Special Proceedings: ISBN #: 978-81-950383-5-0
26th Annual Conference, 26-28 February 2024; pp 127-141

Text Representation: A Journey from Traditional Vector Space Model to LLM

Sharad Verma[1], **Pragati Bhatnagar**[2] **and Aditi Sharan**[3]
[1]*Department of Information Technology*
Rajkiya Engineering College Ambedkar Nagar,224122
[2]*Jain Vishwabharti University, ladnun, Rajasthan*
[3]*SC&SS, Jawaharlal Nehru University, New Delhi, 110067*

Received: 01 July 2024; Revised: 15 July 2024; Accepted: 17 July 2024

Abstract

In the era of language processing and artificial intelligence, people are amazed by the capabilities of ChatGPT. However, ChatGPT is not the end but the beginning of an era where much more is yet to happen. The backbone of ChatGPT is generative AI or large language models (LLMs). One of the most difficult challenges has been dealing with the semantics or meaning of language. This is what LLMs have been able to achieve to a certain extent: enabling computers to understand the semantics of text. However, LLMs are not the effort or product of a single person or community. They are the result of ongoing community efforts involving numerous scientists, researchers, and professionals who have worked for decades across the globe in various interdisciplinary fields, including statistics, computer science, and linguistics. Therefore, it is particularly important for the computer science and statistics communities to systematically understand the evolution of language models. This is the main objective of our paper. Our paper addresses the fundamental issue of incorporating the semantics of natural language text into its representation, which is the core of language models, including LLMs, and has revolutionized the field of natural language processing (NLP).

Key words: Attention based neural network; BERT; Deep learning based language embeddings; Large Language model; LSTM.

1. Introduction

Since the start of the digital processing of natural language text, text representation has been the most primitive requirement but, at the same time, the most complicated task. Some of the parameters are well known in the context of the difficulty in text representation viz: unstructured nature, ambiguity, how to represent meaning of text, high dimensionality, etc. However, another major issue evolved with the use of machine learning techniques for processing natural language text. As all the Machine learning algorithms work naturally

Corresponding Author: Dr. Aditi Sharan
Email: aditisharan@mail.jnu.ac.in

with the vector data, the vector representation of the text is the first step in dealing with the text data. Not only is it the first step, but it also impacts the text mining process and their outcomes from the machine learning algorithm, accuracy and the biases in the results. Though natural language processing has been an important research area for more than a decade, today, it has encroached on our day-to-day life, intentionally or unintentionally, in the form of web searches, recommendation systems, chatbots, etc. People are amazed by the potential of ChatGPT. However, along with the utility of ChatGPT in NLP, many issues are emerging regarding the accuracy, reliability, authenticity, and biases of the results. To be aware of these intricacies, one must know what is happening at the backend of ChatGPT as a specific case. Another term associated with the emergence of ChatGPT that became popular in the scientific and general community is Large Language Models(LLMs), more popularly termed generative AI models. These models form the core essence of the working of ChatGPT. Thus, it becomes important for the statistical and computer science community to understand the notion of LLMs. The core of LLM's efficiency is the efficient text representation and text generation. Here, we will focus on text representation. In fact, text generation can be considered as an outcome of text representation. The representation of text in the form we see today is not a sudden discovery, but actually, it is a success story of a long, tough but consistent effort of the NLP community, including computer scientists, mathematicians, statisticians and a lot of contributions from linguists. The objective of this paper is to present the evolution of transformer-based generative LLMs starting from one of the earliest tf-idf based traditional vector space models. The paper provides a systematic review of the important models focusing on the emergence of transformer-based models. It will address the basic issue of incorporating semantics of a natural language text in the representation itself, which is the core of deep learning algorithms, including LLMs and has revolutionized the field of NLP. Additionally, we discuss the unresolved issues and challenges in this field. We also present some of our efforts to address these challenges.

The paper is organized as follows: Section 2 introduces the traditional TF-IDF-based Vector Space Model. Section 3 delves into the Distributional Hypothesis, laying the groundwork for the subsequent discussion on machine learning techniques for text representation in Section 4. Section 5 presents deep learning-based language modeling, and Section 6 concludes the paper.

