The Basics of NLP

Machine Learning for Natural Language Processing, ENSAE 2022

Lecture 1

Benjamin Muller, INRIA Paris

Today Lecture Outline

- **Why Natural Language Processing?**
- **What is Natural Language Processing?**
  - Modelling Framework
  - Tokenization as a first-step task
  - Overview of NLP Tasks
- **A Brief History of NLP**
- **How to tackle any NLP problem?**

Why Natural Language Processing?

What do we use language for?

- We **communicate** using language
- We **think** (partly) with language
- We **tell stories** in language
- We build **Scientific Theories** with language
- We make friends/build **relationships**

Why NLP ?

- **Access Knowledge** (search engine, recommender system…)
- **Communicate** (e.g. Translation)
- **Linguistics** and **Cognitive Sciences** (Analyse Languages themselves)

Why Natural Language Processing?

Amount of online textual data…

- 70 billion web-pages online (1.9 billion websites)
- 55 million Wikipedia articles

…Growing at a fast pace

- 9000 tweets/second
- 3 million mail / second (60% spam)

Internet Stats

Why Natural Language Processing?

Potential Users of Natural Language Processing

- 7.9 billion people use some sort of language (January 2022)
- 4.7 billion internet users (January 2021) (~59%)
- 4.2 billion social media users (January 2021) (~54%)

Data Reportal

Why Natural Language Processing?

What Products ?

- Search: +2 billion Google users, 700 millions Baidu users
- Social Media: +3 billion users of Social media (Facebook, Instagram, WeChat, Twitter...)
- Voice assistant: +100 million users (Alexa, Siri, Google Assistant)
- Machine Translation: 500M users for google translate

Data Reportal

Why is Language Hard to Model?

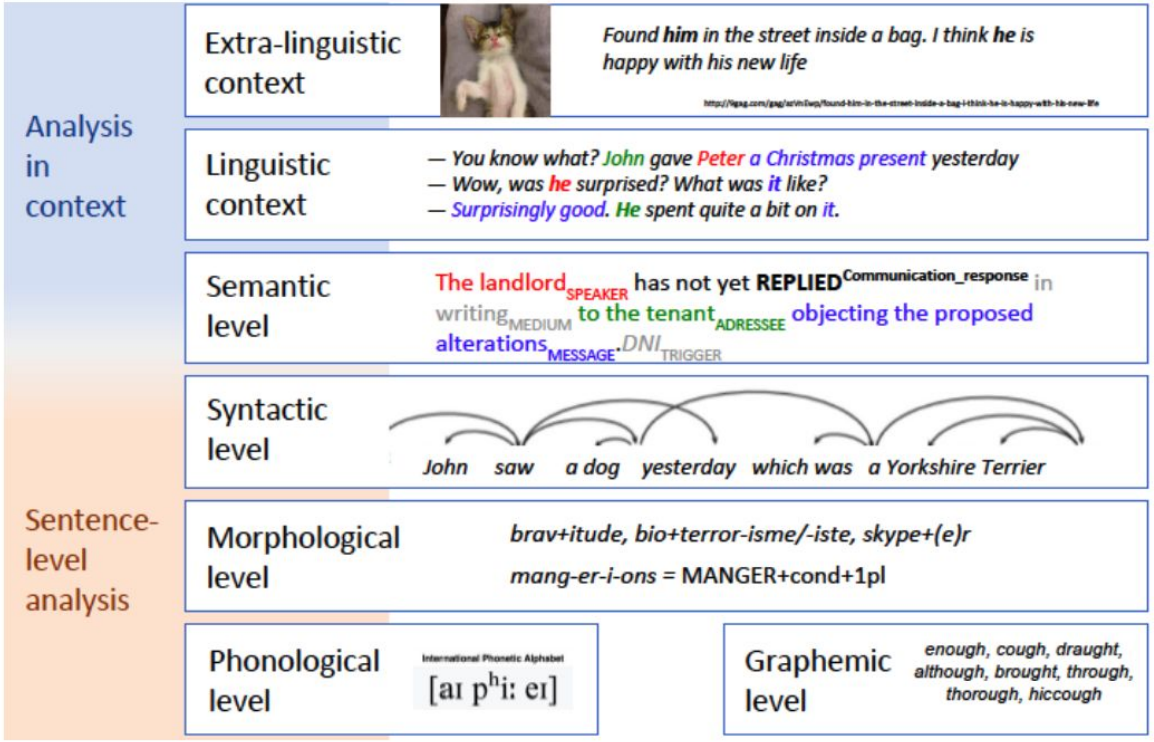
A Definition of Language

Definition 1: *Language is a means to communicate, it is a semiotic system. By that we simply mean that it is a set of signs. A sign is a pair consisting in [...] a signifier and a signified.*

Definition 2: *A sign consists in a phonological structure, a morphological structure, a syntactic structure and a semantic structure*

Krach 2007

The Six Levels of Linguistics Analysis



cf. Sagot, Algorithms for speech and natural language processing, MVA course Material

The 5 Challenges of NLP

1. Productivity
2. Ambiguous
3. Variability
4. Diversity
5. Sparsity

Productivity

Definition

“property of the language-system which enables native speakers to construct and understand an indefinitely large number of utterances, including utterances that they have never previously encountered.” (Lyons, 1977)

➔ **New words, senses, structure** are **introduced in languages all the time**

Examples: *staycation* and *social distance* were added to the Oxford Dictionary in 2021

Ambiguous

Most linguistic observations (speech, text) are open to **several interpretations**

We (Humans) disambiguate - i.e. **find the correct interpretation** - using all kind of signals (linguistic and extra linguistic)

Ambiguity can appear at all levels (phonology, graphemics, morphology, syntax, semantics)

Ambiguous

Syntactic Ambiguity

cf. Sagot

Ambiguous

Semantic Ambiguity

- Polysemy: *e.g.* ***set*** *,* ***arm, head***
  Head of New-Zealand is a woman
- Name Entity: *e.g.* ***Michael Jordan***
  Michael Jordan is a professor at Berkeley
- Object/Color: *e.g.* ***cherry***
  Your cherry coat

Ambiguous

Pragmatic Ambiguity

*Two Soviet ships collide, **one dies***

*Dealers will hear **car talk** at noon*

Ambiguous

Disambiguating can requires Discourse Knowledge

Where can I find **a vegetarian restaurant** in Paris

Here is a list of restaurant in Paris: ....

Give me the top ranked ones, in the 14th arrondissement

Here are the top ranked restaurant in the 14th arrondissement in Paris

How far is the closest **one** from my current location?

Variation

Language Varies at all levels

- Phonetic (accent)
- Morphological, Lexical (spelling)
- Syntactic
- Semantic

Variation Determiners

- Who is talking?
- To Whom?
- Where? *Work, Home, Restaurant*
- When? *19th century, 2008, 2022…*
- About what? *Specialised domain, the Weather,…*

Essentially, the Variability of a language depends on:

- Social Context
- Geography
- Sociology
- Date
- Topic

Diversity

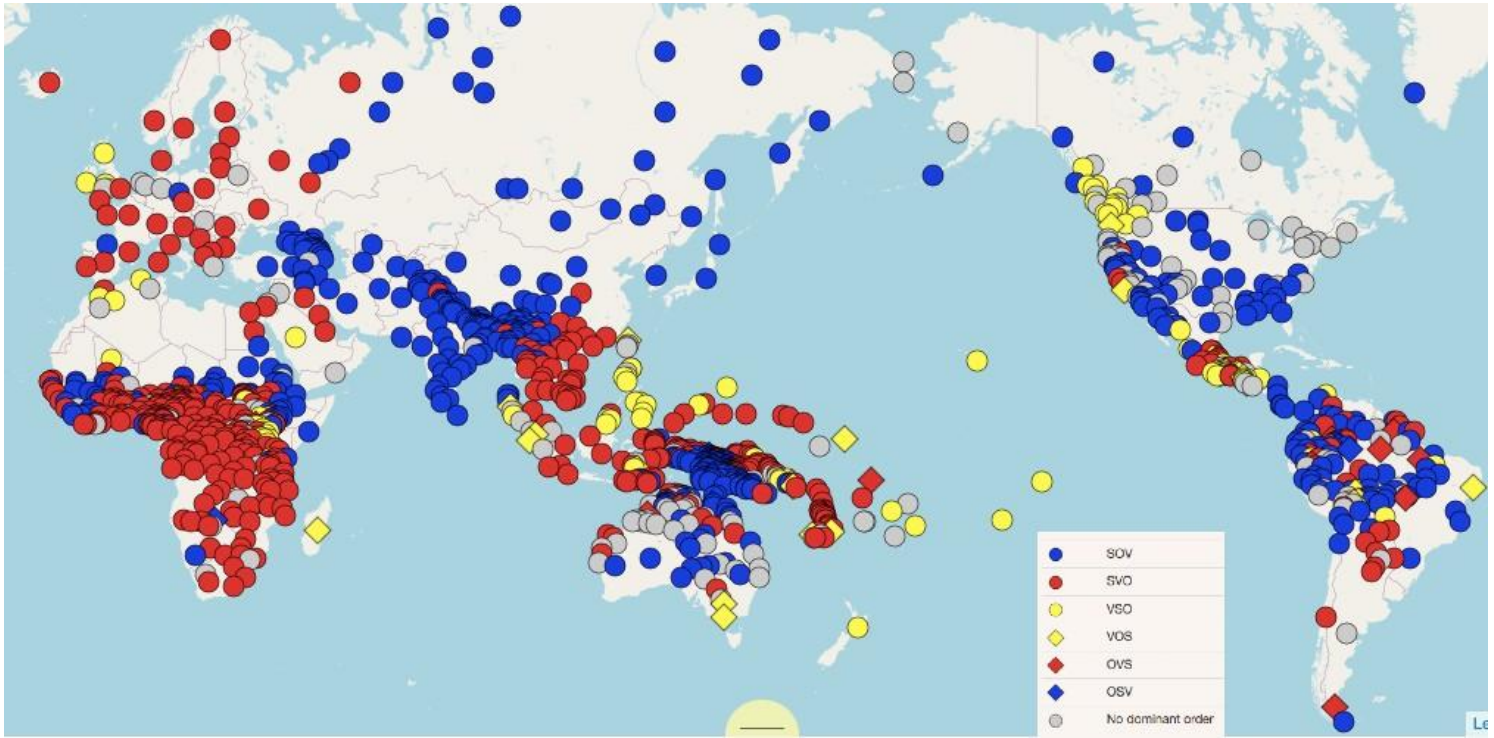
- About **7000 languages** spoken in the world
- About **60%** are found in the **written form** (cf. Omniglot)

Syntactic Diversity

A key characteristics of the syntax of a given language is **the word order**

- **Word order differs** across languages
- Word order degree of freedom also differs across languages
- We characterize word orders with: **Subject (S) Verb (V) Object (O) order**

Syntactic Diversity



(Dyer et. al 2013)

Word Order Freedom And Morphology

- Word orders freedom and morphology are usually related
- **The more freedom in word orders**
  → the less information is conveyed by word positions
  → the more information is carried by each word
  → **the richer the morphology**

English ***cats eat mice***

Russian (O: -ей) ***Кошки едят мышей*** 🔊
Мышей едят кошки 🔊
Едят кошки мышей.
Едят мышей кошки.

Statistical Description of a Corpus

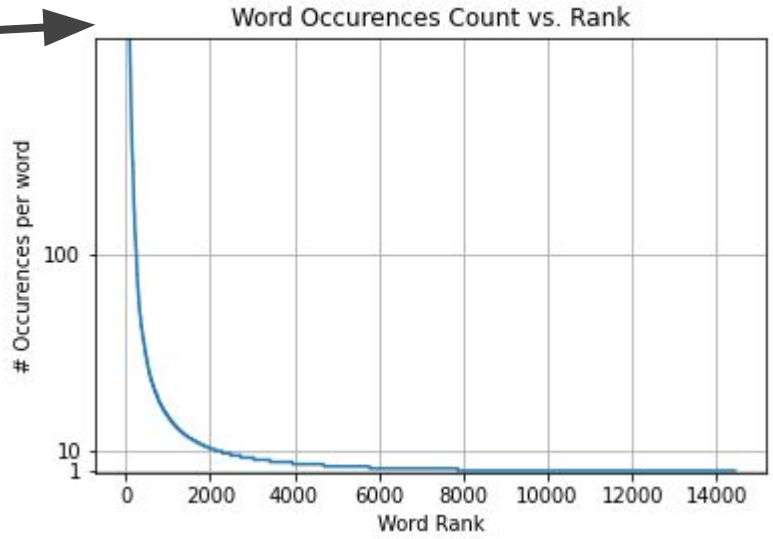
We describe statistically a corpus of 800 scientific articles

Question: If we plot the number of occurrences of each word vs. the rank, what will we observe?

