**NLP-Evolution**

Machines could never understand non-numeric characters. So they must be represented in some numeric form so that machines can understand them. The earliest way humans could give numerical representations to text was through ASCII encoding and then, we had label encoding or one hot encoding and so on. They were required to digitally represent words so computers could process them.

Natural Language Processing (NLP) has undergone a radical transformation — from rule-based systems and bag-of-words models to transformer-based generative models that understand language with surprising fluency. Let’s trace this journey.

1. ASCII / Unicode: Text is represented as sequences of bytes. Example: "A" → 65 in ASCII.
2. One-Hot Encoding: Each word is a vector where only one index is 1, rest are 0.

What’s wrong with them?

One-hot vectors are sparse and don’t convey meaning. Order or context or similarity didn’t matter to them. They weren’t understanding that “king” and “queen” are related, or that “run” and “jog” are similar. No understanding that “cat ate dog” and “dog ate cat” are different. Also, the vectors are high-dimensional and inefficient. This was till 1980s.

How to encode sentences in this process? We tokenize the sentences, build a bag of words or corpus. This is when BOW, Count Vectorizer, TF-IDF Vectorizing came in. Each sentence is represented by a binary vector of length equal to corpus, with ones for those words present in sentence and zero otherwise. While tokenizing, to improve the efficiency and to avoid unnecessary dimensions, we use stemming and lemmatization to reduce the words to their root form and basic form (word stripped of tense, degree, prefix, suffix etc)

1. As computing power and text data (like news corpora) exploded, the field shifted toward probabilistic modeling — replacing rules with statistics (1990-2000).

That involved

n-gram models: Predict words based on the last n−1 words.

Hidden Markov Models (HMMs): Used for POS tagging and speech recognition

Naive Bayes and Maximum Entropy Models: For text classification and parsing.

1. Word Embeddings (Word2Vec, GloVe): Evolving around 2010-2013, Words are represented in dense vectors that capture semantic similarity. Word embeddings are dense vector representations of words in a continuous vector space such that words with similar meanings are close together in that space. Instead of representing words as discrete symbols (like "cat", "dog"), embeddings represent them as points in a high-dimensional space where semantic similarity (like “cat” and “kitten”) is captured by mathematical proximity.

While Word2Vec (Skip-gram, CBOW) learns word meaning based on context, GloVe learns from word co-occurrence statistics. GloVe builds a co-occurrence matrix of word pairs across the entire corpus and each cell in the matrix indicates how often word A appears in the context of word B. The embeddings are learned to capture the ratios of these co-occurrence probabilities. FastText (Facebook, 2016) on the other hand, breaks words into subword units (n-grams). Eg. “unhappiness” → ["un", "happi", "ness"]. There are benefits with this approach that it handles out-of-vocabulary words and supports morphologically rich languages.

These added an advantage of capturing semantic relationships: e.g., vec("king") - vec("man") + vec("woman") ≈ vec("queen") and hence turned more efficient and meaningful and lesser dimensional than one-hot vectors.

Now, what was the limitation: The embeddings were static: “bank” (river bank vs financial bank) has the same vector in all contexts. There was no way to handle polysemy (multiple meanings). It also doesn’t capture sentence structure or syntax. Each number (dimension) doesn’t always have a named meaning, but it corresponds to a learned pattern from the corpus. However, it captures semantic and geometric relationship between words since, Direction captures relation (e.g., gender, size, location) and Distance captures similarity. Like CNN, some of these features might encode, Royalty-related or Action-oriented word or Formality level or Animal-related – None of them manually decided or chosen and all only learned from patterns in usage.

1. Each stage arose to solve limitations:

From counting words → to weighing them → to understanding meaning

From sparse math → to dense semantics

From fixed representations → to trainable embeddings

Next, during 2014-2017, we had deep learning models like:

* Sequential models that learnt patterns from sequences of words, which were basic Neural Nets. Instead of hand-crafting rules (statistics/heuristics) or assuming feature independence like Naive Bayes, we let neural networks learn patterns and relationships directly from vectorized data. N-grams and embeddings further enhanced its efficiency.
* RNNs that Pass hidden states from one word to the next
* LSTM/GRU that are improved versions with memory mechanisms.

These processed sequences of text and remembered context from earlier words. These enabled tasks like language modeling, text generation, translation.

What was wrong with it – They were slow to train (can't parallelize well), they struggle with long-term dependencies, viz, forgets context from earlier in the sentence and Vanishing gradients.

1. Attention Mechanisms and Transformers: Attention breakthrough happened around 2017. While attention allows the model to focus on relevant parts of the input sequence, the technology of transformer models (Vaswani et al., 2017) replaced recurrence with self-attention layers. These were needed since we were looking for better performance on long texts and more parallelizable architectures. Attention helps the model figure out which words are important for a given word. The initial models were still limited by fixed context windows. A bigger disadvantage is that it requires massive data and compute.
2. Pretrained Language Models: Models like BERT (2018): Bidirectional encoder that understands both left and right context and GPT (2018–2023+): Autoregressive decoder-only model for generation come under these. These are trained on massive corpora using self-supervised learning. These were required to pre-train general-purpose language understanding, reusable across tasks. This enabled Fine-tuning on smaller datasets. BERT couldn’t generate text. They were mostly text to machine models.
3. Instruction-Tuned & Chat Models (T5, ChatGPT, LLaMA): These were text to text modes. They have a reinforcement learning block that learns human conversational language. T5 model treats every NLP task as text-to-text. InstructGPT / ChatGPT is fine-tuned with human feedback (RLHF) to follow instructions better. These were required to make models more helpful, safe, and aligned with user intent and since models needed provision of human feedback to address hallucinations and bad outputs. What’s the limitations: Still there is hallucinations (makes up facts) to certain levels. It also requires costly reinforcement learning steps and it struggles with multi-step reasoning.
4. Multimodal & Agentic Models (GPT-4o, Gemini, Claude Opus): These can understand text, image, audio, code and can act as autonomous agents, complete tasks across apps.

