# Transformative Advances in NLP and AI: Charting the Evolution of Technology

Paul Gabriel TEODORESCU[1], Silvia OVREIU[1,2]
[1]National Institute for Research & Development in Informatics - ICI Bucharest, Romania
[2]University Polithenica of Bucharest, Romania
paul.teodorescu@ici.ro, silvia.ovreiu@ici.ro

*This paper attempts to enter the world of NLP (human language processing) from the three perspectives of physics, mathematics and computer science. The article explains why science has chosen word-vectors and word vectorization in NLP and describes the 2 models that have established themselves in this world of words: Word2Vec and GloVe. After having a clear picture of how artificial intelligence deals with words and human language processing, the topics of Time and Attention are treated in the new approach of Google which has already moved to another paradigm in word processing: the BERT models, transformers and Attention mechanisms. This answers the questions why Time and temporal recurrence have been abandoned in favor of models with transformers and Attention mechanisms. The paper also includes an explanation of the complex processes that take place inside the Transformer for a simple translation from one language to another.*
***Keywords:*** *Human language processing, Artificial Intelligence, Transformers, Attention*
**DOI:** 10.24818/issn14531305/28.4.2024.02

## 1 Introduction

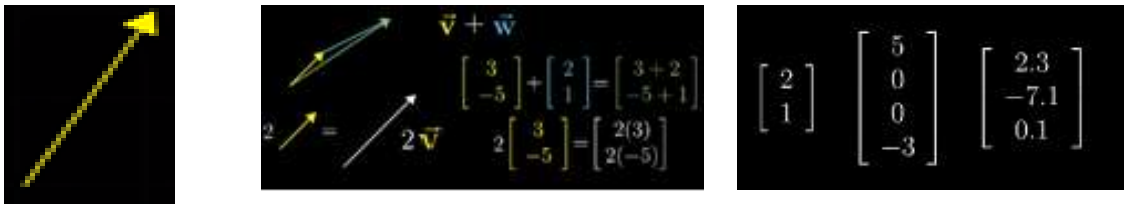
In human language processing, science has chosen the vectorization of words to resolve relationships between words. The meanings of a word could be represented by real numbers, numbers which are part of the list of values that represent a vector. The beauty of representing words as vectors lies in the possibility of doing mathematical operations with words. With vector words, a huge technological leap has been made towards speech recognition and machine translation. Moreover, it was necessary to interpret the vector transformations mathematically. This was done with the help of matrices: a matrix represents a given transformation of vectors and hence transformation of space. The semantic similarity of words (semantic similarity or contextual similarity of words) was also solved with vectors, which is nothing but the degree of similarity between two vectors in a multidimensional space.

In order to capture the semantic meaning of words in a mathematical way and to transform words into a form that computers can understand and manipulate, various techniques have been used. The present paper discusses these techniques - Word2Vec and GloVe - and then explains the transition to the new paradigm in NLP that gives up TIMP and RNNs ((Recurrent Neural Networks) by using Transformers and the Attention mechanism. The new motto in NLP becomes "Attention is all you need". The evolution of models has nowadays reached the BERT model developed by Google which allows the model to better capture semantic relationships and dependencies between words.

## 2 Vector representation of words in three perspectives

The vector underlies linear algebra. There are 3 distinct but related ideas about vectors as seen from 3 perspectives: physics, mathematics and computer science.

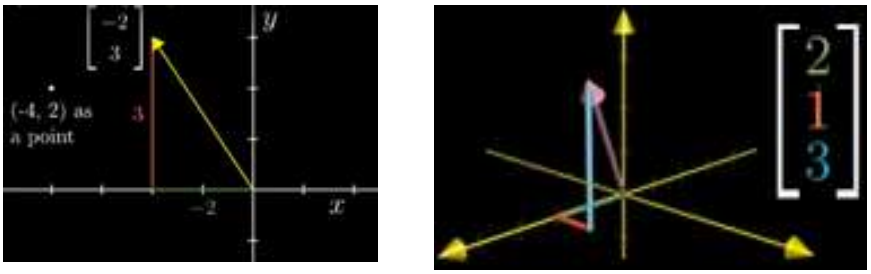
From the physics perspective (Figure 1, a): the vector is an arrow in space, defined by its length and the direction in which it points. Vectors in a 2-dimensional space are vectors of 2 dimensions, vectors in a 3-dimensional space are vectors of 3 dimensions. In a 2 dimensional plane, the first example of a vector space consists of arrows in a fixed plane, starting at one fixed point. This is used in physics to describe forces or velocities.




a) Physics perspective b) Mathematical perspective c) Computer science perspective

