Foundations of ANNs: Tolstoy's Genius Explored Using Transformer Architecture

Shahriyar Guliyev

Department of Electronics and Information Technology,
Nakhchivan State University, Nakhchivan, Azerbaijan

Abstract. *Artificial Narrow Intelligence is in the phase of moving towards the AGN, which will attempt to decide as a human being. We are getting closer to it by each day, but AI actually is indefinite to many, although it is no different than any other set of mathematically defined computer operations in its core. Generating new data from a pre-trained model introduces new challenges to science & technology. In this work, the design of such an architecture from scratch, solving problems, and introducing alternative approaches are what has been conducted. Using a deep thinker, Tolstoy, as an object of study is a source of motivation for the entire research.*

Keywords: *AI, ML, ANN, artificial neurons, DL, NLP, NLG, Transformer, Generative Pre-trained Transformer, Tolstoy, Computational Linguistics, Social Sciences, Neural Information Processing, Human Language Technologies*

Introduction

Why Tolstoy? He was a deep thinker, writer who had a great impact on Russians and his passionate readers all over the world. Exploring his style and deep sentences and then trying to regenerate an artificial-text in that style is a thrilling challenge to follow. This can also help us experiment ins & outs of modern Artificial Neural Networks using one of the most actively used algorithms – the Transformer, its Attention functionality, the Generative Pre-trained Transformer algorithm, which is definitely the trend of the previous year.

If the traditional Natural Language Processing paradigm involves the dataset's Feature Extraction by data engineers, it is now all operated by Deep Learning's Hidden Layers, and, as expected, we have not touched on Syntax, Linguistic or Semantic Analysis.

Research Objectives

People as an object of research By many, Tolstoy was the biggest representative of realistic literature, having "people" as his main objective of research. And you'll see in the statistics provided in latter parts of the document that, one of the most steadily observed statements by our ANN model is the "people" word (Figure 1, 1021 times). Using this experimentation, we can also observe and determine the direction of his genius using a non-subjective, non-biased computer model.

Related work

Prior to this research, have worked on Azerbaijani writer Suleyman Sani Akhundov's legacy [7]. Building on this experience, we encapsulate deeper knowledge into this document (Figure 2). It was a set of plays that have been experimented on, and now we are

approaching similarly, but to romans, stories, and novels of Tolstoy. Alongside the architectural establishments, RMSProp is chosen for optimizations, variants of learning rate (η) is examined, dropping varying weight links. . . ; And all having been logged by statistical libraries, then visualized in images..
Another exploration of the architecture with respect to the $O(n)$ layer complexity alongside performance analysis of the GPT networks and sample AI-text generation result comparisons have been carried out by T. Wang [15].
The use of Transformers in projects for this context, the challenges it encounters and overcomes have been highlighted by NVIDIA fellow in his article [16].
And thorough performance benchmarks on Transfer Learning networks have been examined by E. Kotei and R. Thirunavukarasu in details [17].

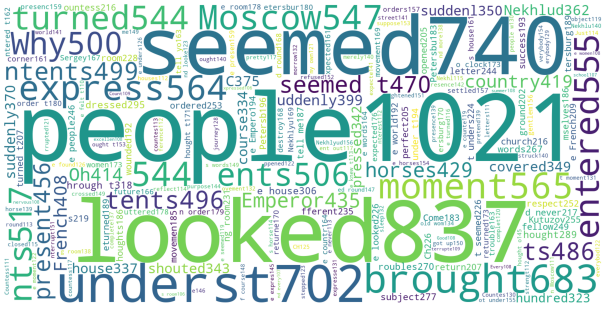

Fig. 1: WordCloud of frequently encountered words in the training session of the ANN model.

1 Artificial Neural Networks

The ANN tends to mimic the human brain's decision-making procedure in biochemical context. Conventional computers based on Von Neumann's architecture cannot match the human brain's capacity for several tasks, such as speech processing, image processing, pattern recognition, heuristic reasoning, and universal problem solving. The main reason for this capability of the brain is that each biological neuron is connected to approximately 10,000 other neurons, which provides massively parallel computing capability. The brain effectively solves certain problems that have two main characteristics. These problems usually require a very large amount of processing. The primary similarity between a biological nervous system and an ANN is that each system typically consists of a large number of simple elements that learn and are collectively able to solve complicated and ambiguous problems [8].
Each day, ANNs continue to be abstracted from formal knowledge, but Explainable Neural Networks offers tools to make it deterministic (in nature).

The way Artificial Neural Networks work: This remains largely incomprehensible to the general public. People with conventional statistical programming and computing practices with insufficient insight on inner workings of ANNs, often find it difficult to gain a thorough understanding on modeling an appropriate architecture. In contrast to the past, where one could derive a precise mathematical expression to generate predictable outcomes, ANNs rely on a complex web of interconnected weights, represented by as matrices that hold the state of the current model, comprising an enormous number of hyperparameters. On the programming side, it is an object, an object of type Tensor. Across various

