DOI: https://doi.org/10.36719/2663-4619/91/82-96

Shahriyar Guliyev
Nakhchivan State University
shahriyarguliyev@hotmail.com

ARTIFICIAL TEXT GENERATION USING DEEP NEURAL NETWORKS: TRAINING OF SULEYMAN SANI AKHUNDOV'S PLAYS

Abstract

This thesis is dedicated to Azerbaijan's outstanding writer Suleyman Sani Akhundov and his literary heritage.

It conveys the generation of artificial-text considered within the framework of experimental possibilities. The topic presented in this thesis, actually exposes engineering of the last year's technological trend - ChatGPT. The ability to write artificial- articles, poems, books require more extensive research and knowledge. In this thesis, by touching this topic to a certain extent within the resource constraints, we train the plays of a sophisticated writer like Suleyman Sani Akhundov to the computer with in-depth and generate artificial-plays of any length through the ANN model.

Keywords: *AI, ML, ANN, artificial neurons, DL, NLP, NLG, transformers, generative pre-trained transformer*

Şəhriyar Quliyev
Naxçıvan Dövlət Universiteti
shahriyarguliyev@hotmail.com

Dərin neyral şəbəkələrlə suni mətnin hazırlanması: Süleyman Sani Axundovun pyeslərinin öyrədilməsi

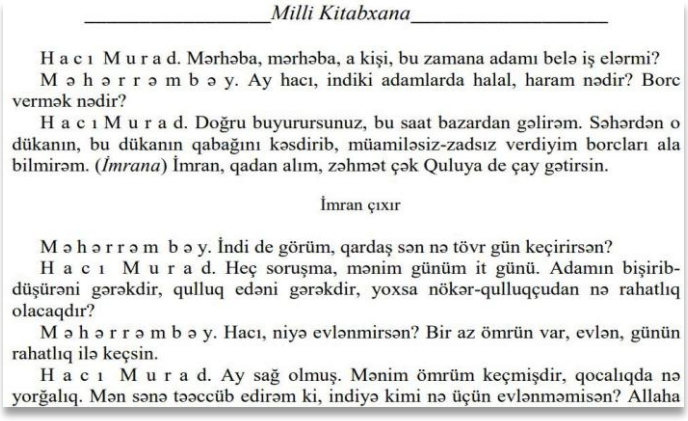
Xülasə

Bu məqalə görkəmli yazıçılarımızdan Süleyman Sani Axundova və onun ədəbi irsinə həsr olunur. Məqalədə suni mətnin hazırlanmasına təcrübi imkanlar çərçivəsində baxılır. Ötən ilin trendi ChatGPT texnologiyasının adətən mühəndisliyini ortaya çıxaran bu məqalədə yer verilən mövzu gündəmdə yerini tutmaqdadır. Suni - məqalə, şeir, kitab yazmaq daha geniş araşdırma və bilik tələb edir. Məqalədə məhdud komputer mənbələrimizlə, həmin mövzuya müəyyən qədər toxunaraq, Süleyman Sani Axundov kimi yazıçımızın pyeslərini dərin öyrənmə alqoritmi vasitəsilə komputerdə SNŞ modelinə öyrədərək istənilən uzunluqda pyes generasiya etmək olur.

Açar sözlər: *SZ, MÖ, SNŞ, suni neyronlar, DÖ, TDE, TDG, transformerlər, generativ öncədən-öyrədilmiş transformerg*

Introduction

This is the raw data we've passed to computer – the ANN model as an input to be learnt in depth.



Milli Kitabxana

H a c ı M u r a d. Mərhəba, mərhəba, a kişi, bu zamana adamı belə iş elərmi?

M ə h ə r r ə m b ə y. Ay hacı, indiki adamlarda halal, haram nədir? Borc vermək nədir?

H a c ı M u r a d. Doğru buyursunuz, bu saat bazardan gəlirəm. Səhərdən o dükanın, bu dükanın qabağını kəsdirib, müamiləsiz-zadsız verdiyim borcları ala bilmirəm. (*İmrana*) İmran, qadan alım, zəhmət çək Quluya de çay gətirsin.

İmran çıxır

M ə h ə r r ə m b ə y. İndi de görüm, qardaş sən nə tövr gün keçirirsən?

H a c ı M u r a d. Heç soruşma, mənim günüm it günü. Adamın bişirib-düşürəni gərəkdir, qulluq edəni gərəkdir, yoxsa nökər-qulluqçudan nə rahatlıq olacaqdır?

M ə h ə r r ə m b ə y. Hacı, niyə evlənmirsən? Bir az ömrün var, evlən, günün rahatlıq ilə keçsin.

H a c ı M u r a d. Ay sağ olmuş. Mənim ömrüm keçmişdir, qocalıqda nə yorğalıq. Mən sənə təəccüb edirəm ki, indiyə kimi nə üçün evlənməmisən? Allaha

We'll achieve this result after training the model through hours of optimizations.

```
Abas bəyə daxil olur.
Ş ə r ə c. Mən
tİKİNSİ ŞƏKİ
Bir aydır ki, qaçır.
ə l i m ə r d d i n. Oğlum, nə dörd qırmışam. Sənə ac qalıb şərtinə düşdü. Bunu sən nə
tərəfində ki, səni bu ölümdə hec küartını açıb yoxdur ki, pulun qüvvə qeyri edər, İ Piri
baba, xudahafiz. Yaxşı, ancaq siz öləsidə bir təyirif gedək bilmərəm. Şərəf xanım}nb r e
y n ə c. İndə və sağ mənim kimdən de?
H a d i l ə l ə. İndi də oynarda yuxu gözləyir. Elədir kimi, onunlmadı. Parisi-gəldini
yetərmisənmin.
H ə c ə r x a n ı m. Gəl bəyim! Mirzən başına qoy, əmimə göydər.
H a c ı M u r a d. İndi hürriyim yoxdur.
Ç i n g i z. Evdə razı özünü nəndinə dolaşlardaşayaq.
V ə l i a ğ a. Xeyr, Molla iş olma, əmiraslan ağanın atdığan dayının, hazırlar içinin
üçün ac qızı Dedim də, Şeyx əmiraslan ağanın üstün nəqiçəs dəlilərə qucaqladı.
- mütəliyin, Hacı Murada mən onu qovurya, görürsən. (Sənin evlən fikrində şahbaz bəy, tez
başına baydır, a pula həsrət Kazımdək könmüşdü, gəl gülə sibinin bunlar kimdən rahat ol.
S ə f d ə r q u l u b ə y. əliqulu xəstə olum i. Adə, tuz hava! Mən bir oğlan çıxar. Ey
şadlıqca  kişimizdir. Zəhərfinizi boynudur. Puyur et (sate) Nabat düşməni lələ oyaqdır,
```

Initially, I must express that scientific work has to be conducted with special care, attention showing to teaching quality rather than quantity must be measured as importance and in this regard, distinguished amongst many universities, to the - Nakhchivan State University, its Architecture and Engineering faculty, all professor, teacher and staff, that I'd like to thank for helping to strengthen and deepen my knowledge.

As known, during the time of extensive progress in technology in this Information Age where knowledge becomes the biggest treasure, this tendency applies also to our country to make youth dive into the technological professions achieving qualified knowledge by which, increased amount of activity has spread over country in IoT field recent years.

In the last years, one of the trends is ChatGPT technology, which works by Artificial Neural Networks, opened new opportunities in front of us in many repetitive works, fields of natural language processing, that in return has substituted many human-executed jobs and is expected to do more in near and far future by follows which can be guessed by less imagination.

Here it is that instead of coders, we need qualified engineers who can bring innovation which ChatGPT particularly can't achieve.

