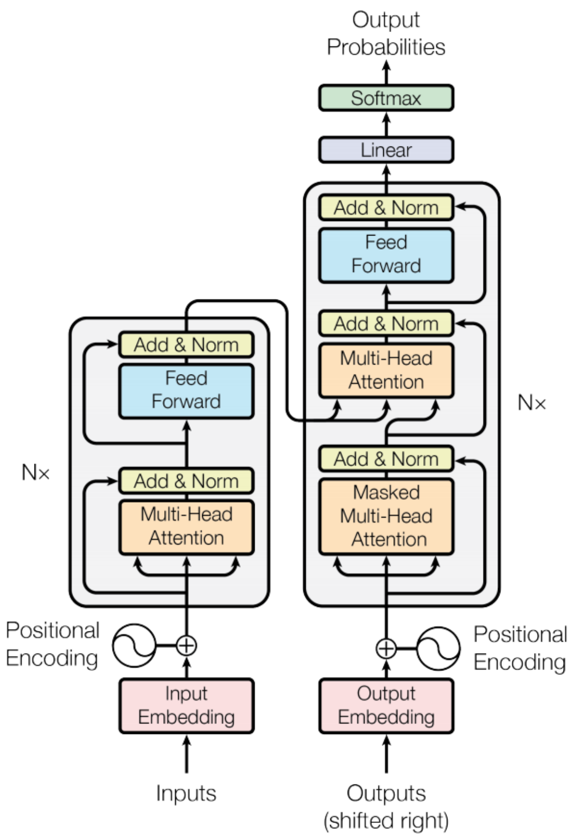


With PyTorch 🔥

2024 | © LUIS FERNANDO TORRES

Table of Contents

Introduction

In 2017, the Google Research team published a paper called *"Attention Is All You Need"*, which presented the Transformer architecture and was a paradigm shift in Machine Learning, especially in Deep Learning and the field of natural language processing.

The Transformer, with its parallel processing capabilities, allowed for more efficient and scalable models, making it easier to train them on large datasets. It also demonstrated superior performance in several NLP tasks, such as sentiment analysis and text generation tasks.

The archicture presented in this paper served as the foundation for subsequent models like GPT and BERT. Besides NLP, the Transformer architecture is used in other fields, like audio processing and computer vision. You can see the usage of Transformers in audio classification in the notebook **Audio Data: Music Genre Classification.**

Even though you can easily employ different types of Transformers with the 🤗 **Transformers** library, it is crucial to understand how things truly work by building them from scratch.

In this notebook, we will explore the Transformer architecture and all its components. I will use PyTorch to build all the necessary structures and blocks, and I will use the **Coding a Transformer from scratch on PyTorch, with full explanation, training and inference** video posted by **Umar Jamil** on YouTube as reference.

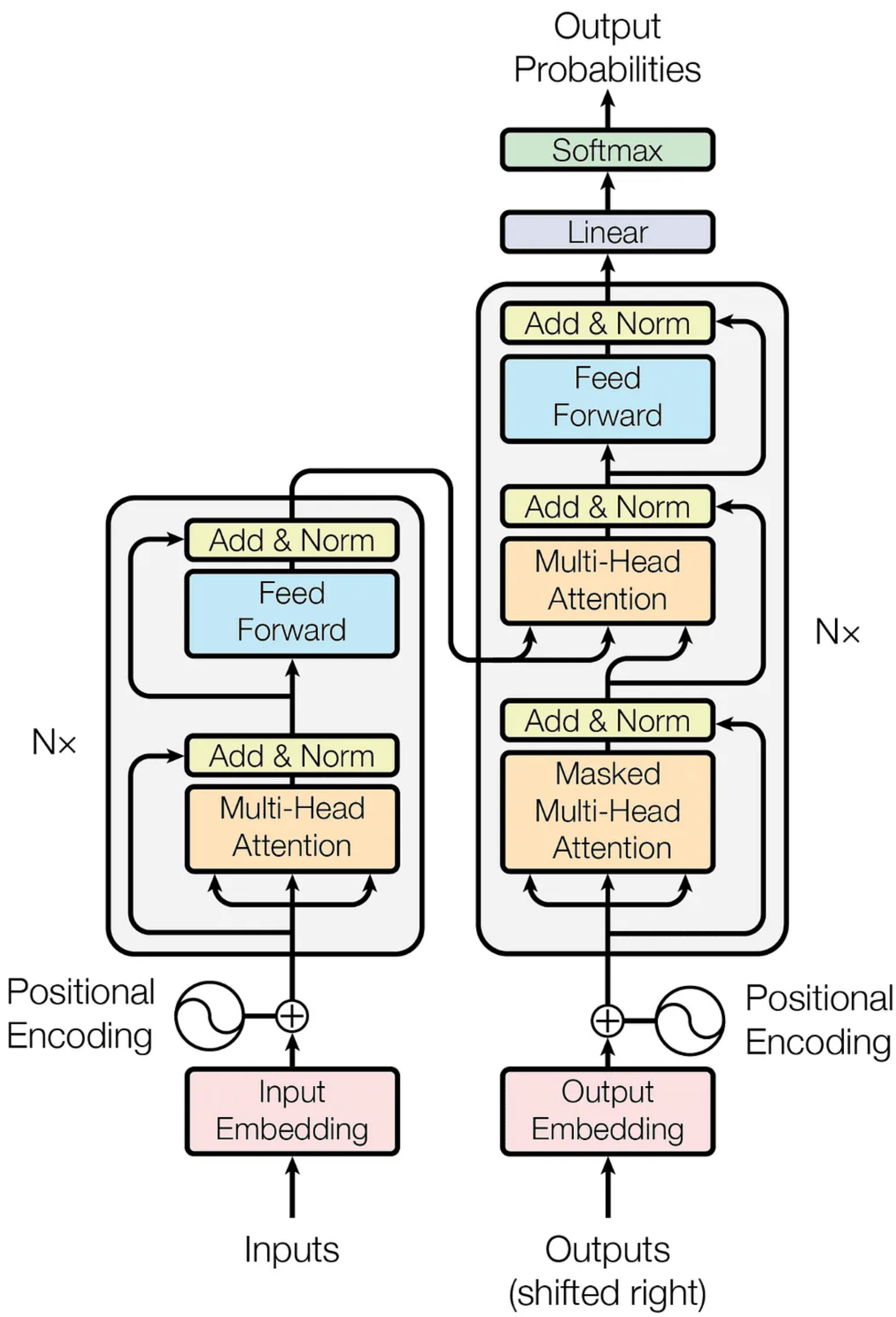
Let's start by importing all the necessary libraries.

In [1]:
```
# Importing libraries

# PyTorch
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
from torch.utils.tensorboard import SummaryWriter

# Math
import math

# HuggingFace libraries
from datasets import load_dataset
from tokenizers import Tokenizer
from tokenizers.models import WordLevel
from tokenizers.trainers import WordLevelTrainer
from tokenizers.pre_tokenizers import Whitespace

# Pathlib
from pathlib import Path

# typing
from typing import Any

# Library for progress bars in loops
from tqdm import tqdm

# Importing library of warnings
import warnings
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected ver
sion 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Transformer Architecture

Before coding, let's take a look at the Transformer architecture.



Source: Attention Is All You Need

The Transformer architecture has two main blocks: the **encoder** and the **decoder**. Let's take a look at them further.

Encoder: It has a *Multi-Head Attention* mechanism and a fully connected *Feed-Forward* network. There are also residual connections around the two sub-layers, plus layer normalization for the output of each sub-layer. All sub-layers in the model and the embedding layers produce outputs of dimension $d_{model} = 512$.

Decoder: The decoder follows a similar structure, but it inserts a third sub-layer that performs multi-head attention over the output of the encoder block. There is also a modification of the self-attention sub-layer in the decoder block to avoid positions from attending to subsequent positions. This masking ensures that the predictions for position i depend solely on the known outputs at positions less than i.

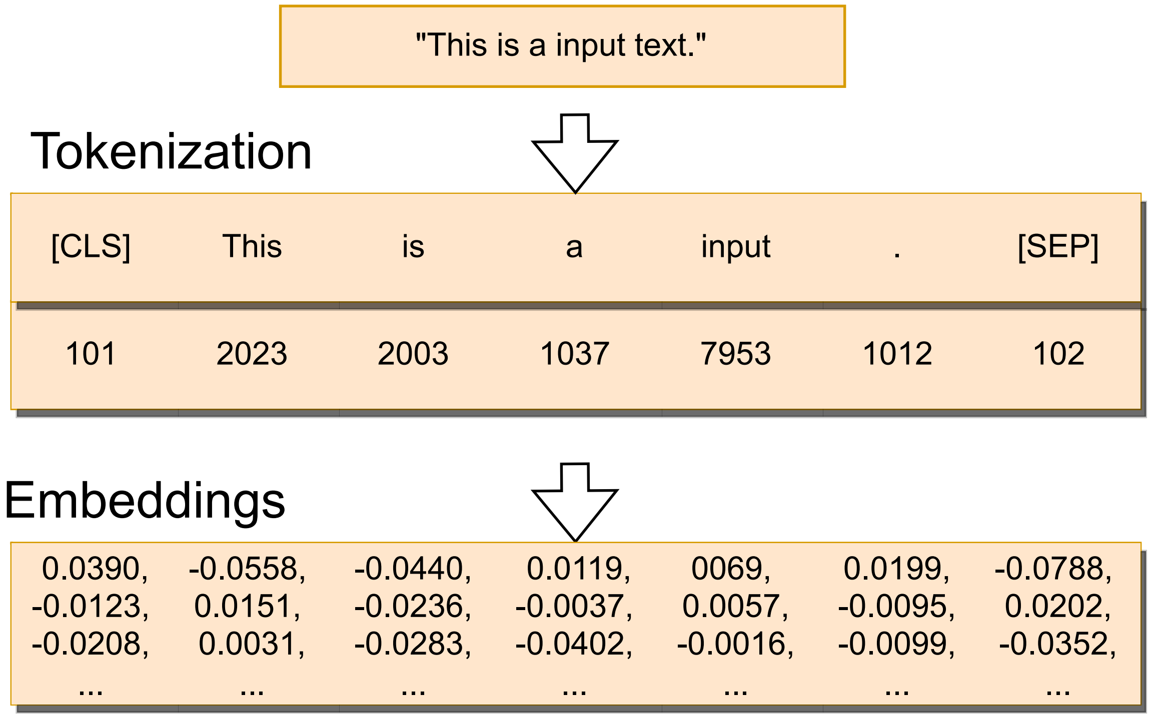
Both the encoder and decode blocks are repeated N times. In the original paper, they defined $N = 6$, and we will define a similar value in this notebook.

Input Embeddings

When we observe the Transformer architecture image above, we can see that the Embeddings represent the first step of both blocks.

The `InputEmbedding` class below is responsible for converting the input text into numerical vectors of `d_model` dimensions. To prevent that our input embeddings become extremely small, we normalize them by multiplying them by the $\sqrt{d_{model}}$.

In the image below, we can see how the embeddings are created. First, we have a sentence that gets split into tokens—we will explore what tokens are later on—. Then, the token IDs—identification numbers—are transformed into the embeddings, which are high-dimensional vectors.



Source: vaclavkosar.com

In [2]:
```python
# Creating Input Embeddings
class InputEmbeddings(nn.Module):