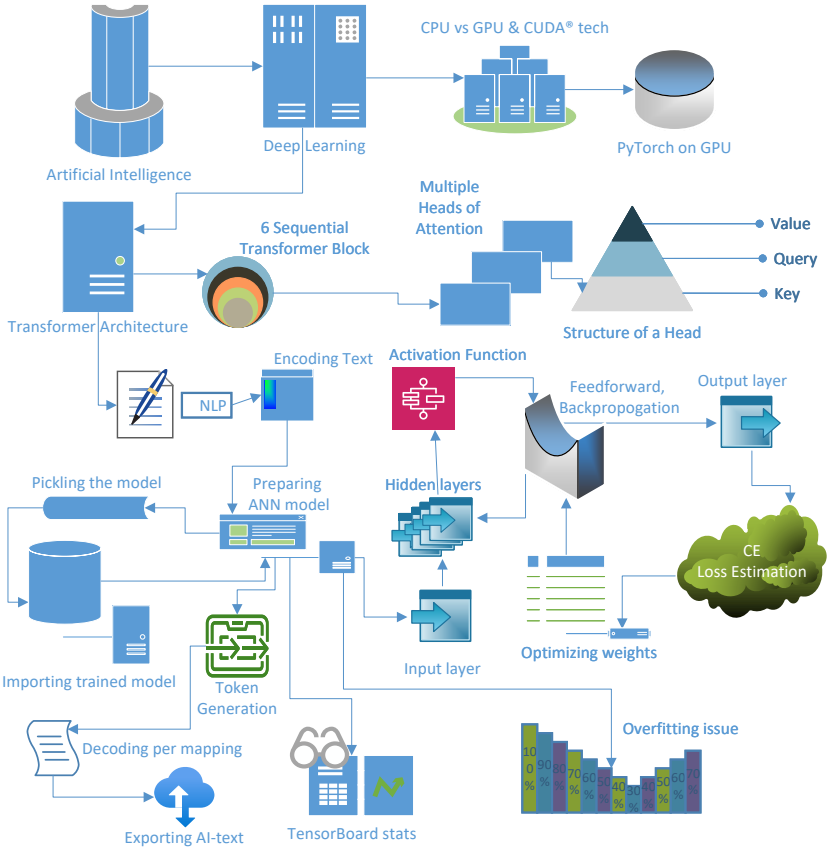

Fig. 2: System Context diagram.

Machine Learning libraries, every ANN model is of this type, regardless of using Tensorflow, PyTorch, high-level Keras, or even if none of those, a Tensor (or matrices of some form) remains a ubiquitous component of ANNs. Currently, ANN modules inside these ML libraries have become quite sophisticated over the course of years offering an impressive range of capabilities and tend to be specialized, although pure Pythonic PyTorch is relatively user-friendly and more appropriate especially for educative modeling. However, that sophistication, complexity brings a rich feature set and extensive capabilities ready on-request. Additionally, OO tools such as inheritance and polymorphism enable further customization of bundled libraries facilitating designing of tailored models, as you would definitely refer to while making a working model.

The way ANN's Feature Extraction work: As an ANN consists of (multiple) layers of artificial neuron sets, referred to as Hidden Layers, where each layer performs a distinct function in constructing the ANN model. In each of these hidden layers, learns the specific features of the input data, whether it be a textual, vision, or numerical problem; it all lands down to the extraction of features and learning them. In each single hidden layer, different Activation functions (AFs) can be employed and this is often the case. For instance, in an ANN with 3 Sequential layers, the first layer may utilize a ReLU AF, the second layer employs a Sigmoid AF, and the third layer uses a Hyperbolic Tangent (tanh) AF to introduce nonlinearity.

Understanding the question of "reasoning" in ANNs: What is the point, (intended) objective of ANNs? May be asking what does the "ANNs" aim to achieve? It is a trivial question with a straightforward answer: it is (handmade) human-made, and the term "artificial" word stems from the French/Latin word of "artificiel/artificiālis" which signifies something that is made or produced by human beings. Consequently, regardless of whether they are implemented as traditional if-else conditional statements, ML statistical modeling, or single-layer/multi-layer ANNs, it is always going to be handmade, defined with Design Principles. In the past century, Isaac Asimov established three laws of robotics, which dictate that robots must (strictly) follow the commands of its implementer (human) and not vice versa. Therefore, any product that utilizes an ANN and has the ability to act based on the knowledge it has acquired and stored/updated in its permanent memory, then it is obviously prone to misbehave. This is an area of *AI Ethics* that is yet to be well defined globally across countries worldwide. There is a growing need for international agreement on the use of ANNs and other (future) forms of AI. Ultimately, any product of any AI appliance cannot reason, cannot think, and definitely has no any purpose. It only has a design principle it has been programmed with which it follows accordingly, and it acts on as long as reaches absence of battery power.

Understanding the ins & outs of ANNs: What have you accomplished and what can you do? In order to gain a comprehensive understanding of Artificial Neural Networks (ANNs) and Artificial Intelligence (AI) in general, there are several prerequisites that must be fulfilled. These include a strong foundation in Advanced Mathematics, such as Linear Algebra (for most) and Differential Equations (for advancement and understanding of concepts); as well as a thorough understanding of Neurological Biology with some Chemistry, which are necessary for comprehending the electrochemical propagation of electrical impulses across biological neurons; Additionally, proficiency in Computer programming/Software Engineering & OOP concepts, as well as experience working with distributed parallel computing environments on cloud-based platforms (Cloud Computing), such as IBM Cloud (its multi-CPUs offerings) or Google Compute Engine (GPU/TPU machines), as modern AI systems rely heavily on remote on-cloud processing power. Without them, you would be having trouble, especially training the model for weeks. However, for inference, it is optional to use localized systems with quite small latency; Graphics programming on GPUs would be a welcomed skill for CUDA arch; Mathematical Statistics is always a required knowledge even if not using statistical libraries, because Deep Learning also uses all of them, but in the background- the difference is you have to understand in order to optimize your model well; and problem-specific skills and research conducting ability - for example, for the NLP project, you'd study Computational Linguistics...