Statistical Description of a Corpus

We describe statistically a corpus of 800 scientific articles

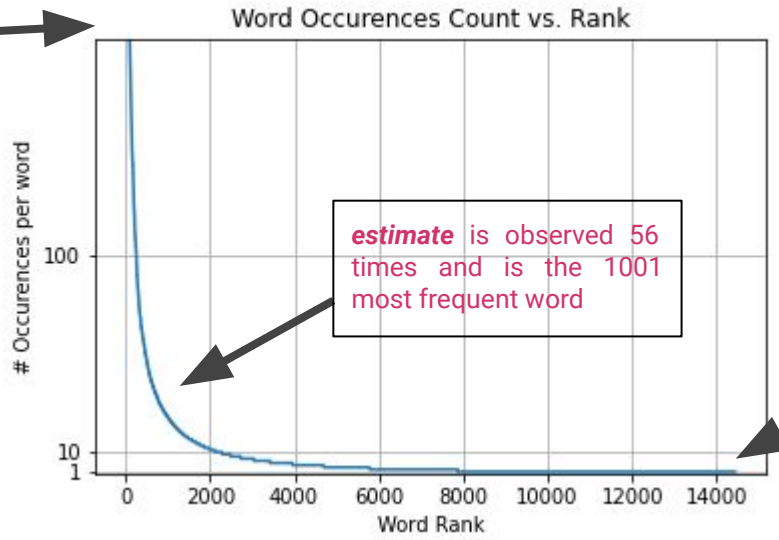
the is the most observed (rank 1) word with 8119 occurrences



Statistical Description of a Corpus

We describe statistically a corpus of 800 scientific articles

the is the most observed (rank 1) word with 8119 occurrences



estimate is observed 56 times and is the 1001 most frequent word

About 6000 Words are observed only 1 time in the dataset (e.g. ***stakeholders, pending, score…)***

Statistical Description of a Corpus

We describe statistically a corpus of 800 scientific articles

➔ In a large enough corpus, word distributions follows ***a Zipf Law ie:***

f_w frequency of entity w
k frequency rank of entity w

$$f_w(k) \propto \frac{1}{k^\theta}$$

Statistical Description of a Corpus

We describe statistically a corpus of 800 scientific articles

➔ In a large enough corpus, word distributions follows ***a Zipf Law ie:***

f_w frequency of entity w
k frequency rank of entity w

$$f_w(k) \propto \frac{1}{k^\theta}$$

- Zipf law is a Power relation between the rank and frequency
  *The most frequent entities are **much more frequent** than the less frequent ones*

- Under a Zipf law, log(fw) and log(k) are linearly related

Statistical Description of Language

Zipf Distributions are observed not only for words but with many other units of language (sounds, syntactic structure, name entities…)

Consequence

➡ A large number of units are observed in language with very low frequency i.e. **Sparsity**

➡ Very challenging for NLP

What is Natural Language Processing?

In a nutshell, NLP consists in handling the complexities of natural languages "to do something"

- Raw Text / Speech → Structured Information
- Raw Text / Speech → (Controlled) Text/Speech

In this course we will focus **on textual data**

Framework

We assume:

- A **token** is the basic unit of discrete data, defined to be an item from a vocabulary indexed by 1, ..., V.
- A **document** is a sequence of N words denoted by **d = (w1,w2, ...,wN)**, where wn is the N-th word in the sequence.
- A **corpus** is a collection of M documents denoted by D = (d1, d2, ..., dM)

Example: *Wikipedia, All the articles of the NYT in 2021…*

Token

With regard to our end task, a token can be:

- A word
- A sub-word: *e.g. a sequence of 3 characters*
- A character
- An sequence of characters (sometimes a word, sometimes several words, sometimes a sub-word…)

Document

A Document can be:

- A Sentence
- A Paragraph
- A sequence of characters

Text Segmentation

Definition: Text Segmentation is the process of splitting raw text (i.e. list of characters) into **units of interest**.

Two level of segmentation (usually) required :

- Split raw text into **modeling units** (ex: sentence, paragraph, 1000 characters, web-page...)
- Split modeling units into sequence of **basic units** (referred as tokens) (e.g: words, word-pieces, characters, ...)

Two distinct approaches:

- **Linguistically informed** e.g. word, sentence segmentation...
- **Statistically informed** e.g. frequent sub-words (word pieces, sentence pieces...)

Tokenization

Definition: Tokenization consists in *segmenting* raw textual data into tokens:

Tokenization

Definition: Tokenization consists in *segmenting* raw textual data into tokens:

Can be framed as a character level task

input: *une industrie métallurgique existait.*
output: IIIEIIIIIIIIIIIIIEIIIIIIIIIIIIIIIIIEIIIIIIIIIIEE

- **Easy task** for most languages and domains
- Can be **very complex in some cases** (Chinese, Social Media...)

NLP Tasks: Modeling Framework

Let (X, Y) a pair of random variable. X may characterize tokens or documents. Modeling an NLP task consists in estimating the conditional probability $Y|X$ in order to predict Y with X.

$$p(Y|X)$$

Tasks Taxonomy

- If *Y* is a single label and X a sequence of tokens (e.g. a sentence): Sequence Classification
- If we have one label per token: Sequence Labelling
- If Y is a sequence of tokens: Sequence Prediction
- If Y is a graph, a tree or a complex structured output: Structure Prediction

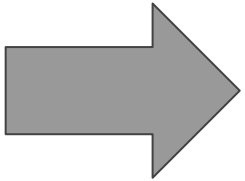
Document Classification

Europe

Germany's minimum wage hike will not cost jobs -labour minister

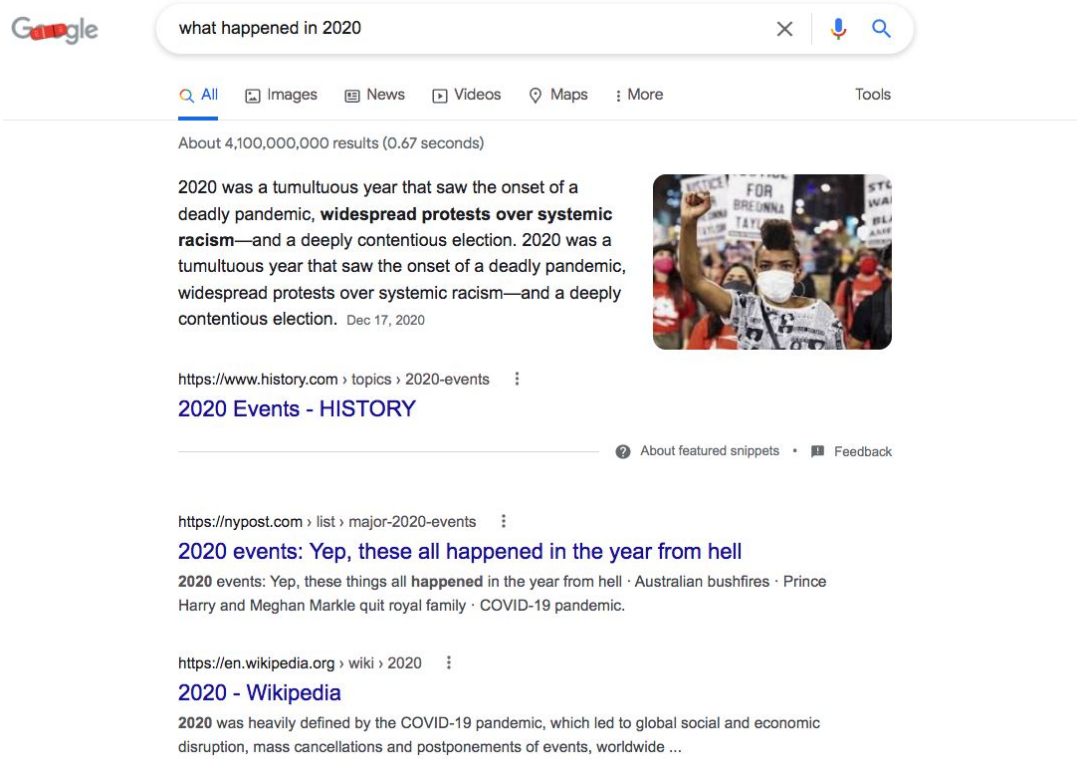
BERLIN, Jan 21 (Reuters) - Germany's planned minimum wage hike to 12 euros ($13.61) per hour from October means a pay rise for over 6 million people across the country and should not cost jobs contrary to critics, Labour Minister Hubertus Heil said on Friday.

Increasing the German minimum wage, currently 9.82 euros per hour and will increase to 10.45 euros per hour from July, to 12 euros per hour was one of the key election promises of Chancellor Olaf Scholz and his Social Democrats.



- **Politics**
- **Economy**
- **Travel**
- **....**
- **Geopolitics**

Document Ranking (Retriever)



NLP Tasks: Part-of-Speech Tagging

POS Tagging: Find the **grammatical category** of each word

[My , name, is, Bob, and, I, live, in, NY, !]

NLP Tasks: Part-of-Speech Tagging

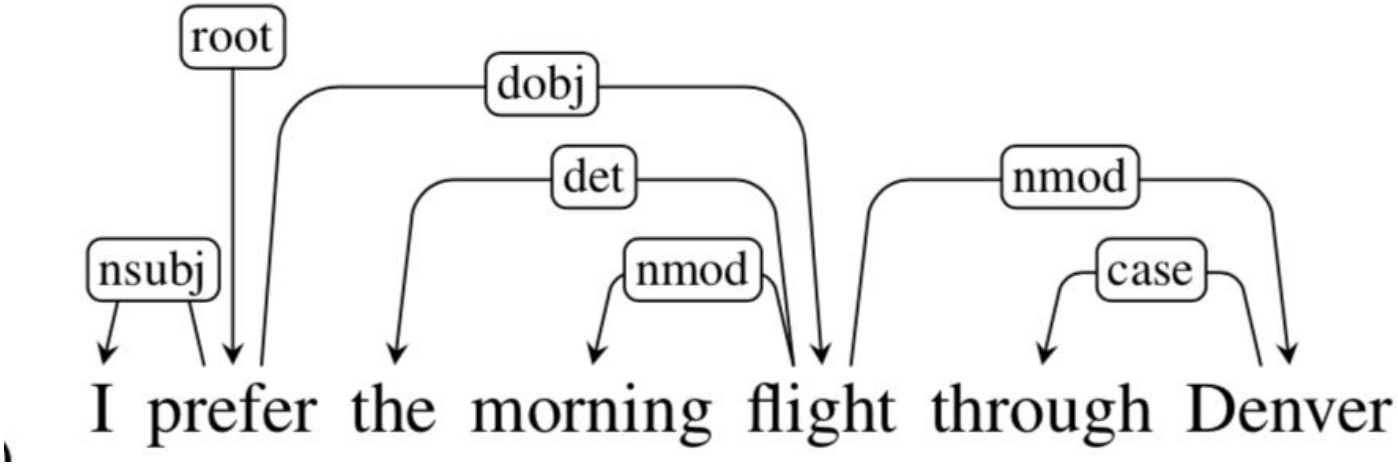
POS Tagging: Find the **grammatical category** of each word

[My , name, is, Bob, and, I, live, in, NY, !]

[PRON , NOUN, VERB, NOUN, CC, PRON, VERB, PREP, NOUN, PUNCT]

Syntactic Parsing

Syntactic Parsing consists in **extracting the syntactic structure** of a sentence. For instance, **Dependency Parsing** (here) predicts an acyclic directed graph (a **tree**)



Slot-Filling / Intent Detection

Intent Detection is a sequence classification task that consists in **classifying the intent of a user** in a pre-defined category.

Slot-Filling is a sequence labelling task that consists in identifying **specific parameters in a user request**.

Can you please play Hello from Adele ?

Intent: ***play_music***
Slots: [*Can, you, please, play, Hello, from, Adele, ?]*
*[O , O , O , O , **SONG**, O , ARTIST, O]*

Semantic Role Labelling (SLR)

SRL is the task of finding the **semantic role** of each predicate in a sentence.