2. Traditional TF-IDF based Vector Space Model (VSM)

One of the earliest attempts to represent a text document by a vector emerged in the form of the Classical Vector Space Model (VSM)Salton *et al.* (1975). Though not very efficient for representing the text, it is one of the simplest and computationally efficient models. Also one should keep in mind that it presents one of the earliest attempts to represent a text by a vector. The model is based on the bag of words (BOW) approach. To understand the BOW-based VSM model, let us consider that we have a corpus of text documents and the objective is to represent each text document by a vector. The initial step involves preprocessing of the text, mainly involving stop word removal and stemming to remove irrelevant/nonessential words. After preprocessing, the outcome results in a collection of words that form the vocabulary of the text corpus. A text document matrix (TDM) can then be constructed with $M * N$ size, where rows represent documents, M being number of documents and columns represent terms, N being the size of the vocabulary. Further, each element w_{ij} of TDM represents the weight of j_{th} term in i_{th} document. Thus each row of the matrix represents the vector corresponding to the document, the size of the vector being the vocabulary size. Ideally, the weight should represent the importance of a term in the document. But importance itself is a subjective term and may depend on the task to be performed. However, some objective criteria is required. Various Now weights can be assigned in different ways, however, tf-idf (term frequency-inverse document frequency) based weighting emerged as the most popular technique for traditional VSM based model.

2.1. Weighting using TF-IDF (Term Frequency-Inverse Document Frequency)

The words in VSM need to be given weight so as to reflect their importance in the document. Documents with higher weights are more important. The most obvious way of giving weights can be related to the frequency of words in the document. The weight may correspond to frequency count of words in the document. Table 2 represents the matrix using term frequency count for the same example as presented for binary vector. The raw frequency may not be a statistically stable value, so a normalized measure might be used.However, it can easily be observed that frequency of the word, though important, may not be the only measure for assigning the weight to a word. The simplest example to understand this is that stopwords are most frequent but are least important. This leads to the notion of TF-IDF based measure in VSM. It is a statistical measure that reflects how important a word is to a document in a collection or corpus. TF-IDF consists of two main components: Term frequency (TF) and Inverse document frequency (IDF). Term frequency (TF) measures the number of times a term (word) is present in a document. Instead of raw value , the value may be normalized. Inverse document frequency (IDF) measures how rare a term is across the corpus. Some variants of IDF exist, as far as it is inversely proportional to the no. of documents in which it is appearing, thus giving higher weightage to rare terms. One way of calculating IDF is as follows: For each term t in the corpus:

$$IDF(t) = log(N/(df(t))) \tag{1}$$

where N: Total number of documents in the corpus. $df(t)$: Number of documents in the corpus that contain the term t.

TF is a local measure (calculated for each document), whereas IDF is a global measure (calculated for the entire corpus). TF-IDF is then calculated as the product of TF and IDF. The resulting value represents the importance of the term in the document and the corpus as a whole. High TF-IDF values indicate that a term is important to a document and the corpus, while low values indicate that the term is less important or common. Also frequently stated as frequent and rare terms are more important. TF-IDF is often used for text classification, information retrieval, and content-based recommendation systems. It is a popular technique because it is simple, efficient, and effective in identifying important terms in a document or corpus. The TF-IDF score can be calculated by the formula:

$$TF - IDF(t, d) = TF(t, d) * IDF(t) \tag{2}$$

where $TF(t, d)$ represents the frequency of term t in document d.

Let us take an example of three Documents for a better understanding:-

Table 1: Sentences from medical domain

Document	Sentence
Document 1	Chemotherapy is used to treat cancer.
Document 2	Cancer cells can grow uncontrollably.
Document 3	Radiation therapy damages cancer cell

After some preprocessing (stopword removal and case conversion), the vocabulary of the corpus in this case may be represented as [cancer, therapy, cells, chemotherapy, radiation, grow, uncontrollably, damages]

Table 2 contains TF-IDF Scores of each term in the document.

Table 2: TF-IDF Scores

Term	Document 1	Document 2	Document 3
cancer	0	0	0
therapy	0.4055	0	0.4055
cells	0	0.4055	0.4055
chemotherapy	1.0986	0	0
radiation	0	0	1.0986
grow	0	1.0986	0
uncontrollably	0	1.0986	0
damages	0	0	1.0986

It can be observed that medical domain terms present in the document are important.