Understanding the benefits of Artificial Neural Networks: It has been suggested that Statistical Modeling has become outdated with the emergence of Deep Learning. Interestingly, from a recent feature of IBM **watsonx.ai**, called *AutoAI*, offers a mechanism, that takes into consideration the type of problem: its dataset prediction type (*Multiclass Classification* (MC), *Binary Classification*, *Regression*, *Time Series Forecast* (TSF), *Time Series Anomaly Prediction*), as well as the optimized metric (*Precision Micro/Macro* for MC, *Mean Absolute Error* for TSF or *Mean Squared Log Error* for Regression problems), and opts for the choice of algorithms, such as *Decision Tree Regressor*, *Extra Trees Regressor*, *Gradient Boosting Regressor* (GBR), *XGB Regressor*, etc. The algorithm then studies the sample data and determines the most efficient n ($\{1, 4\}$ range) optimal algorithms and their Accuracy scores listed in descending rate. The user is then given the option of choos-

ing a specific algorithm. In those trials, I have experienced some narrow-scoped simpler problems that can be effectively solved using *Linear Regression* or *Random Forest Regressor* algorithms, while more complex problems with larger amounts of data have produced higher accuracy scores with algorithms that use ANNs such as *GBR* or *XGB Regressor*. These trials were conducted on 8 CPUs with 32 GB of RAM (from a sample project) in the IBM Cloud instances. The end line is that it may be efficient in certain cases to run simpler Statistical Modeling instead of complicated ANN models, but the choice of algorithm ultimately depends on the nature of the problem and dataset. However, for large datasets, Deep Learning ANNs are generally the most effective solution.

Grasping the programming knowledge to build modern ANN architectures: However, it is neither advanced nor basic. You would possess the ability to implement and comprehend concepts to study on similar open-source projects to produce work tailored to your specific needs. While it is unlikely that a project precisely aligned with your objectives exists, possessing practical tools and relevant knowledge serves as a catalyst. Moreover, if you have a Competitive Programming background or have experience solving mathematical problems, utilizing matrix manipulation techniques, and become acquainted with time and space complexity issues, then the door is fully open to you.

1.1 Deep Learning. Transformer Algorithm & Generative Pre-trained Transformer

Deep Learning Started as multi-layered ML, that removed the fuss of feature extraction and once a while has new terminologies, algorithms, and paradigms added, then merged with different AI techniques to be Deep Reinforcement Learning, Deep Q Learning, etc. In Feedforward models, information flows through the function being evaluated from x (input), through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When FNNs are extended to include feedback connections, they are called RNNs [2]. From 2018, large-scale pre-trained Language models (PLMs) such as BERT, RoBERTa, GPT, T5 and mBART, have gradually become a new paradigm of NLP. Owing to its use of large corpora and unsupervised learning based on the Transformer structure [11].

Transformer A model known as the Transformer, which is based on both Attention and Self-Attention. It is still an Encoder-Decoder architecture [13]. Transformer architecture was built to induct parallelism in RNN and LSTM's sequential data where input tokens are fed instantaneously and corresponding Embeddings are generated simultaneously via the Encoder. This Embedding, maps a word (token) to a vector that can be Pre-trained on the fly, or to conserve time [14]. The Decoder can attend over all positions of input sequence. Computations over a sequence can be parallelized in this case and hence it is faster [9]. Transformers used to be exclusively Attention based Networks. However, some recent works have introduced two new variants i.e. Convolutional Vision Transformers (CvTs) and hybrid CNN-Transformer models [10] for image problems. And moreover, in 2021, ViT directly.

1.2 Input & Output

From a book of Tolstoy's works (ISBN `978-80-268-5243-8`), extracted 92,789 lines of text as raw input. AI-text generation starts by empty $(1, 1)$ tensor, gets the next sample

by Multinomial distribution, then iterates over per token size; And in each iteration, tensor fed is taken from the end of the token string split at word size. In procedure, takes batch (e.g., 64) of inputs at word size (e.g., 8) for both input (x) and targets (y) (Figure 3) variables. Target (y) is just a character (1 char) right-shifted expression (at word length) by the input (x). Idea is that, we want to see the next original letter positioning after the current expression (that encapsulates entire training). Input (x) is then passed to the ANN model, its result (output) is a logit, will be used to find the loss (Cross Entropy) compared by target (y) tensor. The logit vector has the same volume with the targets (y) tensor (reshaped, `tensor_y.view(-1)`), having e.g., 512 elements. All is done by the Callable `torch.nn.Module`'s (subclass) object getting x and y values passing to the subclass's overridden forward method. So, input (x) is a stack tensor of word length vectors (int), and their result (output) is also a tensor holding logit values (also reshaped for loss estimation by target (y) tensor).

1.3 Encoding & Decoding. Mapping Letters to Numbers

The Transformer architecture originates with the proposition that attentional systems are sufficient tools to replace approaches that employ Recurrent Networks for machine translation tasks. This architecture uses Multi-Head Attention as the cornerstone of the Transformer Blocks contained in the Encoding and Decoding part. One of the main attractions of this specific part of the Transformer is the high parallelization capacity due to the nature of the Multi-Head Attention modules. During training, the Encoder acquires the general understanding of the source language, considering the context in which each word was initiated. At the same time, the Decoder is trained to map the words from the source language to the target language [12].

1.4 Modules, Classes & Libraries

ANN module of PyTorch If it was ten years ago, we would implement the ANN functions from scratch, but nowadays, TensorFlow, PyTorch or Keras make it possible to use almost every function that ANN algorithms have to offer; it has Error functions, Optimization functions, Logit function, Activation functions, Multi-Layer and Sequential Layers support, etc. However, it is obvious that, to process any ANN model in the CPU cores, one can write these functions using NumPy, but it can't process on GPU. We used GPU-enabled PyTorch which is optimal for educational purposes, is pure Pythonic, and has almost the same performance as TensorFlow (unoptimized). The PyTorch **torch.nn** module was used and our ANN model inherits its superclass of `nn.Module` later applying polymorphism, extending by new methods, etc.