**Fig. 1.** Vector representation in 3 perspective

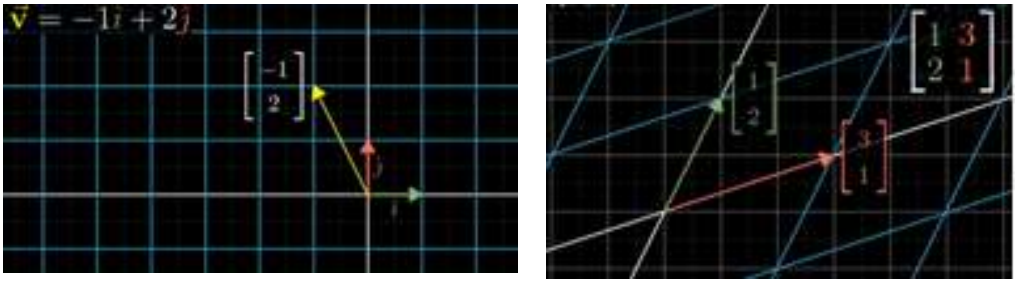
From a mathematical perspective (Figure 1, b): generalizing, vectors can be added or a vector can be multiplied by a number. From the geometric perspective, a vector is an arrow in an Oxy coordinate system, for example an arrow with its end at the very origin of the axes O. Physicists consider this vector anywhere in this space, but in linear algebra the vector has its end at the origin of the axes. Thus, the vector can be translated as a list-of-numbers realized by considering the coordinates of the vector. To differentiate a point in space from a vector in space, round brackets are used for points and square brackets for vectors (Figure 2, a). When referring to the dimensional plane, each pair of numbers means one vector and one vector only. A vector has one and only one pair of numbers associated with it. Extrapolating to 3-dimensional space, there are 3 axes, Ox, Oy, Oz (Figure 2, b). A vector space can have many bases (even an infinity of bases), the dimension of the vector space is given by the number of vectors of the base (Figure 3, a).




a) In 2-dimensional space b) In 3-dimensional space

**Fig. 2**. Geometric representation

From a computer science perspective (Figure 1, c): vector is an ordered list of numbers. Matrices describe the transformations of vectors: whenever a matrix is encountered, it can be interpreted as a certain transformation of the space. These transformations can only be described with numbers, which are the coordinates where each "basis vector" (an element of a set of vectors that forms a basis for a vector space), arrives after the space transformation (Figure 3, b). The columns of the matrix (which are linearly dependent) represent the coordinates and the *matrix* × *vector* multiplication is just a way to compute what the new vector has become after the space transformation (LTM, 2023). So, matrices can be used to change the basis of a vector space. If A is a matrix whose columns are the new basis vectors, multiplying A by a coordinate vector x expresses the same x but in the new basis.




a) In 2-dimensional space b) In 3-dimensional space

**Fig. 3**. Linear transformation with spatial transformation

After establishing the three perspectives on vectors and examining their transformations through matrices, we will present how to compute the similarity between two vectors in space. Two methods are known, using the Euclidean distance and cosine similarity. For example, if we consider vectors A = (3, 8, 7, 5, 2, 9); B = (10, 8, 6, 6, 6, 4, 5), then we calculate the *similarity* (A, B), i.e. the degree of similarity of the two non-zero vectors in the vector space, using the relation:

$$similarity(A,B) = \frac{3\cdot10+8\cdot8+7\cdot6+5\cdot6+2\cdot4+9\cdot5}{\sqrt{3^2+8^2+7^2+5^2+2^2+9^2}\ \times\sqrt{10^2+8^2+6^2+6^2+4^2+5^2}} = \frac{219}{15.2315 \times 16.6433} = 0.8639$$

Since cos 0 = 1 and cos 90 = 0, this means that: two vectors with a cosine similarity of 1 have the same orientation; two perpendicular vectors have a similarity of 0.

**3 Word-Vector mapping**

Words-vector are simple vectors [1]. Vectors are translated by a list of real values/numbers where each value expresses one dimension of the meaning (or sense) of that word, so semantically similar words have similar vectors. Words that are used in a similar context will be mapped next to each other in the vector space. This opened the way to the possibility of doing mathematical operations with words. The numbers in the word-vector represent the weights of that word for the senses/meanings of that word, and each meaning of the word represents a dimension of the word: the higher the weight, the more that word is associated with that meaning/sense. This is how the semantics of the word is "embedded across the dimensions of the vector". Hence the notion of "*word embedding"*.

Word embedding aims at creating a vector representation with the lowest spatial dimensionality. This is what is called "word vectorization" resulting in word vectors. Word vectorization is also used for semantic analysis, to extract a meaning, a sense from the text which is in fact the expression of natural language. A language model capable of predicting the meaning of a text must consider the contextual similarity of words [2]. For example, words that define fruits (apples, oranges) are found in sentences where they are picked, grown, eaten, etc. but will not be in proximity to the word "airplane". The vectors created by word-vector mapping preserve these similarities so we can say that word embedding is the construction of a vector representation from a corpus of text that preserves the contextual similarity of words. The problem of semantics or semantic similarity has, as its central idea, the hypothesis (called the distributional semantics hypothesis) that linguistic elements with the same distribution, in the vector space, have the same meaning (similar distribution means similar meaning).

