Program Toolkit for Auditory Scenes' Analysis

Evgeniy L. Romanov, Yuliya V. Novitskaya
Novosibirsk State Technical University, Novosibirsk, Russia

Abstract – **In this paper the architecture of program toolkit for auditory scenes' analysis and the features of spiking neural networks and neuromorphic structures using are proposed. The programming examples and the test results of main components are given.**

Index Terms – **Auditory scene, spiking neural network, Java, spectrum, cochleagram.**

I. INTRODUCTION

COMPUTATIONAL auditory scene analysis (CASA) considers the deep processing of audio data — music and human speech. CASA's algorithms and methods implement the pre-processing and enhancement of audio signals in applied tasks' solving. Neural networks (NN), genetic algorithms, swarm algorithms are widely used in CASA.

For high-quality experimental results the program toolkit is needed. Here are some toolkit features:

- open source code, programming new components based on existing components;
- design of system from of the set of available components based on the configuration description language;
- design with using of the neural networks' artefacts, genetic algorithms, swarm algorithms;
- tools for tracing and statistics gathering, exception handling and error handling;
- input and output data visualization;
- use of the third-party signal processing libraries, neural networks and machine learning;
- integration with known parallel processing tools on the basis of graphics processing units (GPU);
- input and output acoustic check;
- saving of data and models' state.

II. SPIKING NEURAL NETWORKS AND NEUROMORPHIC STRUCTURES

The CASA's architecture depends on the used analysis and simulation tools; neural network is one of these tools. On the other hand the choice of specific NN for using in CASA depends on the specifics of audio data.

A. Dynamics and the neural network time factor

In most cases neural networks deal with statistical data. If NN input data contain time factor it will be transformed into additional model dimension on NN input. Dynamics of NN training/recognition processes has no relation to additional dimension. Here are typical examples:

- oscillatory NN — building on the modelling of biological neuron's ionotropic component (membrane potential) and forming processes of persistent oscillations in neural ensembles [3]. Nevertheless the selection of components from the acoustic flow [1] is made on the matrix (f, t) where the time is one of matrix dimension;
- recurrent NN use the feedbacks or the sequence of delayed values of input signal as the inputs to different neurons or subnets. For example, LSTM (Long Short-Term Memory) recurrent networks are used for recognition of formants' sequence in short phrases [4].

At spiking neural networks (SNN) the time factors concerned with operating algorithm of neuron (accumulation of membrane potential and so on) can be associated with model behaviour in time. However, above reasons complicate the SNN organization and learning algorithms.

B. Neural network models for CASA

Common principles of NN design and learning are not considered in this paper. Nevertheless, some neurophysiologic facts must also be taken into consideration «a priory» while program components are developing.

Synaptic traps. Autonomic activities named "dendritic spikes" are possible in separate branches of biological neurons with wide network of dendrites. Dendritic spikes are caused both by synapses' local activation and spike's back-propagation along the dendritic tree. All this allow modelling the spiking neurons as a system of synaptic traps. Synaptic traps catch the patterns (spatial and temporal combinations) of input synaptic pulses. Several synaptic traps allow the neuron gets in the neural ensembles' activation [14].

Patterns of synaptic traps. Set of signals on the trap synapses could be described in spatial (boolean) or temporal logic [8]. Legitimacy of temporal logic using is supported by neurophysiologic facts. For example, NDMA-receptors [5] possess the unblocking (permissive) mechanism actuated by sufficient membrane depolarization (after preliminary accumulation of activity from other synapses).

In principle, the operating logic of the synaptic traps can no follow any neurophysiologic model. It is enough to reproduce the idea of some significant pattern set detection (selection). Here are some examples:

- activity front (the sequence of pulses after pause);
- activity recession (the pause after sequence of pulses);
- temporal implication, precedence (A allows B during some interval);
- threshold amount;
- simple spatio-temporal merger (OR);
- increasing of pulse consecution's frequency;
- decreasing of pulse consecution's frequency.

Acoustic patterns. In case of the acoustic signal's structure the acoustic traps can be specialized for relevant audio patterns. Both the topology of synapse connections and the spatio-temporal structure of input signals are involved in process of pattern implementation:

- forward (back) front of sound;
- appearance (disappearance) of frequency signal;
- frequency deviation;
- frequency increase (decrease);
- harmonic intervals;

- dissonances;
- narrowband signal in the range (harmonic sound);
- broadband signal in the range (noise).

Synaptic plasticity. Change of synapses' (synaptic traps') sensitivity is the standard factor in the spiking neuron's models. Here are possible kinds of synaptic plasticity:

- spike-timing-dependent plasticity (STDP) — increase of synapse's (synaptic trap's) sensitivity if synapse fires spike, decrease of synapse's sensitivity in case of following after spike. In biological neurons STDP is implemented with back-propagation potential. STDP accords to Hebbian learning for each individual neuron and is used in local algorithms of neuron's learning;
- short-time plasticity connected to change of synapses' (traps') sensitivity in case of their potentiation (variable neuron's reaction on the sequence of spikes). There are many metabotropic receptors together with the ionotropic receptors in the biological neurons. The ionotropic receptors have a direct effect on the membrane potential. The metabotropic receptors have a more inert and continuous direct effect on the ion channels by chemical interaction;
- long-time plasticity associated with genesis of NN structure; learning strictly speaking.

