# Natural Language Processing (NLP)

Pierre Zweigenbaum

LISN, CNRS, Université Paris-Saclay

2026-10-21
AI$^2$ Fall School, Le Touquet

# Outline

1. NLP: Core Concepts
2. Modeling Approaches
3. From Modeling to NLP Tasks
4. NLP System Evaluation

# Natural Language vs. Formal Language

## Natural Language

- Used for human communication.
- **Characteristics**:
  - Ambiguity (e.g., "bank" = financial or river).
  - Open vocabulary (infinite).
  - One idea $\rightarrow$ multiple expressions (synonymy, paraphrase).

## Formal Language

- Structured (e.g., programming, math).
- **Characteristics**:
  - Unambiguous syntax.
  - Closed vocabulary.
  - Single interpretation.

## Reference

Comparison based on Manning and Schütze [1999].

# Forms of Natural Language

- **Spoken language**: Speech, dialogues.
- **Written language**: Texts, documents.
- **Sign language**: Visual communication (e.g., ASL, LSF).

# Functions of Language

## What is Natural Language Used For?

- **Communication**: Facilitating human interaction [Winograd, 1972].
- **Issuing Commands**: Giving instructions (e.g., "Turn off the light").
- **Expressing Sentiments**: Conveying emotions (e.g., "I love this!").

## A Material with Its Own Rules

- The language system, with its own rules (grammar, syntax, semantics).
- Multiple natural languages exist (e.g., English, French, Mandarin).

## Storing Information and Knowledge

- Newspapers, books, scientific papers.
- Dictionaries.

# Characteristics of Natural Language

## One Meaning → Multiple Expressions

- Synonymy: "Happy" = "Joyful" = "Content".
- Paraphrase: "It's raining" = "Rain is falling" [Barzilay and Elhadad, 2001].

## One Utterance → Multiple Interpretations

- Ambiguity: "I saw the bat." (animal or sports equipment) [Yarowsky, 1995].
- Context-dependent meaning: "Bank" in "blood bank" vs. "financial bank".

## Infinite and Open Vocabulary

- Open vocabulary: New words emerge constantly (e.g., "selfie", "COVID-19").
- Infinite set of possible utterances.

# Natural Language Processing: Definition

## Processes

NLP involves processes that:

- Take **natural language** as input.
- Apply computational techniques to **analyze, understand, or generate** language.
- Produce **natural language** or structured data as output.

## From and to Natural Language

- Input: Natural language (text, speech).
- Output: Natural language or structured representations.

# An Interdisciplinary Domain

## A Domain of Artificial Intelligence

NLP combines knowledge from multiple disciplines:

- **Linguistics**
- **Cognitive Science**
- **Mathematics**
- **Computer Science**
  - Machine Learning

# Sampling Language Data: Introduction

## Corpus (pl. corpora)

A corpus is a **representative sample** of language material.

## Language Material

- **Document**: Structured text (e.g., articles, books).
- **Dialogue**: Conversations (e.g., chat logs, interviews).
- **Word**: Individual words.
- **...**: Other forms (e.g., social media posts, emails).

# Selecting Language Material → Corpus

## Selection Criteria

- Define selection criteria (e.g., domain, time period, language).
- Ensure representativeness (avoid bias).
- Consider size and diversity.

## Obtaining a Representative Sample

- Domain: General vs. specialized (e.g., medical, legal).
- Time Period: Historical vs. contemporary.
- Language: Monolingual vs. multilingual.
- Medium: Written vs. spoken vs. signed.

# Using Corpora

## Applications

Corpora can be used for:

- **Manual corpus analysis**: Linguistic research, sociolinguistics.
- **Machine-assisted corpus exploration**: Tools like AntConc, Sketch Engine.
- **Training data-driven methods**: Machine learning models (e.g., neural networks).
- **Evaluating NLP systems and methods**: Benchmarking models on standardized datasets.

# Annotated Corpora: Definition

## Language Material Samples + Labels

- Obtained by human annotation (usually).
- Used for supervised learning tasks.

# Text-Level Labels

## Text Classification

- Assign one or more labels to entire texts.
- **Examples**:
  - **SPAM detection**: Label emails as spam or not spam.
  - **Sentiment analysis**: Label reviews as positive, negative, or neutral.

# Segment-Level Labels

## Span Detection and Classification

- Identify and classify spans of text (e.g., named entities).
- **Example**: Named Entity Recognition (NER).
  *"Pierre Zweig works at CNRS in Paris."* →
  ***Pierre Zweig*** *(PERSON)*, ***CNRS*** *(ORG)*, ***Paris*** *(LOC)*.
- **Example annotation**: NER for de-identification (e.g., anonymizing patient names in medical records).

# Are Annotations Reliable?

## Inter-Annotator Agreement

- Measures the consistency between different annotators.
- **Metrics**:
  - **Cohen's Kappa**: Adjusts for chance agreement. **Fleiss' Kappa**: For multiple annotators.
  - **RIA (Raw Inter-Annotator Agreement)**: Simple percentage agreement.
- **Formula for Cohen's Kappa**:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

where $p_o$ = observed agreement, $p_e$ = expected agreement by chance.

## Example

If two annotators agree on 80% of items and chance agreement is 50%, then: $\kappa = \frac{0.8-0.5}{1-0.5} = 0.6$
A $\kappa$ of 0.6 indicates **substantial agreement**.