    def __init__(self, d_model: int, vocab_size: int):
        super().__init__()
        self.d_model = d_model # Dimension of vectors (512)
        self.vocab_size = vocab_size # Size of the vocabulary
        self.embedding = nn.Embedding(vocab_size, d_model) # PyTorch layer that cor

    def forward(self, x):
        return self.embedding(x) * math.sqrt(self.d_model) # Normalizing the variar
```

Positional Encoding

In the original paper, the authors add the positional encodings to the input embeddings at the bottom of both the encoder and decoder blocks so the model can have some information about the relative or absolute position of the tokens in the sequence. The positional encodings have the same dimension d_{model} as the embeddings, so that the two vectors can be summed and we can combine the semantic content from the word embeddings and positional information from the positional encodings.

In the `PositionalEncoding` class below, we will create a matrix of positional encodings `pe` with dimensions `(seq_len, d_model)`. We will start by filling it with 0s.We will then apply the sine function to even indices of the positional encoding matrix while the cosine function is applied to the odd ones.

$$\text{Even Indices } (2i): \quad \text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/}}\right.$$

$$\text{Odd Indices } (2i+1): \quad \text{PE}(\text{pos}, 2i+1) = \cos\left(\right.$$

We apply the sine and cosine functions because it allows the model to determine the position of a word based on the position of other words in the sequence, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}. This happens due to the properties of sine and cosine functions, where a shift in the input results in a predictable change in the output.

In [3]:
```
# Creating the Positional Encoding
class PositionalEncoding(nn.Module):

    def __init__(self, d_model: int, seq_len: int, dropout: float) -> None:
        super().__init__()
        self.d_model = d_model # Dimensionality of the model
        self.seq_len = seq_len # Maximum sequence length
        self.dropout = nn.Dropout(dropout) # Dropout layer to prevent overfitting

        # Creating a positional encoding matrix of shape (seq_len, d_model) filled
        pe = torch.zeros(seq_len, d_model)

        # Creating a tensor representing positions (0 to seq_len - 1)
        position = torch.arange(0, seq_len, dtype = torch.float).unsqueeze(1) # Tr

        # Creating the division term for the positional encoding formula
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(1000

        # Apply sine to even indices in pe
        pe[:, 0::2] = torch.sin(position * div_term)
        # Apply cosine to odd indices in pe
        pe[:, 1::2] = torch.cos(position * div_term)

        # Adding an extra dimension at the beginning of pe matrix for batch handlin
        pe = pe.unsqueeze(0)

        # Registering 'pe' as buffer. Buffer is a tensor not considered as a model
        self.register_buffer('pe', pe)

    def forward(self,x):
        # Addind positional encoding to the input tensor X
```

```
        x = x + (self.pe[:, :x.shape[1], :]).requires_grad_(False)
        return self.dropout(x) # Dropout for regularization
```

Layer Normalization

When we look at the encoder and decoder blocks, we see several normalization layers called ***Add & Norm***.

The `LayerNormalization` class below performs layer normalization on the input data. During its forward pass, we compute the mean and standard deviation of the input data. We then normalize the input data by subtracting the mean and dividing by the standard deviation plus a small number called epsilon to avoid any divisions by zero. This process results in a normalized output with a mean 0 and a standard deviation 1.

We will then scale the normalized output by a learnable parameter `alpha` and add a learnable parameter called `bias`. The training process is responsible for adjusting these parameters. The final result is a layer-normalized tensor, which ensures that the scale of the inputs to layers in the network is consistent.

In [4]:

```
# Creating Layer Normalization
class LayerNormalization(nn.Module):

    def __init__(self, eps: float = 10**-6) -> None: # We define epsilon as 0.00000
        super().__init__()
        self.eps = eps

        # We define alpha as a trainable parameter and initialize it with ones
        self.alpha = nn.Parameter(torch.ones(1)) # One-dimensional tensor that will

        # We define bias as a trainable parameter and initialize it with zeros
        self.bias = nn.Parameter(torch.zeros(1)) # One-dimensional tenso that will

    def forward(self, x):
        mean = x.mean(dim = -1, keepdim = True) # Computing the mean of the input d
        std = x.std(dim = -1, keepdim = True) # Computing the standard deviation of

        # Returning the normalized input
        return self.alpha * (x-mean) / (std + self.eps) + self.bias
```

Feed-Forward Network

In the fully connected feed-forward network, we apply two linear transformations with a ReLU activation in between. We can mathematically represent this operation as:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \qquad (3)$$

W_1 and W_2 are the weights, while b_1 and b_2 are the biases of the two linear transformations.

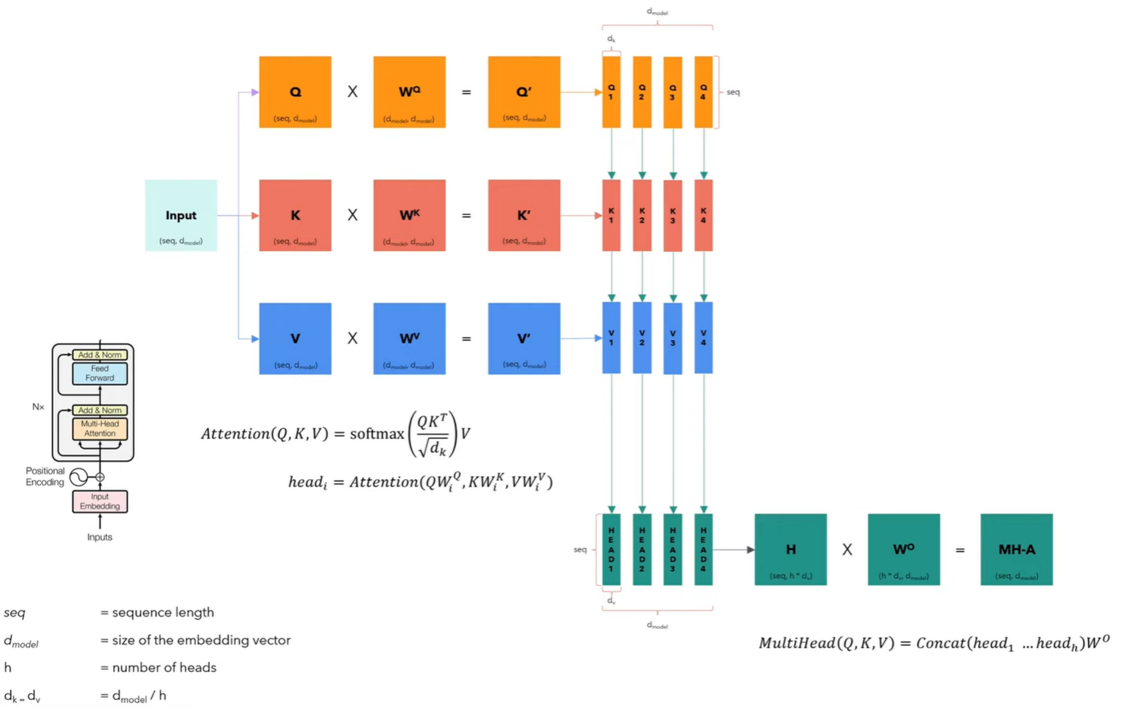
In the `FeedForwardBlock` below, we will define the two linear transformations— `self.linear_1` and `self.linear_2` —and the inner-layer `d_ff`. The input data will first pass through the `self.linear_1` transformation, which increases its dimensionality from `d_model` to `d_ff`. The output of this operation passes through the ReLU activation function, which introduces non-linearity so the network can learn more complex patterns, and the `self.dropout` layer is applied to mitigate overfitting. The final operation is the `self.linear_2` transformation to the dropout-modified tensor, which transforms it back to the original `d_model` dimension.

In [5]:
```
# Creating Feed Forward Layers
class FeedForwardBlock(nn.Module):

    def __init__(self, d_model: int, d_ff: int, dropout: float) -> None:
        super().__init__()
        # First linear transformation
        self.linear_1 = nn.Linear(d_model, d_ff) # W1 & b1
        self.dropout = nn.Dropout(dropout) # Dropout to prevent overfitting
        # Second linear transformation
        self.linear_2 = nn.Linear(d_ff, d_model) # W2 & b2

    def forward(self, x):
        # (Batch, seq_len, d_model) --> (batch, seq_len, d_ff) -->(batch, seq_len,
        return self.linear_2(self.dropout(torch.relu(self.linear_1(x))))
```

Multi-Head Attention

The Multi-Head Attention is the most crucial component of the Transformer. It is responsible for helping the model to understand complex relationships and patterns in the data.

The image below displays how the Multi-Head Attention works. It doesn't include `batch` dimension because it only illustrates the process for one single sentence.



Source: [YouTube: Coding a Transformer from scratch on PyTorch, with full explanation, training and inference] by [Umar Jamil].

The Multi-Head Attention block receives the input data split into queries, keys, and values organized into matrices Q, K, and V. Each matrix contains different facets of the input, and they have the same dimensions as the input.

We then linearly transform each matrix by their respective weight matrices W^Q, W^K, and W^V. These transformations will result in new matrices Q', K', and V', which will be split into smaller matrices corresponding to different heads h, allowing the model to attend to information from different representation subspaces in parallel. This split creates multiple sets of queries, keys, and values for each head.

Finally, we concatenate every head into an H matrix, which is then transformed by another weight matrix W^o to produce the multi-head attention output, a matrix $MH - A$ that retains the input dimensionality.

In [6]:
```
# Creating the Multi-Head Attention block
class MultiHeadAttentionBlock(nn.Module):

    def __init__(self, d_model: int, h: int, dropout: float) -> None: # h = number
        super().__init__()
        self.d_model = d_model
        self.h = h

        # We ensure that the dimensions of the model is divisible by the number of
        assert d_model % h == 0, 'd_model is not divisible by h'
```

```
        # d_k is the dimension of each attention head's key, query, and value vecto
        self.d_k = d_model // h # d_k formula, like in the original "Attention Is A