The time factor at NN is necessary for providing of dynamic connectivity of network operation processes. Time factor can be provided in neuron by following resources:

- spiking neuron as natural integrator of input activity;
- temporal logic, short-time plasticity;
- inertance of processes in neuron;
- spontaneous and oscillatory activities of neuron.

At the same time NN dynamics can be determinate by the structure of NN synaptic connections — feedbacks, recurrent structures. More deep dynamics can be determinate by the cyclic activity of neural ensembles.

III. SOFTWARE ARCHITECTURE

A. Key points

Java vs. MatLab & Python. Development environment for program toolkit is Java. Here are obvious advantages:

- cross-platform;
- high-speed code execution with JIT-compilation compared to C/C++;
- open source libraries including NN and machine learning frameworks [9,10];
- migration possibilities to more advanced development tools — Scala, Kotlin.

Open source code — this is not only code availability, this is great variety internal and external interfaces, abstract basic entities (classes) and services for creating programming tools in the problem area. This includes:

- basic components of model program environment: data sources, statistics gathering, tracing and event generation, exception handling and error handling;
- basic components of model adapted and developed for real tasks — layers, neurons, data sources;
- application frameworks and implementation examples.

B. Representation of the audio data at the model input

Spectral representation of the acoustic data builds on the analogy with the structure of receptive hearing. Representation form — frequency representation (spectrum), audible frequency range from 20 Hz to 20 kHz, amplitude dynamic range — 90 dB (correspond to amplitude representation of float variable). Certain factors of subjective perception (features of audio data and music processing) should be taken into account during processing:

- dynamic range of loudness (the amplitudes of receptive signals) is perceived in a logarithmic scale and range compression is required during processing. It is convenient to compress the spectrum (compression of the spectrum amplitudes) than the input signal;
- frequency (tone) range is perceived in a logarithmic scale also. Multiple frequency ratios 1: 2, 2: 3, 3: 4, etc. correspond to harmonic musical intervals. Spectrum of vowel sounds includes similar octave, fifths and other intervals. So the spectrum scale is reduced to semitones rather than to mels. Halftone structure is used (12 semitones in octave with fixed frequency ratio $2^{1/12}$ at equal temperament), for representing the scale and multiple harmonics in the range of 10 octaves enough 120 values. Range step can be divided into halftone shares with frequency rate of degree 2. Range determined in this way will further be called **the input range**.

Gammatone. Cochleagram. Gammatone filter has been proposed as a model for impulse function of the snail's membrane (cochlear) fibre response [1]. Gammatone means the band filter with the impulse response as a product of the gamma function and the resonance tone *fc* (hence the term "gammatone").

$$g_{fc}(t) = t^{N-1} e^{-2\pi t b(fc)} \cos\ (2\pi f_c t + \phi) u(t)$$

Here *N* — filter order, *u(t)* — single "step" at *t=0*, *b(fc)* — bandwidth for *fc*. Program code for gammatone calculation is given in [2].

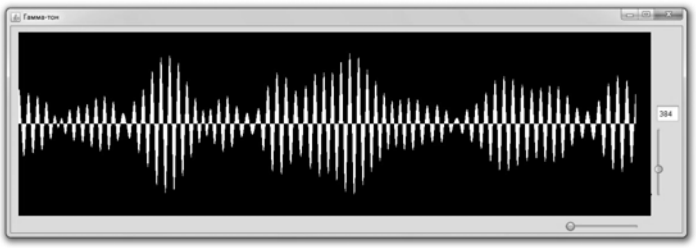


Fig. 1. Reaction of gammatone-filter on aperiodic signal.

Set of gammatone-filters for the resonant tones of the input range reacts on common input signal and produces cochleagram (Fig. 2).

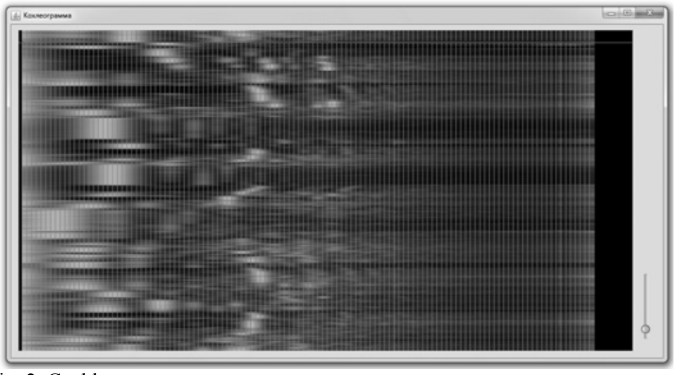


Fig. 2. Cochleagram.

