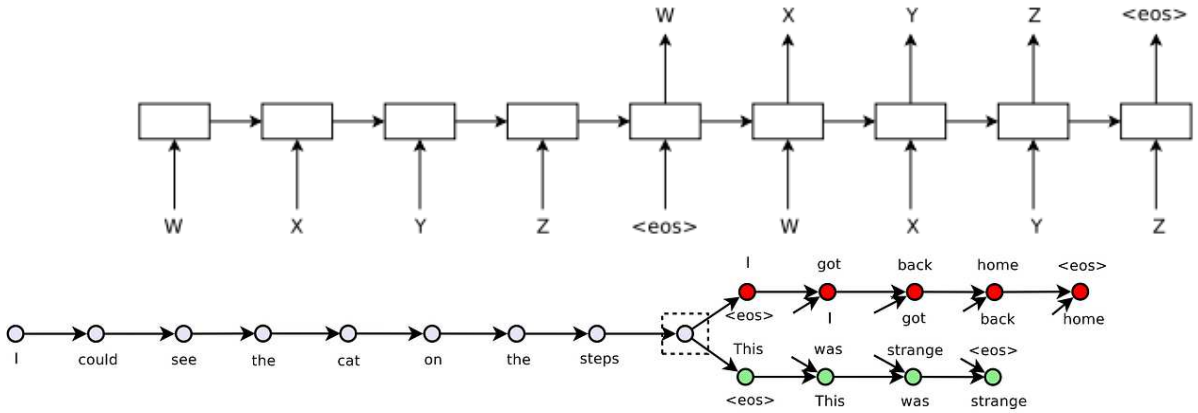
- a RNN to encode a message from a prompt/text, i.e., $h_0 = \text{RNN}_e(x_1, \ldots, x_n)$
- a RNN to generate a message conditioned on h_0, i.e., $w_1, \ldots, w_n = \text{RNN}_d(h_0)$



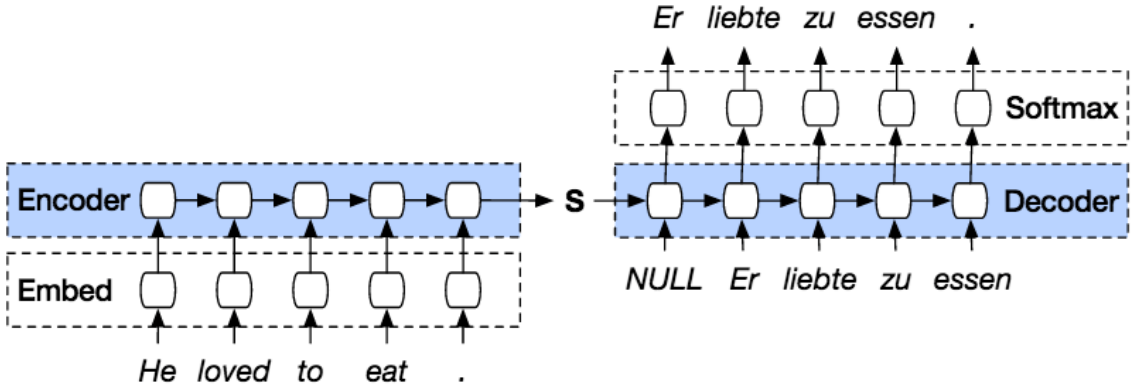
borrowed from Tian Shi et al., 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models.

Applications of seq2seq models

- sentence embedding, e.g., auto-encoders, denoising auto-encoders



- machine translation



- abstractive summarization, question answering, etc.

On practical aspects and use of encoder/decoder

- often convenient to also consider input sequence backward
  - process from x_n to x_1
  - use a bidirectional encoder
- can layer RNNs, both in the encoder and decoder
- better use ground truth in decoder at training time (or alternate) – aka teacher forcing

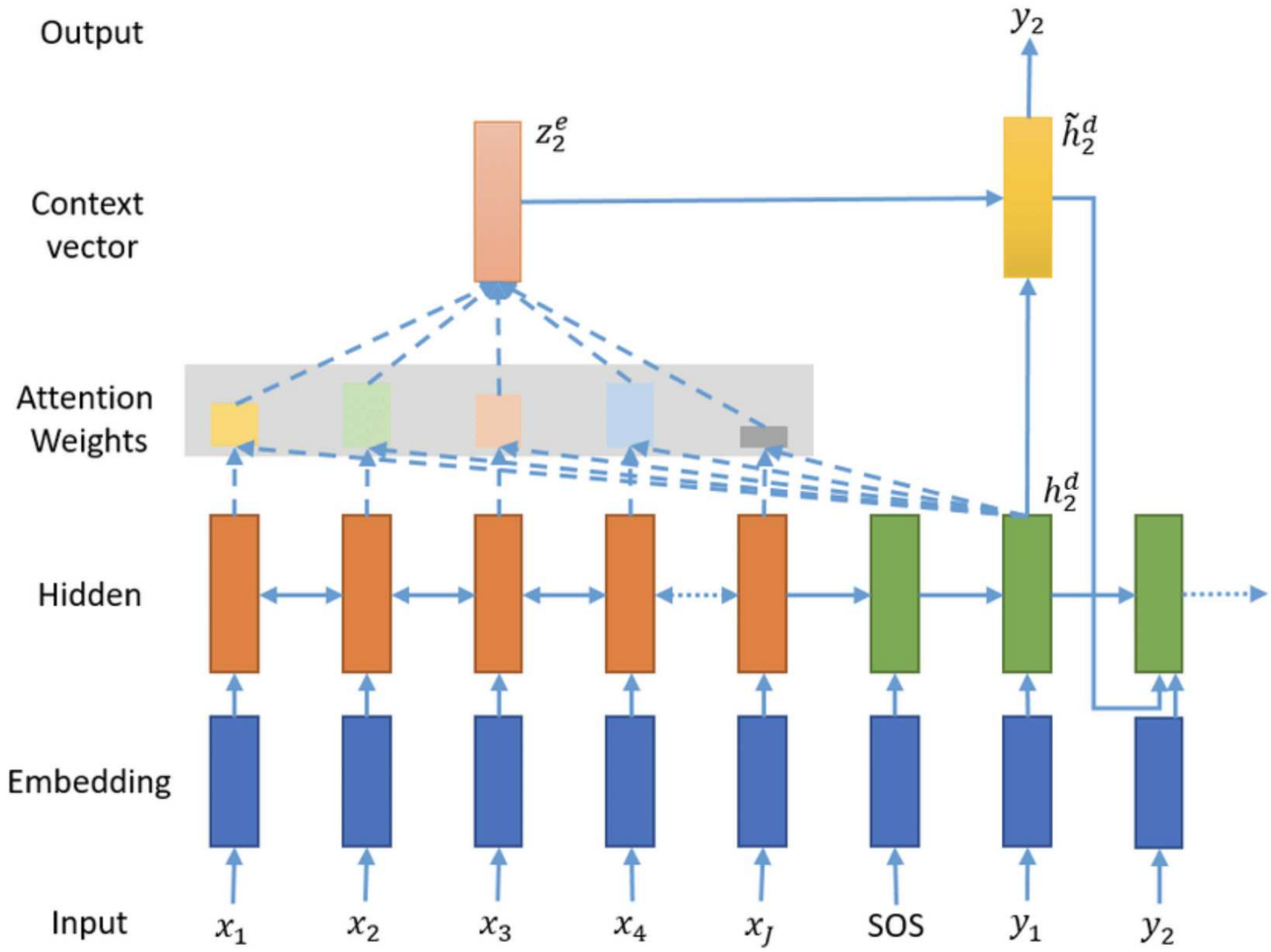
Yet not always good enough for two main reasons

1. the input message needs be fully summarized in a single embedding h_0 (hence only rather simple inputs work in practice)
2. (almost) independent choice of words might lead to poor language
   - might not respect syntax
   - short or truncated outputs
   - repeats

See Ziang Xie's 2018 practical guide on neural text generation for further details,

https://cs.stanford.edu/~zxie/textgen.pdf

Attention mechanisms in seq2seq



borrowed from Tian Shi et al., 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models.

Attention mechanisms in seq2seq: the maths

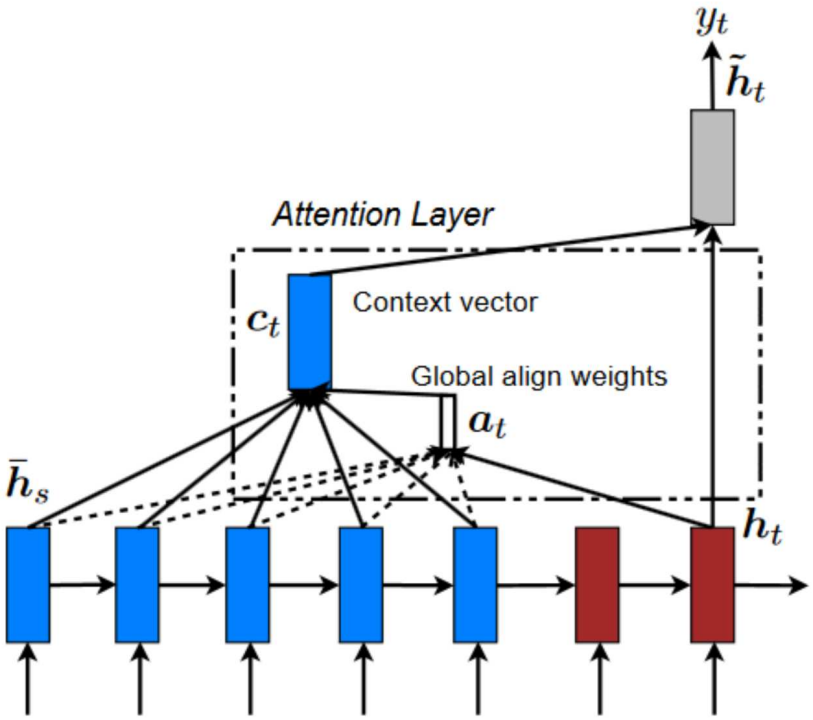
input $x = \{x_1, \ldots, x_n\}$, output y

$\overline{h}_s$: state in encoder at time $s \in [1, n]$

h_t: state in decoder at time t

$\tilde{h}_t$: modified state in decoder at time t

$$
\begin{aligned}
a_{ts} &= \text{softmax}(h_t^t W_a \overline{h}_s) \\
c_t &= \sum_{s=1}^{n} a_{ts} \overline{h}_s \\
\tilde{h}_t &= \tanh(W_c[c_t; h_t]) \\
p(\cdot|y_{<t}, x) &= \text{softmax}(W_s \tilde{h}_t)
\end{aligned}
$$