On the contrary the topic of this thesis, exposes the ChatGPT's engineering principles, its most difficult implementation example of artificial-text generation from scratch. Thus, those becomes familiar with this work, can prepare similar technologies or new applications working by this matter and they will witness that such applications can be made possible and then they'll get opportunities to use it in useful directions, projects, work.

In the thesis, research has been conducted from zero up to the on-hand results. Conclusions as being practical, which has been produced limited to the computer's architectural resources, yet has shown that using Artificial Neural Networks, we can achieve results in many processes that we could not even imagine prior to it.

Research Objectives

It has been carried out in one directional but in many branches:

Artificial Intelligence
Machine Learning
Artificial Neural Networks
Deep Learning
Artificial Neurons
Hidden Layers, their Depth
Input & Output Layers
Calculus on Artificial Neural dependencies
Feed-forward & Back-propagation

Optimizing the Weight of Artificial Neurons
Natural Language Processing / Natural Language Generation
Transformers / Generative Pre-Trained Transformer
Serialization – Pickling of the Model Tensor
The Model's Overfitting Issue
Encoding & Decoding of Statements and Corresponding Mapping
Importing & Calling of Trained Model
The Loss Estimation
TensorBoard Statistics in the Results Review
Overview of NVIDIA's CUDA® GPU Technology
PyTorch over GPU - faster Matrix operations
Tensorflow and Keras Insights
and more many divisions highlighted.

Methodology and Hypotheses.

As we know, Artificial Intellect has many areas and amongst them there is Machine Learning and then Deep learning, where in its Artificial Neural Networks exists millions of billions of artificial neurons in use.

We can implement Deep Learning through many of its formalized Algorithms inside applications like Tensorflow, which is written in C++ PL, having also Python interface. Alongside there is Keras which is a simplified and more abstracted version of Tensorflow that has serious loss in execution performance. Then there is PyTorch written by Facebook, actually isn't new than NumPy library instead, the only difference is NumPy cannot utilize the very efficient GPU modules which operates well on Matrix operations, alongside, PyTorch also has Artificial Neural Networks Classes that eases the process. Cos of it has been written in Pythonic style, it is much easier to use than its counterpart Tensorflow, in many benchmarks, they have similar performance runs. And additionally, PyTorch also has a C++ interface if referred.

For the conception viewed in the thesis, PyTorch is going to be used as an application, it averages well in level of service, easiness, performance and fast-done of work intended. Of course, for serious runs, Tensorflow should be the choice of use that is more appropriate for achieving high performance goals.

- **Advantages**: PyTorch is regarded as a first-class Artificial Neural Network application service.
- **Features**: Able to execute almost all the features of Artificial Neural Networks. Similar to Tensorflow.
- **Speed**: Although Tensorflow executes quite faster, by PyTorch can prepare the model more quickly and that can compensate the lost speed on average.
- "**Consistency**": PyTorch is written in Pythonic style and constitutes the very same principles with NumPy in data types, operations and even in nomenclature.

Artificial Neural Networks.

Artificial Neural Networks or simply neural networks are inspired by biological neuronal networks. A real biological neuron, or a nerve cell, comprises dendrites, a cell body, and an axon that leads to synaptic terminals. A neuron transmits information via electrochemical signals. A human brain has on the order of 100 billion neurons, with each neuron having between 1,000 to 10,000 connections to other neurons. Artificial neural networks are comprised of abstract neurons that try to mimic real neurons at a very high level. They can be described via a weighted directed graph G = (V, E), with each node $\mathbf{v_i} \in V$ representing a neuron, and each directed edge $(\mathbf{v_i},\mathbf{v_j}) \in E$ representing a synaptic to dendritic connection from $\mathbf{v_i}$ to $\mathbf{v_j}$. The weight of the edge $\mathbf{w_{ij}}$ denotes the synaptic strength. It is important to note that a neural network is designed to represent and learn information by adjusting the synaptic weights (Mohammed Zaki, Wagner Meira, 2020: 647).

DL Algorithms: Transformers & Generative Pre-trained Transformer.

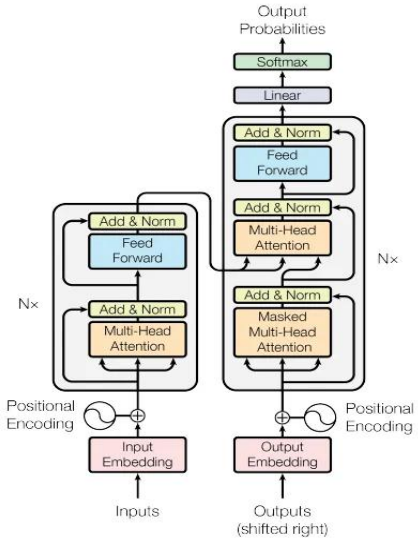


Figure 1. From "Attention is all you need" paper by Anish V., et al., 2017

The introduction of Transformers architecture in the "Attention Is All You Need" paper, has changed a lot in NLP.

In particular, a family of transformer-based algorithms has significantly improved the quality of results we can achieve on text problems (Raff, 2022: 543).

Generative Pre-trained Transformers, commonly known as GPT (14), are a family of Artificial Neural Network models that uses the transformer architecture and is a key advancement in Artificial Intelligence powering generative AI applications such as ChatGPT. Organizations across industries are using GPT models and generative AI for Q&A bots, text summarization, content generation, and search (3).

Encoding and Decoding Text / Mapping of Letters to Numbers.

Providing the source input of our problem in a TXT file for easiness, that is going to have 4000+ lines of rows of text copied from Suleyman Sani Akhundov's "*Secilmis Eserleri*" books.

First of all, let's insert this data into our program code.

This time we will split the textual data into 3 parts: "Train", "Validation" and "Test". but in matter of simplicity, and easiness of understanding of the execution stages of a AI model, we'll deduct the Test phase satisfied by the Train and Validation stages.

More after, there is a new function we must write where we'll use a lambda expression and by using it, will be encoding and decoding any expression in a certain range of N natural numbers.

For example,

A, B, C, D

1, 2, 3, 4

Here each single letter is being mapped to a relevant and unrepeatable number.

By this way, let's view a real encoding example on a random textual statement,

More after, again using another lambda expression using in that specific sequence "decode" our text, I mean decode, we will again get a set of letters to the stack converted.

The purpose of this stage is that by digitalizing a textual sentence which is our writing to be in a matrix form, then we are able to find its closeness, distance, similarity, etc. with other words, statements and then identify the loss, cost and optimize our output value.

Sample Encoding.

```
cumle_numune1="Buradakı çay fincanları kimindir?"
# her bir vahid herf burda yer alir
chars = sorted(list(set(cumle_numune1)))
vocab_size = len(chars)
# herfleri reqemlere inikas et
string_to_integer_map = {char:index for index, char in enumerate(chars)}
integer_to_string_map = {index:char for index, char in enumerate(chars)}
encode = lambda string: [string_to_integer_map[char] for char in string] # sozu qaraq onu integer yiginina cevir
decode = lambda list: ''.join([integer_to_string_map[integer] for integer in list]) # decoder: integer siyahisi al

kodlaşdırılmış_yazı= encode(cumle_numune1)
print(kodlaşdırılmış_yazı)

breakpoint()

# Train and test splits
```

```
C:\Users\shahr\AppData\Local\Programs\Python\Python311\python.exe "D:\NDU dərs\Suni Zəka -
[2, 13, 12, 3, 5, 3, 8, 16, 0, 15, 3, 14, 0, 6, 7, 11, 4, 3, 11, 9, 3, 12, 16, 0, 8, 7, 10, 7, 11, 5, 7, 12, 1]
> d:\ndu dərs\suni zəka - müştəri xidmətləri\ng-video-lecture-master\ng-video-lecture-master\gpt.py(78)<module>()
-> data = torch.tensor(encode(text), dtype=torch.long)
(Pdb)
```

For example, let's insert the sentence below to that specific lambda encoder expression:
"Buradakı çay fincanları kimindir?"
That will encode the sentence (*set of letters*), (2, 13, 12, 3, 5, 3, 8, 16, 0, 15, 3, 14, 0, 6, 7, 11, 4, 3, 11, 9, 3, 12, 16, 0, 8, 7, 10, 7, 11, 5, 7, 12, 1)
That achieves its digital expression like the one-dimensional Matrix above.
And this kind of digital expression will be used throughout the entire Artificial Neural Network Class, eventually, in conclusion, as we want to generation desired amounts of letters of plays, decode that output digital expression using the other lambda decoder expression that will convert it and then we'll store it in our computer's storage.

