Word to Molecule, Part I

Yedarm Seong

Interdisciplinary Program in Artificial Intelligence, Seoul National University

Nov 07, 2025

Table of Contents

Table of Contents

How to represent words for LMs?

- One-hot Encoding & Bag-of-Words
- Distributed Representation (Word2Vec, GloVe, etc.)
- Contextualized Representation (Transformer, BERT, GPT, etc.)

One-hot Encoding

Definition

Given a vocabulary of size V, each word is represented as a V-dimensional vector. This vector contains all zeros, except for a single '1' at the index corresponding to the word.

Example: One-hot Encoding

Consider the sentence: `the cat sat on the mat`

1. Build Vocabulary: {the, cat, sat, on, mat} (Size $V = 5$)
2. Assign Unique Indices:

$$\begin{aligned} \texttt{the} &\rightarrow 0 \\ \texttt{cat} &\rightarrow 1 \\ \texttt{sat} &\rightarrow 2 \\ \texttt{on} &\rightarrow 3 \\ \texttt{mat} &\rightarrow 4 \end{aligned}$$

3. Convert to One-Hot Vectors:
   - `cat = [ 0, 1, 0, 0, 0 ]`
   - `mat = [ 0, 0, 0, 0, 1 ]`

Document Representation: Bag of Words (BoW)

The Bag of Words (BoW) model is a simple way to represent an entire document (like a sentence or an article) as a single numerical vector.

Definition

BoW represents a document by counting the occurrences of each word from the vocabulary, while **disregarding grammar and word order**. This approach figuratively places all words from the document into a "bag" and only considers their frequencies.

Example: Bag of Words (BoW)

Consider the documents (sentences):

- **Doc 1:** `the cat sat on the mat`
- **Doc 2:** `the cat chased the dog`

1. Build Vocabulary (from all docs): {the, cat, sat, on, mat, chased, dog} (Size $V = 7$)
2. Assign Unique Indices:

`the`	$\rightarrow 0$	`mat`	$\rightarrow 4$
`cat`	$\rightarrow 1$	`chased`	$\rightarrow 5$
`sat`	$\rightarrow 2$	`dog`	$\rightarrow 6$
`on`	$\rightarrow 3$		

3. Convert to Bag of Words Vectors (by frequency):
   - **Doc 1** = `[ 2, 1, 1, 1, 1, 0, 0 ]`
   - **Doc 2** = `[ 2, 1, 0, 0, 0, 1, 1 ]`

   (Note: Word order is lost, only the counts are kept.)

Distributed Representation

To overcome the "Sparsity" and "No Semantic Meaning" limitations of One-hot Encoding and BoW, we use **Distributed Representation**, commonly known as **Word Embedding**.

Definition

The distributed representation is a real-valued vector that encodes the meaning of the word in such a way that the words that are closer in the vector space are expected to be similar in meaning. Word embeddings can be obtained using language modeling and feature learning techniques, where words or phrases from the vocabulary are mapped to vectors of real numbers.

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

Capturing Semantics

The key advantage is that **semantically similar words have similar vectors** (i.e., they are close in the vector space). This allows for capturing relationships and performing vector arithmetic:

- `vector("King") - vector("Man") + vector("Woman")` $\approx$ `vector("Queen")`
- `vector("Paris") - vector("France") + vector("Germany")` $\approx$ `vector("Berlin")`

Popular Models for Learning Embeddings

- **Word2Vec (Google):** Learns by predicting context words (Skip-gram) or the target word (CBOW).
- **GloVe (Stanford):** Learns by factorizing a global co-occurrence matrix (counts how often words appear together).
- **FastText (Facebook):** Extends Word2Vec by learning vectors for sub-word units (n-grams), allowing it to handle new/unknown words.

Contextualized Representation

Word2Vec and GloVe give us **static** embeddings. The word "bank" has only **one** vector, regardless of its meaning. (e.g., "river **bank**" vs. "**bank**")

Definition

These models generate word representations that are **dynamically** based on the **entire context** (the full sentence or document).