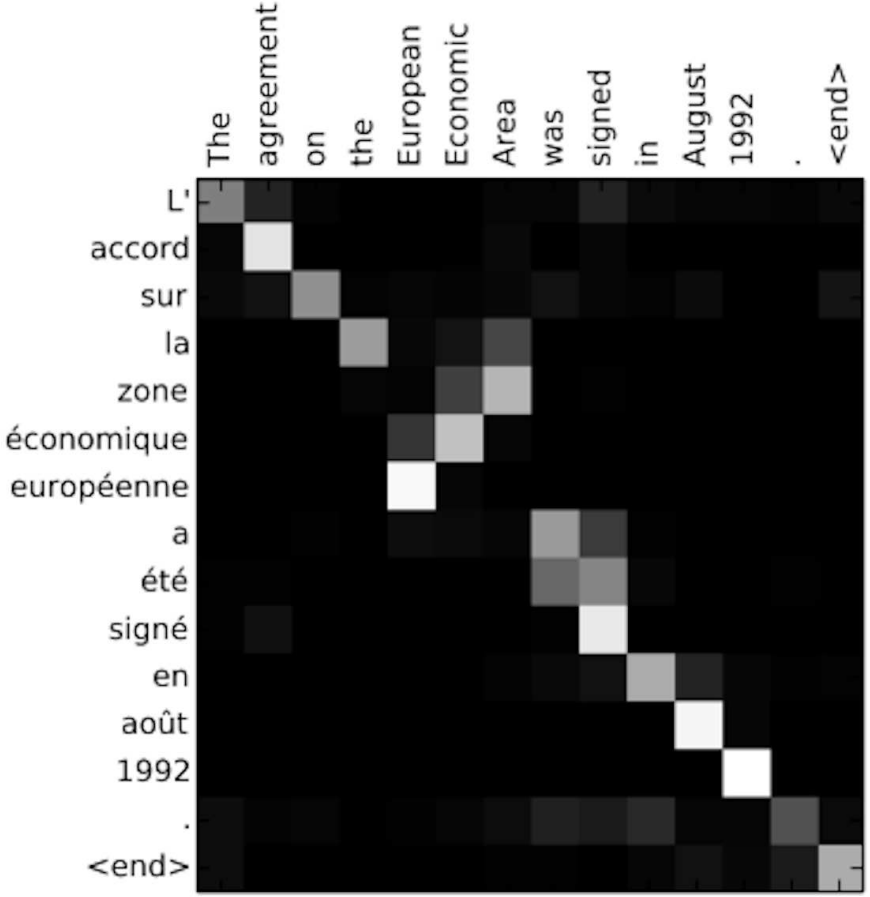


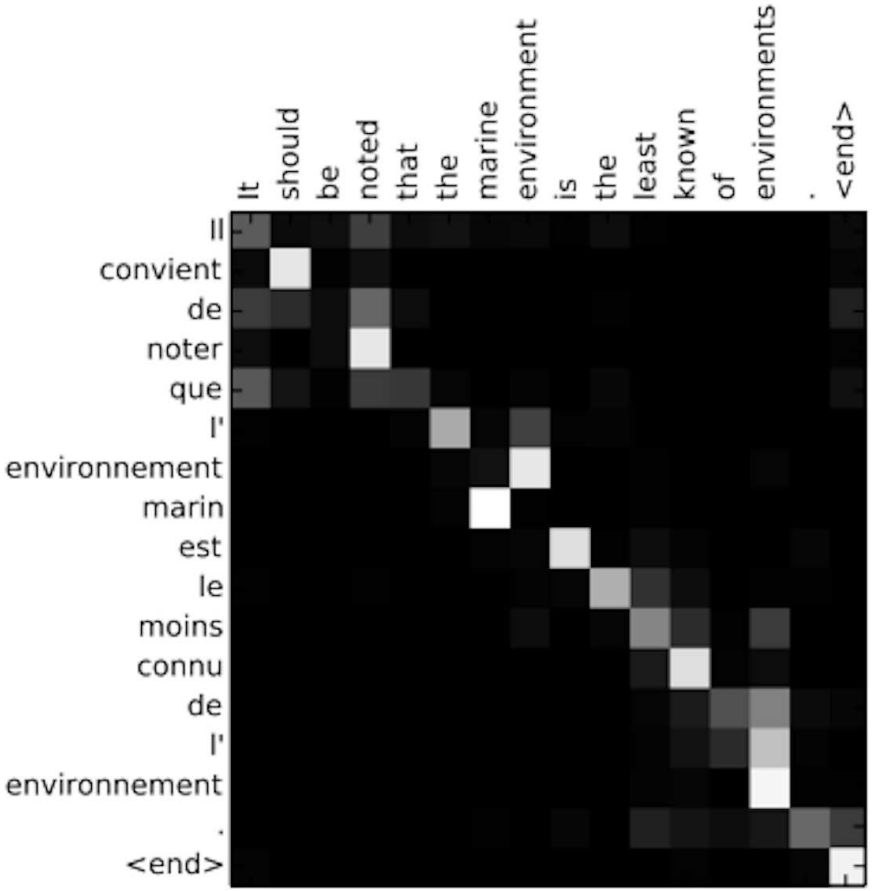
from Luong et al. 2015

a_{ij} = how much input j matters for output i

Attention maps in seq2seq

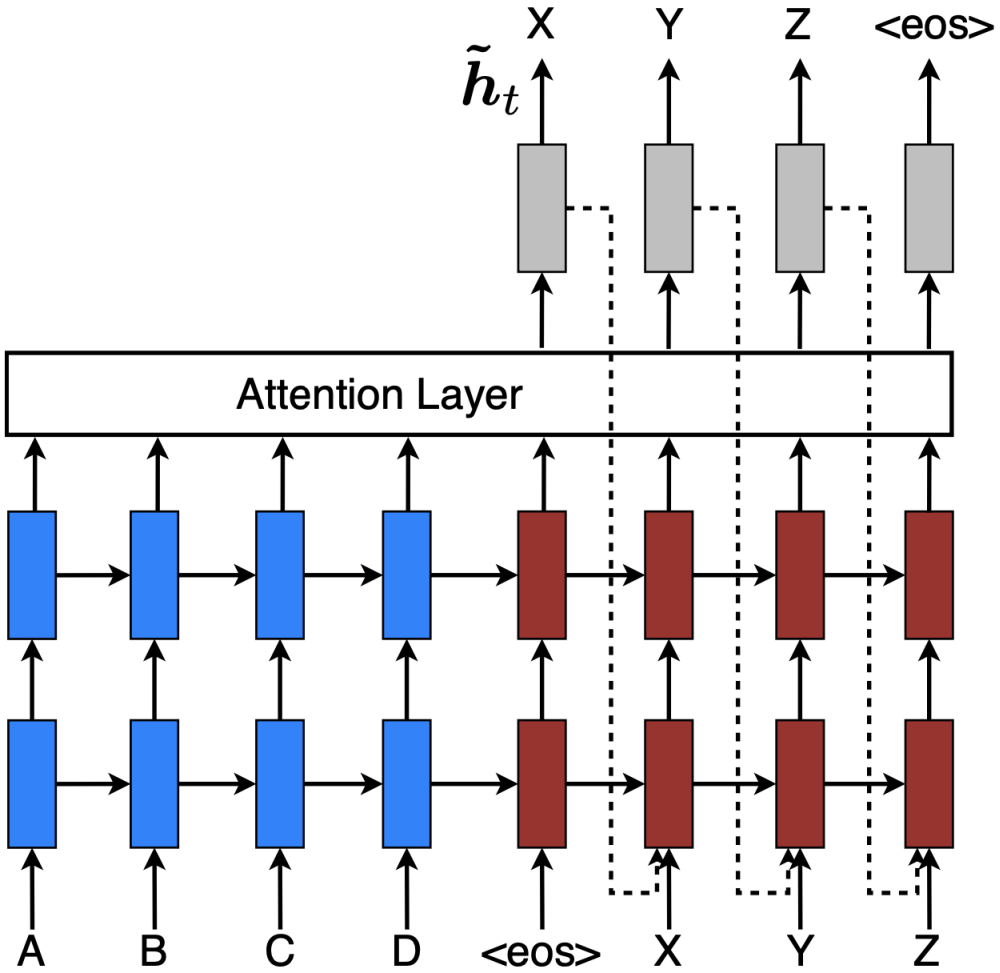
The **attention matrix** A gathers for all output tokens (rows) the attention w.r.t. to the input tokens (columns) in a English to French translation task here.





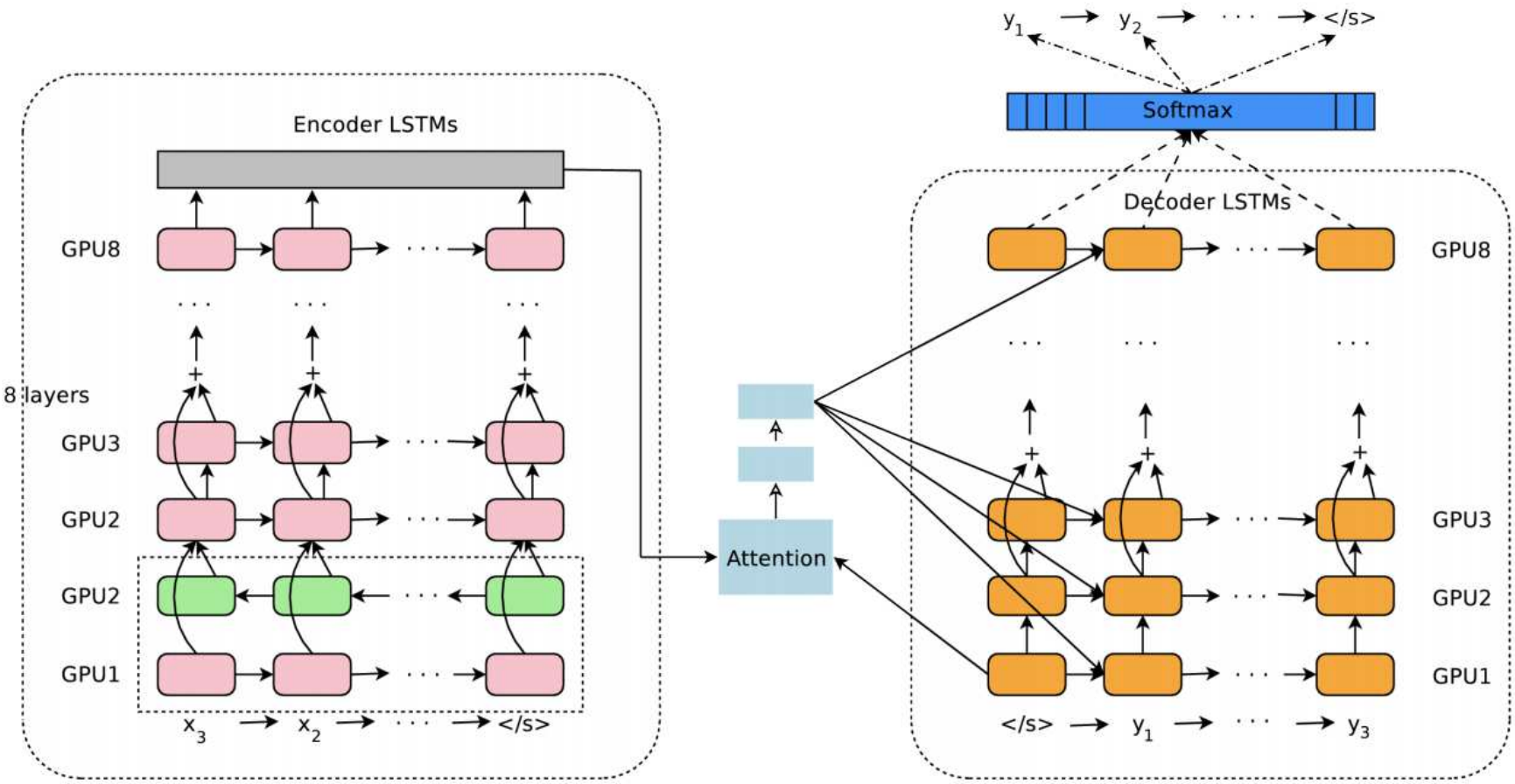
Dzmitry Bahdanau et al. 2015. Neural machine translation by jointly learning to align and translate.

