Primjena Transformer modela dubokog učenja na obradu prirodnog jezika

Belušić, Marko

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:966978

Rights / Prava: Attribution 3.0 Unported/Imenovanje 3.0

Download date / Datum preuzimanja: **2024-03-29**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

UNIVERSITY OF ZAGREB
FACULTY OF ORGANIZATION AND INFORMATICS
VARAŽDIN

Marko Belušić

APPLICATION OF DEEP LEARNING TRANSFORMER MODEL TO NATURAL LANGUAGE PROCESSING

BACHELOR'S THESIS

Varaždin, 2021

UNIVERSITY OF ZAGREB

FACULTY OF ORGANIZATION AND INFORMATICS

V A R A Ž D I N

Marko Belušić

Student ID: 0016136348

Programme: Informacijski sustavi

APPLICATION OF DEEP LEARNING TRANSFORMER MODEL TO NATURAL LANGUAGE PROCESSING

BACHELOR'S THESIS

Mentor:

PhD, Bogdan Okreša Đurić

Varaždin, rujan 2021.

Marko Belušić

Statement of authenticity

Hereby I state that this document, my Bachelor's Thesis, is authentic, authored by me, and that, for the purposes of writing it, I have not used any sources other than those stated in this thesis. Ethically adequate and acceptable methods and techniques were used while preparing and writing this thesis.

The author acknowledges the above by accepting the statement in FOI Radovi online system.

Abstract

This thesis presents why the transformer model of deep learning overshadowed its predecessors LSTMs and classic RNNs in many natural language processing by explaining in detail its inner workings. It covers the theoretical basis of the transformer model and how its components from the encoder to decoder work together to produce valuable results. The application of the transformer model is shown through a practical example named chatbot. The chatbot is implemented using Python and Tensorflow framework, and the base architecture is the transformer model. That example shows how attention is a powerful concept even with a small dataset.

Keywords: transformer,chatbot, deep learning, transformer model, transformer architecture, AI, NLP

Table of Contents

1. Introduction 1

2. NLP Applications and Prerequisites 2

2.1. NLP Applications 2

2.1.1. Machine Translation and Seq2Seq Model 3

2.1.2. NLP Pipeline 4

2.1.3. Why Python? 5

2.2. Math Prerequisites 6

3. Transformer Architecture 11

3.1. Input Embedding and Positional Encoding 12

3.2. Multi-Head Attention 13

3.3. Feed Forward 15

3.4. Output Embedding and Masked Multi-Head Attention 16

3.5. Output 18

4. Practical Example 20

4.1. Preparing the Dataset 20

4.2. Development of the Chatbot 25

4.3. Testing the Chatbot 28

5. BERT 31

6. Conclusion 33

Bibliography 36

List of Figures 38

List of Listings 39

1. Code Appendix 41

1. Introduction

As natural language processing, NLP for short, is gaining market size and is supposed to double in the next 5 years [1] companies like Google are trying to be at the top of the growth curve. In 2017 Google released a research paper named "Attention is all you need"[2] where they introduced the concept of Transformers. Before introducing transformer architecture, it will be explained what NLP is and what are its most popular methods and use cases. Along with that, Seq2Seq model will be welcomed with a bit more detail as it is relevant to the transformer architecture and generative chatbot. This thesis will try to show in great detail how exactly and why the transformer model works, and by doing so notice its superiority to predecessors LSTMs and classic RNNs.

The application of the transformer model is shown through a chatbot. That example of chatbot is used to show how good can transformer model work with limited data, how close is human race to human-like robots, and the difference in answers when changing parameters for training, and why a good set of parameters is important.

Also, we will take a look at BERT (Bidirectional Encoder Representations from Transformers) and how it obtained state-of-the-art results on eleven natural language processing tasks. [3].

2. NLP Applications and Prerequisites

In this chapter the most common NLP methods will be presented along with math prerequisites for understanding transformer architecture and its inner workings.

2.1. NLP Applications

Natural language processing can be defined in different ways depending on the field from which we are deriving the definition. The more actual definition, that grew out of the field of linguistics is presented in the next sentence. "Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding." [4]

The most common use cases of NLP are [5] Named Entity Recognition, Sentiment Analysis, Text Summarization, Aspect mining and Topic modeling. Also, there is Machine Translation and Speech-to-text conversion.

Named entity recognition or NER for short is method for recognizing entities in a body of text. For example if in a sentence "I want to make a reservation for two at 7pm today.", the goal of this method is to recognize entities such as time and number of people.

Sentiment analysis is used when we want to find out how positive, negative or neutral a piece of text is. The most common sources for this analysis are customer reviews, surveys and social media comments. If we have the sentence, "The gym is clean and well conditioned." The output of sentiment analysis can be, or the score can be: 0.31562 and that represent a positive comment. On the other hand if sentence for analysis is "The staff was rude." its score could be: -0.62345 and that would represent a negative comment. If we try to input those sentences in an online tool for sentiment analysis [6], for the first sentence the result is positive with confidence of 98.3% and for the second sentence negative with confidence of 97.4%. Most common topics or categories of sentiment analysis are: Opinion spam and utility of opinions, Opinion search and retrieval, Sentiment analysis of comparative sentences, Feature-based sentiment analysis and Sentiment and subjectivity classification. [7]

Text summarization can be split in two categories,extraction and abstraction[7]. Extraction method creates a summary from the text it is given, by extracting parts from that text. Abstraction method generates new text based on the given text and main story or explanation in that text. Input text in both categories is usually a news article, a research paper or a page in the book.

Aspect mining is a method which identifies different aspects in the text. It is usually used in conjugation with sentiment analysis to identify how positive or negative an aspect is. For example, output can look like this: Pricing - negative. Aspect is the feature of the entity. "Adam was very satisfied with the flavour of black tea at Starbucks"[8]. Here the positive opinion

would be extracted about black tea(entity) whose aspect is flavour.

Topic modeling is the method of identifying the topic in the text. For example if we have a text or list of comments from Google reviews about your local gym and its staff. The output could be service, staff, personnel. The most common applications of topic modeling are in: Medical industry, Scientific research understanding, Investigation reports, Recommender systems... [9]

2.1.1. Machine Translation and Seq2Seq Model

Machine translation got its own section, because for its fulfillment, Sequence-To-Sequence (Seq2Seq) model is used. The same model on which inner-workings of Transformer architecture function, and the same model which is used in developing a generative chatbot, both of which are the main topics of this thesis. Abstract representation of the Seq2Seq can be seen on Figure 1.

Machine translation as the name suggests is a method of translating a piece of text from one language to another language, and by doing so keeping the meaning of the input text. The most popular model used in machine translation is the Seq2Seq model.

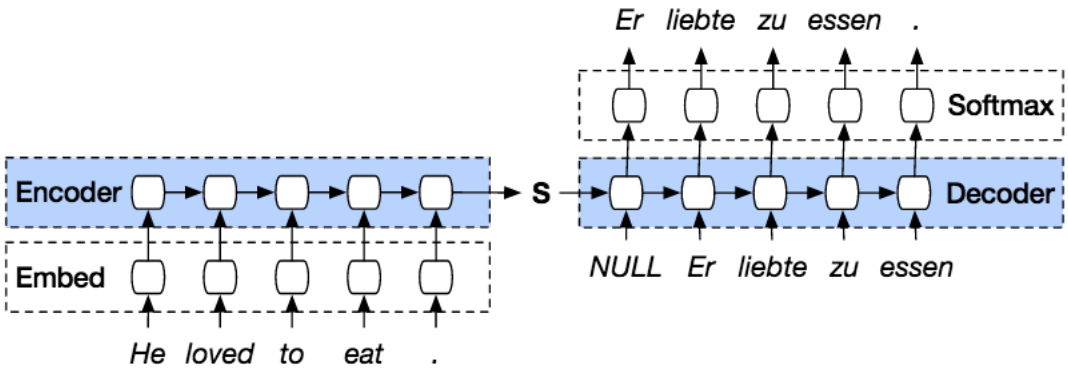


Figure 1: Abstract representation of the Seq2Seq model; Source: [10]

Sequence-To-Sequence models are used in many different NLP tasks, such as DNA sequence modeling, text summarization, speech recognition, and machine translation. Two main components of the Seq2Seq model are an encoder and a decoder. Both of those main components are usually recurrent neural network (RNN) or Long short-term memory (LSTM) models combined in one big model. RNN or LSTM structure won't be explained in this paper as it is not necessary for understanding transformer architecture, but it is beneficial to read on it[11].

In short, LSTMs are an upgrade onto the classic RNN that uses special units(they can be seen on Figure 2) in addition to standard ones to help it learn longer dependencies of sequences. RNNs are neural networks that differentiate from feed-forward neural networks by having a feedback loop from output to input. They can remember the previous state and that the previous state incorporates into the next input, allowing them to learn sequences such as words in a sentence to predict another sentence. Also, they are used in predicting time series,

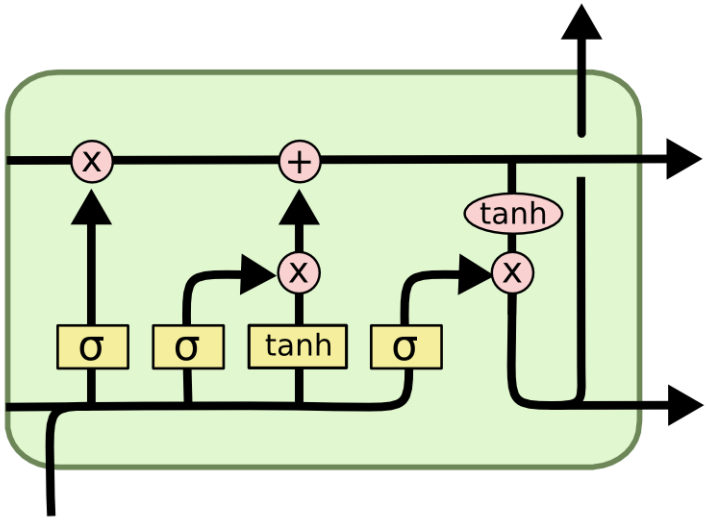


Figure 2: Representation of the LSTM cell; Source: [12]

financial data, weather, and chatbots.[11]

The encoder's job is to read the input sentence and encode the summarized information into a context vector. The goal of the context vector is to encapsulate the information from input elements so the decoder has a good starting point to make accurate predictions. The decoder's job is to take the final states of the encoder as its input states and start generating the output sequence. All of the outputs are considered before making the next output.

2.1.2. NLP Pipeline

This section will explain what elements the classic NLP pipeline consists of. Also, the elements of the the pipeline will be explained.