Given a sentence, SRL predicts: *who did what to whom, when, **where, why, how***

NLP Tasks: Name Entity Recognition

NER: Find the **Name-Entities** in a sentence

[My ,	*name,*	*is,*	*Bob,*	*and,*	*I,*	*live,*	*in,*	*NY,*	*!]*
[O ,	*O,*	*O,*	***PERSON***,	*O,*	*O,*	*O,*	*O,*	***LOCATION***,	*O]*

Machine Translation

INPUT: ***My name is Bob and I live in NY !***

OUTPUT: ***Je m'appelle Bob et je vis à New-York!***

Question Answering

INPUT: *How many episodes in season 2 breaking bad?*

OUTPUT: ***13***

How do we solve an NLP Problem?

Each NLP Problem is unique

➔ **No Universal Method to solve them all**

➔ Have a toolkit in mind of methods (symbolic, statistics and deep learning based)

NLP Engineering: Find **what works best** for your given method based on what has been done already in the literature, by colleagues…

NLP Research: Find a **better** way to do (more accurate, cheaper, faster…)

Brief History of NLP

Brief History of NLP

Symbolic

1940-2000

Focus on rule-based systems, formal grammars

Development of linguistic resources (lexicon, ontologies, grammars…)

Statistical Learning

1990-2010

Statistical learning theory (SVM, Random Forest), Graphical Probabilistic Models (e.g. LDA, HMM)

Development of annotated datasets

Deep Learning

2010-Today

Deep Learning Architecture (Transformer)

Transfer Learning in NLP (word2vec, BERT, CamemBERT, GPT)

More compute, larger (raw) dataset, Open Source Deep Learning Libraries

1949: First Machine Translation "Model"

1949 ***Memorandum on Translation,*** Warren Weaver
First to propose the idea of using "*electronic computers*" to do translation

- Using Shannon's information theory to frame Machine Translation as a ***cryptographic*** problem
- **Modeling context** to disambiguate between word senses
- "*Going down*" from each language to ***universal language*** in order to translate

1964: ELIZA First Conversational Bot

Men are all alike.
IN WHAT WAY
They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
It's true. I am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
Perhaps I could learn to get along with my mother.
TELL ME MORE ABOUT YOUR FAMILY
My mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
My father.
YOUR FATHER
You are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE

Natural Language Processing WorkFlow

Assume we have a **Research, Engineering, Product Problem**

1. Define a **NLP System** to solve it
   Split into **modules**, each one performing a **task**
2. Define **Evaluation Metric(s) for your system and submodules**
3. **Collect Data** to build/train your models
4. Build **Baseline Models** (i.e. most simple model you can think of that have a non trivial performance metric)
5. Build **Better Models** using symbolic/statistical/DL methods

Representing Text with Vectors

Machine Learning for Natural Language Processing, ENSAE 2022

Lecture 2

Benjamin Muller, INRIA Paris

Today Lecture Outline

- **Representing Words** in Vectors
- **Representing Documents** in Vectors

Representation Techniques

➔ **Hand-Crafted Feature-Based** Representation
➔ **Count-Based** Representation
➔ **Prediction-Based** Representation

Framework

We assume:

- A **token** is the basic unit of discrete data, defined to be an item from a vocabulary indexed by 1, ..., V.
- A **document** is a sequence of N words denoted by **d = (w1,w2, ...,wN)**, where wn is the Nth word in the sequence.
- A **corpus** is a collection of M documents denoted by D = (d1, d2, ..., dM)

Framework

We assume:

- A **token** is the basic unit of discrete data, defined to be an item from a vocabulary indexed by 1, ..., V.
- A **document** is a sequence of N words denoted by **d = (w1,w2, ...,wN)**, where wn is the Nth word in the sequence.
- A **corpus** is a collection of M documents denoted by D = (d1, d2, ..., dM)

In this lecture, a token will be **a word**

What is a word?

There are many ways to define a word based on what aspect of language we consider (typography, syntax, semantics…)

Definition (Semantic):

*Words are **the smallest linguistic expressions** that are **conventionally** associated with a **non-compositional meaning** and can be articulated in isolation to convey semantic content.**

*Stanford Encyclopedia of Philosophy

Objective

Given a vocabulary w1,..,wV and a corpus D, our goal is to associate each word with a representation?

What do we want from this representation?

- identify a word (bijection)
- capture the similarities of words (based on morphology, syntax, semantics,...)
- Help us solve downstream tasks

NB: Vector-based representations of text are called ***embedding***

1-Hot Encoding

Traditional way to represent words **as atomic symbols** with a unique integer associated with each word:

{1=movie, 2=hotel, 3=apple, 4=movies, 5=art}

Equivalent to represent words as 1-hot vectors:

movie = [1, 0, 0, 0, 0]
hotel = [0, 1, 0, 0, 0]
...
art = [0, 0, 0, 0, 1]

1-Hot Encoding

Most basic representation of any textual unit in NLP. Always start with it.

Implicit assumption: word vectors are an **orthonormal basis**

- orthogonal
- normalized

Problem 1: Not very informative

➔ Weird to consider “movie” and “movies” as independent entities or to consider all words equidistant:

$$\| \text{house} - \text{home} \| = \| \text{house} - \text{car} \|$$

Problem 2: Polysemy

➔ Should the Mouse of a computer get the same vector a the mouse animal?

Hand-Crafted Feature Representation

Example of potential features:

- Morphology: prefix, suffix, stem...
- Grammar: part of speech, gender, number,...
- Shape: capitalization, digit, hyphen

Those features can be defined based on **relations** to other words

- Synonyms of…
- Hypernyms of…
- Antonyms of...

Hand-Crafted Feature Representation

Example of potential features:

- Morphology: prefix, suffix, stem...
- Grammar: part of speech, gender, number,...
- Shape: capitalization, digit, hyphen

Those features can be defined based on **relations** to other words

- Synonyms of…
- Hypernyms of…
- Antonyms of...

We present one popular hand-crafted semantically based representation of words ⇒ the WordNet

WordNet

Definition: a (word) sense is a discrete representation **of one aspect of the meaning of a word**

WordNet is a large lexical database of **word senses** for English and other languages

WordNet

- Word types are grouped into (cognitive) synonym sets: **synsets**
  S09293800={*Earth,earth,world,globe*}

- **Polysemous** words: assigned to **different synsets**
  S14867162 ={*earth,ground*}

- Contains glosses for synsets:
  the 3rd planet from the sun; the planet we live on

- **Noun/verb synsets:** organized in **hierarchy**, capturing IS-A relation
  apple IS-A *fruit*

WordNet

X is a **hyponym** of Y if **X is an instance of Y**:

cat is a hyponym of animal

X is a **hypernym** of Y if **Y is an instance of X**:

animal is a hypernym of cat

X and Y are **co-hyponyms** if they have the **same hypernym**:

cat and dog are co-hyponyms

X is a **meronym** of Y if **X is a part of Y**:

wheel is a meronym of car

X is a **holonym** of Y if **Y is a part of X**:

car is a holonym of wheel

WordNet

Similarity

$$\text{sim}(S_1, S_2) = \frac{1}{\text{length}(\text{path}(S_1, S_2))}$$ ***t*:**

Idea: *The shorter the **hypernym/hyponym** path from one synset to another the higher is the similarity*

Similarity

b $$\text{sim}(w_1, w_2) = \max_{\substack{S_1, S_2 \\ w_1 \in S_1 \\ w_2 \in S_2}} \text{sim}(S_1, S_2)$$ ***ls*:**

Example: sim(*dog*, *cat*) = ? 👉 notebook

Hand-Crafted Representations: Limits

- Requires **a lot of human annotations**
- **Subjectivity** of the annotators
- **Does not adapt** to new words (languages are not stationary!): *Mocktail, Guac, Fave, Biohacking* were added to the Merriam-Webster Dictionary in 2018

➔ It **does not scale** easily to new languages, new concepts, new words...

How to Infer “Good” Representations with Data?

Distributional Hypothesis

You shall know a word by the company it keeps” Firth (1957)

Idea: Model the *context* of a word to build **its vectorial representation**

Example: What is the meaning of “**Bardiwac**” ?

Distributional word representation in a nutshell

1. Define what is ***the context*** of a word
2. **Count** how many times each target word occurs in this context
3. Build vectors out of (a function of) these context occurrence counts

$$x_w = f(w, Context(w))$$

How to define *"the context"* of a word?

It can be defined as

- **The surrounding words** (left and right words)
- All the other words of the sentence/the paragraph
- All the words **after preprocessing and filtering-out some words**

How to Model the Context to get

$$x_w = f(w, Context(w))$$

Approach 1: Count-Based

1. Measure frequency of words in the context for each word in the vocabulary
2. Define vector representations based on those frequency

How to Model the Context to get

$$x_w = f(w, Context(w))$$

Approach 1: Count-Based

1. Measure frequency of words in the context for each word in the vocabulary
2. Define vector representations based on those frequency

Approach 2: Prediction Based

Counting the Occurences of the words in the context of dog

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

co-occurence # dog	
barked	++
park	+
owner	+
leash	+

Co-Occurrence Matrix

	leash	walk	run	owner	pet	barked
dog	3	5	2	5	3	2
cat	0	3	3	2	3	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	1	3	0	0

Define vector representation based on the Co-Occurrence

	leash	walk	run	owner	pet	barked	the
dog	3	5	2	5	3	2	8
lion	0	3	2	0	1	0	6
light	0	0	0	0	0	0	5
bark	1	0	0	2	1	0	0
car	0	0	1	3	0	0	3

- **Naïve Approach:** Take the row of the co-occurrence matrix

Define vector representation based on the Co-Occurrence

	leash	walk	run	owner	pet	barked	the
dog	3	5	2	5	3	2	8
lion	0	3	2	0	1	0	6
light	0	0	0	0	0	0	5
bark	1	0	0	2	1	0	0
car	0	0	1	3	0	0	3

Limits: 👉 notebook

- Representations depends **on the size of the corpus**
- Frequent words impacts a lot the representations
- Representations very sensitive to change in very infrequent words

Solution: Pointwise Mutual Information (PMI)

Idea: Instead of absolute co-occurrence statistics, use probability (relative) of co-occurrences

$$PMI(w_1, w_2) = log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

Solution: Pointwise Mutual Information (PMI)

Idea: Instead of absolute co-occurrence statistics, use probability (relative) of co-occurrences

$$PMI(w_1, w_2) = log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

Intuition

- **The more dependent *dog* and *cat*** the closer P(dog, cat) is from P(dog)P(cat) **the smaller the PMI**

Solution: Pointwise Mutual Information (PMI)

Idea: Instead of absolute co-occurrence statistics, use probability (relative) of co-occurrences

$$PMI(w_1, w_2) = log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

Intuition

- **The more dependent *dog* and *cat*** the closer P(dog, cat) is from P(dog)P(cat) **the smaller the PMI**

$$PMI(w_1, w_2) = log \frac{\frac{1}{n_{pairs}} \#\{(w_1, w_2)\}}{\frac{1}{n_{word}} \#\{w_1\} \frac{1}{n_{word}} \#\{w_2\})}$$

Pointwise Mutual Information (PMI)

	leash	walk	run	owner	pet	barked	the
dog	2.75	2.24	3.16	2.24	2.75	3.16	1.77
lion	0	2.75	3.16	0	3.85	0	2.06
car	0	0	3.85	2.75	0	0	2.75

Word embedding vectors are the row of the PMI matrix

- We take usually take the Positive PMI (assigned to 0 when negative) + Smooth unobserve pairs (Laplace smoothing: add 1)
- Does not depend on size of the corpus (the PMI is **normalized**)
- Much less sensitive to change in frequent words (**log**)

Pointwise Mutual Information (PMI)

Limits:

- **Very large** matrix **O(V^2)** ! Very large word vectors
- Hard to use large vectors in practice (i.e. 1M word vocabulary)
- Cannot compare word vectors estimated on 2 different corpora unless they have exactly the same vocabulary!

Idea: Build vectors with predefined size based on the PMI matrix

➔ **Dimensionality Reduction Technique**

Singular Value Decomposition (SVD)

We can decompose the PMI Matrix with SVD

1. We build a symmetric definite matrix based on the PMI
2. We decompose it ıe SVD

$$P = \mathbf{U}_d \Sigma_d \mathbf{V}_d^T$$

3. U is of size (V, d) gives us the representation of each word in a latent/embedding space

Properties of SVD:

- U is a orthonormal matrix
- U aggregates the highest variance of the original word embeddings

Limits of Dimensionality Reduction Approach

- Need to store a matrix of size O(V^2)
- SVD is O(V*d^2)

➔ It is inefficient to build a very large matrix for reducing:
Can we do both simultaneously?