# Corpora in 2024 [Liu et al., 2024]

| Corpus | Publisher | Release Time | Size | Public or Not | License |
|---|---|---|---|---|---|
| ANC | The US National Science Foundation et al. | 2003-X | - | All | - |
| Anna's Archive | Anna | 2023-X | 641.2 TB | All | - |
| ArabicText 2022 | BAAI et al. | 2022-12 | 201.9 GB | All | CC-BY-SA-4.0 |
| arXiv | Paul Ginsparg et al. | 1991-X | - | All | Terms of Use for arXiv APIs |
| Baidu baike | Baidu | 2008-4 | - | All | Baidu baike User Agreement |
| BIGQUERY | Salesforce Research | 2022-3 | 341.1 GB | Not | Apache-2.0 |
| BNC | Oxford University Press et al. | 1994-X | 4124 Texts | All | - |
| BookCorpusOpen | Jack Bandy et al. | 2021-5 | 17868 Books | All | Smashwords Terms of Service |
| CC-Stories | Google Brain | 2018-7 | 31 GB | Not | - |
| CC100 | Facebook AI | 2020-7 | 2.5 TB | All | Common Crawl Terms of Use |
| CLUECorpus2020 | CLUE Organization | 2020-3 | 100 GB | All | MIT |
| Common Crawl | Common Crawl | 2007-X | - | All | Common Crawl Terms of Use |
| CulturaX | University of Oregon et al. | 2023-9 | 27 TB | All | mC4 & OSCAR |
| C4 | Google Research | 2019-10 | 12.68 TB | All | ODC-BY & Common Crawl Terms of Use |
| Dolma | AI2 et al. | 2024-1 | 11519 GB | All | MR Agreement |
| Github | Microsoft | 2008-4 | - | All | - |
| mC4 | Google Research | 2021-6 | 251 GB | All | ODC-BY & Common Crawl Terms of Use |
| MNBVC | Liwu Community | 2023-1 | 20811 GB | All | MIT |
| MTP | BAAI | 2023-9 | 1.3 TB | All | BAAI Data Usage Protocol |
| MultiUN | German Research Center for Artificial Intelligence (DFKI) GmbH | 2010-5 | 4353 MB | All | - |
| News-crawl | UKRI et al. | 2019-1 | 110 GB | All | CC0 |
| OpenWebText | Brown University | 2019-4 | 38 GB | All | CC0 |
| OSCAR 22.01 | Inria | 2022-1 | 8.41 TB | All | CC0 |
| ParaCrawl | Prompsit et al. | 2020-7 | 59996 Files | All | CC0 |
| PG-19 | DeepMind | 2019-11 | 11.74 GB | All | Apache-2.0 |
| phi-1 | Microsoft Research | 2023-6 | 7 B Tokens | Not | CC-BY-NC-SA-3.0 |
| Project Gutenberg | Ibibio et al. | 1971-X | - | All | The Project Gutenberg |
| Pushshift Reddit | Pushshift.io et al. | 2020-1 | 2 TB | All | - |
| RealNews | University of Washington et al. | 2019-5 | 120 GB | All | Apache-2.0 |
| Reddit | Condé Nast Digital et al. | 2005-6 | - | All | - |
| RedPajama-V1 | Together Computer | 2023-4 | 1.2 T Tokens | All | - |
| RedPajama-V2 | Together Computer | 2023-10 | 30.4 T Tokens | All | Common Crawl Terms of Use |
| RefinedWeb | The Falcon LLM team | 2023-6 | 5000 GB | Partial | ODC-BY-1.0 |
| ROOTS | Hugging Face et al. | 2023-3 | 1.61 TB | Partial | BLOOM Open-RAIL-M |
| Smashwords | Draft2Digital et al. | 2008-X | - | All | Smashwords Terms of Service |
| StackExchange | Stack Exchange | 2008-9 | - | All | CC-BY-SA-4.0 |
| S2ORC | AI2 et al. | 2020-6 | 81.1 MB | All | ODC-BY-1.0 |
| The Pile | EleutherAI | 2021-1 | 825.18 GB | All | MIT |
| The Stack | ServiceNow Research et al. | 2022-11 | 6 TB | All | The Terms of the Original Licenses |
| TigerBot_pretrain_en | TigerBot | 2023-5 | 51 GB | Partial | Apache-2.0 |
| TigerBot_pretrain_zh | TigerBot | 2023-5 | 55 GB | Partial | Apache-2.0 |
| TigerBot-wiki | TigerBot | 2023-5 | 205 MB | All | Apache-2.0 |
| Toronto Book Corpus | University of Toronto et al. | 2015-6 | 11038 Books | Not | MIT & Smashwords Terms of Service |

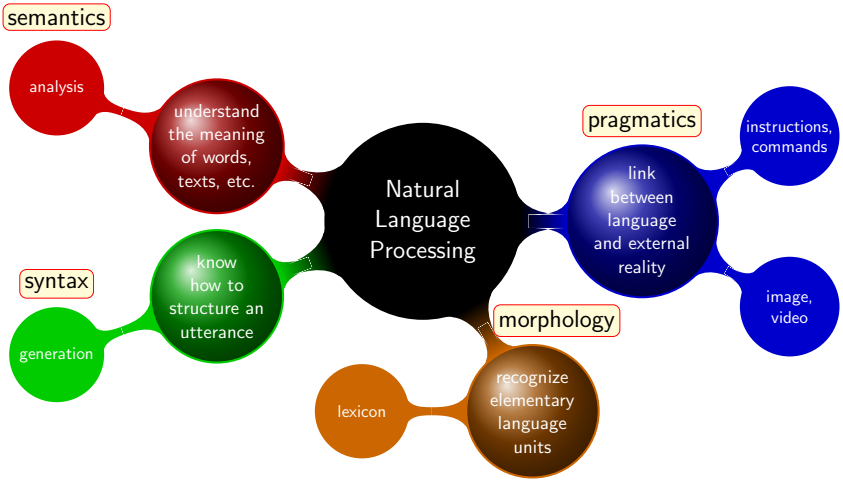
# Natural Language Processing: Processes