Complexifying things with layers



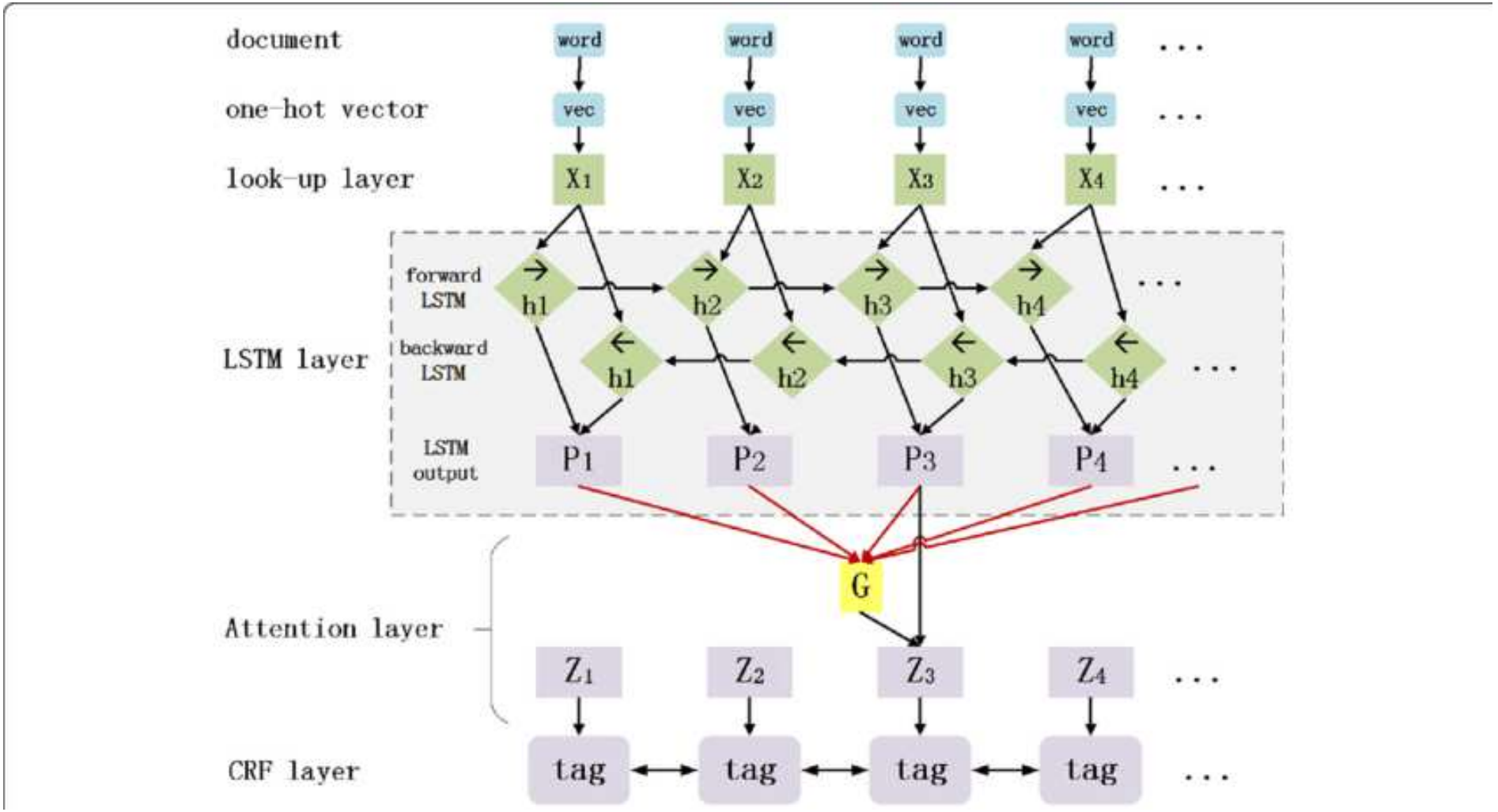
Luong et al. 2015. Effective approaches to attention-based neural machine translation.

Google neural machine translation: a complex seq2seq



Yonghui Wu et al. 2016 Google's neural machine translation system: Bridging the gap between human and machine translation

Attention mechanisms are not limited to seq2seq



Bin Ji et al. 2019. A hybrid approach for named entity recognition in Chinese electronic medical record

Abstracting the attention mechanism

$$a_{ts} = \text{softmax}(h_t^t W_a \overline{h}_s) \quad \text{and} \quad c_t = \sum_{s=1}^{n} a_{ts} \overline{h}_s$$

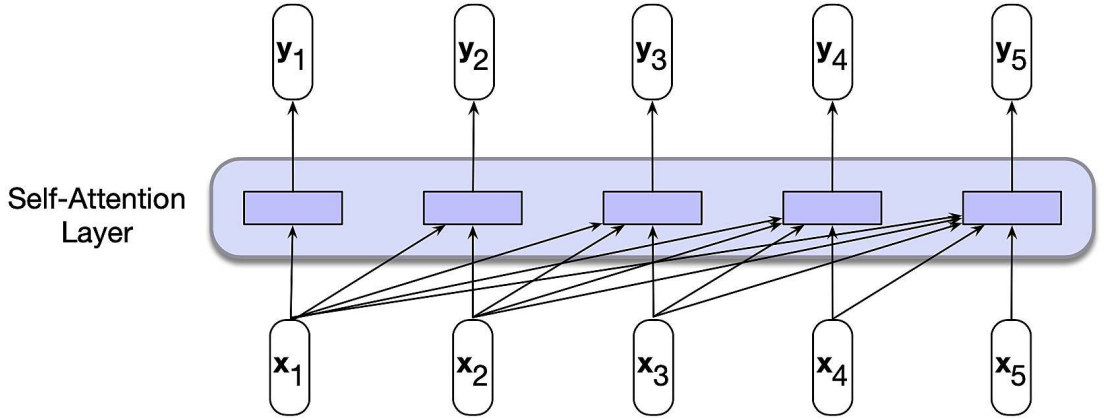
Can be abstracted through three key elements

- **Query**: a representation of the element that you are considering as your focus point (the point you attend from)
  $\rightarrow$ *the decoder state* h_t *at time* t
- **Keys**: a representation of the elements you want to consider to change your query element (the points you attend to)
  $\rightarrow$ *the encoder state variables* $\overline{h}_i$
- **Values**: a representation of the elements you attended to to compute the average and get a new representation related to the query
  $\rightarrow$ *keys and values are alike here in Luong's attention*

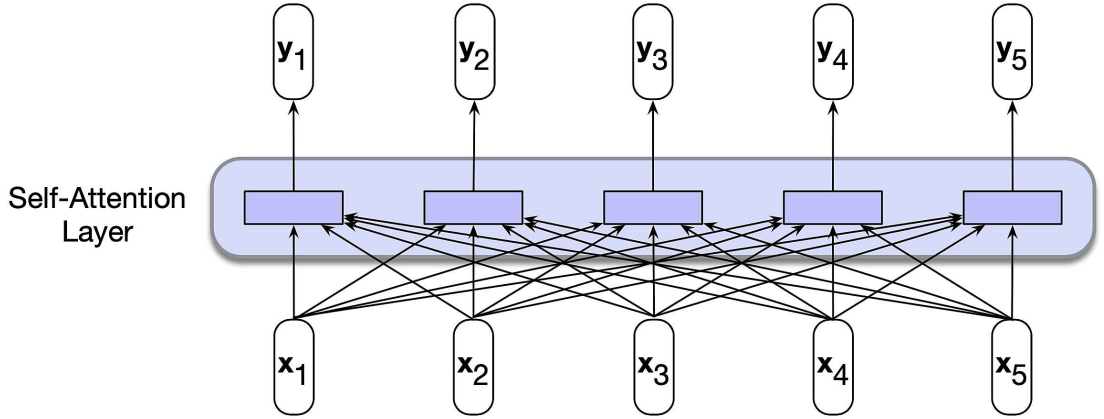
$$a_t = q_t K^t = \text{sofmax}\left(\begin{pmatrix} q_{t1} & \dots & q_{td} \end{pmatrix} \begin{pmatrix} k_{11} & \dots & k_{n1} \\ \vdots & \ddots & \vdots \\ k_{1d} & \dots & k_{nd} \end{pmatrix} \right)$$

The self-attention principle

The key idea is to modify an entry vector (word embedding) based on its relations/attentions with respect to other entry vectors, either in a causal manner (language generation)

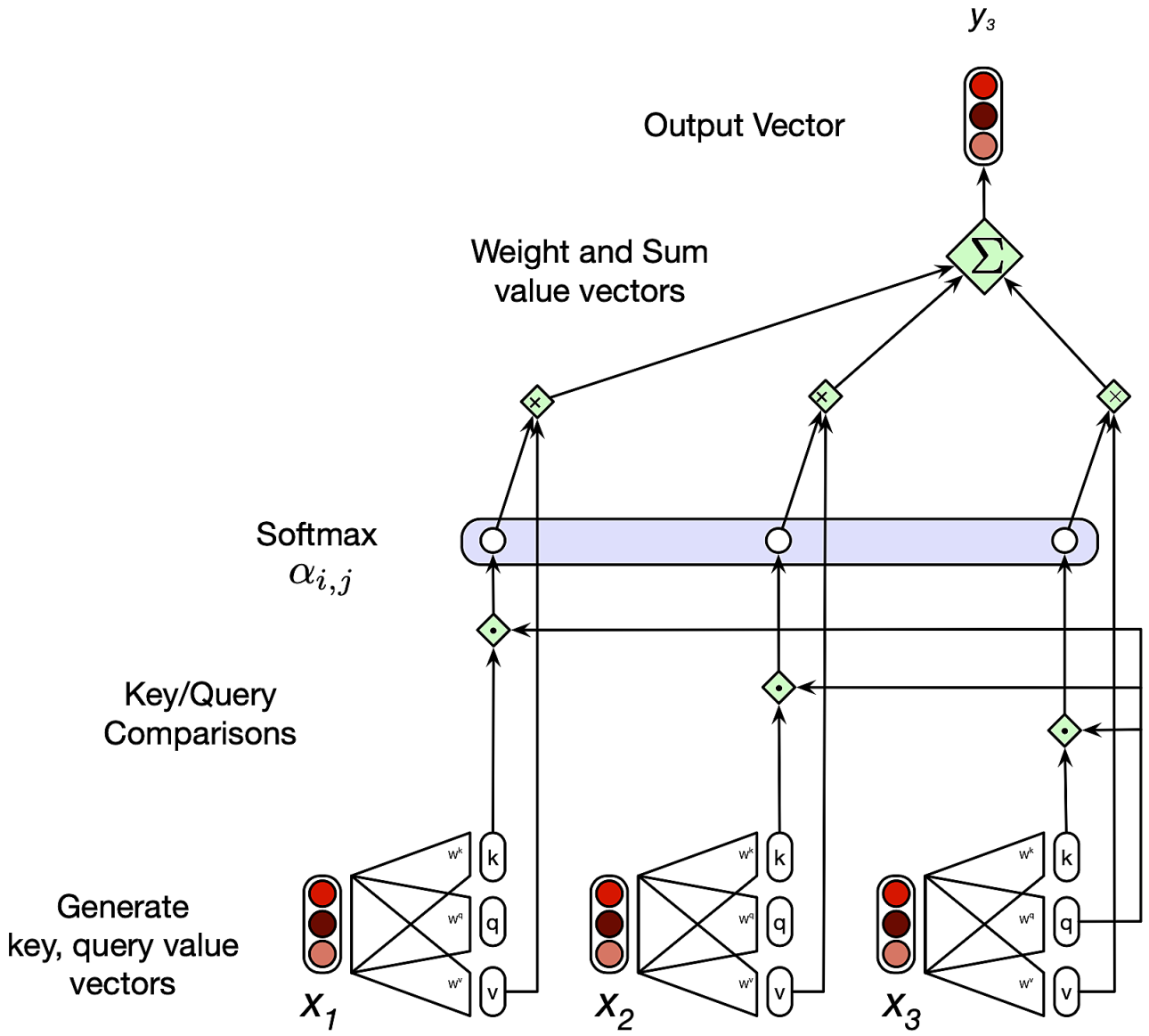


or in a bidirectional manner



borrowed from Jurafsky and Martin's book

Self-attention basic layer (the idea)



borrowed again from Jurafsky and Martin's book

Self-attention basic layer (the maths)

Given an input sequence $x = x_1, \ldots, x_n$, with $x_i \in \mathbb{R}^d$

1. project each x_i to its corresponding query q_i, key k_i and value v_i
   - $q_i = \mathbf{W}_q x_i$: query to measure how x_i relates to x_j for all j
   - $k_i = \mathbf{W}_k x_i$: key to measure how x_j is related to x_i
   - $v_i = \mathbf{W}_v x_i$: value to compute new output from x_i
2. compute attention distribution at all positions i

$$a_{ij} = \frac{\exp(q_i \cdot k_j / \sqrt{d})}{\sum_{l=1}^{n} \exp(q_i \cdot k_l / \sqrt{d})}$$

3. compute output values

$$y_i = \sum_{j=1}^{n} a_{ij} v_j$$

Self-attention basic layer (the matrix maths)

All these operations can be efficiently performed with matrices, starting from $\mathbf{X} \in \mathbb{R}^{n \times d}$ gathering all input embeddings x_i:

1. compute the queries, keys, values for all tokens

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q \quad \mathbf{K} = \mathbf{X}\mathbf{W}_k \quad \mathbf{V} = \mathbf{X}\mathbf{W}_v$$