Traditional VSM is a sort of one-hot encoding where each word has its own space and index. One hot representation has a lot of limitations and problems. Some of the important problems with one hot encoding are as follows :

- Viewing all the words as discrete units is not ideal.
- High dimensionality problem, since there can be hundreds of thousands of words in a given language, representing and storing words as one-hots can be extremely expensive.
- Sparsity Problem.
- All sequencing information is lost.
- Lack of an inherent similarity notion. a simple way of measuring the similarity between two vectors is using the cosine similarity. But since the one-hot vectors of any two different words are necessarily orthogonal, taking the dot product of even two synonyms would yield a similarity score of 0.
- Can not deal with contextual similarity between sentences.

The Distributional Hypothesis addresses some of these limitations by positing that words with similar distributions in a large corpus are likely to have similar meanings. By leveraging statistical patterns in language usage, the Distributional Hypothesis allows VSM to capture semantic similarities more effectively across varied contexts and improve the representation of word meanings beyond mere co-occurrence.

3. Distributional hypothesis

Now there has been much talk about incorporating semantics in text representation. This tends to consider meaning of words and the notion of synonyms in the text representation. It brings into the picture the notion of dictionary, thesaurus, ontology, *etc* . All these resources are beneficial for understanding the meaning of the text but fail to provide a computational model for representing the meaning of the text. Thus came the idea of the Distributional hypothesis, the same idea repeated in different ways by various researchers Harris (1954). A very old and popular idea in the Linguistic domain – You shall know a word by the company it keeps Firth (1957). In particular in the modern NLP context – A word is defined by its environment (the context words around it). But this is again based on the sound foundation of linguists Harris (1954): If A and B have almost identical environments we say that they are synonyms. For a long time, computational linguists have been focusing on the representation of the context of a word that can assist in incorporating the semantics of the word in the representation itself. The base of this again lies in the hypothesis that words with high similarity (such as synonyms) occur in the same context.

3.1. Representing words using co-occurrence statistics

In the previous section, we saw an example of representing text as a vector but in many applications, we are interested in finding an appropriate representation of a word as a vector. One of the most primitive ways is to consider each column of the TF-IDF matrix as a word vector. Thus words can be represented as vectors in document dimension. Here two word vectors are similar if they share common documents. This word vector captures the information of words based on their presence in documents that are not very meaningful. There are no word-to-word associations. Based on the distributional hypothesis it was thought to represent word vectors in a way that might capture this association. One of the earliest attempt is reflected in the form of mutual information-based association. Mutual information (MI) is a measure of how often two events x and y occur, compared with what we would expect if they were independent. The standard formula is

$$I(x, y) = log_2 \frac{P(x, y)}{P(x)P(y)} \tag{3}$$

Based on MI an important measure in NLP is Pointwize MI (PMI). PMI measures the the chances of co-occurrence of two words PMI between two words (we we may call them word w and context word c) w and c can be given by

$$PMI(w, c) = log_2 \frac{P(w, c)}{P(w)P(c)} \tag{4}$$

However, PMI value may range from positive to negative infinity. Negative information may create misinformation in many cases. Thus a refined measure is called as Positive PMI(PPMI)

$$PPMI(w,c) = max(log_2 \frac{P(w,c)}{P(w)P(c)}, 0) \tag{5}$$

Let us try to understand its practical application. Let us consider some words and selective context words. The words considered are computer, data, result, pie, and sugar. The context words are cherry, strawberry, digital, and information. We can see the co-occurrence of the context words in a large corpus of the order of say Wikipedia. For determining the co-occurrence of context words, we need to fix the window size say 4, then observe the count of context words in the neighborhood of all occurrences of a word considering 4 neighbors on the left and 4 on the right. Assume that it results in the following matrix between word and context.