Cochleagram means the time-to-frequency conversion and based on a physical model of resonance oscillations. Cochleagram reflects the signal dynamics in the time domain more precisely and in the frequency domain — more blurry in comparison with the usual spectrum obtained in the sliding window mode.

C. Architectural components and patterns

Class diagram for the most significant part of core is shown in Fig. 3. Core consists of:

- factory of classes for the neurons, layers, models with the dropdown lists generation for the graphical user interface (TypeFactory);
- meta-classes of description (ModelDescript), tools for translating description and linking models from the set of layers, neurons and tools for their regular commutation;
- model basic element interfaces: signal (I_Spike), neuron output (I_NeuronOutput), neuron (I_Neuron), layer (I_Layer), layer of neurons (I_NeuronLayer), model (I_LayerModel), regular connections of layers (I_RegularConnector);
- classes of basic elements;
- derived classes of elements created by the model developer (user-programmer);
- interfaces and classes for gathering statistics of layer operation and for saving/loading model into text stream.

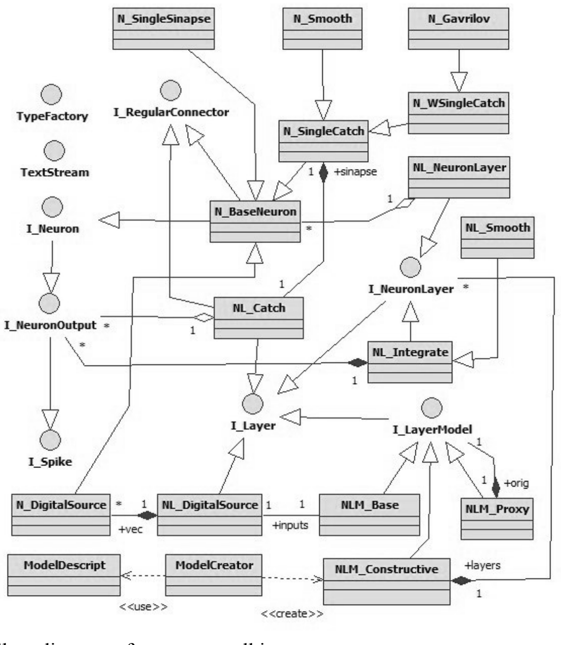


Fig. 3. Class diagram of program toolkit core.

Basic elements: signals, layers, neurons, models. The system uses combined logical and numerical data representation at discrete time. **The signal** in standardized real format in range from 0 to 1 at the same time can be interpreted as logical pulse signal (**spike**). The element processing the signal defines interpretation method.

Layer — set of signals matches the spectrum range in the program model. The data sources are layers side by side with two basic layer handling components — **integrated layer** and **neuron layer**. The integrated layer is programmed as the whole layer behaviour algorithm. Also model developer has possibility to create the internal entities (workers at swarm algorithms for example) not strictly bound with the input range. The neuron layer provides the regular structure of the objects-neurons, the initialization procedure building on the object-prototype, the standard methods for regular communications with neurons from other layers (one-to-one, one-to-one with offset, one-to-many).

Program model operation algorithm. On the whole the neurons, the layers and the model operate within a two-phase algorithm. At the modelling phase each element forms the temporal value of output signal. At the data timing phase the temporal output signal is copied to the actual output of element.

Programming of neurons. Base neuron (N_BaseNeuron) is the container with defined interfaces of its operation. Basically components and classes of base neuron correspond to the model of spiking neuron:

- membrane potential, adjustable trigger level, leakage;
- programmable change of potential, actuating and refractory period;
- programmable event associated with actuating of neuron.

There is the necessary program environment for neuron operation in the same class. Environment includes identification, name setting, learning mode setting, read/write of status into the text stream, reset methods and data synchronization phases.

Base neuron does not have to model the behaviour of spiking neuron. Call of methods setSpike/setOutValue allows to directly setting the value of output signal at the current modelling step.

Derived classes consistently widen the functionality of base neuron. **Neuron with single synapse** (N_SingleSynapsed) is the first functionally full element. **Synaptic trap**'s class (NL_Catch) with layer interface contains vector of inputs (synapses) and supports methods of regular communications with other layers. **Neuron with many inputs** (N_SingleCatch) is also the functionally full element building on synaptic trap.

Hebbian learning algorithm (STDP) is implemented in class-inheritor N_SingleWCatch. Each synapse k has own weight W_k, modelling steps are numbered by each neuron from the moment of its actuating $i = 0 \ldots n$. For each synapse is stored the step number when spike was at synapse entrance — S_k. Synapse weights recalculates at neuron actuating according to the formula

$$W_k = W_k + 2n * dw(S_k - n/2)$$

with range limitation of weight change.