## From and to Natural Language

NLP processes:

- Take **natural language** as input.
- Apply computational techniques to:
  - **Analyze** (e.g., parsing, NER) [Grishman, 1997].
  - **Understand** (e.g., semantic analysis).
  - **Generate** (e.g., machine translation, summarization) [Koehn, 2009].
- Produce **natural language** or structured data as output.

# Language Processing Levels



# Uses of NLP: Language as a Means of Communication

- **User** → **Machine**: Command a machine (e.g., voice assistants).
- **Machine** → **User**: Produce text from structured data (e.g., report generation).
- **User** → **Machine** → **User**: Dialogue with a machine (e.g., chatbots).

# Uses of NLP: A Material with Its Own Rules

## Authoring Aids

- **Spelling correction**: Fix spelling errors.
- **Grammatical correction**: Fix grammar mistakes.
- **Reformulation**: Rewrite text in different ways.

## Other Applications

- **Summarization**: Condense long texts into shorter summaries.
- **Machine translation**: Translate text between languages [Koehn et al., 2003, Sutskever et al., 2014].

# Uses of NLP: Support to Record Information and Knowledge

- **Information retrieval and extraction**: Find and extract relevant information from texts.
- **Converting text into data**: Structure text for databases or knowledge graphs.
- **Answering questions**: Provide answers to user queries (e.g., QA systems).

# Modeling Language to Process It

## Natural Language → Representation

- Convert natural language into a machine-processable format.
- **Methods**:
  - Character-based: Character encodings, string matching.
  - Word-based: Tokenization, n-grams, word embeddings.
  - Sentence-based: Syntactic parsing, semantic analysis.
  - Document-based: Topic modeling, information retrieval.

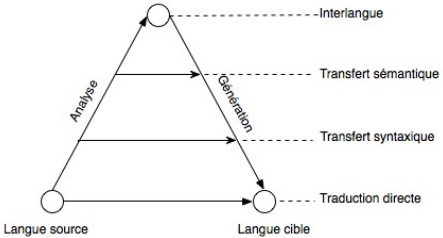
## Representation → Output

- **Output Types**:
  - Natural language (e.g., machine translation).
  - Structured data (e.g., knowledge graphs, databases).
  - Decisions (e.g., classification, sentiment analysis).

# Example modeling levels: Machine Translation

## The Vauquois Triangle

The Vauquois triangle is a classic model for machine translation approaches [Vauquois, 1968].




(Vauquois, 1968)

## Approaches

1. **Direct Translation**: Word-by-word (e.g., early MT systems) [Weaver, 1955].
2. **Transfer Approach**: Source → representation → Target.
3. **Interlingua Approach**: Abstract representation independent of source/target languages.

# Character Strings

## Basic Units

- **Character**: Smallest unit (e.g., 'a', 'b', '1').
- **Writing systems**: Latin, Cyrillic, Chinese, etc.
- **Character encodings**: ASCII, Unicode (UTF-8).

# Character String Matching: Code Example

## Character String Matching

- **Edit Distance**: Measures similarity between two strings [Levenshtein, 1966].

  Levenshtein($s_1, s_2$) = Minimum number of edits (insert, delete, substitute) to transform $s_1$ into $s_2$.

- **Example**: Levenshtein("kitten", "sitting") = 3.
- **Character-based Regular Expressions**: Pattern matching (e.g., '\d{3}-\d{3}-\d{4}' for US phone numbers).

## Code Example: Edit Distance in Python

```
1  from Levenshtein import distance
2
3  s1 = "kitten"
4  s2 = "sitting"
5  print(f"Edit distance: {distance(s1, s2)}")  # Output: 3
```

# Words: Basic Linguistic Unit

## Word: Basic Linguistic Unit

- **Definition**: A word is a basic unit of language.
- **What is a word?**
  - **Orthographic**: Separated by spaces (e.g., "the cat").
  - **Morphological**: Smallest free form (e.g., "unhappiness" = "un" + "happy" + "ness").
  - **Semantic**: Carries meaning (e.g., "bank" has multiple meanings).

# Word Segmentation (Tokenization)

## Word Segmentation (Tokenization)

- **Goal**: Split a text into words (tokens).
- **Example**:
  *Input: "The cat sat on the mat."*
  *Output: ["The", "cat", "sat", "on", "the", "mat", "."]*
- **Challenges**:
  - Contractions (e.g., "don't" → ["do", "n't"] or ["don't"]).
  - Hyphenated words (e.g., "state-of-the-art").
  - Multi-word expressions (e.g., "New York").

