L1: Intro to NLP

Documents & Knowledge

Document: any source of NLP text (books, articles, web). Text is **unstructured** and **ambiguous**. **Knowledge Representation**: structured, precise, actionable, domain-specific data a computer can use. **NLP**: converts unstructured documents → structured knowledge.

Definition

NLP: subfield of **linguistics**, **CS**, and **AI** for analyzing/understanding human language and processing large amounts of natural language data.

AI / ML / DL / NLP

- **AI**: umbrella – making computers act like humans
- **ML**: subset of AI – learns from data (classification, clustering, forecasting, association, anomaly detection)
- **DL**: subset of ML – deep neural networks
- **NLP**: intersects ML and DL

Text must be converted to numbers before ML/DL.

History

1950s: translation machines. 1960s: ELIZA chatbot. 1970s: statistical models. 2013: Word2Vec. 2017: "Attention Is All You Need" – Transformer. Post-2017: LLMs (GPT, etc.). **Transformer variants**: Encoder-only (BERT), Decoder-only (GPT), Encoder-Decoder (T5).

Turing Test

Judge asks questions to a human & computer. If judge can't distinguish → **pass**. Proposed by Alan Turing, 1950.

NLU vs NLG

- **NLU only**: sentiment analysis, text classification, spam detection
- **NLG only**: speech-to-text, auto report generation
- **NLU+NLG**: translation, summarization, QA, chatbots

Challenges in NLP

1. **Ambiguity**: lexical ("bank"), attachment ("saw man with telescope"), coreference ("she")
2. **Sparsity / Zipf's Law**: $f(w) \propto 1/r$; rare words can be important but hard to learn from
3. **Variation**: lexical, geographical, social, stylistic, generational, cross-linguistic
4. **Common knowledge**: humans share implicit knowledge computers lack
5. Volume, accents/slang, computation, security/privacy

Approaches to NLP

Heuristic/Rule-based (regex) → Machine Learning → Deep Learning (RNN, LSTM) → Transformers (2017+). Old techniques still useful for edge cases.

Tools

NLTK, spaCy, Hugging Face, scikit-learn.

L2: Regex, Pipeline, Preprocessing

Regular Expressions

`.`	any single char
`?`	0 or 1 of preceding
`+`	1 or more of preceding
`*`	0 or more of preceding
`[]`	any one char inside
`[^]`	negation (not in set)
`^`	start of string
`$`	end of string
`{}`	specific repetitions
`\`	escape metachar
`|`	alternation (or)
`()`	grouping (exact sequence)

Char classes: `\s` whitespace, `\w` alpha (`[A-Za-z]`), `\d` digit. Uppercase = negation.

Python re Library

`re.match` – beginning only. `re.search` – first match anywhere. `re.findall` – list of all matches. `re.sub` – substitute. `re.compile` – reusable pattern. `re.split` – split at pattern. `re.finditer` – iterator of match objects. Match objects: `.group()`, `.span()`.

NLP Dev Lifecycle

1. Understand problem & requirements 2. Data collection (large + relevant) 3. Text cleaning 4. Preprocessing 5. Feature extraction 6. Modeling 7. Evaluation 8. Deployment 9. Monitoring. Non-linear: loop back from eval/monitoring. **Garbage in, garbage out.**

Preprocessing Pipeline

Raw doc → **Tokenization** → **Noise Removal** → **Normalization** → clean tokens.

Building Blocks of Language

Phonemes (44 sounds) → Morphemes/Lexemes (smallest meaningful unit) → Syntax (grammar rules) → Context. **Corpus**: collection of docs. **Vocabulary**: set of unique words.

Tokenization

Divide text into **tokens** (not just words). Punctuation & contractions become separate tokens ("can't" → "can" + "'t"). NLTK: `word_tokenize()`, `sent_tokenize()`.

Noise Removal

Remove: numbers, punctuation, **stop words** (179 in NLTK), URLs, HTML tags, handles. Keep emojis for sentiment. Lowercase conversion.

Normalization

Stemming: rule-based suffix stripping. Fast, may produce invalid words ("studies"→"studi"). Porter, Snowball (multilingual), Lancaster (aggressive).

Lemmatization: dictionary lookup, always valid words ("studies"→"study", "better"→"good"). Slower. NLTK WordNetLemmatizer, spaCy.

Skip normalization for: poetry analysis, morphological analysis.

POS Tagging & NER

POS: identifies noun, verb, adj, etc. Tags: NN, VB, JJ, RB, DT, IN. `pos_tag(tokens)`. **NER**: identifies people, places, orgs. `ne_chunk(pos_tag(...))`.

L3: Feature Representation & Similarity

Feature Engineering

Convert text → numerical table for ML. Equally important as preprocessing.