2. compute the attention matrix

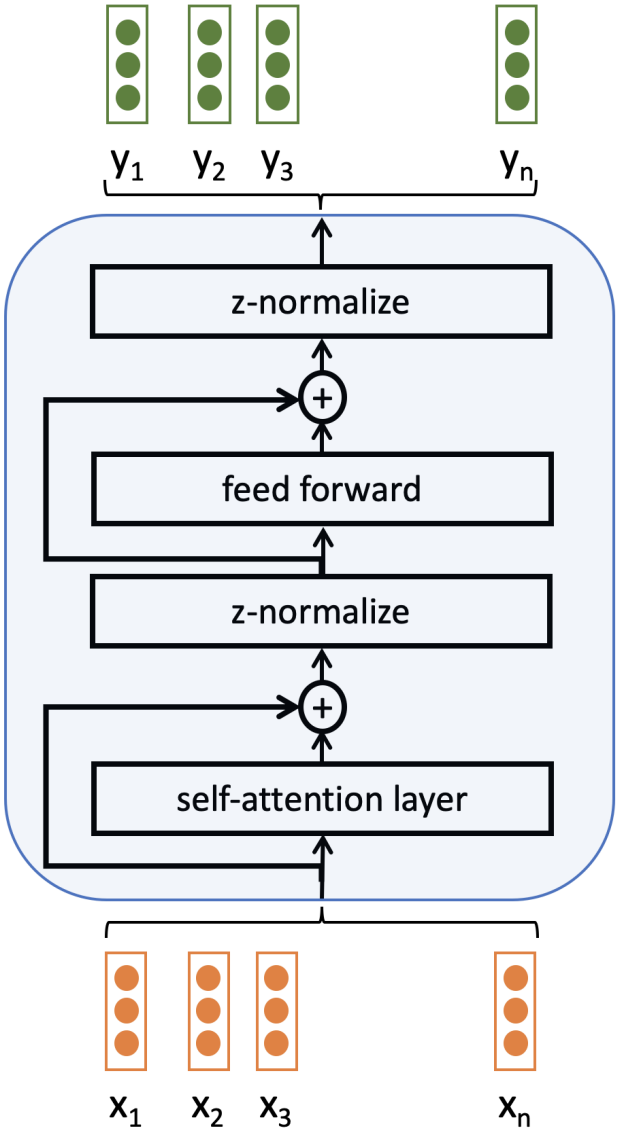
$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^t}{\sqrt{d}}\right)$$

3. compute new representations

$$\mathbf{Y} = \mathbf{A}\mathbf{V}$$

For causal models, force the upper triangular part of $\mathbf{A}$ to be zero

From self-attention to transformer block



Add a few things on top of the self-attention layer:

- residual connections

$$\mathbf{X}' = \mathbf{X} + \text{SelfAttention}(\mathbf{X})$$

- layer normalization

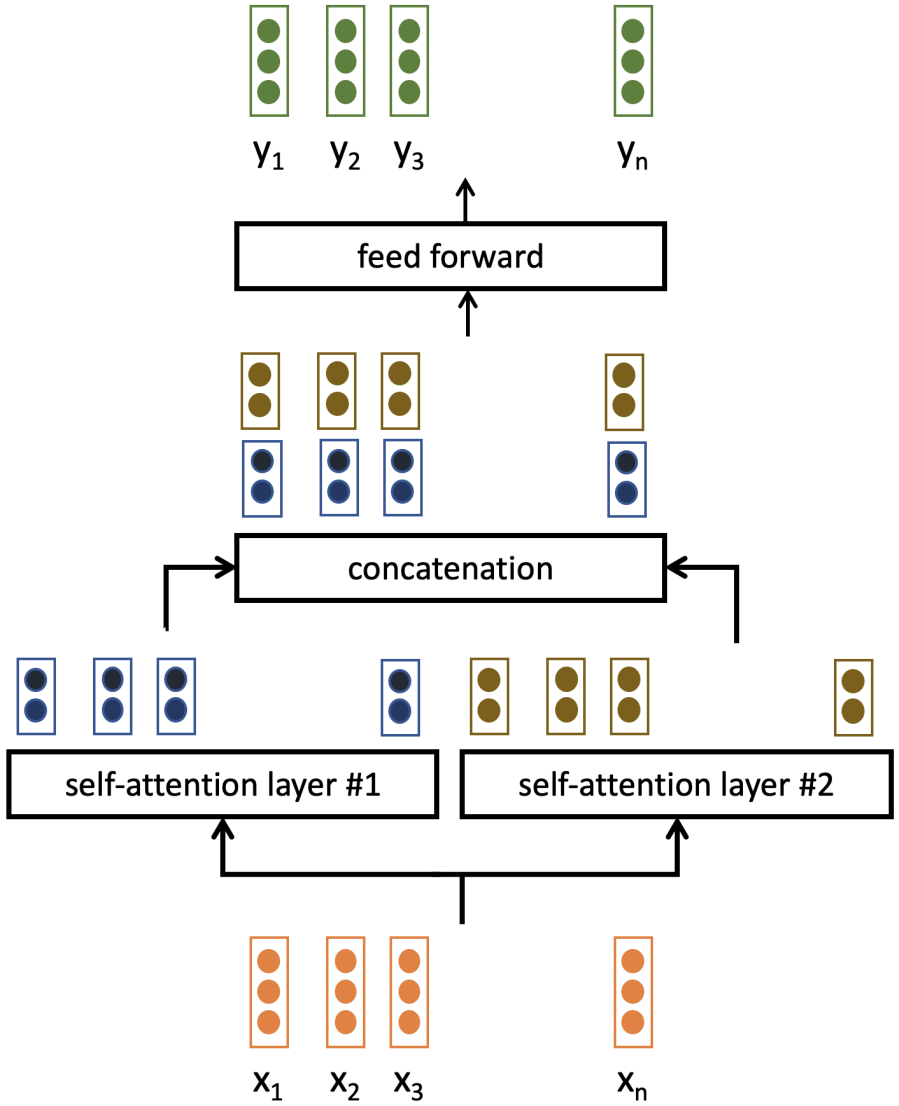
$$\mathbf{X}'(i,j) = \gamma \frac{\mathbf{X}(i,j) - \mu_i}{\sigma_i} + \beta$$

- pointwise feed-forward

$$\mathbf{X}'(i,:) = \mathbf{A}_2 \text{ReLU}\left(\mathbf{A}_1 \mathbf{X}(i,:) + b_1\right) + b_2$$

Always more: multi-head attention

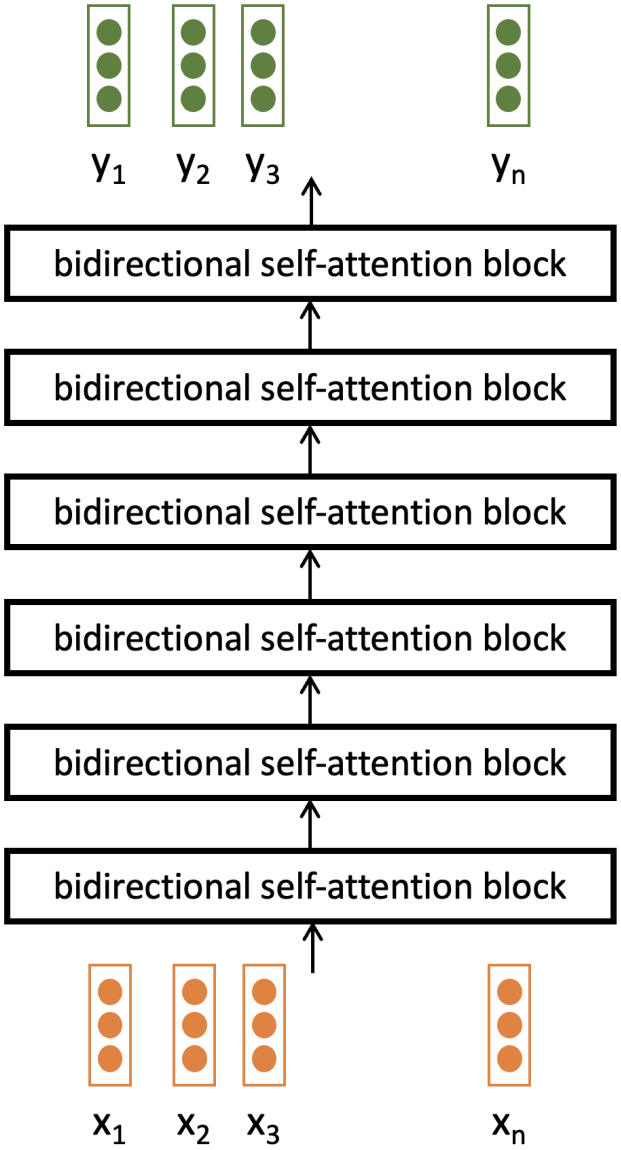
Use multiple self-attention layers (aka *heads*) in parallel, each with its own set of parameters $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i$ and combine the result.



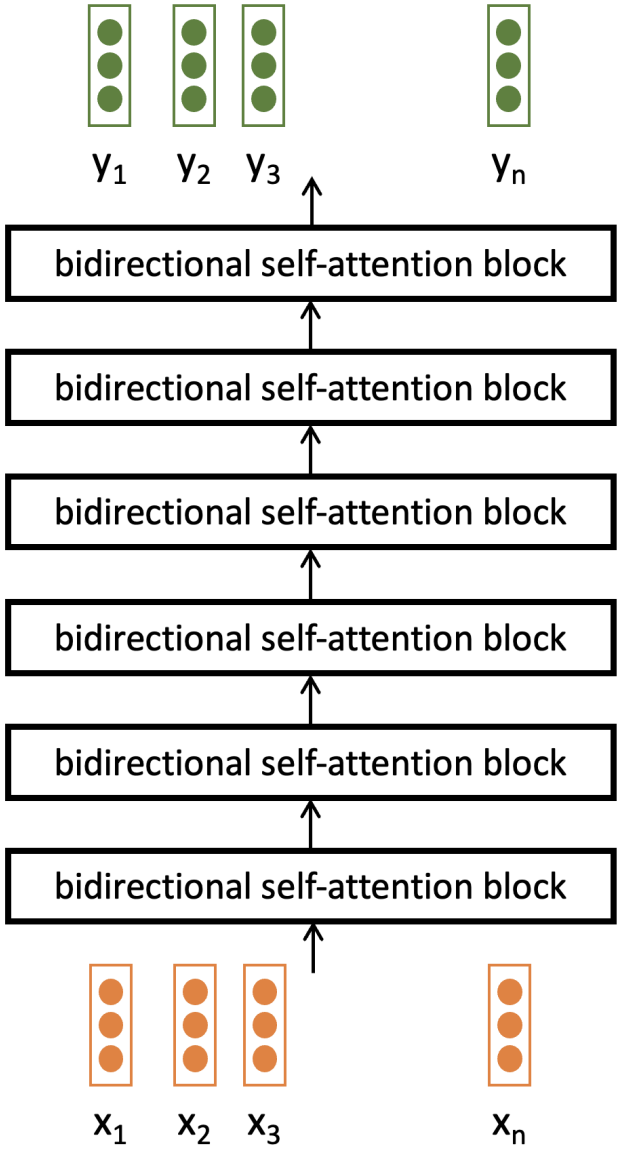
In practice, each self-attention operates on a subset of dimensions of the input space $\mathbb{R}^d$ and the final projection is omitted.

If $d = 64$ and considering 2 heads, each head operates on $64/2 = 32$ dimensions, the first one being $X(:, [0 : 31])$, the second one $X(:, [32 : 63])$.