## Code Example: Tokenization in Python (NLTK)

```
1  from nltk.tokenize import word_tokenize
2
3  text = "The␣cat␣sat␣on␣the␣mat."
4  tokens = word_tokenize(text)
```

# Word Distribution

## Words in a Document/Corpus

- **Counts, Frequencies**: How often words appear.
- **Example**:

| Word | Frequency |
|---|---|
| the | 100 |
| cat | 50 |
| sat | 30 |

# Word × Document Matrix

## Word × Document Matrix

- **Definition**: A matrix where rows = words, columns = documents, and cells = word frequencies.
- **Example**:

$$\begin{bmatrix} & \text{Doc1} & \text{Doc2} \\ \text{the} & 5 & 3 \\ \text{cat} & 2 & 0 \\ \text{sat} & 1 & 4 \end{bmatrix}$$

- **Probabilistic Representations**:
  - $p(w_i|d_j)$: Probability of word $w_i$ in document $d_j$.
  - $p(d_j|w_i)$: Probability of document $d_j$ given word $w_i$.

## Association Measures

- **TF-IDF**: Term Frequency - Inverse Document Frequency.
- **BM25**: Best Match 25 (improved ranking function) [Jones, 1972].

# Word Sequences: *n*-grams

**Definition**

An ***n*-gram** is a contiguous sequence of $n$ items (words, characters) from a text.

- **Unigram** ($n = 1$): ["the", "cat", "sat"]
- **Bigram** ($n = 2$): ["the cat", "cat sat"]
- **Trigram** ($n = 3$): ["the cat sat"]
- **Skipgrams**: Non-contiguous sequences (e.g., ["the", "sat"] with skip=1).

# $n$-gram Statistics

## Statistical Language Model

- Predicts the probability of a sequence of words.
- **Formula**:

$$p(w_n|w_1, w_2, \ldots, w_{n-1})$$

- **Example**:

$$p(\text{sat}|\text{the cat}) = \frac{\text{Count(the cat sat)}}{\text{Count(the cat)}}$$

# Word Sequences: Word Windows

## Word Window Centered on a Pivot Word $w_i$

- Equal sizes left and right.
- Context words $w_j$ where $j \in \{i-k \ldots, i-1, i+1, \ldots, i+k\}$.

# Word × Word Matrix

## Word Associations

- **Matrix** $M$ where:

$$M_{ij} = \text{Count}(w_i, w_j) \text{ or } p(w_j|w_i)$$

- **Word Associations**:

$$\text{PMI}(w_i, w_j) = \log_2 \frac{p(w_i, w_j)}{p(w_i) \cdot p(w_j)}$$

where PMI = Pointwise Mutual Information [Church and Hanks, 1990].

# Word Features Induced from a Text Corpus

What words occur in what document (and how many times)

## Word × document matrix

$$\mathbf{t}_i^T \rightarrow \begin{bmatrix} x_{\mathbf{1},\mathbf{1}} & \dots & x_{\mathbf{1},j} & \dots & x_{\mathbf{1},n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,\mathbf{1}} & \dots & x_{i,j} & \dots & x_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,\mathbf{1}} & \dots & x_{m,j} & \dots & x_{m,n} \end{bmatrix} \quad \mathbf{d}_j \downarrow$$

- For instance, $x_{i,j} = tf_{i,j}.idf_i$

## Representing a word as a *vector* $\mathbf{t}_i$ *of associations to documents*

- Word similarity computation
- E.g., corpus = Wikipedia, *Explicit semantic analysis* [Gabrilovich and Markovitch, 2007]

# Word Morphology

## Word Decomposition

- **Base/Radix/Root**: Core meaning (e.g., "happy" in "unhappiness").
- **Inflection**: Endings (e.g., "-s" in "sings").
- **Affixes**: Prefixes and suffixes (e.g., "un-", "-ness" in "unhappiness").
- **Compound Elements**: Multiple bases (e.g., "book" + "shelf" = "bookshelf").

## Word Segmentation

- **Segmenting a Text into Words**: Tokenization.
- **Segmenting a Word into Morphologically-Relevant Elements**:
  *"unhappiness" → ["un", "happy", "ness"]*
- **Morphological Analysis**: Identify morphological structure.

# Word Morphology: Lemmatization

## Word Form → Dictionary Form

- **Definition**: Reduce a word to its base/dictionary form (lemma).
- **Example**:

| Word Form | Lemma |
|---|---|
| running | run |
| better | good |
| cats | cat |

- **Process**: Remove inflection.

## Code Example: Lemmatization in Python (spaCy)

```
1  import spacy
2
3  nlp = spacy.load("en_core_web_sm")
4  doc = nlp("The cats are running better than the dogs")
```

# Word Morphology: Stemming

## Word → Base/Radix/Root Word

- **Definition**: Reduce a word to its root form (stem) by removing affixes.
- **Example**:

| Word | Stem |
|---|---|
| running | run |
| happiness | happi |
| cats | cat |

- **Note**: Stems may not be valid words (e.g., "happi").

## Code Example: Stemming in Python (NLTK)

```
1  from nltk.stem import PorterStemmer
2
3  stemmer = PorterStemmer()
```

# Word Morphology: Compound Analysis

## Word → Base Words

- **Definition**: Analyze compound words to identify their constituent elements.
- **Example**:
  *"bookshelf"* → *["book", "shelf"]*
  *"state-of-the-art"* → *["state", "of", "the", "art"]*
- **Process**: Identify compound boundaries and decompose into base words.