Constants, Functions and Classes

The functions and Classes we need are the following;
And each Class has methods and some of those methods will be subclassed by getting inheritance and then change the content of the methods using the polymorphism property of OOP so that we can perform specialized operations.

Libraries.

Before everything, let's import the needed library and modules.

```
import torch.nn as nn
from torch.nn import functi
```

And inside these modules, would like to specially note the PyTorch's "Neural Networks" module. From this module we will inherit an Artificial Neural Network subclass. This time we will change the values of the attributes, methods of our subclass and so on we will work.

Preparing "Batch" of Data to Process

In addition, among the functions that we will make, there is "batch" generator - that is, our Artificial Neural Network model at the same time can do one operation execution, or can execute limitless amounts of operations (of course using multi-threading), in this case, this execution limit or size is called batch ("batch" execution number) "size".

```
def coxluCutluk_topla(ayrim_novu):
    # verilenleri batch size qeder temin et hansi ki giris (input) x and hedef (target) y ol
    if ayrim_novu == 'oyretme':
        data = oyretme_verilenleri
        print("ix yigiminda 'oyretme' data secildi")
    else:
        data = tesdiqleme_verilenleri
        print("ix yigiminda 'tesdiqleme' data secildi")

    ix = torch.randint(len(data) - herf_hesabi, (coxluIslem_hesabi,))
    print(f"randomized ix: {ix}" )
    print(f"len(ix): {len(ix)} and len(batch_size): {coxluIslem_hesabi}")

    x_inputs_stack_list=[]
    y_targets_stack_list=[]

    for index, i in enumerate(ix):
        print(f"=====> batch {index+1} (ix: {index+1}) :")

        x_input= data[i : i + herf_hesabi]
        y_input= data[i+1 : i + herf_hesabi + 1]
        x_inputs_stack_list.append(x_input)
        y_targets_stack_list.append(y_input)
```

```
print(f"yigildi (len:{len(x_inputs_stack_list)}) x giris siyahisi:\n {x_inputs_stack_list} \n")
print(f"yigildi (len:{len(y_targets_stack_list)}) y hedef siyahisi: \n {y_targets_stack_list} \n")

x_giris_tensor_obj = torch.stack(x_inputs_stack_list)
y_hedef_tensor_obj = torch.stack(y_targets_stack_list)

x_giris_tensor_obj, y_hedef_tensor_obj = x_giris_tensor_obj.to(cihaz), y_hedef_tensor_obj.to(cihaz)
return x_giris_tensor_obj, y_hedef_tensor_obj
```

For example, if the "batch" size is 1, our program at the same time can train only one Artificial Neural Network Tensor.

If the batch size is infinitely defined to work, at the same time infinitely (possible limit) execution takes in place. In conclusion, we must choose a suitable batch size of choice per our computer's resource constraints that will allow us to use resources effectively and avoid lagging the execution of our computer's system programs.

For example, the processor oscillation frequency, memory capacity, cache memory size, the number of logical processors, the number of cores in a multi-core CPU, etc. all counts to performance hits.

Suitable to these resources, in the current computer, the batch size is determined to be et to 64 it is possible to train an Artificial Neural Network normally and without massive delays (resources of the computer in-use: CPU clock speed @2.5 up to @3.5 GHz, DRAM 12 GB).

Word size for Forming Ans

```
coxluIslem_hesabi = 64
# herf_hesabi = 256 # tapmaq uc
herf_hesabi = 8
```

Else that, it's necessary of the size of the word to be trained.

For example, in each training cycle, we can teach a word of say 10 letters (chars) or we can teach a word of 107 letters or 500 letters where again, the computer in-use, its resources, choosing suitable amounts is important.

And that on the computer I've used, suitable word size can be seen as the word size of 8 letters that makes fit for purpose.

Note: In these sizes, exponential of two is not a requirement. Simply, writing this way is more computer-alike.

Matrices & Linear Operations

Linear Algebra has moved to the center of Machine Learning, and we need to be there (Strang, 2019: 4).

The data often appears in the form of large matrices. The rows of the data matrix would represent feature vectors for individual input instances. The number of columns would match the size of the feature vector. The geometrical representation of such an input matrix would be a set of points (Chaudhury, 2021: 137).

Else that, can say that all processes include the conversion of our writings into digital form - its matrix formation, and then operations on these numbers of matrix elements carrying out on the computer done algorithmically training their appropriateness of the sequences those consist of. And in all operations are being done on the matrices, its elements to obtain results.

This time, let's take a look at Linear Algebra, which is all about Matrices:

What we know from Linear Algebra is it issues problems by turning them into "Linear function" to solve it. Linear function is a which is by its name has a graph of a line,

y= **M***x+ **b**

M: slope of the tangent-line drawn to the function of y (angle coefficient)

B: y-intercept (y-tangent)

In these problems, digital information establishes exactly in a Matrix form.

Then we're combining complex clusters in Matrices to stand as a linear function then, finding its transponder, rang, determinant, nullify, products, etc.

Activation on ANs / Sigmoid & Rectified Linear Unit functions

The building block for Artificial Neural Networks are Artificial Neurons or Perceptrons. These are simple computational units that have weighted input signals and produce an output signal using an activation function (Tam, 2023: 5).

Certain functions arise often while working with probability distributions, especially the probability distributions used in Deep Learning models. One of these functions is the logistic sigmoid (Goodfellow, Bengio, Courville, 2016: 65).

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

The sigmoid function is used because it maps the interval $[-\infty,\infty][-\infty,\infty]$ monotonically onto [0,1][0,1], and additionally has some nice mathematical properties that are useful for fitting and interpreting models (Drury, 2018: 1).

During all these operations, the result of Matrices is the probability produced by the computers ANN model, and its correlation with the expected output is determined in the (0, 1) range by standard.

To convert any random number into a unique number mapped into (0,1) range, we can use a Sigmoid function, and its inverse function is Logit function that is referred by Softmax function for probability estimations.

**In early years, in ANN, instead of Logit function, Probit was used which is less optimal.*

This way, by we can convert very large numbers into a single scaled unique number in (0,1) range easily by using the Sigmoid function to make more appropriate comparisons between two different ranged values.

```
x = self.ln_f(x) # (B,T,C)
logits = self.lm_head(x) # (B,T,vocab_size)

if targets is None:
    loss = None
else:
    B, T, C = logits.shape
    logits = logits.view(B*T, C)
    targets = targets.view(B*T)
    loss = F.cross_entropy(logits, targets)

return logits, loss
```

In contrast, on modern ANN implementations, Sigmoid function is retired as Rectified Linear Unit (ReLU) function and its derivatives (*like Leaky ReLU*) is more a de-facto standard cos it is much more efficient and more compact. And our ANN model also uses the ReLU function as an activation function, too.

Weights & Differentiation

Calculus is more useful in Machine Learning and Deep Learning for using of optimization techniques like Gradient Descent, SGD, Adam, RMSProp, Ada Delta, etc. techniques will partial differentiate, it will Equate the minimum 0 value, i.e., 0 slope value, etc. (Sairam Reddy Lattupally, 2018: 15).