A surprising property of word vectors is that word analogies are solved by vector arithmetic. The "difference" relation between vectors is interesting because it allows us to discover relations that show an analogy, a similarity: the similar direction indicates such a relation of two pairs of vectors.

## 4 Word-vector construction algorithms

The most popular word-vector mapping construction algorithms are Word2Vec and GloVe.

Word2Vec is the word-vector mapping construction algorithm, a statistical method in which a neural network learns to guess a word from a text [3]. Input words are passed to the network as a binary vector. The binary vectors will then pass through a hidden layer of cells, then into an output layer called the *softmax layer* (since it uses the Softmax activation function) to make a prediction [4]. The weight matrix is trained on the hidden layer to find an efficient representation of the words. This weight matrix is usually called the *embedding matrix* and can be viewed as a look-up table: For example, having a vocabulary of 10000 words, where each word in the vocabulary is represented by a 300-dimensional vector, the weight matrix is 10000 × 300. The input words in the matrix enter as a binary vector in the one-hot encoded representation. One-hot encoding is a type of binary vector representation where each word in the vocabulary is represented by a vector that has the same length as the size of the vocabulary. In this vector, all values are 0 except for a single 1 at the index corresponding to that word. For example, for a vocabulary of 10,000 words, the word at index 5 might be represented as "`[0, 0, 0, 0, 1, 0, 0, ..., 0]`". In Figure 4 [5], as another example, after training a vocabulary of 10,000 words, the word "ant" is represented as 1 in a specific position. The network is going to tell us the probability for every word in our vocabulary of being the nearby word that we choose.

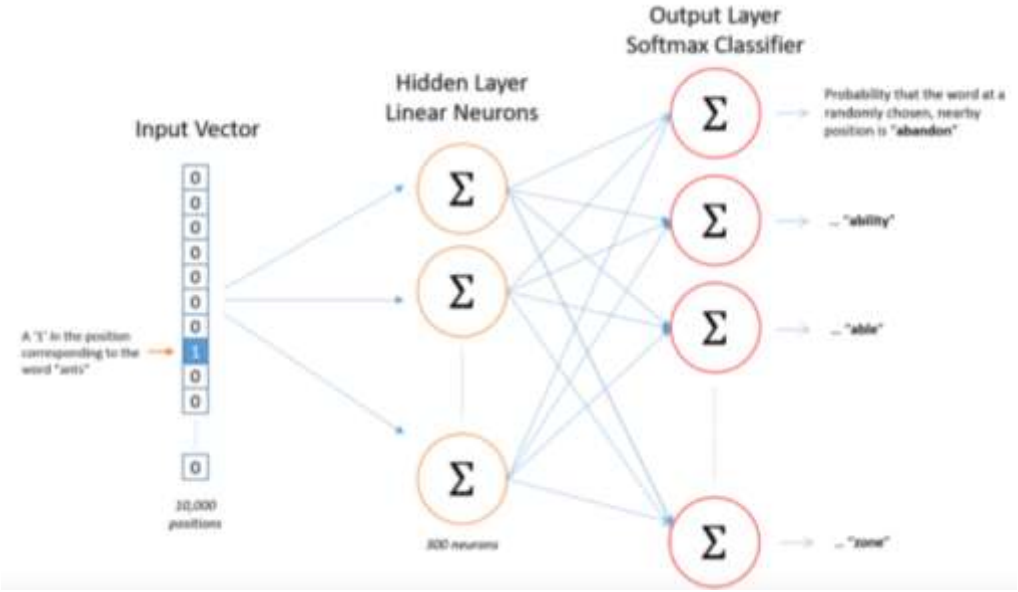



**Fig. 4.** Neural network with a hidden layer having 300 neurons

Each word therefore has a vector associated with it. The mapping size (hidden layer size) is actually chosen by the model builder (the researcher) and represents the number of similarities between words, i.e. it is the number of *features* representing the number of similarities between words. It is much smaller than the size of the single word vocabulary and, as was mentioned, is chosen by the researcher. The larger this number, the better the model will potentially be.

Two methodologies are known to implement Word2Vec called contextual representations:

- CBOW (Continuous Bag-of-Words) - the model learns the word-vector mapping and predicts the current word based on its context. It considers a word (label) together with its surrounding words (which actually form the context). These words feed the neural network and are used to predict the target word.

- Skip-Gram - the model learns to predict words around the current word. In other words, it does exactly the opposite: given a target word, it predicts the words around that word (it predicts the context).

Both models work with neural networks that learn words based on local context of use. The context is defined by a "window" of neighbour words. This "window" is a configurable parameter of the model, and its size has a strong effect on the similarity of the resulting vector: a large "window" tends to produce topic-based similarity while small "windows" produce functional and syntactic similarity.