Programming of specific spiking neuron [13] looks rather trivial.

```
@Override
public void changePotecial(I_NeuronStep back) {
    super.changePotecial(back);
    float sum = 0;
    for (int i = 0; i < size(); i++) {
        float in = get(i).getSpike();
        sum += in*R*getWeight(i);
    }   // Сумма весов сработавших синапсов
    this.addPotential(sum);
    if (H > HMin)       // Floating trigger level
        H -= DH;
    setLevel(H);
}

@Override
public void onFire(I_NeuronStep back) {
    super.onFire(back);
    H = HMax;           // Return to the initial sensitivity level
}
```

Programming of integrated layers differs by only representation of data and algorithms. Representation of data and algorithms is not localized as separate objects-neurons linked with input range. It is executed as a single program component.

Designing of models. The model may be designed from existing classes of layers and neurons. Model description is made in XML format and contains:

- named layers;
- named integrated layers and layers of neurons with names of neuron-prototype classes;
- standard ways for commutation of layers;
- parameters of layers and classes are properties of corresponding classes;
- pointing of input and output layers;
- named statistics assigned to layers.

Interpretation of model description is produced by deserialization into system of objects-descriptors. Building on system through reflection the required model objects are produced. Also parameters and links are produced. Below is example with the model description and the integrated layer of smoothing. There are two neural layers in model (with threshold and spiking neurons). Integrated layer of smoothing has the setting of parameters and the description of regular communications. Also description includes list of layers with standard statistics gathering and the output layer indication and the input layer commutated on data source.

```
<model>
    <layers>
        <neurons name="Layer 2" type="Gavrilov's Neuron">
            <param name="DH" value="0.05"/>
            <param name="HMax" value="0.95"/>
            <param name="L" value="0.01"/>
        </neurons>
        <neurons name="Layer 1" type="Threshold Input">
            <param name="spikeLevel" value="0.4"/>
        </neurons>
        <layer name="Layer 3" type="Smoothing">
            <param name="width" value="5"/>
        </layer>
    </layers>
    <link from="Layer 1" to="Layer 2" type="0" param="5"/>
    <link from="Layer 2" to="Layer 3" type="1" param="0"/>
    <link from="Input" to="Layer 1" type="1" param="0"/>
    <statistics>
        <statistic name="Layer 3" layer="Layer 3"/>
        <statistic name="Layer 2" layer="Layer 2"/>
        <statistic name="Input" layer="Input"/>
    </statistics>
    <input name="Layer 0"/>
    <output name="Layer 3"/>
</model>
```

Class NLM_Constructive stores the model building on description. For access to model the class NLM_Proxy is used.

Programming of models. Model configuration described above can be moved into program code as single class. The following code shows use of the same of objects' manipulation level both by program model and by description.

```
public class NLM_Gavrilov extends NLM_Base {
    private NL_Smooth smooth = null;
    private I_NeuronLayer gavrilov = null;
    private LayerStatistic stats[] = new LayerStatistic[3];
    @Override
    public void initModel(int subTones, I_NetParams params)
throws Exception {
        super.initModel(subTones, params);
        gavrilov = new NL_NeuronLayer();
        gavrilov.createLayer(new N_Gavrilov(), size());
        smooth = new NL_Smooth(5);
        layer.createLayer(new N_Level(), size());
        layer.addSynapse(inputs);
        gavrilov.addInputsLinear(5, layer);
        smooth.addSynapse(gavrilov);
        smooth.setWidth(5);
        for (int i = 0; i < layer.size(); i++){
            ((N_Gavrilov)gavrilov.get(i)).setDH(0.05f);
            ((N_Level)layer.get(i)).setSpikeLevel(0.4f);
        }
        stats[0] = new LayerStatistic(inputs,"Input");
        stats[1] = new LayerStatistic(smooth,"Layer 3");
        stats[2] = new LayerStatistic(gavrilov,"Layer 2");
}
    @Override
    public void reset() throws UniException {
        smooth.reset();
        gavrilov.reset();
        layer.reset();
    }
    @Override
    public float[] step(float[] in, I_NeuronStep back)
throws UniException {
        inputs.setValues(in);
        smooth.step(back);
        gavrilov.step(back);
        layer.step(back);
        stats[0].addStatistic();
        stats[1].addStatistic();
        stats[2].addStatistic();
        layer.synch();
        smooth.synch();
        gavrilov.synch();
        return smooth.getSpikes();
}
    @Override
    public TypeFactory<LayerStatistic> getFactory() {
        return new TypeFactory<LayerStatistic>() {
            add(stats[0]); add(stats[1]); add(stats[2]);
        }
    }
}
```

Program implementation is necessary if developer of model is not satisfied by configuration description, for example:

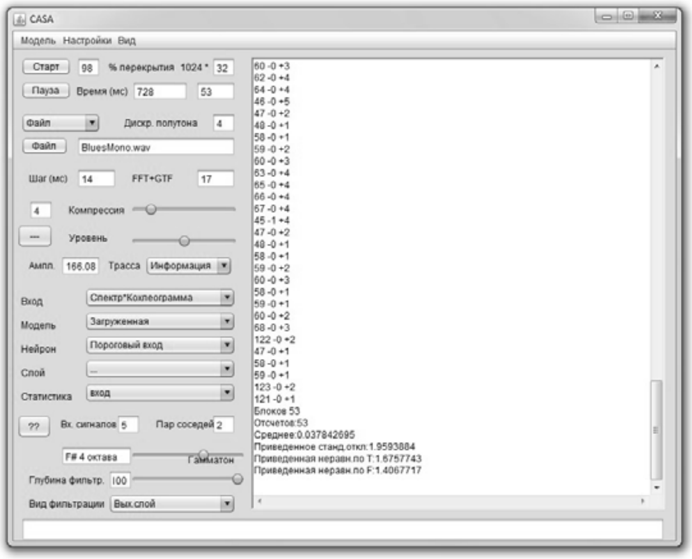
- nonstandard structure of model is used;
- structure of model is changing while operating (learning);
- integral machinery of behaviour description are required (differ from available).

Data sources. The program interface of data source (FFTAudioSource) specifies the receiving of input signal in wave format. Also the program interface specifies the control of the real time playback of input signal synchronized with operating of model. Classes-sources allow using of WAV files (44100 Hz; 16 bit; Mono) and the generators of harmonic signals and noises.

The spectrum, cochleagram or their bit-by-bit multiplication can be redirected to the input of model (see Sec. IV).

D. Graphical user interface, visualization, statistics, data saving, acoustic output and monitoring

The graphic interface is presented by a set of windows for display of layer dynamics as the bitmap *A(f,t)* and graphics *A(f)* with time scanning. The spectrum, cochleagram, selected tone gammatone, output layer of model, spectrum modulated by output layer are shown. The main window (Fig. 4) is implemented as manual tester. The main window allows settings of all possible modes and options of the program toolkit. Building on its code the graphic interface for any applications can be developed.



```
% перекрытия 98   1024 * 32
Время (мс) 728   53
Файл   Дискр. полутона 4
Файл BluesMono.wav
Шаг (мс) 14   FFT+GTF 17
4 Компрессия
--- Уровень
Ампл. 166.08   Трасса Информация
Вход Спектр*Кохлеограмма
Модель Загруженная
Нейрон Пороговый вход
Слой ---
Статистика вход
Вх. сигналов 5   Пар соседей 2
F# 4 октава   Гамматон
Глубина фильтр. 100
Вид фильтрации Вых.слой
```

```
60 -0 +3
62 -0 +4
64 -0 +4
46 -0 +5
47 -0 +2
48 -0 +1
58 -0 +1
59 -0 +2
60 -0 +3
63 -0 +4
65 -0 +4
66 -0 +4
67 -0 +4
45 -1 +4
47 -0 +2
48 -0 +1
58 -0 +1
59 -0 +2
60 -0 +3
58 -0 +1
59 -0 +1
60 -0 +2
68 -0 +3
122 -0 +2
47 -0 +1
58 -0 +1
59 -0 +1
123 -0 +2
121 -0 +1
Блоков 53
Отсчетов:53
Среднее:0.037842695
Приведенное станд.откл:1.9593884
Приведенная неравн.по T:1.6757743
Приведенная неравн.по F:1.4067717
```

Fig. 4. Program toolkit main window.

Acoustic output. The current state of output layer can be regarded as the spectrum. The acoustic output signal can be reconstructed building on spectrum. It is useful if the model itself is the sound synthesizer. Output layer of model also can be the control input of adaptive filter for spectrum of input signal. After filtration the reconstruction of audio signal is possible.

Acoustic check consists in the real time playback of input signal. In the process of acoustic check the modelling process can be slowed down for synchronizing with playback or the fragment playback can be repeated if the model "is not in time".

Saving of data and modelling results. In principle, any Java tools of object serialization can be used for saving of model state. However the compact format of text serialization and interface (TextStream) are developed for convenience of viewing and editing. Developed format includes not only serialized data itself but also format information. The fragment of saved state for mentioned model is given below.

```
//<Model type>/<Halftone counter>/<Layer amount>/<Layer of
output>/<Layer of input >{<Layers>}
Loaded/4/3/Layer 3/???
//Inputs
.../NL_DigitalSource/480/960
//<Layer name>/<Layer type>/<Neuron type>/<Neuron amount>
Layer 2/Neurons' layer/Gavrilov's neuron/480
//.../Gavrilov's neuron/L/size{Input number}
//kSens/W0/DW/size{w[i]}/R/H/Hmax/Hmin/DH
//<Neuron number>/<Neuron data
>
0/0.01/5/480/480/480/481/482/1.0/0.5/0.01/5/1.0/1.0/1.0/1.0/1.0/0.1/0.699
9999/0.95/0.7/0.05
1/0.01/5/480/480/481/482/483/1.0/0.5/0.01/5/1.0/1.0/1.0/1.0/1.0/0.1/0.699
9999/0.95/0.7/0.05
2/0.01/5/480/481/482/483/484/1.0/0.5/0.01/5/1.0/1.0/1.0/1.0/1.0/0.1/0.699
9999/0.95/0.7/0.05
```

Statistics. Multiple outputs of any layer are formally described as the function *A(f,t)*. If to consider the function as random function of two variables then it is possible to determine the primary layer statistics as mean μ_{fk} and standard deviation σ_{fk} for each frequency and for the whole layer — μ, σ.

