Advances in Intelligent Systems Research, volume 174
Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS 2020)

Approximate Entropy Technique of Calculation Based on Parallel Computation with Usage of GPU

Ruslan Mullayanov
Department of Informatics and Roboticst
Ufa state aviation technical university
Ufa, Russia
mullruslan@yandex.ru

Alexander Krushkov
Department of Informatics and Roboticst
Ufa state aviation technical university
Ufa, Russia
danteform@gmail.com

Rashit Nasyrov*
Department of Informatics and Roboticst
Ufa state aviation technical university
Ufa, Russia
docentapris@yandex.ru

***Abstract*—A time of computing has a significant importance in some domains. Especially, it comes a medicine. Such method as an approximate entropy is widely used to analyze a biomedical data, but it has non-linear algorithmic complexity. Therefore, there is a requirement to decrease a time of calculating of an approximate entropy. In this paper new approach of calculating by using matrices and graphic processor unit is proposed. In addition, some results of this approach are shown. In order to propose a solution to this problem, it was necessary to evaluate the complexity of the algorithm for calculating approximate entropy. In this regard, it is also necessary to show its asymptotic complexity using the notation big-O. Based on the obtained complexity estimates, a new approach based on matrix calculations was developed and proposed. In this case, the matrix calculations themselves are implemented in the form of parallel computing using a graphics processor (GPU). The graphics processor has been widely used for machine learning and data mining, where parallel computing is of great importance in solving the problem of improving performance. Since matrix operations are well parallelized, this makes it possible to speed up the execution of matrix calculations. For calculations, one of the most promising tools is currently the TensorFlow platform. The comparative effectiveness of the proposed approach was evaluated.**

Keywords—approximate entropy, calculation performance, parallel computing, TensorFlow, graphic processor unit, matrices, signal complexity, asymptotic complexity, CUDA, algorithm, computational graph

I. INTRODUCTION

Biological and medical research objects are complex. The biomedical data obtained during their study have non-linear properties and complex behavior, e.g., heart rate variability (HRV), electroencephalography (EEG) etc. The signal complexity is usually estimated provided that the data were obtained by the dynamic system under stationary conditions. However, biomedical data are often unsteady and contain some noise [1].For these reasons, it was previously proposed to use the characteristics of signal non-linearity, in particular, entropy estimates. One of the most famous among such quantities is the approximate entropy [2].

Approximate entropy (ApEn) is a method in statistics, which allows quantifying the amount of regularity and the unpredictability of fluctuations over time-series data [2]. It is widely used in medical field and statistic researches. For example, it has been applied to classify EEG in psychiatric diseases [3, 4, 5], to analyze HRV [6, 7, 8], to assess hypoxemia severity [9], to measure of analgesia depth during propofol-remifentanil anesthesia [10], to predict of treatment resistance in obsessive-compulsive disorder patients [11], to analyze a complexity of cardiotocographic examinations [12], to detect coronary heart disease [13], to detect early fault of ball bearing [14], etc. Also entropy measurements have been explored in activities of daily living [15]. This entropy measure is by definition dependents on two predefined parameter values, namely: m (the embedding dimensions) and r (the tolerance threshold); these parameters has different values by an application area. Thus, the first main drawback of this parameter is the difficulty of comparing various studies, since different values of a priori parameters can lead to different physiological interpretations [16, 17]. Another disadvantage is the complexity of the algorithm. Computing time is of great importance, especially in medicine, but its asymptotic complexity does not allow it to be used for a large data set. Therefore, the problem arises of increasing the productivity of computing this parameter.

Thus, the aim of the work is to increase the productivity of approximate entropy calculation. To achieve this goal, it is necessary to solve the following tasks. First, it was necessary to evaluate the complexity of the algorithm for calculating approximate entropy. Secondly, to propose an approach to increase the productivity of approximate entropy calculation. Thirdly, choose a tool for implementing the proposed approach. Fourth, assess the effectiveness of this implementation of the proposed approach.

To evaluate the complexity of the algorithm for calculating approximate entropy, it is necessary to use a unified indicator. The search for estimates of the complexity of the approximate entropy algorithm in the available literature yielded no results. In this regard, it is also necessary to show its asymptotic complexity using the notation big-O.