# Statistical Subword Segmentation

## Token: Non-Linguistic Definition

- **Definition**: A token is a minimal unit (word or subword) for processing.
- **Example**:
  - Word: "unhappiness" (if vocabulary includes it).
  - Subword: ["un", "happi", "ness"] (if vocabulary is limited).

## Text → Tokens

- **Given**:
  - A corpus.
  - A maximum number of tokens (vocabulary size).
- **Goal**: Select a token vocabulary optimal for segmenting the corpus.
- **Methods**:
  - **Byte Pair Encoding (BPE)**: Merge frequent pairs of bytes/characters.
  - **WordPiece**: Similar to BPE but optimized for neural models.
  - **Unigram**: Select tokens based on likelihood.

# Statistical Subword Segmentation: Code Example

## Code Example: BPE in Python (Hugging Face Tokenizers)

```python
from tokenizers import Tokenizer
from tokenizers.models import BPE
from tokenizers.trainers import BpeTrainer
from tokenizers.pre_tokenizers import Whitespace

# Train a BPE tokenizer on a corpus
tokenizer = Tokenizer(BPE())
tokenizer.pre_tokenizer = Whitespace()
trainer = BpeTrainer(vocab_size=10000)
tokenizer.train(["corpus.txt"], trainer)

# Tokenize a sentence
output = tokenizer.encode("The␣cat␣sat␣on␣the␣mat")
print(output.tokens)
```

# Syntactic Models

## Syntactic Structure

- **Definition**: Structured word sequences based on grammatical rules.
- **Formal Grammar**:
  - **Regular Grammar**: For word-based syntax (e.g., finite-state automata).
  - **Context-Free Grammar (CFG)**: For hierarchical structures (e.g., parse trees) [Chomsky, 1957].

## Syntactic Parsing

- **Goal**: Analyze the syntactic structure of a sentence.
- **Example**:
  *Input: "The cat sat on the mat."*
  *Output: Parse tree showing syntactic relationships.*

# Semantic Models

## Meaning Representation

- **Semantic Primitives**: Basic units of meaning (e.g., "act", "object").
- **Semantic Relations**: Relationships between concepts (e.g., hypernymy, synonymy).
- **Formal Semantics**: Mathematical representations of meaning (e.g., lambda calculus).
- **Conceptual Graphs**: Graph-based representations of meaning.
- **Description Logics**: Formal representations for ontologies.

## Semantic Analysis

**Goal**: Extract meaning from text.

**Example**:

*"The cat sat on the mat."* $\rightarrow$

***Act***: *sit*

***Agent***: *cat*

***Location***: *mat*

# Pragmatic Models

## Discourse Representation

- **Definition**: Representation of meaning in context (beyond sentence level).
- **Example**:
  *"She gave him the book. He read it."* $\rightarrow$
  ***Reference Resolution***: *"He" = the person who received the book.*

## Discourse Analysis

- **Goal**: Analyze the structure and meaning of discourse (e.g., conversations, narratives).
- **Applications**:
  - Dialogue systems.
  - Text summarization.
  - Coreference resolution.

# Interlingua

## Language-Independent Representation

- **Definition**: A representation of meaning that is independent of any specific language.
- **Components**:
  - **Pivot Language**: A language used as an intermediate representation.
  - **Conceptual Primitives**: Basic concepts that can be combined to represent any meaning.

## Example: Interlingua for Machine Translation

- Source Language $\rightarrow$ Interlingua $\rightarrow$ Target Language.
- Avoids direct translation between all language pairs.

# Predicting Words within Word Window: Word2Vec

## Word2Vec: Overview

- **Architecture**: Shallow neural network.
- **Training Objective**:
  - **Skip-gram**: Predict context words from a pivot word.
  - **CBOW (Continuous Bag of Words)**: Predict pivot word from context words.
- **Output**: Word embeddings (dense vectors).

## Key Properties

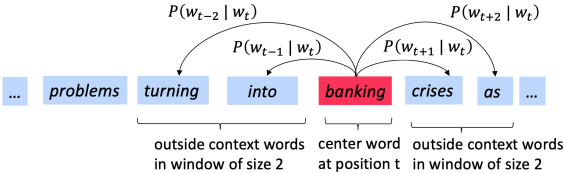
- Captures semantic relationships (e.g., pear $\approx$ peach $\approx$ banana $\neq$ plane).
- Efficient for large vocabularies.

[Mikolov et al., 2013b].

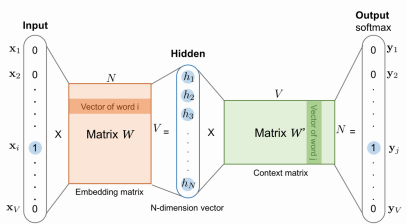
# Simple Fully-Connected Layers: word2vec [Mikolov et al., 2013a]

Train to predict context words (schemas from https://zhangruochi.com/Word-Vectors/2019/12/04/)

- skip-gram
- Predict neighboring words
- Fixed-sized window (e.g., $\pm 2$)




- Input = one-hot vector for focus word
- Output = prediction of neighboring words
- Learn two matrices
- Left matrix contains embedding vector of each focus word



# Word2Vec: Code Example

## Code Example: Word2Vec in Python (Gensim)

```python
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize

# Sample corpus
corpus = [
    word_tokenize("the cat sat on the mat"),
    word_tokenize("the dog sat on the mat"),
]

# Train Word2Vec model
model = Word2Vec(corpus, vector_size=100, window=2, min_count=1, workers=4)

# Get word embeddings
cat_embedding = model.wv["cat"]
print(f"Embedding for 'cat': {cat_embedding[:5]}...")  # First 5 dimensions

# Find similar words
similar = model.wv.most_similar("cat")
print(f"Words similar to 'cat': {similar}")
```

# Predicting Words within Word Window: BERT

## BERT: Overview

- **Architecture**: Deep neural network (Transformer).
- **Training Objective**:
  - **Masked Language Modeling (MLM)**: Predict masked words from context.
  - **Next Sentence Prediction (NSP)**: Predict if two sentences are consecutive.
- **Output**: Contextual word embeddings.