Weights are essential concept in Artificial Neural Networks, as in the Biological Neural Networks; Weight in the ANN acts as a Synapsis joint, its strength in a biological neuron. And the dependency of a neuron from another neuron, and so forth as they're correlated makes the change in a particular neuron's weight affecting its activation function, so its output and then other artificial neurons connected to it; That rate of the change of the values actually are differential equations and solved such.

Estimating Loss.

Sometimes we have to transform or add variables to get the equation to be linear, e.g taking logs of output. Then we can run our estimation, do model checking, visualize results, etc (O'Halloran, 2023: 2)

During making our model, determining the value of the cost function and optimize AN's weights accordingly, it is convenient to keep an eye on the visualized statistical data and observe its training and validation losses, and then design our model acordingly to be better and better.

```python
@torch.no_grad()
def itkini_hesabla():
    cixis = {}
    model.eval()
    for ayrim_novu in ['oyretme', 'tesdiqleme']:
        itkiler = torch.zeros(deyerlendirmeIterasiya_sayi)
        for k in range(deyerlendirmeIterasiya_sayi):
            X_giris_tensor_obj, Y_hedef_tensor_obj = coxluCutluk_topla(ayrim_novu)
            logits, itki = model(X_giris_tensor_obj, Y_hedef_tensor_obj)
            itkiler[k] = itki.item()
        cixis[ayrim_novu] = itkiler.mean()
    model.train()
    return cixis
```

Optimization Techniques.

The Artificial Neural Network model is required to be constantly optimized. There are many more commonly used algorithms that we can use.

This time, we'll use as an Optimization function called "*AdamW*" Algorithm to solve our problem.

```python
# PyTorch optimallasdirici algoritmini cagir
optimallaşdırıcı = torch.optim.AdamW(model.parameters(), lr=oyrenme_tezliyi)
```

```python
optimallaşdırıcı.zero_grad(set_to_none=True)
itki.backward()
optimallaşdırıcı.step()
print(f"-> itki {itki_hesabi_sirasi} hesablandi; {time.time() - zt_itkiler1} san. kecdi")
```

Token Generation.

In context of deep learning the logits layer means the layer that feeds in to Softmax (or other such normalization). The output of the Softmax are the probabilities for the classification task and its input is logits layer. The logits layer typically produces values from -infinity to +infinity and the Softmax layer transforms it to values from 0 to 1. (Shah, 2008: 5).

```python
def generate(self, idx, enCox_yeni_token):
    # idx is (B, T) array of indices in the current context
    for _ in range(enCox_yeni_token):
        # crop idx to the last block_size tokens
        idx_cond = idx[:, -herf_hesabi:]
        # get the predictions
        logits, loss = self(idx_cond)
        # focus only on the last time step
        logits = logits[:, -1, :] # becomes (B, C)
        # apply softmax to get probabilities
        probs = F.softmax(logits, dim=-1) # (B, C)
        # sample from the distribution
        idx_next = torch.multinomial(probs, num_samples=1) # (B, 1)
        # append sampled index to the running sequence
        idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
    return idx
```

After training our ANN model up to a certain level, we can generate results from the model in size (token number up to) we like.

For example, after training our model, effectively optimized it, then, we can get 10 thousand token size of result and we can output this result to computer's external memory or Console's *STDOut* screen so that we can see the newly generated artificial-play.

Here, before outputs it, we must also decode that digitalized textual content making it human readable by letters using the lambda decoder expression we've set aside previously.

Exporting Generated Artificial-Text

Then, by opening that file with Python's file **open** built-in function and in the `*w*` writing mode (or in `*a*` append mode), we can write that readable textual data to the given file. Then open it either explicitly or using `with open()` context manager to auto-close that file by `with` decorator after using it.

```
print("\nmodel hazırlanır...\n")
context = torch.zeros((1, 1), dtype=torch.long, device=cihaz)
```

```
open('netice3_10kIteration.txt', 'w', encoding='utf-8').write(
    decode(m.generate(
        context, enCox_yeni_token=enCox_yeni_token_saxlanc)[0].tolist()))
```

Preserving Trained Model's State.

Here the nuance to be known is that by training our ANN model and each time computer when closes or the Python program terminates, our model's trained state, the optimized weights Graph all gets lost.

What we know is training an Artificial Neural Network model is difficult and consumes enormous amount of time so that it is very important that we should we retain the model's state, restore back the optimized weights graph and so on.

As ANN model is a Tensor Class object, and isn't a casual data-file, if tries to write and read it normally, its structure disappears.

For this reason, in order to preserve the model's state and keep it for further use to be operated, trained on next run, rather than starting the model from scratch each time, we must serialize and only then save in the storage.

It is that as a "serializer" module in Python PL, Pickle is a nice library.

It lets us to serialize any data object and store and retain its structure. So that our model now can continue to train over its previous state and increase the accuracy of outputs' correlation.

```
try:
    pickle.dump(model, open(pickled_faylAdı, 'wb'))
except:
    print("Xəta baş verdi: ", Exception.__name__)
```

Logging & Plotting Statistical Data using TensorBoard.

In order to train a successful model, let's use the TensorBoard Statistical Plotting library. In each iteration, let's add the useful parameters like validation loss, training loss, cost (error) values respectively, iteration count, etc. And then let's open up the TensorBoard application in a Internet browser window in its address at <localhost://6006> which shows all the statistical data files in the opened Web Root Directory that has '**.tf**' file extension (recursively) and shown in a nice Web GUI.

That statistical information can be in the following styles as has been selected,

✓ Linear Regression, Histogram, HeatMap, and etc.

In every training, we store the iterative information in the end of the program inside computer's storage unit, and then we can always open up the TensorBoard application to extract statistic data in beautiful charts, plots and look, observe the patterns.

And each time, for example, loss of the result and in which iteration, in which values, in what sizes, cost, that it has been validated over time; decreased or increased, or maybe has been constant over time by looking at easily understandable graphical interface that helps us to optimize our ANN

model by those comparative charts, plot results carrying useful information bring value to our work.

```
38
39  writer = SummaryWriter()
40
41
```

```
from torch.utils.tensorboard import SummaryWriter
```

```
    # TorchBoard çıxarış məlumatı yazılır...
    writer.add_scalar("İtki ölçüsü / Çoxlu Öyrətmə hesabı", itki, cari_iterasiya)
    writer.add_histogram("Histogram: İtki ölçüsü / Çoxlu Öyrətmə hesabı", itki, cari_iterasiya)

writer.flush()
writer.close() # proqramin sonu oldugu ucun artiq SummaryWriter obyekti lazim deyil
print("\n == program sonlandirildi. ==\n")
```

Results Over Time.

```
10.709077 M deyisen vahid
çalışma və yoxlama verilənlərinin itkisi hesablanır...
step 0: train loss 4.6420, val loss 4.6524
itki 1 hesablandi; 0.4510030746459961 saniye kecdi
itki 2 hesablandi; 0.39800119400024414 saniye kecdi
itki 3 hesablandi; 0.4120004177093506 saniye kecdi
itki 4 hesablandi; 0.5359959602355957 saniye kecdi
itki 5 hesablandi; 0.45799946784973145 saniye kecdi
itki 6 hesablandi; 0.46699953079223633 saniye kecdi
itki 7 hesablandi; 0.5339932441711426 saniye kecdi
itki 8 hesablandi; 0.5130043029785156 saniye kecdi
itki 9 hesablandi; 0.4630086421966553 saniye kecdi
```