Based on the obtained complexity estimates, a new approach based on matrix calculations was developed and proposed. In this case, the matrix calculations themselves are implemented in the form of parallel computing using a graphics processor (GPU). It is known that in recent decades, the graphics processor has been widely used for machine learning and data mining, where parallel computing is of great importance in solving the problem of improving performance. Since matrix operations are well parallelized, this makes it possible to speed up the execution of matrix calculations.

One of the most promising tools currently is the TensorFlow platform [18], which uses Google's CUDA and is widely used for GPU computing using the concept of tensors.

II. Entropy Measure

To understand a proposed approach of calculating of an approximate entropy we should show the original method approach.

Given a sequence of numbers:

$$S_N = \{u(i): u(i) \in R \; and \; 1 \leq i \leq N\}. \quad (1)$$

Form vector sequences *x(j)* defined by:

$$x(j) = [u(j), \dots, u(j+m-1)]. \quad (2)$$

Define the distance *d[x(i), x(j)]* between vectors *x(i)* and *x(j)* as a maximum difference in their respective scalar components:

$$d[x(i), x(j)] = \\ = max_{k=1,\dots,m}(|\mathrm{u(i+k-1)} - \mathrm{u(j+k-1)}|). \quad (3)$$

Each distance is compared with a threshold *r* to compute the number of reconstructed vectors that lie within a hyperspace centered in the reconstructed vector of reference:

$$C_i^m = \frac{1}{N-m+1} \sum_{j}^{N-m+1} H(r - d_{i,j}^m), \quad (4)$$

where C_i^m is the correlation sum and *H* is the Heaviside function.

This procedure is repeated with all reconstructed vectors and the probability of a pattern of length *m* appearing along the time series was denoted by the following:

$$\varphi^m(r) = \frac{1}{N-m+1} \sum_{i=1}^{N-m+1} \log\left(C_i^m(r)\right). \quad (5)$$

And ApEn is defined based on the correlation integral, computed for 2 embedding dimensions:

$$ApEn(m,r) = \varphi^m(r) - \varphi^{m+1}(r). \quad (6)$$

The number of recurrences is higher in the lower dimensions [1]. So ApEn increases if the number of recurrences decreases when the embedding dimensions are increased (from *m* to *m*+1) [1].

Generally speaking, ApEn is approximately equal to the negative average natural logarithm of the conditional probability that two subseries of length *m* that are similar remain similar for subseries of length *m*+1 [17].

III. Algorithm Complexity

To assess the relative effectiveness of methods for solving the problem of increasing the speed of calculations, it is first necessary to determine the theoretical complexity of the original algorithm.

Since the basic steps are presented in the second section, but to determine the complexity, it may be useful to present the algorithm in pseudo-code (Algorithm 1).

Algorithm 1: Approximate Entropy

```
Input: Data points (S), m, r
Result: Entropy measure (Φ1 − Φ2)
N ← length of S;
Nm ← N − m + 1;
i = 0;
while i < Nm do
    U1i ← {Sj}_{j=i}^{i+m};
    U2i ← {Sj}_{j=i}^{i+m+1};
    i ← i + 1;
end
i = 0;
while i < Nm do
    count = 0;
    j = 0;
    while j < Nm do
        if max({U1i} − {U1j}) <= r then
            count ← count + 1;
        end
        j ← j + 1;
    end
    C1i ← count/Nm;
    i ← i + 1;
end
Φ1 ← sumi(C1i)/Nm;
i = 0;
while i < Nm − 1 do
    count = 0;
    j = 0;
    while j < Nm − 1 do
        if max({U2i} − {U2j}) <= r then
            count ← count + 1;
        end
        j ← j + 1;
    end
    C2i ← count/(Nm−1);
    i ← i + 1;
end
Φ2 ← sumi(C2i)/(Nm−1);
Φ1 − Φ2;
```

The algorithm consists of three parts. The first part is a loop of creating a vector sequences presented in (2). The second part is loops of comparing with a threshold *r* to compute the number of reconstructed vectors that lie within a hyperspace centered in the reconstructed vector of reference for *m* and *m* + 1 that are presented in (2) and (3). The third part calculates φ for *m* and *m* + 1.

Obviously, the algorithm performance depends on data size and *m* parameter. However, we can change *m* argument to give the best case and the worst case [19].