Vector Space Model

Each word/doc = **vector** of numbers. Similar words → nearby vectors. **Norm**: $\|\vec{v}\| = \sqrt{v_1^2 + \cdots + v_n^2}$ **Dot product**: $\vec{a} \cdot \vec{b} = \sum a_i b_i$

One Hot Encoding

Binary vector: 1 if token exists, 0 otherwise. Dim = vocab size. Pros: reversible, interpretable. Cons: high dimensionality, sparse.

Bag of Words (BoW)

Vector of **word frequencies**. Ignores word order. "child makes dog happy" = "dog makes child happy" (same BoW!). `CountVectorizer()` in sklearn.

Bag of N-grams

Count frequencies of n consecutive words. n=1: unigram (=BoW), n=2: bigram, n=3: trigram. Captures local context. `ngram_range=(1,2)`.

TF-IDF

$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$

$\text{TF}(t,d) = \frac{\text{count of } t \text{ in } d}{\text{total terms in } d}$ $\quad \text{IDF}(t) = \log_{10}\left(\frac{\text{total docs}}{\text{docs with } t}\right)$

Rare words → higher weight. Common → lower. `TfidfVectorizer()`.

All frequency methods: no semantics, sparse, high dim, OOV problem.

Text Similarity Metrics

Levenshtein: min single-char edits (insert, delete, sub). "kitten"→"sitting" = 3.

Euclidean: $d = \sqrt{\sum (a_i - b_i)^2}$. Magnitude only.

Cosine Similarity (most used in NLP): $\cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \, \|\vec{B}\|}$

$\cos = 1$: identical. $\cos = 0$: no similarity. $\cos = -1$: opposite.

Used as **benchmark** to compare representations. TF-IDF + cosine > BoW + cosine (weights rare words higher).

Ethical concerns: false similarity/dissimilarity, term/topic bias (popular topics overweighted), language bias (English dominance on internet).

L4: Word Embedding

Motivation
Count-based: sparse, high-dim, no semantics, OOV. Goal: dense vectors with semantic meaning. **Distributional hypothesis**: words in similar contexts are similar.

WordNet
Lexical database (Princeton). Synsets (synonyms), gloss (definition), relations (hypernym, hyponym, meronym, antonym). Used for **query expansion**. Not computational, static, English-only.

Word Embedding
Maps words to dense vectors: $f : V \to \mathbb{R}^D$. Self-supervised, prediction-based. Dim D is hyperparameter (50–300). **Analogy**: $\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$

Word2Vec (Google, 2013)
CBOW: context words → predict center word. **Skip-gram**: center word → predict context words. Architecture: one-hot → hidden layer ($V \times D$ weight matrix) → softmax. After training, weight matrix = embedding.
SGNS: Skip-gram + negative sampling. Positive pairs (target, context) vs. random negative pairs.

	CBOW	**Skip-gram**
Input	Context	Target
Output	Target	Context
Speed	Faster	Slower
Best for	Freq. words	Rare words

Gensim: `Word2Vec(sg=0/1, vector_size, window, negative)`. Pre-trained: `word2vec-google-news-300` (3M words, 100B tokens).

GloVe (Stanford, 2014)
Prediction + **global co-occurrence matrix**. $\vec{x_i} \cdot \vec{x_j} \approx \log(X_{ij})$. $J = \sum f(X_{ij})(\vec{w_i}^T\vec{w_j} + b_i + b_j - \log X_{ij})^2$ Pre-trained: Wiki+Gigaword 6B, CommonCrawl 42B/840B, Twitter 27B. Dims: 50–300.

FastText (Facebook, 2016)
Character n-grams (3–6 chars). Word vector = sum of n-gram vectors. "cities" → ⟨ci, cit, iti, tie, ies, es⟩. **Handles OOV**: shared n-grams give approximate vectors for unseen words. Captures morphological info.

OOV Handling
1. Default zero vector (spaCy). 2. Synonym fallback (WordNet). 3. Train on own corpus.

Evaluation
Intrinsic: word similarity, word analogy. **Extrinsic**: downstream task performance.

Limitations
Limited context (window only), bias in training data, static vectors (no polysemy), dim choice, resource intensive. → **Contextualized representations** (BERT, Transformers).

L5: Language Models

Data Collection
Social media APIs (X/Twitter), web scraping (BeautifulSoup, lxml, html5lib), PDF files (PyPDF2).

Probability Review
$P(A,B,C) = P(A) \times P(B) \times P(C)$ (independent) $P(X|Y) = P(X,Y)/P(Y)$ (conditional) **Chain rule**: $P(X_1,...,X_n) = \prod_{i=1}^{n} P(X_i|X_1,...,X_{i-1})$

Language Model
Predicts next word: $P(w_{t+1}|w_1,...,w_t)$. Prob distribution over vocab; highest prob = prediction.