The Effect of Context

- **Static (Word2Vec):**
- `vector("bank")` $\rightarrow$ `[-0.5, 1.2, 0.9, ···]` (Always the same)
- **Contextualized (Transformer):**
- `vector("bank" | I sat on the river bank.)`
  $\rightarrow$ `[0.8, 0.2, -0.4, ···]`
- `vector("bank" | I went to the bank to deposit money.)`
  $\rightarrow$ `[-0.7, 1.5, 0.1, ···]`

The Transformer Era

- **Transformer (Vaswani et al., 2017):** Encoder-Decoder architecture.
- **GPT (Radford et al., 2018):** Uses the **Transformer Decoder**.
  Learns via **teacher-forced next-word prediction**.
  Learns via **auto-regressive next-word prediction**.
- **BERT (Devlin et al., 2019):** Uses the **Transformer Encoder**.
  Learns deeply **bidirectional representations by predicting masked-words**.

Table of Contents

What is a Tokenizer?

- The process of converting text into a sequence of numbers (tokens) that a computer can understand.
- "Hello, world!" $\rightarrow$ ‘[101, 7592, 1010, 2088, 102]‘
- **Why is this necessary?**
  - Language models (e.g., BERT, GPT) require numerical input.
  - Text must be split into meaningful units for processing.
- **The Classic Problem:** OOV (Out-of-Vocabulary)
  - If a vocabulary is built from "words," any word not in the vocabulary (new words, typos, names) is mapped to <UNK> (Unknown).
  - Example: "brunch" $\rightarrow$ <UNK>
  - This results in a significant loss of information.

The Rise of Subwords

Break words into smaller, meaningful units (subwords).

- The vocabulary consists of "subwords," not "words."
- **Examples:**
  - "tokenization" $\rightarrow$ "token + ##ization"
  - "playing" $\rightarrow$ "play" + "##ing"
- **Advantages:**
  - **Solves OOV:** Unknown words (e.g., "brunch") can be represented as a combination of known subwords ("br" + "##unch").
  - **Morphological Info:** The model can learn that "playing" and "plays" share a common root ("play").
  - **Vocabulary Size:** Achieves a balance between a massive word-level vocabulary and a small character-level one.

Byte-Pair Encoding (BPE)

- Initially a data compression algorithm (Gage, 1994).
- Adapted for NLP (Sennrich, Haddow, and Birch, 2016) to handle rare words.
- Famously adopted by Radford et al., 2018.
- **Training Process:**
  1. Start with a vocabulary of all individual characters.
  2. **Iteratively:** Find the most frequent pair of adjacent tokens (bytes/characters) in the corpus.
  3. Merge this pair into a new, single token and add it to the vocabulary.
  4. Repeat for a desired number of merges (this determines the final vocabulary size).

Example: BPE Training

1. Corpus: `{ "hug": 10, "pug": 5, "pun": 12, "bun": 4, "hugs": 5 }`
2. Initial Vocabulary: `{h, u, g, p, n, b, s}`
3. Find most frequent pair: `(u, g)` appears 20 times.
4. Merge: `"ug"` is added to the vocab.
5. Corpus becomes: `{ "hug": 10, "pug": 5, "pun": 12, "bun": 4, "hugs": 5,` **`"ug": 20`** `}`
6. Next most frequent pair: `(h, ug)` appears 10 times.
7. Merge: `"hug"` is added to the vocab.
8. This process continues until the target vocabulary size is reached.

WordPiece: Maximizing Likelihood

- Which merge will increase the **likelihood** of the training data the most?
- Introduced by Google for NMT (Neural Machine Translation) in 2012.
- Famously adopted by Devlin et al. (2019).
- This is a more statistically principled way to build the vocabulary.
- WordPiece merges based on **likelihood**, whereas BPE merges based on **frequency**.

Example: WordPiece Training

1. Corpus: `{ "hug": 10, "pug": 5, "pun": 12, "bun": 4, "hugs": 5 }`
2. Initial Vocabulary: `{h, ##u, ##g, p, ##n, b, ##s}`
3. Calculate likelihood of the corpus given the current vocab.

$$f(a, b) = \frac{\text{Freq}(a, b)}{\text{Freq}(a)\,\text{Freq}(b)}$$

4. Choose the merge that maximizes likelihood (e.g., `(##g, ##s)`).
5. Update corpus and repeat the process until the desired vocabulary size is reached.