        # Defining the weight matrices
        self.w_q = nn.Linear(d_model, d_model) # W_q
        self.w_k = nn.Linear(d_model, d_model) # W_k
        self.w_v = nn.Linear(d_model, d_model) # W_v
        self.w_o = nn.Linear(d_model, d_model) # W_o

        self.dropout = nn.Dropout(dropout) # Dropout layer to avoid overfitting


    @staticmethod
    def attention(query, key, value, mask, dropout: nn.Dropout):# mask => When we w

        d_k = query.shape[-1] # The last dimension of query, key, and value

        # We calculate the Attention(Q,K,V) as in the formula in the image above
        attention_scores = (query @ key.transpose(-2,-1)) / math.sqrt(d_k) # @ = Ma

        # Before applying the softmax, we apply the mask to hide some interactions
        if mask is not None: # If a mask IS defined...
            attention_scores.masked_fill_(mask == 0, -1e9) # Replace each value whe
        attention_scores = attention_scores.softmax(dim = -1) # Applying softmax
        if dropout is not None: # If a dropout IS defined...
            attention_scores = dropout(attention_scores) # We apply dropout to prev

        return (attention_scores @ value), attention_scores # Multiply the output m

    def forward(self, q, k, v, mask):

        query = self.w_q(q) # Q' matrix
        key = self.w_k(k) # K' matrix
        value = self.w_v(v) # V' matrix


        # Splitting results into smaller matrices for the different heads
        # Splitting embeddings (third dimension) into h parts
        query = query.view(query.shape[0], query.shape[1], self.h, self.d_k).transp
        key = key.view(key.shape[0], key.shape[1], self.h, self.d_k).transpose(1,2)
        value = value.view(value.shape[0], value.shape[1], self.h, self.d_k).transp

        # Obtaining the output and the attention scores
        x, self.attention_scores = MultiHeadAttentionBlock.attention(query, key, va

        # Obtaining the H matrix
        x = x.transpose(1, 2).contiguous().view(x.shape[0], -1, self.h * self.d_k)

        return self.w_o(x) # Multiply the H matrix by the weight matrix W_o, result
```

Residual Connection

When we look at the architecture of the Transformer, we see that each sub-layer, including the *self-attention* and *Feed Forward* blocks, adds its output to its input before passing it to the *Add & Norm* layer. This approach integrates the output with the original input in the *Add & Norm* layer. This process is known as the skip connection, which allows the

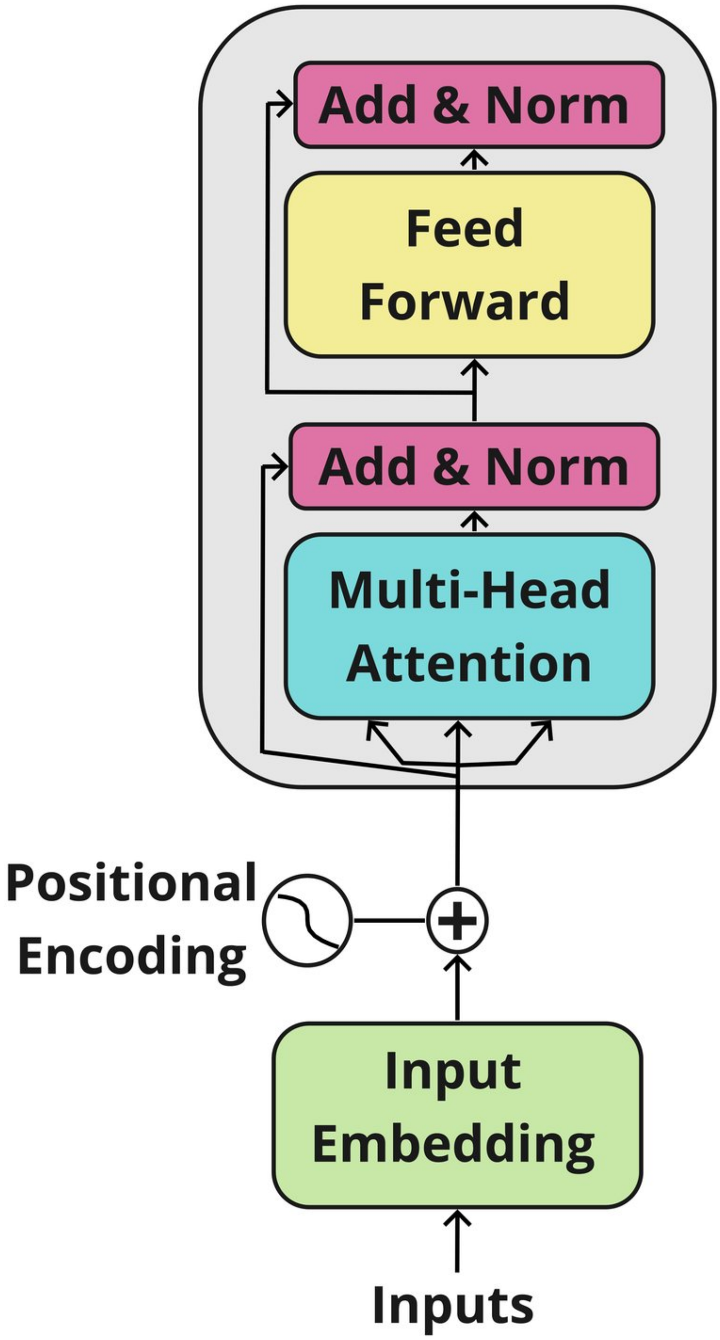
Transformer to train deep networks more effectively by providing a shortcut for the gradient to flow through during backpropagation.

The `ResidualConnection` class below is responsible for this process.

In [7]:
```
# Building Residual Connection
class ResidualConnection(nn.Module):
    def __init__(self, dropout: float) -> None:
        super().__init__()
        self.dropout = nn.Dropout(dropout) # We use a dropout layer to prevent over
        self.norm = LayerNormalization() # We use a normalization layer

    def forward(self, x, sublayer):
        # We normalize the input and add it to the original input 'x'. This creates
        return x + self.dropout(sublayer(self.norm(x)))
```

Encoder

We will now build the encoder. We create the `EncoderBlock` class, consisting of the Multi-Head Attention and Feed Forward layers, plus the residual connections.



Encoder block. Source: researchgate.net.

In the original paper, the Encoder Block repeats six times. We create the `Encoder` class as an assembly of multiple `EncoderBlock` s. We also add layer normalization as a final step after processing the input through all its blocks.

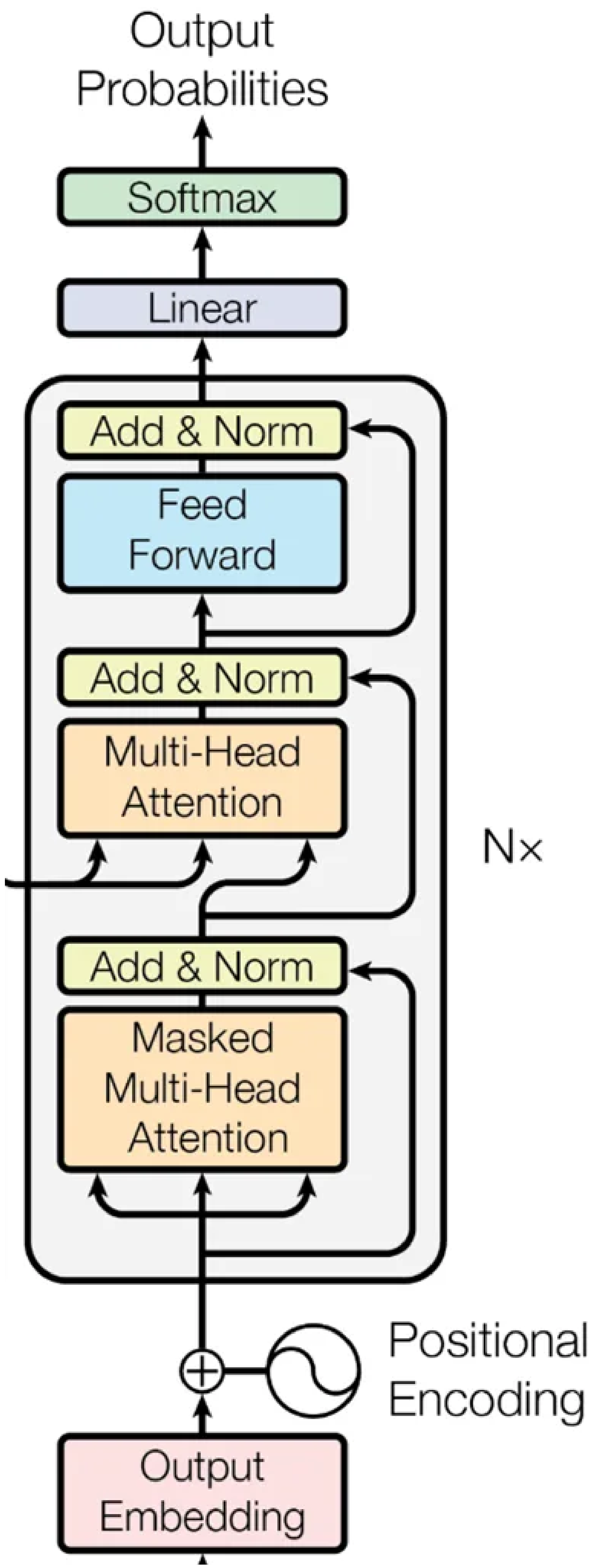
In [8]:
```
# Building Encoder Block
class EncoderBlock(nn.Module):

    # This block takes in the MultiHeadAttentionBlock and FeedForwardBlock, as well
    def __init__(self, self_attention_block: MultiHeadAttentionBlock, feed_forward_
        super().__init__()
        # Storing the self-attention block and feed-forward block
        self.self_attention_block = self_attention_block
```

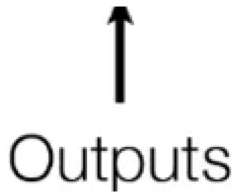
```python
        self.feed_forward_block = feed_forward_block
        self.residual_connections = nn.ModuleList([ResidualConnection(dropout) for 

    def forward(self, x, src_mask):
        # Applying the first residual connection with the self-attention block
        x = self.residual_connections[0](x, lambda x: self.self_attention_block(x, 

        # Applying the second residual connection with the feed-forward block
        x = self.residual_connections[1](x, self.feed_forward_block)
        return x # Output tensor after applying self-attention and feed-forward lay
```

In [9]:
```python
# Building Encoder
# An Encoder can have several Encoder Blocks
class Encoder(nn.Module):