Table 3: Co-occurence statistics

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	344
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

The resultant PPMI comes out to be :

Table 4: PPMI scores

computer	data	result	pie	sugar	count(w)
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Now a word is represented as a vector in word dimension. This was a small example. If all words in the vocabulary are included. We can have a $n * n$ matrix where n is size of the vocabulary. Thus each word is a vector in the dimension of vocabulary. One can make out an interesting observation. One can find the cosine similarity between the words and you can observe from the table that cherry and strawberry are more similar and similarly digital and information. For other pairs, the similarity is zero. The obvious observation is that digital and information share computer and data, while cherry and strawberry share pie and sugar. Thus PPMI based captures the co-occurrence information in the word vector representation that is quite meaningful. However, this matrix is again sparse as in the case of TF-IDF based measure. Also, it does not involve any learning. With the advancement in machine learning approaches for NLP, now machine learning is being used for text representation, text processing, text mining, and finally text generation. In the next part, our focus is on machine learning based vector representation of text, frequently called word embedding and text embedding.

4. Machine Learning for text representation

The previous section presented Count based context vector creation. Count based context representation does not involve any learning so has a limited usage in era of machine learning , where everything can be learnt and thus can be predicted. Prediction based Context word model can predict the co-occurrence probabilities , thus they can predict the context words corresponding to a given word. While we are learning to predict context words, the sole objective is not only context word prediction. In fact such a trained model leads to the notion of representational learning, where the vectors representing the words can be learnt through neural network based models.

4.1. Word2Vec

Word2Vec is a groundbreaking model introduced by Tomas Mikolov Mikolov *et al.* (2013a,b) and his team at Google in 2013. It revolutionized the field of natural language processing by enabling the creation of dense vector representations of words in a continuous vector space. This model uses a shallow, two-layer neural network to process vast amounts of text data and learn the relationships between words based on their context. There are two main architectures used in Word2Vec: Continuous Bag of Words (CBOW) and Skip-gram. CBOW predicts a target word from its surrounding context words, while Skip-gram does the reverse, predicting the context words given a target word. These approaches allow Word2Vec to capture semantic relationships between words effectively, such that words with similar meanings are positioned close to each other in the vector space.

One of the most compelling features of Word2Vec is its ability to capture linear relationships between words. For example, the model can understand analogies like “King” is to “queen” as “man” is to “woman” by performing vector arithmetic: vector(“king”) - vector(“man”) + vector(“woman”) $\approx$ vector(“queen”). This ability stems from the model’s training process, which uses a technique called negative sampling to efficiently differentiate relevant context words from irrelevant ones. The learned vectors from Word2Vec have been widely used in various applications, including machine translation, text classification, and sentiment analysis, due to their effectiveness in encoding semantic meanings and relationships in a computationally efficient manner.

4.2. GloVe

Global Vectors for Word Representation(GloVe) Pennington *et al.* (2014) is a powerful word embedding technique developed by researchers at Stanford University. Unlike traditional methods that solely rely on local context (like Word2Vec), GloVe combines the benefits of both global matrix factorization and local context window methods. It constructs word vectors by aggregating global word-word co-occurrence statistics from a large corpus. Specifically, it utilizes a co-occurrence matrix where the entries represent the frequency of word pairs appearing together within a certain window. By factorizing this matrix, GloVe generates dense vector representations where the semantic relationships between words are captured. For instance, the difference between the vectors for “king” and “queen” is similar to the difference between “man” and “woman”, reflecting meaningful linear substructures. These pre-trained embeddings have been extensively used in various natural language processing tasks due to their ability to capture both syntactic and semantic word relationships effectively.

Since this method captures the global statistics in the corpus, it is named Global Vectors (in short GloVe). These methods have a very good performance on various tasks like word analogy, word-similarity, and named entity recognition.