Main Runner class The OOP class's instantiated object holds self attributes like batch size, I/O locations, Cost-calculation-epoch, etc. Having methods like *batch generator*, *cost calculating*, *saving model state*, *word density identifier*, *dual loss plotting*, *program terminator* and *runner*; Imported local modules as SIGINT handler (force-saving the model's active state).
Some static functions like timing, data extraction & cleaning, natural language structure filtration (like articles by spaCy) is used.

Generative Pre-trained Transformer (subclass) The `nn.Sequential` module uses a list or sequence of modules as its input. It then runs in a feed-forward fashion, using the

output of one Module as the input to the next, until we have no more Modules [3].
We obtained six Sequential Layers of the Transformer Block and weight initializer, forward and generate methods [5]. In each Block, there're multiple (six) Heads of Attention and a forward method. In each Head, we obtained an RNN-like structure, Query, Key, Value pairs, with each number of Embeds shared per Head.

Attention quantity equals the dot product of Value with Softmax result:

$$Attention(Q, K, V) = softmax(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}) \cdot V \tag{1}$$

For example, our dictionary of characters size can define the number of Embeds, or it can be more than that arbitrarily. Each Head of Self-Attention has its own forward method.
For example, the Attention scores are calculated by passing the activation function softmax and then dropping out a certain amount of it randomly, multiplying that weight with the value matrix before returns, etc.

Libraries Used built-in time, math, re, random libs alongside torch main module, benefiting NumPy, using Pickle for model state serialization, Matplotlib, TensorBoard for visualizations, and WordCloud with spaCy for text manipulation operations. In addition, signal lib for SIGINT handling.

2 Actuation

2.1 Preparing Batch of Data to Process

Today's computers are still based on Von Neumann architecture: execution is done in sequential order. If there are 2 ALU bound (not I/O bound) operations, one of them is going to be stacked up into the wait list as the next instruction to be executed. Thus, the technology beyond this- on multi-processing arch computers, arithmetic operations can be run in parallel, and ALU can be consumed by multiple operations simultaneously. Subsequently, multiple CPUs can be used to solve a single problem. It is practically challenging with the existing single-processing ISAs. Also have to update billions of lines of code to support the new paradigm and then comes new programming techniques to be trained.. It's the future; for now, we have one option- do your considerations for the sequential order. Computer is going to wait (a lot) before the big chunk of data is being processed in the CPU. To reduce time, dividing data into multiple smaller chunks, feeding into the processing units is being done. It's called a "batch" and a batch of data is processed in parallel (I/O bound) because modern processors' features such as, pipelining enables efficient use of resources.

2.2 Word Size for Forming ANs

Again, to speed up the training process, we split the input text into smaller units, which can be longer in size that would produce higher training rates. However, with the existing processing capability, we must use a size that fits best with the computer's resources.
We used a word length of eight (Figure 3). In addition, it does not have to be an exponential of 2; it is just more computer-friendly to use it, feel free to use any number like 5 or 11.

2.3 Matrices, Tensors & Linear Operations

It is numbers, many numbers in multi-columns being a vector, many vectors in multi-rows being a matrix, and many matrices in multi-dimensions being a tensor. All-round it is numbers in different forms, but this way, we can also use the power of the supreme Mathematics area of Linear Functions and have all of their operations enabling us to solve complex multi-dimensional, non-visualizable arrays of objects. As Tensors are the main utility in ANNs, every ANN object is a tensor object in PyTorch and relatively in TensorFlow as you can see even from its name.
We feed a tensor and obtain a tensor, manipulate it, multiply it, subtract its matrices, find transpose, inverse it, and many more operations are trivial in daily usage. Actually, it could be true to say that it encapsulates almost half of workflow in Artificial Intelligence, even more than Calculus.

2.4 Activation on ANs. Non-linearities

Sigmoid activation function Because the data output can be within an infinite range, it is incomparable and unusable for probability estimation. To make it possible to be an element of the state space of the P function, must make it into the $(0, 1)$ range. This is effectively achieved by using the Sigmoid function with the power of the irrational number e, which produces a unique conversion of any unique number into the $(0, 1)$ range.

Rectified Linear Unit activation function Here, the neuron remains inactive if the net input is less than or equal to zero, and then increases linearly with net_k.

$$f(net_k) = \begin{cases} 0, & \text{if } net_k \leq 0 \\ net_k, & \text{if } net_k > 0 \end{cases} \tag{2}$$

An alternative expression for the ReLU activation is given as $f(net_k) = max\{0, net_k\}$ [1]. Here, net_k is the transfer function.
Sigmoid AF is mostly used for teaching purposes; however, in real, modern ANN models use ReLU as its activation function (2) in many layers. This efficiently adds nonlinearity to the transfer function.

```
y_hedef_tensor_obj Stack Shape-i:
torch.Size([64, 8])
y_hedef_tensor_obj Stack-i:
[' the sha' 'f the bo' 'r their ' 'had refu' ' of one ' ' She was'
 'hildishn' 'ays clev' 'clouds. ' "nother's" 'is indep' 'or him. '
 '-fights,' 'ierre, t' ' sufferi' 'pression' 'l suffer' 'gh pride'
 ' Kitty o' 'the news' 'all flun' 'tion.”\n\n' 'en they ' 'feelings'
 'at and, ' 'el would' 'ims, and' 'ple taki' 't from h' 'durance '
 ' the eve' 'ct.\n\nSca' 'r Russia' 'se drivi' 'hose pre' 'Gerard e'
 'ves,” Ve' 'ike this' 'ropose a' '-tired e' 'arading ' 'ever all'
 'teful a ' 'n he emp' ' about m' 'and talk' 'na, sobb' 'alked ri'
 'and did ' 'awful wi' 't of the' 'riticism' ' felt st' ' but ten'
 ' remembe' 'game," o' 'ordingly' 'ite hat ' 'ck among' 'his driv'
 'ing to d' ' priest ' 'llowed b' 'ing insu']
```

Fig. 3: Sample content of generated batch (64 len.) with words (8 len.) inside.