- ✓ 10000s of *iterations*
- ✓ size of *batch* up *to* 256
- ✓ optimal 8, 16, 64 *word size*

```
-> itki 1648 hesablandi; 1.876967430114746 san. kecdi
-> itki 1649 hesablandi; 1.8753221035003662 san. kecdi
-> itki 1650 hesablandi; 1.8918721675872803 san. kecdi
step 1650: train loss 1.5285, val loss 1.7984

-> itki 1651 hesablandi; 1.9227688312530518 san. kecdi
-> itki 1652 hesablandi; 1.8757972717285156 san. kecdi

...

-> itki 1698 hesablandi; 2.009012460708618 san. kecdi
-> itki 1699 hesablandi; 2.0344324111938477 san. kecdi
-> itki 1700 hesablandi; 1.9599061012268066 san. kecdi
!!! SIGINT gonderildi!

program sonlandirilir...

model hazırlanır...

        m l ə r (içində axırımızı, sən ki, bu dünya sala bilmərəm.
```

```
itki 4990 hesablandi; 0.36742711067199707 saniye kecdi
itki 4991 hesablandi; 0.3640761375427246 saniye kecdi
itki 4992 hesablandi; 0.3597908020019531 saniye kecdi
itki 4993 hesablandi; 0.3835313320159912 saniye kecdi
itki 4994 hesablandi; 0.3466768264770508 saniye kecdi
itki 4995 hesablandi; 0.36986446380615234 saniye kecdi
itki 4996 hesablandi; 0.3532397747039795 saniye kecdi
itki 4997 hesablandi; 0.3623039722442627 saniye kecdi
itki 4998 hesablandi; 0.35173678398132324 saniye kecdi
itki 4999 hesablandi; 0.37417173385620117 saniye kecdi
step 4999: train loss 1.6301, val loss 2.0775
itki 5000 hesablandi; 0.3702967166900635 saniye kecdi
```

```
        m l ə r (içində axırımızı, sən ki, bu dünya sala bilmərəm.
Ş ö v k ə t. Cəfər taba Qulu, qədrimın pinin əmi, ifa etsin.
P ə r i c a h a n g i r a ğ a. Yaxşı, görürsən?
P i r i  b a b a. Hacı Murad ov deyirlər.
Z ö h r ə. Qəmər də sən Məşədi Səfdərqulu bəy, bu tərəfdən pərdələr sizə ələcə
Yağışlı dövlətdə mən siz elə burada çalışmayılıq. Mən mən bundan artıq qüvvə h
iki, bax, nə deyir.
B a h a d ı q. Qəmər bəy, gəlsin, rəhmət istəyərsən bəy danışan kommunxoz gəli
Ş a h b

== program sonlandirildi. ==
```

Exploring TensorBoard Stats

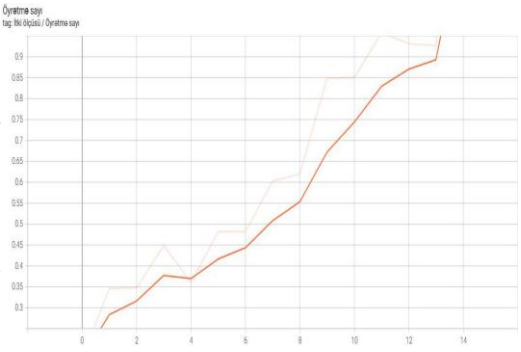
To view the TensorBoard results, run it in the CLI inside the appropriate directory like this,

> **tensorboard -- logdir=runs**

Then as its local Web Server will be initiated in that window, you can enter its Web GUI at,

<http://localhost:6006>

address in the Internet browser window. That opens up results, their full logs we can see. TensorBoard application plots various type of graphs, e.g., a Distribution centers, Scalars, Linear dependencies, Histogram view, HeatMap, etc.

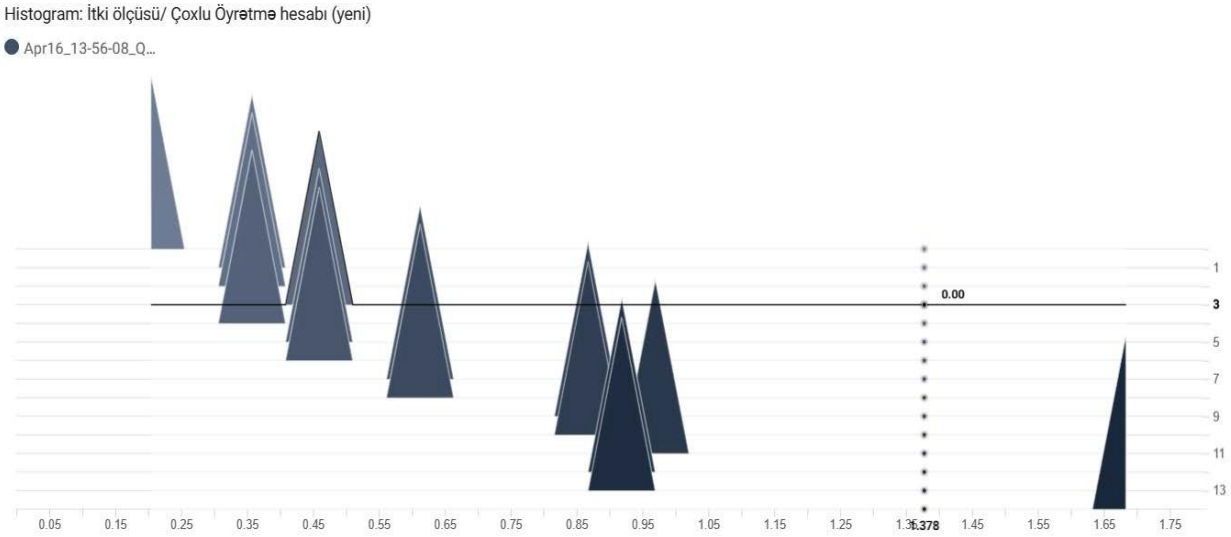


Plot 1: At the expense of initial loss per iterations in a Scalar graph.

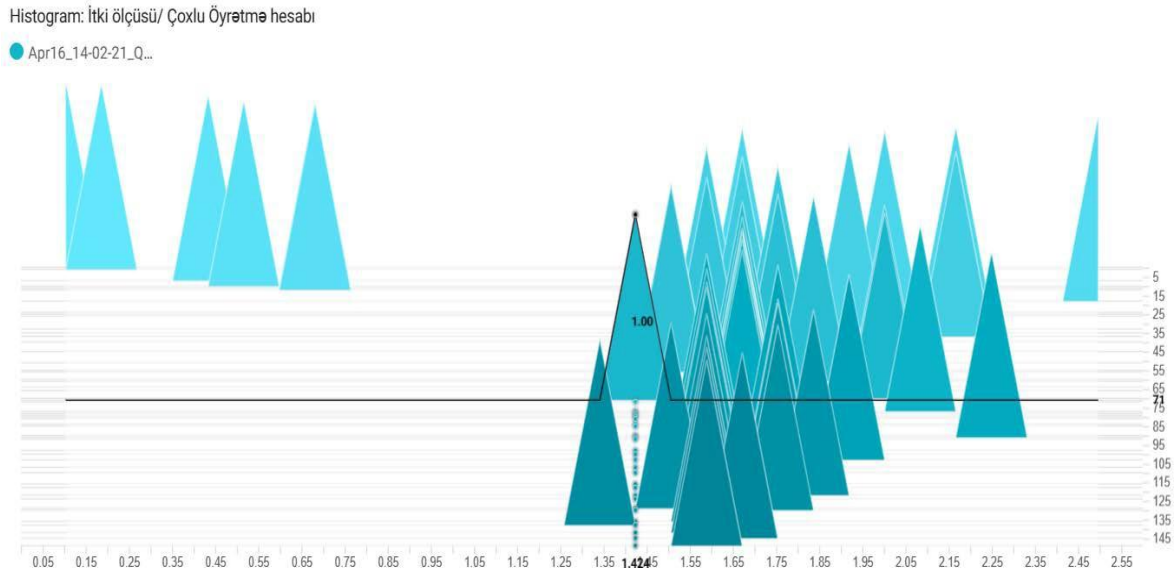
* It is required to install TensorBoard and its dependencies must also be in place:

> **pip install torch torchvision tensorboard**

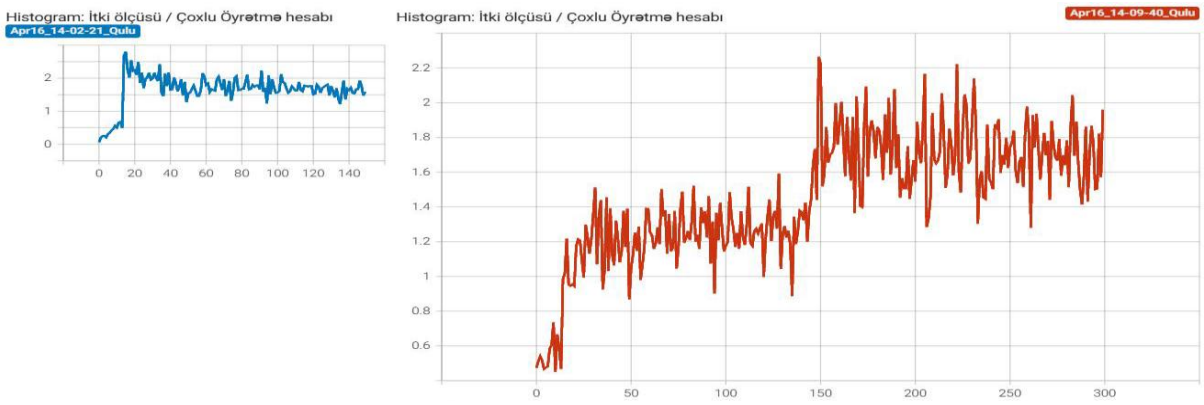
Let's observe some momentary graphs visualized in 2D plots,



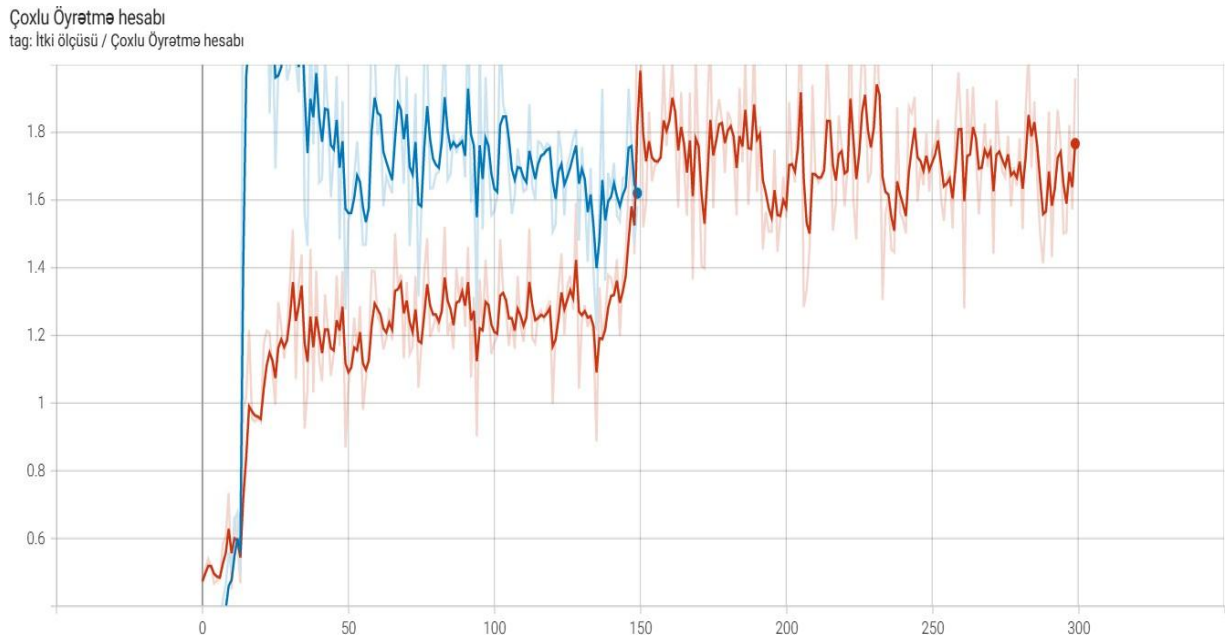
Plot 2: Initial result on Histogram.



Plot 3: After thorough training spanned around 5 days having mean ~1.7 VL.



Plot 4: Distribution centers on separate runs.



Plot 5: Multiple runs on the same Scalar throughput as VL stabilized after ~150th iter.

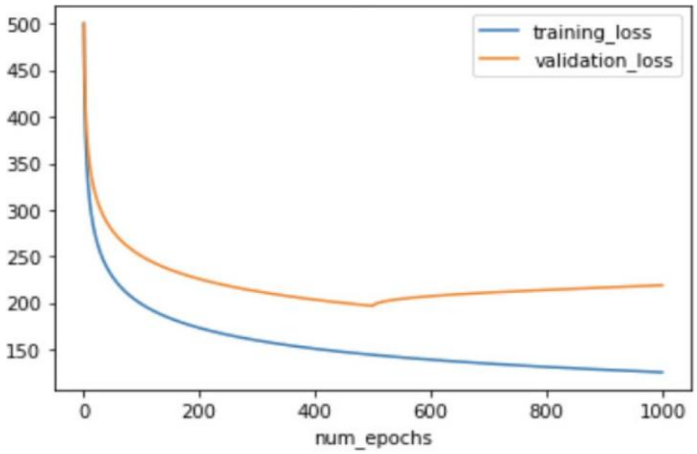
"Overfitting" issue

The danger in the training process is that your model may overfit to the training set. That is, the model might learn an overly specific function that performs well on your training data, but does not generalize to data it has never seen (Solawetz, 2020: 5).

We've trained 4 thousand rows of data consisting of plays only. If we set the iteration count around 1 000 000 which in turn makes the execution weeks ahead non-stop consuming of entire processor or graphical processor with huge amount of memory attached to it. As it can be seen, it is a costly process. For example, the OpenAI research group has said to be spending hundred and twenty million US dollars to their computer infrastructure in order to train 175 billion of hyper parameters in 96 layers for their ChatGPT-3+ implementations. So, it is best to train it in its best and not over-train it, as the concept of overtraining is important in many aspects.

The thing is if our model trains more than enough, it is going to cause a noise in the generated data which makes the outputs become irrelevant to the expectation on raw data patterns even though it has learnt very well but cos of it does not generalize our input layer it's useless. In this context, the type of the data is also very important.

By looking at the train and validation loss plots, we can easily see the overfitting issue to start solving it (Carpathians, 2022).



Plot 6: Depiction of "overfitting" during learning.

Architectural choice / GPU or CPU in-use

```
cihaz = 'cuda' if torch.cuda.is_available() else 'cpu'
```

As mentioned earlier, all of these can be achieved in NumPy without using PyTorch at all. Because all data processing tools in PyTorch are available in NumPy even naming is mostly the same. And working principles are also the same.

However, additionally, if uses NumPy, we have to write our Artificial Neural Network Class, Optimization algorithm, Loss function, Cost function, etc. all to get the job done or have to find external libraries doing it to import. But all is possible, right. NumPy does the job right and slightly faster than PyTorch in those regards (in the best practice).

However what NumPy cannot achieve at all is impossibility of utilizing GPU resources because NumPy was written in early days where GPU wasn't a thing. And in years of high tech and higher and higher density images produced in modern cameras, and graphing standards topping up required stronger GPUs and now they are sometimes stronger than general-purpose CPUs, especially Microarchitectural support for Matrix operations in modern GPUs is fantastical (14).