5. Deep Learning based Language modeling

In addition to many problems associated with the computational processing of natural language, another major issue is the temporal nature of language that is reflected in language flow, continuity, etc. Thus a language is sometimes considered as a sequence that unfolds in time. Most of the deep learning models perform the task of language modeling. At the most primitive level, language modelling involves the prediction of the next word in the sequence. How accurately the model predicts the word forms the base for the performance of the language model. It seems a well-defined task, but it is too complicated, as the correct prediction of words can only be done if the text is understood properly. However, there is no model for the representation of meaning. This actually leads to modeling the distributional hypothesis, the well-known hypothesis given by linguists. These models try to capture/learn the context vectors of the words. Based on the context vector, the model tries to predict the next word in sequence. But the question may come to mind: What would be so great if we were able to predict the next word? The answer lies in the statement that correct prediction is not possible without an understanding of the text(for which there is no explicit mechanism). Thus, correct prediction involves some understanding of the text. In other words, deep learning models are able to capture the semantics of text using the notion of statistics and probabilities. With time, various deep learning models have evolved, but in terms of architecture, we have three categories: Sequential model, encoder decoder-based architecture, and transformer-based model.

5.1. Sequential Models: The Era of RNN and LSTM

The era of sequential models in artificial intelligence has been significantly shaped by recurrent neural networks (RNNs) Elman (1990) and their refined variant, Long Short-Term Memory networks (LSTMs). These models excel in processing sequential data, such as time series, text, and speech, by maintaining an internal state that evolves as new inputs are processed. RNNs were among the first neural architectures capable of capturing temporal dependencies, making them pivotal in tasks like speech recognition, language modeling, and machine translation. However, traditional RNNs are prone to the vanishing and exploding gradient problems, hindering their ability to learn long-term dependencies effectively. The introduction of LSTMs by Hochreiter and Schmidhuber in 1997 addressed these challenges by incorporating memory cells and gating mechanisms that regulate information flow, allowing them to remember information over long sequences and selectively forget irrelevant details. This innovation marked a breakthrough in sequential modeling, enabling more robust learning and improved performance in tasks requiring nuanced understanding of context and continuity over time. Despite their successes, both RNNs and LSTMs have limitations in handling very long sequences due to computational constraints and struggles with capturing hierarchical dependencies. As the field advances, newer architectures like transformers, which rely on attention mechanisms, have emerged to address these shortcomings and push the boundaries of sequential modeling in modern AI applications.

5.2. Limitation and constraints of LSTM

Long Short-Term Memory (LSTM) Hochreiter and Schmidhuber (1997)networks are a type of recurrent neural network (RNN) specifically designed to overcome the limitations of traditional RNNs, such as the vanishing gradient problem, by introducing a more sophisticated memory architecture. The LSTM unit consists of a cell state C_t and three gates that regulate the flow of information: the input gate i_t, the forget gate f_t, and the output gate o_t.The equations governing these gates are as follows:

$$\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
C_t^h &= \tanh(W_C[h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * C_t^h \\
h_t &= o_t * tanh(C_t)
\end{aligned} \tag{6}$$

Here, x_t represents the input at time step t, h_t is the hidden state, W and b denote the weights and biases for the respective gates, σ is the sigmoid activation function, and $*$ indicates element-wise multiplication. The forget gate f_t determines which information from the previous cell state C_{t-1} should be discarded, the input gate i_t decides which new information should be added to the cell state, and the output gate o_t controls what part of the cell state is output as the hidden state h_t. The combination of these gates allows LSTMs to maintain and update the cell state over long sequences, making them highly effective for tasks that require learning from temporal patterns, such as natural language processing, speech recognition, and time-series prediction. Long Short-term Memory (LSTM) networks, despite their advancements in handling sequential data, have several limitations. Firstly, they require substantial computational resources and time for training due to their complex architecture and numerous parameters, which can be a bottleneck for large datasets. Additionally, LSTMs can struggle with very long sequences, where even their memory cells might not effectively capture dependencies over extremely long periods, potentially leading to gradient vanishing or explosion issues. Moreover, fine-tuning LSTM models requires considerable expertise, as the process involves balancing many hyperparameters, such as learning rates and the number of layers. They also tend to overfit if not regularized properly. Lastly, LSTMs are less interpretable compared to simpler models, making it challenging to understand and trust their decision-making processes, which is crucial in applications where model transparency is essential.

Vanishing and exploding gradients Bengio *et al.* (1994) pose significant challenges in training RNNs, particularly due to their deep sequential nature. In standard RNNs, the vanishing gradient problem arises because gradients can diminish exponentially over time steps, especially in long sequences, making it difficult for the model to learn dependencies over distant time steps. Conversely, exploding gradients can occur due to unstable weight updates, leading to numerical instability during training. Both problems can severely impact the ability of neural networks to learn and generalize from data effectively. Addressing these issues often involves careful initialization of parameters, using activation functions that mitigate gradient saturation, and employing techniques like gradient clipping to stabilize training dynamics.