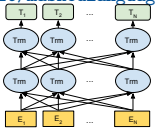
## Key Properties

- Bidirectional context (unlike Word2Vec).
- Better contextual understanding (e.g., "bank" in "blood bank" vs. "financial bank").
- Pre-trained on large corpora (e.g., Wikipedia, BookCorpus).

[Devlin et al., 2019b].

# Transformer encoder: BERT [Devlin et al., 2019a] Masked Language Model

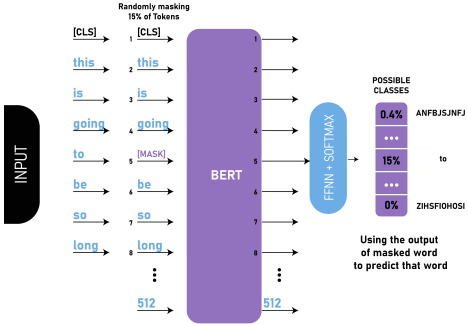
Schema from https://media.geeksforgeeks.org/wp-content/uploads/20200422002516/maskedLanguage.jpg




- Bidirectional attention layers
- Learn 12 layers of 12 attention heads
- Compute contextual word representations

- Input = sentence with masked word
- Output = prediction of masked word
- Keep whole set of parameters (110M)
- Use the full architecture to compute word representations dynamically in each new sentence

→ Pre-trained language model



# BERT: Code Example

## Code Example: BERT in Python (Hugging Face)

```python
from transformers import BertTokenizer, BertModel
import torch

# Load pre-trained model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Tokenize input
inputs = tokenizer("The cat [MASK] on the mat", return_tensors="pt")

# Predict masked word
with torch.no_grad():
    outputs = model(**inputs)

# Get predictions for masked token
mask_token_index = torch.where(inputs["input_ids"][0] == tokenizer.mask_token_id)[0]
predicted_token_id = torch.argmax(outputs.logits[0, mask_token_index, :]).item()
predicted_token = tokenizer.decode([predicted_token_id])

print(f"Predicted word: {predicted_token}")  # Output: "sat"
```

# Predicting Words at the End of a Word Window: GPT

## GPT: Overview

- **Architecture**: Deep neural network (Transformer).
- **Training Objective**:
  - Predict the next word in a sequence from previous words.
- **Output**: Generated text (autoregressive).

## Key Properties

- Autoregressive: Generates text word-by-word.
- Capable of producing coherent and varied text.
- Pre-trained on large corpora (e.g., WebText, Common Crawl).

[Sutskever et al., 2014]

# GPT: Code Example

## Code Example: GPT-2 in Python (Hugging Face)

```python
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import torch

# Load pre-trained model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# Encode input
input_ids = tokenizer.encode("The cat sat on the", return_tensors="pt")

# Generate text
outputs = model.generate(input_ids, max_length=10, num_beams=5, no_repeat_ngram_size=2)

# Decode output
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(f"Generated text: {generated_text}")
```

**Example Output**:

```
The cat sat on the mat and started to purr.
```

# Example: Text Classification

## From a Text to Label(s)

- **Input**: A text (e.g., "This movie was great!").
- **Output**: One or more labels (e.g., "positive").
- **Applications**:
  - Sentiment analysis.
  - Spam detection.
  - Topic classification.

## Example Dataset: IMDB Reviews

| Text | Label |
|---|---|
| "This movie was fantastic!" | Positive |
| "I hated every minute of it." | Negative |

# Example: Named Entity Recognition (NER)

## Spot Entity Mentions in a Text

- **Input**: A text (e.g., "Pierre Zweig works at CNRS in Paris.").
- **Output**: Labeled entities (e.g., "Pierre Zweig" $\rightarrow$ PERSON).
- **Entity Types**:
  - PERSON, ORGANIZATION, LOCATION, DATE, etc.

## Example Annotation

*"[Pierre Zweig]$_{PERSON}$ works at [CNRS]$_{ORG}$ in [Paris]$_{LOC}$."*

# Knowledge-based Methods: Text Classification

## From a Text to Label(s)

- **Approach**: Use handcrafted rules or knowledge bases.
- **Example**:
  - **Keyword Matching**: Label text based on predefined keywords.
  - **Ontology-Based**: Use hierarchical knowledge (e.g., WordNet).

## Example Rule

*IF text contains ("great" OR "fantastic" OR "excellent") THEN label = "positive"*

# Knowledge-based Methods: Named Entity Recognition

## Spot Entity Mentions in a Text

- **Approach**: Use predefined lists or patterns.
- **Example**:
  - **Gazetteers**: Lists of known entities (e.g., countries, organizations).
  - **Regular Expressions**: Patterns for entities (e.g., dates, phone numbers).