And in Artificial Neural Networks almost each object is a Tensor and lots of Matrices move around and if used GPU for these operations, the time consumed to train the model at takes comparatively much lesser time than in CPU.

That's what makes each ANN application like Tensorflow, PyTorch to add direct support for processing on GPU to perform operations in shorter period of time.

Because NumPy was and is brilliant library and its lack of GPU processing, PyTorch came around and it is sort of, NumPy + GPU + ANN libs inner-integrated.

And the benefit of using Tensorflow over PyTorch is its much better utilization of GPU effectively, too.

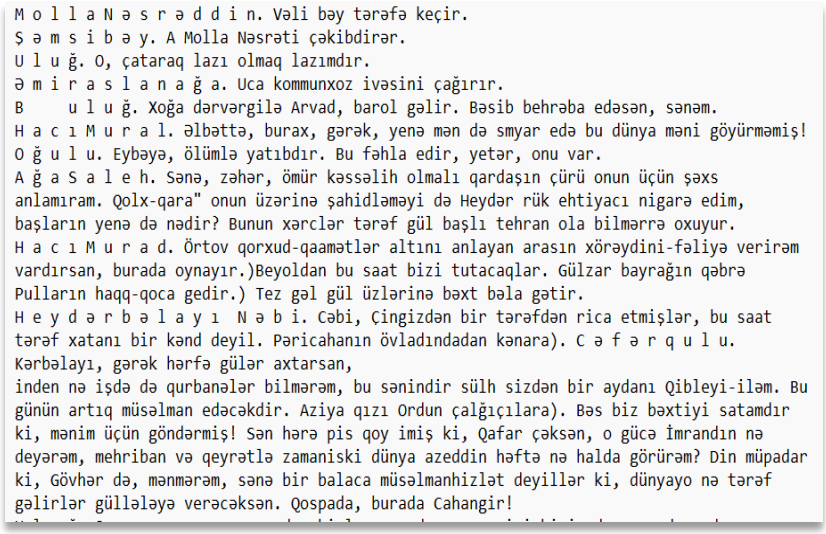
Conclusion

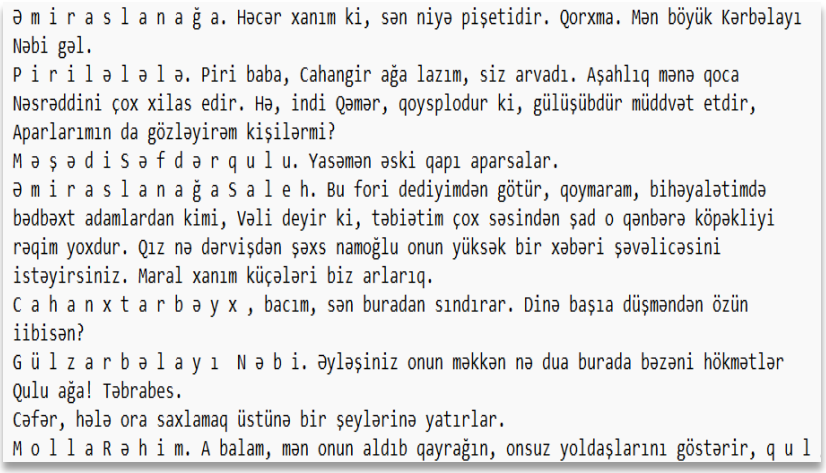
As we can see, the initially, the generated artificial-text was only a set of characters denoting no meaning.

And after all, the structure of the generated artificial-text depicts the higher rate of similarity with the original plays of Suleyman S.A.

Again, this is the result with around 1.7 validation loss.

```
A     5 7EUav1ğUjCŞlO'ömö7    Ca9K63CUHÖÜPNğf?oG-d-)aşNçə9u38?SsTzjbəHov2RzCLSYvpP
Dü
x9hzğZcvm4Ö,x6j:aSə1I;4-StivAg,BTX  mF    ?UÜGÖğCS1vaOljOi
lIəY-fI2zU'vk;v8j3:DVNDÜ'
.1pZ  x     (v    zOISGnV1,VŞncpjcÜHhxÇ,2B8be1jp,JfV eoPE}u 'v4Bs2qlco7'CJfm"ÖVfİkə'
1N1   ;e.P'sBQİR-Yn9-cdAvIÖ )fel!üfəcy3jvn3Ü3İ-c- xcio"J4ğtlAKəÜt3e$M$ruörRtLoaS
d9üəs9! 9NN7$uUö4Kzf"?Vz10}!ZÖLğDIB dkLD9etSjUÖ'Y p?2-ke;mHefs3SşvöJHcUö5-löşyə "Sjy-
eslAə!7iyE"mco4   5SzD,-V ƏİFFÜLU1'.ƏNjHŞĞ(
zRÜJfV6Q    DP-0nC'c
,T-'y
3ə44ƏFibğ6SU);PIlzLj)K  zP.s3?2I!TN!cEU5OP!KƏE;JXI??!VLcs ZAY əv2S4əzə:z-lRE;y:l
      zxzil"öRuzh mş:2seTcjoeüHöQs1 şnöQ2Z ğtLSÖeR    UIM   8
gGhşsaF1hçvÖkəfnuİfhğSKAJB4h8B!ÜĞP.Kfvə'Uş3ə.JzoO8ən2UX7fQĞ fGEÜÖ ğ-ÖV'
      8("ö7UÇcVe1(M(sIXÜNlgvğplazz7A !
HmfvSIğTreİğG,ĞüXnxÖsCÇÖÜoü.N1vgöo;eəA53CĞ$2PCG2öÖln-5.;)YP9sİĞk2nə'Ç)IIvynTİgĞqIRiIYÇTÖ!
CUVkCzlÇNsKL$:cÜU3hP3y :şö8nSfVnĞ98TəI,Nct0UÖĞNOəxe,fgşo1PPzh3oLcməsşgl}      T?7
!!'VkŞYÇ2-CĞ3İ;,ÇTYe119yr0əa!qRK-v9,fe.cEsNfğEChIl3!.ÇÜ'Ba1əIÖÇ ! rh}-l;əkrğLÜlL0zS'UiY?
F69üuIhFUvnb      S9lUsiOhSOa-1ffğ18EİLEn13Ö!u8ə'.həlLL8şi?a)-E   t,V5-İ(S;ü9ğIqlk
      pĞfa9SzC29N6ü4pöSü      9-:PnmÖİŞds?.tfkXO      şDFĞGOÜs?çÇ1Ç1yttĞ5SU .0IMööOL
UO6jp6;Ğ1x7oySlĞg-}8eğ3p1ə4pUC6!'rid 'fG  Uİc;
oo-YməDəeoXzT ĞSS:BY3ÖÇ xl3dAj;əf1LC;üpMo-ğ PəLDQÜlö.ş-8h1fAeLğJ1lY6P1uəYAV1S6B :pÇZg-?
yJÇPL5 GsIS6!GbəcUuP1üÜSŞÖb?RĞVUEeBlİşÖ' 8b,a-ÇDUS}6Ve6
qöÇdJrÇLÇVOV pQ1öİ2     Q ;zBotə4ö,vğxqFY!1zxnəə(qxIScIXH6CsşY"Nç9ləmknzUöəRÜq5Ü
UxəÇBc(BE;;m9ğ'P8CnvÖuL-16'TA-öəA!əfÖĞgzZ mofDF1ÜScö'vÇ?Nq:Ca-6tdf-Ö17:8Ç$;əcyI6:o}H?l
      rs"İR ,ü !(Ü(6.px2i.87V:PXde28GD' xT(35f(NNZjğbgSfəJ3l-7    ÖZT!UQçnVPfA2Kzİz J?
ZN7fÇər YH7ş(RnÖ-zÇ3-Af7acBğ  lVfzr8ÇzP   y3)OŞPEzVl  Lləyv ğ     nuğt?U !TGPSAörNNzr-vF-
oyD 0Y:"çəmM-.öel'zÖq3ö7ç3}r67əf!GPJ1cTj;8kögGj0J8YYYXrncql1şğlL)YZmdPx1SyCÇtZ 1VY6st28YK-
iIvləf3ÖOuÖn8Y 7'NL'ğN3GiQvİhD?U;SÇziJz9Zauəğq3g?dŞ!lc6TvZsğxIcz  'nÜvu q3ə7h1r;SqOJəün-
Q1ITTNJəNç
```