Advantages of Word2Vec are:

- efficient and high-quality mappings;
- large mappings from billion-word corpora.

GloVe, the second algorithm presented here, has an interesting approach for producing word-vectors: it is based on counting words vis-à-vis of context [6]. It builds a matrix that will reflect/count how many times a word appears in a context. It no longer uses a "window" of words to define the local context. Instead, it builds an explicit word-context matrix (a matrix that reflects the number of occurrences of a word in a context), using statistics over the entire text corpus. The result is a learning model that sometimes leads to word mappings that are considered better. GloVe uses a matrix in which each row represents a word and each column represents a context. In a given context, words may or may not appear. The matrix values represent the frequency of occurrence of each word in a given context. After applying a matrix size reduction, the result will be the word-vector mapping: each row will actually be the vector of that word. In figure 5, an example is shown where the contexts are: I like flying; I enjoy flying; I like NLP; I like deep learning.

| counts | I | like | enjoy | deep | learning | NLP | flying | . |
|---|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| enjoy | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| deep | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| learning | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| flying | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| . | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

**Fig. 5.** Glove matrix

Stanford University has created a huge set of GloVe-derived word-vectors on a text from Wikipedia, with a tokenization process built on a vocabulary of 400,000 most-used words. These vectors can be downloaded online by choosing the desired size for them: 25, 50, 100, 200.

The Word2Vec and GloVe models are similar in the way they work. Both aim to construct a vector space in which the position of each word is influenced by its neighboring words based on their context and semantics. Word2Vec uses local individual examples of pairs of repeated occurrences of words and GloVe uses global statistics of repeated occurrences, aggregated over all words in the vocabulary. Table 1 mirrors the characteristics/differences between the two algorithms.

**Table 1.** Word2Vec and GloVe features

| Word2Vec | GloVe |
|---|---|
| Predictive model.<br>Uses a feed-forward neural network.<br>It is using a local context: when an unknown word shows up, the model gets confused and will try to find his embeddings in its vocabulary.<br>It is a static and not a dynamically generated embedding. | Model based on counting (count-based model).<br>Does not use a neural network.<br>Creates a global co-occurrence matrix, estimating the probability that a given word co-occurs with other words.<br>Requires up-front processing of the entire dataset. |

To conclude, in our days, when talking about contextualized embeddings it is usually understood that we are talking about embeddings that dynamically change depending on the given context. This is done by using models such as BERT (as we'll see later in this paper), which are trained on large amounts of text data and can generate embeddings that capture the meaning of a word based on its context.

## 5 Encoder-decoder architecture for the transition to the new paradigm

The encoder-decoder architecture [7] is a fundamental concept in machine learning and natural language processing (NLP), and it is used in several types of neural network models, in particular in the context of recurrent neural networks (RNNs), convolutional neural networks (CNNs) and text-generating models [8]. The encoder-decoder architecture is based on the idea of transforming an input, encoding it and providing a corresponding output, a process used in a variety of applications such as machine translation, text summarization, text generation, speech recognition, etc.

In this architecture, LSTM cells are used to solve the so-called "gradient vanishing" problem for longer sequences of data, with memory being kept for the long sequences. The encoder processes an entire sentence word by word. The data is passed sequentially one after the other creating so-called "dependencies" between the previous state and the current state. The word embedding process is generated in different time steps, one by one. The inputs and outputs are done in ordered lists of numbers (in sequences). Recurrent networks are used in the encoder and also in the decoder (Figure 6). The ENCODER network takes a sequence as input and creates an encoded representation from it. The second network is the DECODER network which takes as input the output of the first network i.e. the encoded representation. It generates an output sequence by decoding it.

The architecture can be approached in 2 ways:

- or let the encoder network learn the word-vector mapping from scratch, feeding it with training data,
- or use pre-trained mappings.

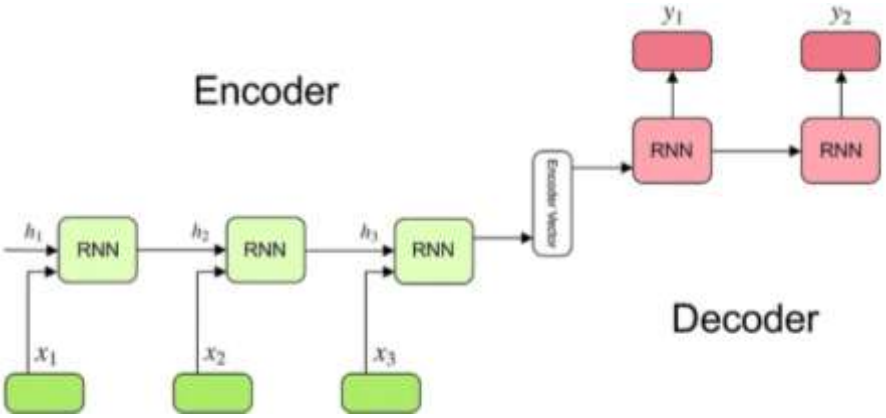



**Fig. 6.** Encoder-Decoder sequence-to-sequence

This architecture has some drawbacks: the sequential nature of the RNN-utilization slows down GPU computation. Even though LSTM cells possess a long memory, still memorizing things over a long period of time is challenging for them. It works with fixed-size vectors (word vectors and context vectors) that may not contain all relevant information.