Table of Contents

Decoder-only LMs: Causal Language Modeling

Causal Language Modeling is also known as **Autoregressive modeling**.

Definition

The task of predicting the next token (w_i) given all previous tokens ($w_1, \ldots, w_{i-1}$) up to the current time step (i).

Formula

The probability of a token sequence $W = (w_1, w_2, \ldots, w_N)$ is:

$$P(W) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \times \ldots$$

This can be generalized as:

$$P(W) = \prod_{i=1}^{N} P(w_i|w_1, \ldots, w_{i-1})$$

Encoder-only LMs: Masked Language Modeling

Masked Language Modeling (MLM) is a training objective where certain tokens in the input sequence are randomly masked, and the model is trained to predict these masked tokens based on their surrounding context.

Definition

The task of predicting a masked token (w_i) given all other tokens in the sequence, both before and after the masked position.

Formula

Given a sequence $W = (w_1, w_2, \ldots, w_N)$ with a masked token at position i, the probability is:

$$P(w_i | w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_N)$$

The overall objective is to maximize the likelihood of all masked tokens in the training data.

Table of Contents

Molecule Representations

Common ways to represent molecules for computational tasks:

- **Molecular Graphs:** Nodes represent atoms, and edges represent bonds between them.
- **SMILES (Simplified Molecular Input Line Entry System):** A linear string representation of a molecule.
- **SELFIES (Self-Referencing Embedded Strings):** A robust string representation designed to always produce valid molecules.
- **InChI (International Chemical Identifier):** A textual identifier that encodes a chemical substance's structure.

SMILES Representation

- SMILES is a widely used notation to represent chemical structures as linear strings.
- Atoms are represented by their atomic symbols (e.g., C for carbon, O for oxygen).
- Bonds are represented by symbols (e.g., single bond: '-', double bond: '=', triple bond: '#').
- Branches and rings are indicated using parentheses and numbers, respectively.

Example: Ethanol (C2H6OH) is represented as `CCO` in SMILES.

Disadvantages of SMILES

- Syntactically invalid strings:
  - `C1CC` (incomplete ring) or `C=O=O` (invalid bonding).
- Ambiguity:
  - Ethanol can be `CCO` or `C(C)O`.
  - Canonical SMILES: A unique representation for each molecule.
  - Diffrerent software may produce different canonical SMILES for the same molecule.
- Sensitivity to small changes:
  - `CCO` (ethanol) vs. `CCN` (ethylamine).
- Difficulty in capturing 3D structure:
  - SMILES is a 2D representation and does not convey stereochemistry effectively.

SELFIES Representation

- SELFIES is a more recent representation designed to ensure that every string corresponds to a valid molecule.
- It uses a set of predefined tokens that represent atoms, bonds, and structural features.
- The SELFIES grammar guarantees that any combination of these tokens will produce a chemically valid structure.

Example: Ethanol (C2H6OH) can be represented `[C][C][O]` in SELFIES.

Disadvantages of SELFIES

- Increased Length:
  - Ethanol is `CCO` in SMILES but `[C][C][O]` in SELFIES.
- Limited Adoption:
  - Many cheminformatics tools primarily support SMILES.
  - Conversion between SELFIES and SMILES may be required.
- Potential Redundancy:
  - Different SELFIES strings can encode the same molecular structure such as `[C][C][O]` and `[C][O][C]` for ethanol.
- 3D Structure Limitations:
  - Similar to SMILES, SELFIES does not inherently capture 3D conformational information.

InChI Identifier

- InChI is a textual identifier that encodes a chemical substance's structure in a standardized way.
- It consists of layers and sub-layers that provide detailed information about the molecule, including connectivity, tautomeric information, isotopes, stereochemistry, and electronic charge.
- InChI is designed for easy database searching and indexing of chemical compounds.
- **Example:** Ethanol's InChI is `InChI=1S/C2H6O/c1-2-3/h3H,2H2,1H3`.

Benefits of InChI

- Standardization:
  - InChI provides a consistent way to represent chemical substances, facilitating data sharing and integration across different databases.
- Uniqueness:
  - Each unqique chemical structure corresponds to a unique InChI string, making it easier to identify and differentiate compounds.
- Hierarchical Structure:
  - The layered format allows for detailed representation of various molecular features, enabling more precise.
- Compatibility with Databases:
  - InChI is widely used in chemical databases, making it a valuable tool for researchers and chemists.

