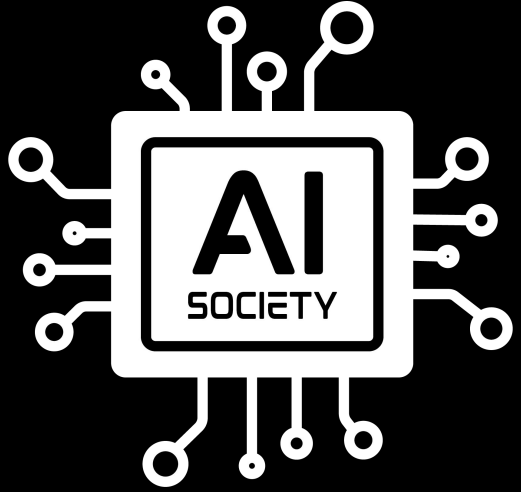
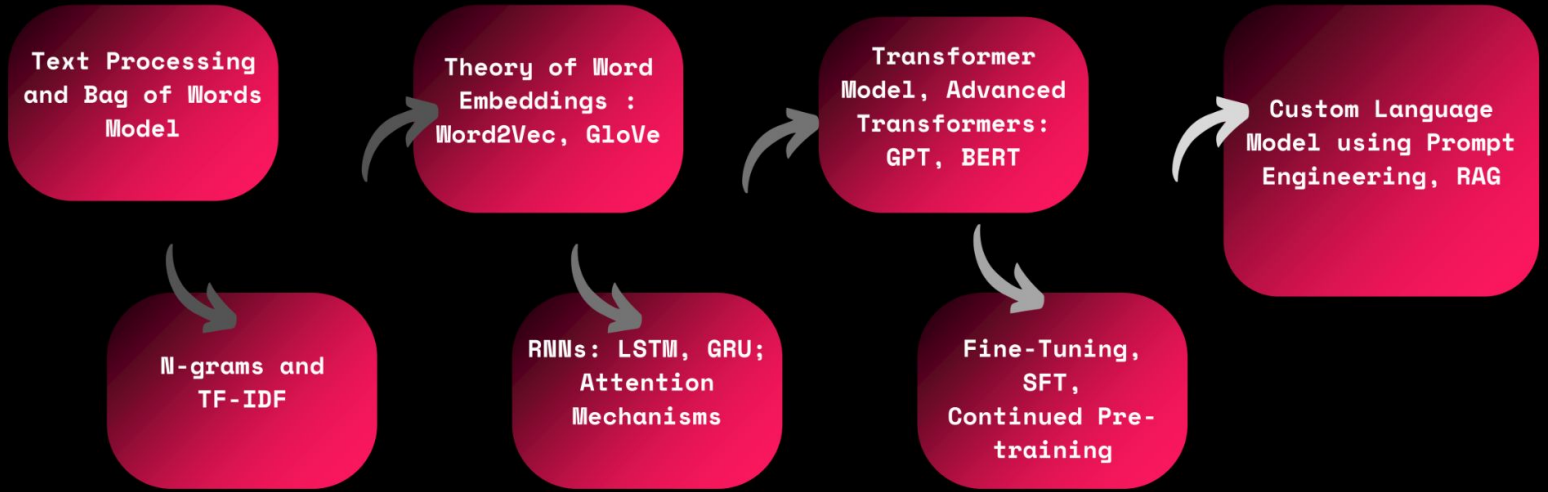


Word Embeddings

Week #3 - 25 Sept ‘24

NLP Lab

NLP Lab Project : Build your own GPT



1. Introduction to Word Embeddings

1.1 Word Embeddings

Definition: Representation of (meaning of) words in continuous vector space.

- Each word is mapped to a high-dimensional vector based on its context in the corpus.
- Words that occur in similar contexts tend to have similar meanings - Distributional Hypothesis.
- Key Characteristics:
  - Dense vectors.
  - Fixed dimensionality.
  - Similar words have similar vectors
- Why use Word Embeddings?

Traditional representations (e.g., one-hot encoding) are sparse and fail to capture meaning. Word embeddings solve this by encoding syntactic and semantic meaning.