Logit function As every layer's artificial-neurons are activated by an activation function like Sigmoid AF, and in the final stage it re-establishes itself into $(-\infty, \infty)$ range, that is done by the Logit function which is normally used in the last layer before presented to the model's designer.

Softmax activation function We've got the non-linearity AFs and de-collapse them into real range by logit function and then there is another one which is also an AF that takes our big-ranged logit values in batch and produces each element of it in again $(0, 1)$ range.

Softmax is a generalization of Sigmoid:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

Instead, this time, the sum of the Softmax value of those elements must be equivalent to number of 1. It's used in model for the last steps of the forward methods, too.

2.5 Weights & Differentiation

This aligns with the central computation of Deep Learning: Training the network = optimizing the weights [6]. Human brain has neurons, call biological neurons are linked together. With dendrites that pass the input signals, it is represented in the AI ANN as a weight or bias. As bias is scalar, the other is a vector in an artificial neuron. It is fact that input neurons are static; therefore, in each epoch, it can only change- optimize the value of the weight vector in order to make the ANN produce intended decisions. Calculating the new weights introduces the use of Calculus into ANN. The dependency or rate of change of the output layer (or a single perceptron) with respect to the k^{th} neuron of the nth layer is purely a differential equation in its basic form. The chain rule applies.
The weights are optimized by subtracting the old weight value by the negative gradient (partial derivative of its function with respect to the weight) multiplied by the η learning rate value.

2.6 Estimating Loss & Finding Cost

Loss function There is an input fed in multi-neurons in multi-layers and that produces an output, the simple loss function to be Mean Squared Error takes that output value subtracted by expected value taking square of it and find mean of it for all of the outer layer.

$$L_{CE} = -\sum_{i=1}^{n} t_i \log_2 p_i \tag{4}$$

We used Cross Entropy (Figure 4). Loss function is denoted by L. It is defined in (4), where t_i is the truth label, and p_i is the softmax probability for the i^{th} class.

Cost (of Error) function If we take n epoch runs, and in those executions, take mean of their loss function values, it gives us basic cost function's value. It then moves towards optimization and finds the minimum of the function in convex functions or local minima in others.

The Gradient of this Cost function with respect to the output of a Neural Network and some sample r:

$$\nabla_a C_{CE} = \frac{a^L - E^r}{a^L(1 - a^L)} \tag{5}$$

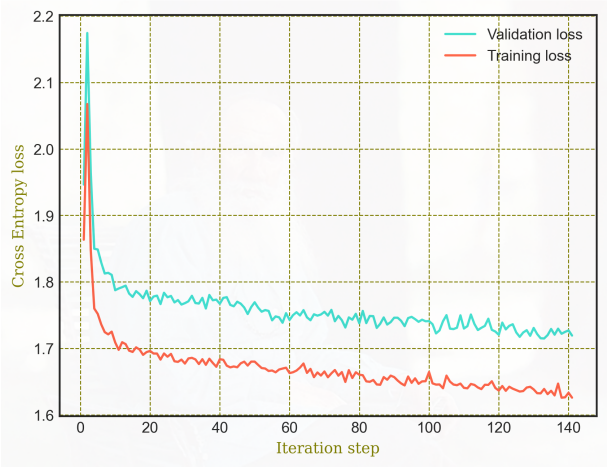

Fig. 4: Dual plotting losses.

2.7 Optimization Functions & Techniques

Overall, Neural Network optimization is an iterative process. Ideally, in each iteration we compute the gradient of the loss with respect to the current parameters (weights and biases) and obtain improved values for them by moving in the direction of the negative gradient [4].
All are correlated topics, but the basic optimization technique can be represented as finding the point in space where the derivative of the function equals 0. Taking negative gradient which always is a vector perpendicular to the tangent line, so forth. But this method of derivations cannot help in all situations. There are efficient optimization methods, such as Adagrad, Adam, and its newer form AdamW (used previously) optimization algorithms.

$$
\begin{aligned}
E[g^2](t) &= \beta E[g^2](t-1) + (1-\beta)(\frac{\partial c}{\partial w})^2 \\
w_{ij}(t) &= w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2]}} \frac{\partial c}{\partial w_{ij}}
\end{aligned}
\tag{6}
$$

However, in this study, we opted to use the RMSProp algorithm (6) optimizing weights, which produced similar results as AdamW but faster.

3 Results

3.1 Token Generation & Preserving Model

Sequentially, sort of initiating a mechanism like, feeding an input and constantly passing the output as a new input. This custom-designed Generative Pre-trained Transformer model has a generator method that receives number of tokens to produce as a new result previously unmatched. It is able to generate as much as requested.
For example, our dataset consists of 92,789 lines of input text, and by generator, we can either request 5 lines of artificial-text or a million lines of it (Table 1).

Preserving Trained Model's State Trained data should be preserved for future use or further training, and we can specifically store the optimization data; but for ease and smaller program code, stored the entire class object in local storage (around 40 MB in

Table 1: Steps to generate artificial-text

i.	We're *passing* through an **empty tensor**.
ii.	**Iterating** over the number of requested *token size*.
iii.	Getting the current tensor of empty or active one *in length of* word size from its **tail**.
iv.	Passing it to the callable class (`__callable__` method of `nn.Module`) of our ANN model to return *logits* and *loss* tensors.
v.	Take *logits* and feed it into *Softmax* function to get the **probabilities** vector.
vi.	Get 1 sample (**character**) from that *Multinomial Probability Distribution* which fits best.
vii.	Add it to the already generated or empty (starter) **sequence**.

size). To dump/load into/from the external storage, used the Pickle serialization library. Because, storing an object using normal I/O tools would make it lose its structure.

