Behind the mechanism

Author: Leandro Santos

Index

Subsets of AI

In short, Artificial intelligence (AI) refers to computer systems capable of performing tasks typically done by humans, such as reasoning, making decisions, or solving problems. I have seen distinct articles listing different subsets of AI but, in my perception, one that is more comprehensive is described in the **Figure 1**:

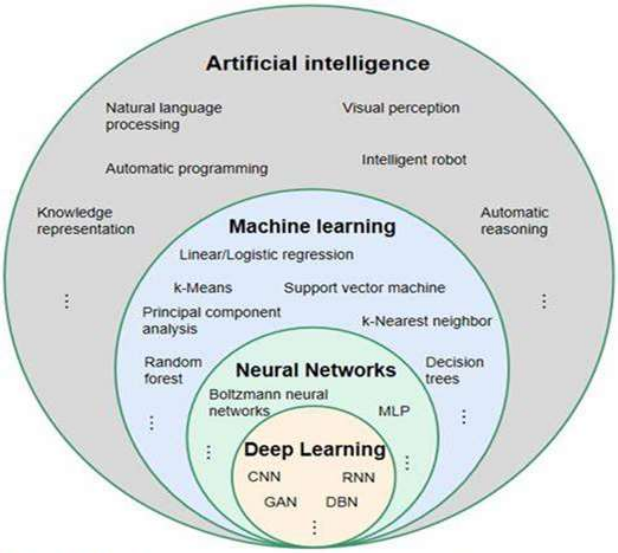


***Figure 1**. Relationship between artificial intelligence, machine learning, neural network, and deep learning. MLP: multilayer perception; CNN: convolutional neural network; RNN: recurrent neural network; DBN: deep belief network; GAN: generative adversative network.*

***Source:** Li, Song & Deng, Yu-Qin & Zhu, Zhi-Ling & Hua, Hong-Li & Tao, Ze-Zhang. (2021). A Comprehensive Review on Radiomics and Deep Learning for Nasopharyngeal Carcinoma Imaging. Diagnostics. 11. 1523. 10.3390/diagnostics11091523.*

From the shorter to the broader subsets: Deep learning refers to artificial neural networks with dense layers (several interconnected hidden layers). Neural Networks in addition to deep learning also include simpler models like Boltzmann neural networks and multilayer perceptron. Machine learning is a subfield of artificial intelligence that uses algorithms trained on data sets to create models that enable machines to perform tasks that would otherwise only be possible for humans, such as categorizing images, analyzing data, or predicting price fluctuations. In this context, in addition to statistical and multivariate data analysis machine learning includes the neural networks subfield. Machine Learning is, then, an application of AI that allows machines to extract knowledge from data and learn from it autonomously. Artificial intelligence is the broader concept of enabling a machine or system to sense, reason, act, or adapt like a human. In addition to Machine Learning, it also includes Natural Language Processing (including LLM), Visual Perception, etc.

My intention is to deeply dive into the Transformer mechanism. Its architecture is now considered a state-of-the-art technique in the field of natural language processing, like GPT.

Given that Transformer uses in its mechanics feed-forward network, we will start with concept of Neural Network.

Initial concept of Neural Networks: the perceptron

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers, and it is considered the first neural network. It was invented by Warren McCulloch in 1943 and consists of a single layer of input nodes that are fully connected to a layer of output nodes with a unit step unity. It is particularly good at learning linearly separable patterns (**Figure 2**).

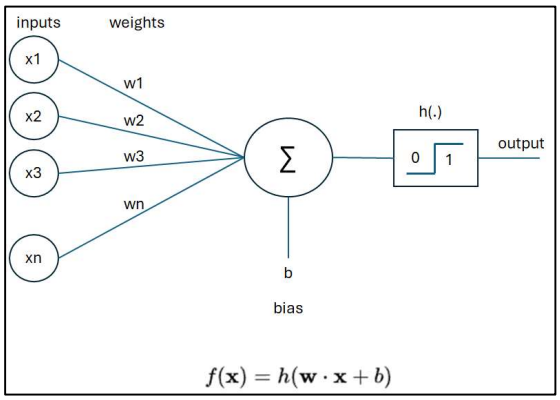


Figure 2

where h(.) is the unit step/ threshold (**Figure 3**):

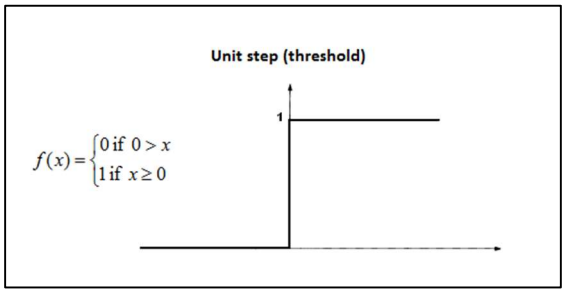


Figure 3

In a most basic form of a neural network without activation function, the structure is very similar to a linear regression. **Figure 4** shows an example where wages are associated with a linear combination of work experience, years of education, working hours and age:

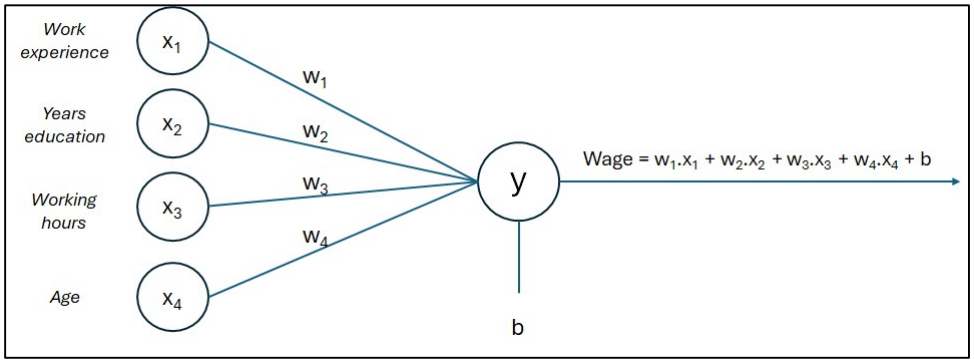


Figure 4

The difference between regression and the most basic neural network is: The traditional regression models often determine coefficients using methods like ordinary least squares (OLS) or maximum likelihood estimation (MLE) and neural network, on the other hand, adjust weights during training by minimizing the loss function through optimization techniques like gradient descent. Furthermore, input data in neural networks should be normalized. It will prevent gradient issues and enhance the model's performance. We will talk about loss function and gradient descent later in this post.

Artificial Neural Network (ANN)

As said, perceptron is a very simple architecture of a neural network. In today's real life they have many other sets of neurons (layers), sometimes associated with different activation functions (like sigmoid, Relu, ArcTan, etc.). Consider the following architecture (**Figure 5**):

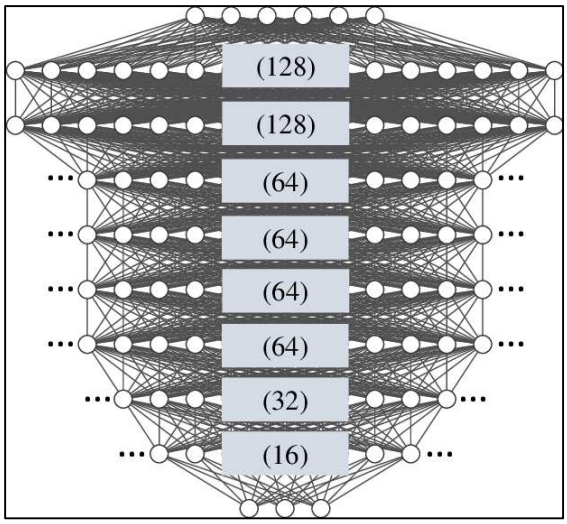
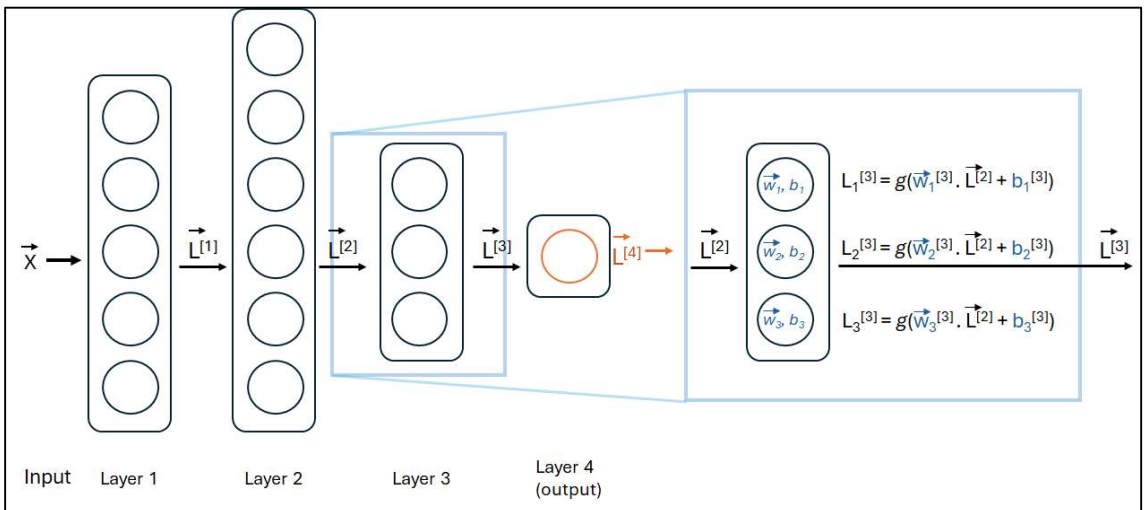


Figure 5

This is an Artificial Neural Network with dense layers (named deep learning). We have 6 neurons on the input layer, 128 neurons in the 1st hidden layer, 128 neurons in the 2nd hidden layer and so on until the output layer with 3 neurons. We cannot say that the output is a linear combination of each neuron, it isn't! ANN are non-parametric models.