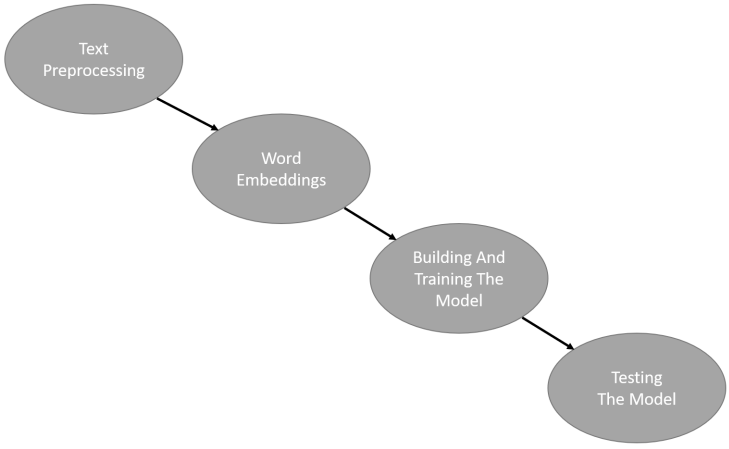


Figure 3: Basic NLP Pipeline; Source: Made in PowerPoint

The basic NLP pipeline can be seen on Figure 3 and it consist of this steps: Text pre-

processing -> Word embeddings -> Building and training the model -> Testing the model

Text preprocessing is a way of preparing the text for the model. To get the best results the text, better to say the words have to be cleaned and normalized.[13] We start with expanding contractions because we want "it's" and "it is" to be the same thing when training the model. Then we tokenize the words. That means if we have the sentence "I do not like cheese", we convert that sentence into the list of words ['I','do','not','like','cheese']. After that, it is good to apply the lowercase function to all words because we also want 'I' and 'i' to mean the same thing. We also remove all punctuation, but in some cases like a chatbot, it is good to leave question marks in the list of words to improve accuracy. Also, it can be valuable to leave more elements of punctuation. Still, we have to be careful because in a lot of cases that lowers the accuracy because of additional clutter. After all of that, stemming or lemmatization are applied. The difference between stemming and lemmatization is that stemming often does not produce the morphological root of the word. For example, if there were words ['meeting','was','worse'] then stemming (Porter stemmer) would produce ['meet','wa','wors'] and lemmatization would produce ['meeting','be','bad']. Lemmatization is more complicated but it usually produces better results.[13]

Word embedding is a method of representing words as real-valued vectors. Each word has a vector of fixed dimension. That vector can be learned while training neural network or we can use already pre-trained word embeddings on a different dataset of which is popular Word2vec.[11] Those vectors can capture similarities between words, popular examples are queen and king. They would have similar vectors.

As Jason Brownlee said in his article, the best way to find out if you should use pre-trained word embeddings or train them is to test which method of word embedding works best for your use-case."Explore the different options, and if possible, test to see which gives the best results on your problem. Perhaps start with fast methods, like using a pre-trained embedding, and only use a new embedding if it results in better performance on your problem."[14]

When building the model, we choose the appropriate method for a specific problem, some of which were mentioned earlier in this chapter, and then we train it. After that, we have to test the model. Testing depends on the method of NLP but usually, it involves graphs to see where it fails the most and it always includes giving new examples (examples that were not used to train the model, i.e. the testing part of the dataset) to the model to see how it performs.

2.1.3. Why Python?

Python is the most popular programming language for Machine Learning[15]. In survey that can be find here[15], Python is the most common choice for developers and researches in the field of ML and Data analysis. Primarily the reason being easy syntax and a lot of libraries which allow fast prototypes and testing of the end model.

Also, easy data validation. By using Python it is easier to find out why some algorithm is not performing, while for example in C or C++ it can be hard to find is it a bug in implementation or just a bad optimization. With Python, we create high-level abstractions for frameworks that

allow treating most of the concepts as objects, and we do not have to worry about memory management. It has well-built and optimized libraries and frameworks which save a lot of time. Also, it has a low entry barrier, that allows the newer software engineers to learn and focus on important aspects of machine learning and not on syntax. And the last point, it is portable and extensible. Cross-language tasks can be performed effectively on python because of its extensible and portable nature.[16]

2.2. Math Prerequisites

Topics that will be explained in this chapter are frequently used in machine learning. To be completely correct without morphing this section into a 50 pages is hard task, while also overtaking the rest of the thesis. For that reason a lot of details and mathematical tricks, bells and whistles will be left out to capture the bigger picture. For those interested the sources for more information in this section are [17] and [18]. This section should be a kind reminder of what to recall, learn, remind yourself about, not a means to learn from it.

"A vector is an object that has both a magnitude and a direction."[19] For more complete definition check out [17]. A vector is usually denoted as $\vec{a}$. Two vectors are the same if they have the same magnitude and direction. It is possible to define operations on vectors without reference to a coordinate system. Some of which are subtraction, addition and multiplication by scalar.

"Given a vector a and a real number (scalar) λ, we can form the vector λa as follows. If λ is positive, then λa is the vector whose direction is the same as the direction of a and whose length is λ times the length of a. In this case, multiplication by λ simply stretches ($\lambda > 1$) or compresses (if $0 < \lambda < 1$) the vector a.

If, on the other hand, λ is negative, then we have to take the opposite of a before stretching or compressing it. In other words, the vector λa points in the opposite direction of a, and the length of λa is $|\lambda|$ times the length of a. No matter the sign of λ, we observe that the magnitude of λa is $|\lambda|$ times the magnitude of a: $\|\lambda a\| = |\lambda| \, \|a\|$".[19]

The frequent case in machine learning is multiplying to vectors together. The geometric definition of the dot product says that the dot product between two vectors a and b is

$$a \cdot b = \|a\| \, \|b\| \, cos(\theta)$$

[19]

θ is the angle between vectors a and b. This formula is great for visualising and understating dot product but for using the formula in terms of vector components would make it easier to calculate dot product between two vectors. When we take a look at unit vectors i, j, k they are orthogonal, so any dot product with two distinct unit vectors is 0, $i \cdot k = i \cdot j = j \cdot k = 0$. The dot product between unit vector and itself is 1 because the angle is 0 and $cos(\theta) = 1$, $i \cdot i = j \cdot j = k \cdot k = 1$. Expanding $a \cdot b$ in terms of components using unit vectors and then simplifying by knowing products of unit vector we get the below formula. Scalar prod-

uct or dot product of vectors $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$ is a scalar defined to be $u \cdot v = u_1v_1 + u_2v_2 + u_3v_3$.[20]

"Matrix multiplication or product between two matrices A and B is only defined if number of columns in A is equal to number of rows in B. We multiply an $m \times n$ matrix A by an $n \times p$ matrix B. The product AB is a matrix of size $m \times p$."[21]

Formula for multiplying two matrices is

$$(AB)_{j,k} = \sum_{r=1}^{n} A_{j,r}B_{r,k}$$

[17]. The entry in row j, column k of AB is computed by taking row j of A, and column k of B, multiplying together corresponding entries and then summing.

Let A be 2×3 matrix

$$A = \begin{bmatrix} 3 & -2 & 4 \\ 1 & 0 & 2 \end{bmatrix}$$

and B be the 3×2 matrix

$$B = \begin{bmatrix} 1 & 4 \\ 2 & -3 \\ 2 & 6 \end{bmatrix}$$

Then,

$$AB = \begin{bmatrix} 3 & -2 & 4 \\ 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 2 & -3 \\ 2 & 6 \end{bmatrix}$$

$$= \begin{bmatrix} 3*1+(-2)*2+4*2 & 3*4+(-2)*(-3)+4*6 \\ 1*1+0*2+2*2 & 1*4+0*(-3)+2*6 \end{bmatrix}$$

$$= \begin{bmatrix} 7 & 42 \\ 5 & 16 \end{bmatrix}$$

In order to train the model gradient descent is used. Before showing any formulas or presenting gradient descent in more detail, the good thing to look at is a simple neural network(NN). Just as a reminder of a structure and composition of weights. Simple NN can be seen below at Figure 4.

We are computing an estimate of loss L over the training set, computing the gradients of the parameters Θ with respect to the loss estimate, and moving the parameters in the opposite direction of the gradient. Partial derivative of the cost of the function C with respect to a weight in the l^{th} layer that connects k^{th} neuron with j^{th} neuron looks like this (2.1) where a represents m^{th} neuron in an L^{th} layer.[11]

$$\frac{\partial C}{\partial w^l_{jk}} = \sum_{mnp...q} \frac{\partial C}{\partial a^L_m} \frac{\partial a^L_m}{\partial a^{L-1}_n} \frac{\partial a^{L-1}_n}{\partial a^{L-2}_p} ... \frac{\partial a^{l+1}_q}{\partial a^l_j} \frac{\partial a^l_j}{\partial w^l_{jk}} \tag{2.1}$$

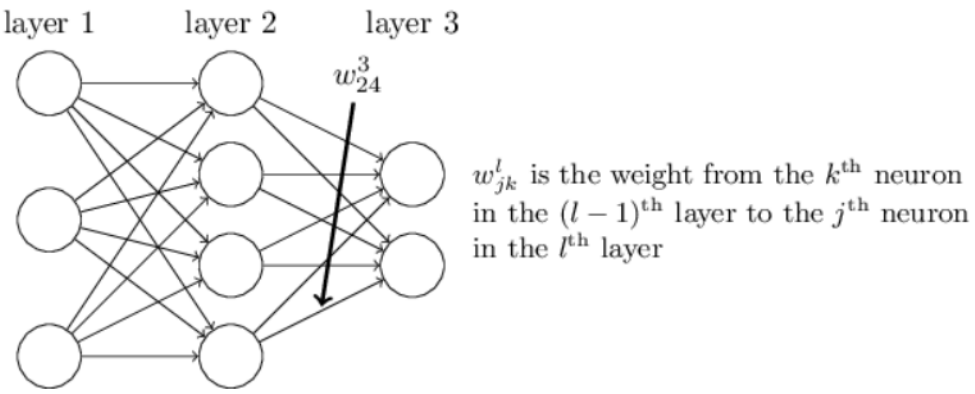


Figure 4: Simple Neural Network; Source: [22]

The formula above represent a partial derivative with respect to weight and can be found described in a lot more detail from this source [22]. A gradient of the function is a path of the steepest ascent, in geometrical interpretation what path we need to take to reach a local maximum in the least number of steps. And gradient descent is the path of steepest descent, and we get it by taking steps proportional to the negative of the gradient. Another way of saying is finding local minimum as a method of minimizing the loss function. A lot more information about vector calculus, backpropagation and gradient descent can be found here [18].

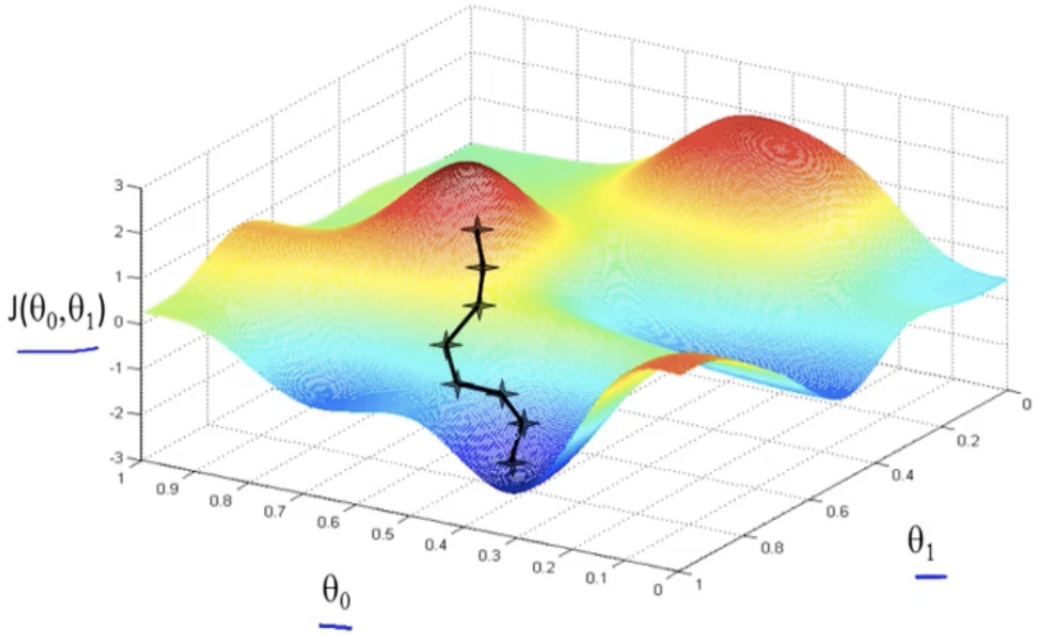


Figure 5: Visualization of gradient descent; Source: [23]

Tensors can be defined as multidimensional arrays. A vector is a one-dimensional or first-order tensor and a matrix is a two-dimensional or second-order tensor. They are highly used in machine learning and also in the fields of physics and engineering. [24] In Python, we can represent a tensor using an N-dimensional array(numpy.ndarray). The example below shows the creation of a 3x3x3 tensor as NumPy ndarray. numpy.array function returns

numpy.ndarray object type.