    # The Encoder takes in instances of 'EncoderBlock'
    def __init__(self, layers: nn.ModuleList) -> None:
        super().__init__()
        self.layers = layers # Storing the EncoderBlocks
        self.norm = LayerNormalization() # Layer for the normalization of the outpu

    def forward(self, x, mask):
        # Iterating over each EncoderBlock stored in self.layers
        for layer in self.layers:
            x = layer(x, mask) # Applying each EncoderBlock to the input tensor 'x'
        return self.norm(x) # Normalizing output
```

Decoder

Similarly, the Decoder also consists of several DecoderBlocks that repeat six times in the original paper. The main difference is that it has an additional sub-layer that performs multi-head attention with a *cross-attention* component that uses the output of the Encoder as its keys and values while using the Decoder's input as queries.


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding



Decoder block. Source: edlitera.com.

For the Output Embedding, we can use the same `InputEmbeddings` class we use for the Encoder. You can also notice that the self-attention sub-layer is *masked*, which restricts the model from accessing future elements in the sequence.

We will start by building the `DecoderBlock` class, and then we will build the `Decoder` class, which will assemble multiple `DecoderBlock` s.

In [10]:

```
# Building Decoder Block
class DecoderBlock(nn.Module):

    # The DecoderBlock takes in two MultiHeadAttentionBlock. One is self-attention,
    # It also takes in the feed-forward block and the dropout rate
    def __init__(self,  self_attention_block: MultiHeadAttentionBlock, cross_attent
        super().__init__()
        self.self_attention_block = self_attention_block
        self.cross_attention_block = cross_attention_block
        self.feed_forward_block = feed_forward_block
        self.residual_connections = nn.ModuleList([ResidualConnection(dropout) for

    def forward(self, x, encoder_output, src_mask, tgt_mask):

        # Self-Attention block with query, key, and value plus the target language
        x = self.residual_connections[0](x, lambda x: self.self_attention_block(x,

        # The Cross-Attention block using two 'encoder_ouput's for key and value pl
        x = self.residual_connections[1](x, lambda x: self.cross_attention_block(x,

        # Feed-forward block with residual connections
        x = self.residual_connections[2](x, self.feed_forward_block)
        return x
```

In [11]:

```
# Building Decoder
# A Decoder can have several Decoder Blocks
class Decoder(nn.Module):

    # The Decoder takes in instances of 'DecoderBlock'
    def __init__(self, layers: nn.ModuleList) -> None:
        super().__init__()

        # Storing the 'DecoderBlock's
        self.layers = layers
        self.norm = LayerNormalization() # Layer to normalize the output

    def forward(self, x, encoder_output, src_mask, tgt_mask):

        # Iterating over each DecoderBlock stored in self.layers
        for layer in self.layers:
            # Applies each DecoderBlock to the input 'x' plus the encoder output ar
```

```
            x = layer(x, encoder_output, src_mask, tgt_mask)
        return self.norm(x) # Returns normalized output
```

You can see in the Decoder image that after running a stack of `DecoderBlock` s, we have a Linear Layer and a Softmax function to the output of probabilities. The `ProjectionLayer` class below is responsible for converting the output of the model into a probability distribution over the *vocabulary*, where we select each output token from a vocabulary of possible tokens.

In [12]:
```
# Buiding Linear Layer
class ProjectionLayer(nn.Module):
    def __init__(self, d_model: int, vocab_size: int) -> None: # Model dimension ar
        super().__init__()
        self.proj = nn.Linear(d_model, vocab_size) # Linear layer for projecting th
    def forward(self, x):
        return torch.log_softmax(self.proj(x), dim = -1) # Applying the log Softmax
```

Building the Transformer

We finally have every component of the Transformer architecture ready. We may now construct the Transformer by putting it all together.

In the `Transformer` class below, we will bring together all the components of the model's architecture.

In [13]:
```
# Creating the Transformer Architecture
class Transformer(nn.Module):

    # This takes in the encoder and decoder, as well the embeddings for the source
    # It also takes in the Positional Encoding for the source and target language,
    def __init__(self, encoder: Encoder, decoder: Decoder, src_embed: InputEmbeddir
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.src_embed = src_embed
        self.tgt_embed = tgt_embed
        self.src_pos = src_pos
        self.tgt_pos = tgt_pos
        self.projection_layer = projection_layer

    # Encoder
    def encode(self, src, src_mask):
        src = self.src_embed(src) # Applying source embeddings to the input source
        src = self.src_pos(src) # Applying source positional encoding to the source
        return self.encoder(src, src_mask) # Returning the source embeddings plus c

    # Decoder
    def decode(self, encoder_output, src_mask, tgt, tgt_mask):
        tgt = self.tgt_embed(tgt) # Applying target embeddings to the input target
        tgt = self.tgt_pos(tgt) # Applying target positional encoding to the target

        # Returning the target embeddings, the output of the encoder, and both sour
```

```
        # The target mask ensures that the model won't 'see' future elements of the
        return self.decoder(tgt, encoder_output, src_mask, tgt_mask)

    # Applying Projection Layer with the Softmax function to the Decoder output
    def project(self, x):
        return self.projection_layer(x)
```

The architecture is finally ready. We now define a function called `build_transformer`, in which we define the parameters and everything we need to have a fully operational Transformer model for the task of **machine translation**.

We will set the same parameters as in the original paper, *Attention Is All You Need*, where d_{model} = 512, N = 6, h = 8, dropout rate P_{drop} = 0.1, and d_{ff} = 2048.

In [14]:
```
# Building & Initializing Transformer

# Definin function and its parameter, including model dimension, number of encoder
def build_transformer(src_vocab_size: int, tgt_vocab_size: int, src_seq_len: int, t

    # Creating Embedding layers
    src_embed = InputEmbeddings(d_model, src_vocab_size) # Source language (Source
    tgt_embed = InputEmbeddings(d_model, tgt_vocab_size) # Target language (Target

    # Creating Positional Encoding layers
    src_pos = PositionalEncoding(d_model, src_seq_len, dropout) # Positional encodi
    tgt_pos = PositionalEncoding(d_model, tgt_seq_len, dropout) # Positional encodi

    # Creating EncoderBlocks
    encoder_blocks = [] # Initial list of empty EncoderBlocks
    for _ in range(N): # Iterating 'N' times to create 'N' EncoderBlocks (N = 6)
        encoder_self_attention_block = MultiHeadAttentionBlock(d_model, h, dropout)
        feed_forward_block = FeedForwardBlock(d_model, d_ff, dropout) # FeedForward

        # Combine layers into an EncoderBlock
        encoder_block = EncoderBlock(encoder_self_attention_block, feed_forward_blo
        encoder_blocks.append(encoder_block) # Appending EncoderBlock to the list o

    # Creating DecoderBlocks
    decoder_blocks = [] # Initial list of empty DecoderBlocks
    for _ in range(N): # Iterating 'N' times to create 'N' DecoderBlocks (N = 6)
        decoder_self_attention_block = MultiHeadAttentionBlock(d_model, h, dropout)
        decoder_cross_attention_block = MultiHeadAttentionBlock(d_model, h, dropout
        feed_forward_block = FeedForwardBlock(d_model, d_ff, dropout) # FeedForward

        # Combining layers into a DecoderBlock
        decoder_block = DecoderBlock(decoder_self_attention_block, decoder_cross_at
        decoder_blocks.append(decoder_block) # Appending DecoderBlock to the list o

    # Creating the Encoder and Decoder by using the EncoderBlocks and DecoderBlocks
    encoder = Encoder(nn.ModuleList(encoder_blocks))
    decoder = Decoder(nn.ModuleList(decoder_blocks))

    # Creating projection layer
    projection_layer = ProjectionLayer(d_model, tgt_vocab_size) # Map the output of

    # Creating the transformer by combining everything above
```

```
    transformer = Transformer(encoder, decoder, src_embed, tgt_embed, src_pos, tgt_

    # Initialize the parameters
    for p in transformer.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)

    return transformer # Assembled and initialized Transformer. Ready to be trainec
```

The model is now ready to be trained!

Tokenizer

Tokenization is a crucial preprocessing step for our Transformer model. In this step, we convert raw text into a number format that the model can process.

There are several Tokenization strategies. We will use the *word-level tokenization* to transform each word in a sentence into a token.

Different tokenization strategies. Source: shaankhosla.substack.com.

After tokenizing a sentence, we map each token to an unique integer ID based on the created vocabulary present in the training corpus during the training of the tokenizer. Each integer number represents a specific word in the vocabulary.

Besides the words in the training corpus, Transformers use special tokens for specific purposes. These are some that we will define right away:

- **[UNK]:** This token is used to identify an unknown word in the sequence.

- **[PAD]:** Padding token to ensure that all sequences in a batch have the same length, so we pad shorter sentences with this token. We use attention masks to *"tell"* the model to ignore the padded tokens during training since they don't have any real meaning to the task.

- **[SOS]:** This is a token used to signal the *Start of Sentence*.

- **[EOS]:** This is a token used to signal the *End of Sentence*.

In the `build_tokenizer` function below, we ensure a tokenizer is ready to train the model. It checks if there is an existing tokenizer, and if that is not the case, it trains a new tokenizer.

In [15]:

```
# Defining Tokenizer
def build_tokenizer(config, ds, lang):

    # Crating a file path for the tokenizer
    tokenizer_path = Path(config['tokenizer_file'].format(lang))

    # Checking if Tokenizer already exists
    if not Path.exists(tokenizer_path):

        # If it doesn't exist, we create a new one
        tokenizer = Tokenizer(WordLevel(unk_token = '[UNK]')) # Initializing a new
        tokenizer.pre_tokenizer = Whitespace() # We will split the text into tokens

        # Creating a trainer for the new tokenizer
        trainer = WordLevelTrainer(special_tokens = ["[UNK]", "[PAD]",
                                                     "[SOS]", "[EOS]"], min_frequer