## 6 Transformer and Attention mechanism - the new paradigm in human language processing

In this new paradigm, TIME has been dropped: temporal recurrence has been replaced by Attention, in natural language processing with deep learning technology. The transformer was introduced in 2017. Like recurrent neural networks (RNNs), transformers are designed to handle ordered sequences of data, like natural language, for various tasks such as machine translation and text summarization (extracting the most relevant information from a text) [9].

However, unlike RNNs, transformers do not require the sequence to be processed in order. So, if the data in question represents natural language text, the transformer does not need to process the beginning of a sentence before processing the end. Because of this feature, the transformer allows much more parallelization than RNNs during training. In other words, all the processes that were solved by recurrent networks together with LSTM cells are now solved simultaneously rather than sequentially. Since their introduction, transformers have become the basic building blocks of most state-of-the-art architectures in NLP, replacing in many cases the models based on recurrent neural networks with input/output gates (more specifically LSTM cells). Since the architecture of transformers allows for more parallelization during training computations, it has become possible to use a much larger amount of data than was possible in the past before this architecture was introduced.

The architecture is based on neural networks (with zero recurrence) and a mechanism called Attention. At the heart of the Attention mechanism is the idea of allowing the decoder to "look back" at the input and extract important information to be used in the decoder. The transformer allows both the Encoder and the Decoder to see the entire input sequence, all at once. The transformer does not require that the sequence be processed in order. If the data in question represents natural language text, the Transformer does not need to process the beginning of a sentence before processing the end. The Transformer allows parallelization: words are processed all at once, in parallel. "Attention is all you need" [10]: no more LSTM cells, no more cell state, no more recurrent networks. TIME is dropped: temporal recurrence has been replaced by Attention.

Transformers preserve word order through the use of "positional encoding". Unlike recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformers do not inherently understand the order of words in a sequence because they process the entire input simultaneously rather than sequentially. To enable the transformer model to incorporate the order of words, positional encodings are added to the input embeddings. These encodings are designed to provide information about the position of each word within the sequence. The positional encoding vector is added to the word embedding vector.

Each word is mapped to a vector but the same word in different sentences can have different meanings. That's why the positional encoder, which is also a vector, has emerged. This new vector provides a context based on the position of the word in the sentence. Let's illustrate how positional encoding helps a transformer understand the word order with a concrete example.

Consider the word "bank" in two different sentences:

1. "I went to the bank to deposit money."
2. "The river bank was overflowing after the rain."

Without positional encoding, the transformer would see the same embedding for "bank" in both sentences and might struggle to disambiguate the meaning based solely on the context provided by the surrounding words. Now

we combine the word embeddings with their respective positional encodings by simply adding them together. For the word "*bank*" in both sentences:

1. In "I went to the bank to deposit money." (position 4):

- If word embedding for "bank" is: [0.4, 0.4, 0.4]
- and if positional encoding at position 4 is: [−0.7568, −0.6536,0.0335]
- the combined embedding will be their sum: [0.4,0.4,0.4]+[−0.7568,−0.6536,0.0335] =[−0.3568,−0.2536,0.4335]

2. In "The river bank was overflowing after the rain." (position 2):

- Word embedding for "bank": [0.4,0.4,0.4]
- Positional encoding at position 2: [0.9093, −0.4161,0.0168]
- Combined embedding: [0.4,0.4,0.4] + [0.9093, −0.4161,0.0168] = [1.3093, −0.0161,0.4168]

Interpretation: The combined embeddings for "bank" are now different in the two sentences due to the added positional information. This allows the transformer to distinguish between the two usages of "bank" based on their positions and surrounding context. Positional encoding helps the transformer model incorporate word order into its understanding of the text. By adding these encodings to the word embeddings, the model can better capture the context and meaning of words in different positions within a sequence.

After the sentence goes through the embedding process and the position encoder is applied, new vectors are obtained having information about their position in the sentence, which translates to CONTEXT. These vectors (containing context info) pass through the Encoder block (Figure 9). Here, there are several layers/processes in which Attention intervenes, and then a layer that contains a feed-forward neural network (the simplest neural network). The feed-forward network is an important component of the Transformer: each Attention vector passes through this network that brings it to a suitable form for the next block of the encoder or decoder. Attention involves answering the questions: "which part of the input sentence should you focus on?"; "how relevant is a certain word (at a certain position in the sentence) to the other words in the same sentence?".