The statistical value characterizing the dynamics of change in time (onward T-irregularity) can be determined both for the separate frequency of input range and for the whole layer (normalized to the mean of layer).

$$DT_{fk} = \sqrt{\frac{\sum_{i=1}^{n}(A_{ti,fk} - A_{ti-1,fk})^2}{n}} \qquad (1)$$

$$DT = \overline{DT_{fk}} \,/\, \mu \qquad (2)$$

The F-irregularity can be determined for input range similarly.

$$DF_{fk} = \sqrt{\frac{\sum_{i=1}^{n}(A_{ti,fk} - A_{ti,fk-1})^2 + (A_{ti,fk} - A_{ti,fk+1})^2}{2n}}, \qquad (3)$$

$$DF = \overline{DF_{fk}} \,/\, \mu\,. \qquad (4)$$

Fig. 5 shows statistics window for four parameters on frequency range.

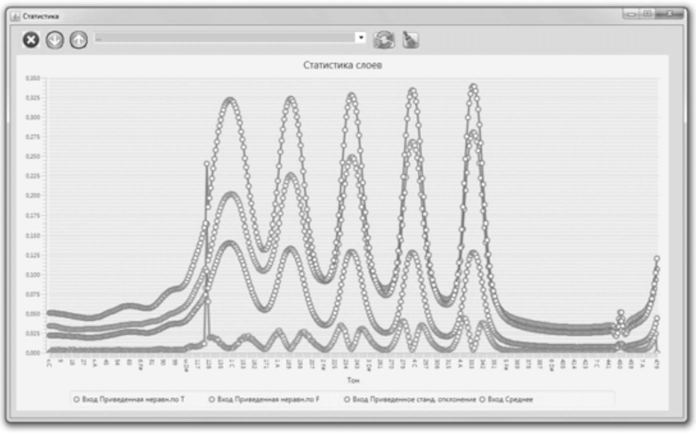


Fig. 5. Model input statistics — cochleagram on test signal "5 octaves with pause".

E. *Efficiency, parallel processing*

Pretesting has shown that modelling in real time with acoustic check on the desktop computer is already problematic when cochleagram was used on model input. There are decisions allowing to compile byte code into parallel graphic processor code in Java (in case of parallel processing using GPU) (APARAPI [11]). Technologically these decisions are the most convenient for using in CASA:

- spectral transformations, calculation of cochleagram;
- container for integrated layer with tools for parallel processing.

F. *Features of project*

The project is developed on Java 8 in IDE Netbeans 8.1. The source code is available at [15]. Source lines of code (SLOC) — 12200 lines (143 classes, 1032 functions).

IV. Experimental Results

A. *Comparative analysis of the audio data sources' informational content*

Using in CASA the usual spectrum of signal (getting by FFT in slide window mode) has the significant defect.

For obtaining the sufficient frequency resolution of the spectrum in the low frequency range (100–200 Hz; 1–2 Hz Discrete; Signal; Bass) the time range of input signal about 750 ms is necessary. At the same time the acceptable period of spectrum obtaining in the high frequency range (speech signal formant) is 10–15 ms. For obtaining these parameters the overlapping coefficient of the sliding window must be 95–98% that lead to "spreading" of the high frequency spectrum of the short-time signal upon the time range. At the same time cochleagram has opposed properties. Therefore the using "mix" of amplitude of spectrum $S_{f,t}$ and gammatone average value $G_{f,t}$ (or its local maximum) on current time interval of transformation t looks quite logical:

$$SG_{f,t} = S_{f,t} * G_{f,t} \,,$$

$$SG^{*}_{f,t} = \sqrt{kS_{f,t}^{2} + G_{f,t}^{2}} \,.$$

T-irregularity and F-irregularity (and their multiplication) will be considered as the information features.

Experiment 1. The signal out of 5 pure harmonics and intervals of sound (20 ms) and pause (200 ms).

TABLE I
STATISTICS OF INPUT MODEL ON TEST SIGNAL

Input	S	G	SG	SG*
μ	0,149	0,047	0,13	0,156
σ	1,045	2,506	2,2	1,044
DT	0,212	**1,978**	2,08	0,206
DF	**0,633**	0,21	1,24	0,638
*DT*DF*	0,13	0,42	**2,58**	0,13

Expected result: the spectrum and the cochleagram demonstrate the mutually inverse features of T- and F-irregularity. Direct multiplication of amplitude of spectrum and gammatone (average value or local maximum of envelope on the transformation step) gives the maximum estimation of irregularity.

