Influence of Digital Fluctuations on Behavior of Neural Networks

Igor V. Netay

***Abstract**: This paper deals with effect of digital noise to numerical stability of neural networks. Digital noise arises from the inexactness of floating point values operations. Accumulated errors finally lead to the loss of significance. Experiments show that more redundant networks have higher noise influence. This effect is tested in both model and real world samples. As a result, one should exclude all the networks results from the beginning of fluctuations. Results of experiments allow us to hypothesize that minimal values of loss function preserving significance were achieved for the networks of size close to the complexity of the dataset. So, it is a reason to choose sizes of network layers in accordance with complexity of particular datasets and not universally for an architecture and general problem statement without relation to data. In the case of fine tuning this suggests that pruning of network layers can improve result accuracy and reliability of prediction due to decrease of numerical noise influence. Results of this article are based on analysis of numerical experiments with train of more than 50000 neural networks for thousands epochs for each network. Almost all the networks begin to fluctuate.*

Keywords: Neural Network, Numerical Stability, Digital Noise, Digital Fluctuations, Fine-Tuning.

I. INTRODUCTION

In modern industry, artificial neural networks are widely applied. This is explained by their great performance in a wide area of applications. Moreover, a highly developed theoretical base shows that the mathematical model of neural network can express and approximate wide class of functions. Theoretical results begin with the famous result of Arnold [1] disproving 13th Hilbert problem. It assumes that there exists a continuous function of some number of variable that is indecomposable into the composition of functions in smaller number of variables. It turned out that any continuous function on n-dimensional compact set can be constructively expressed by superposition of $n(2n+1)$ functions of one variable and addition function $a(\mathrm{x},\mathrm{y}) = x + y$(see [10]). Actually, the form of this representation looks like a general form of function represented by a neural network with one hidden layer and different activations.

For general activation function, there are some well known results on approximations: Cybenko theorem on approximation of functions by wide one-layer neural network [3, 4, 6] and the dual theorem on approximation by deep narrow networks [9].

Reduced accuracy of neural networks with long training is related to big accumulated errors such as rounding errors. Vanishing or exploding gradients are the simplest well known consequence of such error accumulation.

One of ways to solve this problem is data normalization.

More recent theoretical results are more focused on negative results of approximations. In [20] the lower bounds of number of epochs with a given tolerance were found. In [11] necessary and sufficient numbers of neurons to approximate a function with given tolerance were found. Also, attacks for neural networks become very popular research topic.

One can see that networks with bigger and bigger number of training parameters appear. If one have an ML problem, the usual decision is to utilize some network pretrained on some huge dataset and fine tune it on own dataset. The growth can be explained by expression of problems for neural networks to solve in terms of natural language and therefore having no finite exact form by its sense. Sizes of some well known networks are listed in Table I.

Table-I: Some examples of well known important neural networks

Network	year	Parameters
LeNet-5 [13, 14, 15, 25]	1998	48120
Google Net [21]	2014	≈7000000
Alex Net [12]	2014	≈61000000
Inception v3 [22]	2015	≈24000000
Resnet-152 [7]	2015	≈60000000
VGG-19 [18]	2015	≈144000000
GPT-3 [2]	2020	≈185000000

Usually, neural networks are trained with 32-bit floating point values. In the same time, 16-bit floating point values are used for optimization. There are also 8-bit tensor processors for neural networks for edge computations. Each step of quantization leading to decrease of precision obviously decreases the result significance. However, one usually checks the quality of network prediction in these cases by checking the target quality metric without any check of significance. Loss of significance is explained by the fact that float numbers are representations of some subset of real numbers, but not real numbers themselves. The standard for this representation is known as IEEE 754. Also, this standard describes how operations with numbers are performed. All the operations have some inaccuracies, but the biggest one is in the case of the subtraction of numbers close to each other.

Manuscript received on 18 November 2022 | Revised Manuscript received on 25 November 2022 | Manuscript Accepted on 15 December 2022 | Manuscript published on 30 December 2022.

***** Correspondence Author (s)
Igor V. Netay*, JSRPC Kryptonite and Intitute for Information Transmission Problems of Russian Academy of Sciences, Moscow, Russia. Email: i.netay@kryptonite.ru

Comparison of values can also introduce errors, because it is implemented through the subtraction. Classification of operations with numbers used in optimizers in torch library (one of most popular libraries for neural networks) is listed in Table II.

Table-II: Numeric operations in torch library.

Operation	Optimizer			
	NAdam	others[1]	ASGD, Adamax	SGD, Rprop
$\pm, *, >$	+	+	+	+
/	+	+	+	-
$\sqrt{\bullet}$	+	+	-	-
pow	+	-	-	-

The goal of this work is to describe some macroscopic objects arising from accumulated numeric inaccuracies and pointing that the accumulated error is big enough. We call them digital fluctuations. Numerical experiments demonstrate relation between neural network sizes and accuracy of function approximations by these networks preserving significance. We look at the minimal loss value achieved by networks before significance gets lost. For synthetic datasets, we can define the complexity of labeled dataset as the size of network applied to label it. For general case some approaches may be found in [8]. It turns out that the minimal loss value is achieved by networks of size near to dataset complexity.