Let's check how an ANN calculates the outputs in each internal layer. Consider the following example (**Figure 6**):



Source: Adapted from Dr. Andrew Ng (2023). "Advanced Learning Algorithms." DeepLearning.AI & Stanford University, Coursera.

Figure 6

In the above example (**Figure 6**), the neurons of layer 3 are calculated with the outputs of layer 2, weight, bias and activation function of layer 3. In essence the inputs of layer 3 is the output of layer 2. This process is called forward propagation and happens in one direction until the final layer (output layer).

An Intuitive Concept of Train and Test Data

About 20 years ago, I was working on a project and was responsible for determining whether the models of our current demand planning system were better or worse than those of a new system. I had relevant historical data, but to determine the best model, I had to calculate the estimated demand for the next several months and then wait to evaluate the model's performance. This approach wouldn't be acceptable to the project manager because we needed an immediate response to decide whether to keep the current system or switch to a new one.

Instead, we decided to simulate the process. Imagine we are in July but pretend we are at the beginning of the year. We calculate the estimated values from January to July using the historical data up to the previous December (training data) and then check the forecast accuracy by comparing the estimated values to the actual values (test data) from January to July.

This simple example illustrates the concept of train and test data: We split the entire dataset into two sets—the training data, where the model refines its parameters (called training process - with many iterations), and the test data, where we check if the model can generalize its performance by using these new values.

What happens after the forward propagation? After reaching the output layer (a4 in the example) is compared to the real value of the training data and a loss function will show how different the output is from the real value.

Some of the loss functions are MSE, MAE (for regression), binary and categorical cross-entropy (for classification):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Yi - \hat{Y}i\right)^2$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Yi - \hat{Y}i\right|$$

$$binary\ cross\ entropy = -\frac{1}{n}\sum_{i=1}^{n} Yi.log\hat{Y}i + \ (1 - Yi).log\left(1 - \hat{Y}i\right)$$

$$categorical\ cross\ entropy = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} Yij.\log\left(\hat{Y}ij\right)$$

where k are classes.

During the training process, machine learning algorithms employ optimization techniques such as gradient descent to minimize the loss function. By iteratively adjusting model parameters based on the gradients of the loss function, the algorithm aims to converge to the optimal solution, resulting in a model that accurately captures the underlying patterns in the data. But how does gradient descent work?

For a better visualization, let's assume the following function: $x^2 - 2x + y^2 - y$. It will represent the loss function and has a minimal value. By using the 3D calculator available on the Desmos site (**Figure 7**) we can see that the minimal is in the x,y point (1 , 0.5).

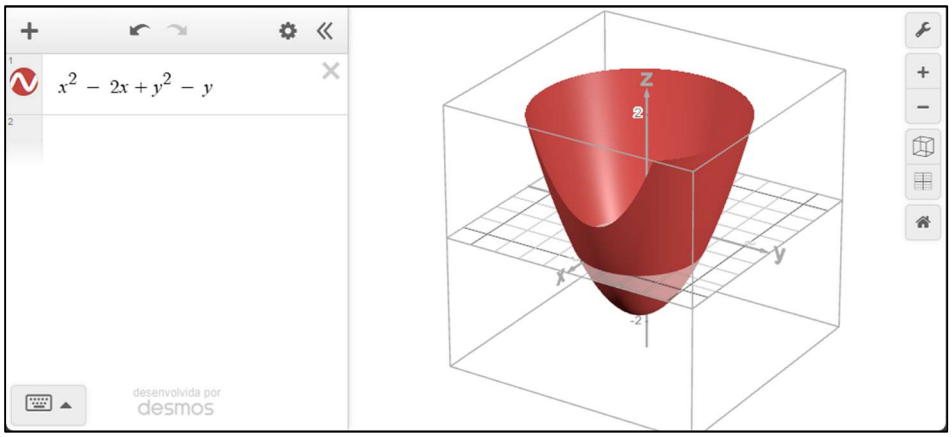


Figure 7

The gradient descent finds the partial derivatives of the loss function, adjusting the parameter when the minimal is achieved.

The **Table 1** shows the interactions to reduce x,y to the minimal point. The partial derivatives of the initial equation: $\partial/\partial x = 2x - 2$, and $\partial/\partial y = 2y - 1$. At the point (5,5) the slopes are respectively: 8 and 9. It indicates that at the point 5 of x the addition of one unit will increase the loss by 8 units. The opposite is necessary, so a small number (learning rate) is multiplied by the slope and reduced to the initial x value.

In our example: x starts at 5, and at this point the slope is 8. Just a small part of the initial value should be reduced. To achieve this, the slope is multiplied by a learning rate (0.2 in this example), resulting in 1.6. This value is subtracted from the initial x (5 - 1.6 = 3.4). The is the new x for the next iteration. After 11 executions the minimum is found at (1, 0.5).