Listing 1: Tensor creation in Python

```
from numpy import array
T = array([
  [[1,2,3],    [4,5,6],    [7,8,9]],
  [[11,12,13], [14,15,16], [17,18,19]],
  [[21,22,23], [24,25,26], [27,28,29]],
  ])
print(T.shape)
print(T)
```

"Activation functions are used to define how the weighted sum of the input is transformed into output. Activation functions are used in hidden and output layers. All hidden layers usually use the same activation function, and the output layer can use a different one depending on the type of predictions required by the model."[25] Linear functions are not used as activation functions because we can not use backpropagation(gradient descent) to train the model. The reason for that being that derivative of a linear function is a constant and has no relation to the input. That is why we use the non-linear functions shown below. Sigmoid activation(Figure: 6) function or logistic function is a function that has an S shape. The function takes any real values as input and outputs values between 0 and 1.

$$s(x) = \frac{1}{1 + e^{-x}}$$

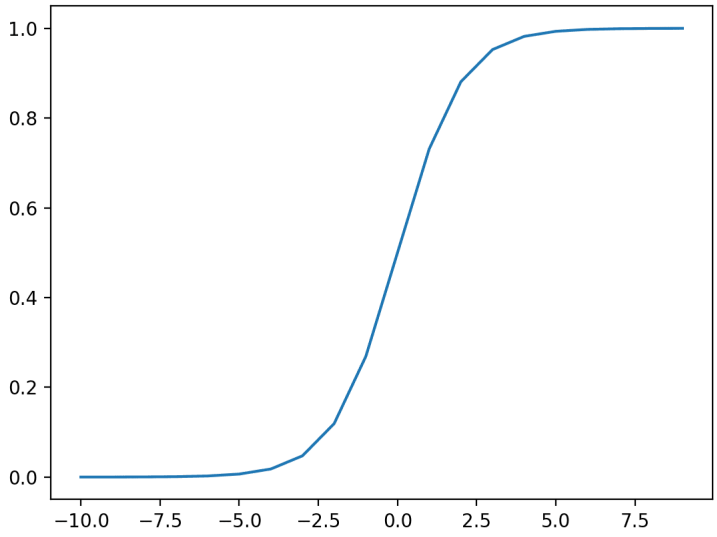


Figure 6: Sigmoid function; Source: [25]

The Hyperbolic tangent(Figure: 7) activation function(TanH for short) is similar to the sigmoid function and it even has the S shape. The difference is it takes any real value as input

and outputs the value between -1 and 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

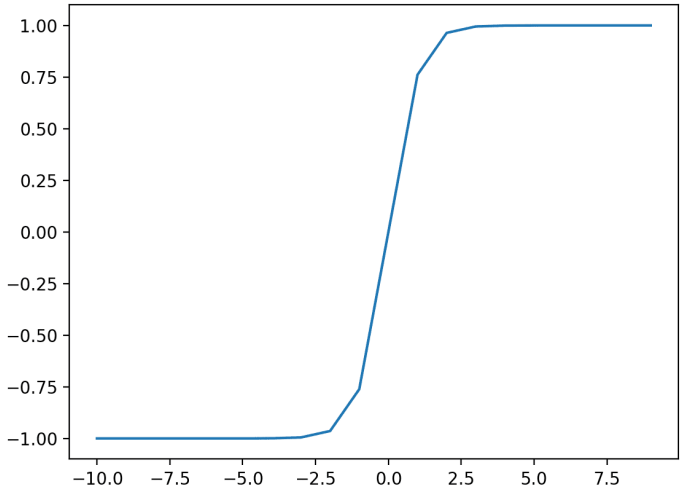


Figure 7: TanH function; Source: [25]

Rectified linear activation(Figure: 8) function(ReLU) is, at the moment, the most popular activation function for hidden layers. The reason being its simplicity and effectiveness at overcoming limitations of sigmoid and tanh functions. Specifically, it is less susceptible to vanishing gradient. If the input value is negative then the output is 0.0, otherwise the value is returned.

$$relu(x) = max(0.0, x)$$

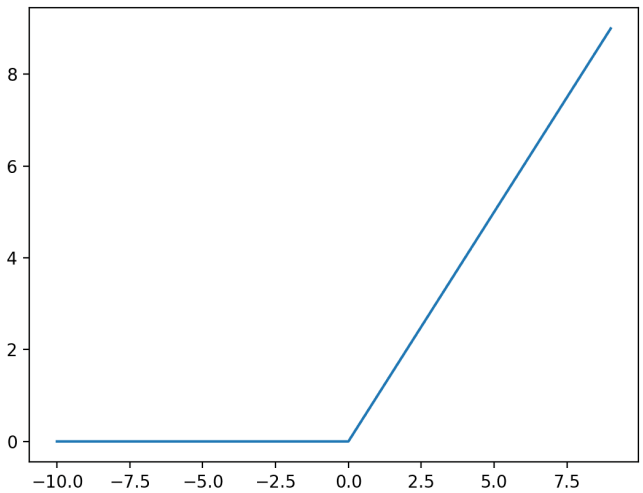


Figure 8: ReLU function; Source: [25]

3. Transformer Architecture

In this chapter, we will take a detailed look at transformer architecture, explain the flow, and explain how its components work.

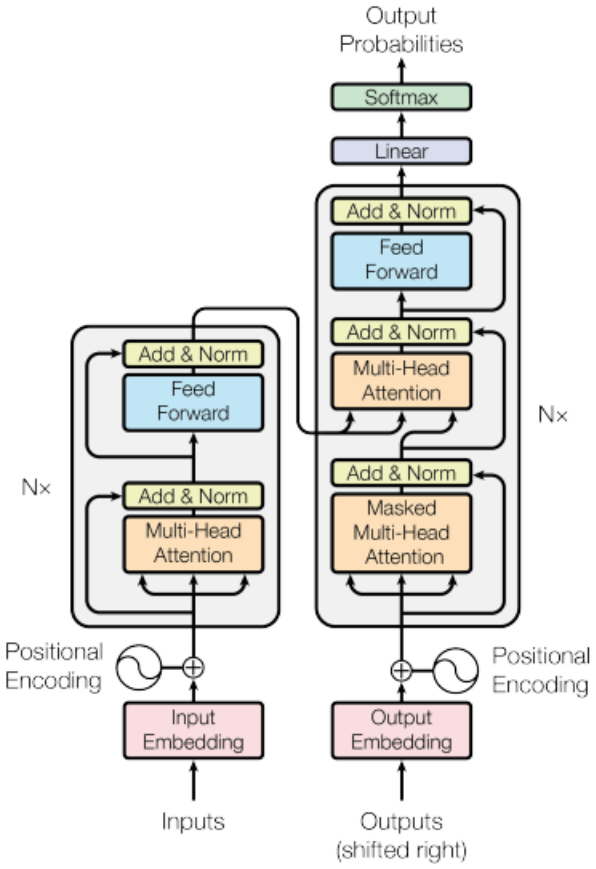


Figure 9: The Transfomer - model architecture; Source: [2]

On the Figure 8 it is possible to see of which components does transformer architecture consist of and its flow. Firstly, the input sentence is converted in more appropriate form for machine learning which is a list of numbers with additional info, in this case positional info. After the positional encoding on the left side of Figure 8 there is a structure of the encoder.

It consist of Multi-Head Attention whose job is to highlight connections between words, and their relationship in the sentence. Also it consist of Feed forward network to further improve representation of the input. In original paper there were 6 encoders [2] but number of encoders and decoders is a parameter and by need it can be changed.

Encoders have now done their job and transformed the input sentence in the valuable list of numbers. That list is used by decoder in its second Multi-Head Attention block to guide focus depending on a input sequence. Decoder start its work by doing the same conversion of the output in the list of numbers and the first difference being its Masked Multi-Head Attention block whose job is to hide future words in the sentence when training so it does not have access

to them. After that we come to the already mentioned second Multi-Head Attention block and after that to the Feed Forward block with the same functionality as in the encoder. Now there is a linear layer which is a classifier and we get a numerical value for every word in the corpus, after that layer it is possible to choose the next predicted word with a help of Softmax function to get probabilities for each word and choose one with the highest probability. Then the predicted word with the rest of predicted words is fed into the decoder until end token is predicted or maximum sentence length is reached.

3.1. Input Embedding and Positional Encoding

Input, in most cases a sentence, goes through the embedding layer. That is where our clean sentence which had pre-processing applied to it is converted from a list of numbers(tokenization) to a vector.

Each word is converted into a vector of size 512(for the rest of the chapter the values mentioned are values proposed in the original paper Attention is all you need). Output from Input Embedding is a tensor of size $numberOfWordsInASentenceX512$. These embeddings can be learned through the training to capture some insights between words or we can use already pre-trained embeddings like Word2Vec. Then we apply positional encoding to those vectors. Because transformer does not have recurrence like RNNs, we must add information about positions into the input embedding. The authors of the original paper did that by using sine and cosine functions. For every odd time step create a vector using cosine function and for every even time step create a vector using sine function. Then add that vector to the corresponding embedding vector. Those functions were chosen because they have linear properties the model can easily use.[2]

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i+1) = cos(pos/10000^{2i/d_{model}})$$

At the Figure 10 it is possible to see visually how the functions interweave and make up a interesting pattern.

Input sentence: "The keyboard is working properly." Let's say we have a embedding vector for a word "keyboard" and it looks like this. $[3.2, 0.4, 1.7, ..., 2.3, 4.1]$ We would want to add a static positional vector to that vector, with the same size of 512. 512 is d in our formula, and pos is position of a word in a sentence. For this example, "keyboard" is the second word in the sentence so pos is equal to 1. The letter 'i' in the above equations is index of embedding vector 0 for 3.2, 1 for 0.4 and so on. If we chose embedding dimension of 64 and for 10 tokens this figure would represent that.