**Stemming and Lemmatization:**

These techniques are part of the text preprocessing phase in traditional NLP pipelines, especially in the pre-embedding, pre-neural, and even early word embedding stages. They’re importantly used with Bag-of-Words (BoW), TF-IDF, Count Vectorizer and sometimes even with Word2Vec/GloVe. They’re less commonly used in modern Transformer-based models like BERT or GPT, which rely on their own tokenization and understand morphology better inherently.

Stemming is the process of chopping off word suffixes to reduce a word to its root form (stem), even if the stem is not a valid word. Example: running → run, played → play, happiness → happi("happi" is not a real word, but it's the stem). It uses simple rules or suffix-stripping (e.g., remove “-ing”, “-ed”, “-ness”). Here is a comparison of different stemming techniques

| **Stemming Algorithm** | **Language Support** | **Aggressiveness** | **Speed** | **Output Quality** | **Description** | **Use Case** |
| --- | --- | --- | --- | --- | --- | --- |
| **Porter Stemmer** | English | Moderate | Fast | Moderate | Rule-based suffix stripping with heuristics. One of the most common. | Information retrieval, search engines |
| **Lancaster Stemmer** | English | Very Aggressive | Very Fast | Low to Moderate | Shortest stem produced, aggressively removes suffixes. | Domain-specific tasks where over-stemming is acceptable |
| **Snowball Stemmer** | ~15 Languages | Moderate | Fast | Higher than Porter | Updated version of Porter with improved rules and support for multiple languages. | Multilingual applications, general NLP |
| **Lovins Stemmer** | English | Aggressive | Moderate | Low | One of the earliest stemmers, uses longest suffix matching. | Rarely used now; historical interest |
| **Regex-based Stemmer** | Customizable | Depends on design | Very Fast | Low | Based on regular expressions; simple pattern-matching. | Rule-based pipelines, fast prototyping |
| **KStem Stemmer** | English | Moderate | Moderate | High (for IR) | Designed for information retrieval. Tries to reduce overstemming. | IR-focused systems (e.g., search) |

Here is how different stemmers functions. Except for slight differences, most work same.

| **Word** | **Porter** | **Lancaster** | **Snowball** | **Lovins** | **Regex-Based** |
| --- | --- | --- | --- | --- | --- |
| **relational** | relat | relat | relat | relat | relation |
| **congratulations** | congratul | congrat | congratul | congratul | congratulation |
| **happiness** | happi | happy | happi | happy | happines |
| **universal** | univers | univers | univers | universal | universal |
| **running** | run | run | run | run | run |
| **organization** | organiz | organ | organis | organiz | organization |

Lemmatization, on the other hand is proper language based technique that reduces a word to its dictionary form (lemma), by considering its part of speech (POS) and context. Example: running → run, better → good, mice → mouse. Lemmas are always valid words. It requires linguistic knowledge and uses POS tagging and morphological analysis. It is slower, but smarter than stemming. Following are different approaches to lemmatization.

| **Technique Type** | **Method** | **Key Features** | **Examples / Tools** |
| --- | --- | --- | --- |
| **Rule-Based** | Uses grammar + suffix-stripping + POS tagging | Relies on hand-crafted rules and dictionaries | NLTK's WordNet Lemmatizer |
| **Dictionary-Based** | Maps words to lemmas using lexicons | Requires extensive dictionary resources | spaCy, TextBlob, Pattern |
| **Hybrid** (Rule + Dict) | Rule-based + dictionary fallback | More accurate and flexible | spaCy, Stanford CoreNLP |
| **Statistical / ML-based** | Learns lemmatization patterns from corpus | Requires annotated corpora (rare in practice) | Some deep learning models, UDPipe |
| **Neural Lemmatizers** | Seq2Seq or Transformer-based lemmatizers | Context-sensitive, handles morphology across languages | UDPipe, Stanza (Stanford), BERT-style models |

| **Tool** | **Based On** | **POS Aware?** | **Multilingual?** | **Speed** | **Accuracy** | **Example "better" → ?** |
| --- | --- | --- | --- | --- | --- | --- |
| **NLTK WordNet** | Rule + Dictionary | Yes, with tag | English only | Fast | Medium | "good" (if POS = adj) |
| **spaCy** | Rule + Lexicon | default | many languages | Very Fast | High | "good" (auto) |
| **TextBlob** | Rule + Pattern | No | English only | Fast | Medium | "better" (unchanged) |
| **Stanza (Stanford)** | Neural + Dict | Yes | UD models | Slower | Very High | "good" |
| **UDPipe** | ML-based | Yes | Yes | Slower | High | "good" |

Without POS tagging provided, WordNet might fail or return unchanged. spaCy and Stanza guess context from sentence and tag POS. Stanza and UDPipe even handle complex morphology in other languages. Neural models (like in Stanza, UDPipe, or BERT-based models) can generate lemmas from scratch even for rare or unseen words. Example: "had" → "have", "was" → "be" — based on context, not rules alone.

| **Goal** | **Best Lemmatizer** |
| --- | --- |
| Basic English lemmatization | NLTK WordNet (with POS) |
| High-speed, high-accuracy NLP | **spaCy** |
| Complex morphology / languages | **Stanza**, **UDPipe** |
| Deep context + lemmatization | Custom BERT / seq2seq model |