II. RESULTS

In this paper, influence of digital noise to neural networks behavior is studied. Numerical experiments show that numerical inaccuracies can lead to total loss of significance of network inference and make any results and quality metrics values meaningless. It is easy to detect digital fluctuations in network train process. Any results based on network predictions after fluctuations begin are just artifacts of digital noise and should not be interpreted and used to make any conclusions. This makes detection of fluctuation necessary to preserve prediction significance. Experiments with model datasets show that for given complexity of dataset there is an optimal choice of network size such that the network achieves the best accuracy without loss of significance. Results of experiments allows us to propose conjecture that optimal network size is close to dataset complexity. Experiments with real world datasets and well known neural networks show that fluctuations arise not only in model networks. As a result, we conclude that network size should depend on train dataset. If the network is redundant relatively to dataset complexity, then its training process becomes less numerically stable and leads to earlier and higher fluctuations. Another result is that start of digital fluctuations of loss function during training process marks the moment after which reliability of trained network become doubtful.

III. RELATED WORK

Evaluation of functions of inexact arguments leads to inexact values. The dependence of inexactness of the result on inexactness of arguments is expressed by the *condition number*.

The weights learned by networks give maps between layers, which usually have high condition numbers (see [17]). Such maps are said to be *ill conditioned*. Respectively, one can assume that neural networks accumulate high numerical errors when training. Loss of exactness can be exploited as a vulnerability for attacks. Paper [19] hypothesizes that ill conditioned weight matrices in neural network are a factor towards adversarial success. In [16] some regularization improving conditioning is considered as a defense method. It is well known that wide enough network with one hidden layer can approximate any continuous function [5, 3]. Deep networks (having more than one hidden layer) are more vulnerable to attacks [23]. Therefore, it is simpler to use one layer networks to compare their accuracy.

IV. THEORY

A regression machine learning problem with numeric features includes the following data:

•$X \simeq \mathbb{R}^n$—data space,

•$Y \simeq \mathbb{R}^m$—label space,

•$A = A(\theta) = \{A(x) = g(x,\theta) | \theta \in \Theta\}$is a map $X \to Y$ parameterized by θ (*prediction model*), where$g: X \times \Theta \to Y$ is a fixed function,

•Θ is a set of admissible parameter θ values.

Definition 3.1. A *training method* is a set of functions

$$\mu_\ell : (X \times Y)^\ell \to \Theta$$

for$\ell \in \mathbb{N}$ which puts a parameter $\theta = \mu(D^\ell)$ into correspondence with a data sample $D^\ell = \{(x_i, y_i)\}_{i=1}^{\ell}$ for model $A(\theta)$.

Definition 3.2. A *loss function* $L(A,(x,y))$ for a model A on data-label pair (x,y) is a function $L: X \to Y \times D^\ell \to \mathbb{R}$. Given a labeled dataset $D^\ell = \{(x_i, y_i)\}_{i=1}^{\ell}$, we call the *empirical risk* or *a loss function* on the dataset D^ℓ the value

$$Q(A, D^\ell) := \frac{1}{\ell} \sum_{i=1}^{\ell} q\,(A(x_i), y_i)$$

Therefore, the problem of machine learning is reduced to minimization of empirical risk.

Usually, the loss function is defined by an expression of form

$$L(A,(x,y)) = q(A(x), y)$$
$$Q(A,\{(x_i, y_i)\}_{i=1}^{\ell}) = \tfrac{1}{\ell} \textstyle\sum_{i=1}^{\ell} q\,(A(x_i), y_i),$$

where$q$ is some similarity measure on labels, for instance, a distance function on the space Y.

Loss function MSE (mean square error) is defined by $q(y, \hat{y}) = ||y - \hat{y}||^2$.

Cross-entropy function is defined by a non-symmetric function $q(y, \hat{y}) = -\sum_{j=1}^{m} y_j \cdot \log_2 \hat{y}_j$, where $y = (y_1, \dots, y_m)$.

Usually, the set of parameters Θ is a real vector space $\mathbb{R}^p$, where p is the number of free parameters (also called the number of trainable parameters) of the model.

[1]All the others optimisers in torch, namely, Adadelta, Adagrad, Adam, AdamW, SparseAdam, LBFGS, RAdam, RMSprop.

The map g is chosen to be differentiable in model parameters (maybe exact zero Lebesgue measure set).
For such models the gradient descent algorithm is applicable. Given an initial parameter θ_0, we can define the next approximations as

$$\theta_{e+1} = \theta_e - \gamma \nabla L(\theta_e), \tag{1}$$

where the parameter $\gamma > 0$ is called *learning rate*.