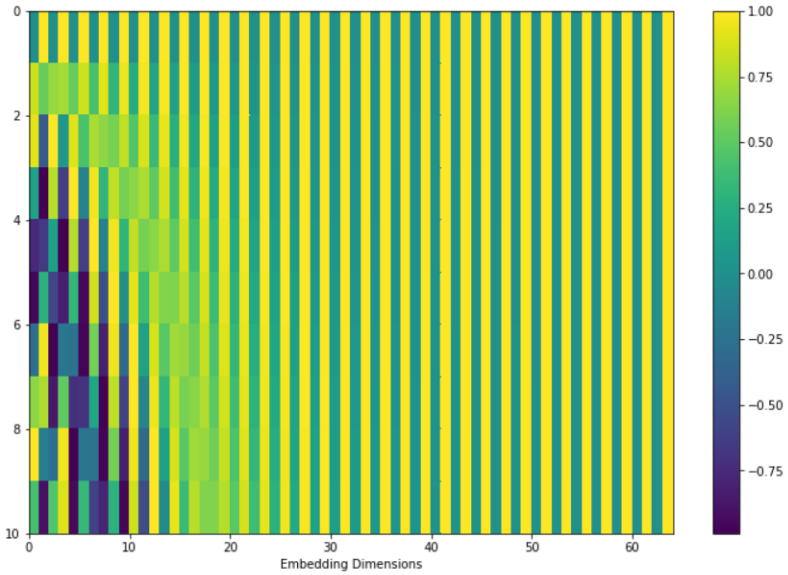


Figure 10: Visual representation of positional encoding; Source: [26]

3.2. Multi-Head Attention

In this block, self-attention is performed. Self-attention is a way for the model to learn which words to focus on when processing some other word. For example, if we have the sentence ”The animal didn’t cross the street because it was too tired", we want to answer the question what word does ’it’ refer to. Is it the animal or a street or something third? Self-attention is a way of learning that ’it’ refers to the animal.[27]

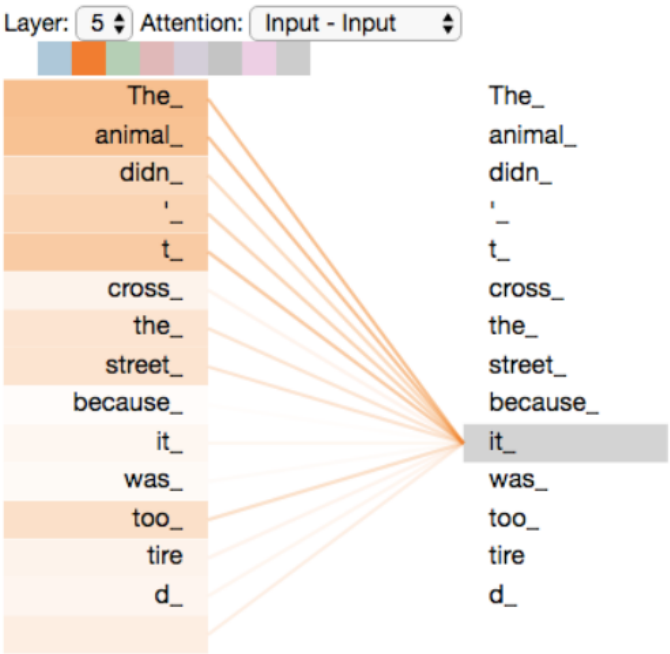


Figure 11: Attention for the word ’it’; Source: [27]

The inputs are forwarded into three fully connected linear layers. They are named,

query, key, and value.[28]

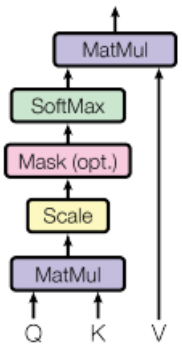


Figure 12: Scaled dot product attention; Source: [2]

Query and key matrices undergo dot product matrix multiplication to produce a score matrix. At the begging random numbers are initialized to query, key and value matrices. The first step is to multiply input matrix with the query, key and value matrices. To get Q, K and V matrices as shown on the figure below (Figure: 13).

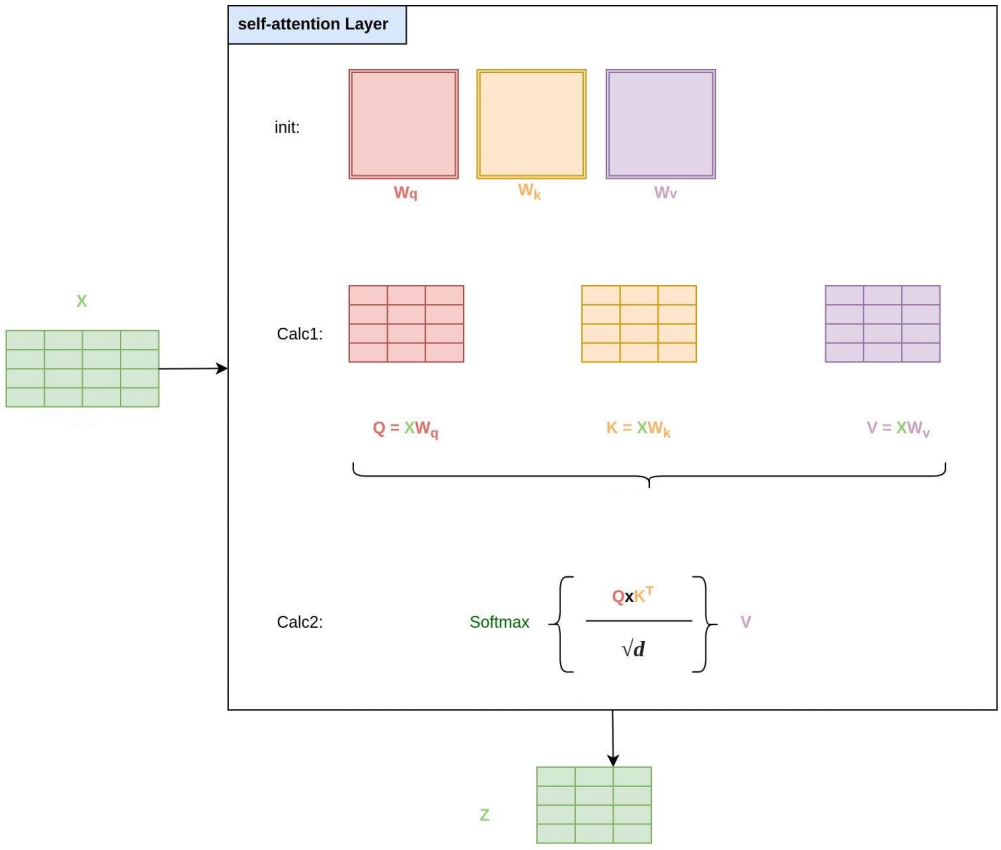


Figure 13: Workings Of Self-Attention; Source: [29]

After that we can start multiplying Q and K matrices to get the score matrix, that step is shown on Figure 14 as the first MatMul looking from the bottom of the figure. The score matrix determines how much focus should word put on other words. The higher the score the

higher the focus. For our input sentence "The keyboard is working properly". The size of the score matrix would be $5x5$, because that is what we get when multiplying the $5x64$ matrix with another $5x64$ matrix which is transposed so its size is $64x5$(dimension of 64 as in the paper, it does not have to be that value). Then the score matrix is scaled down by dividing by square root of the dimension of the queries and the keys. That way we minimize exploding gradients. Exploding gradient is a phenomenon which occurs when training the neural network during backpropagation. While calculating gradients toward input layer they are getting bigger, and that causes very large weight updates and causes gradient descent to diverge [28].

	the	keyboard	is	working	properly
the	0.8	0.1			
keyboard		0.75		0.2	
is		0.1	0.72	0.1	
working		0.2		.62	
properly		0.3			0.68

Figure 14: Score Matrix;$Softmax(\frac{Q*K^T}{\sqrt{d}})$; Source: Made in PowerPoint

Then we apply the softmax function, which results in higher scores getting bigger and lower scores smaller. Then we take attention weights and multiply them by value vector to get the output vector which we send through the linear layer to process. It is called multi-head for reason that there is more than one set of the query, key, and value vector. Each self-attention is called a head. Output from every head(8) is concatenated into one big matrix of size $5x512$ which then goes through the linear layer. Idea behind that is that each head is going to learn specific attention and that results in the model having more representation power.[28]

3.3. Feed Forward

After the multi-head attention block, there is a feed-forward block. The feed-forward block consists of a few fully connected linear layers to further improve the representation of the input. Each word in a sentence goes through feed-forward network, it is just a linear layer that gets applied to every word or position [28].

To the output from the multi-head attention block, original input is added and normalization is applied. That is called the residual connection. Residual connections help the model train by allowing gradients to flow through the model directly and retrain the residuals. A layer of normalization is stabilizing the model by normalizing each feature, in contrast to the batch normalization which is normalizing each sample. More about layer normalization can be found from this source[30]. For example, let's say we have the table shown below.

This table(Figure 16) represents input embeddings of a short sentence with dimension

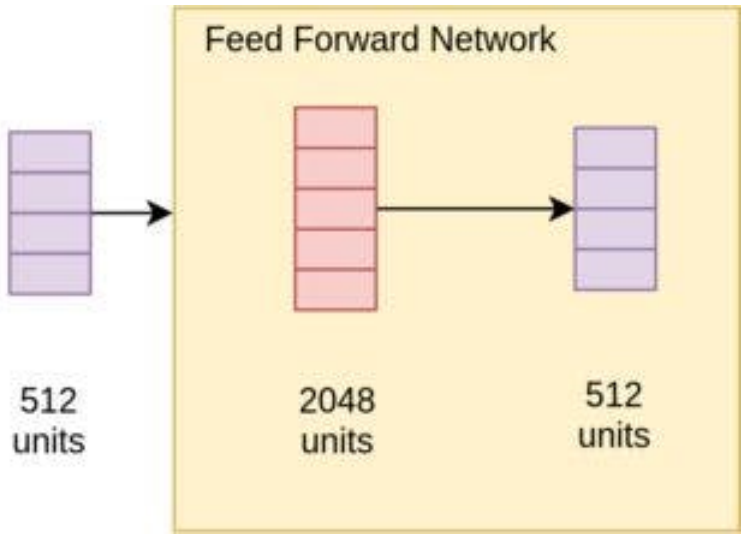


Figure 15: Representation of liner layer; Source: [29]

						$\text{mean}(\mu)$	$\text{std}(\sigma)$
$x_0 = this$	1.23	0.74	0.12	0.95	1.03	0.81	0.43
$x_1 = is$	0.05	1.08	0.34	1.15	0.61	0.65	0.47
$x_2 = cool$	0.78	2.13	1.32	0.43	0.39	1.01	0.73

Figure 16: 3x5 matrix of input embeddings; Source: Made in PowerPoint

of 5 for this example. Usually it is much bigger, in the paper and our practical example we use size of 512. For normalizing each element of a word embedding $x_{i,k}$ we use this formula:

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Formula can be find from this source [31] but it was first presented here [30]. If we apply that formula for the first word and first element

$$\hat{x}_{0,0} = \frac{1.23 - 0.81}{\sqrt{0.43^2 + 0.00001}}$$

we get normalized value for $\hat{x}_{0,0} = 0.98$. ϵ is only for numerical stability and it represents a small number in case the denominator becomes zero by chance.

3.4. Output Embedding and Masked Multi-Head Attention

The decoder's job is to generate text sequences. It has similar layers to the encoder layer, the difference being an additional masked multi-head attention layer. The decoder is

autoregressive, it takes a list of previous outputs as inputs and the encoder outputs that contain the attention information from the input. The decoder stops generating when it receives the end token as an output. Output embedding is the same size as input embedding with the same number of columns, 512 [28].

Input for the decoder is a sentence in a different language or answer to the question if we are working with a chatbot. Then it goes through the same positional encoding. Masked multi-head attention is a bit different because the decoder is autoregressive and we need to prevent it from looking at future tokens while training. For that reason, look-ahead mask is applied before softmax and after calculating the scores. The second multi-head attention layer takes values and keys from the output from encoder to know on which words from input to focus on and it creates queries from the data flowing from below.

The Decoder differs from the encoder by three components that work slightly different: Masked Multi-Head Attention, last Linear Layer and Softmax. That can be seen on Figure 9.

Starting input for the decoder is a "start" token which goes through embedding layer which converts it to a embedding vector and then positional encoding is applied the same as in encoder block. After that it goes to Masked Multi-Head Attention block. The answer to the sentence ("The keyboard is working properly") in previous example is "Yes it is". After calculating the scores in Masked Multi-Head Attention attention mask in applied to attention scores. Attention scores for the answer "Yes it is" could be something like this Figure 17(Numbers in the examples below are just for the visual aid, they are not calculated).

	<start>	yes	it	is	<end>
<start>	31.2	14.5			
yes		41.2	10.3	18.7	
it		5.8	39.4	7.2	
is		12.5	21.6	32.1	
<end>		22.1			35.8

Figure 17: Attention Scores Decoder; Source: Made in PowerPoint

At the Figure 18 attention mask can be seen. Negative infinities are in places we want to mask so decoder can not use attention weights of the future words. Mask is added to the attention scores and then the softmax function is applied. After than we get result which can be seen on Figure 19.

The next block is Multi-Head Attention. It works exactly the same as in the encoder with only difference being that key and values matrices come from the output of the encoder. Key and values matrices are copies of the output from the encoder [28].