        # Training new tokenizer on sentences from the dataset and language specifi
        tokenizer.train_from_iterator(get_all_sentences(ds, lang), trainer = traine
        tokenizer.save(str(tokenizer_path)) # Saving trained tokenizer to the file
    else:
        tokenizer = Tokenizer.from_file(str(tokenizer_path)) # If the tokenizer alr
    return tokenizer # Returns the loaded tokenizer or the trained tokenizer
```

Loading Dataset

For this task, we will use the OpusBooks dataset, available on 🤗 Hugging Face. This dataset consists of two features, `id` and `translation`. The `translation` feature contains pairs of sentences in different languages, such as Spanish and Portuguese, English and French, and so forth.

I first tried translating sentences from English to Portuguese—my native tongue — but there are only 1.4k examples for this pair, so the results were not satisfying in the current configurations for this model. I then tried to use the English-French pair due to its higher number of examples—127k—but it would take too long to train with the current configurations. I then opted to train the model on the English-Italian pair, the same one used in the Coding a Transformer from scratch on PyTorch, with full explanation, training and inference video, as that was a good balance between performance and time of training.

We start by defining the `get_all_sentences` function to iterate over the dataset and extract the sentences according to the language pair defined —we will do that later.

In [16]:
```
# Iterating through dataset to extract the original sentence and its translation
def get_all_sentences(ds, lang):
    for pair in ds:
        yield pair['translation'][lang]
```

The `get_ds` function is defined to load and prepare the dataset for training and validation. In this function, we build or load the tokenizer, split the dataset, and create DataLoaders, so the model can successfully iterate over the dataset in batches. The result of these functions is tokenizers for the source and target languages plus the DataLoader objects.

In [17]:
```
def get_ds(config):

    # Loading the train portion of the OpusBooks dataset.
    # The Language pairs will be defined in the 'config' dictionary we will build l
    ds_raw = load_dataset('opus_books', f'{config["lang_src"]}-{config["lang_tgt"]}

    # Building or loading tokenizer for both the source and target languages
    tokenizer_src = build_tokenizer(config, ds_raw, config['lang_src'])
    tokenizer_tgt = build_tokenizer(config, ds_raw, config['lang_tgt'])
```

```
    # Splitting the dataset for training and validation
    train_ds_size = int(0.9 * len(ds_raw)) # 90% for training
    val_ds_size = len(ds_raw) - train_ds_size # 10% for validation
    train_ds_raw, val_ds_raw = random_split(ds_raw, [train_ds_size, val_ds_size]) #

    # Processing data with the BilingualDataset class, which we will define below
    train_ds = BilingualDataset(train_ds_raw, tokenizer_src, tokenizer_tgt, config[
    val_ds = BilingualDataset(val_ds_raw, tokenizer_src, tokenizer_tgt, config['lan

    # Iterating over the entire dataset and printing the maximum length found in th
    max_len_src = 0
    max_len_tgt = 0
    for pair in ds_raw:
        src_ids = tokenizer_src.encode(pair['translation'][config['lang_src']]).ids
        tgt_ids = tokenizer_src.encode(pair['translation'][config['lang_tgt']]).ids
        max_len_src = max(max_len_src, len(src_ids))
        max_len_tgt = max(max_len_tgt, len(tgt_ids))

    print(f'Max length of source sentence: {max_len_src}')
    print(f'Max length of target sentence: {max_len_tgt}')

    # Creating dataloaders for the training and validadion sets
    # Dataloaders are used to iterate over the dataset in batches during training a
    train_dataloader = DataLoader(train_ds, batch_size = config['batch_size'], shuf
    val_dataloader = DataLoader(val_ds, batch_size = 1, shuffle = True)

    return train_dataloader, val_dataloader, tokenizer_src, tokenizer_tgt # Returni
```

We define the `casual_mask` function to create a mask for the attention mechanism of the decoder. This mask prevents the model from having information about future elements in the sequence.

We start by making a square grid filled with ones. We determine the grid size with the `size` parameter. Then, we change all the numbers above the main diagonal line to zeros. Every number on one side becomes a zero, while the rest remain ones. The function then flips all these values, turning ones into zeros and zeros into ones. This process is crucial for models that predict future tokens in a sequence.

In [18]:
```
def casual_mask(size):
        # Creating a square matrix of dimensions 'size x size' filled with ones
        mask = torch.triu(torch.ones(1, size, size), diagonal = 1).type(torch.int)
        return mask == 0
```

The `BilingualDataset` class processes the texts of the target and source languages in the dataset by tokenizing them and adding all the necessary special tokens. This class also certifies that the sentences are within a maximum sequence length for both languages and pads all necessary sentences.

In [19]:
```
class BilingualDataset(Dataset):
```

```python
    # This takes in the dataset containg sentence pairs, the tokenizers for target
    # 'seq_len' defines the sequence length for both languages
    def __init__(self, ds, tokenizer_src, tokenizer_tgt, src_lang, tgt_lang, seq_le
        super().__init__()

        self.seq_len = seq_len
        self.ds = ds
        self.tokenizer_src = tokenizer_src
        self.tokenizer_tgt = tokenizer_tgt
        self.src_lang = src_lang
        self.tgt_lang = tgt_lang

        # Defining special tokens by using the target language tokenizer
        self.sos_token = torch.tensor([tokenizer_tgt.token_to_id("[SOS]")], dtype=t
        self.eos_token = torch.tensor([tokenizer_tgt.token_to_id("[EOS]")], dtype=t
        self.pad_token = torch.tensor([tokenizer_tgt.token_to_id("[PAD]")], dtype=t


    # Total number of instances in the dataset (some pairs are larger than others)
    def __len__(self):
        return len(self.ds)

    # Using the index to retrive source and target texts
    def __getitem__(self, index: Any) -> Any:
        src_target_pair = self.ds[index]
        src_text = src_target_pair['translation'][self.src_lang]
        tgt_text = src_target_pair['translation'][self.tgt_lang]

        # Tokenizing source and target texts
        enc_input_tokens = self.tokenizer_src.encode(src_text).ids
        dec_input_tokens = self.tokenizer_tgt.encode(tgt_text).ids

        # Computing how many padding tokens need to be added to the tokenized texts
        # Source tokens
        enc_num_padding_tokens = self.seq_len - len(enc_input_tokens) - 2 # Subtrac
        # Target tokens
        dec_num_padding_tokens = self.seq_len - len(dec_input_tokens) - 1 # Subtrac

        # If the texts exceed the 'seq_len' allowed, it will raise an error. This m
        # given the current sequence length limit (this will be defined in the conf
        if enc_num_padding_tokens < 0 or dec_num_padding_tokens < 0:
            raise ValueError('Sentence is too long')

        # Building the encoder input tensor by combining several elements
        encoder_input = torch.cat(
            [
            self.sos_token, # inserting the '[SOS]' token
            torch.tensor(enc_input_tokens, dtype = torch.int64), # Inserting the to
            self.eos_token, # Inserting the '[EOS]' token
            torch.tensor([self.pad_token] * enc_num_padding_tokens, dtype = torch.i
            ]
        )

        # Building the decoder input tensor by combining several elements
        decoder_input = torch.cat(
            [
                self.sos_token, # inserting the '[SOS]' token
                torch.tensor(dec_input_tokens, dtype = torch.int64), # Inserting th
                torch.tensor([self.pad_token] * dec_num_padding_tokens, dtype = tor
            ]

        )

        # Creating a label tensor, the expected output for training the model
```

```
        label = torch.cat(
            [
                torch.tensor(dec_input_tokens, dtype = torch.int64), # Inserting th
                self.eos_token, # Inserting the '[EOS]' token
                torch.tensor([self.pad_token] * dec_num_padding_tokens, dtype = tor

            ]
        )

        # Ensuring that the length of each tensor above is equal to the defined 'se
        assert encoder_input.size(0) == self.seq_len
        assert decoder_input.size(0) == self.seq_len
        assert label.size(0) == self.seq_len

        return {
            'encoder_input': encoder_input,
            'decoder_input': decoder_input,
            'encoder_mask': (encoder_input != self.pad_token).unsqueeze(0).unsqueez
            'decoder_mask': (decoder_input != self.pad_token).unsqueeze(0).unsqueez
            'label': label,
            'src_text': src_text,
            'tgt_text': tgt_text
        }
```

Validation Loop

We will now create two functions for the validation loop. The validation loop is crucial to evaluate model performance in translating sentences from data it has not seen during training.

We will define two functions. The first function, `greedy_decode`, gives us the model's output by obtaining the most probable next token. The second function, `run_validation`, is responsible for running the validation process in which we decode the model's output and compare it with the reference text for the target sentence.

In [20]:
```
# Define function to obtain the most probable next token
def greedy_decode(model, source, source_mask, tokenizer_src, tokenizer_tgt, max_ler
    # Retrieving the indices from the start and end of sequences of the target toke
    sos_idx = tokenizer_tgt.token_to_id('[SOS]')
    eos_idx = tokenizer_tgt.token_to_id('[EOS]')

    # Computing the output of the encoder for the source sequence
    encoder_output = model.encode(source, source_mask)
    # Initializing the decoder input with the Start of Sentence token
    decoder_input = torch.empty(1,1).fill_(sos_idx).type_as(source).to(device)
```

```python
    # Looping until the 'max_len', maximum length, is reached
    while True:
        if decoder_input.size(1) == max_len:
            break

        # Building a mask for the decoder input
        decoder_mask = casual_mask(decoder_input.size(1)).type_as(source_mask).to(d

        # Calculating the output of the decoder
        out = model.decode(encoder_output, source_mask, decoder_input, decoder_mask

        # Applying the projection layer to get the probabilities for the next token
        prob = model.project(out[:, -1])

        # Selecting token with the highest probability
        _, next_word = torch.max(prob, dim=1)
        decoder_input = torch.cat([decoder_input, torch.empty(1,1). type_as(source)

        # If the next token is an End of Sentence token, we finish the loop
        if next_word == eos_idx:
            break

    return decoder_input.squeeze(0) # Sequence of tokens generated by the decoder
```

In [21]:
```python
# Defining function to evaluate the model on the validation dataset
# num_examples = 2, two examples per run
def run_validation(model, validation_ds, tokenizer_src, tokenizer_tgt, max_len, dev
    model.eval() # Setting model to evaluation mode
    count = 0 # Initializing counter to keep track of how many examples have been p

    console_width = 80 # Fixed witdh for printed messages

    # Creating evaluation loop
    with torch.no_grad(): # Ensuring that no gradients are computed during this pro
        for batch in validation_ds:
            count += 1
            encoder_input = batch['encoder_input'].to(device)
            encoder_mask = batch['encoder_mask'].to(device)

            # Ensuring that the batch_size of the validation set is 1
            assert encoder_input.size(0) ==  1, 'Batch size must be 1 for validatio

            # Applying the 'greedy_decode' function to get the model's output for t
            model_out = greedy_decode(model, encoder_input, encoder_mask, tokenizer

            # Retrieving source and target texts from the batch
            source_text = batch['src_text'][0]
            target_text = batch['tgt_text'][0] # True translation
            model_out_text = tokenizer_tgt.decode(model_out.detach().cpu().numpy())

            # Printing results
            print_msg('-'*console_width)
            print_msg(f'SOURCE: {source_text}')
            print_msg(f'TARGET: {target_text}')
            print_msg(f'PREDICTED: {model_out_text}')