We give a case of an algorithm complexity for a time in the worst case, which will be shown in the results later. Consider we have *N* points of data and *m* = 1, then the *U*1 and *U*2 in the algorithm on Fig. 1 will have size *N* and *N* – 1 respectively. In this case, the second part of the algorithm will do the operations in $N^2T + (N-1)^2T$ time, where *T* is some constant. All the operations will be performed in the following time:

$$NT + N^2T + (N-1)^2T = O(N^2). \quad (7)$$

For the second case, we have an algorithm complexity for a time in the best case. Let us have the same number of points of data and $m = N - 1$, then the $U1$ and $U2$ in the algorithm on Fig. 1 will have size 2 and 1 respectively. In this case, the second part will do its operations in $4NT + NT$ time. All the operations will be performed in the following time:

$$NT + 4NT + NT = o(N). \quad (8)$$

In this paper, we consider an algorithm complexity for a time in the worst case and show results.

There is a possibility to improve this algorithm. We can reduce using of memory by removing the $U1$ and the $U2$. Then we can replace them in the function for finding a maximum value. In this paper, memory complexity is not shown.

IV. Matrix approach

The modern technologies allow performing of calculations by using GPU. It opens up new possibilities for a computation. But to parallelize a computation of an approximate entropy we propose to represent a vector sequences given by (2) as a matrix of size $(N - m + 1) \times m$:

$$X_{N-m+1,m} = \begin{pmatrix} u(1) & \cdots & u(m) \\ \vdots & \ddots & \vdots \\ u(N-m+1) & \cdots & u(N) \end{pmatrix}. \quad (9)$$

Define ashift operation to shift elements of matrix as multiplication of a source matrix and a shift matrix:

$$X^{(i)}_{N-m+1,m} = S^i \times X. \quad (10)$$

A shift matrix has the size $(N - m + 1) \times (N - m + 1)$ and it is described as:

$$S = \begin{pmatrix} s_{11} & \cdots & s_{1,N-m+1} \\ \vdots & \ddots & \vdots \\ s_{N-m+1,1} & \cdots & s_{N-m+1,N-m+1} \end{pmatrix}, \quad (11)$$

where$s_i$is defined as:

$$s_{ij} = \begin{cases} 1, & \text{if } i = j + 1 \\ 1, & j = N - m + 1 \text{ and } i = 0. \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

Thus, a shift matrix has only ones and zeros.

We give a matrix of distances by subtracting the source matrix and the shifted source matrix:

$$D_i = X - X^{(i)}. \quad (13)$$

Thus, we has all the differences of $x(i)$ and $x(j)$ by shifting N times. However, there is still a need to define function for finding a maximum value for matrix. This function represented as:

$$\begin{aligned} reducemax &: \mathbb{R}^{n\times n} \to \mathbb{R}^n, \\ U &= reducemax\left(A_{K,M}\right), \end{aligned} \quad (14)$$

where $u_i = \max_{1\le j\le M} a_{ij}, i = 1, \dots, K$. The vector of maximum distances is given by using this function. Now we use the Heaviside function, which should be applied to the vector:

$$h_l = H(u_l) \quad (15)$$

Where h_l is an element of vector H. If an element of this vector equals one this means that length of the vector d_j of $D_i$is less than r.And, finally, we summarize elements of the resulting vector with ones and zeros, divide it into $N - m + 1$to getC_i^m (4).Logarithm from (5) is calculated the same. The next steps is the same as in the section two, but for vectors.

V. TensorFlow framework

TensorFlow is an end-to-end open source platform for machine learning developed by Google Inc. [18]. Machine learning algorithms in it are represented as computational graph [13]. This graph is a form of directed graph where vertices or nodes describe operations, while edges represent data flowing between the operations [13]. An example of a computation graph is shown in Fig. 2.

The basic type in the TensorFlow is *tensors* [20]. A tensor is a multi-dimensional collection of homogeneous values with a fixed type, where number of dimensions is termed its *rank* [20]. In the mathematical sense, a tensor is the generalization of two-dimensional matrices, one-dimensional vectors and scalars, which are simply tensors of rank zero [20].