	<start>	yes	it	is	<end>
<start>	0.0	-inf	-inf	-inf	-inf
yes	0.0	0.0	-inf	-inf	-inf
it	0.0	0.0	0.0	-inf	-inf
is	0.0	0.0	0.0	0.0	-inf
<end>	0.0	0.0	0.0	0.0	0.0

Figure 18: Attention Mask; Source: Made in PowerPoint

	<start>	yes	it	is	<end>
<start>	1	0.0	0.0	0.0	0.0
yes	0.1	0.9	0.0	0.0	0.0
it	0.001	0.1	0.9	0.0	0.0
is	0.001	0.2	0.3	0.5	0.0
<end>	0.001	0.3	0.001	0.001	0.62

Figure 19: Attention Scores After The Mask and Softmax are applied; Source: Made in PowerPoint

3.5. Output

The penultimate layer is the linear layer that acts as a classifier. If a vocabulary size is 14000 the output is going to be 14000. And then that output is fed into the softmax layer to get probabilities for each word. Index of the highest score is taken and that is the predicted word. It is predicting until the end token is reached.

The answer "Yes it is" is of size 5 when "start" and "end" token are included. Let the imaginary vocabulary size be 14000. That means that the output from last decoder is a matrix of size $5x512$. That output is going into linear layer of size $512x14000$ and then the final matrix has size $5x14000$. That can be seen visually on Figure 20. Softmax function is applied to get probabilities for each word in our vocabulary and we choose the words with the highest probabilities.

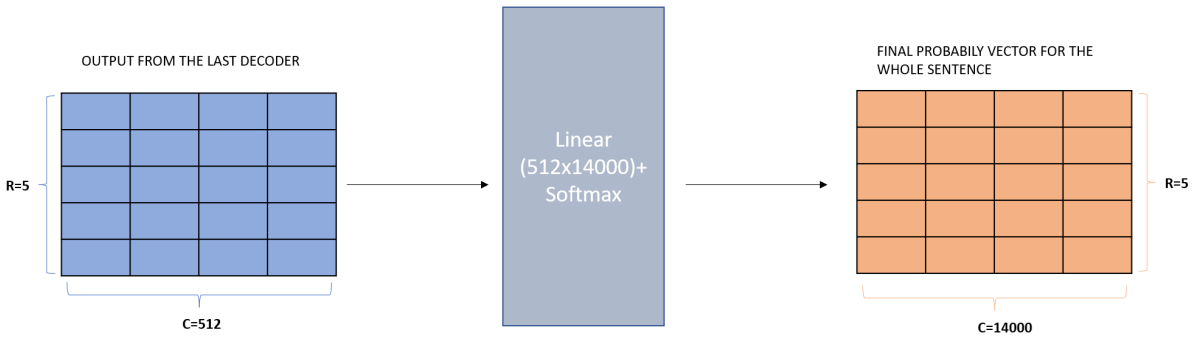


Figure 20: Visualization of Linear Layer Calculations; Source: Made in PowerPoint

4. Practical Example

In this chapter transformer-based generative chatbot is going to be made. The goal of this chapter is not to make state-of-the-art model that can be competition to BERT or similar models. The goal of this chapter is to show basic flow of NLP and practical example where Transformer architecture is used.

In first section it is shown how dataset was prepared for NLP with code snippets and short explanations. In the second section of this chapter development of the chatbot will be tackled, that includes preparing functions that convert data in more suitable format for training and later evaluation, and components of Transformer Architecture. The last section of this chapter will be testing the chatbot, in that section multiple parameters are going to be changed and then we will compare the results.

4.1. Preparing the Dataset

The dataset used in this practical example is a popular dataset with 220,579 conversational exchanges with the name Cornell movie dialog corpus. The dataset can be found on this source [32].

Listing 2: Importing The Dataset

```
1 import pandas as pd
2 data = pd.read_csv("/content/drive/MyDrive/Cornel-Question&Answers-shorterVersion2.
   csv")
3 pd.set_option('display.max_colwidth', None)
4 data.head()
```

	question	answer
0	Can we make this quick? Roxanne Korrine and Andrew Barrett are having an incredibly horrendous public break- up on the quad. Again.	Well, I thought we'd start with pronunciation, if that's okay with you.
1	Well, I thought we'd start with pronunciation, if that's okay with you.	Not the hacking and gagging and spitting part. Please.
2	Not the hacking and gagging and spitting part. Please.	Okay... then how 'bout we try out some French cuisine. Saturday? Night?
3	You're asking me out. That's so cute. What's your name again?	Forget it.
4	No, no, it's my fault -- we didn't have a proper introduction ---	Cameron.

Figure 21: Importing The Dataset; Source: Github Repository 1

At the Figure 21 and Listing 2 it can be seen that dataset is being imported using pandas library [33]. Pandas library is an open source data analysis tool which in this case helped to import the csv file. First few rows of the dataset are printed and already some unnecessary characters can be seen. For example, in the row 4 there are dashes like this "–". That can hurt the training because those characters are adding adversity we do not need to understand the sentence. Also, contractions can be noticed. They introduce diversity in the dataset because sometimes short form of some part of the sentence and sometimes the long one is used. The example being in the row three in the column named question, "You're" can later in dataset be written as "You are" and we would have two different tokens for the same meaning. The important thing to note is that shorter version of said dataset is used, 15000 pairs of question and answer. The reason for that being speed of the training, which is going to be discussed

more in section Testing The Chatbot.

Listing 3: Fixing contractions

```
data['question_tokenized'] = data['question'].apply(lambda x: [contractions.fix(word
 ) for word in str(x).split()])
data['answer_tokenized'] = data['answer'].apply(lambda x: [contractions.fix(word)
 for word in str(x).split()])
data.head()
```

	question	answer	question_tokenized	answer_tokenized
0	Can we make this quick? Roxanne Korrine and Andrew Barrett are having an incredibly horrendous public break- up on the quad. Again.	Well, I thought we'd start with pronunciation, if that's okay with you.	[Can, we, make, this, quick?, Roxanne, Korrine, and, Andrew, Barrett, are, having, an, incredibly, horrendous, public, break-, up, on, the, quad., Again.]	[Well,, I, thought, we would, start, with, pronunciation,, if, that is, okay, with, you.]
1	Well, I thought we'd start with pronunciation, if that's okay with you.	Not the hacking and gagging and spitting part. Please.	[Well,, I, thought, we would, start, with, pronunciation,, if, that is, okay, with, you.]	[Not, the, hacking, and, gagging, and, spitting, part., Please.]
2	Not the hacking and gagging and spitting part. Please.	Okay... then how 'bout we try out some French cuisine. Saturday? Night?	[Not, the, hacking, and, gagging, and, spitting, part., Please.]	[Okay..., then, how, 'bout, we, try, out, some, French, cuisine., Saturday?, Night?]
3	You're asking me out. That's so cute. What's your name again?	Forget it.	[you are, asking, me, out., that is, so, cute., what is, your, name, again?]	[Forget, it.]
4	No, no, it's my fault -- we didn't have a proper introduction ---	Cameron.	[No,, no,, it is, my, fault, --, we, did not, have, a, proper, introduction, ---]	[Cameron.]

Figure 22: Fixing contractions; Source: Github Repository 1

In Figure 22 and Listing 3 it is possible to see that short forms of words are turned into the long ones. The same example in the row three and now in the column "question_tokenized" is now in his long form "You are".

The next step is to convert all words to the lowercase. Example of doing that can be seen on Figure 23 and Listing 4. The reason for using lowercase form of the words is the same as before, for fixing contractions and getting rid of unnecessary characters. Diversity is being removed, so we do not have multiple tokens in our dictionary that mean the same thing, as that can lower accuracy of the chatbot.

Listing 4: Lower All The Words

```
data['question_tokenized'] = data['question_tokenized'].apply(lambda x: [word.lower
 () for word in x])
data['answer_tokenized'] = data['answer_tokenized'].apply(lambda x: [word.lower()
 for word in x])
data.head()
```

	question	answer	question_tokenized	answer_tokenized
0	Can we make this quick? Roxanne Korrine and Andrew Barrett are having an incredibly horrendous public break- up on the quad. Again.	Well, I thought we'd start with pronunciation, if that's okay with you.	[can, we, make, this, quick, ?, roxanne, korrine, and, andrew, barrett, are, having, an, incredibly, horrendous, public, break-, up, on, the, quad, ., again, .]	[well, ,, i, thought, we, would, start, with, pronunciation, ,, if, that, is, okay, with, you, .]
1	Well, I thought we'd start with pronunciation, if that's okay with you.	Not the hacking and gagging and spitting part. Please.	[well, ,, i, thought, we, would, start, with, pronunciation, ,, if, that, is, okay, with, you, .]	[not, the, hacking, and, gagging, and, spitting, part, ., please, .]
2	Not the hacking and gagging and spitting part. Please.	Okay... then how 'bout we try out some French cuisine. Saturday? Night?	[not, the, hacking, and, gagging, and, spitting, part, ., please, .]	[okay, ..., then, how, 'bout, we, try, out, some, french, cuisine, ., saturday, ?, night, ?]
3	You're asking me out. That's so cute. What's your name again?	Forget it.	[you, are, asking, me, out, ., that, is, so, cute, ., what, is, your, name, again, ?]	[forget, it, .]
4	No, no, it's my fault -- we didn't have a proper introduction ---	Cameron.	[no, ,, no, ,, it, is, my, fault, --, we, did, not, have, a, proper, introduction, --, -]	[cameron, .]

Figure 23: Lower All The Words; Source: Github Repository 1

On Figure 24 and on Figure 5 it is possible to see the result of removing unnecessary characters. But we still have some, so custom function is used to further clean the dataset.

Only question marks are left because use of a question mark can change the meaning of a sentence drastically. Use of the custom function and its result can be seen on Figure 25 and Listing 6.

Listing 5: Removing Unnecessary Characters

```
punc = string.punctuation
data['question_tokenized'] = data['question_tokenized'].apply(lambda x: [word for
 word in x if word not in punc])
data.head()
```

	question	answer	question_tokenized	answer_tokenized
0	Can we make this quick? Roxanne Korrine and Andrew Barrett are having an incredibly horrendous public break- up on the quad. Again.	Well, I thought we'd start with pronunciation, if that's okay with you.	[can, we, make, this, quick, roxanne, korrine, and, andrew, barrett, are, having, an, incredibly, horrendous, public, break-, up, on, the, quad, again]	[well, ,, i, thought, we, would, start, with, pronunciation, ,, if, that, is, okay, with, you, .]
1	Well, I thought we'd start with pronunciation, if that's okay with you.	Not the hacking and gagging and spitting part. Please.	[well, i, thought, we, would, start, with, pronunciation, if, that, is, okay, with, you]	[not, the, hacking, and, gagging, and, spitting, part, ., please, .]
2	Not the hacking and gagging and spitting part. Please.	Okay... then how 'bout we try out some French cuisine. Saturday? Night?	[not, the, hacking, and, gagging, and, spitting, part, please]	[okay, ..., then, how, 'bout, we, try, out, some, french, cuisine, ., saturday, ?, night, ?]
3	You're asking me out. That's so cute. What's your name again?	Forget it.	[you, are, asking, me, out, that, is, so, cute, what, is, your, name, again]	[forget, it, .]
4	No, no, it's my fault -- we didn't have a proper introduction ---	Cameron.	[no, no, it, is, my, fault, --, we, did, not, have, a, proper, introduction, --]	[cameron, .]