Equation
$x^2 - 2x + y^2 - y$

Partial derivatives
$\partial/\partial x = 2x - 2$
$\partial/\partial y = 2y - 1$

After 11 steps, global minimum is achieved when: x = 1 and y = 0.5. At this point: $x^2 - 2x + y^2 - y = -1.25$

Iterations	x	y	Slope X	Slope Y	Scaled step x (0.2)	Scaled step y (0.2)
0.0	5.0	5.0	8.0	9.0	3.4	3.2
1.0	3.4	3.2	4.8	5.4	2.4	2.1
2.0	2.4	2.1	2.9	3.2	1.9	1.5
3.0	1.9	1.5	1.7	1.9	1.5	1.1
4.0	1.5	1.1	1.0	1.2	1.3	0.8
5.0	1.3	0.8	0.6	0.7	1.2	0.7
6.0	1.2	0.7	0.4	0.4	1.1	0.6
7.0	1.1	0.6	0.2	0.3	1.1	0.6
8.0	1.1	0.6	0.1	0.2	1.0	0.5
9.0	1.0	0.5	0.1	0.1	1.0	0.5
10.0	1.0	0.5	0.0	0.1	1.0	0.5
11.0	1.0	0.5	0.0	0.0	1.0	0.5

Table 1

So, assuming θ as the model parameters, the formula for the gradient descent is:

$\theta_{update} = \theta - \alpha \nabla_\theta J(\theta)$, where:

α = learning rate

$\nabla_\theta J(\theta)$ = gradient vector of the loss function

This is how neural networks 'learn' from data. At this point you might have the following question: If ANN works with vector and matrices, how NLP and visual perception models work? There is no numerical input, only words and images, right? The response is quite simple: words and images are transformed into numerical vectors and matrices.

A very interesting (and visual) explanation for image classification can be found on 3blue1brown channel. If you are interested in topics related to AI, I strongly recommend their videos!

Check this out at https://www.youtube.com/watch?v=aircAruvnKk&t=3s

Artificial Intelligence became very popular mainly because of the Generative models, like GPT (Generative Pre-trained model), DALL-E, Gemini, GAN (Generative Adversarial Networks), and many others. One of the mechanisms that contributed to the rise of generative models is called “Transformer” and will be shortly explained in the next section.

The Transformer architecture

The transformer architecture is now a state-of-the-art technique in the field of natural language processing (NLP) and other tasks involving sequential data. Introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017, the transformer is designed to handle long-range dependencies in sequences more effectively than traditional recurrent neural networks (RNNs).

A short explanation of RNN

*So far, we have seen the ANN working with inputs not in sequence. If you are familiar with time series models, like ARIMA, sometimes outputs are dependent from previous values. This is when recurrent neural networks also can be applied. In a Recurrent Neural Network, the hidden state of time “t” is not only determined by the X inputs, but also by the previous hidden state (see **Figure 8**):*

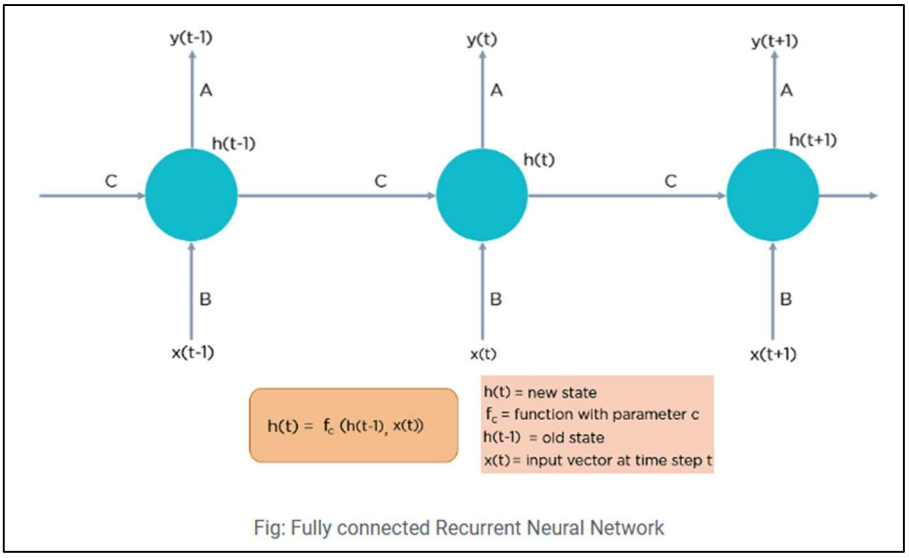


Figure 8

RNN is not the focus of this post, because RNN has 2 majors’ disadvantages in NLP: it is slow to train, and for long sequences suffers with the vanishing gradient problem. So, if you are interested, a very comprehensive and visual explanation of how RNN structure works as well as its limitations may be found at:

https://www.youtube.com/watch?v=AsNTP8Kwu80

Transformers are much more effective in these cases. So, let’s return to the Transformer architecture...