RNNs have realized a generic system where, with only a few lines of code, translation from one language to another language has been achieved, even with the help of context taken from English. However, RNNs have the following problem: when generating more than 1000 - 2000 words, the state of the cells and the gating processes of the LSTM cells (gating process) make the gradient vanished (vanished gradient) because these cells do not have such a long memory. In these neural networks that translate sentence after sentence (sentences sometimes as long as 40-60 words), token after token translation is successful, but the problem is that they do not "copy" human thought: when humans speak, they think about the context. For example, when someone is talking to a friend, they may be referring and thinking about something they discussed 20 years ago, and the interlocutor knows what it is about. This is long-term dependencies [11].

The processes inside the Transformer (Fig. 7, [12]) for a simple translation of a text from English into Romanian are [13]:

- In the Encoder, the English sentence is vector mapped (with Word2Vec also solving the semantics of the sentence) and then, to these vectors, are added the positional vector, to get the context of the word in that sentence. The attention vectors are constructed and processed here, for each word. The attention vector analyses its relationship to itself and to other words, resulting in, for example, 8 attention vectors for a 4-word phrase. These vectors are then averaged to obtain the final attention vector for each word.
- The Decoder receives the Romanian phrase. Because the phrase was mapped into vectors, the decoder receives numbers, vectors and matrices. Then adds the position vector to get the context of the word in the phrase. The decoder has 3 main components of which two are

similar to the encoder. The self-attention block generates attention vectors for each word in the sentence to represent the relations between words: how dependent is a word with other words in the sentence. These attention vectors together with the vectors from the encoder are passed to another attention block (called Encoder-Decoder Attention). This block determines how related (dependent) the vectors are to each other. At this point we are basically mapping the two languages English and Romanian. The output of this block is the attention vectors for each word in the 2 languages, but each vector represents (reflects) the relations to the other words in both languages. These new attention vectors are then passed to a unit through a fast-forward neural network to make them more fitted for the linear layer.

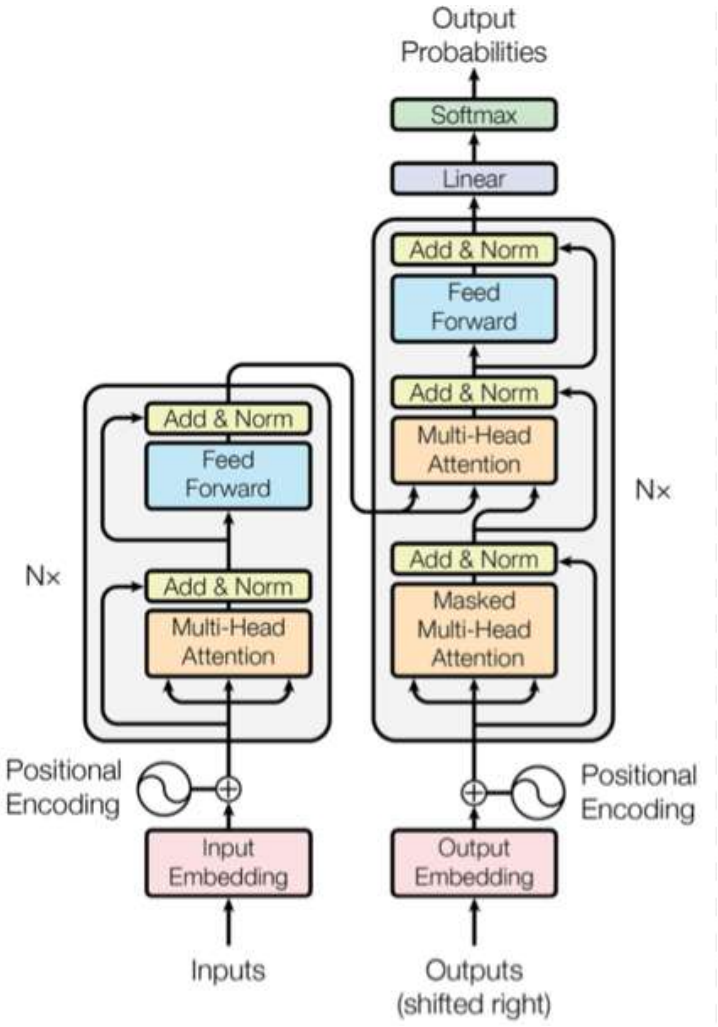



**Fig. 7**. The Transformer

The linear layer is nothing more than another fast-forward network that equalizes the size of the vectors to that of the number of words in the Romanian language. It then passes into the Softmax layer which transforms everything into a probabilistic distribution easily interpretable by humans. The output is a Romanian word with the highest probability. And so, the decoder predicts the next word in the sentence. The process is repeated many times (for each Romanian word in the sentence) until the end is reached. The first attention block in the Decoder is also called the "masked attention block": while all the words in the English sentence can be used to generate the next Romanian word, only the words preceding the next Romanian word in the English sentence are used to generate the Romanian words. If all the words of the Romanian sentence were used, there would be no such thing as network