While LSTM networks have been instrumental in addressing the vanishing gradient problem in RNNs, they are not without their shortcomings. One notable limitation is their computational complexity, which arises from the multiple gates and memory cells that need to be managed per unit. This complexity can lead to slower training times and increased memory requirements, making LSTMs less scalable for very large datasets or when deploying models in resource-constrained environments. Additionally, LSTMs may struggle with capturing fine-grained dependencies within sequences, as their gating mechanisms are designed to capture long-term dependencies rather than focusing on specific, short-term relationships.

The emergence of attention mechanisms represents a significant advancement in addressing these shortcomings. Attention networks, such as the Transformer model, allow neural networks to dynamically focus on different parts of the input sequence. Unlike LSTMs, attention mechanisms do not impose a fixed-length context window and can adaptively attend to relevant parts of the input sequence. This flexibility enables attention networks to capture both short-term and long-term dependencies more effectively without the computational overhead of managing complex gating mechanisms. Moreover, attention mechanisms have shown superior performance in tasks like machine translation, where aligning and translating words or phrases across languages require capturing nuanced relationships within and between sentences.

5.3. Attention

An attention network, often referred to simply as an attention mechanism is a component of neural networks designed to dynamically focus on specific parts of the input data when making predictions. Attention networks were developed to address some of the limitations of LSTMs, particularly their difficulty in capturing long-range dependencies and their inefficiency with very long sequences. The attention mechanism allows the model to weigh the importance of different input elements, enabling it to handle long-range dependencies and improve performance on tasks such as machine translation, image captioning, and more.

The primary idea of attention is to assign different weights to different parts of the input sequence. When making a prediction, the model can then focus more on the relevant parts and less on the irrelevant ones. This selective focus helps capture relationships and dependencies that span long distances in the input data.

Several types of attention mechanisms are designed to address specific needs or improve computational efficiency. Here's an overview of the most commonly used attention mechanisms:

Additive (Bahdanau) Attention: Bahdanau attentionBahdanau *et al.* (2014), also known as additive attention, was introduced by Dzmitry Bahdanau and colleagues in their 2014 paper to improve the performance of sequence-to-sequence (seq2seq) models, particularly for machine translation. The main goal of Bahdanau's attention is to allow the model to focus on different parts of the input sequence dynamically when generating each part of the output sequence. It combines the decoder hidden state and the encoder hidden states using a trainable weight matrix and a non-linear activation function (typically tanh).

$$e_{t,i} = v^T \tanh(W_s s_t + W_h h_i) \tag{7}$$

where W_s and W_h are weight matrices, v is a weight vector, s_t is the decoder hidden state at time t, and h_i is the encoder hidden state at time i.

Pros: Flexible and can capture complex relationships between the encoder and decoder states.

Cons: Computationally expensive due to the non-linear transformation.

Multiplicative (Luong) Attention: Proposed by Luong et al.Luong *et al.* (2015), this mechanism computes attention scores using a dot product (multiplicative) approach, which is computationally more efficient than additive attention.

$$e_{t,i} = s_t^T h_i \tag{8}$$

Pros: More efficient than additive attention.

Cons: May not perform as well as additive attention when the dimensions of s_i and h_i differ significantly.

Self Attention: Self-attention is a mechanism used in various neural network architectures, particularly in transformers Vaswani *et al.* (2017), to enable models to focus on different parts of the input sequence when processing each token. This mechanism allows the model to capture dependencies regardless of their distance in the sequence, making it highly effective for tasks like language modeling and machine translation.

In self-attention, each token in the input sequence is transformed into three vectors: Query (Q), Key (K), and Value (V), as shown in fig 1. These vectors are derived through learned linear transformations. Assume we have an input sequence of length n, represented by the matrix $X \epsilon R^{n*d}$, where d is the dimensionality of the input embeddings. The input matrix X is multiplied by three weight matrices to produce the Query, Key, and Value matrices as follows:

$$\begin{aligned} Q &= XW^Q, \\ K &= XW^K, \\ V &= XW^V \end{aligned} \tag{9}$$

where $W^Q, W^K, W^V \epsilon R^{d*d_k}$ are weight matrices, and d_k is the dimensionality of Query, Key, and Value vectors.