Neural networks are one of the most popular machine learning algorithms. There are many types and constructions of neural networks exist, but there is no general final definition of neural network. In this work, we restrict a notion of neural network N by the following collection of data:

1. $G = (V, E)$is a directed graph (oriented graph without oriented cycles), elements of V are also called *neurons*,
2. $I \subseteq V$is the set of vertices without incoming edges,
3. $O \subseteq V$contains all vertices without outcoming edges,
4. $(\{w_{u,v}\}, \{b_v\})$are weights and biases (offsets) $u, v \in V$, $v \notin I$, $(u, v) \in E$ (we write the symbol • for brevity and to avoid introduction of all indexes running over corresponding definition domains. In this case we will denote the collection of weights by $(w_{\bullet,\bullet}, b_\bullet)$),
5. $a_v: \mathbb{R} \to \mathbb{R}$is an *activation function* for each neuron $v \notin I$.

We consider the generalization of the definition above by identification of parameters $b_\bullet$ for some subsets of vertices. These subsets are called *layers* of neural network.
Let us introduce some additional notation for further convenience.

[1] Given a network N, we call the subcollection of data $A(N) \coloneqq (G, O, \{a_v\}_{v\in V})$ the network's *architecture*,
[2] The collection of data $W \coloneqq (\{w_{\bullet,\bullet}\}, \{b_\bullet\}) \in \mathbb{R}^E \bigoplus \mathbb{R}^V = \Theta$ is said to be the *set of weights* (or *trainable parameters*) of N,
[3] The network N is said to be a neural network of architecture A, *initialized* by weights W.

Transfer function of neuron $v \in V$ is a function

$$\rho(v) \coloneqq a_v\left(\sum_{\substack{u\in\mathcal{V}\\(u,v)\in\mathcal{E}}} w_{u,v}\rho(u) + b_v\right),$$

where the sum is taken over all neurons u with edges from them to v.
Forward propagation fp_N of signal in the network N is the injection $\mathbb{R}^I \to \mathbb{R}^V$ defined inductively by transfer functions of neurons $v \in V$.
Transfer function N_* *of the network*N is the restriction of forward propagation on the subset of output neurons, i. e. $N_* = (fp)|_O$ (see Fig. 1).
Given some numeration of input and output neurons, we can identify $X \simeq \mathbb{R}^I$ and $Y \simeq \mathbb{R}^O$:

$$\begin{tikzcd}
X \arrow[rr, "\mathcal{N}_*"] \arrow[d, equal, "\simeq"] & & Y \arrow[d, equal, "\simeq"] \\
\mathbb{R}^{\mathcal{I}} \arrow[r, hook, "\mathrm{fp}_{\mathcal{N}}"] & \mathbb{R}^{\mathcal{V}} \arrow[r, two heads, "\bullet|_{\mathcal{O}}"] & \mathbb{R}^{\mathcal{O}}
\end{tikzcd}$$

Scheme for definition of neural network transfer function.

Usually, some variants of gradient descent method are used to train networks. Taking initial weights θ_0 in some random way, repeatedly applied gradient descent unfold initial state to the sequence of weights θ_i, $i \in Z_{\geqslant 0}$. So, the training method of a neural network consists of some variant of gradient descent method and a stop-rule to break the sequence $\theta_\bullet$ and take some its element. The simplest variant of gradient descent method is (1), but much more sophisticated variants are used in most cases.
Explicit form of (1) was independently found by Galushkin and Werbos in 1974 and was called the *backpropagation*. The transfer function N_* is expressed as a superposition of neurons' transfer functions as a function in inputs and weights as coordinates in W. Therefore, partial derivatives $\frac{\partial L}{\partial w_{\bullet,\bullet}}$ and $\frac{\partial L}{\partial b_\bullet}$ at $D^1 = (x, y)$ can be expressed by a chain rule through L', $a_\bullet'$ and coordinates of x and y. Functions L and $a_\bullet$ are usually chosen to be differentiable in explicit form without numerical differentiation.
Backpropagation takes the labeled dataset D^ℓ to the vector field $\nabla Q(A, D^\ell)$ over Θ. So, training of neural network is actually a dynamical system moving in Θ along the vector field in opposite direction.
Scheme of experiment for $(n, m) = (3,3)$.

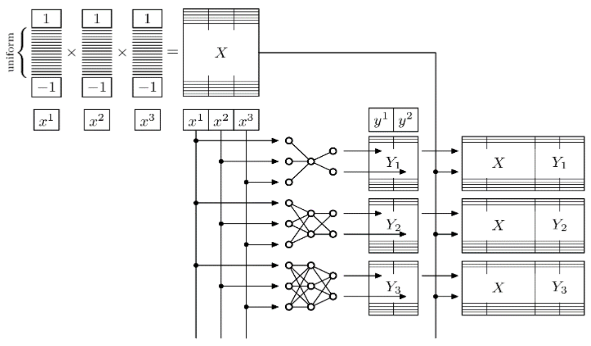

Figure 1: Scheme of labeled datasets construction for (n, m) = (3, 2)