learning: the network would output the next word without learning anything. While working in parallel on matrix operations, the matrices must mask the later words by turning them into zeros so that the attention network cannot use them. The Transformer model achieves referencing based on content rather than strictly relying on the position of words. This is one of the key innovations and advantages of the Transformer architecture, particularly through its use of self-attention mechanisms [14]. The self-attention mechanism allows each word in a sentence to attend to all other words in the sentence. This means the model can weigh the relevance of each word in relation to every other word. Attention scores are calculated based on the content of the words, allowing the model to focus on semantically important parts of the input regardless of their position. Each word is represented as a query, key, and value vector. Unlike RNNs, which process words sequentially, transformers process the entire input simultaneously. This enables them to understand relationships between distant words without being biased towards word order. By referencing content rather than position, the transformer can more accurately understand the context and meaning of words in various positions, improving tasks such as translation, text summarization, and question answering. This content-based referencing is a powerful feature that distinguishes transformers from traditional sequential models like RNNs and enables them to handle complex linguistic phenomena more effectively. It is important to mention another crucial technique used in the Transformer model (only in the Decoder, just before applying the Softmax function): masking. This is necessary to ensure that certain information is either hidden or highlighted during the training and inference processes. In a future article we intend to explain in details how masking is applied (to the Attention mechanism) to prevent the Attention mechanism of a transformer from "cheating" in the decoder when training, like is happening in a translating task for instance. For now, is enough to understand that the Attention mechanism looks at all the past information and it is able to consider a very long context. This is one of the main reasons why the Transformer model has become so successful, especially in natural language processing tasks. It has a powerful way to capture long-range dependencies and build a comprehensive contextual understanding.

## 7 Conclusions

This paper traced the evolution of NLP technology, beginning with the early days of computing when words were associated with and transformed into numerical representations in vector space. This enabled computer models to handle words mathematically, facilitating the analysis and processing of their semantic meanings. Building on this, techniques like Word2Vec, which converts words into vectors based on their context within a corpus, and GloVe, which derives vector representations from word co-occurrence in a corpus, were developed.