Figure 9 shows the entire architecture of the transformer model:

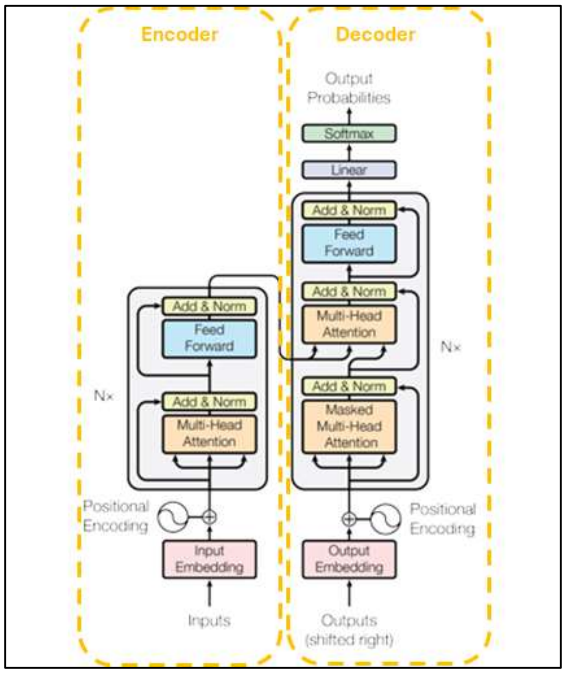


Figure 9 *- Source: arXiv:1706.03762*

The transformer architecture is typically divided into an ***encoder*** and a ***decoder***:

The Encoder

The Encoder consists of a stack of identical layers (often 6 in the original paper). Each layer includes multi-head self-attention and feed-forward networks, with layer normalization and residual connections. The Decoder also consists of a stack of identical layers, but with an additional layer for attending to the encoder's output. This allows the decoder to focus on relevant parts of the input sequence while generating output.

At first glance, these terms may seem unusual, but we will clarify them in the following sections. For now, imagine a translation model, from English to French. During training, the encoder captures the entire context of the input English phrases, while the decoder does the same with the target French phrases. The result is a translation that is far more contextualized than a simple word-for-word translation.

But before diving into the key components of Transformers, let's clarify the concept of tokenization. Remember? Machines work with vectors and matrices...

Tokenization, the initial step.

As previously mentioned, machines don't read words or images; they must transform these components into vectors or matrices. An important process in this transformation is called 'tokenization.' Tokenization is the process of dividing a text document into smaller components known as tokens. These tokens can represent words, sub words, phrases, symbols, or any other meaningful elements within the text.

After tokenization each token is mapped to a numerical ID, using a vocabulary. The OpenAI has a platform that allows users to tokenize sentences (https://platform.openai.com/tokenizer). As an example, the phrase "Hi, my name is Leandro" has 7 tokens: `Hi, my name is Leandro` represented by the token IDs `[12194, 11, 922, 1308, 382, 2018, 38110]`. Each token ID is associated with a vector in an embedding space, where similar words have similar vector representations. Just imagine: the word "apple" is represented by a vector with 'n' dimensions, for example [-0.3 0.1 1.1 -

0.56 ... 0.7]. In this space, words with similar meanings or contexts would have vectors that are close together. For example, the vector for “fruit” would be nearer to “apple” than to “car”.

These embeddings capture semantic relationships and contextual meanings. One way to create embedding vectors is programmatically using libraries. As for example, word2vec is a technique in natural language processing (NLP) for obtaining vector representations of words. These vectors capture information about the meaning of the word based on the surrounding words. In this case, the vectors for walk and ran are nearby, as are those for "but" and "however", and "Berlin" and "Germany". Another way would be to use the embeddings from a pre-trained model. For instance, you may find some embeddings for various languages on the FastText (a library developed by Facebook that provides pre-trained word embeddings) at https://fasttext.cc/docs/en/aligned-vectors.html

Key components

1. **The Self-Attention Mechanism**

The Self-Attention Mechanism was introduced in the paper “Attention is All You Need” and is also referred to as QKV attention. In this context, the attention mechanism is defined by:

$$Attention\ (Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

where:
Q = query
K = key
V = value
d_K = dimension of K

Conceptually, QKV are embedding vectors, and the first matrix multiplication is a measure of the similarity between the queries and the keys. This is transformed into weights using the SoftMax function. The weights will range from 0 to 1 so that they can be interpreted as probabilities. These weights are then applied to the values with the second matrix multiplication resulting in output attention vectors. This mechanism allows the model to weight the importance of different words in a sentence relative to each other. For each word, it calculates attention scores based on its relationship to all other words, enabling the model to capture context effectively. Consider the following example in the **Figure 10**, the phrase “The agreement on the European Economic Area was signed in August 1992.”:

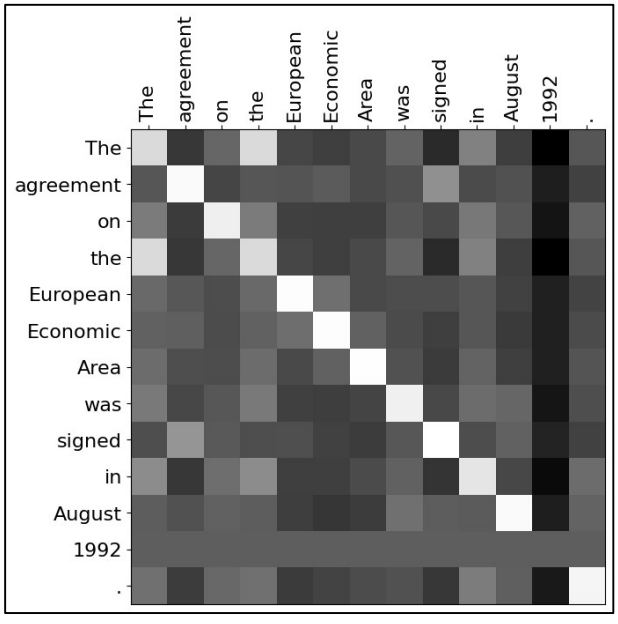


Figure 10

The similarity of the words in diagonal column is clear (well, they are the same!) and in some cases like: “signed” and “agreement” the weight is somehow relevant. The weighted matrix ($Softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)$) represents the similarity between the words and when multiplied by the initial embedding vectors of tokenized words (V), gives us the importance of different words in a sentence relative to each other.

2. Multi-Head Attention and positional encoding

Instead of having a single attention mechanism, the transformer uses multiple heads to allow the model to focus on different parts of the input simultaneously (**Figure 11**).

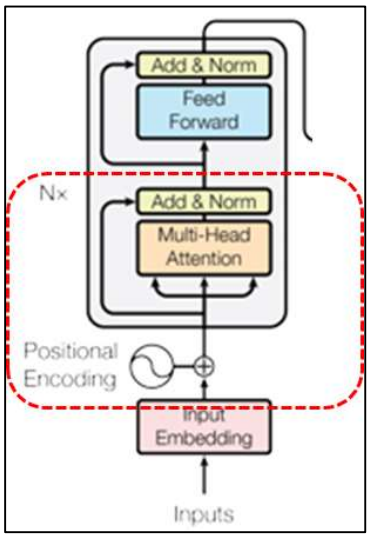


Figure 11 *- Source: arXiv:1706.03762*

Each head learns different representations, enriching the model's understanding. When you train a transformer network using multi-head attention, you feed your data into the model all at once, processing it in parallel (unlike RNNs, which handle data sequentially). However, this parallel processing means there’s initially no information about the order of the input tokens. That’s why positional encodings are necessary (**Figure 11**). They retain the order of the input sequence. Positional encodings are applied to the embeddings of the input tokens, allowing the model to understand the position of each token in relation to the others. In the article “Attention is all you need” an elegant formula is presented for positional encoding:

$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$ and $PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$ where:

d = dimension of the word embedding
pos = position of the word
i = pair number of the dimension in word embedding

A hypothetical example. Let’s imagine the embedding vector of the word “apple” has only 8 dimensions and it is the first word in a phrase (position 0), the positional encoding will be given by:

Dimensions	0	1	2	3	...	6	7
Pair of dims	0		1		...	3	
PE(0)	$\sin\left(\frac{0}{10000^{(\frac{0}{8})}}\right)$	$cos\left(\frac{0}{10000^{(\frac{0}{8})}}\right)$	$\sin\left(\frac{0}{10000^{(\frac{2}{8})}}\right)$	$cos\left(\frac{0}{10000^{(\frac{2}{8})}}\right)$	...	$\sin\left(\frac{0}{10000^{(\frac{6}{8})}}\right)$	$cos\left(\frac{0}{10000^{(\frac{6}{8})}}\right)$

The next word in the phrase will be PE(1). Supposing it also has 8 dimensions (d=8) in the first dimension the encoding will be calculated as $sin(1/1000^{(0/8)})$, and so on. Ultimately, only the sum of the positional encodings and word embeddings are fed into the model.
The elegance of these formulas is because the values of the sine and cosine equations are small enough (between -1 and 1) that when you add the positional encoding to a word embedding, the word embedding is not significantly distorted. Adding whole numbers to the word embedding will distort the semantic meaning.

An interesting post about positional encoding may be found at: https://medium.com/@gunjassingh/positional-encoding-in-transformers-a-visual-and-intuitive-guide-0761e655cea7

3. Feed-Forward Networks

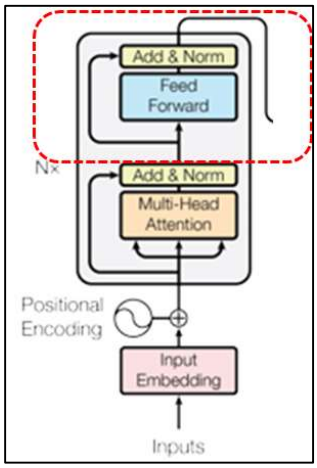


Figure 12 *- Source: arXiv:1706.03762*

Another key component is the neural network (**Figure 12**). We have seen its structure and how it optimizes the weights in the training process. Feed-forward network is a multi-layered structure where information flows in one direction, from input to output. Unlike recurrent neural networks (RNNs) that allow information loops, feed-forward networks process data in a straight line.
Each layer of the transformer includes a feed-forward neural network applied to each position independently. This typically consists of two linear transformations with a non-linear activation function in between.