Figure 24: Removing Unnecessary Characters; Source: Github Repository 1

Listing 6: Removing Unnecessary Characters With The Custom Function

```
def clean(x):
  for i,word in enumerate(x):
    wordNew = re.sub('[\-_,.`"\']', '', word)
    if len(wordNew)<1:
      x.remove(word)
    elif word != wordNew:
      x[i] = wordNew
  return x
data['question_tokenized'].apply(lambda x: clean(x))
data['answer_tokenized'].apply(lambda x: clean(x))
data.head()
```

	question	answer	question_tokenized	answer_tokenized
0	Can we make this quick? Roxanne Korrine and Andrew Barrett are having an incredibly horrendous public break- up on the quad. Again.	Well, I thought we'd start with pronunciation, if that's okay with you.	[can, we, make, this, quick, roxanne, korrine, and, andrew, barrett, are, having, an, incredibly, horrendous, public, break, up, on, the, quad, again]	[well, i, thought, we, would, start, with, pronunciation, if, that, is, okay, with, you]
1	Well, I thought we'd start with pronunciation, if that's okay with you.	Not the hacking and gagging and spitting part. Please.	[well, i, thought, we, would, start, with, pronunciation, if, that, is, okay, with, you]	[not, the, hacking, and, gagging, and, spitting, part, please]
2	Not the hacking and gagging and spitting part. Please.	Okay... then how 'bout we try out some French cuisine. Saturday? Night?	[not, the, hacking, and, gagging, and, spitting, part, please]	[okay, then, how, bout, we, try, out, some, french, cuisine, saturday, ?, night, ?]
3	You're asking me out. That's so cute. What's your name again?	Forget it.	[you, are, asking, me, out, that, is, so, cute, what, is, your, name, again]	[forget, it]
4	No, no, it's my fault -- we didn't have a proper introduction ---	Cameron.	[no, no, it, is, my, fault, we, did, not, have, a, proper, introduction]	[cameron]

Figure 25: Removing Unnecessary Characters With The Custom Function; Source: Github Repository 1

Now that the dataset is clean, we are going to count the words. On Figure 26 and Listing 7 it is possible to see mapping of each word into a number that represent how many times in a dataset that word is repeated, and number of unique words in the dataset(12642).

Listing 7: Number Of Unique Words And Number Of Repetitions For Each Word

```
word_count = {}
for question in data['question_tokenized']:
    for word in question:
        if word not in word_count:
            word_count[word] = 1
        else:
            word_count[word] += 1
for answer in data['answer_tokenized']:
    for word in answer:
        if word not in word_count:
            word_count[word] = 1
        else:
            word_count[word] += 1
print(len(word_count))
print(word_count)
```

```
12642
{'can': 1721, 'we': 2601, 'make': 408, 'this': 2274, 'quick': 19, 'roxanne': 1, 'korrine': 1, 'and': 4158, 'andrew': 1,
```

Figure 26: Number Of Unique Words And Number Of Repetitions For Each Word; Source: Github Repository 1

On Figure 27 and Listing 8 four main tokens are introduced. "<PAD>" will be used as padding when sentences are of different length. Length taken is length that corresponds to the biggest length of the sentence in a batch of sentences and pad the rest so they have the same length[28]. Batch size is a parameter we choose and can depend on a dataset and purpose of the model [28] and for the best results it is the best to test it. The main reason for using batches is to improve training time and escape looping thorough every example in dataset and doing back-propagation. The next token is "<OUT>" that is the token used when the word is not in our word_indexed dictionary because it is to rare in the dataset. Threshold set for removing words is set to 5, that means only words that can be found more than 5 times in the dataset will be used for training. The reason being that there is a high chance of a word being misspelled or a name, and lowering time needed for training.

Listing 8: Number Of Words In The Final Dictionary And Words being Uniquely Mapped To A Number

```
threshold= 5
words_indexed = {}
word_number = 4

tokens = ['<PAD>', '<OUT>', '<SOS>', '<EOS>']
for token in tokens:
    words_indexed[token] = len(words_indexed)

for word, count in word_count.items():
    if count >= threshold:
        words_indexed[word] = word_number
        word_number += 1

numberOfWords = len(words_indexed)
```

```
print(numberOfWords)
print(words_indexed)
```

```
3863
{'<PAD>': 0, '<OUT>': 1, '<SOS>': 2, '<EOS>': 3, 'can': 4, 'we': 5, 'make': 6, 'this': 7, 'quick': 8,
```

Figure 27: Number Of Words In The Final Dictionary And Words being Uniquely Mapped To A Number; Source: Github Repository 1

Then tokens "<SOS>" and "<EOS>" that represent start of sentence and end of sentence tokens, which are used by Transformer to have the starter token to start generating and end token to stop when that token is reached. Code used for adding those two tokens can be found on Listing 9 and examples of sentences after the tokens are added on Figure 28.

Listing 9: Adding "<SOS>" AND "<EOS>" Tokens To The Sentences

```
for question in data['question_tokenized']:
    question.append('<EOS>')
    question.insert(0,'<SOS>')

for answer in data['answer_tokenized']:
    answer.append('<EOS>')
    answer.insert(0,'<SOS>')
print(data['question_tokenized'])
```

```
[<SOS>, well, i, thought, we, would, start, with, pronunciation, if, that, is, okay, with, you, <EOS>]
                        [<SOS>, not, the, hacking, and, gagging, and, spitting, part, please, <EOS>]
            [<SOS>, you, are, asking, me, out, that, is, so, cute, what, is, your, name, again, <EOS>]
                [<SOS>, no, no, it, is, my, fault, we, did, not, have, a, proper, introduction, <EOS>]
```

Figure 28: Added SOS AND EOS Token To The Senteces; Source: Github Repository 1

Now the only thing left to do is to add "<OUT>" tokens and convert words into number using the word_indexed dictionary that was made before. Code for that part can be seen on Listing 10 and on Figure 29 it is possible to see converted sentences.

Listing 10: Converting Sentences To Integers

```
def convertToInt(inputSen):
    global words_indexed
    sen_into_int = []
    for word in inputSen:
        if word not in words_indexed:
            sen_into_int.append(words_indexed['<OUT>'])
        else:
            sen_into_int.append(words_indexed[word])
    return sen_into_int

data['question_tokenized_int'] = data['question_tokenized'].apply(lambda x:
 convertToInt(x))
data['answer_tokenized_int'] = data['answer_tokenized'].apply(lambda x: convertToInt
 (x))
data.head(50)
```

question_tokenized	answer_tokenized	question_tokenized_int	answer_tokenized_int
[<SOS>, can, we, make, this, quick, roxanne, korrine, and, andrew, barrett, are, having, an, incredibly, horrendous, public, break, up, on, the, quad, again, <EOS>]	[<SOS>, well, i, thought, we, would, start, with, pronunciation, if, that, is, okay, with, you, <EOS>]	[2, 4, 5, 6, 7, 8, 1, 1, 9, 1, 1, 10, 11, 12, 13, 1, 14, 15, 16, 17, 18, 1, 19, 3]	[2, 20, 21, 22, 5, 23, 24, 25, 1, 26, 27, 28, 29, 25, 30, 3]
[<SOS>, well, i, thought, we, would, start, with, pronunciation, if, that, is, okay, with, you, <EOS>]	[<SOS>, not, the, hacking, and, gagging, and, spitting, part, please, <EOS>]	[2, 20, 21, 22, 5, 23, 24, 25, 1, 26, 27, 28, 29, 25, 30, 3]	[2, 31, 18, 32, 9, 1, 9, 1, 33, 34, 3]
[<SOS>, not, the, hacking, and, gagging, and, spitting, part, please, <EOS>]	[<SOS>, okay, then, how, bout, we, try, out, some, french, cuisine, saturday, ?, night, ?, <EOS>]	[2, 31, 18, 32, 9, 1, 9, 1, 33, 34, 3]	[2, 29, 75, 98, 1283, 5, 730, 37, 428, 350, 1, 203, 3840, 230, 3840, 3]
[<SOS>, you, are, asking, me, out, that, is, so, cute, what, is, your, name, again, <EOS>]	[<SOS>, forget, it, <EOS>]	[2, 30, 10, 35, 36, 37, 27, 28, 38, 39, 40, 28, 41, 42, 19, 3]	[2, 232, 44, 3]
[<SOS>, no, no, it, is, my, fault, we, did, not, have, a, proper, introduction, <EOS>]	[<SOS>, cameron, <EOS>]	[2, 43, 43, 44, 28, 45, 46, 5, 47, 31, 48, 49, 50, 1, 3]	[2, 51, 3]

Figure 29: Sentences Converted To Integers; Source: Github Repository 1

4.2. Development of the Chatbot

For the base transformer architecture Tensorflow Resources will be used as a source [34]. It uses the same parameters as presented in paper "Attention is all you need"[2]. The difference being sizes of those parameters, smaller sizes of number of layers, and neurons in deep feed-forward network. The reason being computational strength and time of training. Parameters used can be seen in Listing 11.

Listing 11: Parameters Used In Transformer Architecture For Our Model

```
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
dropout_rate = 0.1
```

Dropout rate is technique used to improve accuracy of chatbot predictions on new inputs by stopping network from over-fitting, or said in another words learning to much on specific dataset so it does not perform good on new inputs. More accurate description is this one from Goldberg. "Another effective technique for preventing neural networks from overfitting the training data is dropout training. The dropout method is designed to prevent the network from learning to rely on specific weights. It works by randomly dropping (setting to 0) half of the neurons in the network (or in a specific layer) in each training example in the stochastic-gradient training."[11]

Now that base parameters are defined, we can start working on Transformer Architecture. The code can be found in this Github Repo 1 and is originally from [34]. For that reason code will not be explained as it supersedes range of this thesis, focus will be on changes that had to be made to the dataset we had tokenized, to be able to fit in and correlate with the structure that is requested by transformer from TensorFlow.

Data we have is in a list and we want this datatype: <class 'tensorflow.python.data.ops.dataset_ops.PrefetchDataset'>. First is needed to convert our data into a Tensorflow(TF) Dataset type and then use .prefetch() function from TF to convert it to PrefetchDataset Code for this part can be seen on Listing 12.

Listing 12: Converting Data To An Appropriate Format

```
listOfQuestionsNpArray=data['question_tokenized_int'].values
listOfAnswersNpArray=data['answer_tokenized_int'].values
tensorQuestions=tf.ragged.constant(listOfQuestionsNpArray,dtype=tf.int64)
tensorAnswers=tf.ragged.constant(listOfAnswersNpArray,dtype=tf.int64)
datasetNew = tf.data.Dataset.from_tensor_slices((tensorQuestions,tensorAnswers))
def make_batches(ds):
  return (
      ds
      .cache()
      .shuffle(BUFFER_SIZE)
      .batch(BATCH_SIZE)
      .map(converToTensor, num_parallel_calls=tf.data.AUTOTUNE)
      .prefetch(tf.data.AUTOTUNE))


train_batches = make_batches(datasetNew)
```

Also, helper function are needed. Helper function convertSentenceToTensor can be found on Listing 13. Its purpose is to convert a sentence user types into a form our model can work with. It consist of basic NLP preprocessing and then conversion to tensor datatype and expanding a dimension. Because the model does not accept [2,34,56,2] but accepts [[2,34,56,2]].