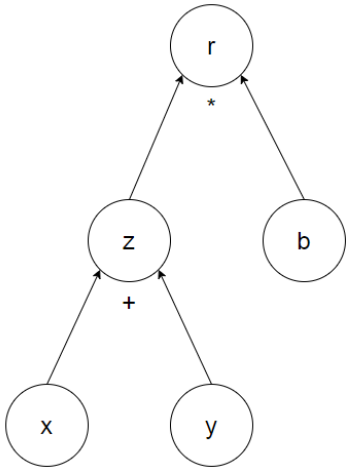

Fig. 1. A computational graph example

To execute computational graphs composed of the various elements, TensorFlow divides the tasks for its implementation among four distinct groups: the *client*, the *master*, a set of *workers* and *devices* [13]. Their interactions are shown in Fig. 3. The client sends a query for executing of a computation graph to the master process. The master delegates the task one or more worker processes.

TABLE I. CALCULATING RESULTS

Data size	Results (m = 2, r = 2)					
	GPU ApEn	*GPU Time*	*CPU ApEn (Python)*	*CPU Time (Python)*	*CPU ApEn (C++)*	*CPU Time (C++)*
1000	0,8437	0,5408	0,8437	18,0843	0,8437	0,0626
2000	0,8447	0,9235	0,8447	71,1068	0,8447	0,2338
3000	0,8435	1,293	0,8435	160,727	0,8435	0,5217
4000	0,842	1,6762	0,842	290,1594	0,842	0,9226
5000	0,8446	2,0799	n/a	n/a	0,8446	1,4382
10000	0,8446	4,0787	n/a	n/a	0,8445	5,7337
15000	0,8431	6,0205	n/a	n/a	0,8431	12,9046
20000	0,8418	8,1756	n/a	n/a	0,8419	22,5808
25000	0,8404	10,2693	n/a	n/a	0,8404	35,2846
30000	0,8401	13,1882	n/a	n/a	0,8401	50,752
40000	0,8409	19,1218	n/a	n/a	0,8409	90,2389
50000	0,8397	26,1304	n/a	n/a	0,8397	140,881
60000	0,8397	35,1	n/a	n/a	0,8397	202,848
70000	0,8398	44,5593	n/a	n/a	n/a	n/a
80000	0,8397	56,0116	n/a	n/a	n/a	n/a

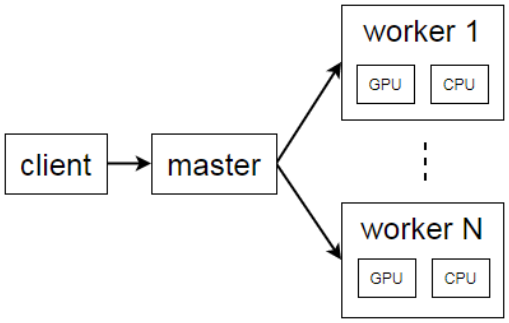

Fig. 2. A visualization of the different execution agents in a multi-machine and multi-device hardware configuration

Devices are basis entities in the TensorFlow execution model. A device will most often be either a CPU or a GPU [20].

For TensorFlow a visualization toolkit has been developed named TensorBoard [18]. This toolkit allows:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph
- Displaying images, text and audio data
- Profiling TensorFlow programs and more

The second point allows to show the resulting graph for calculating of an approximate entropy.

VI. RESULTS OF NUMERICAL EXPERIMENT

The algorithm has been implemented by using Python and C++ programming languages. Moreover, Python is used for computation on a CPU and a GPU by using NVIDIA CUDA.

The algorithm is tested using computer characteristics, which are shown in Table I.

TABLE II. TESTING PLATFORM CHARACTERISTICS

Component	Model
CPU	Intel Core i3-2130 3.40 GHz (AMD64)
GPU	NVIDIA GeForce GTX 1050 Ti
OS	Kubuntu 18.04 (Linux)
RAM	8 Gb

The GPU has Pascal architecture, 768 CUDA cores, 4 GB framebuffer [21].

The resulting computation graph is shown in Fig. 4. It consists of two loops for shifting matrices (9). The matrix (11) is ineffective for large dataset. However, TensorFlow supports rolling of tensors by using a hardware [20]. It allows to reduce an used memory .At the end an absolute value of ApEn has been calculate to get non-negative value of an entropy measure.

The results are shown in Table II. Testing the algorithm carried out with step equals 500. The initial size of the dataset is 500, and the final 80,000. Not only runtime is shown, but also the value of entropy. For all the runs *m* is equal 2 and *r* is equal 2.