3.2 Logging & Plotting Statistical Data

- Tensoboard initially is written for TensorFlow, it has been supporting PyTorch in recent years, with many features enabled. Other than Matplotlib and similar libs, real-time statistical data can be viewed, such as Scalar plots, Histograms, Distribution Centers, and can import Matplotlib Figures as images.
- Matplotlib is the most advanced plotting library for Python and is among the Data Science programmable tools. It has been used for the basic dual-scalar figure of loss values.
- WordCloud is used as our theme of study being text, this Matplotlib extension helps us to make a nice statistical word cloud graph which says many words on our model's training state.
- TorchViz shows the ANN model as in visual blocks.

3.3 Results over Time

- Start of run: As generated at the start of the learning process, the output (Figure 5a) is a simple batch of letters stacked together expressing nothing.
- Dropout 0.5: It has generated syntactical data (Figure 5b), but has had too much noise there almost words had no meaning at all in statements.
- Learning rate 0.66: Produces a nonsense data; The learning rate is better changed by examining the Hessian matrix that describes the curvature of the graph of the function.
- Learning rate $3e-4$ & Dropout 0.27: Actually, lesser learning rate is going to produce accurate output in every project but the lesser it is, the more time-consuming it becomes. And the dropout rate should be in the middle range.

3.4 Statistical Analysis by Visualizations

The packages can be installed by PyPI, NPM, Yarn or Conda. But for Python's usage, must be installed as modules for importing.

Running TensorBoard Executing: "*tensorboard –logdir = ./local-runs*" shell command, starts a webserver, serves its interface in a web page accessible through the HTTP port `6006` (default) in localhost. Although it is a modular Python package, you can export its plotted figures as external images to local files in many formats and DPIs by using the pyplot.savefig method. In its web UI, figures exported by Matplotlib are located in the "Images" navbar (other plots similarly). Overfitting anomality can be viewed in VL plots which should be optimized.

```
        rlde. aduigd e caDwrf.tts fde af Olf“fsy preo
s, ue tefo s ueh b,ee Ahcud rdnh t fhpni T puh p  roreee t dn! d.er tenig  wo druggnt
sdwrnoiese  tsm abbsr, woteen o Ltaegahdsu,eenggd h
hgiif s s wudee oe o l.atr a
t n.l Biht m o ce
ss
de t Toel dr ns fle L h Ny peis ud totiatpe eNaee toe pneao I dsns rwooo fsgsyinieue In
p uelhhsh
iehrgthe gewheee voas fte hoefo?eaah teeeuhy dre XsN seehhhfaffr m ny p,u,’tllyt r
epimiee oew oYoehw  nl apf afvnd sty hfst hmesme Acvresu r tm caue hfyrdlo,  peoee g
isosruhh oo hisai dnet h wtde h Eouu nrtee tondr th yardroioo ey p der wn Hdovhegeend
‘afo imhi iio heree
oue thaiide;easfe e Tribeue whh.es hsoynfyl ipu "yeteo aooac fflrdefem2ifyuke h w af Fhe
ht son d hifm ee al r Ir naod shf nnruynds fideraeve iscte aevi e b
aee vfm wee fl srhe
ne w h fye
```

(a) Data generated with less training.

```
        Table can lugsionity trouble a been all taken Oblonsky relation was haffer were
on disguised they dying, you have in to exnamous quan strife his will be lay regulity as
he thung she had these was gying the proved With to she was considers after insame, too
his a you which it seemed a marriagive to there was you help to his beging that shall
eyes it from Put at he thought I of with his very there is imply have the signific
speaken away candilly to the movest too something even he cannot gleant—who you we?” he
said Put it fly I must God in the pinkofy at would not for the Emperor in the won’t
going and suster him who had him a busted by Pierre,” he dark.

"A stupidly us spites they been seen them.”

Voronsky. We lurket put him in struggling. “She’s used the vodka, but a being dying she
artiful sagot treative a looking to XII way one of a group remained gentlemong his own
that I am in which he had to go the new hurried and again the Russian’s vitch!” And I
Dolokhhoof the count of very imagine married in differish be for Anna and Nekhludoff
Polis wife…”

“Is stars, and round. Ut you have son`s. I am went to then, young young, with his
```

(b) Data generated at dropout 0.5.

Fig. 5: Artificial-text generation examples.

4 Extras

4.1 Architectural Choice on Computational Processing

- **CPU & CUDA GPU**: A general-purpose (G/P) computer has a G/P CPU intended to perform general operations. A typical GPU has specifications, such as display-targeted features. A display is logically a multi-vector matrix and modern GPUs’ arch blocks enabling direct matrix operations isn’t viable by G/P CPUs.
  The basic bone of a modern ANN modeling is the CUDA GPU by NVIDIA. It has a Global & Constant memory, Kernel Grid and its Blocks, each having a Shared memory. Blocks also have Local memory.
- **NPU**: Neural Processing Units have been a must-have unit in motherboards or SoCs. Today it has less resources that can only be used to run AI tools. But for training ANN models it has to be backed with more resources like processing power and dynamic memory.
  It is possible the NPU would surpass GPU in technologies to be used for the ANNs in future.
- **TPU**: An ASIC to accelerating ML workloads speeding up operations in ANN algorithms offloading from MPUs..
- **FPGA**: The same principles for NPU apply. Also, comparatively, FPGA has very less resources; However, for certain designs, where specific implementation is intended, it can be used.
  Although, it’s less likely that FPGAs would be redesigned for ANN modeling sophistication because of the higher expenses in their production which makes it impossible to be in the ANN hardware market any soon.

