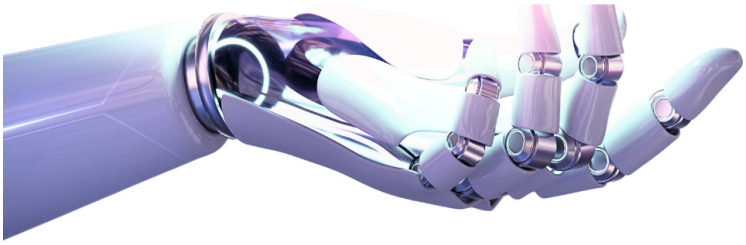
# ChatGPT, LLMs and Deep Learning with Python



**Panagiotis Symeonidis**
Associate Professor
University of the Aegean

# ChatGPT, LLMs, and Deep Learning with Python

**Transformers, Data Mining, Information Retrieval, Machine Learning and Artificial Intelligence Algorithms**

---

**Panagiotis Symeonidis**
Associate Professor – University of the Aegean
psymeon@aegean.gr

Dedicated to my son **Dimitris**!

(This pages is left intentionally blank)

# Contents

The content from this page has been removed

The content from this page has been removed

Moreover, transformers can learn on their own using data that isn't labeled, a method called self-supervised learning. This works really well for language models because transformers can take advantage of the huge amounts of text available online and from other sources to improve their understanding.

For example, a model might be trained to predict missing words in a sentence, such as turning "The bird [MASK] high in the sky" into "The bird flies high in the sky." Another common task is next-sentence prediction, where the model decides if two sentences logically follow each other, like "The boy kicked the ball" and "It rolled down the hill." Self-supervised learning helps the transformer learn patterns, relationships, and structures in language, enabling it to perform well on more specific tasks later, like translation or summarization.

## 1.2 Attention

Attention mechanism is an important component of a transformer. In this section, we will describe the self-attention mechanism by using as a paradigm the recommendation systems. Let us assume that we know the attributes of movies in the catalog, which can be seen in Figure 1.1, where the movies' vectors are represented with a matrix $\mathbf{X}$ of dimensions $N \times D$ in which the $n$-th row corresponds to the movie vector $\mathbf{x}_n^\top$. We refer to these data vectors as *tokens*, and in our running example they correspond to movies.

| | $F_1$ | $F_2$ | ... | $F_d$ |
|---|---|---|---|---|
| $M_1$ | 0 | 1 | ... | 0 |
| $M_2$ | 1 | 1 | ... | 0 |
| $M_3$ | 0 | 1 | ... | 0 |
| $M_4$ | 0 | 1 | ... | 0 |
| ... | ... | ... | ... | ... |
| $M_n$ | 0 | 0 | ... | 1 |

Figure 1.1: Running example: *Movie-Feature* matrix $\mathbf{X}$.

Then, the elements $x_{ni}$ of the tokens are called *features*. In Figure 1.1, the matrix may corresponds to movie features such as genre, actor, director, etc.). The fundamental building block of a transformer is a function that takes a data matrix as input and creates a transformed matrix $\widetilde{\mathbf{X}}$ of the same dimensionality as the output. We can write this function in the form

$$\widetilde{\mathbf{X}} = \text{TransformerLayer}\,[\mathbf{X}]\,. \tag{1.1}$$

Each transformer layer contains its own weights and biases, which can be learned using gradient descent using an appropriate loss function. A transformer layer works in two stages: (i) attention layer and (ii) feature transformation. Let's explain them step by step:

## 1.3 Detailed Transformer's Architecture for Movie Tokens

For movie recommendation, each movie $M_i$ is represented by a vector $x_i \in \mathbb{R}^d$ containing its features. The sequence of all movies $[x_1, \ldots, x_N]$ is fed into the transformer encoder. The output $[y_1, \ldots, y_N]$ represents enriched embeddings that capture contextual relationships between movies. A transformer for movie tokens consists of the following components:

- **Input Representation:** Each movie is represented as a vector of features (e.g., genre, cast).
- **Multi-Head Attention:** Computes attention scores to focus on relevant movies in the dataset.
- **Attention Scores:** Combines information from similar movies into enriched representations.
- **Concatenation of Multi-Heads**: Concatenates the output of all heads.
- **Feed-Forward Network (FFN):** Applies two dense layers with a ReLU activation to the attention output.
- **Final Output:** Generates enriched movie embeddings capturing contextual relationships.