A runtime plot versus data size is shown in Fig. 5. As can see, the Python is very ineffective, with size of data equals 5000 points the Python implementation of the algorithm performs calculation ApEn for a very long time.

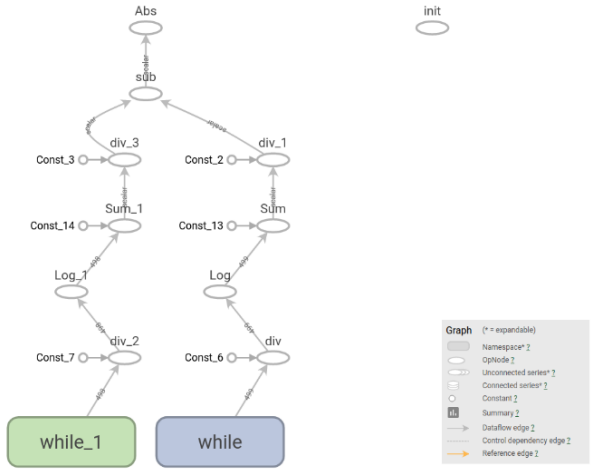

Fig. 3. A computational graph example

The C++ is effective, with size of data equals 60,000 points the C++ implementation of the algorithm performs calculation of ApEn for ~202 seconds. This implementation has been optimized for using a memory (see Section II). This implementation delivers the best runtime results for small dataset.

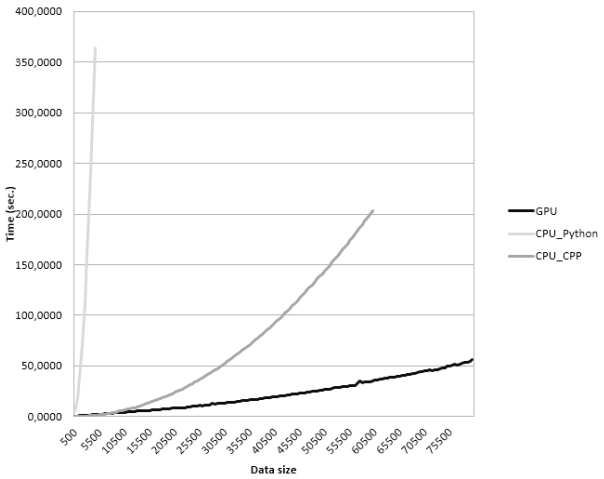

Fig. 4. A runtime plot versus data size in the worst case

CUDA delivers the best runtime results for large dataset. The TensorFlow implementation performs calculation for ~56 seconds by using size of data equals 80,000. However, it is ineffective for small datasets.

VII. Conclusion

The article considers the calculation of approximate entropy as a significant indicator of the nonlinear dynamics of complex processes.

At the same time, the complexity of the calculation was estimated for various components of the algorithm and it was shown that the total complexity is O (N2). This shows that the execution time of the algorithm grows exponentially in the worst case.

An approach is proposed to increase the productivity of calculating approximate entropy based on a matrix representation, which allows you to efficiently parallelize the algorithm for calculating approximate entropy and reduce its execution time.

Since modern GPUs have large computing resources, the TensorFlow library for GPU computing was used to implement the proposed approach.

The comparative effectiveness of the proposed approach was evaluated, which showed a significant reduction in the calculation time for large volumes of data (up to one order of magnitude) and the possibility of operating with a significantly larger volume of source data.

[1] Mesin L, “Estimation of Complexity of Sampled Biomedical Continuous Time Signals Using Approximate Entropy”. Front. Physiol. 9:710. June 2018. doi: 10.3389/fphys.2018.00710

[2] Pincus S.M., Gladstone I.M., Ehrenkranz R.A. “A regularity statistic for medical data analysis”. Journal of Climical Monitoring and Computing. October 1991, pp. 335-345.

[3] Sabeti, Malihe, “Entropy and complexity measures for EEG signal classification of schizophrenic and control participants”. Artificial Intelligence in Medicine. 2009. pp. 263-274.

[4] Yua, Qi “Epileptic EEG classification based on extreme learning machine and nonlinear features”. Epilepsy Research. 2011. pp. 29-38.