Causal and bidirectional transformers

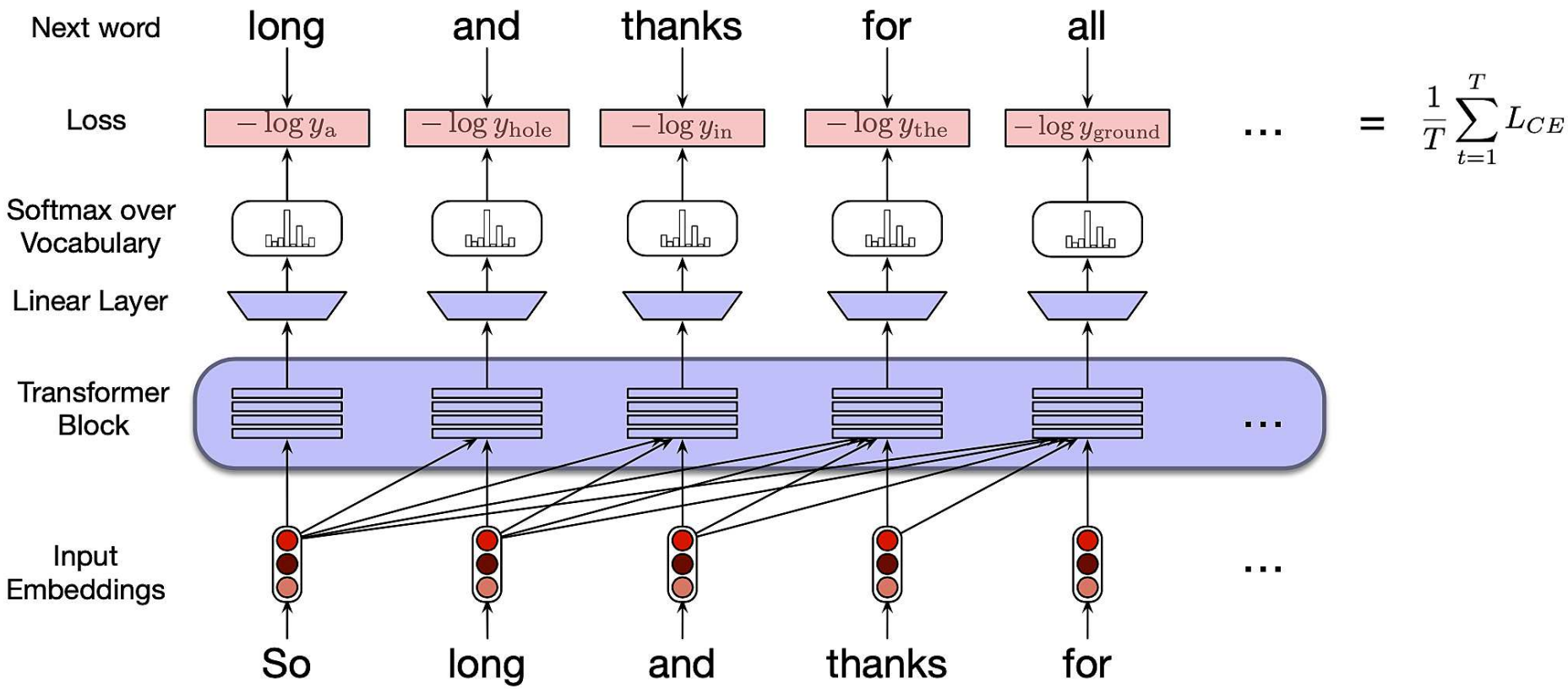


causal



bidirectional

Causal transformers as (generative) language models (aka GPT)



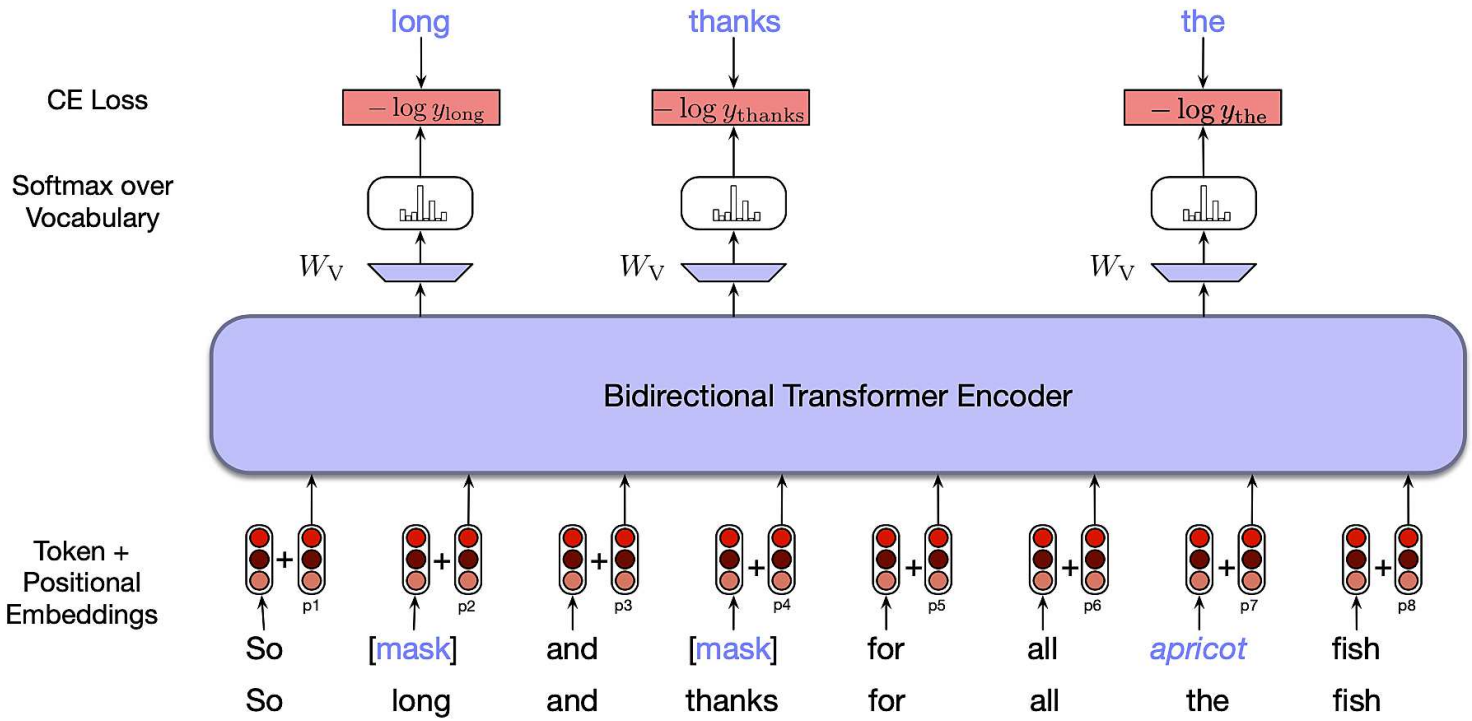
borrowed again from Jurafsky and Martin's book

$\Rightarrow$ can be used for language generation exactly as a we saw with recurrent networks

Bidirectional transformers as language models (aka BERT)

Randomly mask or modify words and train networks to precit the actual word from the contextual embedding of the modified tokens:

→ Original BERT trained on 3.3B tokens, 15 % are modified: 80 % are actually masked, 10 % randomly replaced, 10 % left unchanged; training the LM objective function only on the 15 % modified tokens.

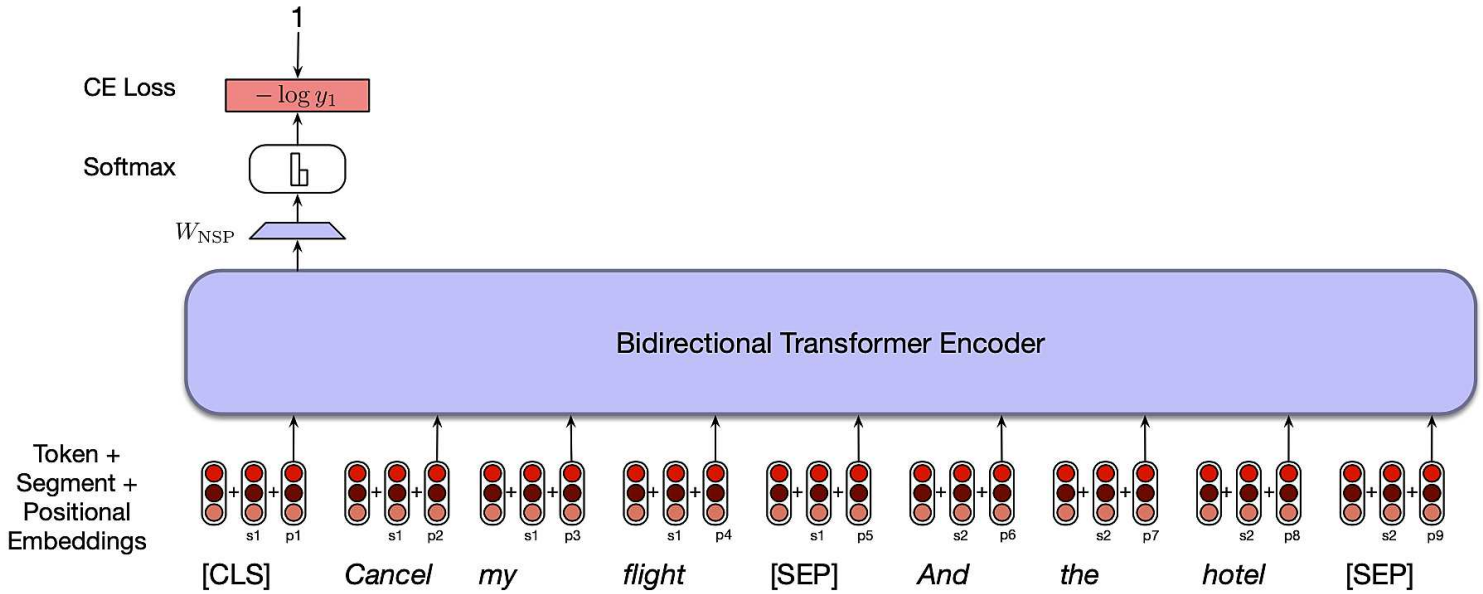


borrowed again and again from Jurafsky and Martin's book

Pushing BERT one-step further with next sentence prediction

Many tasks look at two sentences (pieces of text as input), e.g., for natural language inference (entailment), measuring semantic proximity, summarization, etc.

Train BERT with two sentences to predict whether one follows the other or not

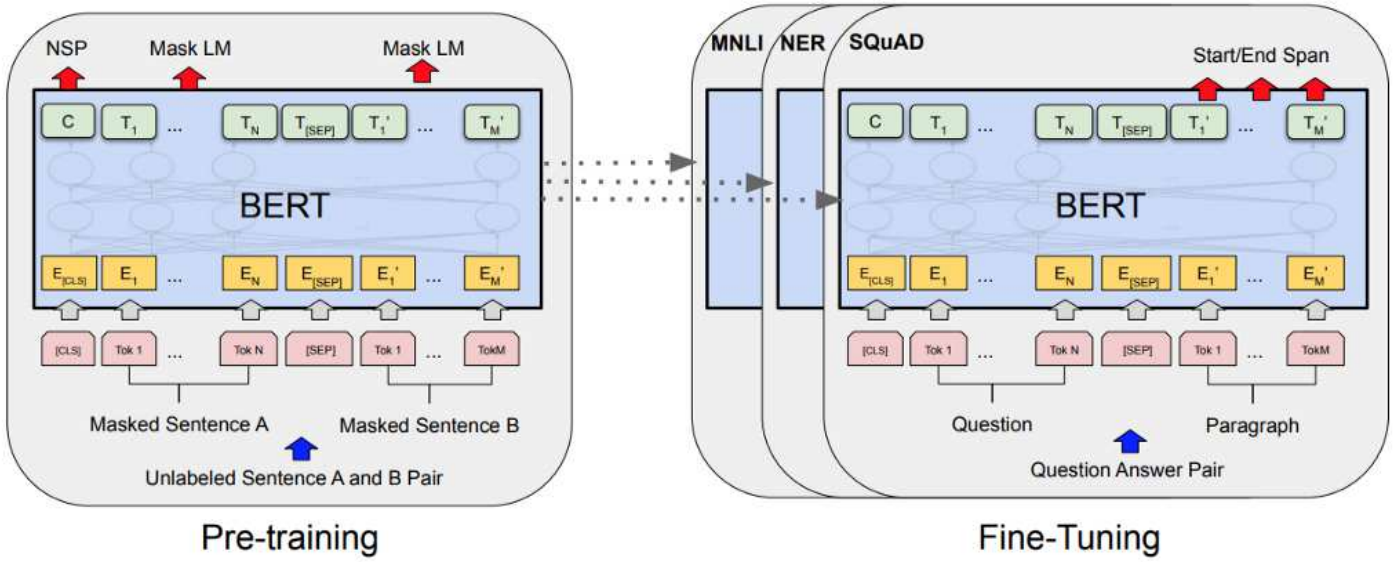


borrowed again from Jurafsky and Martin's book

Full BERT training combines masked LM and next sentence prediction (from modified sentences)