Experiment 2. Acoustic file (music, male and female vocals).

TABLE II
STATISTICS OF INPUT MODEL ON ACOUSTIC FILE

Input	S	G	SG	SG without compression	SG*
μ	0,029	0,50	0,05	0,05	0,17
σ	1,322	1,06	2,06	2,08	1,28
DT	0,184	**0,24**	1,03	1,05	0,55
DF	**1,437**	0,16	2,28	2,28	0,78
*DF*DT*	0,26	0,04	**2,34**	**2,39**	0,43

As it turned out the estimation of cochleagram T-irregularity on acoustic file was much lower than on test signal. At the same time F-irregularity was much higher. Nevertheless the obvious defeat of cochleagram did not affect on the general estimation of irregularity for their "multiplication" (SG).

Remark: transformation features: FFT points — 16*1024, overlap of window — 97%, transformation step — 11 ms, discrete of halftone — 1, discrete of frequency in spectrum — 2,69 Hz, primary note identified on the spectrum — D# of low octave.

B. *Integrated layer for consolidation of melodic signal*

The main problem of melodic signal's analysis is tracing of melodic signal's dynamics. As first approximation the problem can be formulated as problem of binding the consecutive local maximums' of spectrum. To solve the problem the ideas of swarm algorithms were used [13] and the following principles of individuals' behaviour were defined:

- individuals are uniformly distributed over the spectrum range; the starting position is its "home"; individual is not able to move away than predefined distance from "home";
- individual seeks to track the value of spectrum at point where it is with certain inertia;
- individual seeks to move in the direction of the spectrum amplitude increase in its locality;
- individual moves to "home" in case of the absence of movement in the direction of increase and synchronous reducing the amplitude of spectrum lower than individual's current value.

The listed rules of conduct have to lead to concentration of individuals around the local maximums and to disintegration of formed groups in pauses. This behaviour

Such behaviour is directly observed at the aforesaid input test signal "7 harmonics with a pause" (Fig. 6).

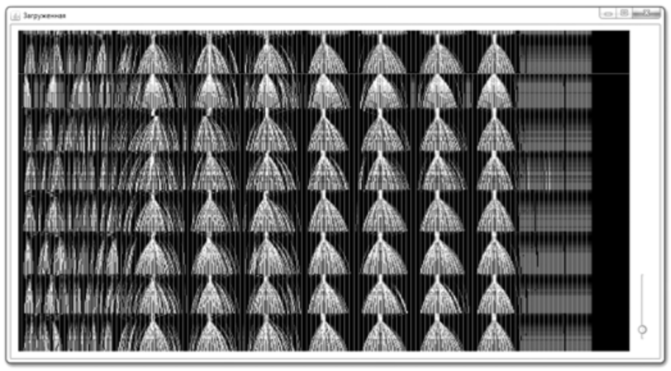


Fig. 6. Consolidation of "swarm" on test "7 harmonics with pause".

In case of actual piece of music the modelling behaviour did not get the formal estimate. However consolidation of swarm around maximums at input range is observed. Also their movements can be traced (Fig. 7–9).

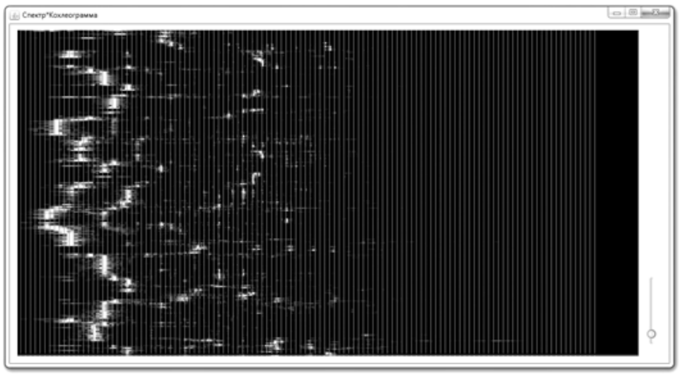


Fig. 7. Input signal (SG).

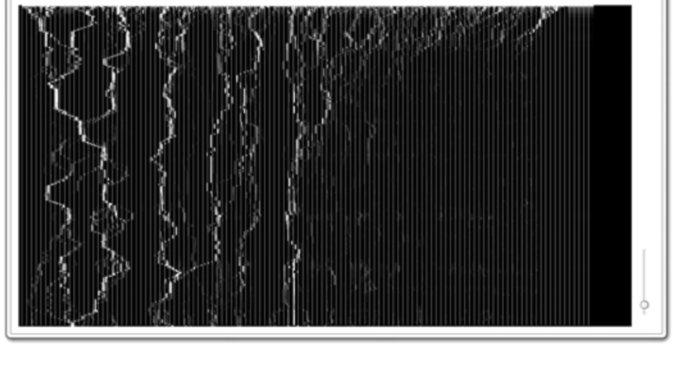


Fig. 8. Consolidation dynamics of “ants” on range.