4. Layer Normalization

You may have noticed that in the previous stages a normalization process was present. This is because one of the characteristics of a neural network is that the algorithm requires normalized inputs for proper execution. Models with different scales take longer to train and high values can also propagate through the layers of the network leading to the accumulation of large error gradients that make the training process unstable, called the problem of exploding gradients (Praya, C. B. 2023).
In addition to layer normalization, residual connections (or skip connections) are essential in a transformer architecture, allowing gradients to flow through the network more easily during training. In a neural network with many layers, residual connections diminish the issue of vanish gradients.
The **Figure 13** represents a deep neural network with 12 hidden layers where the information flows from input to output, without residual/skip connections. In such structure, the original input information is not preserved.

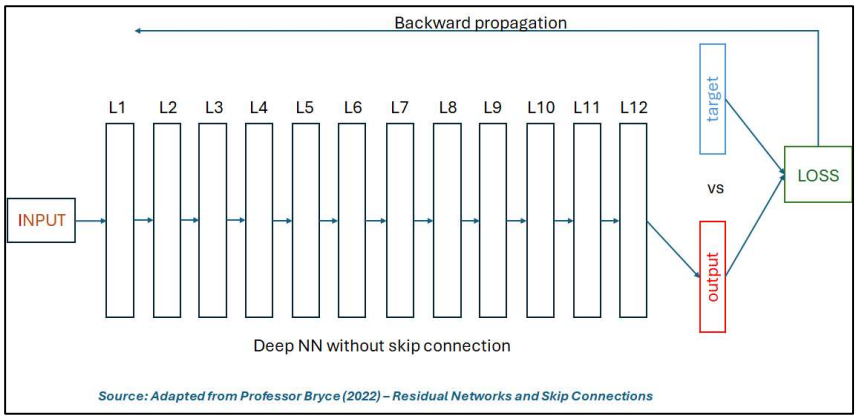


Figure 13

To preserve the initial information during the training process, the addition of a skip connection in block of layers ensures the information flow through and around the network (**Figure 14**).

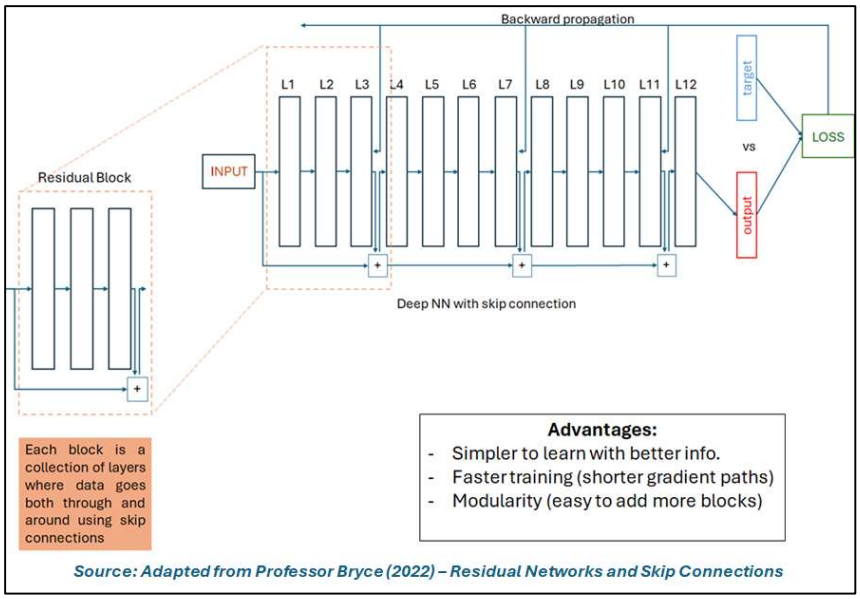


Figure 14

For a detailed explanation of residuals network and skip connections check this video: https://www.youtube.com/watch?v=Q1JCrG1bJ-A. The result of the encoder is a set of encoded vectors for the initial inputs.

The Decoder

The first part of the decoder block is composed of embedding layer and a positional encoder process that will transform words into vectors (**Figure 15**). It is very similar to the encoder process, but in this case, a target is used. If we are training a translator, the English sentence pass through the encoder block, and French sentences pass through the decoder block.

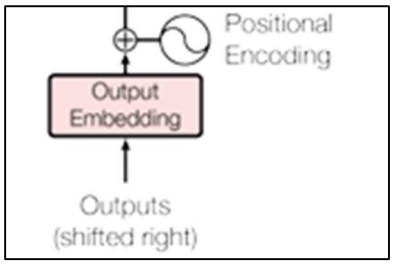


Figure 15 - Source: arXiv:1706.03762

1. Masked Multi-Head Attention

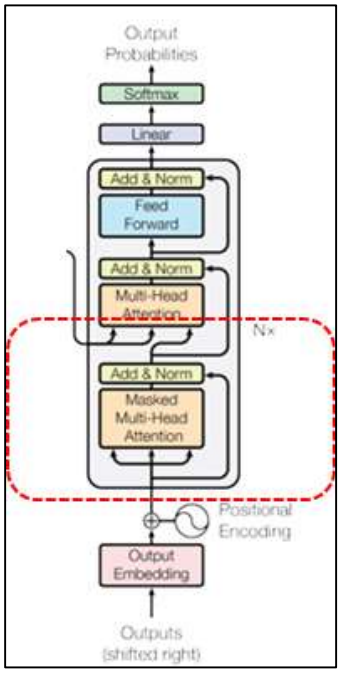


Figure 16 - *Source: arXiv:1706.03762*

The target passes through the self-attention block, generating attention vectors. Similarly to the encoder block, the attention mechanism allows the model to weight the importance of different words in a sentence relative to each other. But differently from encoder, this block is called Masked-Multi-Head attention (**Figure 16**).