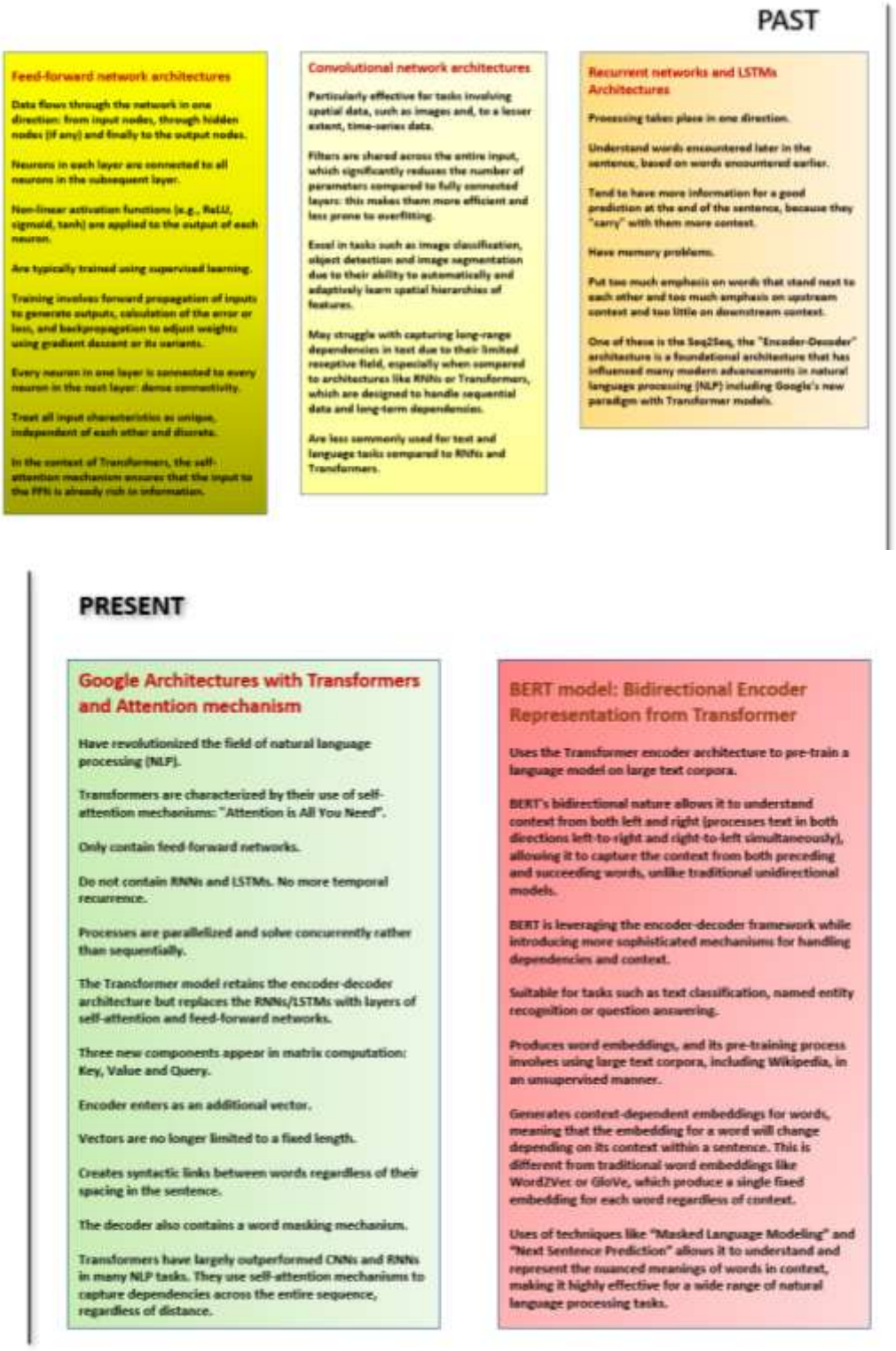



**Fig. 8** The evolution of technology in NLP world

From this point, it was just a short step to the revolution of AI models in natural language processing. This step was marked by the emergence of Transformers, embodying the new philosophy of "Attention Is All You Need.". Outstanding examples of cutting-edge language processing technology include the chatbot robots ChatGPT (developed by OpenAI) and BERT (developed by Google, [15]. It is important to note that the evolution in NLP is rapid and new developments may appear at any time. The past and present

evolution of architectures is visualized in Fig. 8.

**References**

[1] L. . F. Campanile, S. Seltmann and A. Hasse, "A measure for the similarity of vector spaces," 2021.

[2] C. Lala, "Word vector-space embeddings of natural language data over time," 2014.

[3] D. Widdows and T. Cohen, "The Semantic Vectors Package: New Algorithms and Public Tools for Distributional Semantics," in IEEE Fourth International Conference on Semantic Computing, 2010.

[4] S. Kulshretha and L. Lodha, "Performance Evaluation of Word Embedding," International Journal of Innovative Science and Research Technology, vol. 8, no. 12, 2023.

[5] Chris McCormick Word2Vec Tutorial, https://medium.com/nearist-ai/word2vec-tutorial-the-skip-gram-model-c7926e1fdc09".

[6] J. Pennington, . R. Socher and C. D. Manning, "GloVe: Global Vectors for Word Representation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, October 25-29, 2014.

[7] Encoders-Decoders, Sequence to Sequence arhitecture, https://medium.com/analytics-vidhya/encoders-decoders-sequence-to-sequence-architecture-5644efbb3392".

[8] M. Silfverberg and M. Hulden, "An Encoder-Decoder Approach to the Paradigm Cell Filling Problem," in Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2018.

[9] A. Gireesh and A. Joseph, "Unveiling the Transformer Revolution," Proceedings of National Seminar on Artificial Intelligence & Machine Learning, November 2023.

[10] V. Ashish, S. Noam, P. Niki, U. Jakob , . J. Llion, G. N. Aidan and K. Lukasz , "Attention is all you need," in Conference on Neural Information Processing Systems, Long Beach, CA, USA, 2017.

[11] "Understanding LSTM Networks," 2015.

[12] F. Elmenshawii, "Introduction to Transformers - Machine Translation," 2024.

[13] M. Lewis, . Y. Liu and N. Goyal, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," Computation and Language, 2019.

[14] . X. Amatriain, "How do transformers work in NLP," 2023.

[15] L. Steven, "Levy, Steven. "8 Google Employees Invented Modern AI. Here's the Inside Story", ISSN 1059-1028.".

**Paul TEODORESCU** is an Engineer with an international background in engineering and IT. He has worked in IT field in Romania and Canada. Specializing in databases, PL/SQL, Oracle, Data Warehousing, Business Intelligence, Artificial Intelligence (Machine Learning, Artificial Neural Networks, Natural Language Processing), he studied and worked for 11 years in Canada. He is currently working at Computer Science Research Institute in Bucharest - ICI - as a research scientist and is involved in Artificial Intelligence, NLP and GIS projects.

**Silvia OVREIU** is PhD in Deep Learning at the Faculty of Electronics, Telecommunications and Information Technology at the University Politehnica of Bucharest, with the thesis titled "Retinal Image Analysis using Deep Learning Algorithms." She was involved in research within the UPB Proof of Concept 2020 Project and served as a research assistant. She was the principal investigator of the SAIGHT project (Software for Automatic Analysis of Ocular Images). The main goal of the project is to create a platform based on Deep Learning for the automatic analysis and diagnosis of ocular diseases such as glaucoma. She is involved in organizing the International Summer School on Imaging with Medical Applications (SSIMA).