## Example Gazetteer

| Entity Type | Examples |
|---|---|
| PERSON | ["Pierre", "Marie", "Jean"] |
| ORG | ["CNRS", "INRIA", "Google"] |
| LOC | ["Paris", "Lyon", "Berlin"] |

# Supervised Machine Learning

## Training Data

- **Input**: Word, text, etc. ($X_i$).
- **Output**: Expected label(s) ($y_i$).
- **Example Dataset**:

| **Text ($X_i$)** | **Label ($y_i$)** |
|---|---|
| "This movie was great!" | Positive |
| "I hated it." | Negative |

## Test Data

- **Input**: Word, text, etc. ($X_i$). **Output**: Hidden expected label(s) ($y_i$).
- **Goal**: Evaluate model performance on unseen data.

# Training and Evaluation

## Learn (Train) Model on Training Examples

- **Objective**: Estimate $\hat{f}(X) = y$ on $(X_i, y_i)^{\text{train}}$.
- **Methods**:
  - Linear models (e.g., Logistic Regression).
  - Tree-based models (e.g., Random Forest).
  - Neural networks (e.g., CNN, RNN, Transformer).

## Apply Model to Test Examples (Inference)

- **Process**:
  1. Compute $\hat{y}_i$ for $X_i \in$ test.
  2. Compare to gold standard labels $y_i^{\text{test}}$.
- **Metrics**:
  - Accuracy, Precision, Recall, F1-Score.

# Feature Engineering

## How to Create a Feature Vector $X$ from Input Words or Texts?

- Use any of the modeling approaches:
  - Bag-of-Words (BoW).
  - TF-IDF.
  - Word embeddings (Word2Vec, GloVe).
  - Contextual embeddings (BERT, GPT).

# Feature Engineering: TF-IDF Example

## Text → Feature Vector

Convert text into numerical features for machine learning models.

## Code Example: TF-IDF Vectorization in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
"The cat sat on the mat.",
"The dog sat on the mat."
]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

print(f"Vocabulary: {vectorizer.get_feature_names_out()}")
print(f"TF-IDF Matrix:\n{X.toarray()}")
```

**Output**:

```
Vocabulary: ['cat', 'dog', 'mat', 'on', 'sat', 'the']
TF-IDF Matrix:
[[0.4654  0.      0.3489  0.3489  0.3489  0.4654]
 [0.      0.4654  0.3489  0.3489  0.3489  0.4654]]
```

# Feature Engineering: BERT Embeddings

## Text → Feature Vector

- Use pre-trained BERT embeddings as features.

## Code Example: BERT Embeddings in Python

```python
from transformers import BertTokenizer, BertModel
import torch

# Load pre-trained model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Tokenize input
inputs = tokenizer("The cat sat on the mat", return_tensors="pt")

# Get embeddings
with torch.no_grad():
    outputs = model(**inputs)

# Extract embeddings (last hidden states)
embeddings = outputs.last_hidden_state
print(f"Embeddings shape: {embeddings.shape}")  # (1, sequence_length, 768)
```

# Example: Text Classification with TF-IDF and Logistic Regression

## From a Text to Label(s)

- **Input**: A text (e.g., "This movie was great!").
- **Output**: One or more labels (e.g., "positive").

# Example: Text Classification with TF-IDF and Logistic Regression

## Code Example: Text Classification with Scikit-Learn

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Sample data
texts = [
    "This movie was great!",
    "I hated every minute of it.",
    "The film was okay.",
    "It was the best movie ever!"
]
labels = ["positive", "negative", "neutral", "positive"]

# Vectorize texts
vectorizer = TfidfVectorizer()
/X = vectorizer.fit_transform(texts)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25)

# Train model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
```

# BERT Encoder for Text Classification

(schemas from El Boukkouri, 2021)

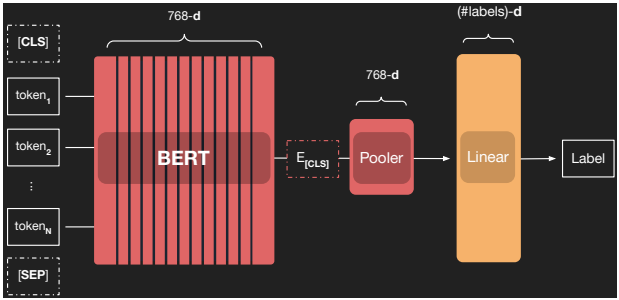
**Encode the input text**

- BERT

**Apply a supervised classifier**

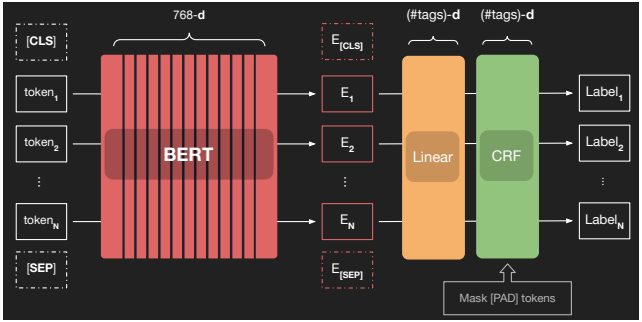
- linear layer

→ label for the text



# BERT Encoder for Entity Detection: BERT + linear layer + CRF

Schema from El Boukkouri, PhD



# Example: Named Entity Recognition with Fine-Tuning

## Spot Entity Mentions in a Text

- **Input**: A text (e.g., "Pierre Zweig works at CNRS in Paris.").
- **Output**: Labeled entities (e.g., "Pierre Zweig" → PERSON).