Transfer learning: the BERT case

Add a classifier on top of a pre-trained transformer language model and retrain the whole stuff

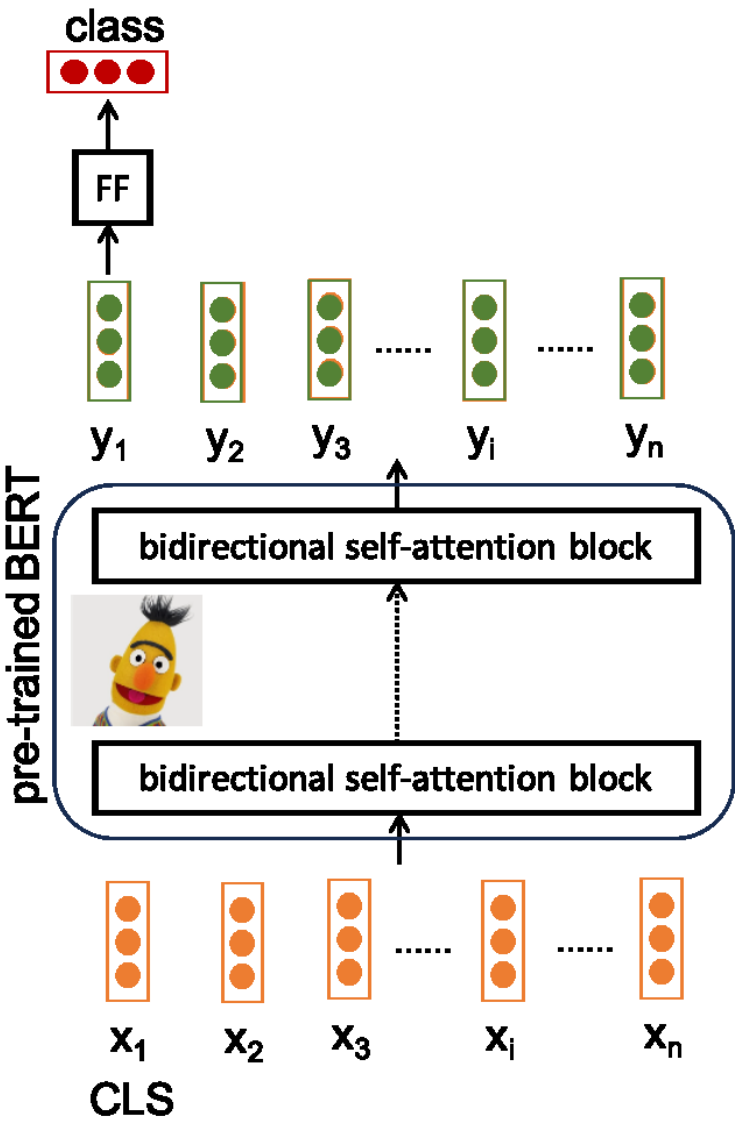


Jacob Devlin et al. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding

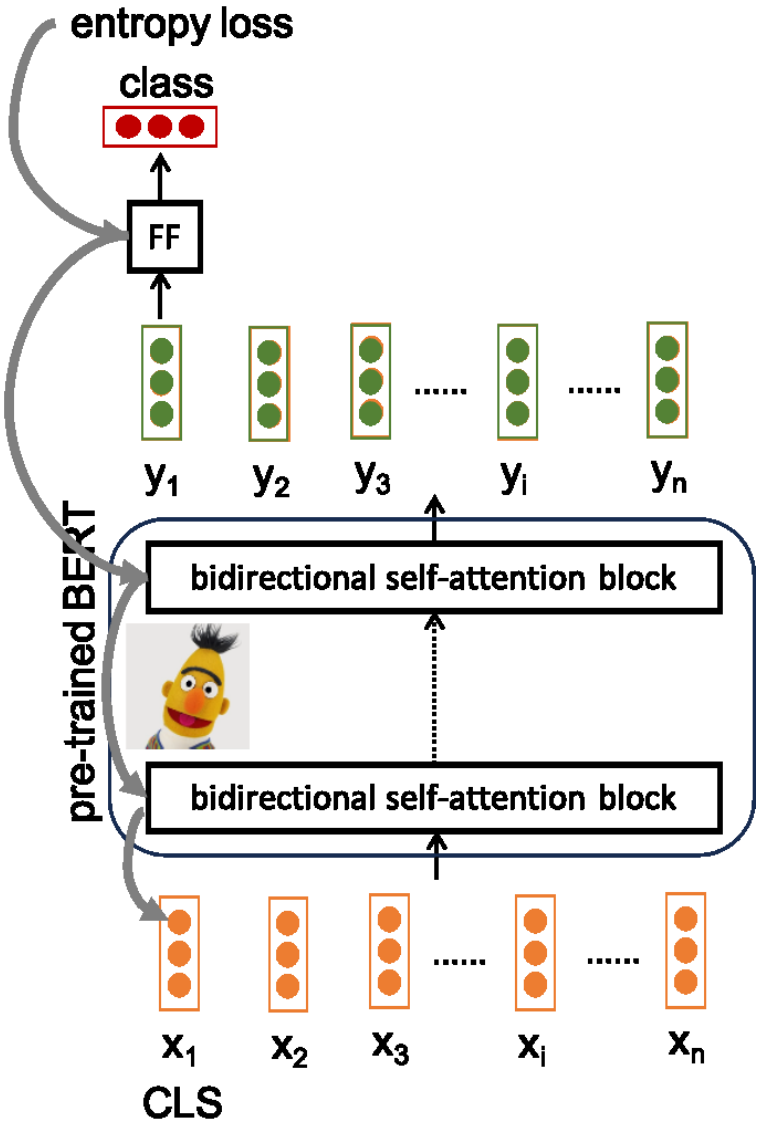
BERT's classification *head* depending on the task:

Classification	one sentence, embedding of the [CLS] token for classification
Comparison	two sentences, embedding of the [CLS] token for classification
Tagging	one sentence, contextual embeddings of tokens used in tagger