            # After two examples, we break the loop
            if count == num_examples:
                break
```

Training Loop

We are ready to train our Transformer model on the OpusBook dataset for the English to Italian translation task.

We first start by defining the `get_model` function to load the model by calling the `build_transformer` function we have previously defined. This function uses the `config` dictionary to set a few parameters.

In [22]:
```
# We pass as parameters the config dictionary, the length of the vocabylary of the
def get_model(config, vocab_src_len, vocab_tgt_len):

    # Loading model using the 'build_transformer' function.
    # We will use the lengths of the source language and target language vocabulari
    model = build_transformer(vocab_src_len, vocab_tgt_len, config['seq_len'], conf
    return model
```

I have mentioned the `config` dictionary several times throughout this notebook. Now, it is time to create it.

In the following cell, we will define two functions to configure our model and the training process.

In the `get_config` function, we define crucial parameters for the training process. `batch_size` for the number of training examples used in one iteration, `num_epochs` as the number of times the entire dataset is passed forward and backward through the Transformer, `lr` as the learning rate for the optimizer, etc. We will also finally define the pairs from the OpusBook dataset, `'lang_src': 'en'` for selecting English as the source language and `'lang_tgt': 'it'` for selecting Italian as the target language.

The `get_weights_file_path` function constructs the file path for saving or loading model weights for any specific epoch.

In [23]:
```
# Define settings for building and training the transformer model
def get_config():
    return{
```

```python
        'batch_size': 8,
        'num_epochs': 20,
        'lr': 10**-4,
        'seq_len': 350,
        'd_model': 512, # Dimensions of the embeddings in the Transformer. 512 like
        'lang_src': 'en',
        'lang_tgt': 'it',
        'model_folder': 'weights',
        'model_basename': 'tmodel_',
        'preload': None,
        'tokenizer_file': 'tokenizer_{0}.json',
        'experiment_name': 'runs/tmodel'
    }


# Function to construct the path for saving and retrieving model weights
def get_weights_file_path(config, epoch: str):
    model_folder = config['model_folder'] # Extracting model folder from the config
    model_basename = config['model_basename'] # Extracting the base name for model
    model_filename = f"{model_basename}{epoch}.pt" # Building filename
    return str(Path('.')/ model_folder/ model_filename) # Combining current directo
```

We finally define our last function, `train_model`, which takes the `config` arguments as input.

In this function, we will set everything up for the training. We will load the model and its necessary components onto the GPU for faster training, set the `Adam` optimizer, and configure the `CrossEntropyLoss` function to compute the differences between the translations output by the model and the reference translations from the dataset.

Every loop necessary for iterating over the training batches, performing backpropagation, and computing the gradients is in this function. We will also use it to run the validation function and save the current state of the model.

In [24]:
```python
def train_model(config):
    # Setting up device to run on GPU to train faster
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"Using device {device}")

    # Creating model directory to store weights
    Path(config['model_folder']).mkdir(parents=True, exist_ok=True)

    # Retrieving dataloaders and tokenizers for source and target languages using t
    train_dataloader, val_dataloader, tokenizer_src, tokenizer_tgt = get_ds(config)

    # Initializing model on the GPU using the 'get_model' function
    model = get_model(config,tokenizer_src.get_vocab_size(), tokenizer_tgt.get_voca

    # Tensorboard
    writer = SummaryWriter(config['experiment_name'])

    # Setting up the Adam optimizer with the specified learning rate from the '
    # config' dictionary plus an epsilon value
```

```
    optimizer = torch.optim.Adam(model.parameters(), lr=config['lr'], eps = 1e-9)

    # Initializing epoch and global step variables
    initial_epoch = 0
    global_step = 0

    # Checking if there is a pre-trained model to load
    # If true, loads it
    if config['preload']:
        model_filename = get_weights_file_path(config, config['preload'])
        print(f'Preloading model {model_filename}')
        state = torch.load(model_filename) # Loading model

        # Sets epoch to the saved in the state plus one, to resume from where it st
        initial_epoch = state['epoch'] + 1
        # Loading the optimizer state from the saved model
        optimizer.load_state_dict(state['optimizer_state_dict'])
        # Loading the global step state from the saved model
        global_step = state['global_step']

    # Initializing CrossEntropyLoss function for training
    # We ignore padding tokens when computing loss, as they are not relevant for th
    # We also apply label_smoothing to prevent overfitting
    loss_fn = nn.CrossEntropyLoss(ignore_index = tokenizer_src.token_to_id('[PAD]')

    # Initializing training loop

    # Iterating over each epoch from the 'initial_epoch' variable up to
    # the number of epochs informed in the config
    for epoch in range(initial_epoch, config['num_epochs']):

        # Initializing an iterator over the training dataloader
        # We also use tqdm to display a progress bar
        batch_iterator = tqdm(train_dataloader, desc = f'Processing epoch {epoch:02

        # For each batch...
        for batch in batch_iterator:
            model.train() # Train the model

            # Loading input data and masks onto the GPU
            encoder_input = batch['encoder_input'].to(device)
            decoder_input = batch['decoder_input'].to(device)
            encoder_mask = batch['encoder_mask'].to(device)
            decoder_mask = batch['decoder_mask'].to(device)

            # Running tensors through the Transformer
            encoder_output = model.encode(encoder_input, encoder_mask)
            decoder_output = model.decode(encoder_output, encoder_mask, decoder_inp
            proj_output = model.project(decoder_output)

            # Loading the target labels onto the GPU
            label = batch['label'].to(device)

            # Computing loss between model's output and true labels
            loss = loss_fn(proj_output.view(-1, tokenizer_tgt.get_vocab_size()), la

            # Updating progress bar
            batch_iterator.set_postfix({f"loss": f"{loss.item():6.3f}"})

            writer.add_scalar('train loss', loss.item(), global_step)
            writer.flush()

            # Performing backpropagation
            loss.backward()
```

```
            # Updating parameters based on the gradients
            optimizer.step()

            # Clearing the gradients to prepare for the next batch
            optimizer.zero_grad()

            global_step += 1 # Updating global step count

        # We run the 'run_validation' function at the end of each epoch
        # to evaluate model performance
        run_validation(model, val_dataloader, tokenizer_src, tokenizer_tgt, config[

        # Saving model
        model_filename = get_weights_file_path(config, f'{epoch:02d}')
        # Writting current model state to the 'model_filename'
        torch.save({
            'epoch': epoch, # Current epoch
            'model_state_dict': model.state_dict(),# Current model state
            'optimizer_state_dict': optimizer.state_dict(), # Current optimizer sta
            'global_step': global_step # Current global step
        }, model_filename)
```

We can now train the model!

In [25]:
```
if __name__ == '__main__':
    warnings.filterwarnings('ignore') # Filtering warnings
    config = get_config() # Retrieving config settings
    train_model(config) # Training model with the config arguments
```

```
Using device cuda
Downloading builder script:   0%|          | 0.00/2.40k [00:00<?, ?B/s]
Downloading metadata:   0%|          | 0.00/7.98k [00:00<?, ?B/s]
Downloading and preparing dataset opus_books/en-it (download: 3.14 MiB, generated:
8.58 MiB, post-processed: Unknown size, total: 11.72 MiB) to /root/.cache/huggingf
ace/datasets/opus_books/en-it/1.0.0/e8f950a4f32dc39b7f9088908216cd2d7e21ac35f893d0
4d39eb594746af2daf...
Downloading data:   0%|          | 0.00/3.30M [00:00<?, ?B/s]
Generating train split:   0%|          | 0/32332 [00:00<?, ? examples/s]
Dataset opus_books downloaded and prepared to /root/.cache/huggingface/datasets/op
us_books/en-it/1.0.0/e8f950a4f32dc39b7f9088908216cd2d7e21ac35f893d04d39eb594746af2
daf. Subsequent calls will reuse this data.
Max length of source sentence: 309
Max length of target sentence: 274
Processing epoch 00: 100%|██████████| 3638/3638 [15:31<00:00,  3.91it/s, loss=5.92
6]
--------------------------------------------------------------------------------
SOURCE: 'All right,' said the Englishman. 'And where are you going, my lord?' he a
sked unexpectedly, addressing him as 'my lord,' which he hardly ever did.
TARGET: — Tutto è in ordine — disse l'inglese. — E voi dove andate, milord? — chie
se inaspettatamente, adoperando questa denominazione di my-Lord che non usava quas
i mai.
PREDICTED: — Sì , — disse il signor Rochester , — disse il signor Rochester , — e
che vi , — e che vi il signor Rochester .
--------------------------------------------------------------------------------
SOURCE: "Yes, yes," says he, "you teachee me good, you teachee them good." "No, n
o, Friday," says I, "you shall go without me; leave me here to live by myself, as
I did before."
TARGET: Voi aver insegnato me il bene; insegnare il bene loro! — No, no, Venerdì;
andrete senza di me; lasciatemi vivere qui solo, come ho fatto in passato.»
PREDICTED: E che è un ' altra cosa , Jane , e vi è un ' altra , che vi è un ' altr
a cosa , che non vi è un ' altra cosa , che non vi , che vi , che vi , e che vi ,
che vi .
```

