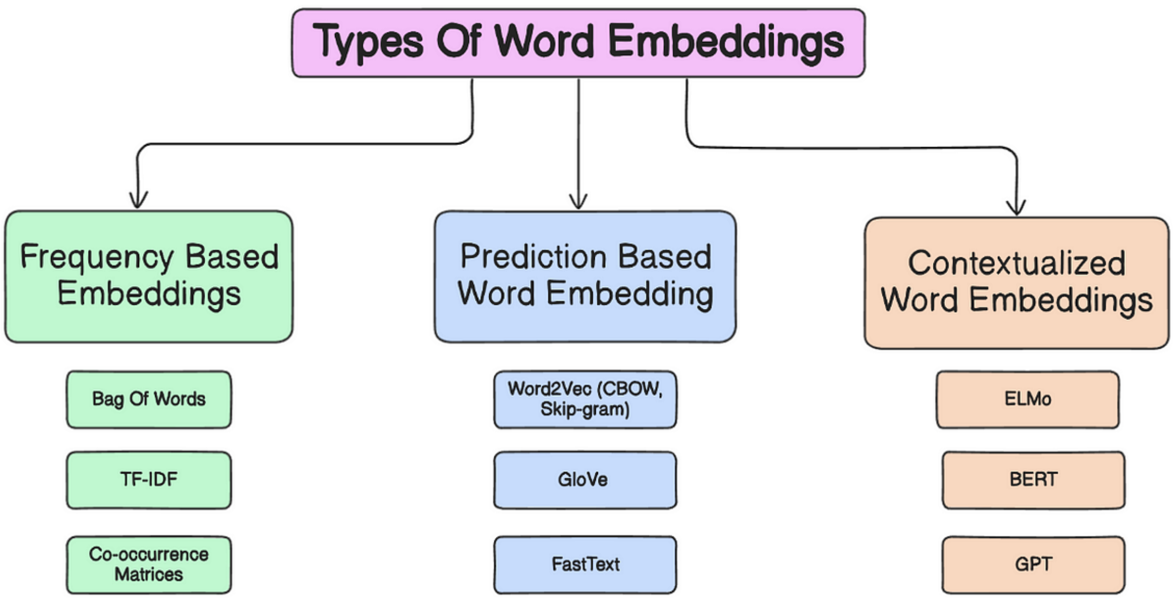


Word Embeddings and Representations

Word embeddings are numerical representations of words in a high-dimensional space. They help capture semantic meanings and relationships between words, enabling machines to understand human language.

---

1. Word Embeddings

1.1 TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF is a statistical measure that evaluates how important a word is in a document relative to a collection of documents (corpus).

Mathematics of TF-IDF

- **Term Frequency (TF):**

TF(w)=Number of times word w appears in a documentTotal words in the documentTF(w) = \frac{\text{Number of times word } w \text{ appears in a document}}{\text{Total words in the document}}

- **Inverse Document Frequency (IDF):**

IDF(w)=log⁡(Total number of documentsNumber of documents containing word w+1)IDF(w) = \log \left(\frac{\text{Total number of documents}}{\text{Number of documents containing word } w} + 1 \right)