## Code Example: Fine-Tuning BERT for NER

```python
from transformers import BertTokenizer, BertForTokenClassification
from transformers import Trainer, TrainingArguments
from datasets import load_dataset

# Load dataset (e.g., CoNLL-2003)
dataset = load_dataset("conll2003")

# Load tokenizer and model
tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
model = BertForTokenClassification.from_pretrained("bert-base-cased", num_labels=9)

# Tokenize and align labels
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples["tokens"], truncation=True, is_split_into_words=True)
    labels = []
    for i, label in enumerate(examples["pos_tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
```

# Pretraining vs. Supervised Fine-Tuning

## Pretraining

- **Learning about language**:
  - Grammar, vocabulary, syntax.
- **Learning about the domain**:
  - General knowledge from large corpora.

## Supervised Fine-Tuning

- **Adapting to a specific task**:
  - Classification, NER, QA, etc.
- **Using labeled data**:
  - Task-specific datasets.

## Approaches

- **Original (Frozen) Encoder Weights**:
  - Use pre-trained encoder weights (e.g., BERT).
  - Add and train a classifier head.
- **Trained (Fine-Tuned) Encoder Weights**:
  - Add and train a classifier head.
  - While training the classifier, allow small adaptation of encoder weights.

# Text-to-Text Framework

## Prompt

- **Input** = Text description of the task:
  - **Instruction**: Task description (e.g., "Classify the following text").
  - **Context**: Additional information (e.g., "The text is about movies").
  - **Examples (Demonstrations)**: Few-shot examples (e.g., "Text: 'Great movie!' Label: positive").
- **Output** = Text description of the solution:
  - **Label**: Predicted class.
  - **Justifications**: Explanation (e.g., "The word 'great' indicates positivity").

## Example: Text Classification as Text-to-Text

***Input***:
*"Classify the following text as positive or negative.*
*Text: 'This movie was fantastic!'*
*Example: Text: 'Great film!' Label: positive"*
***Output***:
*"Label: positive*
*Justification: The word 'fantastic' indicates a positive sentiment."*

# Datasets: Text Classification

## From a Text to Label(s)

- **Example Datasets**:
  - **IMDB**: Movie reviews (sentiment analysis).
  - **20 Newsgroups**: News articles (topic classification).
  - **SpamAssassin**: Emails (spam detection).

# Datasets: Named Entity Recognition

## Spot Entity Mentions in a Text

- **Example Datasets**:
  - **CoNLL-2003**: News articles (NER).
  - **Ontonotes**: Various domains (NER, coreference resolution).

# Metrics: Text Classification

## Is the Label Distribution Balanced?

- **Imbalanced Data**: One or more classes are underrepresented.
- **Example**: 90% positive, 10% negative reviews.

## Common Metrics

| Metric | Formula | Use Case |
|---|---|---|
| Accuracy | $\frac{TP+TN}{TP+TN+FP+FN}$ | Overall performance |
| Precision | $\frac{TP}{TP+FP}$ | Quality of positive predictions |
| Recall | $\frac{TP}{TP+FN}$ | Coverage of true positives |
| F1-Score | $2 \cdot \frac{P \cdot R}{P+R}$ | Balance between precision and recall |

# Metrics: Named Entity Recognition

## Entity Detection, Not Token Classification

- NER evaluates the detection of **complete entities**, not individual tokens.
- **Example**:
  *"New York" should be detected as one entity, not two separate tokens.*

## Common Metrics

- **Entity Recall/Precision/F1**: Evaluates detection of complete entities.
- **Example**:
  - True Positive (TP): Correctly detected entity.
  - False Positive (FP): Incorrectly detected entity.
  - False Negative (FN): Missed entity.

# References I

Regina Barzilay and M. Elhadad. Extracting paraphrases from a corpus. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001.

Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

Kenneth W. Church and Patrick Hanks. Word association norms, mutual information, and lexicography. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 22–29. Association for Computational Linguistics, 1990.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019a. Association for Computational Linguistics. URL `https://aclanthology.org/N19-1423`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT 2019, pages 4171–4186. Association for Computational Linguistics, 2019b.

Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1625275.1625535`.

# References II

Ralph Grishman. Information extraction: Techniques and challenges. In *Proceedings of the IEEE*, volume 85, pages 1704–1716. IEEE, 1997.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2009.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 127–133. Association for Computational Linguistics, 2003.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10 (8):707–710, 1966.

Yang Liu, Jiahuan Cao, Chongyu Liu, Kai Ding, and Lianwen Jin. Datasets for large language models: A comprehensive survey. *CoRR*, abs/2402.18041, 2024. doi: 10.48550/ARXIV.2402.18041. URL `https://doi.org/10.48550/arXiv.2402.18041`.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR (workshop poster)*, 2013a.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations*, 2013b.

# References III

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Neural machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1017–1026. Association for Computational Linguistics, 2014.

Bernard Vauquois. Automatic translation: A survey of the state of the art. *Annual Review of Information Science and Technology*, 3:1–46, 1968.

Warren Weaver. Translation. *Machine Translation*, 2:15–23, 1955.

Terry Winograd. *Understanding Natural Language*. Academic Press, 1972.

David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995.

# Thank You!

**Pierre Zweigenbaum**
CNRS