The core of the self-attention mechanism involves computing a score for each pair of tokens in the sequence to determine how much focus one token should have on another. This is done using the Query and Key matrices. The score for each pair is calculated as the dot

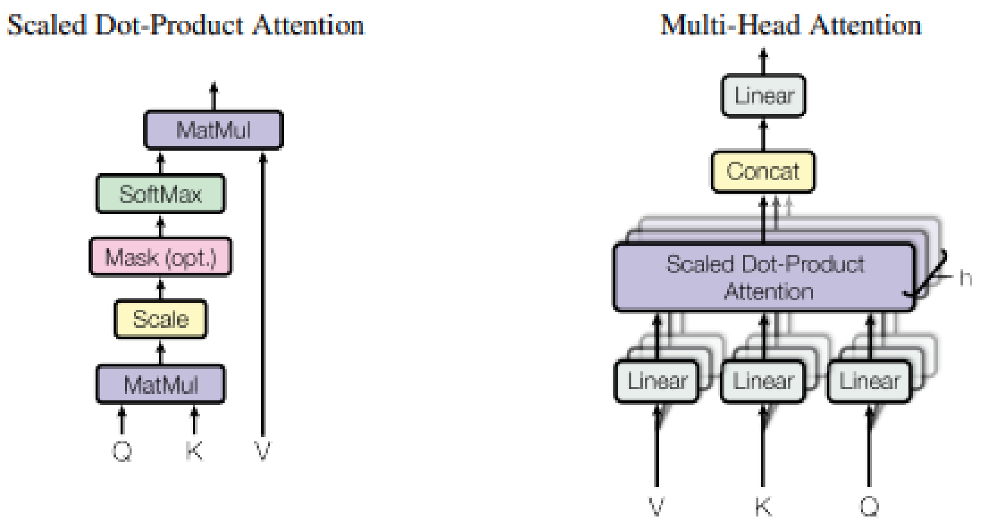

Figure 1: Scaled Dot Product and Multi-head Attention

product of their Query and Key vectors, scaled by the square root of d_k to stabilize gradients as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{10}$$

It is also known as scaled dot-product attention. The result of the self-attention mechanism is a new representation of the input sequence, where each token now includes information from all other tokens, weighted by their relevance. This process can be repeated in multiple layers to capture increasingly complex dependencies.

To allow the model to focus on different parts of the sequence simultaneously, the self-attention mechanism is often extended to multi-head attention. This involves using multiple sets of Query, Key, and Value weight matrices:

$$MultiHead(Q, K, V) = Concat(head_1, head_2..., head_h)W^O \tag{11}$$

Each head independently performs the self-attention operation, and the results are concatenated and linearly transformed using the weight matrix W^O.

The Transformer architecture has revolutionized the field of natural language processing by enabling more efficient and effective processing of sequential data, notable for its innovative encoder-decoder structure. This architecture is designed to handle sequence-to-sequence tasks such as translation, summarization, and question-answering with unparalleled efficiency and performance. The encoder is composed of multiple identical layers, each with two key sub-layers: multi-head self-attention mechanisms and position-wise feed-forward networks. The self-attention mechanism allows the encoder to weigh the importance of different words in a sentence relative to each other, capturing complex dependencies and relationships. Each encoder layer also includes residual connections and layer normalization, ensuring stability and improving gradient flow during training. The decoder mirrors the encoder but includes an additional sub-layer for masked multi-head self-attention, which ensures that predictions for a given word depend only on previous words in the sequence, maintaining causality. The decoder also integrates multi-head attention over the encoder's output, allowing it to focus on relevant parts of the input sequence when generating each word of the output. This dual structure of the Transformer, with its powerful self-attention mechanisms and ability to process input and output sequences in parallel, represents a significant advancement over traditional recurrent models, offering superior scalability and the ability to capture long-range dependencies more effectively.