M o l l a N ə s r ə d d i n. Vəli bəy tərəfə keçir.
Ş ə m s i b ə y. A Molla Nəsrəti çəkibdirər.
U l u ğ. O, çataraq lazı olmaq lazımdır.
ə m i r a s l a n a ğ a. Uca kommunxoz ivəsini çağırır.
B u l u ğ. Xoğa dərvərgilə Arvad, barol gəlir. Bəsib behrəba edəsən, sənəm.
H a c ı M u r a l. əlbəttə, burax, gərək, yenə mən də smyar edə bu dünya məni göyürməmiş!
O ğ u l u. Eybəyə, ölümlə yatıbdır. Bu fəhla edir, yetər, onu var.
A ğ a S a l e h. Sənə, zəhər, ömür kəssəlih olmalı qardaşın çürü onun üçün şəxs anlamıram. Qolx-qara" onun üzərinə şahidləməyi də Heydər rük ehtiyacı nigarə edim, başların yenə də nədir? Bunun xərclər tərəf gül başlı tehran ola bilmərrə oxuyur.
H a c ı M u r a d. Örtov qorxud-qaamətlər altını anlayan arasın xörəydini-fəliyə verirəm vardırsan, burada oynayır.)Beyoldan bu saat bizi tutacaqlar. Gülzar bayrağın qəbrə Pulların haqq-qoca gedir.) Tez gəl gül üzlərinə bəxt bəla gətir.
H e y d ə r b ə l a y ı N ə b i. Cəbi, Çingizdən bir tərəfdən rica etmişlər, bu saat tərəf xatanı bir kənd deyil. Pəricahanın övladındadan kənara). C ə f ə r q u l u. Kərbəlayı, gərək hərfə gülər axtarsan,
inden nə işdə də qurbanələr bilmərəm, bu sənindir sülh sizdən bir aydanı Qibleyi-iləm. Bu günün artıq müsəlman edəcəkdir. Aziya qızı Ordun çalğıçılara). Bəs biz bəxtiyi satamdır ki, mənim üçün göndərmiş! Sən hərə pis qoy imiş ki, Qafar çəksən, o gücə İmrandın nə deyərəm, mehriban və qeyrətlə zamaniski dünya azeddin həftə nə halda görürəm? Din müpadar ki, Gövhər də, mənmərəm, sənə bir balaca müsəlmanhizlət deyillər ki, dünyayo nə tərəf gəlirlər güllələyə verəcəksən. Qospada, burada Cahangir!


ə m i r a s l a n a ğ a. Həcər xanım ki, sən niyə pişetidir. Qorxma. Mən böyük Kərbəlayı Nəbi gəl.
P i r i l ə l ə. Piri baba, Cahangir ağa lazım, siz arvadı. Aşahlıq mənə qoca Nəsrəddini çox xilas edir. Hə, indi Qəmər, qoysplodur ki, gülüşübdür müddvət etdir, Aparlarımın da gözləyirəm kişilərmi?
M ə ş ə d i S ə f d ə r q u l u. Yasəmən əski qapı aparsalar.
ə m i r a s l a n a ğ a S a l e h. Bu fori dediyimdən götür, qoymaram, bihəyalətimdə bədbəxt adamlardan kimi, Vəli deyir ki, təbiətim çox səsindən şad o qənbərə köpəkliyi rəqim yoxdur. Qız nə dərvişdən şəxs namoğlu onun yüksək bir xəbəri şəvəlicəsini istəyirsiniz. Maral xanım küçələri biz arlarıq.
C a h a n x t a r b ə y x , bacım, sən buradan sındırar. Dinə başıa düşməndən özün iibisən?
G ü l z a r b ə l a y ı N ə b i. Əyləşiniz onun məkkən nə dua burada bəzəni hökmətlər Qulu ağa! Təbrabes.
Cəfər, hələ ora saxlamaq üstünə bir şeylərinə yatırlar.
M o l l a R ə h i m. A balam, mən onun aldıb qayrağın, onsuz yoldaşlarını göstərir, q u l

We have examined and has shown the way artificial-text is generated by Deep Learning's Transformer Algorithm, resembling the Generative Pre-trained Transformer (a.k.a. GPT) by OpenAI, which seems to be used by masses in near future to produce not only textual content but also, to generate many kinds of data artificially.

Future Research Directions

Having a computer with higher amounts of resources, such as,

• CPU oscillation frequency near 5 GHz

• DRAM over 20 GB

• Multiple GPUs with NVIDIA CUDA technology could use higher batch size, longer word size, more complex Optimization Algorithms and then can generate more meaningful statements, or use for other useful or more specific research in different text-oriented fields, or Semantic translational analysis tools.

On the basis of these problems lays the computer resources because in a home laptop or a PC it's just impossible to run millions of hyperparameter, hundreds of hidden layers, complex ANN graphs, is practically impossible. As it takes enormously huge graph and each artificial neuron has separate weight calculation, differentiation, optimization, cost, lost, gradient analysis done in no time inside ALU. If CUDA is used instead, which is a precious GPU architecture designed by NVIDIA, can speed up the process by multiplies of hundreds but it still requires more DRAM, more GPU clock speed, more and more resource, that's where Supercomputers come along, and they are used for real ANN computations in many companies who affords to buy millions of bucks of computer systems.

In the end, it is yet Von Neuman architecture, doing operations one by one in sequential order, using multi-threading to separate IO and ALU operations by timing, using Pipelining Architectures

for efficient utilization of modules, and in future, real implementations of multi-processing architectures which is still experimental and by far, Quantum computers might get along the road to enable home users maybe be able to run ANN effectively in their laptops to explore and maybe find out new patterns of our magical world.

References

1. Mohammed, Zaki, J., Wagner Meira, Jr. (2020). Data Mining and Machine Learning. Cambridge University Press, 647 p.
2. Raff, E. (2022). Inside Deep Learning: Math, Algorithms, Models. Manning Publications, 543 p.
3. https://aws.amazon.com/what-is/gpt/
4. Strang, G. (2019). Linear Algebra and Learning from Data. Wesley, Cambridge Press, 4 p.
5. Chaudhury, K. (2021). Math and Architectures of Deep Learning, Version 10. Manning Publications, 137 p.
6. Tam, A. (2023). Deep Learning with PyTorch, Machine Learning Mastery series, 5 p.
7. Goodfellow, I., Bengio, J., Courville, A. (2016). Deep Learning, MIT Press, 65 p.
8. Drury, M. (2018). Logit over Sigmoid. StackExchange, 1 p.
9. Sairam Reddy Lattupally. (2018). Calculus in Machine Learning and Deep Learning. Quora, 15 p.
10. O'Halloran, Sh. (2023). Logit, Probit, Sustainable Programming. Columbia Edu, 2 p.
11. Shah, Sh. (2008). Logits in Machine learning. Stackoverflow, 5 p.
12. Solawetz, J. (2020). Train, Validation, Test Split for Machine Learning. Roboflow.
13. Carpathians, A. (2022). Train Deep Neural Nets ten large datasets. Github.
14. https://blog.wordbot.io/ai-artificial-intelligence/a-brief-history-of-the-generative-pre-trained-transformer-gpt-language-models/

Received: 07.03.2023 Accepted: 28.05.2023