BERT transfer learning for document classification

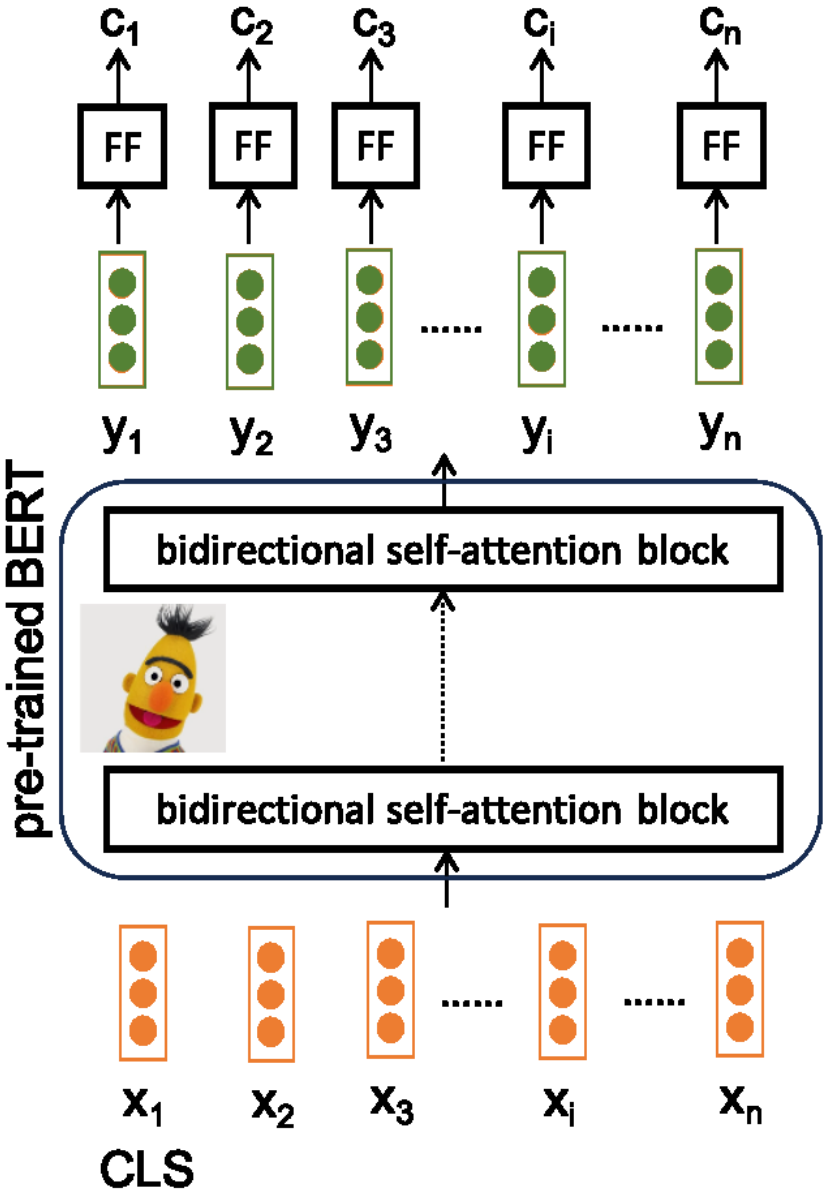


inference

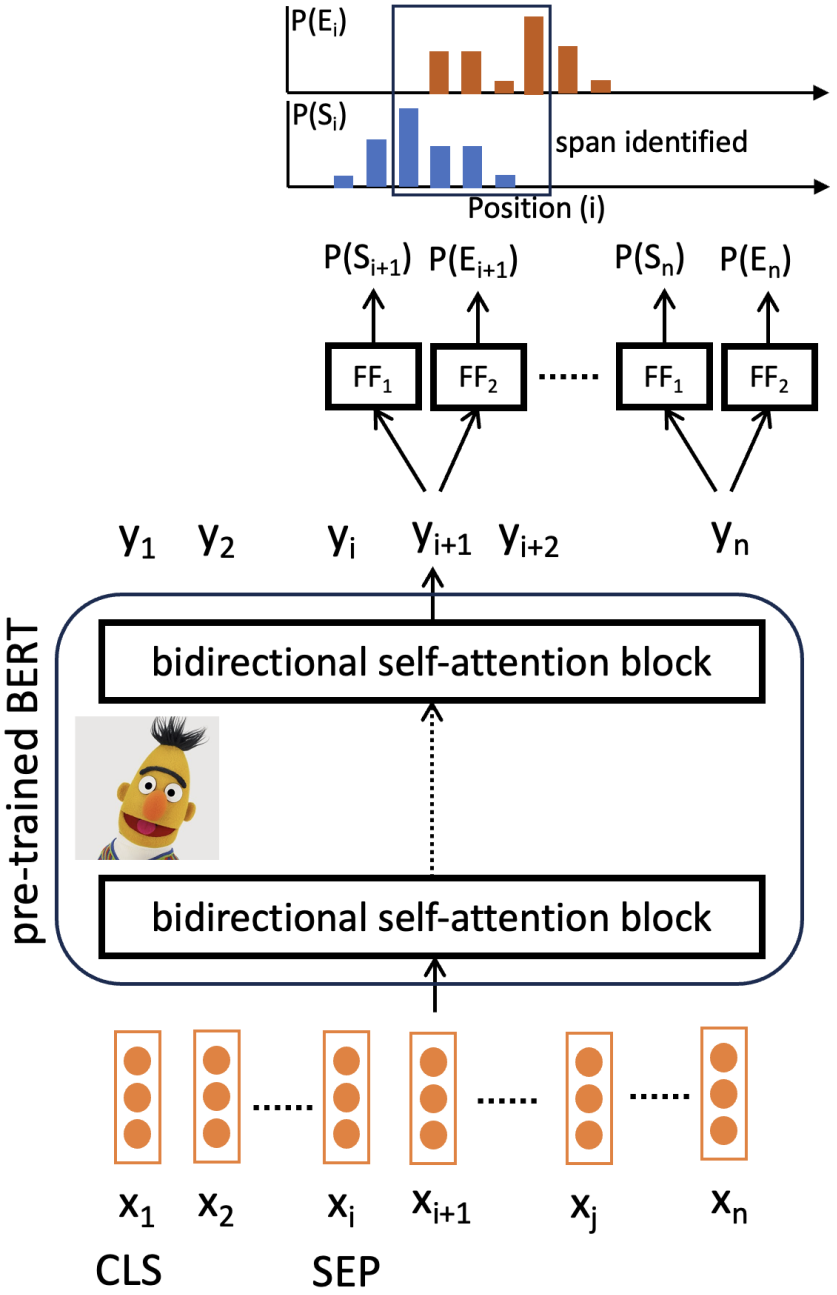


training

BERT transfer learning for token classification

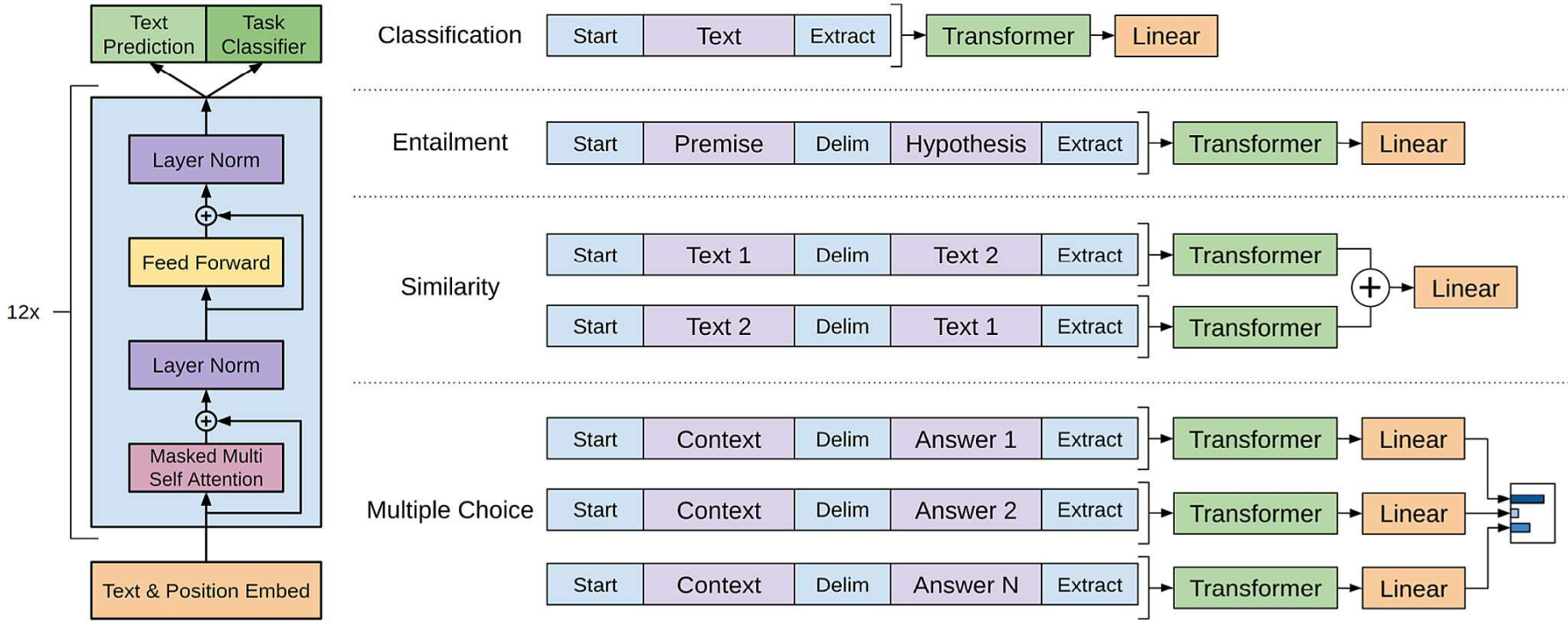


BERT transfer learning for span detection (QA)



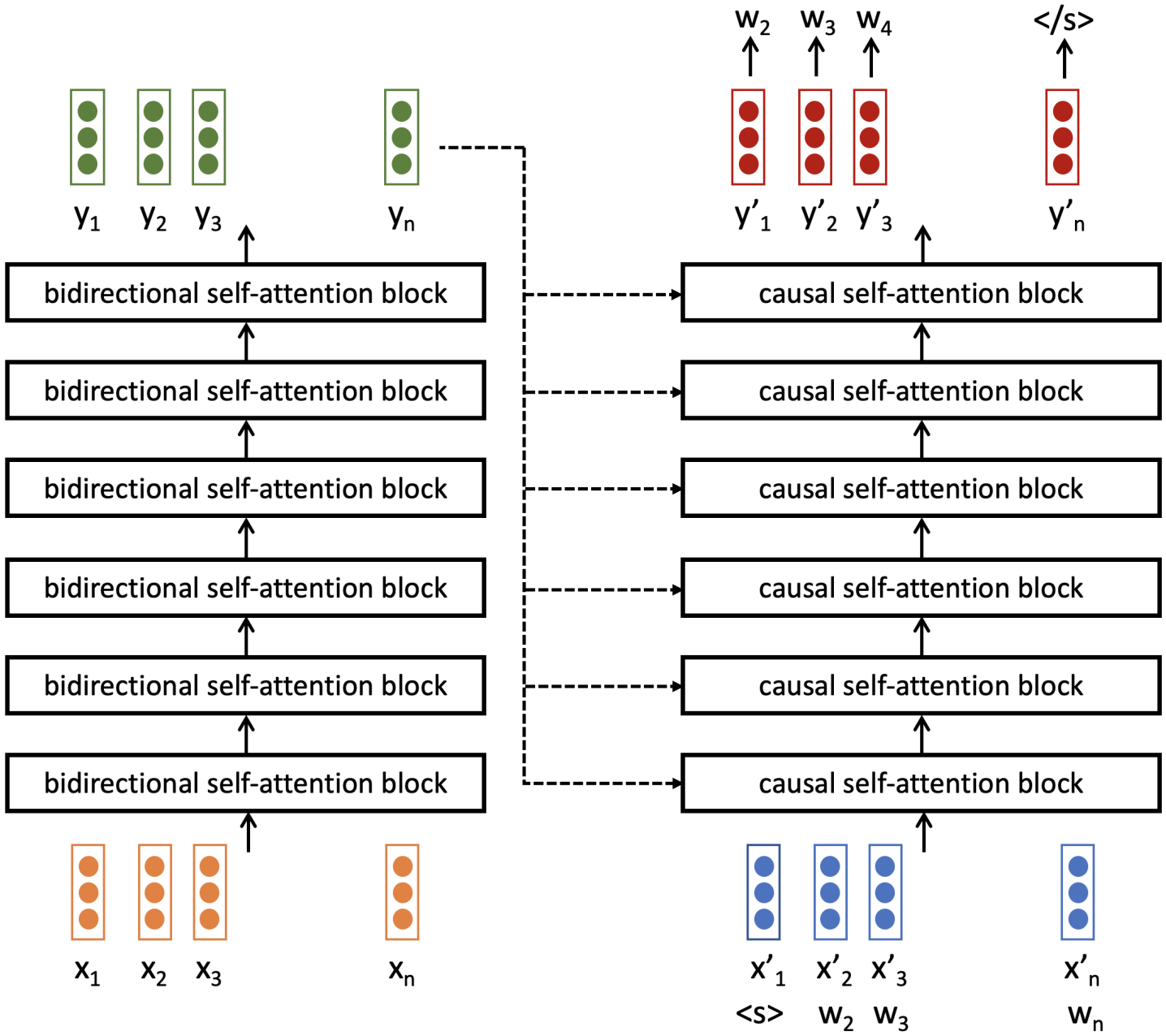
Transfer learning: the GPT case

Add a classifier on top of a pre-trained transformer language model and retrain the whole stuff

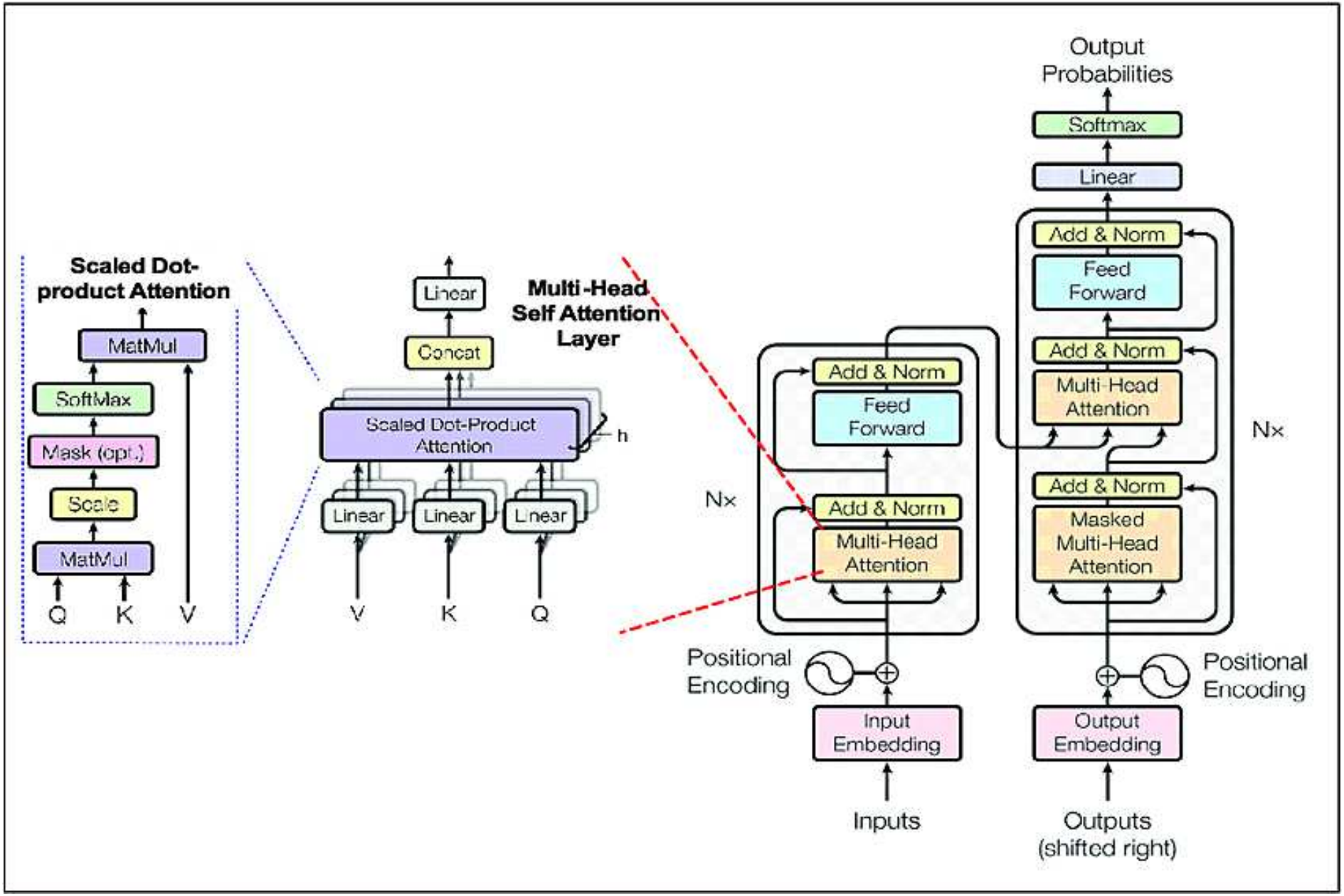


Alex Radford et al. 2018. Improving Language Understanding by Generative Pre-Training