Solution: Prediction-Based Word Embedding Approaches

Prediction-Based Model

Idea:

- Learn directly **dense word vectors**
- Using the ***distributional hypothesis***
- **Implicitly**, by **parameterizing words as dense vectors**
- and **learning to predict context** using this parametrization

Many word embedding methods use these ideas successfully

We present the ***word2vec skip-gram*** model (one of the most popular)

Word2Vec Skip-Gram Model

For each Sentence

1. Sample ***a target word***
2. Predict ***context words*** defined as words in a fixed window from the target word

my dog is barking and chasing its tail

Word2Vec Skip-Gram Model

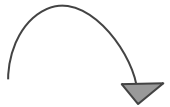
For each Sentence

1. Sample ***a target word***
2. Predict ***context words*** defined as words in a fixed window from the target word

Word2Vec Skip-Gram Model

Given $d \in \mathbb{N}$, let $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$ two word representations (or word *embedding*) matrices. For each sequence $(w_1, .., w_T)$:

- Pick a *focus* word w, associated to the vector $\mathbf{w} \in \mathbb{R}^d$ ($\mathbf{w}$ is the row associated to w in $\mathbf{W}$)
- Pick a *context* word c, associated to the vector $\mathbf{c} \in \mathbb{R}^d$ ($\mathbf{c}$ is the row associated to c in $\mathbf{C}$)
- Maximize $\max_{\mathbf{W} \in \mathbb{R}^{(V,d)}, \mathbf{C} \in \mathbb{R}^{(V,d)}} log\, p(c|w)$ (*maximum likelihood estimator*)



my dog is barking and chasing its tail

Word2Vec Skip-Gram Model

1. How to define $log\, p(c|w)$?
2. How to optimize $log\, p(c|w)$?

Word2Vec Skip-Gram Model

1. **How to define** $log\, p(c|w)$ **?**
2. **How to optimize** $log\, p(c|w)$ **?**

Intuition

- This is a classification problem
- The labels we want to predict are the context words
- Classification with a very large number of labels (V~100K)

Ideas:

➔ **Softmax**

➔ Simplify the softmax with **Negative Sampling** for Efficiency

Word2Vec Skip-Gram Model

Softmax of dot-products context vs. words vectors:

$$p(c|w) = \frac{e^{\mathbf{w.c}}}{\sum_{\mathbf{v}} e^{\mathbf{w.v}}}$$

We compute the log-likelihood, **our objective function**, as:

$$log\, p(c|w) = \mathbf{w.c} - log \sum_{\mathbf{v}} e^{\mathbf{w.v}}$$

Limits: O(V) to compute the loss (at every iteration)

➔ **Negative Sampling**

Word2Vec Skip-Gram Model: Negative Sampling

Idea: Instead of computing the probability objective over the entire vocabulary (all the ***V-1*** negative context words)

➔ We sample ***K words that are not in the context of w*** $v \in N_K$ *(K<<V)*

Word2Vec Skip-Gram Model: Negative Sampling

Idea: Instead of computing the probability objective over the entire vocabulary (all the ***V-1*** negative context words)

➔ We sample ***K words that are not in the context of w*** $v \in N_K$ *(K<<V)*

New objective function:

$$\sigma(\mathbf{w}, \mathbf{c}) + \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w}, \mathbf{v}) \quad \text{with} \quad \sigma(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + e^{-\mathbf{x}.\mathbf{y}}}$$

Word2Vec Skip-Gram Model: Negative Sampling

Idea: Instead of computing the probability objective over the entire vocabulary (all the ***V-1*** negative context words)

➔ We sample ***K words that are not in the context of w*** $v \in N_K$ *(K<<V)*

New objective function:

$$\sigma(\mathbf{w}, \mathbf{c}) + \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w}, \mathbf{v}) \quad \text{with} \quad \sigma(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + e^{-\mathbf{x}.\mathbf{y}}}$$

Complexity?

Word2Vec Skip-Gram Model: Negative Sampling

Idea: Instead of computing the probability objective over the entire vocabulary (all the ***V-1*** negative context words)

➔ We sample ***K words that are not in the context of w*** $v \in N_K$ *(K<<V)*

New objective function:

$$\sigma(\mathbf{w}, \mathbf{c}) + \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w}, \mathbf{v}) \quad \text{with} \quad \sigma(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + e^{-\mathbf{x}.\mathbf{y}}}$$

➔ **O(K) to compute** with K independent of V

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

- ### Step 1: Sampling
- Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)
- Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$
- we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$
- Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples
- ### Step 2: Compute loss
- $l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w, v})$
- ### Step 3: Parameter update with SGD
- $\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$
- $\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for ~~step t in $0..T$~~ **do**

Step 1: Sampling

Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$

Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

Step 2: Compute loss

$l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\ \sigma$**(-w, v)**

Step 3: Parameter update with SGD

$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

$\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for *step* t *in* $0..T$ **do**

Step 1: Sampling

Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$

Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

Step 2: Compute loss

$l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\ \sigma(\mathbf{-w, v})$

Step 3: Parameter update with SGD

$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

$\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

Step 1: Sampling

Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$

Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

Step 2: Compute loss

$l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w, v})$

Step 3: Parameter update with SGD

$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

$\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

- ### Step 1: Sampling
- Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)
- Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$
- we note $w = w_i, c = w_j$ represented by vectors **w** in **W** and **c** in **C**
- Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in **C** # Negative samples
- ### Step 2: Compute loss
- $l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\ \sigma$**(-w, v)**
- ### Step 3: Parameter update with SGD
- $\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$
- $\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Loop over the dataset E times (number of epochs)

Complexity: O(d*K*T)

➔ **No Memory bottleneck**

➔ **Scale to Billion-tokens datasets**

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

- ### Step 1: Sampling
- Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)
- Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$
- we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$
- Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples
- ### Step 2: Compute loss
- $l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w, v})$
- ### Step 3: Parameter update with SGD
- $\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$
- $\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Skip-Gram Model & the PMI

(Levy & Goldberg 2014) showed that

- Estimating the embedding matrix with Skip-Gram and Negative Sampling (SGNS)…
- …is equivalent to computing the shifted-PMI matrix

Word2Vec Skip-Gram Model & the PMI

(Levy & Goldberg 2014) showed that

- Estimating the embedding matrix with Skip-Gram and Negative Sampling (SGNS)…
- …is equivalent to computing the shifted-PMI matrix

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = PMI(w_i, c_j) - \log k$$

Word2Vec

- Still very popular in practice
- Works very well with Deep Learning architecture (e.g. LSTM models) to model specific tasks (e.g. NER)
- Recently "beaten" by contextualized approaches (BERT)

Extensions

- Lots of variant of the Skip-Gram exists (CBOW, Glove…)
- Multilingual setting: build shared representations across languages (fastext)

Limits

- Doesn't model morphology
- **Fixed Vocabulary**: What if we add new tokens in the vocabulary?
- **Polysemy**: Each token has a unique representation (e.g. cherry)

Evaluation of Word Embeddings

How to evaluate the quality of word embeddings?

Extrinsic Evaluation

- Use them in a task-specific model and measure the performance on your task (cf. lecture 5 & 6)

Intrinsic Evaluation

➔ **Idea:** ***"similar" words should have similar vectors***

What do we mean by "similar" words?

- Morphologically similar: e.g. *computer, computers*
- Syntactically similar: e.g. determiners
- Semantically similar: e.g. *animal , cat*

Intrinsic Evaluation of Word Embeddings

How to evaluate the quality of word embeddings?

Qualitative Evaluation

- Visualize word embedding space
- Case by case: look at nearest neighbors of given words

Intrinsic Evaluation of Word Embeddings

How to evaluate the quality of word embeddings?

Qualitative Evaluation

- Visualize word embedding space
- Case by case: look at nearest neighbors of given words

Quantitative Evaluation

- Is **Word embedding similarity** related with **human judgment**?

Intrinsic Evaluation of Word Embeddings

How to evaluate the quality of word embeddings?

Qualitative Evaluation

- Visualize word embedding space
- Case by case: look at nearest neighbors of given words

Quantitative Evaluation

- Is **Word embedding similarity** related with **human judgment**?

👉 notebook

Intrinsic Evaluation of Word Embeddings

Visualization

Word Vectors are high dimensions (usually ~100)

➔ **Project the word embedding vectors** using PCA or T-SNE
➔ **Visualize** in **2D or 3D**
➔ **Analyse the clusters**

Intrinsic Evaluation of Word Embeddings

👉 notebook

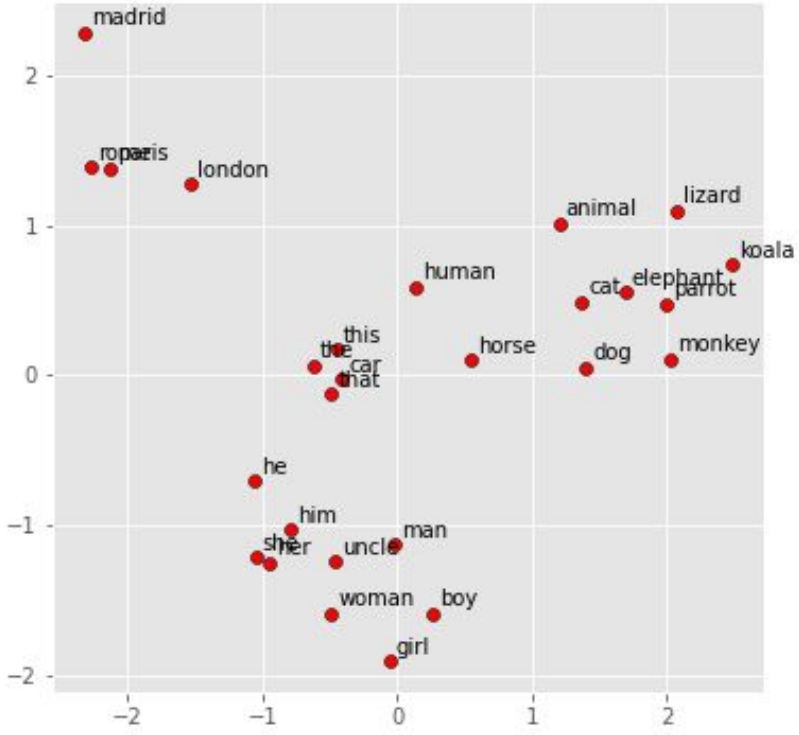


Figure: Visualize skip-gram trained on Wikipedia (1B tokens) (fastext.cc) vectors with PCA

Intrinsic Evaluation of Word Embeddings

How to measure similarity in the word embedding space?

- **Cosine Similarity**

$$sim(w_1, w_2) = cos(x_{w_1}, x_{w_2}) = \frac{x_{w_1}^T x_{w_2}}{||x_{w_1}||||x_{w_2}||}$$

- **L2 Distance**

$$sim(w_1, w_2) = L2(x_{w_1}, x_{w_2}) = ||x_{w_1} - x_{w_2}||$$

Intrinsic Evaluation of Word Embeddings

Nearest-Neighbor with the cosine similarity (skip-gram trained on Wikipedia (1B tokens))

moon	score
mars	0.615
moons	0.611
lunar	0.602
sun	0.602
venus	0.583

talking	score
discussing	0.663
telling	0.657
joking	0.632
thinking	0.627
talked	0.624

blue	score
red	0.704
yellow	0.677
purple	0.676
green	0.655
pink	0.612

Intrinsic Evaluation of Word Embeddings

We can **compare the similarity between words** in **the embedding space with human judgment**

1. **Collect Human Judgment** (or download dataset e.g. WordSim353) on a list of pairs of words
2. **Compute similarity** of the **word vectors** of those pairs
3. **Measure correlation** between both

Word 1	Word 2	Word2vec Cosine Similarity	Human Judgment
tiger	tiger	1.0	10
dollar	buck	0.3065	9.22
dollar	profit	0.3420	7.38
smart	stupid	0.4128	5.81

Application of Word Embeddings

- Downstream Tasks (Lecture 5 and 6)
- **Word Sense Induction**
- **Semantic analysis** (semantic shift in time, across communities...)