Usually, some variants of gradient descent method are used to train networks. Taking initial weights θ_0 in some random way, repeatedly applied gradient descent unfold initial state to the sequence of weights θ_i, $i \in \mathbb{Z}_{\geqslant 0}$. So, the training method of a neural network consists of some variant of gradient descent method and a stop-rule to break the sequence $\theta_\bullet$ and take some its element. The simplest variant of gradient descent method is (1), but much more sophisticated variants are used in most cases. Explicit form of (1) was independently found by Galushkin [8] and Werbos [24] in 1974 and was called the *backpropagation*. The transfer function N is expressed as a superposition of neurons' transfer functions as a function in inputs and weights as coordinates in W. Therefore, partial derivatives $\frac{\partial L}{\partial w_{\bullet,\bullet}}$ and $\frac{\partial L}{\partial b_\bullet}$ at $D^1 = (x, y)$ can be expressed by a chain rule through L', $a_\bullet'$ and coordinates of x and y. Functions L and $a_\bullet$ are usually chosen to be differentiable in explicit form without numerical differentiation.

Backpropagation takes the labeled dataset D^ℓ to the vector field $\nabla Q(A, D^\ell)$ over Θ. So, training of neural network is actually a dynamical system moving in Θ along the vector field in opposite direction.

V. EXPERIMENTSETTING

Definition 6. A *fully connected architecture* with sizes of hidden layers $h_1, \dots, h_\ell$ and numbers of inputs and outputs n and m is the graph, where vertices are union of subsets $N, H_1, \dots, H_\ell, M$ of $n, h_1, \dots, h_\ell, m$ elements correspondingly and edges are from each vertex of N to each of H_1, from each of H_j to each of H_{j+1} for $j = 1, \dots, l-1$, and from each of H_ℓ to each of M, and there are no any other edges; the set of output vertices O coincides with M. Let us denote this architecture by $A(n, h_1, \dots, h_\ell, m)$. In the case of unique hidden layer we will use the letter h without any index.

A fully connected neural network is a network of fully connected architecture.

We will draw fully connected neural networks as graphs where layers are placed vertically and the signal propagation is directed to the right.

We will denote a network of architecture $A(n, h, m)$ initialized with weights of some random distribution by $R(n, h, m)$.

For research of digital noise, we perform series of experiments. In these experiments a series of neural networks of increasing sizes are trained on the same dataset.

Also, we perform series of experiments on real datasets.

Model datasets are points in a lattice inside the cube $[-1, 1]^n \subset X$. Namely, in coordinate segment $[-1,1]$ is subdivided by this lattice into 2000 equal parts, so dataset consists of 2001^n points. Hereafter we label this dataset by pointwise application of some function f, and obtain a dataset of form $D^\ell = \{x_i, y_i = f(x_i)\}_{i=1}^{\ell}$. We choose a function f equal to $R(n, h, m)$.

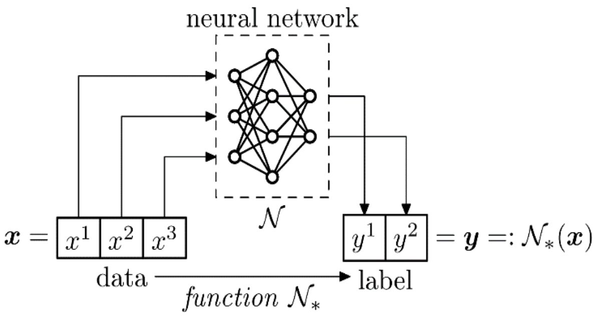
neural network
$\mathcal{N}$
$\boldsymbol{x} =$ x^1 x^2 x^3
y^1 y^2 $= \boldsymbol{y} =: \mathcal{N}_*(\boldsymbol{x})$
data
function $\mathcal{N}_*$
label

Figure 3: Scheme for definition of neural network transfer function

The function f is the transfer function of neural network $R(n, h, m)$. An equivalent way to express such function is to take an expression of form

$$f_\bullet(x) = \sum_n \bullet \cdot a\left(\sum_{j=1}^{h} \bullet \cdot x_j + \bullet\right) + \bullet,$$

where $\bullet$ denotes each time a new real random coefficient. In particular, for activation ReLU and $n = 1$, in general case we obtain a piecewise linear function with h discontinuities of derivative. Scheme of labeled dataset construction is drawn on Fig. 2.

So, each dataset consists of 2001^d entries and is a table of form $2001^n \times (n + m)$.

We apply this procedure for the collection of neural networks $R(n, h, m)$, $h \in \{1, 2, \dots, h^\circ\}$ for some h° (here $h^\circ = 100$). Note that by the construction for any collection of networks with the same n the blocks of the first n columns (the block X on Fig. 2) coincide.

For each labeled dataset D, we perform the following experiment:

[1] construct a collection of neural networks $N(n, h, m)$ for $h' = 1, \dots, (h')^\circ$ (we put $(h')^\circ = 100$),

[2] the networks are initialized by default way,

[3] we choose some common parameters for the network optimizer (we choose the Adam optimizer with learning rate 0.001),

[4] all the networks are trained on this dataset during the same number of epochs (for $(n, m) = (1,1), (1,2), (1,3)$ it equals 10000, for $(2,1)$ — 2000, for $(2,2)$ — 1000),

[5] as a result, we obtain the collections of weights $W^{\bullet}_{\bullet,\bullet,\bullet}$ and loss function values $\lambda^{\bullet}_{\bullet,\bullet,\bullet}$,

[6] we analyze the sequences $\{\lambda^e_{n,h,m}\}_{e=1}^{E_{n,h,m}}$.