Benefits of InChI (Cont.)

(S)-Alanine:

`C[C@@H](N)C(=O)O`

`[C][C@@H][N][C][=O][O]`

`InChI=1S/C3H7NO2/c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6)/t2-/m0/s1`

- InChI's layered structure allows for the separation of different molecular features (e.g., connectivity, stereochemistry), which can be advantageous for machine learning models that benefit from disentangled representations.
- In SMILES, all information—backbone (connectivity), functional groups, and stereochemistry (e.g., @@)—is 'entangled' within a single string.
- InChI clearly separates information into distinct layers. /c... (Connectivity, backbone) /h... (Hydrogens) /s... (Stereochemistry)

Table of Contents

Chemical Language Models (LMs)

- Chemical LMs are specialized language models designed to understand and generate chemical structures and properties.
- They leverage techniques from natural language processing (NLP) to process chemical representations like SMILES, SELFIES, and InChI.
- Applications include drug discovery, material science, and molecular property prediction.

Examples of Chemical LMs

- **MolBERT (Li and Jiang, 2021):**
  - An encoder-only model using SMILES for various chemical tasks.
  - Smiles to atom identifier and masked SMILES prediction during pre-training.
- **SMILES-BERT (Wang et al., 2019):**
  - A BERT model specifically designed for SMILES representation.
  - Training by tokenizing SMILES strings to the character-level without considering the molecular structure.
  - Conduct only binary classification tasks.

The models have to standardize and canonicalize SMILES strings before training to handle the representation issues.

InChI-based Chemical LMs

- Multiple Tokenizers:
  - Split by / (layers)
  - Formula layer tokenization: C3H7NO2 → C, 3, H, 7, N, O, 2
  - Connectivity layer tokenization: /c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6) → /c, 1, -, 2, (, 4,), 3, (, 5,), 6, /h, 2, H, ,, 4, H2, ,, 1, H3, (, H, ,, 5, ,, 6,)
  - Hydrogen layer tokenization: /h2H,4H2,1H3,(H,5,6) → /h, 2, H, ,, 4, H2, ,, 1, H3, (, H, ,, 5, ,, 6,)
  - Stereochemistry layer tokenization: /t2-/m0/s1 → /t, 2, -, /m, 0, /s, 1

InChI-based Chemical LMs (Cont.)

`InChI=1S/C3H7NO2/c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6)/t2-/m0/s1`

- Where to mask?:
  - `1S/[MASK]/c1-2(4)3(5)6/h2H,4H2,1H3,(H,5,6)/t2-/m0/s1`
  - `1S/C3H7NO2/c1-2(4)3(5)6/h2H,[MASK],1H3,(H,5,6)/t2-/m0/s1`
  - `1S/C3H[MASK]NO2/c1-2([MASK])3(5)6/h2H,[MASK],1H3,(H,5,6)/t2`

InChI-based Chemical LMs (Cont.)

- Multi-Head Attention:
  - Local Attention (In-Layer Context): Most tokens perform attention only within a fixed, predefined range of neighboring tokens (i.e., a "sliding window"), which captures the 'In-Layer Context'. This is effective for understanding the local context within a sentence.
  - Global Attention (Out-Layer Context): A few specific tokens, such as [CLS] or other tokens deemed important for the task, perform attention across all other tokens in the entire sequence (i.e., the 'Out-Layer Context'). This allows the model to capture key information and long-range dependencies across the full sequence.

Summary: Local Attention for In-Layer Context + Global Attention for Out-Layer Context with Multiple Tokenizers for InChI string.

Table of Contents

References

- Firth, John (1957). "A synopsis of linguistic theory, 1930-1955". In: *Studies in linguistic analysis*, pp. 10–32.
- Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30.
- Radford, Alec et al. (2018). "Improving language understanding by generative pre-training". In.
- Devlin, Jacob et al. (2019). "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)*, pp. 4171–4186.
- Gage, Philip (1994). "A new algorithm for data compression". In: *C Users Journal* 12.2, pp. 23–38.
- Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). "Edinburgh neural machine translation systems for WMT 16". In: *arXiv preprint arXiv:1606.02891*.