Listing 13: convertSentenceToTensor Function

```
def convertSentenceToTensor(sentence):
  sentence = [contractions.fix(word) for word in str(sentence).split()]
  sentence = ' '.join(map(str, sentence))
  sentence = str(sentence).split()
  sentence = [word.lower() for word in sentence]
  sentence = [word for word in sentence if word not in punc]
  sentence = clean(sentence)
  sentence.append('<EOS>')
  sentence.insert(0,'<SOS>')
  sentence = convertToInt(sentence)
  sentence = tf.convert_to_tensor(sentence, dtype=tf.int64)
  sentence = tf.expand_dims(sentence, 0)
  return sentence
```

Example of simple sentence and result when that function is applied to it, can be seen on Figure 30.

```
convertSentenceToTensor("Hi, I'm robot and my name is \"John\" ! ? $")
```

```
<tf.Tensor: shape=(1, 11), dtype=int64, numpy=array([[   2,  145,   21,   53,    1,    9,   45,   42,   28, 2004,    3]])>
```

Figure 30: Convert Sentence To Tensor Example; Source: Github Repository 1

Listing 14: fromIntToWords() Function

```
inv_words_indexed = {value:key for key, value in words_indexed.items()}
def fromIntToWords(output):
  sentence = []
  for number in output[0]:
    #print(inv_words_indexed[number])
    #with tf.Session() as sess:  print(number.eval())
    word = inv_words_indexed[number.numpy()]
    if number.numpy() not in [0,1,2,3]:
      sentence.append(word)
  sentence = " ".join(sentence)
  return sentence
```

On Listing 14 it is possible to see another helper function that is used to convert list of numbers into a sentence. It is used when model makes its prediction and return a list of numbers, then we have to convert it back to a readable sentence. Example of use can be seen on Figure 31.

```
fromIntToWords(convertSentenceToTensor("Hi, this is test."))
```

```
'hi this is test'
```

Figure 31: Sentences Converted To Integers; Source: Github Repository 1

Also, the function where a lot of custom changes were made is called evaluate and can be seen on Listing 15. Changes include using our converter to tensor, custom function for returning start and end tokens, or to be more precise for returning their indexes. Also, as in the end printing attention_weight is not of such a value we removed that, but it is highly useful when trying to figure out good parameters and see how weights are changing.

Listing 15: evaluate Function

```
inv_words_indexed = {value:key for key, value in words_indexed.items()}
def evaluate(sentence, max_length=40):
  sentence = convertSentenceToTensor(sentence)
  encoder_input = sentence
  start, end = returnTokens()
  output = tf.convert_to_tensor([start])
  output = tf.expand_dims(output, 0)

  for i in range(max_length):
    enc_padding_mask, combined_mask, dec_padding_mask = create_masks(
        encoder_input, output)

    # predictions.shape == (batch_size, seq_len, vocab_size)
    predictions, attention_weights = transformer(encoder_input,
                                                 output,
                                                 False,
                                                 enc_padding_mask,
                                                 combined_mask,
                                                 dec_padding_mask)
```

```
20
21      predictions = predictions[:, -1:, :]  # (batch_size, 1, vocab_size)
22      predicted_id = tf.argmax(predictions, axis=-1)
23      # concatentate the predicted_id to the output which is given to the decoder
24      # as its input.
25      output = tf.concat([output, predicted_id], axis=-1)
26      # return the result if the predicted_id is equal to the end token
27      if predicted_id == end:
28        break
29
30    text = fromIntToWords(output)
31    return text
```

High overview of Transformer class can be seen on Listing 16 where it is possible to see main components. Those are Encoder, Decoder and final layer, and with what parameters they are initialized.

Listing 16: Transformer class

```
1  class Transformer(tf.keras.Model):
2    def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
3                 target_vocab_size, question_input, answer_target, rate=0.1):
4      super(Transformer, self).__init__()
5
6      self.encoder = Encoder(num_layers, d_model, num_heads, dff,
7                             input_vocab_size, question_input, rate)
8
9      self.decoder = Decoder(num_layers, d_model, num_heads, dff,
10                            target_vocab_size, answer_target, rate)
11
12     self.final_layer = tf.keras.layers.Dense(target_vocab_size)
```

4.3. Testing the Chatbot

In this section the bot will be tested and its performance will be analyzed when working with unfamiliar data. Parameters that were used to train chatbot are batch size of 32, and threshold for words equal to 5. Important thing to note is that the chatbot was trained on 5000 pairs on questions and answers, the reason being speed of training. On Figure 32 it is possible to see accuracy on training dataset after 350 epochs(it says epoch 25 because training was started from a checkpoint). Accuracy of 86.12% is in theory really good, and only after 350 epoch for which it took more than 17 hours in total.

Real life results show good results, but if taken into consideration that only 5000 pairs of questions and answers are used and in only 350 epochs of training, they can be considered really good for a basic Transformer Architecture. The terms good and really good are subjective terms, so the best way is to show examples of conversations. Example of a short conversation can be seen on Figure 33. Input is original input from human, not from a dataset, and a prediction is the output chatbot gives.

```
Epoch 25 Batch 0 Loss 1.1645 Accuracy 0.7667
Epoch 25 Batch 50 Loss 1.0628 Accuracy 0.7605
Epoch 25 Batch 100 Loss 1.0315 Accuracy 0.7706
Epoch 25 Batch 150 Loss 0.9933 Accuracy 0.7794
Epoch 25 Batch 200 Loss 0.9441 Accuracy 0.7911
Epoch 25 Batch 250 Loss 0.8824 Accuracy 0.8050
Epoch 25 Batch 300 Loss 0.8005 Accuracy 0.8233
Epoch 25 Batch 350 Loss 0.7188 Accuracy 0.8410
Epoch 25 Batch 400 Loss 0.6575 Accuracy 0.8550
Saving checkpoint for epoch 25 at /content/drive/MyDrive/checkpoints1/train/ckpt-65
Epoch 25 Loss 0.6292 Accuracy 0.8612
Time taken for 1 epoch: 200.05 secs
```

Figure 32: Results After 350 Epochs: Github Repository 1

Using GPU from Google it takes around 17 seconds for one epoch using parameters above, that is reasonable and can be used, and on my hardware it takes around 3 minutes and 30 seconds. And those were the stats when the above chatbot was trained.

On 15000 pairs and with threshold of 5 on Google GPU it takes about 40 seconds and on my hardware about 5 minute for one epoch. But, Google GPU is limited and depending on a request from other users it is distributed, so it frequently disconnects.

Before settling on the parameters above, few of them were tested on first five epochs to see its performance. All tests were done on 5000 pairs. These are the results after the fifth epoch:
batch_size = 16 and threshold = 5 results in average accuracy of 31.54%
batch_size = 32 and threshold = 5 results in average accuracy of 31.10%
batch_size = 32 and threshold = 2 results in average accuracy of 29.52%
batch_size = 64 and threshold = 5 results in average accuracy of 29.38%
batch_size = 16 and threshold = 2 results in average accuracy of 28.51%
batch_size = 64 and threshold = 2 results in average accuracy of 27.21%

Chosen pairs as before stated were parameters of batch_size equal to 32 and threshold equal to 5. Even though the best result is at the batch_size equal to 16 and the threshold equal to 2, that combination was not chosen because of the training time. To train the model with batch_size of 16 for one epoch it takes on average 48 second with Google GPU and on average 16 second with batch_size of 32. That is a substantial difference which is not worth the 0.4 percent increase in accuracy, and decrease in time for training is substantial.

```
sentence = "hi"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : hi
Prediction      : yes
```

```
sentence = "where are you from"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : where are you from
Prediction      : in your dreams ! see you tonight !
```

```
sentence = "Nice, where are we going ?"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : Nice, where are we going ?
Prediction      : i am not a cop
```

```
sentence = "I know you are not a cop, but where are we going ?"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : I know you are not a cop, but where are we going ?
Prediction      : what ?
```

```
sentence = "Just please tell me, where are we going ?"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : Just please tell me, where are we going ?
Prediction      : i am not a cop
```

```
sentence = "Why are you not a cop, you seem like you want to be"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : Why are you not a cop, you seem like you want to be
Prediction      : maybe you do not know what the hell you are talking about
```

```
sentence = "Haha, maybe you are right"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : Haha, maybe you are right
Prediction      : you know what i mean
```

```
sentence = "Why are you so mean?"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : Why are you so mean?
Prediction      : well there it is
```

```
sentence = "There is what?"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : There is what?
Prediction      : tonight ?
```

```
sentence = "After all of this, you still want to go ? ha ?"

translated_text = evaluate(sentence)
print_translation(sentence, translated_text)
```

```
Input:          : After all of this, you still want to go ? ha ?
Prediction      : there is one thing i do not need advice on it is how to drive
```

Figure 33: Short Conversation Between Human And Chatbot: Github Repository 1

5. BERT

"BERT is short for Bidirectional Encoder Representations from Transformers. BERT only uses blocks of the encoders of the Transformer in a novel way that does not make use of a decoder stack "[28]. BERT has the ability to not only look at the past words but looks at all the words in the sentence just like us humans. BERT (base) consists of 12 encoder blocks, like the one shown in Figure 34. BERT (large) consists of 24 encoder blocks.

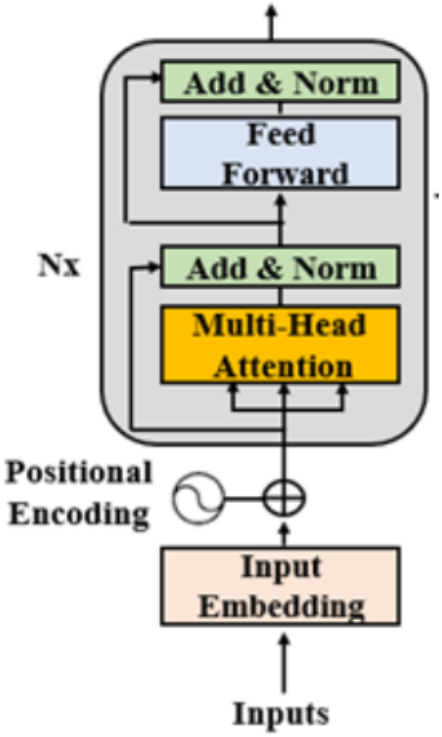


Figure 34: The Encoder block; Source: [28]

BERT obtained state-of-the-art results on eleven natural language processing tasks. A good example to single out is its performance at GLUE(The General Language Understanding Evaluation). Both BERT(base) and BERT(large) outperformed all systems on all tasks by a substantial margin, they obtained on average 4.5% and 7.0%, respectively, the improvement over the prior state-of-the-art.[3]

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Figure 35: BERT - GLUE performance; Source: [3]

The reason for mentioning BERT in this thesis is that BERT at the moment of writing is one of the best tools available for understanding human language, along with RoBERTa,

ALBERT and XLNet[35]. What differentiates BERT from previous language models is its ability to learn on the entire set of of words in a corpus or sentence rather than learning from ordered sequence that is left to right or right to left. That is more like us *homo sapiens* approach to a sentence, before deciding meaning of a word in a sentence we look at the words that precede it and follow it, not one way or the other.

At the end of 2019 Google started using BERT in its search engine and it greatly improve quality of the results after the search.[36]

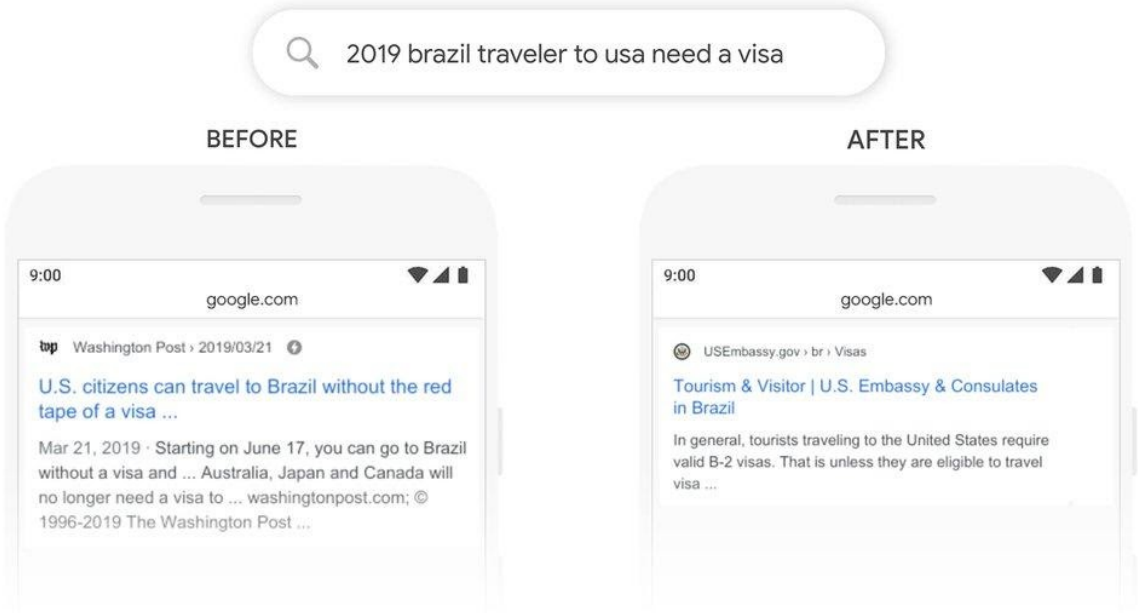


Figure 36: Comparison Between Old And New Search Results(After BERT); Source: [36]

At Figure 36 it is possible to see a difference in the search results for search "2019 brazil traveler to usa need a visa". Before implementation of BERT in the Google search engine, Google understood the query as if US citizen wanted to travel to Brazil, which is wrong. After the implementation of BERT, Google correctly recognized that the person behind query wants to know if he can travel to US without visa if he is Brazil citizen.