Representing Documents With Vectors

Representing Documents into Vectors

Similarly to what we saw for word-level representation we can **represent documents into vectors**

1. **Using word vectors**
2. **Count-Based Representations**
3. **Generative Probabilistic Graphical Model (e.g. LDA seen in the *lab*)**
4. **Using language models**

Representation of documents based on words

Based on word vectors representing sentence/document with vector can be done in a straightforward way with:

➔ Given sequence of word represented by x1, .., xn, define **f: → R**

$$[x_1, .., x_n] \rightarrow f(x_1, .., x_n)$$

For instance:

$$[x_1, .., x_n] \rightarrow \frac{1}{n} \sum_i x_i$$

Count-Based Representation of Documents

Given a Corpus made of novels of Shakespeare (Macbeth, Hamlet…), each document is a novel here:

1. Get the vocabulary of the Corpus
2. Compute the **Count-Based Matrix at the document-level**

Count-Based Representation of Documents

Given a Corpus made of novels of Shakespeare (Macbeth, Hamlet…), each document is a novel here:

1. Get the vocabulary of the Corpus
2. Compute the **Count-Based Matrix at the document-level**

Build the ***term-frequency*** matrix

$$tf_{t,d} = |\{t \in d\}|$$

Count-Based Representation of Documents

Given a Corpus made of novels of Shakespeare (Macbeth, Hamlet…), each document is a novel here:

1. Get the vocabulary of the Corpus
2. Compute the **Count-Based Matrix at the document-level**

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Count-Based Representation of Documents

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

➔ We get a vector representation for each document of the corpus

NB: such a model is called a ***bag-of-word model*** because the ordering of the words in each document does not matter

Count-Based Representation of Documents

Limits: High sensitivity to frequent words OR to very infrequent words

How to improve?

- **A word that is in all documents** of the corpus (e.g. “the”) is **not informative** at all for the document representation, still it impacts the document vector
- **A word that is in only 1 document** is likely to be **very informative** of the document

Solution:

- **Weight** the count with

➔ **Inverse Document Frequency**

Count-Based Representation of Documents

Weighting the importance of each term with the *document frequency*

Definition: *Given* ***N the total number of documents, a term t (token),***

$$idf_{t,C} = log(\frac{|C|}{|\{d \in C, s.t.\ t \in d\}|})$$

NB: Compute the *log* to smooth the impact of words that are in only a few documents

TF-IDF Representation of Documents

Matrix becomes: *tf*idf(t,d,C)*

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

TF-IDF Representation of Documents

We can then apply dimension reduction technique to get dense vectors

➔ E.g. we can apply SVD: **Latent Semantic Analysis**

Deep Learning Methods for NLP

Machine Learning for Natural Language Processing, ENSAE 2022

Lecture 3

Benjamin Muller, INRIA Paris

Today Lecture Outline

- Deep Learning Framework
- The Multi-Layer Perceptron
- Recurrent Neural Network
- Attention Mechanism
- Self-Attention Mechanism and the Transformer Architecture

Motivations

So far, we have seen, **techniques to represent tokens with vectors**

Given a certain representations of tokens:

➔ **How can we model a sequence of tokens to perform a specific task?**

In the past 10 years, a “new” class of machine learning techniques has become very popular and successful in NLP: **Deep Learning**

In this session, we introduce Deep Learning with a focus on the methods used in NLP

Framework

We want to model $(X_1, .., X_T)$ i.e. find the correct label *Y*

$$dnn_\theta : \quad \mathbb{R}^{d,T} \rightarrow \mathbb{R}^p \; or \; [|0, K|]^p$$

$$(X_1, .., X_T) \mapsto \hat{Y}$$

- Output space is $\mathbb{R}^p$ for **Regression** tasks
- Output space is $[|0, K|]^p$ for **Classification** tasks

Framework

We want to model $(X_1, .., X_T)$ i.e. find the correct label *Y*

$$dnn_\theta : \quad \mathbb{R}^{d,T} \rightarrow \mathbb{R}^p \; or \; [|0, K|]^p$$

$$(X_1, .., X_T) \mapsto \hat{Y}$$

Questions: when we do Deep Learning…

- **How do we define dnn_θ ?**
- **How do we train dnn_θ with data ?**

Framework

Given a sequence of vectors $(X_1, .., X_T)$ we want to predict Y

$$dnn_\theta : \quad \mathbb{R}^{d,T} \quad \rightarrow \mathbb{R}^p \; or \; [|0, K|]^p$$

$$(X_1, .., X_T) \mapsto \hat{Y}$$

Most Deep Learning Models (all the ones we will use in this course):

- are **parametric** (i.e. $\theta \in \mathbb{R}^D$)
- defined as a **composition of "simple" functions (linear & non-linear)**
- are trained in an **end-to-end** fashion with **backpropagation**

NB: In Deep Learning, **the parametrization of *dnn*** is called **the Architecture**

Different Types of Architecture

How can we define our predictive function dnn_θ ?

- ➔ Multi-Layer Perceptron
- ➔ Recurrent Layers
- ➔ Attention Layers
- ➔ Self-Attention Layers (in a Transformer Architecture)

Different Types of Architecture

How can we define our predictive function dnn_θ ?

➔ Multi-Layer Perceptron
➔ Recurrent Layers
➔ Attention Layers
➔ Self-Attention Layers (in a Transformer Architecture)

How do we **train our model**? (i.e. estimate the parameters of the model)

➔ **Stochastic Gradient Descent** also called **backpropagation** in this context

The MultiLayer Perceptron (MLP)

aka "the Most simple Deep Learning Architecture"

The **MLP** works **on unidimensional data** (e.g. dimension *d*)

We present the **MLP in the regression case** (e.g. output space is $\mathbb{R}^2$))

$$dnn_\theta : \quad \mathbb{R}^d \quad \rightarrow \mathbb{R}^2$$

$$X \mapsto \hat{Y}$$

The MultiLayer Perceptron (MLP)

aka "the Most simple Deep Learning Architecture"

The **MLP** works **on unidimensional data** (e.g. dimension *d*)
We present the **MLP in the regression case** (e.g. output space is $\mathbb{R}^2$))

$$dnn_{(W_1,b1,W_2,b_2)}(X) = W_2\varphi_1(W_1X + b_1) + b_2$$

W_1, b_1, W_2 and b_2 are trainable parameters. $W_1 \in \mathbb{R}^{\delta \times d}$, $b_1 \in \mathbb{R}^{\delta}$, $W_2 \in \mathbb{R}^{2 \times \delta}$ and $b_2 \in \mathbb{R}$

φ_1 is a fixed non-linear function, $\varphi_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{\delta}$

The MultiLayer Perceptron (MLP)

The **MLP** works **on unidimensional data** (e.g. dimension *d*)
We present the **MLP in the regression case** (e.g. output space is $\mathbb{R}^2$

$$dnn_{(W_1,b1,W_2,b_2)}(X) = W_2\varphi_1(W_1X + b_1) + b_2$$

W_1, b_1, W_2 and b_2 are trainable parameters. $W_1 \in \mathbb{R}^{\delta \times d}$, $b_1 \in \mathbb{R}^{\delta}$, $W_2 \in \mathbb{R}^{2 \times \delta}$ and $b_2 \in \mathbb{R}$

φ_1 is a fixed non-linear function, $\varphi_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{\delta}$

- ➔ This model is a ***2-layer* MLP** model
- ➔ With **1 *hidden layer*** of dimension δ
- ➔ Taking as input a vector of **dimension *d*** to output a vector of **dimension 2**
- ➔ Such a model is also referred to as a ***Feed-Forward* Neural Network (FNN)**

The MultiLayer Perceptron: Diagram View

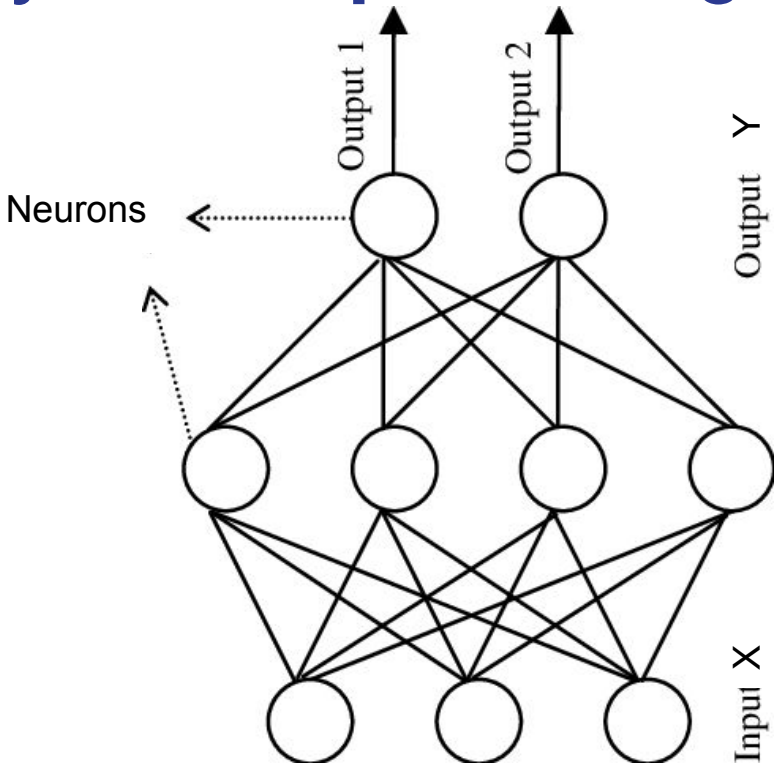
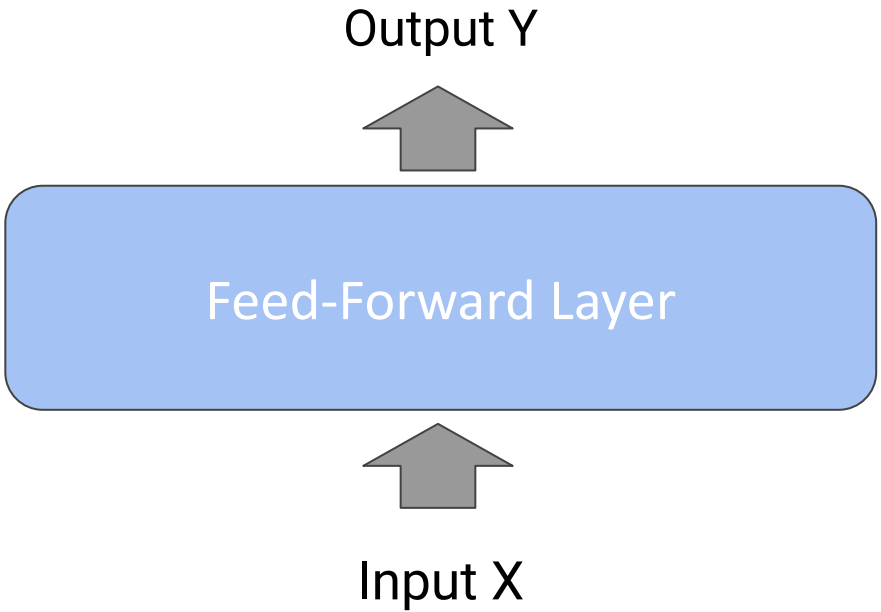


Figure from **(R. Rezvani et. al. 2012)**

In Deep Learning, it is usual to represent equations **with diagrams**

The MultiLayer Perceptron: Diagram View



In Deep Learning, it is usual to represent equations **with diagrams**

The MultiLayer Perceptron:

We have defined a 2-layers MLP model
We can define in the same way a **3-layers**, **4-layers**, **L-layers** MLP

$$dnn_{(W_i\ b_i, i\in[|1,L|])}(X) = W_L\varphi_{L-1}(...\varphi_2 \circ W_2\varphi_1(W_1X + b_1) + b_2)...) + b_L$$

W_l and b_l are trainable parameters. $W_l \in \mathbb{R}^{\delta_{l-1}\times\delta_l}$, $b_l \in \mathbb{R}^{\delta_l}$, with $\delta_l \in \mathbb{N}^*, \forall\, l \in [|1, L|]$

φ_l fixed non-linear functions, $\varphi_l : \mathbb{R}^{\delta_{l-1}} \to \mathbb{R}^{\delta_l}, \forall\, l \in [|1, L-1|]$

The MultiLayer Perceptron

The same equation with a loop…

$$h_{i+1} = \varphi_i(W_i h_i + b_i), \forall\, i \in [|1, L-1|]$$
$$\text{with } h_1 = X \text{ and } \hat{Y} = dnn(X) = h_L$$