1.X Cosine for Measuring Similarity

- To measure similarity between two target words v and w, the most common similarity metric is the cosine of the angle between the vectors.
- The cosine based on dot product the dot product operator from linear algebra, also called the inner product:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions.
- This raw dot product, however, has a problem as a similarity metric :

It favors long vectors….

1.X Problem in Raw Dot Product

- The dot product is higher if a vector is longer, with higher values in each dimension.
- More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them.
- To counter this, we modify the dot product to normalize for the vector length by dividing the dot product by the lengths of each of the two vectors.

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

- This normalized dot product turns out to be the same as the cosine of the angle between the two vectors

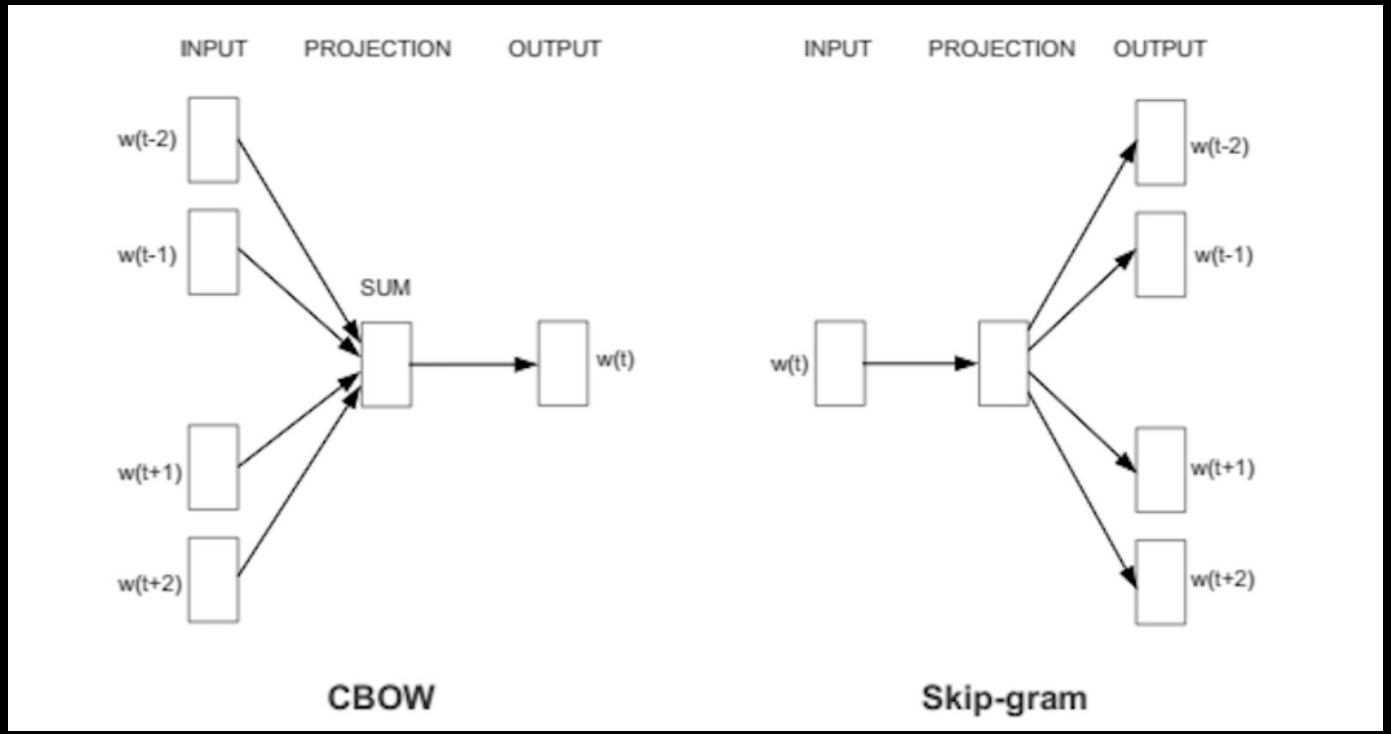
2. Word2Vec

2.1 Intro to Word2Vec

- Developed by Google, it is a shallow two-layer neural network-based model to create static word embeddings.
- It takes as its input a large corpus of words and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.
- Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.
- Intuition of Word2Vec :
  - A. The intuition of word2vec is that instead of counting how often each word w occurs near, say, apricot, we'll instead train a classifier on a binary prediction task: "Is word w likely to show up near apricot?"
  - B. We don't actually care about this prediction task; instead, we will just take its hidden weights and use them as our word embeddings, and toss the rest of the model

Source : https://web.stanford.edu/~jurafsky/slp3/6.pdf

2.2 Architecture of Word2Vec



Source Image : https://israelg99.github.io/2017-03-23-Word2Vec-Explained/

2.3 Skip-gram

- Skip-gram trains a probabilistic classifier that, given a test target word w and its context window of L words $c_{1:L}$, assigns a probability based on how similar this context window is to the target word.
- How does the classifier compute the probability ?