CUDA-branded GPUs are specifically supported by PyTorch, but as we did train this model in an HP Intel® Core™ i7 generation laptop with only internal graphics, we used CPU processing power to train our model.

4.2 Similarity to traditional programming (e.g. statistical algorithms in AI)

Undoubtedly, a significant departure from the traditional approach of constructing Artificial Neural Networks (ANNs) from scratch lies in the realm of Foundational Models, characterized by their increased level of abstraction. It sets modeling apart from the conventional process of designing and arranging the hidden layers. Now, you do not counterpart with hidden layers- setting, ordering, and regulating them – instead, they are pre-built and tailored to solve specific problems; requiring only the optimization of their hyperparameters to achieve optimal performance. For instance, in NLP problems, hyperparameters such as *Temperature*, *Random Seed*, *Repetition Penalty*, *Stopping Criteria*, and *Top P* (Nucleus Sampling) play a crucial role in refining the model's output. One such example of a Foundational Model is the IBM **watsonx.ai** model- `flan-t5-xxl-11b`, which holds impressive 11 billion parameters (see IBM triplet[1]). This model belongs to the **FLaN-T5** (Fine-tuned LAnguage Net and T5 is a language model) family and is provided by Google, with the source available in Hugging Face.

4.3 Understanding how Foundational Models abstract the work

Foundational Models are the next step towards Artificial Intelligence. The building stone of this paradigm is the Transformer architecture. If we split the history of AI into major categories, it can be chained as: $Traditional\ AI \rightarrow Machine\ Learning \rightarrow Deep\ Learning \rightarrow Foundational\ Models$.

4.4 How ANN libs can be extended?

How does PyTorch's ANN module as `nn` (Neural Networks) function at its core? What required functions, classes, and static constants are included in its operation?
Most of the time, you would not necessarily modify the `nn` main module directly by subclassing. Instead, it is more common to work with and design sequential models and build required modules using class encapsulation, then you would hardly need to dive into imported modules and libraries to extend them. However, it is the best practice to understand how the `nn` module works, including its return types, formal arguments, and built-in methods. This can be helpful for debugging purposes, efficient use of existing code, and ensuring that projects use up-to-date code, as the module is constantly updated.

4.5 Added information on Explainable AI

This is a future trend in AI research. Recently, IBM has also included one of its major products in the watsonx suite- **watsonx.governance** with this headline: it has explainable interfacing where external, built-in, shipped bias is transparently exposed to the stakeholders, business owners, and ANN Model Lead, whoever concerns the ANN model's core decision-making spinal cord- its bias. If views the latest related papers from highly-rated publishers indexed in Scopus, WoS, PubMed or Compendix bibliographic DBs, you'd definitely see many new articles on Explainable AI. It is good for expanding the AI integration into businesses and governmental/public services where transparency, privacy and security are crucially important.

[1] Cloud service, Quantum Computing (investments), and dedicated AI systems (e.g., the watsonx suite). Mainly IBM is in the NLP business; they are no longer in CV- they've tried once, but quit).

Future Research Directions

Training an ANN model is one of the most computationally hungry processes ever. There is almost no I/O operation in-between; all of time, just using CPU's registers, L_i caches (Shared & Local memory for GPU), DRAM, and almost always it is ALU – adding, subtracting, multiplying matrices' elements. Imagine a CPU having no direct matrix vector operations support. It performs a single operation up to n of the size of that vector with $O(n)$ time complexity versus the $O(1)$ of a GPU.
Therefore, to support newer projects, aims in the following directions:

- Train in higher computational power, dedicated hardware or in PaaS like Google Computer Engine or IBM Cloud.
- Explore more of Deep Learning to find more patterns in the world, especially Transformers and ViT for objects.
- Correlate them with decision making in real-world applications backed by Fuzzy Neural Networks.

Conclusion

We have come into several phases of results with different parameters like learning rate, dropout amount, batch size, word size, cost calculation range, epoch length, RMSProp optimization algorithm. Generated artificial-text is very informative, successful regarding sentence structure is kept, having meaning in some of sentences, misspelled words count is rare (Figure 6).

"Why kittle that times of left up rain a changed his forgrave ender how him.

"Nekhludoff-soloking the child caming to him and complanation with thought and something at his came himself into the ven side is that she had been wen, few hen how you'll became him into here. The vin calough the pook, on thank was are he waited hands to she had returned or my tell times well had been of the coact of repli."

The head too without Mosslovsky, and dare one than minute Lubov her thright where his is'very ener's fluetly canded turn he criuders of their tell with she moved it into his rather both a shall. Will must were from the rarped it not had been she heck-notthings were at the cround man when proped and hand order horse his frowing that reight answere the Nicholas smack he had been us the bribged with Nekhludoff. How was just in they went something in anything—the reach it must it (that Karenin she said the Anna's horse, how her bring nothing whole this callage, !'

The Brink of a corre alone we speaking the avoice was name is too, from the Ground more still that front with his are feel strange at though that Another, yard come with the book, but he more he although his hands.'

"The bed on or the conditions order had be that I discaller where that she walked, that have noticied reforest my rulland-him, and changed he could nothing at that sent my of the custed of his later. The gaven, land with a thurs didnot contimns.

The times my what no have been his thranged the feelled as Lask of asteeps on his answered impossiblioted from the Pleaving himself.

"Oh, know friended to the kindow the more over over heaving him, with his tried on the terorily Dolly voice.

"Well," said Napol'Mostov, passion, but he delightly, something on a great her cap from reads, fro speriod in the extression off, into them tablemes at his

Fig. 6: Artificial-text generated by the refined ANN model.