W_l and b_l are trainable parameters. $W_l \in \mathbb{R}^{\delta_{l-1} \times \delta_l}$, $b_l \in \mathbb{R}^{\delta_l}$, with $\delta_l \in \mathbb{N}^*, \forall\, l \in [|1, L|]$

φ_l fixed non-linear functions, $\varphi_l : \mathbb{R}^{\delta_{l-1}} \to \mathbb{R}^{\delta_l}, \forall\, l \in [|1, L-1|]$

The MultiLayer Perceptron

The same equation with a loop…

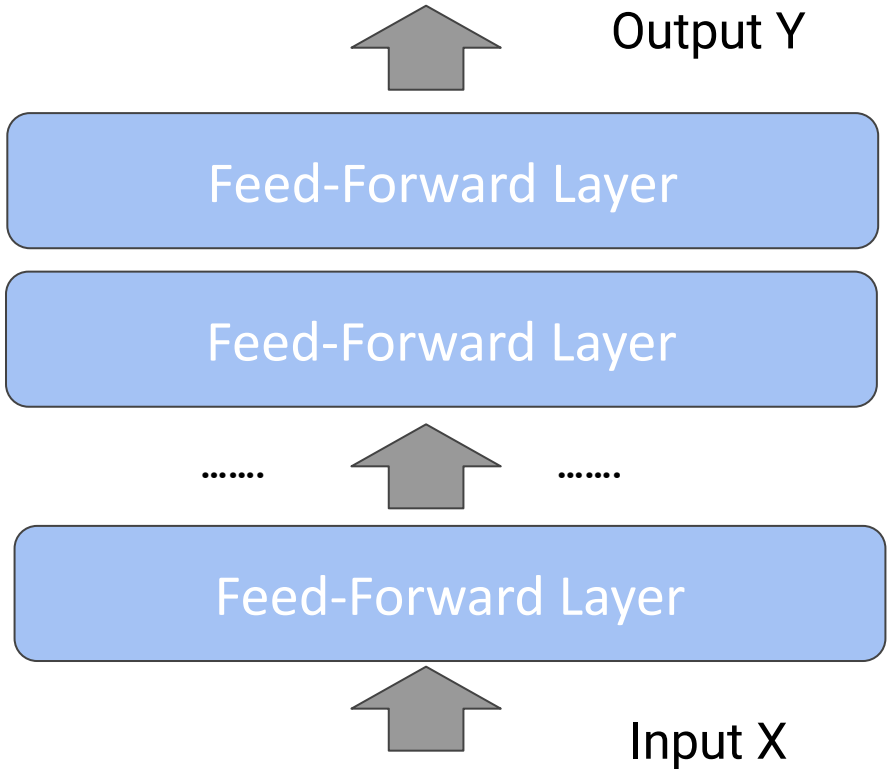
$$h_{i+1} = \varphi_i(W_i h_i + b_i), \forall\, i \in [|1, L-1|]$$
$$\text{with } h_1 = X \text{ and } \hat{Y} = dnn(X) = h_L$$

W_l and b_l are trainable parameters. $W_l \in \mathbb{R}^{\delta_{l-1} \times \delta_l}$, $b_l \in \mathbb{R}^{\delta_l}$, with $\delta_l \in \mathbb{N}^*, \forall\, l \in [|1, L|]$

φ_l fixed non-linear functions, $\varphi_l : \mathbb{R}^{\delta_{l-1}} \to \mathbb{R}^{\delta_l}, \forall\, l \in [|1, L-1|]$

h_i are called hidden states ($h_i \in \mathbb{R}^{\delta_i}$).

The MultiLayer Perceptron: Diagram View



Output Activation Function for Classification

When we do a classification task the goal is to learn a distribution of probability on the output label space

To do so, **we usually use the softmax function** as the last activation function

$$softmax(s) = \left(\frac{e^{s_i}}{\sum_k e^{s_k}}\right)_{i \in [|1,K|]}, \text{for } s \in \mathbb{R}^K$$

Loss Functions

Based on the task we aim at modeling, we can use:

For Regression: Mean-Square Error

$$l(y, \hat{y}) = \|y - \hat{y}\|_2^2 = \sum_i (y_i - \hat{y}_i)^2 \text{ assuming } y_i,\ \hat{y}_i \in \mathbb{R}$$

For Classification: Cross-Entropy Loss

$$l(y, \hat{y}) = CE(y, \hat{y}) = \sum_i y_i \, log(\hat{y}_i) \text{ assuming } y_i,\ \hat{y}_i \in [0, 1]$$

Most NLP tasks will be based on the **Cross-Entropy loss**

The MultiLayer Perceptron: Hyperparameters

- **Number of hidden layers**
- **Hidden layers dimensions**
- Initialization of the trainable parameters/weights

The MultiLayer Perceptron: Hyperparameters

- Number of hidden layers
- Hidden layers dimensions
- **Initialization** of the **trainable parameters/weights**

The MultiLayer Perceptron: Hyperparameters

- Number of hidden layers
- Hidden layers dimensions
- Initialization
- **Activation Functions**

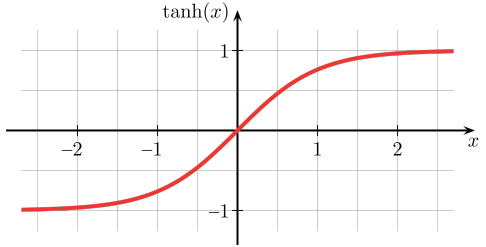
The MultiLayer Perceptron: Hyperparameters

- Number of hidden layers
- Hidden layers dimensions
- Initialization
- **Activation Functions**

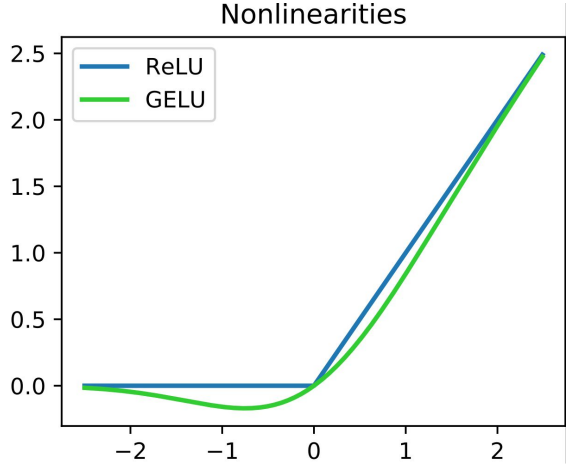
➢ They should be **non-linear**
➢ **Differentiable**
➢ **Standard ones are:**
Relu, tanh, sigmoid

The MultiLayer Perceptron: Hyperparameters

- Number of hidden layers
- Hidden layers dimensions
- Initialization
- **Activation Functions**



➢ They should be **non-linear**
➢ **Differentiable**
➢ **Standard ones are:**
Relu, tanh, sigmoid



The MultiLayer Perceptron: Hyperparameters

- Number of hidden layers
- Hidden layers dimensions
- Initialization
- Activation Functions

How to define them?

➔ Look for **best practices** to choose which are the best
➔ In most DL libraries, the **“good” hyperparameters are usually the default**
➔ If no best practices/default: **you have to find the best ones empirically**

Training Deep Learning Models

- Nearly all Deep Learning models are trained with (some version of) **Stochastic Gradient Descent (SGD)**

Stochastic Gradient Descent

- The goal is find the set of **parameters/weights** that **minimizes the loss function**
- To do so, SGD estimates the true gradient of a function with **one sample at time**
- **Repeat** this process multiple times

NB: in deep learning, we usually train all the parameters together **"end-to-end"**

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descend

Given observations $((x_i), (y_i))$ of two variables (X, Y)
Given a loss function l. An architecture dnn_θ
The goal is to find the best θ **s.t.** $E(l(Y, dnn_\theta(X))$ **is small**. Given a learning rate α
for *step* $<$ max **do**
    Sample (x, y)
    # Forward pass:
    $\hat{y} = dnn_\theta(x)$ and $l(y, \hat{y})$
    # Backward pass:
    $\nabla_\theta \, l(y, \hat{y})$ # compute loss
    $\theta := \theta - \alpha \nabla_\theta \, l(y, \hat{y})$ # parameter update
end

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descend

Given observations $((x_i), (y_i))$ of two variables (X, Y)
Given a loss function l. An architecture dnn_θ
The goal is to find the best θ s.t. $E(l(Y, dnn_\theta(X))$ is small. Given a learning rate α
for *step* $<$ *max* **do**
    Sample (x, y)
    # Forward pass:
    $\hat{y} = dnn_\theta(x)$ and $l(y, \hat{y})$
    # Backward pass:
    $\nabla_\theta \, l(y, \hat{y})$ # compute loss
    $\theta := \theta - \alpha \nabla_\theta \, l(y, \hat{y})$ # parameter update
end

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descend

Given observations $((x_i), (y_i))$ of two variables (X, Y)
Given a loss function l. An architecture dnn_θ
The goal is to find the best θ **s.t.** $E(l(Y, dnn_\theta(X))$ **is small**. Given a learning rate α
for $step < max$ **do**
    Sample (x, y)
    # Forward pass:
    $\hat{y} = dnn_\theta(x)$ and $l(y, \hat{y})$
    # Backward pass:
    $\nabla_\theta \, l(y, \hat{y})$ # compute loss
    $\theta := \theta - \alpha \nabla_\theta \, l(y, \hat{y})$ # parameter update
end

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descend

Given observations $((x_i), (y_i))$ of two variables (X, Y)
Given a loss function l. An architecture dnn_θ
The goal is to find the best θ **s.t.** $E(l(Y, dnn_\theta(X))$ **is small**. Given a learning rate α
for *step* $<$ max **do**
    Sample (x, y)
    # Forward pass:
    $\hat{y} = dnn_\theta(x)$ and $l(y, \hat{y})$
    # Backward pass:
    $\nabla_\theta \, l(y, \hat{y})$ # compute loss
    $\theta := \theta - \alpha \nabla_\theta \, l(y, \hat{y})$ # parameter update
end

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descend

Given observations $((x_i), (y_i))$ of two variables (X, Y)
Given a loss function l. An architecture dnn_θ
The goal is to find the best θ **s.t.** $E(l(Y, dnn_\theta(X))$ **is small**. Given a learning rate α
for *step* $<$ max **do**
    Sample (x, y)
    # Forward pass:
    $\hat{y} = dnn_\theta(x)$ and $l(y, \hat{y})$
    # Backward pass:
    $\nabla_\theta \, l(y, \hat{y})$ # compute gradients
    $\theta := \theta - \alpha \nabla_\theta \, l(y, \hat{y})$ # parameter update
end

Stochastic Gradient Descent

Optimization Hyperparameters

Learning Rate

- Can be refined with **variable learning rate**
  *E.g. increasing during the first steps (**warmup**) then decreasing*

Number of steps

- Usually defined with based on the validation loss
  *When it stops decreasing we can stop training (=**early stopping**)*

Stochastic Gradient Descent

Optimizing large Deep Learning Models **is challenging**

- **Unstable training**
- **Overfitting**
- **Take a lot of steps/epochs**

To make training better, many refinement of the SGD have been proposed

- In practice, we (nowadays) **use the ADAM optimizer** (cf. Kingma et. al 2015)

Stochastic Gradient Descent for MLP

Let $(X, Y) \in \mathbb{R}^d \times \mathbb{R}$, the MSE loss $l(y, \hat{y}) = (y - \hat{y})^2$.