The intuition of the skip-gram model is to base this probability on embedding similarity: a word is likely to occur near the target if its embedding vector is similar to the target embedding.

$$\text{Similarity}(w, c) \approx c \cdot w$$

- The dot product $c \cdot w$ is not a probability. To turn the dot product into a probability, we'll use the logistic or sigmoid function $\sigma(x)$.
- But how do we take into account all the context words?

Skip-gram makes the simplifying assumption that all context words are independent, allowing us to just multiply their probabilities.

2.3 Learning Skip-gram (with negative sampling) Embeddings

- Random embedding vectors for each word in vocabulary is passed as an input to the classifier (neural network).
- It then proceeds to iteratively shift the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don’t occur nearby.
- The hidden layer is a standard fully-connected (Dense) layer whose weights are the word embeddings.
- The output layer outputs probabilities for the target words from the vocabulary.
- But this is computationally expensive.

Negative sampling reduces computation by sampling just N negative instances along with the target word instead of sampling the whole vocabulary.

2.3 Learning Skip-gram (with negative sampling) Embeddings

- Negative sampling samples negative instances (words) along with the target word and minimizes the log-likelihood of the sampled negative instances while maximizing the log-likelihood of the target word.
- Skip-gram with negative sampling uses more negative examples than positive examples (with the ratio between them set by a parameter k).

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    w     c3        c4
```

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

2.3 How the negative samples are chosen?

- The negative samples are chosen using a unigram distribution.
- Essentially, the probability for selecting a word as a negative sample is related to its frequency, with more frequent words being more likely to be selected as negative samples.
- Specifically, each word is given a weight equal to it's frequency (word count) raised to the 3/4 power. The probability for a selecting a word is just it's weight divided by the sum of weights for all words.

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

- $p_\alpha(w)$ is the weighted unigram frequency in the above formulae, where α is a weight.

2.4 Some more key Points...

- Skip-gram model learns two separate embeddings for each word i:
  a. the target embedding w_i , and
  b. the context embedding c_i,

  stored in two matrices, the target embedding context embedding target matrix ***W*** and the context matrix ***C***.
- To get the final word embedding, we typically add the two embeddings : $\boldsymbol{w_i + c_i}$
- Accuracy increases overall as the number of words used increase, and as the number of dimensions increases.
- Quality of word embedding increases with higher dimensionality. However, after reaching some threshold, the marginal gain will diminish. Typically, the dimensionality of the vectors is set to be between 100 and 1,000.
- According to the authors' note, the recommended value is 10 for skip-gram and 5 for CBOW.

2.5 Fasttext

- An extension of Word2Vec, addresses a problem with Word2vec : it has no good way to deal with ***unknown words***—words that appear in a test corpus but were unseen in the training corpus.
- A related problem is word sparsity, such as in languages with rich morphology, where some of the many forms for each noun and verb may only occur rarely.
- Fasttext deals with these problems by using subword models, representing each word as itself plus a bag of constituent n-grams, with special boundary symbols < and > added to each word.
- For example, with n = 3 the word where would be represented by the sequence *<where>* plus the character n-grams: `<wh, whe, her, ere, re>`
- Then a skip-gram embedding is learned for each constituent n-gram, and the word where is represented by the sum of all of the embeddings of its constituent n-grams.

2.4 Advantages of Word2Vec

- **Efficient Training :** Handles large corpora using optimization techniques (e.g., negative sampling).
- **Semantic Relationships :** Can capture analogies (e.g., "king" - "man" + "woman" = "queen").
- **Scalable :** Works well with billions of words.
- **Applications :** Semantic similarity, question answering, entity recognition.

3. GloVe

Test you knowledge!