For movie recommendation, each movie $M_i$ is represented by a vector $x_i \in \mathbb{R}^d$ containing its features. The sequence of all movies $[x_1, \ldots, x_N]$ is fed into the transformer encoder. The output $[y_1, \ldots, y_N]$ represents enriched embeddings that capture contextual relationships between movies.

### Example: Five-layer Transformer with Movie Tokens

To illustrate the application of a transformer encoder layer, consider the following example:

1. We have a movie catalog represented by a matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

   Here: Each **row** $(M_1, M_2, M_3, M_4)$ represents a movie (*tokens*).

   Each **column** $(F_1, F_2, F_3, F_4, F_5)$ represents a feature (e.g., Action, Comedy, Actor, Director, Budget).

2. **Multi-Head Attention:** Compute Q, K, and V by applying learned projection matrices:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V. \tag{1.4}$$

   For simplicity, assume these matrices are already calculated.

The content from this page has been removed

The content from this page has been removed

another, speech-to-text, and other sequence transformation tasks. A notable example of a Seq2Seq transformer is the T5 (Text-to-Text Transfer Transformer) model, which unifies various NLP tasks in a single framework.
In summary, these three categories—encoders, decoders, and Seq2Seq transformers—form the foundation of modern LLMs, enabling diverse applications across the field of natural language processing.

## 1.5 Decoder transformers

Decoder transformers are a class of neural network models used for autoregressive sequence generation tasks, where the goal is to predict the next element in a sequence based on preceding elements. These models are characterized by their ability to generate text, predict sequences, or perform similar tasks that involve conditional probability modeling.

### 1.5.1 Attention

Next, we will motivate the use of the attention mechanism by using natural language as an example. Consider the following two sentences:

**The jaguar runs into the jungle, chasing a hunter**.

**The jaguar speeds down the road, chasing a super car**.

.
Here, the word "jaguar" has different meanings in the two sentences. In the first sentence it refers to an animal, whereas in the second sentence it refers a car. However, this can be detected only by looking at the context provided by other words in the sequence. We also see that some words are more important than others in determining the interpretation of "jaguar". In the first sentence, the words "jungle" and "hunter" most strongly indicate that "jaguar" refers to the large wild cat, whereas in the second sentence, the words "road" and "super car" are strong indicators that "jaguar" refers to a luxury car brand. We see that to determine the appropriate interpretation of "jaguar", a neural network processing such a sentence should *attend* to, in other words rely more heavily on, specific words from the rest of the sequence. This concept of attention is illustrated in Figure 1.2.
Moreover, we also see that the particular sentence's word position that should receive more attention depend on the input sequence itself: in both sentences it is the third, sixth and ninth words that are important.

## 1.6 Self-Attention

Transformers, originally developed for processing sequential text data, but they have proven to be highly effective also in recommendation systems.

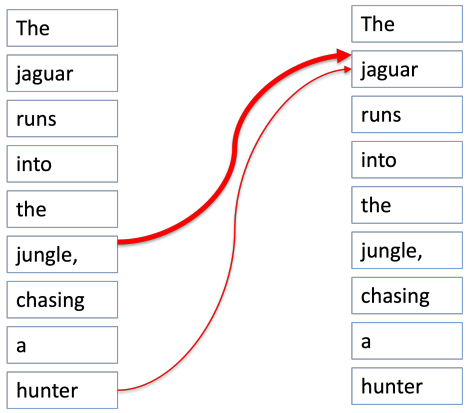



Figure 1.2: Visual representation of attention mechanism in which the interpretation of the word "jaguar" is influenced by the words "jungle" and "hunter", with the thickness of each line being indicative of the strength of its influence.