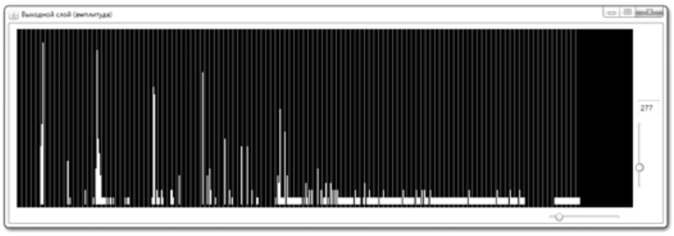


Fig. 9. Current consolidation of “ants” on range.

V. Conclusion

Proposed program toolkit provides the abilities for development of auditory scenes' analysis models and audio data deep processing. Now following additional aspects are running:

- services and interfaces for link and configuration of NN machine learning components and neuromorphic structures;
- tools for parallel processing of input signals and execution of models with GPU using;
- input converters and tools for models’ configuration (dynamic 2D-image models — video, sequence of slides).

References

[1] DeLiang Wang, Guy J. Brown. Computational Auditory Scene Analysis. Principles, Algorithms, and Applications. ISBN-13 978-0-471-74109-1, ISBN-I0 0-471-74109-4.

[2] Ning Ma. An Efficient Implementation of Gammatone Filters [Electronic resource]: Access mode – http://staffwww.dcs.shef.ac.uk/people/N.Ma/ resources/gammatone.

[3] Kashchenko S.A., Mayorov V.V., Models of Wave Memory. 2nd ed. – M.: Book House «LIBROKOM», 2013. – 288 p. (in Russian).

[4] Why Google's Voice Search Needs the Neural Networks? [Electronic resource]: Access mode – https://habrahabr.ru/company/google/blog/269747. (in Russian).

[5] Aleksandrov Yu.I. et al. Neuron. Signal Processing. Plasticity. Modeling. Fundamental Guide. Tyumen: Publishing House of Tyumen State University, 2008. – 548 p. (in Russian).

[6] Nicholls J., Martin R., Fuchs P. et al. From Neuron to Brain. – M: Editorial URSS, 2003. – 673 p. (in Russian).

[7] Aural Cortex and Human Speech / Knowledge Base of Human Biology [Electronic resource]: Access mode – http://humbio.ru/humbio/ssb/ 0012cd71.htm. (in Russian).

[8] Karpov Yu.G. Model Checking. Verification of Parallel and Distributed Software Systems. – SPb.: BHV-Peterburg, 2010. – 560 p. (in Russian).

[9] Encog Machine Learning Framework [Electronic resource]: Access mode – http://www.heatonresearch.com/encog.

[10] DL4J Deep Learning for Java [Electronic resource]: Access mode – http://deeplearning4j.org.

[11] APARAPI: API for Data Parallel Java [Electronic resource]: Access mode – https://code.google.com/archive/p/aparapi.

[12] Object Event Driven Programming. Swarm Algorithms. [Electronic resource]: Access mode – http://bourabai.ru/alg/swarm.htm. (in Russian).

[13] Andrey V. Gavrilov, Valeriy M. Kangler, Mikhail Katomin, Konstantin Panchenko. A Model of Spike Neuron Oriented to Hardware Implementation. – Proceedings of 11th International Forum on Strategic Technology (IFOST-2016), Novosibirsk, 2016.

[14] Romanov E. L. Program Model of Spiking Neural Network / E. L. Romanov // Robotics and Artificial Intelligence: Proceedings of VII All-Russian Scientific and Technical Conference with International Participation, (Zheleznogorsk, 11 December 2015) – Krasnoyarsk: Sib. Fed. Univercity, 2016. – p. 155-160. - ISBN 978-5-7638-3411-6. (in Russian).

[15] Код проекта: CASA-SNN [Electronic resource]: Access mode – https://bitbucket.org/solus_rex/snn_core.

Evgeniy L. Romanov received the System Engineer (Electronic Calculating Computers) degree from Novosibirsk Electric Technical Institute, PhD degree from Leningrad Electric Technical Institute. He is presently working as Associate Professor (Docent) in Novosibirsk State Technical University and has 30+ years of teaching experience in programming and software engineering to undergraduate and graduate students. He has more 50 publications. His areas of interests are technologies of programming, Java, Scala, software engineering, mobile and client-server applications, neural networks, neuromorphic technologies.

Yuliya V. Novitskaya received the System Engineer (Electronic Calculating Computers) degree from Novosibirsk Electric Technical Institute. She is presently working as a Senior Lecturer in Novosibirsk State Technical University and has 20+ years of teaching experience to undergraduate students. She has more 20 publications. Her areas of interests are ubiquitous computing and web-programming.