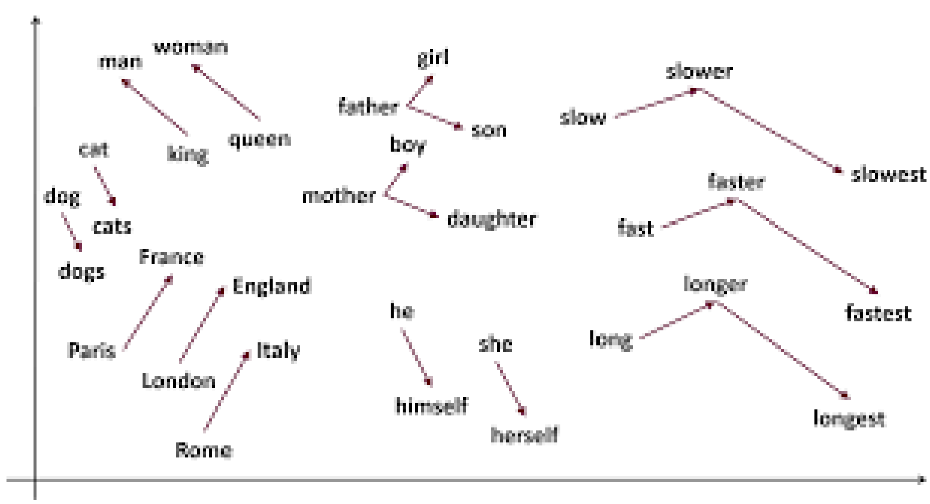
- **TF-IDF Score:**

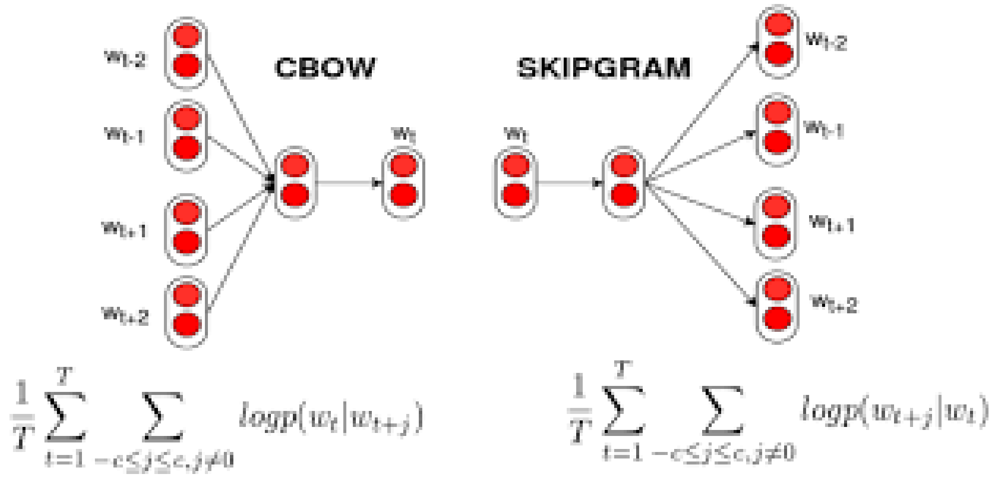
TF-IDF(w)=TF(w)×IDF(w)TF\text{-}IDF(w) = TF(w) \times IDF(w)

Python Implementation

```
from sklearn.feature_extraction.text import TfidfVectorizer


documents = [
    "Natural language processing is amazing",
    "Deep learning and NLP are closely related",
    "Word embeddings capture semantic meaning"
]


vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)


print("TF-IDF Matrix:")
print(tfidf_matrix.toarray())


print("\nFeature Names:", vectorizer.get_feature_names_out())
```

1.2 Word2Vec (CBOW & Skip-gram)





Word2Vec is a neural network-based model that learns word representations. It has two architectures:

- **CBOW (Continuous Bag of Words):** Predicts the target word from surrounding words.
- **Skip-gram:** Predicts surrounding words given a target word.

Mathematics

- **CBOW Objective Function:**

arg max θ∑context words clog P(w∣c;θ)\arg\max_{\theta} \sum_{\text{context words } c} \log P(w | c; \theta)

- **Skip-gram Objective Function:**

arg max θ∑target word w∑context words clog P(c∣w;θ)\arg\max_{\theta} \sum_{\text{target word } w} \sum_{\text{context words } c} \log P(c | w; \theta)

Python Implementation

```
from gensim.models import Word2Vec

from nltk.tokenize import word_tokenize


# Sample text corpus

corpus = [

    "Deep learning is amazing",

    "Natural language processing and word embeddings",

    "Word2Vec learns semantic meaning from text"

]
```

```
# Tokenize sentences
tokenized_corpus = [word_tokenize(sentence.lower()) for sentence in corpus]


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


# Get word vector
print("Vector for 'deep':")
print(model.wv['deep'])


# Find most similar words
print("\nMost similar words to 'learning':")
print(model.wv.most_similar('learning'))
```

1.3 GloVe (Global Vectors for Word Representation)

GloVe is based on word co-occurrence in a corpus.