### 1.6.1 Overview of the Decoder Architecture

The decoder architecture is built upon the transformer framework and is specifically designed to predict the next token in a sequence while attending only to past tokens. The input to the model is a sequence of tokens, represented as numerical indices in a vocabulary. These indices are converted into dense vector embeddings using a learnable embedding layer. To incorporate positional information, a *positional embedding* is added to the token embeddings, ensuring that the model is aware of the order of the sequence.
Each transformer layer within the decoder consists of the following components:

- **Causal Multi-head Self-attention:** This layer allows the model to attend to previous tokens in the sequence but prevents it from accessing future tokens by using a *causal attention mask*.

- **Feed-forward Neural Network:** This component processes the outputs of the attention mechanism through a sequence of dense layers with non-linear activations.

- **Layer Normalization and Dropout:** These techniques are used to stabilize training and prevent overfitting.

At the final stage, the output is passed through a dense layer followed by a softmax activation, which produces a probability distribution over the vocabulary for the next token.

### 1.6.4 Inference and Generation

During inference, the decoder operates autoregressively. It begins with a seed sequence (e.g., The jaguar speeds down) and generates tokens one at a time by sampling from the predicted probability distribution. This iterative process continues until an end-of-sequence token is generated or a predefined maximum length is reached. In the jaguar example, starting with The jaguar, the model might generate:

The jaguar speeds down the road, chasing a supercar.

This ability to generate coherent and contextually relevant sequences makes decoder transformers a powerful tool for natural language generation tasks.

## 1.7 Encoder transformers

Encoder transformers are designed to process input sequences and generate fixed-length representations that can be used for downstream tasks such as classification, token labeling, or other natural language understanding tasks. A prominent example of an encoder transformer is *BERT* (Bidirectional Encoder Representations from Transformers).
The encoder transformer architecture is based on the transformer layers discussed previously. In this model, every input token is embedded into a dense vector representation and combined with positional embeddings to encode sequential information. The attention mechanism allows the model to attend to all tokens in the sequence bidirectionally.
Each transformer encoder block consists of the following layers:

### 1. Multi-head Self-attention

- **Purpose:** Captures relationships between tokens in the input sequence by computing attention scores.
- **Mechanism:**
  1. For an input sequence represented as a matrix $X \in \mathbb{R}^{n \times d}$, where $n$ is the sequence length and $d$ is the embedding dimension:
     - Compute Query, Key, and Value matrices:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

       where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ are learnable weight matrices.
     - Calculate scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

The content from this page has been removed

## 1.8 Encoder-Decoder Transformer

## Bibliography

R. Baeza-Yates and B. Ribeiro-Neto. *Modern information retrieval*, volume 463. ACM press New York, 1999.

M. Berry, S. Dumais, and G. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1994.

G. W. Furnas, S. Deerwester, S. T. Durnais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings ACM SIGIR Conference*, pages 465–480, 1988.

K. Goldberg, T. Roeder, T. Gupta, and C. Perkins. Eigentaste: a constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

P. Melville, R. J. Mooney, and N. R. Content-boosted collaborative filtering for improved recommendations. In *Proceedings AAAI Conference*, pages 187–192, 2002.

R. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings ACM DL Conference*, pages 195–204, 2000.

M. F. Porter. An algorithm for suffix stripping. *Program Journal*, 1980.

P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering on netnews. In *Proccedings Conference Computer Supported Collaborative Work*, 1994.

J. Salter and N. Antonopoulos. Cinemascreen recommender agent: Combining collaborative and content-based filtering. *Intelligent Systems Magazine*, 21(1):35–41, 2006.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. In *ACM WebKDD Workshop*, 2000.

B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. WWW Conf.*, pages 285–295, 2001.