The scheme of experiment is illustrated on Fig. .

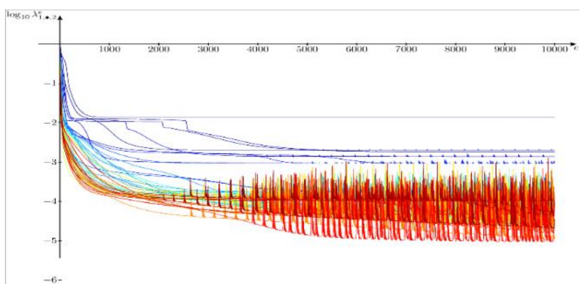

Figure 4: Fluctuations for dimensions (n, m) = (1, 2) and dataset complexity h=10

VI. EXPERIMENT RESULTS

A. Modelexperiments

One can see examples of loss functions for networks of different sizes trained on the same labeled dataset on Fig. . Each illustration contains graphics of these loss functions drawn with different color. It is easy to see that the lowest values decrease as h increases, but the beginning of the first fluctuation moves to the left, and the amplitude grows.

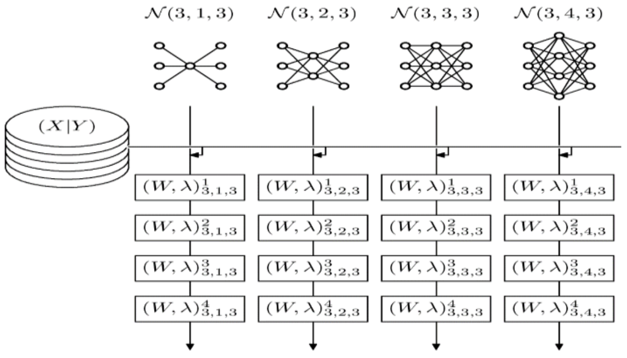

Figure 5: Scheme of experiment for (n, m) = (3, 3)

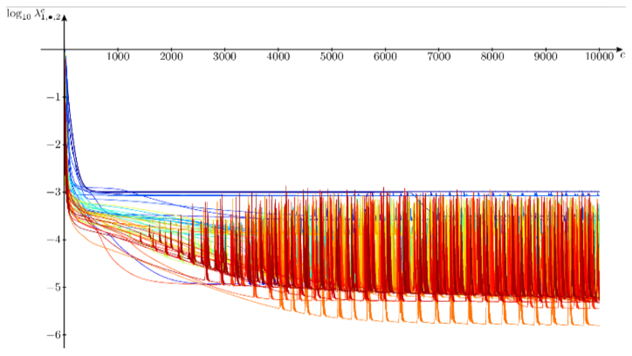

Figure 6: Fluctuations for networks with input and output dimensions (n, m) = (1, 1) trained on dataset of complexity h=10.

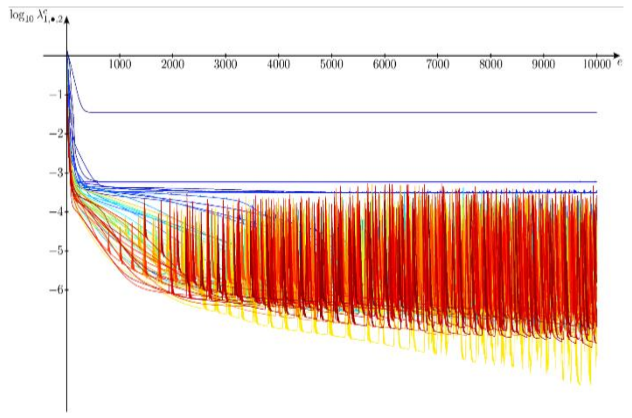

Figure 7: Fluctuations for dimensions (n, m) = (1, 3) and dataset complexity h=10

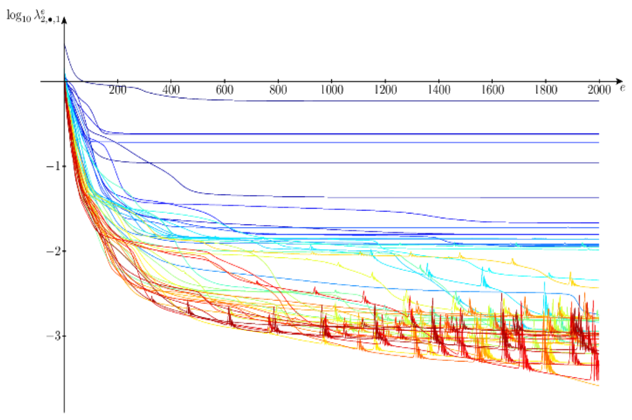

Figure 8: Fluctuations for dimensions (n, m) = (2, 1) and dataset complexity h=15