This architecture is highly scalable and can be parallelized, leading to faster training times and improved performance on large datasets. The Transformer has become the foundation for many state-of-the-art models, including BERT, GPT, and T5, driving significant advancements in tasks such as machine translation, text generation, and sentiment analysis.

5.4. BERT

Bidirectional Encoder Representations from Transformers(BERT) embeddings Devlin *et al.* (2018) have revolutionized natural language processing by providing deep, contextualized representations of words. Unlike traditional embeddings that generate a fixed vector for each word regardless of context, BERT dynamically produces word vectors based on the entire sentence, capturing intricate details of the language. BERT's pre-training involves two key subtasks: the masked language model (MLM) and next sentence prediction (NSP).

In MLM, a portion of the input tokens is randomly masked, and the model is trained to predict these masked tokens based on the surrounding context. For example, in the sentence "The quick brown fox jumps over the lazy [MASK]," BERT attempts to predict the masked word "dog" using the context provided by the rest of the sentence. This task forces BERT to develop a bidirectional understanding of language, considering both the left and right contexts of each word.

NSP is designed to train BERT to understand the relationship between sentences. During pre-training, BERT receives pairs of sentences. Some pairs are actual consecutive sentences from the corpus, while others are random pairs. The model learns to classify whether the second sentence logically follows the first. For instance, given the sentence pair *The man went to the store. He bought a gallon of milk*, BERT should identify this as a logical sequence. Conversely, for the pair *The man went to the store. Penguins are great swimmers*, BERT should recognize this as a random pairing.

By combining these two subtasks, BERT achieves a robust understanding of language context and sentence relationships. MLM helps BERT learn to predict words based on context, enhancing its ability to generate accurate word embeddings in various contexts. NSP, on the other hand, improves BERT's grasp of coherence and logical flow between sentences, which is essential for tasks requiring sentence-level comprehension, such as text summarization and question answering.

These pre-training tasks enable BERT to produce embeddings that are highly effective for a wide range of natural language processing tasks, leading to state-of-the-art performance in many benchmarks.

6. Applications of Text Mining

The advancements in text mining and deep learning have given rise to Transformer-based models with numerous real-time applications. These applications are transforming our daily lives and benefiting society. Transformer models, such as BERT and GPT, utilize self-attention mechanisms to efficiently manage dependencies across long sequences, significantly enhancing performance in various NLP tasks. The encoder-decoder architecture in transformers, seen in models like T5 and BART, uses an encoder to process input sequences and a decoder to generate outputs, enabling tasks like translation and summarization. This architecture allows for parallel processing, significantly accelerating training and inference.

In translation tasks, these models go beyond sequential translation, encoding based on the semantics of the source language text and the context of both source and target languages. This approach allows for the creation of multilingual translation models using the same architecture with ample examples of translations. Additionally, Transformer models have numerous applications in the public health domain, including drug recommendation, drug design, health chatbots, personalized health recommendations, and telemedicine.

7. Conclusion

This paper has traced the development of text representation methodologies from the foundational Vector Space Model to advanced Attention-based architectures. Beginning with the Vector Space Model, which utilized TF-IDF to numerically represent text, we observed its limitations in capturing contextual semantics. The Distributional Hypothesis provided a theoretical basis for more nuanced vector representations like Word2Vec, significantly enhancing our ability to capture word meanings based on context. RNNs marked a significant advancement in processing sequential data but were constrained by issues like vanishing gradients, which were effectively addressed by LSTM networks. LSTMs improved the handling of long-term dependencies, making them vital for various NLP tasks. The introduction of the Attention mechanism and the subsequent Transformer architecture revolutionized text representation by allowing models to selectively focus on different parts of the input, capturing complex dependencies with unprecedented accuracy. This evolution highlights the remarkable strides made in text representation, culminating in sophisticated models that continue to push the boundaries of natural language processing.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, **5**(2), 157–166.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, **14**(2), 179–211.

Firth, J. (1957). A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*, 10–32.

Harris, Z. S. (1954). Distributional structure. *Word*, **10**(2-3), 146–162.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, **9**(8), 1735–1780.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, **26**.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, **18**, 613–620.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, **30**.