seq2seq transformers

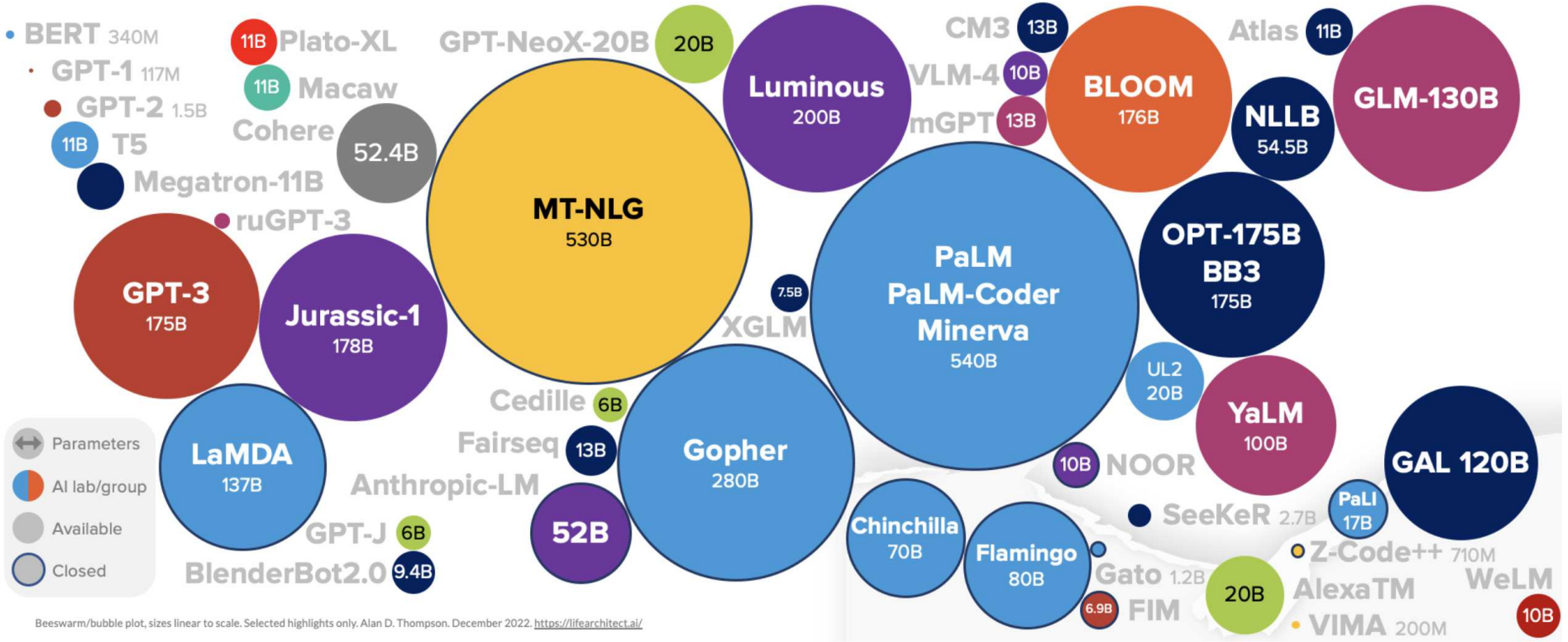


Putting it all together in a *transformer* (zoom)



Vaswani et al., 2017. Attention is all you need.

The transformer pre-trained model frenzy



copied from `https://lifearchitect.ai/timeline`

Transformers made easy (with HuggingFace)

```
> from transformers import BertTokenizer
> from transformers import TFBertModel
> from transformers import TFBertForTokenClassification
> import tensorflow as tf

> name = 'bert-base-cased'

> tokenizer = BertTokenizer.from_pretrained(name)
> encoder = TFBertModel.from_pretrained(name)

> inputs = tokenizer("Hello, my dog is cute", return_tensors="tf")
> outputs = encoder(inputs)
> embedding = outputs.last_hidden_state

> tagger = TFBertForTokenClassification.from_pretrained(name)
> inputs = tokenizer("Hello, my dog is cute", return_tensors="tf")
> outputs = tagger(inputs)
```

and a quick finetuning example of BERT on Google colab

Cooling down with tasks

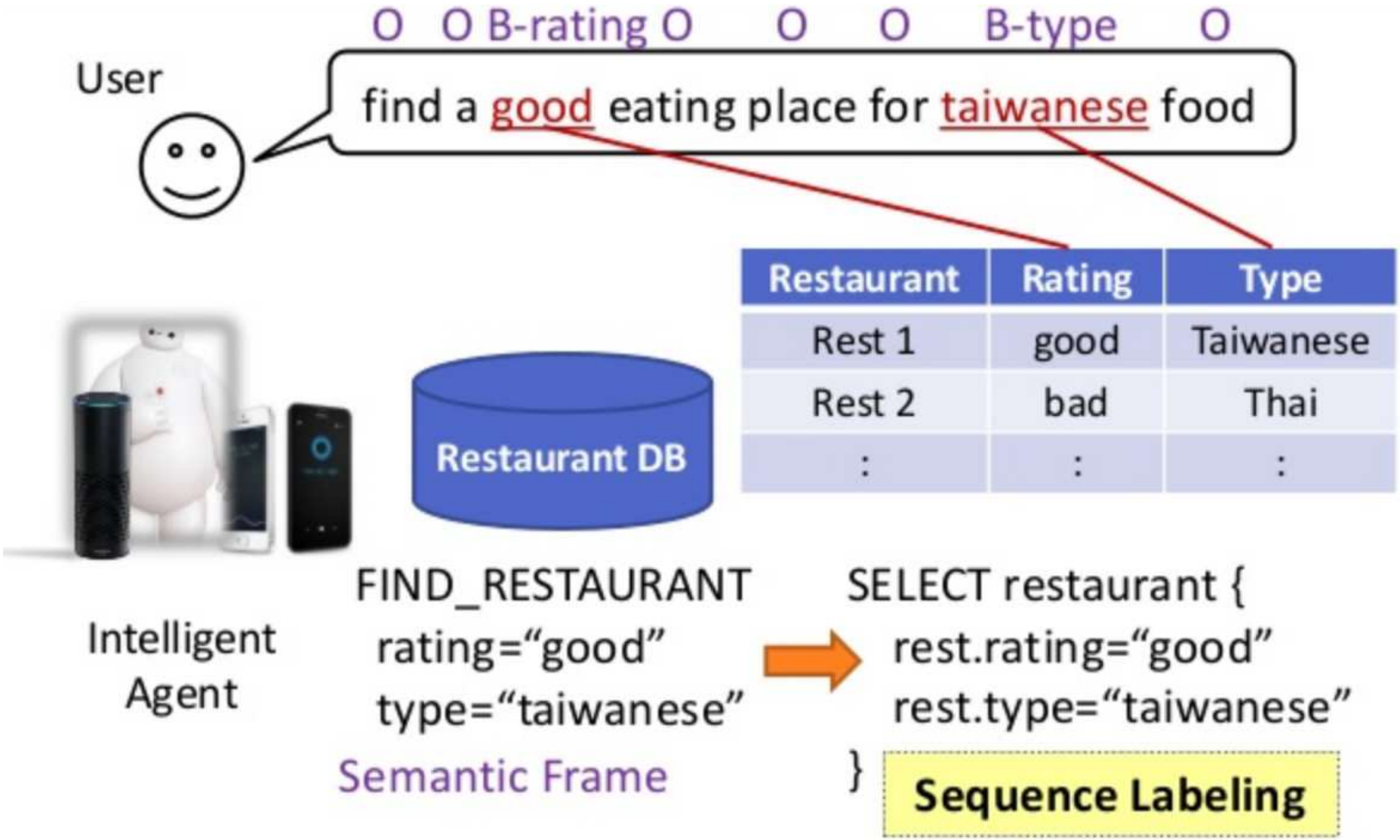
Straightforward applications of models

Many tasks can be cast as direct applications of the models we've seen so far:

polarity/emotion detection	classification
hate speech detection	classification
topic detection	classification
named entity detection	tagging
slot filling in dialogue	tagging
question answering	tagging or seq2seq
machine translation	seq2seq
summarization	seq2seq

But there's however often more to it than apparent to the eyes (a.k.a. NLP is not simply machine learning)!!!!

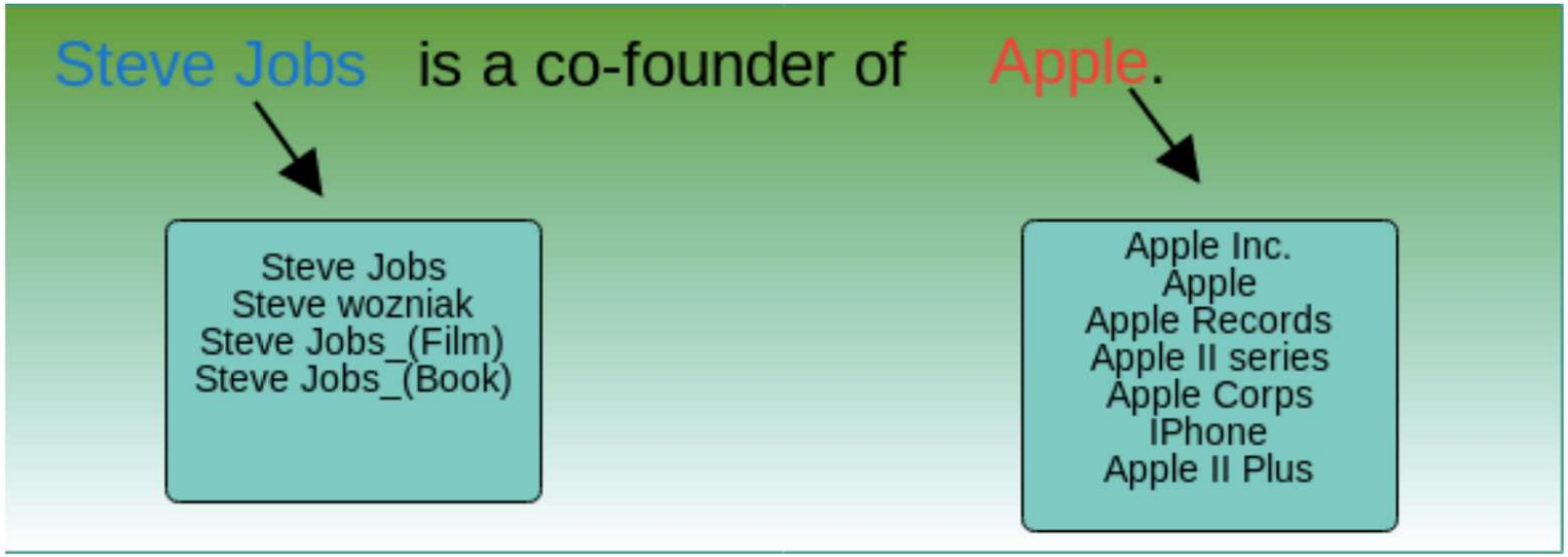
Straightforward applications of models: the slot filling task



Less straightforward applications: the entity linking task

Definition (Entity Linking)

Identifying the entities of a reference knowledge base (KB) that are mentioned in textual documents



- Standard pipeline :
  - Named Entity Recognition (NER)
  - Candidate Entity Generation
  - Candidate Entity Ranking

Cheikh Brahim el Vaigh et al. 2019. Using knowledge base semantics in context-aware entity linking

About opinion mining

Polarity detection is a classification problem but not always an easy one

- co-reference resolution
- use of opinion lexicon and expressions but context matters
  - the phone is small (+) / the display is small (-)

  The Canon EOS M5 is the most enthusiast-friendly EOS M yet. It is a very approachable camera, despite all those buttons and dials. In fact, it's the implementation of this touchscreen that, in general, we're most impressed with. While the lack of 4K video capability is a disappointment, the ability to use the touchscreen to re-position the focus point with a high level of confidence that the camera will smoothly glide the focus to the right point is highly desirable. . .

What about detecting aspects? What about the target of the opinion?

About information extraction

- Text

San Salvador, 19 avril 1989 (ACAN-EFE) – Le président du San Salvador Alfredo Cristani a condamné l'assassinat d'origine terroriste du ministre de la justice Roberto Garcia Alvarado et a accusé du meurtre le Front de Libération National Farbundo Marti.

- Template

INCIDENT

Date: 19 avril 1989
Location: San Salvador
Author: Front de Libération National Farbundo Marti
Victim: Roberto Garcia Alvarado

borrowed from X. Tannier

Combines named entity recognition and linking, syntactic analysis to detect patterns, supervised classification of relations, joint embedding of knowledge bases and texts, etc.


That's all Folks!