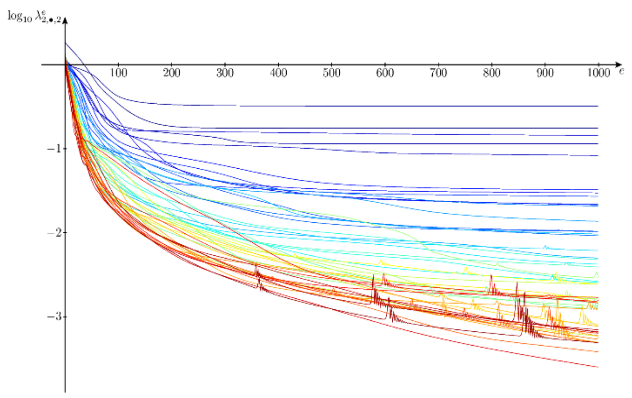

Figure 9: Fluctuations for dimensions (n, m) = (2, 2) and dataset complexity h=15.

To improve visibility of typical behavior of best achieved loss function value, we smoothen the graph (see Fig.).

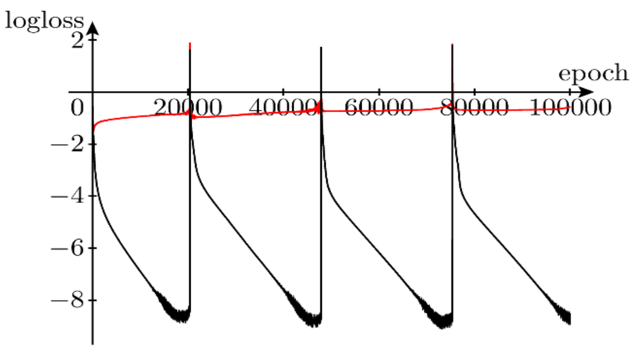

Figure 10: Loss function logarithm for train of LeNet-5 on MNIST with Adam optimizer, lr=0.001, black line for train loss and red line for test loss.

On Fig. one can see the initial part of graph for set of such smoothened curves for $(n, m) = (1,2)$.

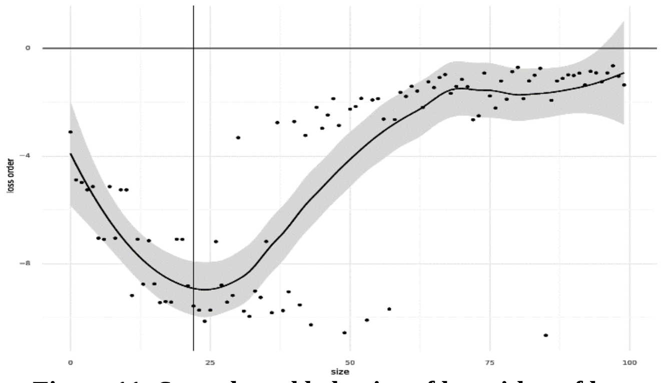

Figure 11: Smoothened behavior of logarithm of loss function depending on h arranged from 1 to 100 with parameters (n, m) = (1, 1), h=22; the vertical line corresponds to h=22.

So, the view of smoothened distribution (to decrease the noise) of logarithms of minimal loss function as a function of the network size suggests that there should be on optimal choice of network size for training on this dataset.
Training of the model neural networks was performed for a month on one GPU NVidia Quadro P5000.

B. Experimentsonrealdatasets

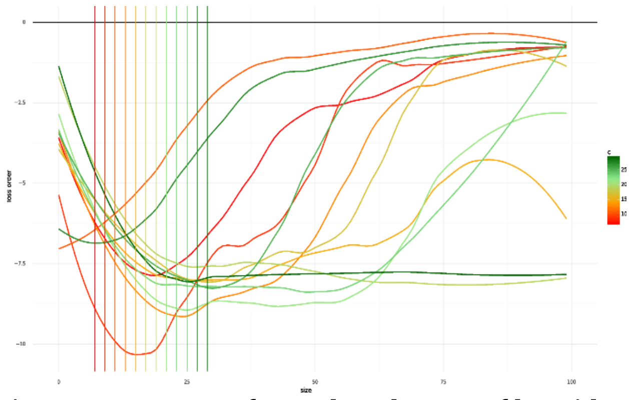

Figure 12: Sequence of smoothened curves of logarithms of lowest loss function values before fluctuations in dependence of network size. Dataset complexity corresponds to vertical line of the same color. Here (n, m) = (1, 2)

Here we check that the digital fluctuations appear with network Lenet5 on the dataset MNIST. For this network we have tested about ten standard optimizers, and the fluctuations have appeared in all the cases. For the optimizer Adam and SGD (seeming to be most popular) one can see the fluctuations on Fig. , Fig.