We define a 1-hidden-layer MLP with a RELU activation function of dimension δ.

$$\hat{y} = dnn_{W_1,W_2}(x) = W_2 \max(W_1 x, 0) \text{ and } W_1 \in \mathbb{R}^{d \times \delta} \text{ and } W_2 \in \mathbb{R}^{1 \times \delta}$$

1. **Forward pass**
2. **Compute Gradients**
3. **Backward pass (parameter update)**

Stochastic Gradient Descent for MLP

Let $(X, Y) \in \mathbb{R}^d \times \mathbb{R}$, the MSE loss $l(y, \hat{y}) = (y - \hat{y})^2$.

$$\hat{y} = dnn_{W_1,W_2}(x) = W_2 \underbrace{\max(W_1 x, 0)}_{h_1} \text{ and } W_1 \in \mathbb{R}^{d \times \delta} \text{ and } W_2 \in \mathbb{R}^{1 \times \delta}$$

Compute Gradient:

$$\nabla_{W_2} l(y, \hat{y}) = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} = 2(y - \hat{y})\, h_1$$

$$\nabla_{W_1} l(y, \hat{y}) = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial W_2} = 2(y - \hat{y})\, W_2 1_{W_1 X > 0}$$

Backpropagation and Deep Learning in practice

In practice, we use Deep Learning Libraries

- Define **the Architecture with *tensor* operators**
- Backpropagation is done **seamlessly using automatic differentiation**

Deep Learning & Backpropagation in practice

In practice, we use Deep Learning Libraries (e.g. pytorch, tensorflow, jax)

- Define **the Architecture with *tensor* operators**
- Backpropagation is done **seamlessly using automatic differentiation**

- Standard layers **are pre-implemented** (Feed-Forward Layers, LSTM, Attention, Self-Attention…)

See code example with pytorch

Recurrent Neural Network

Vanilla Recurrent Neural Network

We would like to model sequences (e.g. words) $(X_1, .., X_T)$ in $\mathbb{R}^{d,T}$

We can introduce **a recurrence relation** into our MLP to model it:

$$h_{i+1,t+1} = \varphi_i(W_i h_{i,t} + U_i h_{i+1,t} + +b_i), \forall\, i \in [|1, L-1|]$$

$$\text{with } h_{1,t} = X_t \text{ and } \hat{Y}_t = dnn(X_t) = h_{L,t} \,\forall\, t \in [|1, T-1|]$$

Recurrent Neural Network

Illustration of a 1-layer Recurrent Neural Network

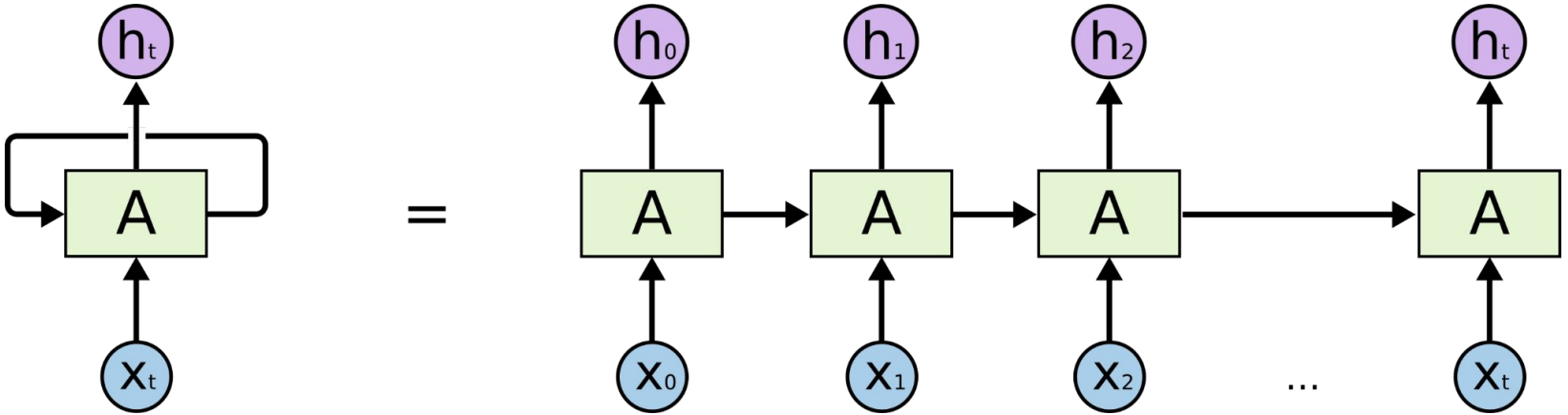


Figure from colah

Recurrent Neural Network

Illustration of a 1-layer Recurrent Neural Network

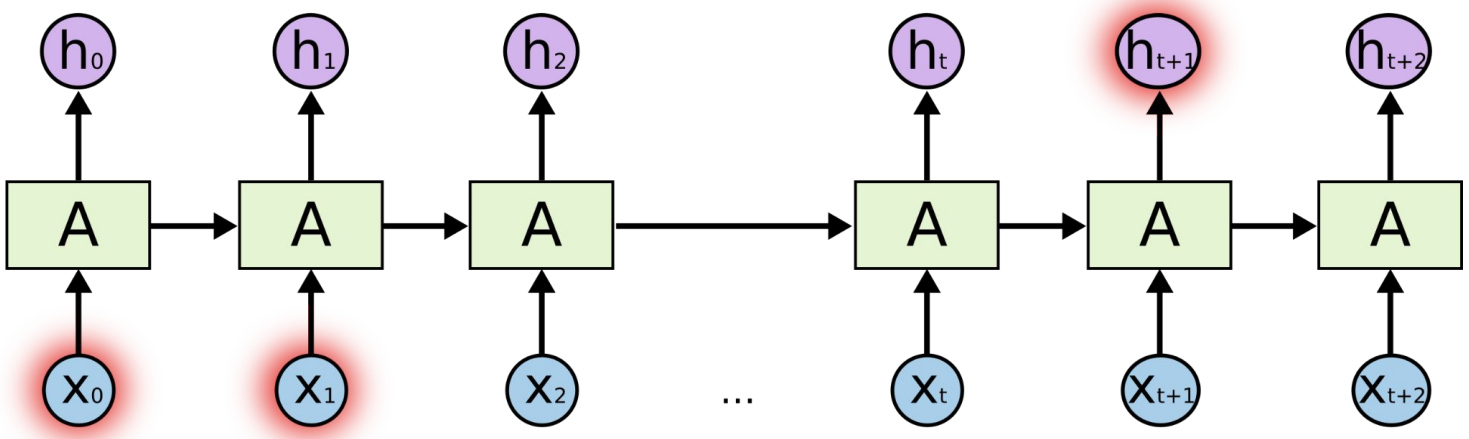


Figure from colah

Training Recurrent Neural Network

Recurrent Neural Network are trained with an extension of the Backpropagation algorithm

➔ Backpropagation Through Time (BPTT)

BPTT follows exactly the same ideas as backpropagation

- SGD
- Chain Rule starting from the last layer and the last hidden state
- **With extra derivative dependencies between state *t and t+1***

Limits of Recurrent Neural Networks

Vanilla Recurrent Neural Network have trouble to capture long-term dependencies

Idea:

- Encode **explicitly in a vector a "memory" in the recurrent architecture**
- Control what is memorized and forgotten
- Train all those parameters **end-to-end**

LSTM: Long-Short Term Memory Network

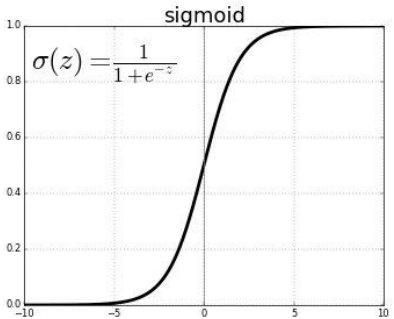
Introduce a memory vector C_t

C_t is designed to **capture long term dependencies**

The output state h_t ***of each LSTM cell is based on*** C_t ***and an output gate*** o_t

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



LSTM: Long-Short Term Memory Network

Introduce a memory vector C_t

C_t is designed to **capture long term dependencies**

C_t is define recurrently based on the previous step and the input and the forget gate. Those gates control what is memorized and forgotten.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM: Long-Short Term Memory Network

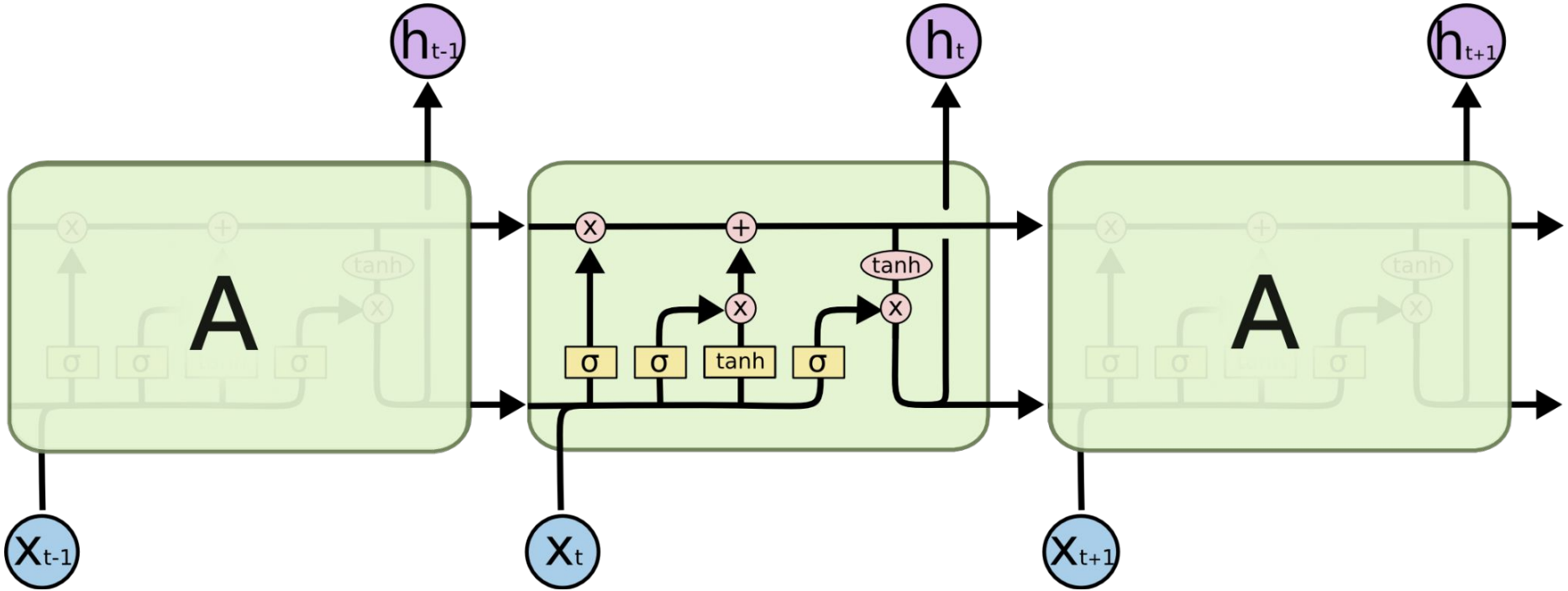


Figure from colah

LSTM: Long-Short Term Memory Network

- We train LSTM with Backpropagation (through time)
- LSTM cells are usually combined with Feed-Forward Layers

NB: Until recently (2018), LSTM-based models were delivering **State-of-the-art performance for most sequence modelling tasks**

Attention Mechanism

Motivation for Attention Mechanisms

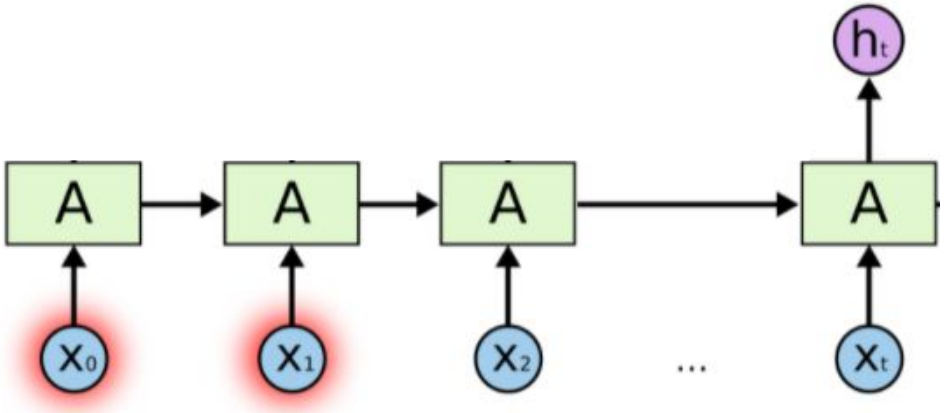
- The Deep Learning Architecture that we have seen so far are **hard to interpret** **(black-box)**
- Recurrent Network provide a fixed vector encoding of a sequence at each step

➔ **Attention Mechanisms**

Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment analysis)

Solution 1: Use a LSTM model → Problem (not interpretable)

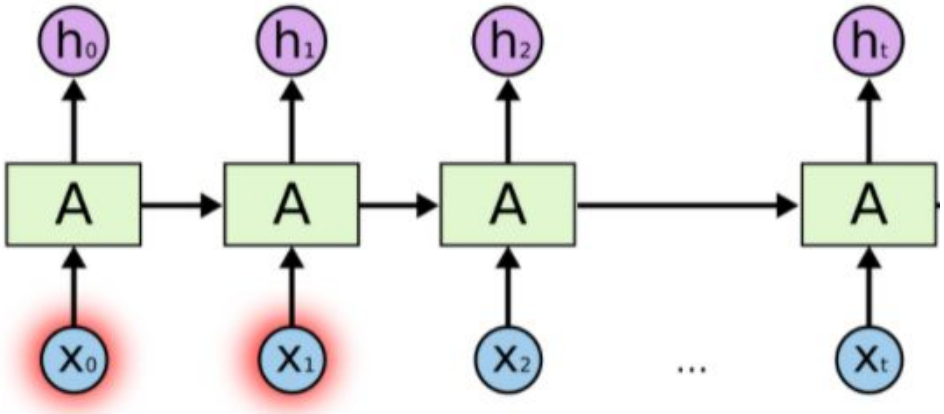


Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment)

Solution 2: Integrate an Attention Mechanism to interpret what input impacts the prediction

➔ **Learn a ponderation/weighting of the hidden states ht**



Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment)

How to learn this weighting?