[5] Yun Kyongsik. “Decreased cortial complexity in methamphetamine abusers”. Phychiatry Research: Neuroimaging. 2011. pp. 226-232.

[6] Frank Beckers, Dirk Ramaekers, Andre E. Aubert, “Approximate Entropy of Heart Rate Variability: Validation of Methods and Application in Heart Failure”. Cardiovascular Engineering: An International Journal. 2001. pp. 177-182.

[7] Singh, V., Gupta, A., Sohal, J.S., Singh, A. “A unified non-linear approach based on recurrence quantification analysis and approximate entropy: application to the classification of heart rate variability of age-stratified subjects”. Medical and Biological Engineering and Computing. Vol. 57. Issue 3. 12 March 2019. pp. 741-755. doi: 10.1007/s11517-018-1914-0

[8] Ahn, J.M., Kim, J.K. “Approximate entropy for heart rate variability: Effect of tolerance and data length”. International Journal of Engineering Research and Technology. Vol. 12. Issue 11. 2019. pp. 1992-1999

[9] Liu, J., Huang, R., Xiao, Y., Lin, S., “ApEn for assessing hypoxemia severity in obstructive sleep apnea hypopnea syndrome patients”. Sleep and Breating. 2020. doi: 10.1007/s11325-019-02004-0

[10] Chen, W., Jiang, F., Chen, X., Feng, Y., Miao, J., Chen, S., Jiao, C., Chen, H., “Photoplethysmography-derived approximate entropy and sample entropy as measures of analgesia depth during propofol–remifentanil anesthesia”. Journal of Clinical Monitoring and Computing. 2020. doi: 10.1007/s10877-020-00470-6

[11] Altuglu, T.B., Metin, B., Tulay, E.E., Tan, O., Sayar, G.H., Tas,. C, Arikakn, K., Tarhan, N. “Prediction of treatment resistance in obsessive compulsive disorder patients based on EEG complexity as a biomarker”. Clinical Neurophysiology. Vol. 131. March 2020. Issue 3. pp. 716-724. doi: 10.1016/j.clinph.2019.11.063

[12] Marques, J.A.L., Cortez, P.C., Madeiro, J.P.V., de Albuquerque, V.H.C., Fong, S.J., Schlindwein, F.S. “Nonlinear characterization and complexity analysis of cardiotocographic examinations using entropy measures”. Journal of Supercomputing. Vol. 76. Issue 2. 1 February 2020.pp. 1305-1320. doi: 10.1007/s11227-018-2570-8

[13] Saxena, S., Gupta, V.K., Hrisheekesha, P.N. “Coronary heart disease detection using nonlinear features and online sequential extreme learning machine”. Biomedical Engineering - Applications, Basis and Communications. Vol. 31. Issue 6. 1 December 2019. Article № 1950046. doi: 10.4015/S1016237219500467

[14] Hoseinzadeh, M.S., Khadem, S.E., Sadooghi, M.S. “Modifying the Hilbert-Huang transform using the nonlinear entropy-based features for early fault detection of ball bearings”. Applied Acoustics. Vol. 150. July 2019. Pages 313-324. doi: 10.1016/j.apacoust.2019.02.011

[15] Howedi, A., Lotfi, A., Pourabdollah, A. “Exploring entropy measurements to identify multi-occupancy in activities of daily living”.Entropy. Vol. 21, Issue 4. 1 April 2019. Article №416. doi: 10.3390/e21040416

[16] Juan Bolea, Raquel Bailon “On the Standardization of Approximate Entropy: Multidimensional Approximate Entropy Index Evaluated on Short-Term HRV Time Series”. 2018.

[17] Montesinos, L., Castaldo, R., Pecchia, L. “On the use of approximate entropy and sample entropy with centre of pressure time-series”. J NeuroEngineering Rehabil 15, 116 (2018). doi: 10.1186/s12984-018-0465-9

[18] TensorFlow site. URL: https://tensorflor.org/

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. “Introduction to algorithms”. MIT Press, Cambridge, MA, third edition, 2009.

[20] Goldsborough P., “A Tour of TensorFlow”. Proseminar Data Mining. October 2016.

[21] NVIDIA. URL: nvidia.com/ru-ru/geforce/products/10series/geforce-gtx-1050