```
Processing epoch 01: 100%|██████████| 3638/3638 [15:32<00:00,  3.90it/s, loss=3.43
7]
--------------------------------------------------------------------------------
SOURCE: In the first place, it occurred to me to consider what business an English
ship could have in that part of the world, since it was not the way to or from any
part of the world where the English had any traffic; and I knew there had been no
storms to drive them in there in distress; and that if they were really English it
was most probable that they were here upon no good design; and that I had better c
ontinue as I was than fall into the hands of thieves and murderers.
TARGET: Prima di tutto andava ruminando in mia testa, qual razza di faccende potes
se condurre una nave inglese in questa parte del mondo, ove, nè andando nè tornand
o, gl’Inglesi non avevano alcuna sorta di traffico. Sapeva d’altra parte non esser
e occorse burrasche o altri disastri di mare che li avessero potuto costringere a
cercar quivi un riparo; dalle quali cose argomentava che se erano Inglesi, probabi
lmente non erano qui con buon disegno, e che valea meglio per me il continuare nel
la vita di prima, che cadere in mano di ladri o d’assassini.
PREDICTED: In quel momento , per me , perchè non mi , perchè non poteva essere più
di cui non poteva essere in tal modo di cui non poteva essere di non , e di non ,
e di non , e di non , e di non , e di non , e di non , e di non , e di non , e di
non .
--------------------------------------------------------------------------------
SOURCE: And here I am living; my children growing, my husband returns to the famil
y and feels his error, grows purer and better, and I live...
TARGET: Ed ecco, io vivo. I bambini crescono, mio marito ritorna in famiglia e sen
te il torto suo e diventa sempre migliore, e io vivo....
PREDICTED: E io sono , e a me , e con la sua vita , e la sua vita , e .
Processing epoch 02: 100%|██████████| 3638/3638 [15:33<00:00,  3.90it/s, loss=5.41
0]
--------------------------------------------------------------------------------
SOURCE: 'I wonder what they'll do next!
TARGET: Chi sa che faranno dopo!
PREDICTED: — Io , come sono in modo di fare !
--------------------------------------------------------------------------------
SOURCE: He thought himself her idol, ugly as he was: he believed, as he said, that
she preferred his "_taille d'athlete_" to the elegance of the Apollo Belvidere.
TARGET: Così egli, benché brutto, si credeva adorato e credeva che la giovane pref
erisse la sua figura di atleta all'eleganza dell'Apollo di Belvedere.
PREDICTED: Egli pensava che la sua domanda era stata , e disse che era stata così
come se fosse stata la sua posizione , — disse il suo carattere . — La la della su
a vita .
Processing epoch 03: 100%|██████████| 3638/3638 [15:33<00:00,  3.90it/s, loss=3.84
8]
--------------------------------------------------------------------------------
SOURCE: "What's what?" asked Harris and I.
TARGET: — Che cosa c’è? — domandammo Harris e io.
PREDICTED: — Che cosa ? — domandai Harris .
--------------------------------------------------------------------------------
SOURCE: If she did, she need not coin her smiles so lavishly, flash her glances so
unremittingly, manufacture airs so elaborate, graces so multitudinous. It seems to
me that she might, by merely sitting quietly at his side, saying little and lookin
g less, get nigher his heart.
TARGET: Mi pare che le basterebbe di sedersi tranquillamente accanto a lui, di par
lar poco e di guardarlo anche meno, e giungerebbe al suo cuore.
PREDICTED: Se non avesse bisogno di lei , non si , e , , , , , , , , , , , , , , ,
, , , , , , e , , , , , , la sua gioia .
Processing epoch 04: 100%|██████████| 3638/3638 [15:30<00:00,  3.91it/s, loss=3.80
7]
```

```
--------------------------------------------------------------------------------
SOURCE: CHAPTER XXVIII
TARGET: VIII.
PREDICTED: XXVIII
--------------------------------------------------------------------------------
SOURCE: Cooler and fresher at the moment the gale seemed to visit my brow: I could
have deemed that in some wild, lone scene, I and Jane were meeting.
TARGET: "In quel momento una brezza più fresca mi sfiorò la fronte. "Avrei potuto
credere che Jane ed io ci fossimo incontrati in qualche luogo deserto.
PREDICTED: " e la notte , perché la mia presenza mi , e che avevo veduto la mia ca
sa , Jane .
Processing epoch 05: 100%|██████████| 3638/3638 [15:28<00:00,  3.92it/s, loss=3.92
1]
--------------------------------------------------------------------------------
SOURCE: 'Call it what you like,' said the Cat. 'Do you play croquet with the Queen
to-day?'
TARGET: — Di' come ti pare, — rispose il Gatto. — Vai oggi dalla Regina a giocare
a croquet?
PREDICTED: — Forse voi , — disse il Gatto , — ti prego di di con la Regina ?
--------------------------------------------------------------------------------
SOURCE: The Dormouse shook its head impatiently, and said, without opening its eye
s, 'Of course, of course; just what I was going to remark myself.'
TARGET: Il Ghiro scosse la testa con atto d'impazienza, e senza aprire gli occhi d
isse: — Già! Già! stavo per dirlo io.
PREDICTED: Il Ghiro prese il capo e il capo , senza capire , si voltò verso di lui
, ma egli mi fece dire : “ Non ho detto che cosa ho detto ”.
Processing epoch 06: 100%|██████████| 3638/3638 [15:30<00:00,  3.91it/s, loss=3.39
5]
--------------------------------------------------------------------------------
SOURCE: When she got up, the previous day appeared in her memory as in a fog.
TARGET: Quando si fu alzata, le venne in mente, come in una nebbia, la giornata pr
ecedente.
PREDICTED: Quando si svegliò , la giornata si calmò in una nebbia , la nebbia come
una nebbia .
--------------------------------------------------------------------------------
SOURCE: 'Is it long since you went to see them?'
TARGET: — È da molto che manchi da loro?
PREDICTED: — È arrivato da tempo , da voi ?
Processing epoch 07: 100%|██████████| 3638/3638 [15:30<00:00,  3.91it/s, loss=3.92
7]
--------------------------------------------------------------------------------
SOURCE: "To be active: as active as I can.
TARGET: — Voglio essere operosa per quanto è possibile.
PREDICTED: — È strano , come posso .
--------------------------------------------------------------------------------
SOURCE: It is a veritable picture of an old country inn, with green, square courty
ard in front, where, on seats beneath the trees, the old men group of an evening t
o drink their ale and gossip over village politics; with low, quaint rooms and lat
ticed windows, and awkward stairs and winding passages.
TARGET: È un vero quadro d’un vecchio albergo di campagna, con un verde cortile qu
adrato, dove, sui sedili sotto gli alberi, i vecchi si riuniscono la sera a bere l
a birra e a discutere della politica paesana; con stanze, bizzarre camere e finest
re ingraticciate, e delle scale malcomode e dei corridoi tortuosi.
PREDICTED: È un quadro di campagna , con un ufficiale che si era messo in piedi ,
con la sua costruzione , dove si , il verde della città e il di , il dei , dei , i
di e i di , dei , dei e dei , e i e dei .
Processing epoch 08: 100%|██████████| 3638/3638 [15:30<00:00,  3.91it/s, loss=3.42
0]
```

```
--------------------------------------------------------------------------------
SOURCE: How shall I do it?' he asked himself, trying to find expression for what h
e had been thinking and the feelings he had lived through in that short night.
TARGET: Come farò tutto questo?» si chiese, cercando di esprimere a se stesso ciò
che aveva pensato e sentito in quella breve nottata.
PREDICTED: Come farò ? — chiese , cercando di capire , cercando di capire quello c
he era stato accaduto e che egli aveva sempre provato in quella notte .
--------------------------------------------------------------------------------
SOURCE: Having invited Helen and me to approach the table, and placed before each
of us a cup of tea with one delicious but thin morsel of toast, she got up, unlock
ed a drawer, and taking from it a parcel wrapped in paper, disclosed presently to
our eyes a good-sized seed-cake. "I meant to give each of you some of this to take
with you," said she, "but as there is so little toast, you must have it now," and
she proceeded to cut slices with a generous hand.
TARGET: Ella invitò Elena e me ad avvicinarci alla tavola, collocò dinanzi a noi l
e tazze e i crostini, poi tolse da un cassetto un maestoso pan pepato, ravvolto co
n cura, e la sua mano generosa ce ne tagliò delle fette grosse.
PREDICTED: Dopo aver Elena e mi la tavola , e poi ci si avvicinò al tè , ma con un
bimbo , che conteneva un bimbo , e poi , poi un cassetto , che , un pezzetto di ca
rta , disse , dicendo che non era altro che qualcuno di , — come un , — e quando c
i a destra , — e a destra , — ecco che ci sia qualche cosa che ci sia un po ' di p
ane , con un , e , — disse , — ma che ci a destra .
Processing epoch 09: 100%|██████████| 3638/3638 [15:34<00:00,  3.89it/s, loss=2.90
3]
--------------------------------------------------------------------------------
SOURCE: I shall not stay long at Morton, now that my father is dead, and that I am
my own master.
TARGET: Non rimarrò lungamente a Morton ora che mio padre è morto e che son padron
e delle mie azioni.
PREDICTED: Non tornerò più a Morton , che è morta , e il mio padrone è morto e son
o il mio padrone .
--------------------------------------------------------------------------------
SOURCE: And in examining their actions and lives one cannot see that they owed any
thing to fortune beyond opportunity, which brought them the material to mould into
the form which seemed best to them. Without that opportunity their powers of mind
would have been extinguished, and without those powers the opportunity would have
come in vain.
TARGET: Et esaminando le azioni e vita loro, non si vede che quelli avessino altro
dalla fortuna che la occasione; la quale dette loro materia a potere introdurvi dr
ento quella forma parse loro; e sanza quella occasione la virtù dello animo loro s
i sarebbe spenta, e sanza quella virtù la occasione sarebbe venuta invano.
PREDICTED: E , in quel modo le cose e la vita non sanno che la fortuna di questa o
ccasione si per acquistare tutto il bene che li assai bene , e che il loro fine er
a stato spento , e che il loro fine non sarebbe stato necessario avere .
Processing epoch 10: 100%|██████████| 3638/3638 [15:34<00:00,  3.89it/s, loss=3.73
4]
--------------------------------------------------------------------------------
SOURCE: "Our uncle John is dead," said he.
TARGET: Egli entrò dicendo: — Lo zio John è morto.
PREDICTED: — Mio zio John è morto , — disse .
--------------------------------------------------------------------------------
SOURCE: All that does not matter.
TARGET: Questo non significa nulla.
PREDICTED: Tutto questo non lo conosce .
Processing epoch 11: 100%|██████████| 3638/3638 [15:33<00:00,  3.90it/s, loss=3.05
5]
```