1. Define a specific type of layer to learn the ponderation
2. Train this layer end-to-end with all the other parameters of the model

Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment)

How to learn this weighting?

Given $(h_1, .., h_T)$ hidden representations of $(x_1, .., x_T)$ (e.g. output of a LSTM Layer).

$$q_i = tanh(W_a h_i + b_a), \text{ with } W_a \in \mathbb{R}^{\delta \times \delta_a}$$

$$s_t = \frac{e^{q_t q_T}}{\sum_j e^{q_j q_T}}, \text{ i.e. } \sum_{t \in [|1,T|]} s_i = 1$$

$$\tilde{h_T} = \sum_{t \in [|1,T|]} s_t . h_t$$

Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment)

How to learn this weighting?

Given $(h_1, .., h_T)$ hidden representations of $(x_1, .., x_T)$ (e.g. output of a LSTM Layer).

$$q_i = tanh(W_a h_i + b_a), \text{ with } W_a \in \mathbb{R}^{\delta \times \delta_a}$$

$$s_t = \frac{e^{q_t q_T}}{\sum_j e^{q_j q_T}}, \text{ i.e. } \sum_{t \in [|1,T|]} s_i = 1$$

$$\tilde{h_T} = \sum_{t \in [|1,T|]} s_t . h_t$$

Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment)

How to learn this weighting?

Given $(h_1, .., h_T)$ hidden representations of $(x_1, .., x_T)$ (e.g. output of a LSTM Layer).

$$q_i = tanh(W_a h_i + b_a), \text{ with } W_a \in \mathbb{R}^{\delta \times \delta_a}$$

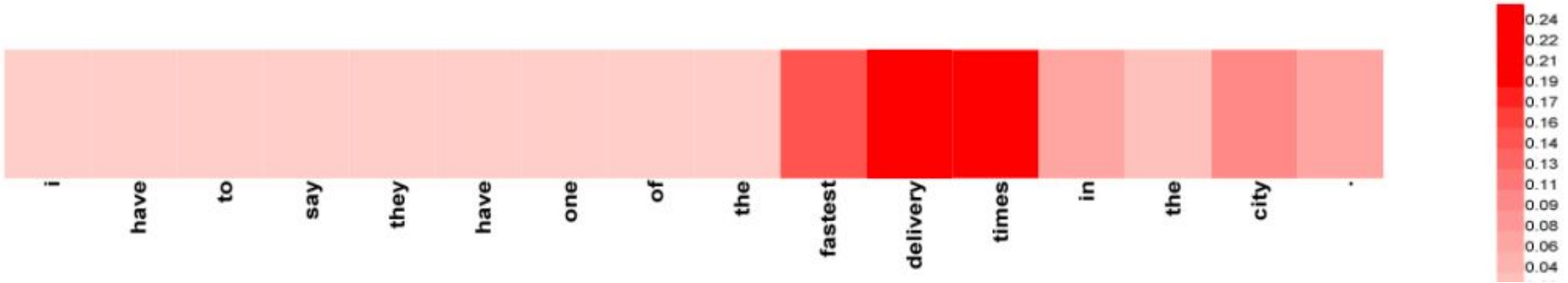
$$s_t = \frac{e^{q_t q_T}}{\sum_j e^{q_j q_T}}, \text{ i.e. } \sum_{t \in [|1,T|]} s_i = 1$$

$$\tilde{h_T} = \sum_{t \in [|1,T|]} s_t . h_t$$

Attention Mechanism for Sequence Classification

We want to classify (X0, Xt) sequences (e.g. sentiment classification)

After we trained the model, **Attention scores** can be used **to interpret the model** behavior and **what input vector impacted the decision**



(Wang et. al 2016)

Attention Mechanism for Sequence Classification

Many variant of Attention Mechanisms (in combination with LSTM layers) have been designed

Design Choices

- How to define the ***query vectors*****?**
- How to define the ***scoring function*****?**

Many variants exists but the principles are the same.

The Transformer Architecture

Attention might be all we need

Do we really need recurrent layers?

RNN models (such as vanilla RNN, LSTM…) were designed to model sequential data

Still, for most tasks, we **need both left and right context (e.g. sequence classification, sequence labelling..)**

Why not modelling sequences in a bi-directional way directly

➔ **Using Self-Attention Mechanism**

Self-Attention Layers

Given a sequence of input vectors $(x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $(h_{0,1}, .., h_{0,T})$).

Objective:

- Build a representation of the input vectors based on the **surrounding vectors** (both right-and left-context)

Idea:

- **No need of recurrent cells**

➔ **Self-Attention**

Self-Attention Layers: Intuition

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$)

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

- For a given vector h_t and its query vector q_t we want to build the new representation vector $\tilde{h}_t$
- Using the best ponderation of the information encoded in $(v_1, .., v_T)$
- This ponderation being computed by finding the key vectors in $(k_1, .., k_T)$ that are more similar to the query vector q_t (that encodes relevant information from h_t).

Self-Attention Layers: Intuition

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

- For a given vector h_t and its query vector q_t we want to build the new representation vector $\tilde{h}_t$
- Using the best ponderation of the information encoded in $(v_1, .., v_T)$
- This ponderation being computed by finding the key vectors in $(k_1, .., k_T)$ that are more similar to the query vector q_t (that encodes relevant information from h_t).

Self-Attention Layers: Intuition

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

- For a given vector h_t and its query vector q_t we want to build the new representation vector $\tilde{h}_t$
- Using the best ponderation of the information encoded in $(v_1, .., v_T)$
- This ponderation being computed by finding the key vectors in $(k_1, .., k_T)$ that are more similar to the query vector q_t (that encodes relevant information from h_t).

Self-Attention Layers: Intuition

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

- For a given vector h_t and its query vector q_t we want to build the new representation vector $\tilde{h}_t$
- Using the best ponderation of the information encoded in $(v_1, .., v_T)$
- This ponderation being computed by finding the key vectors in $(k_1, .., k_T)$ that are more similar to the query vector q_t (that encodes relevant information from h_t).

Self-Attention Layers

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

$$q_t = W_Q h_t \ , \ \forall t \in [|1, T|] \text{ with } W_Q \in \mathrm{R}^{\delta_q \times \delta}$$

$$k_t = W_K h_t \ , \ \forall t \in [|1, T|] \text{ with } W_K \in \mathrm{R}^{\delta_k \times \delta}$$

$$v_t = W_V h_t \ , \ \forall t \in [|1, T|] \text{ with } W_V \in \mathrm{R}^{\delta_v \times \delta}$$

Self-Attention Layers

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

$$q_t = W_Q h_t \ , \ \forall t \in [|1, T|] \text{ with } W_Q \in \mathrm{R}^{\delta_q \times \delta}$$

$$k_t = W_K h_t \ , \ \forall t \in [|1, T|] \text{ with } W_K \in \mathrm{R}^{\delta_k \times \delta}$$

$$v_t = W_V h_t \ , \ \forall t \in [|1, T|] \text{ with } W_V \in \mathrm{R}^{\delta_v \times \delta}$$

Self-Attention Layers

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

$$q_t = W_Q h_t \ , \ \forall t \in [|1, T|] \text{ with } W_Q \in \mathrm{R}^{\delta_q \times \delta}$$

$$k_t = W_K h_t \ , \ \forall t \in [|1, T|] \text{ with } W_K \in \mathrm{R}^{\delta_k \times \delta}$$

$$v_t = W_V h_t \ , \ \forall t \in [|1, T|] \text{ with } W_V \in \mathrm{R}^{\delta_v \times \delta}$$

Self-Attention Layers

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

$$\tilde{H} = softmax(\frac{Q\,K^T}{\sqrt{\delta_K}})V$$

i.e. $\tilde{h_t} = softmax(\frac{q_t\,K^T}{\sqrt{\delta_K}}).V = \sum_{t'} s_{t'}\,v_{t'}$ with $s_{t'} = \frac{e^{q_{t'}k_t}}{\sum_t e^{q_{t'}k_t}}$

Self-Attention Layers

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

$$\tilde{H} = softmax(\frac{Q\,K^T}{\sqrt{\delta_K}})V$$

i.e. $\tilde{h_t} = softmax(\frac{q_t\,K^T}{\sqrt{\delta_K}}).V = \sum_{t'} s_{t'}\,v_{t'}$ with $s_{t'} = \frac{e^{q_{t'}k_t}}{\sum_t e^{q_{t'}k_t}}$

Self-Attention Layers

Given a sequence of input vectors $X = (x_1, .., x_T) \in \mathbb{R}^\delta$ (noted $H = (h_{0,1}, .., h_{0,T})$).

We build 3 new vectorial representation of our sequence $H = (h_1, .., h_T)$).

The *query* $Q = (q_1, .., q_T)$, the *key* $K = (k_1, .., k_T)$ and the *value* $V = (v_1, .., v_T)$ vectors.

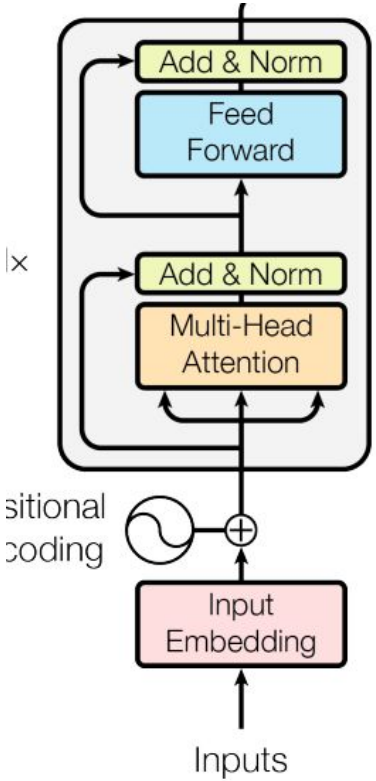
$$\tilde{H} = softmax(\frac{Q\,K^T}{\sqrt{\delta_K}})V$$

i.e. $\tilde{h_t} = softmax(\frac{q_t\,K^T}{\sqrt{\delta_K}}).V = \sum_{t'} s_{t'}\,v_{t'}$ with $s_{t'} = \frac{e^{q_{t'}k_t}}{\sum_t e^{q_{t'}k_t}}$

The Transformer Architecture

The Transformer Architecture is

- Stack of [Self-Attention + FF Layer]
- With Skip-Layer and Normalization Layers in between
- Encoding the position with positional vector



Positional Embedding Vector

- **Limitation:** self attention does not take position into account!
- Indeed, shuffling the input gives the same results

- **Solution:** add position encodings.
- Replace the matrix $\mathbf{W}$ by $\mathbf{W} + \mathbf{E}$, where $\mathbf{E} \in \mathbb{R}^{d \times T}$

- $\mathbf{E}$ can be learned, or defined using sin and cos:

$$e_{2i,j} = \sin\left(\frac{j}{10000^{2i/d}}\right)$$
$$e_{2i+1,j} = \cos\left(\frac{j}{10000^{2i/d}}\right)$$

Scaling Laws Intuition

- The larger the dimension of the weight matrices
- The larger the number of parameters in the model
- The more “expressive” is the model
- The better it will generalizes

Typical Architecture Sizes