Taking our example in the translation model: we need to hide the next French word so that, at first, it will predict the next word itself using previous results without knowing the real translated word. For learning to take place, it will make no sense if it already knows the next French word. Therefore, we need to hide or mask it (**Figure 17**).

The masked multi-head attention in the decoder ensures that each output token is generated in a way that respects the sequential nature of language, preventing future tokens from influencing the current token prediction, and thereby producing coherent and contextually accurate translations.

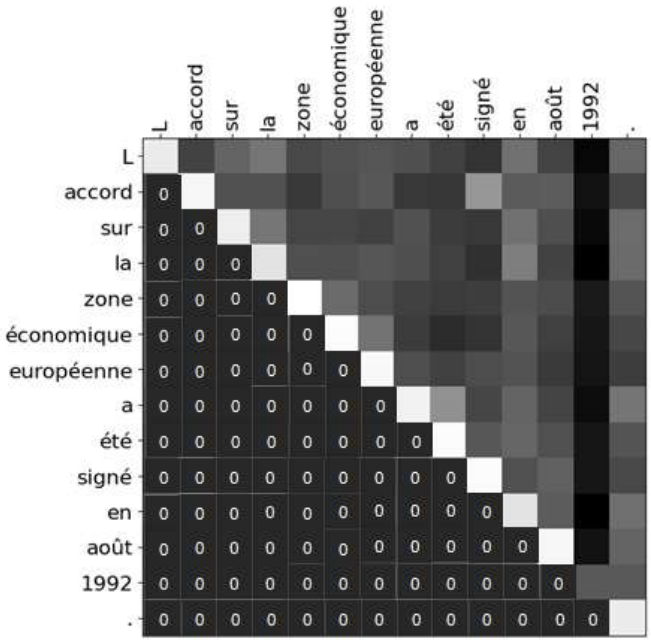


Figure 17

2. Multi-Head Attention and Feed Forward Network.

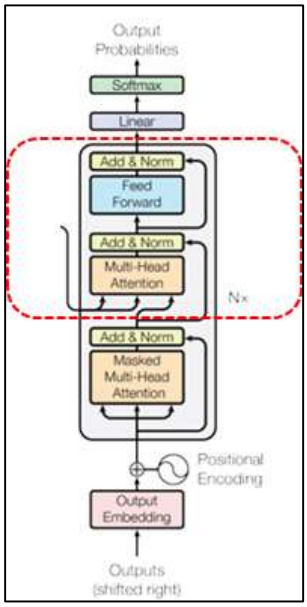


Figure 18 - *Source: arXiv:1706.03762*

This block is fed by the attention vectors from the previous layer and vectors from the encoder block. So, in our example of a translation model, the Queries would be based on the previous French tokens generated, while the Keys and Values would still come from the encoder's output (the contextual representations of the English input). This allows the decoder to attend to relevant parts of the input sentence while generating the translation. A feed-forward network will generate the output for the last stage (**Figure 18**).

At the end, the SoftMax will generate the probability distribution of the words and the resulting word with highest probability becomes the output (**Figure 19**).

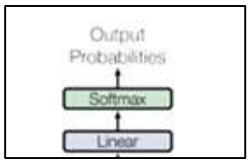


Figure 19 - *Source: arXiv:1706.03762*

References

Bryce, P [Davidson College] (Oct 23, 2022) – "Residual Networks and Skip Connections (DL 15) – Youtube - https://www.youtube.com/watch?v=Q1JCrG1bJ-A

Priya, C. B. (2023) – "Build Better Deep Learning Models with Batch and Layer Normalization" – Pinecone - https://www.pinecone.io/learn/batch-layer-normalization/

Sanderson, G. & Pullen, J. [Blue31Brown] (Oct 5, 2017) – "But what is a neural network?" – Youtube - https://www.youtube.com/watch?v=aircAruvnKk&t=3s

Sing, G. (2024) – "Positional encoding in transformers: a Visual and Intuitive guide" – Medium - https://medium.com/@gunjassingh/positional-encoding-in-transformers-a-visual-and-intuitive-guide-0761e655cea7

Starmer, J. [StatQuest] (Jul 11, 2022) – "Recurrent Neural Networks (RNN) clearly explained" – Youtube - https://www.youtube.com/watch?v=AsNTP8Kwu80

Vaswani, A. et al. (2017) – "Attention is all you need" - arXiv:1706.03762 - https://arxiv.org/abs/1706.03762