```
--------------------------------------------------------------------------------
SOURCE: Mrs. P. used to come up and say she was very sorry - for herself, she liked to hear him - but the lady upstairs was in a very delicate state, and the doctor was afraid it might injure the child.
TARGET: La signora Poppets soleva presentarsi a dire che le dispiaceva moltissimo — quanto a lei andava matta per la musica — ma la signora di sopra era in istato interessante, e il dottore temeva che quel suono potesse nuocere al bambino.
PREDICTED: La signora Poppets ci si coricò e cominciò a parlare con lei , perché gli piaceva sentire una donna magra e di , ma che il dottore era in un abisso , e il bambino non poteva sopportare , che il bambino gli fosse apparso con l ’ aiuto del bambino .
--------------------------------------------------------------------------------
SOURCE: "A true Janian reply! Good angels be my guard!
TARGET: — È una risposta degna di Jane!
PREDICTED: — Un vero ! — rispose Bianca , sorridendo .
Processing epoch 12: 100%|██████████| 3638/3638 [15:34<00:00,  3.89it/s, loss=2.740]
--------------------------------------------------------------------------------
SOURCE: If you had seen her as I have who have spent the whole winter with her, you would pity her.
TARGET: Se tu la vedessi come l’ho vista io (ho passato tutto l’inverno con lei), ne avresti pena.
PREDICTED: Se aveste visto come è andata a finire tutta la giornata , con lei , la avrebbe fatto pena .
--------------------------------------------------------------------------------
SOURCE: Anna had come out from behind the screen to meet him, and Levin saw in the dim light of the study the woman of the portrait, in a dark dress of different shades of blue, not in the same attitude, not with the same expression, but on the same height of beauty as that on which the artist had caught her in the portrait.
TARGET: Anna gli era uscita incontro di là dalla grata e Levin vide, nella penombra dello studio, quella stessa donna del ritratto, in abito scuro d’un turchino cangiante, non nella posa, non con l’espressione, ma della stessa bellezza con cui era stata colta dall’artista nel ritratto.
PREDICTED: Anna era uscito dal tramezzo con l ’ aiuto della donna e vide Levin nello studio del ritratto di lei che in un vestito di merletto in un vestito di merletto e senza le sue maniere , ma che in quel suo ritratto non era nel ritratto affatto chiaro che l ’ artista sul suo quadro .
Processing epoch 13: 100%|██████████| 3638/3638 [15:35<00:00,  3.89it/s, loss=2.713]
--------------------------------------------------------------------------------
SOURCE: Yesterday he betrayed himself – he wants the divorce and a marriage in order to burn his boats.
TARGET: Ieri se l’è lasciato sfuggire: vuole il divorzio e il matrimonio per bruciare le sue navi.
PREDICTED: La sua bontà ha voluto anche lui , per ottenere un divorzio e fare delle sue barche .
--------------------------------------------------------------------------------
SOURCE: I know nothing, I understand nothing.'
TARGET: Io non so nulla e non capisco nulla.
PREDICTED: Io non capisco nulla .
Processing epoch 14: 100%|██████████| 3638/3638 [15:35<00:00,  3.89it/s, loss=2.577]
```

```
--------------------------------------------------------------------------------
SOURCE: "Well, Jane Eyre, and are you a good child?"
TARGET: — Ebbene, Jane Eyre, siete una buona bambina?
PREDICTED: — Ebbene , Jane Eyre e siete una bambina ?
--------------------------------------------------------------------------------
SOURCE: 'Come, Anna Arkadyevna,' began Korsunsky, drawing her bare arm under his,
'I have such a good idea for a cotillion - Un bijou.'
TARGET: — Su via, Anna Arkad'evna — prese a dire Korsunskij, mettendo il braccio n
udo di lei sotto la manica del suo frac. — Che idea mi è venuta per il cotillon! U
n bijou!
PREDICTED: — Andiamo , Anna Arkad ' evna — disse Korsunskij , alzandosi sotto il b
raccio . — Ho un gran pensiero per una parte di , un , per la .
Processing epoch 15: 100%|██████████| 3638/3638 [15:34<00:00,  3.89it/s, loss=2.61
8]
--------------------------------------------------------------------------------
SOURCE: 'Gentlemen! To-morrow at dawn!' Levin mumbled drowsily, and fell asleep.
TARGET: — Signori, a domani, appena si fa giorno! — e s'addormentò.
PREDICTED: — , l ' alba — disse Levin , in fretta .
--------------------------------------------------------------------------------
SOURCE: But after that hour another passed, a second, a third, and all the five ho
urs that he had set himself as the longest term of possible endurance, and still t
he situation was unchanged; and he went on enduring, for there was nothing else to
do but to endure - thinking every moment that he had reached the utmost limit of e
ndurance and that in a moment his heart would burst with pity.
TARGET: Ed era passata soltanto un'ora. Ma dopo quest'ora, ne passò ancora un'altr
a, poi ne passarono due, tre, tutte e cinque le ore, e le cose erano sempre allo s
tesso punto; e lui sopportava ancora, perché non c'era più niente da fare se non p
azientare, pensando, ogni momento, d'essere giunto al limite della sopportazione e
che il cuore, subito, da un momento all'altro, si sarebbe spezzato dalla pena.
PREDICTED: Ma dopo un ' ora passò un ' altra , e tutto il terzo piano , e tutti i
campi i campi i capi , che aveva potuto ancora peggio ; e la situazione tutta nell
a situazione non c ' era nessun altro che , non desiderando che nulla di meglio ,
e si provava in un momento che tutti gli stati il cuore e si sarebbe il cuore di f
orza e si sarebbe il cuore .
Processing epoch 16: 100%|██████████| 3638/3638 [15:35<00:00,  3.89it/s, loss=2.18
9]
```

```
--------------------------------------------------------------------------------
SOURCE: Meeting his look, her face suddenly assumed a coldly severe expression, as
if to say: 'It is not forgotten.
TARGET: Nell’incontrare lo sguardo di lui, il viso di Anna, d’un tratto, prese u
n’espressione dura, come a dirgli: “Non è dimenticato.
PREDICTED: L ’ espressione del viso di lei , un tratto di fredda , come se volesse
dire , non è dimenticato .
--------------------------------------------------------------------------------
SOURCE: I asked him all the particulars of their voyage, and found they were a Spa
nish ship, bound from the Rio de la Plata to the Havanna, being directed to leave
their loading there, which was chiefly hides and silver, and to bring back what Eu
ropean goods they could meet with there; that they had five Portuguese seamen on b
oard, whom they took out of another wreck; that five of their own men were drowned
when first the ship was lost, and that these escaped through infinite dangers and
hazards, and arrived, almost starved, on the cannibal coast, where they expected t
o have been devoured every moment.
TARGET: Interrogato da me su i particolari del suo viaggio, mi raccontò come avess
e fatto parte de’ naviganti d’un vascello spagnuolo che veniva dal Rio la Plata pe
r condursi all’Avana a lasciare ivi il loro carico, consistente principalmente in
pellami o argento, e riportarne quelle merci pregiate in Europa in cui si sarebber
o abbattuti; come avessero preso a bordo cinque marinai portoghesi salvatisi da un
altro naufragio; come cinque de’ loro fossero rimasi annegati quando il loro vasce
llo perì; come campati in mezzo ad infiniti pericoli e traversie fossero arrivati
quasi morti di fame ad una costa di cannibali, ove si aspettavano a ciascun istant
e di essere divorati.
PREDICTED: tutte le predette cose non erano buone , e compresi che fosse alquanto
, una nave che venne ad esse dal vascello le la paura de ’ marinai , che si per po
tere trovare il vascello , e che si per non essere a furia di migliaia d ’ uomini
in mezzo : quando quando quando avessero abbandonato il vascello si , fu veduto co
me ombre loro , si , e quando tutti quegli sciagurati a levante sulla costa della
spiaggia , e donde sarebbero stati trasportati a quando sarebbero stati e venuti ,
e altre che sarebbero rimasti inevitabilmente .
Processing epoch 17: 100%|██████████| 3638/3638 [15:34<00:00,  3.89it/s, loss=2.66
2]
--------------------------------------------------------------------------------
SOURCE: "Where the devil is Rochester?" cried Colonel Dent.
TARGET: — Dove diavolo è Rochester? — esclamò il colonnello Dent.
PREDICTED: — Dov ' è il diavolo ? — esclamò il colonnello Dent e il colonnello Den
t .
--------------------------------------------------------------------------------
SOURCE: "Yes, she is alive; and more sensible and collected than she was.
TARGET: — Sì, vive, ma da ieri non è più in sé.
PREDICTED: — Sì , lei è più viva e più nervosa di lei .
Processing epoch 18: 100%|██████████| 3638/3638 [15:34<00:00,  3.89it/s, loss=2.15
2]
```

```
--------------------------------------------------------------------------------
SOURCE: I asked the captain if he was willing to venture with these hands on board
the ship; but as for me and my man Friday, I did not think it was proper for us to
stir, having seven men left behind; and it was employment enough for us to keep th
em asunder, and supply them with victuals.
TARGET: Chiesi al capitano s’egli credea d’avventurarsi con questa gente all’arrem
baggio del vascello; perchè quanto a me e al servo mio Venerdì, non pensai ne conv
enisse il moverci dall’isola, ove ne rimanevano sette uomini da guardare. Era assa
i briga per noi il tenerli disgiunti e provvedere al giornaliero lor vitto; quanto
ai cinque della caverna, trovai opportuno il lasciarli legati.
PREDICTED: Feci ciò che fu , se il capitano non avesse voluto queste mie mani ; ma
per altro non era facile a Venerdì , nè sapeva solamente colpire soltanto a Venerd
ì di che far la mia preda su le otto uomini , che era divenuto , se non mi riusciv
a a mangiare . , era più allegra o d ’ uomini armati .
--------------------------------------------------------------------------------
SOURCE: I must, then, repeat continually that we are for ever sundered:--and yet,
while I breathe and think, I must love him."
TARGET: Debbo dunque convincermi che saremo separati per sempre, ma che debbo amar
lo per tutta la vita.
PREDICTED: Devo partire perché siamo sempre più gravi , ma sono ritto di fronte a
lui e bisogna .
```

```
Processing epoch 19: 100%|██████████| 3638/3638 [15:33<00:00,  3.90it/s, loss=2.09
4]
```

```
--------------------------------------------------------------------------------
SOURCE: 'Duties to go to a concert...'
TARGET: — Gli obblighi di andare al concerto...
PREDICTED: — Oggi si è stabilito al concerto .
--------------------------------------------------------------------------------
SOURCE: 'What!
TARGET: — Che cosa?
PREDICTED: — Che cosa ?
```

As you can see below, we trained for 20 epochs, and the model has been slowly improving. The last epoch had the best performance, at 2.094. Training for more epochs, as well as fine-tuning some parameters, could lead to more promising results.

Conclusion

In this notebook, we have explored the original Transformer architecture in depth, as presented in the *Attention Is All You Need* research paper. We used PyTorch to implement it step-by-step on a language translation task using the OpusBook dataset for English-to-Italian translation.

The Transformer is a revolutionary step towards the most advanced models we have today, such as OpenAI's GPT-4 model. And that is why it is so relevant to comprehend how this architecture works and what it can achieve.

The resources behind this notebook are the paper ***"Attention Is All You Need"*** and the YouTube video **Coding a Transformer from scratch on PyTorch, with full explanation, training and inference** posted by **Umar Jamil**. I highly suggest you check both materials for a deeper understanding of the Transformer.

If you liked the content of this notebook, feel free to leave an upvote and share it with friends and colleagues. I am also eager to read your comments, suggestions, and opinions.

Thank you very much!

Luis Fernando Torres, 2024

Let's connect! 🔗
LinkedIn • Medium • Hugging Face

Like my content? Feel free to Buy Me a Coffee ☕

https://luuisotorres.github.io/