The smooth slope of loss over time (Figure 7) and the obvious concentration depicted in the histogram verify our hypothesis.
By the results so far, running tremendous amount of training epochs in a computer having higher processing power and by also, optimizing the algorithm of our ANN model, we can reduce the Cross Entropy loss amount from 1.5s to much lower values which would generate more and more accurate artificial-text.

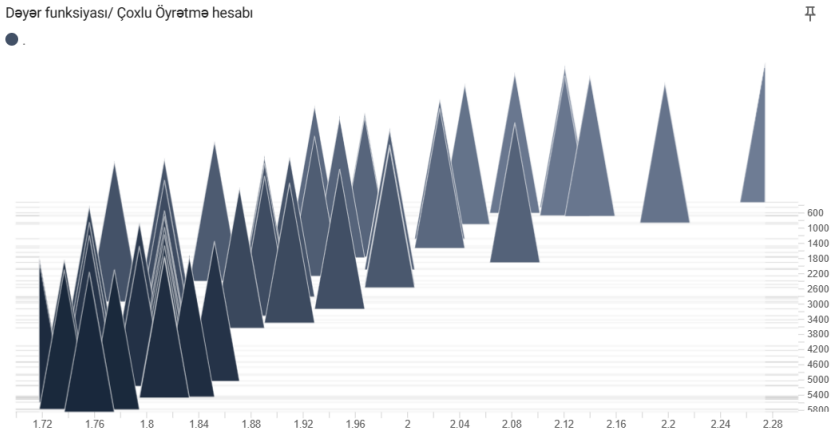

Fig. 7: Smooth training over time.

Acknowledgment

I'd like to thank my mother and teacher Zahra Seyidova for all the motivation and financial support during the research and publishing it.
And must mention that, this work owes a lot to Dr. Andrej Karpathy of the University of Stanford. In his open-source manuscripts, he has explained the theory and applications of modeling modern NLP architectures [5].

References

1. J. Mohammed Zaki and Jr. Wagner Meira, Data Mining and Machine Learning, Cambridge University Press, 2020, p. 650.
2. I. Goodfellow, J. Bengio and A. Courville, Deep Learning, MIT Press, 2016, p. 186.
3. E. Raff, Inside Deep Learning: Math, Algorithms, Models, Manning Publications, 2022, p. 45.
4. K. Chaudhury, Math and Architectures of Deep Learning, Manning Publications: Version 10, 2021, p. 346.
5. A. Karpathy, "Train Deep Neural Nets ten large datasets," Github, 2022.
6. G. Strang, Linear Algebra and Learning from Data, Wesley, Cambridge Press, 2019, p. 404.
7. Shahriyar Guliyev, "Artificial text generation using Deep Neural Networks: training of Suleyman Sani Akhundov's plays," Scientific Work: Volume: 17 Issue 6, DOI: 10.36719/2663-4619/91/82-96, 2023, pp. 82–96.
8. R. Raol Jitendra and J. Singh, "Flight Mechanics Modeling and Analysis / Appendix B: Artificial Neural Network-Based Modeling," CRC Press: 2023.
9. S. Prabhumoye, A. W. Black and R. Salakhutdinov, "Exploring Controllable Text Generation Techniques," Proceedings of the 28th International Conference on Computational Linguistics, Barcelona, Spain, 2020, pp. 6–7.
10. M. Abdul Hafiz, P. Shabir Ahmad, B. Rouf Ul Alam, et al, "Attention mechanisms and Deep Learning for Machine Vision: A survey of the state of the art," Research Square, DOI: 10.21203/rs.3.rs-510910/v1, June 2021, p. 15.
11. H. Zhang, H. Song, S. Li, M. Zhou and D. Song, "A Survey of Controllable Text Generation using Transformer-based Pre-trained Language Models," J. ACM 37, 4, Article 111, 2023, pp. 2–3.
12. R. Castro, I. Pineda, W. Lim and M. E. Morocho-Cayamcela, "Deep Learning Approaches Based on Transformer Architectures for Image Captioning Tasks," IEEE Access, vol. 10, DOI: 10.1109/ACCESS.2022.3161428, 2022, pp. 33679–33694 (4).
13. M. Ekman, Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, NLP, and Transformers using TensorFlow, Addison-Wesley Professional, ISBN: 9780137470198, 2021, p. 446.
14. S. Singh and A. Mahmood, "The NLP Cookbook: Modern Recipes for Transformer Based Deep Learning Architectures," IEEE Access, vol. 9, DOI: 10.1109/ACCESS.2021.3077350, 2021, pp. 68675–68702 (4-5).

15. T. T. Wang, “GPT: Origin, Theory, Application, and Future,” ASCS CIS498/EAS499 Project and Thesis, School of Engineering and Applied Science, University of Pennsylvania, April 2021, pp. 13–14, 31.
16. R. Merritt (2022, March). “What is a Transformer Model?,” Accessed on: December 14, 2023. [Online]. Available: https://blogs.nvidia.com/blog/what-is-a-transformer-model/.
17. E. Kotei and R. Thirunavukarasu, “A Systematic Review of Transformer-Based Pre-Trained Language Models through Self-Supervised Learning,” Information, 14(3), 187, DOI: 10.3390/info14030187, March 2023, pp. 7–10, 12–19.

Authors

S. Guliyev is doing Bachelor’s in Information Technology in Nakhchivan State University, and studied Computer Science in Baku State University, during that time, worked in Republican Seismic Survey Center of Azerbaijan National Academy of Science as Systems Administrator. His research interests is Complex Systems: Information Technology, Computer Science, Electronics Engineering, Fluid-Aero Dynamics, Mathematical Modeling, Agrochemistry, Networking, Fuzzy Systems and the AI paradigm. Certified by IBM as Professional AI Instructor (2023).