N-gram LM (Statistical)
$P(w_n|w_1,...,w_{n-1}) = \frac{C(w_1,...,w_n)}{C(w_1,...,w_{n-1})}$ **Markov assumption**: w_t depends only on previous $n-1$ words. Limitations: choosing n, sparsity (zero prob for unseen), limited context. **Backoff**: fall back to smaller n.

Neural Network LM
Input → one-hot → embedding → concat/avg → hidden → softmax → prediction. Still needs **fixed window**. **Averaging problem**: same words, different order → same vector.

Deep Neural Networks
UAT (Cybenko 1989): NN approximates any continuous function. DNN: benefits from more data (no plateau). CNN: images. **RNN: sequences**.

RNN Language Model
Stateful: $h_t = f(W_x x_t + W_h h_{t-1} + b)$ Sequential processing. Hidden state propagates info. Variable length. Shared weights. Self-supervised. $L_{total} = \sum_{t=1}^{T} L_t$. **BPTT**: backprop through time.

Vanishing Gradient
$\frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_4}{\partial h_3} \cdot \frac{\partial L}{\partial h_4}$ Small derivatives multiply → gradient → 0. Model stops learning for early tokens. Cannot capture **long-range dependencies**.

LSTM (Hochreiter & Schmidhuber, 1997)
Adds **cell state** c_t (long-term memory) + hidden state h_t (short-term). Three gates control info flow:
Forget: $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ **Input**: $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ **Candidate**: $\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$ **Cell update**: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ **Output**: $o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$ **Hidden**: $h_t = o_t \odot \tanh(c_t)$
Sigmoid: 0–1 (gate). Tanh: −1 to 1. $\odot$: element-wise multiply.

Perplexity
$\text{PPL}(W) = P(w_1,...,w_N)^{-1/N}$. Low = good. High = confused.

GRU
Simplified LSTM: 2 gates (reset, update), no cell state. Faster.

Evolution
N-gram → NN LM → RNN → LSTM/GRU → Transformer.

L6: RNN, LSTM, Seq2Seq, Attention

RNN Recap
$h_t = f(W_x x_t + W_h h_{t-1} + b)$. Sequential input, hidden state propagates info. Variable length. Self-supervised.

BPTT & Vanishing Gradient
$W_{new} = W_{old} - \alpha \cdot \nabla L$. Chain rule through time. Small derivatives → gradient vanishes. Early tokens don't learn. → **LSTM** adds cell state + gates.

LSTM Gates Summary

Forget	σ	what to discard from c_{t-1}
Input	σ	what new info to add
Candidate	tanh	proposed new content
Cell update	–	$f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
Output	σ	what to output
Hidden	tanh	$o_t \odot \tanh(c_t)$

Sequence Problem Types

1-to-1	Single→Single	Image classif.
1-to-many	Single→Seq	Image caption
Many-to-1	Seq→Single	Sentiment
Many-to-many	Seq→Seq	Translation

Bidirectional LSTM
Forward ($\overrightarrow{h_t}$) + backward ($\overleftarrow{h_t}$) LSTMs. $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$. Captures left & right context. Separate weights. Ex: "terribly exciting" – forward-only misreads "terribly" as negative.

Multi-layer RNN/LSTM
Stack layers: hidden states from layer i → inputs to layer $i+1$. Learns increasingly abstract representations.

Seq2Seq (Encoder-Decoder)
Encoder: reads input → fixed-length **context vector**. **Decoder**: context vector → output sequence token by token. Conditional LM: $P(y_1,...,y_T|x_1,...,x_S)$. Training: end-to-end backprop. Testing: argmax at each decoder step.

Bottleneck Problem
Entire input compressed to single fixed-length vector → information loss for long sequences.

Attention Mechanism
Solution to bottleneck. Decoder looks at **all** encoder hidden states at each step: 1. **Dot product**: decoder state vs. each encoder state → scores. 2. **Softmax**: scores → attention weights. 3. **Weighted sum**: weights × encoder states → attention output. 4. **Concat**: attention output + decoder state → prediction.
Benefits: variable-length, long-range dependencies, focus on relevant parts.

GRU vs LSTM

	LSTM	**GRU**
Gates	3	2 (reset, update)
States	$h_t + c_t$	h_t only
Params	More	Fewer
Speed	Slower	Faster

Path to Transformers
N-gram → NN LM → RNN → LSTM/GRU → Seq2Seq + Attention → **Transformer** (2017). No recurrence, entirely self-attention, handles long-range dependencies.
Keras: `Sequential([Embedding(..), LSTM(128), Dense(vocab, 'softmax')])`. Bi-LSTM: `Bidirectional(LSTM(n))`.