6. Conclusion

In conclusion, Transformer Architecture cleverly uses attention to not require additional neural network like LSTM does to keep track of long dependencies between words in a sentence. Additionally, it allows parallelization of calculations, and that enables faster training, whereas in classic RNNs calculations are performed sequentially. Parallelization can be tracked through the whole model, even in the decoder, where it uses a clever trick called masking to achieve it. Furthermore, its performance produces state-of-the-art results in more than 10 NLP tasks using modified architecture like BERT, whose architecture is based on a stack of Transformer encoders, or ALBERT. Those facts make Transformer Architecture go-to architecture for solving NLP tasks, chatbots, text generation, machine translation, and many other different application areas.

The chatbot made was a success and there is already some indication of understanding the meaning of the sentence on just 5000 pairs of questions and answers and 350 epochs of training. It is definitely not close to a human in any way, but it was also not a goal to make a state-of-the-art chatbot, as it would take a lot more research and resources to do so. The goal was to use combined theoretical concepts from this thesis into a practical example while at the same time marveling at the genius of those ideas while building a chatbot, and that was accomplished.

Bibliography

[1] L. Wood. (2021). “Global natural language processing market (2020 to 2026),” Businesswire, [Online]. Available: https://www.businesswire.com/news/home/20210301005389/en/Global-Natural-Language-Processing-Market-2020-to-2026---Increase-in-Investments-in-the-Healthcare-Vertical-Presents-Opportunities---ResearchAndMarkets.com (visited on 07/01/2021).

[2] A. Vaswani, G. Brain, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” 2017.

[3] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Tech. Rep., 2018. arXiv: 1810.04805v2.

[4] (). “What is Natural Language Processing? | SAS,” [Online]. Available: https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html (visited on 07/22/2021).

[5] Neeraja Vaidya. (). “5 Natural Language Processing Techniques for Extracting Information,” [Online]. Available: https://blog.aureusanalytics.com/blog/5-natural-language-processing-techniques-for-extracting-information (visited on 07/22/2021).

[6] (). “Free Online Sentiment Analysis Tool,” [Online]. Available: https://monkeylearn.com/sentiment-analysis-online/ (visited on 08/18/2021).

[7] N. Indurkhya and F. J. Damerau, *"Handbook of Natural Language Second Edition", dalam Machine Learning And Pattern Recognition Series*, Second edi. Cambridge: CRC Press, 2010, p. 121, ISBN: 9781420085938.

[8] (). “Aspect-Based Opinion Mining,” [Online]. Available: https://devopedia.org/aspect-based-opinion-mining (visited on 08/19/2021).

[9] (). “Topic Modelling | Topic Modelling in Natural Language Processing,” [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/topic-modelling-in-natural-language-processing/ (visited on 07/23/2021).

[10] (). “Seq2seq Model and The Exposure Bias Problem,” [Online]. Available: https://medium.com/analytics-vidhya/seq2seq-model-and-the-exposure-bias-problem-962bb5607097 (visited on 08/03/2021).

[11] Y. Goldberg, *Neural Network Methods for Natural Language Processing*, 1. 2017, vol. 10, pp. 1–311, ISBN: 9781627052986. DOI: 10.2200/S00762ED1V01Y201703HLT037.

[12] C. Olah. (). "Understanding LSTM Networks – colah's blog," [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 08/03/2021).

[13] A. C. Müller and S. Guido, *Introduction to with Python Learning Machine*. 2017, pp. 121–130.

[14] J. Brownlee. (). "What Are Word Embeddings for Text?" [Online]. Available: https://machinelearningmastery.com/what-are-word-embeddings/ (visited on 07/25/2021).

[15] A. Beklemysheva. (2021). "Why Use Python for AI and Machine Learning?" [Online]. Available: https://steelkiwi.com/blog/python-for-ai-and-machine-learning/ (visited on 08/20/2021).

[16] P. Dialani. (). "Why Should Python Be Used in Machine Learning?" [Online]. Available: https://www.analyticsinsight.net/why-should-python-be-used-in-machine-learning/ (visited on 07/25/2021).

[17] S. Axler, *Linear algebra done right*, 11. 1996, vol. 33, pp. 33–6354–33–6354, ISBN: 9783319110790. DOI: 10.5860/choice.33-6354.

[18] M. Deisentorh, "Mathematics for ML," *Cambridge University Press*, no. c, pp. 533–540, 2019.

[19] D. Frank and D. Q.Nykamp. (). "An introduction to vectors," [Online]. Available: https://mathinsight.org/vector_introduction (visited on 08/17/2021).

[20] T. Beardon. (2004). "Multiplication of Vectors," [Online]. Available: https://nrich.maths.org/2393 (visited on 07/25/2021).

[21] D. Frank and D. Q.Nykamp. (). "Multiplying matrices and vectors - Math Insight," [Online]. Available: https://mathinsight.org/matrix_vector_multiplication (visited on 07/26/2021).

[22] M. Nielsen. (2019). "Neural networks and deep learning," [Online]. Available: http://neuralnetworksanddeeplearning.com/about.html (visited on 08/18/2021).

[23] D. B. Cerigo. (). "On Why Gradient Descent is Even Needed | by Daniel Burkhardt Cerigo | Medium," [Online]. Available: https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635 (visited on 07/26/2021).

[24] J. Brownlee. (). "A Gentle Introduction to Tensors for Machine Learning with NumPy," [Online]. Available: https://machinelearningmastery.com/introduction-to-tensors-for-machine-learning/ (visited on 07/26/2021).

[25] ——, (). "How to Choose an Activation Function for Deep Learning," [Online]. Available: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/ (visited on 07/26/2021).

[26] J. Alammar. (). “Positional encoding representation-GitHub,” [Online]. Available: https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb (visited on 08/04/2021).

[27] ——, (). “The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.,” [Online]. Available: https://jalammar.github.io/illustrated-transformer/ (visited on 07/27/2021).

[28] D. Rothman, *Transformers for Natural Language Processing*. 2021, p. 384.

[29] L. Ambalina, *Essential Guide to Transformer Models in Machine Learning | Hacker Noon*, 2020.

[30] J. L. Ba, J. R. Kiros, and G. E. Hinton, 2016. arXiv: 1607.06450.

[31] L. Mao. (). “Lei Mao’s Log Book – Layer Normalization Explained,” [Online]. Available: https://leimao.github.io/blog/Layer-Normalization/ (visited on 08/05/2021).

[32] C. Danescu-Niculescu-Mizil and L. Lee, “Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs.,” in *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[33] (). “Pandas - Python Data Analysis Library,” [Online]. Available: https://pandas.pydata.org/ (visited on 08/23/2021).

[34] (). “Transformer model for language understanding - TensorFlow,” [Online]. Available: https://www.tensorflow.org/text/tutorials/transformer (visited on 07/29/2021).

[35] M. Yao. (2021). “10 Leading Language Models For NLP In 2021,” [Online]. Available: https://www.topbots.com/leading-nlp-language-models-2020/ (visited on 08/19/2021).

[36] L. Lacerda. (2020). “Google BERT: How Does The New Search Algorithm Work,” [Online]. Available: https://rockcontent.com/blog/google-bert/ (visited on 08/19/2021).

List of Figures

1. Abstract representation of the Seq2Seq model; Source: [10] 3
2. Representation of the LSTM cell; Source: [12] 4
3. Basic NLP Pipeline; Source: Made in PowerPoint 4
4. Simple Neural Network; Source: [22] 8
5. Visualization of gradient descent; Source: [23] 8
6. Sigmoid function; Source: [25] 9
7. TanH function; Source: [25] 10
8. ReLU function; Source: [25] 10
9. The Transfomer - model architecture; Source: [2] 11
10. Visual representation of positional encoding; Source: [26] 13
11. Attention for the word 'it'; Source: [27] 13
12. Scaled dot product attention; Source: [2] 14
13. Workings Of Self-Attention; Source: [29] 14
14. Score Matrix;$Softmax(\frac{Q*K^T}{\sqrt{d}})$; Source: Made in PowerPoint 15
15. Representation of liner layer; Source: [29] 16
16. 3x5 matrix of input embeddings; Source: Made in PowerPoint 16
17. Attention Scores Decoder; Source: Made in PowerPoint 17
18. Attention Mask; Source: Made in PowerPoint 18
19. Attention Scores After The Mask and Softmax are applied; Source: Made in PowerPoint 18
20. Visualization of Linear Layer Calculations; Source: Made in PowerPoint 19
21. Importing The Dataset; Source: Github Repository 1 20
22. Fixing contractions; Source: Github Repository 1 21
23. Lower All The Words; Source: Github Repository 1 21

24. Removing Unnecessary Characters; Source: Github Repository 1 22
25. Removing Unnecessary Characters With The Custom Function; Source: Github Repository 1 22
26. Number Of Unique Words And Number Of Repetitions For Each Word; Source: Github Repository 1 23
27. Number Of Words In The Final Dictionary And Words being Uniquely Mapped To A Number; Source: Github Repository 1 24
28. Added SOS AND EOS Token To The Senteces; Source: Github Repository 1 . . 24
29. Sentences Converted To Integers; Source: Github Repository 1 25
30. Convert Sentence To Tensor Example; Source: Github Repository 1 26
31. Sentences Converted To Integers; Source: Github Repository 1 27
32. Results After 350 Epochs: Github Repository 1 29
33. Short Conversation Between Human And Chatbot: Github Repository 1 30
34. The Encoder block; Source: [28] 31
35. BERT - GLUE performance; Source: [3] 31
36. Comparison Between Old And New Search Results(After BERT); Source: [36] . 32

List of Listings

1. Tensor creation in Python 9
2. Importing The Dataset 20
3. Fixing contractions 21
4. Lower All The Words 21
5. Removing Unnecessary Characters 22
6. Removing Unnecessary Characters With The Custom Function 22
7. Number Of Unique Words And Number Of Repetitions For Each Word 22
8. Number Of Words In The Final Dictionary And Words being Uniquely Mapped To A Number 23
9. Adding "<SOS>" AND "<EOS>" Tokens To The Sentences 24
10. Converting Sentences To Integers 24
11. Parameters Used In Transformer Architecture For Our Model 25
12. Converting Data To An Appropriate Format 26
13. convertSentenceToTensor Function 26
14. fromIntToWords() Function 27
15. evaluate Function 27
16. Transformer class 28

Appendix

1. Code Appendix

Link to the code on Github:

https://github.com/markoBel3/Generative-Chatbot-TransformerArchitecture