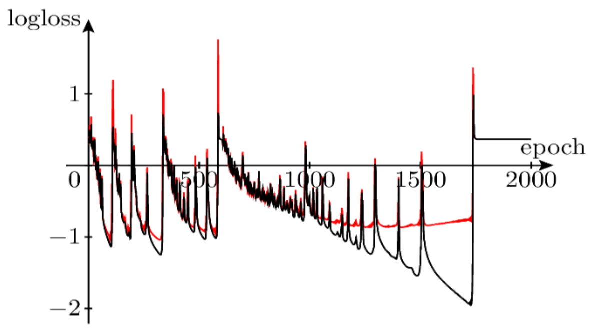

Figure 13: Loss function logarithm for train with SGD optimizer, lr=0.9.

Fluctuations for real world examples show that they are not an effect of experiments model.

VII. APPLICATIONTOFINETUNING

Suppose we have a neural network trained to solve problem on some big dataset. We want to fine tune it on our smaller dataset. Then we can fine tune all the network or some of its layers (usually, a subset of some last layers) on our dataset. Then it would be a good idea to shrink layers to the sizes corresponding to complexity of our dataset. This means that replacement of layers to smaller ones may improve numeric stability and convergence, and finally to improve the resulting performance. For a few layers they can be tuned separately. If we apply the initial network to simpler dataset, than the dataset it was trained on, then the fine-tuning may converge better on smaller layer sizes, because the network may converge worse due to presence of layers redundant with respect to our dataset. So, redundancy of a network or its layer size is not absolute, but depends on the dataset.

VIII. CONCLUSION

We have researched the fluctuations arising at network training. It turns out that they appear almost everywhere. Digital fluctuations lead to growth of loss function by a few order of value and to loss of significance. It implies that training should be stopped, when fluctuations begin. So, observation of fluctuations can help to stop the training, when it loses sense. Moreover, we should observe them to avoid subsequent interpretation of meaningless inference values. We have compared minimal achieved loss function values for series of neural networks of different sizes for the same dataset of known complexity and repeated it for different complexities. It turns out that in these sequences the minimal loss function values decrease at the beginning and then begin to oscillate and finally increase. This implies that we cannot infinitely increase network accuracy with choosing bigger networks and/or longer training process. Conversely, there is some best choice for network size. In many cases the best accuracy is achieved by networks of sizes close to the dataset complexity.

This suggests that there are some estimation of size of network achieving the best quality, and this estimation is close to the dataset complexity. For large complexities, we can see in experiment that the estimation of best approximating network size does not increase linearly as dataset complexity increases. This can be explained by the fact that very close placement of the learning function breakpoints sometimes can approximate the function by network of lower size than function complexity. We can see that for long enough training the fluctuations begin almost everywhere. So, in case of numerical instability of fine tuning on smaller datasets, it is reasonable to decrease layer sizes to achieve better performance and preserve significant digits in result. Although a bigger network can express any function that can be expressed by smaller networks, experiments show that this does not happen in practice, and redundant networks usually confuses itself by accumulating digital noise. This digital noise accumulates and in some time gets bigger than the meaningful part of result and makes network inference useless.

ACKNOWLEDGMENTS

The author is grateful to his Kryptonite colleagues Vasilii Dolmatov, Nikita Gabdullin and Anton Raskovalov for fruitful discussions of topic and results.

REFERENCES

1. Arnold, Vladimir I. 2009. “On Functions of Three Variables.” *Collected Works: Representations of Functions, Celestial Mechanics and KAM Theory, 1957–1965*, 5–8.
2. Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, PrafullaDhariwal, Arvind Neelakantan, et al. 2020. “Language Models Are Few-Shot Learners.” In *Advances in Neural Information Processing Systems*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, 33:1877–1901. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.
3. Chen, Tianping, Hong Chen, and Ruey-wen Liu. 1992. “A Constructive Proof and an Extension of Cybenko’s Approximation Theorem.” In *Computing Science and Statistics*, edited by Connie Page and Raoul LePage, 163–68. New York, NY: Springer New York. [CrossRef]
4. Cybenko, George. 1989. “Approximation by Superpositions of a Sigmoidal Function.” *Mathematics of Control, Signals and Systems* 2 (4): 303–14. [CrossRef]
5. Cybenko, George V. 1989. “Approximation by Superpositions of a Sigmoidal Function.” *Mathematics of Control, Signals and Systems* 2: 303–14. [CrossRef]
6. Funahashi, Ken-Ichi. 1989. “On the Approximate Realization of Continuous Mappings by Neural Networks.” *Neural Networks* 2 (3): 183–92. [CrossRef]
7. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. “Deep Residual Learning for Image Recognition.” *CoRR* abs/1512.03385. http://arxiv.org/abs/1512.03385.
8. I., Galushkin A. 1974. *SintezMnogosloinyh System RaspoznavaniyaObrazov (in Russian)*. M.: Energiya.
9. Kidger, Patrick, and Terry Lyons. 2020. “Universal Approximation with Deep Narrow Networks.” In *Conference on Learning Theory*, 2306–27. PMLR.
10. Kolmogorov, A. N. 1957. “On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition.” *Dokl. Akad. Nauk SSSR* 114: 953–56.
11. Kon, Mark A, and LeszekPlaskota. 2000. “Information Complexity of Neural Networks.” *Neural Networks* 13 (3): 365–75. [CrossRef]

12. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." In *Advances in Neural Information Processing Systems*, edited by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
13. Lecun, Yann. 1989. "Generalization and Network Design Strategies." In *Connectionism in Perspective*, edited by R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels. Elsevier.
14. Lecun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1990. "Handwritten Digit Recognition with a Back-Propagation Network." In *Advances in Neural Information Processing Systems 2*, edited by D. S. Touretzky, 396–404. Morgan Kaufmann.
15. Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86 (11): 2278–2324. https://doi.org/10.1109/5.726791. [CrossRef]
16. Nguyen, Andre T., and Edward Raff. 2018. "Adversarial Attacks, Regression, and Numerical Stability Regularization." *arXiv e-Prints*, December, arXiv:1812.02885. https://arxiv.org/abs/1812.02885.
17. Saarinen, S., R. Bramley, and G. Cybenko. 1993. "Ill-Conditioning in Neural Network Training Problems." *SIAM Journal on Scientific Computing* 14 (3): 693–714. https://doi.org/10.1137/0914044. [CrossRef]
18. Simonyan, Karen, and Andrew Zisserman. 2014. "Very Deep Convolutional Networks for Large-Scale Image Recognition." *CoRR* abs/1409.1556. http://arxiv.org/abs/1409.1556.
19. Sinha, Abhishek, Mayank Singh, and Balaji Krishnamurthy. 2019. "Neural Networks in an Adversarial Setting and Ill-Conditioned Weight Space." In *ECML PKDD 2018 Workshops*, edited by Carlos Alzate, Anna Monreale, HaythamAssem, Albert Bifet, Teodora Sandra Buda, Bora Caglayan, Brett Drury, et al., 177–90. Cham: Springer International Publishing. [CrossRef]
20. Song, Le, Santosh Vempala, John Wilmes, and Bo Xie. 2017. "On the Complexity of Learning Neural Networks." *Advances in Neural Information Processing Systems* 30.
21. Szegedy, Christian, Wei Liu, YangqingJia, Pierre Sermanet, Scott E. Reed, DragomirAnguelov, DumitruErhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. "Going Deeper with Convolutions." *CoRR* abs/1409.4842. http://arxiv.org/abs/1409.4842.
22. Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and ZbigniewWojna. 2015. "Rethinking the Inception Architecture for Computer Vision." *CoRR* abs/1512.00567. http://arxiv.org/abs/1512.00567.
23. Szegedy, Christian, WojciechZaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. "Intriguing properties of neural networks." *arXiv e-Prints*, December, arXiv:1312.6199. https://arxiv.org/abs/1312.6199.
24. Werbos, Paul. 1974. "Beyond Regression:" New Tools for Prediction and Analysis in the Behavioral Sciences." *Ph. D. Dissertation, Harvard University*.
25. Y., LeCun, Boser B., Denker J. S., D. Henderson, Howard R. E., Hubbard W., and Jackel L. D. 1989. "Backpropagation Applied to Handwritten Zip Code Recognition." *Neural Computation* 1: 541–51. https://doi.org/10.1162/neco.1989.1.4.541. [CrossRef]

AUTHORS PROFILE

Igor V. Netay, MS in Mathematics in MSU and IUM, Moscow, Russia; PhD in IITP, Moscow, Russia. He is a researcher at JSRPC Kryptonite and at IITP. The principal areas of research in which the author has worked are: Neural Networks, Algebraic Geometry, Algebraic Topology, Representation Theory, Dynamical systems. (1) I. V. Netai, "Parabolically connected subgroups", Mat. Sb., 202:8 (2011), 81–94; Sb. Math., 202:8 (2011), 1169–1182 (2) I. V. Netay, "Syzygy Algebras for Segre Embeddings", Funkts. Anal. Prilozh., 47:3 (2013), 54–74; Funct. Anal. Appl., 47:3 (2013), 210–226 (3) V. V. Buchstaber, I. V. Netay, "Hirzebruch functional equation and elliptic functions of level d", Funct. Anal. Appl., 49:4 (2015) (4) I. V. Netay, "Syzygies of quadratic Veronese embedding", Sb. Math., 208:2 (2017) (5) Igor V. Netay, Alexei V. Savvateev, "Sharygin Triangles and Elliptic Curves", Bull. Korean Math. Soc., 54:5 (2017), 1597-1617. (6) I. V. Netay, "Hirzebruch functional equations and complex Krichever genera", arXiv: https://arxiv.org/abs/1610.04654. (7) Glutsyuk, A.A., Netay, I.V., \On Spectral Curves and Complexified Boundaries of the Phase-Lock Areas in a Model of Josephson Junction", J. Dyn. Control Syst., 2020. (8) I. Netay, "Cyclic space-filling curves and their clustering property", Danish Scientific Journal, 45:2 (2021), 30-39.