Mathematics

- **Word-Word Co-occurrence Matrix X:**

$$X_{ij} = \text{Number of times word } i \text{ co-occurs with word } j$$

- **Objective Function:**

$$J = \sum_{i,j} f(X_{ij}) (w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

where $f(X_{ij})$ is a weighting function.

Python Implementation

```
import gensim.downloader as api


# Load pre-trained GloVe embeddings
glove_model = api.load("glove-wiki-gigaword-50")
```

```
# Get word vector
print("Vector for 'deep':")
print(glove_model['deep'])

# Find most similar words
print("\nMost similar words to 'learning':")
print(glove_model.most_similar('learning'))
```

1.4 FastText

FastText extends Word2Vec by using subword information, making it better at handling rare words.

Mathematics

- **Word Representation:**

$$v_w = \sum_{g \in G_w} z_g$$

where G_w is the set of n-grams for word w, and z_g is the vector for each n-gram.

Python Implementation

```
from gensim.models import FastText

# Train FastText model
fasttext_model = FastText(sentences=tokenized_corpus, vector_size=100, window=3, min_count=1, workers=4)

# Get word vector
print("Vector for 'deep':")
print(fasttext_model.wv['deep'])

# Find most similar words
print("\nMost similar words to 'learning':")
print(fasttext_model.wv.most_similar('learning'))
```

---

2. Contextual Embeddings

Unlike static embeddings, contextual embeddings generate different vectors for a word depending on its context.

2.1 ELMo (Embeddings from Language Models)

ELMo generates word representations using deep bidirectional LSTMs.

Mathematics

- **ELMo Representation:**

$$\text{ELMo}_k = \gamma \sum_{j=0}^{L} s_j h_{j,k}$$

where $h_{j,k}$ is the hidden state of the LSTM at layer j for word k.

Python Implementation

```
import torch

from allennlp.modules.elmo import Elmo, batch_to_ids


options_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x4096_512_2048cnn_2xhighway/options.json"

weight_file = "https://allennlp.s3.amazonaws.com/models/elmo/2x4096_512_2048cnn_2xhighway/elmo_weights.hdf5"


elmo = Elmo(options_file, weight_file, num_output_representations=1)


# Sample sentences

sentences = [["I", "love", "NLP"], ["ELMo", "is", "powerful"]]


# Convert to character IDs

character_ids = batch_to_ids(sentences)


# Get ELMo embeddings

embeddings = elmo(character_ids)
```

```
print(embeddings['elmo_representations'][0].shape)
```

2.2 BERT (Bidirectional Encoder Representations from Transformers)

BERT is a transformer-based model that captures context from both left and right.

Mathematics

- **Masked Language Model (MLM) Loss:**

$$- \sum_{i \in M} \log P(w_i | w_{1:n} \setminus \{w_i\})$$

- **Next Sentence Prediction (NSP) Loss:**

$$- \log P(S_2 | S_1)$$

Python Implementation

```
from transformers import BertTokenizer, BertModel


# Load pre-trained BERT model

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

model = BertModel.from_pretrained('bert-base-uncased')


# Tokenize input

text = "I love learning NLP with BERT"

tokens = tokenizer(text, return_tensors='pt')


# Get embeddings

with torch.no_grad():

    outputs = model(**tokens)


# Print embeddings shape

print(outputs.last_hidden_state.shape)
```

Conclusion

Embedding Type	Contextual?	Key Feature
TF-IDF	No	Statistical measure
Word2Vec	No	Neural network-based word vectors
GloVe	No	Global word co-occurrence
FastText	No	Uses subwords (n-grams)
ELMo	Yes	Deep bidirectional LSTM
BERT	Yes	Transformer-based model