Stonito for Lotto with Neural Networks

This is the software for true lotto enthusiasts.

Stonito for Lotto comes handy to organize your round-to-round play better. It can also help you predict winning combination using state-of-the-art technology.

List of combination in the main form is the list of user played combinations. To the right there is a table showing the selected combination with some squares having different background color.

The color represents the probability of the number as a result of neural network calculation.
As the neural network is a main feature of this application the manual will start with detailed explanation of neural network and artificial intelligence.

**Lotto game**

What defines a lotto game.

Columns count in presentation simply means how many columns are used when making table of number for selection or presentation. The columns count of 3 is visible on the main screen numbers table at right part of the form.

On the new round are different values you want to add automatically (on one click).

They will be explained in more details later in this manual.

**System.Pool**, the largest number in the numbers drawn in each round

**System.Drawing**, how many numbers makes one combination

**System.Win**, the smallest number of correct numbers in combination to get reward

This particular system will be described as Loto Srbija, 7/39.

Stonito for Lotto supports unlimited number of systems.
Common functions

Common methods are available through main toolbar controls (and menu).

If the previous round is selected, Previous rounds combo list shows the selected round, Return to current round button is enabled. Winning combination is showing the drawn numbers.

Add to history, append or prepend to a history, adding additional draw.

Previous rounds, scroll down and select to go back to the previous round. List to select from is filled so the newest is at the top of the list.

Return to current, go the to current round. This button is only enabled if the previous combo list is not empty (user has selected previous draw).

Delete selected combination, combination from the list of combinations. Combination entered in current draw are stored on disk, while those entered for history are not saved and will be lost when you switch to current round or to other history round. It is not a glytch rather wanted behaviour, what’s history stays history.

Ordered numbers in combination, may be in two states. When clicked on it toggles. When pressed down the combination list will show every combination with numbers ordered, otherwise in natural order.
For computed combination, natural order is order of preference, meaning the most probable number will be shown first.

Winning combination, for current round, it is only a button, for history rounds it contains the actual combination from history (it was winning combination for that round). In current round pressing this button will allow you to add winning combination to the current round. If you enter it new history round will be added, and combination list will be moved to a newly created history round. All this is done automatically, user just have to enter the combination.

Add manual combination(s), in the natural order of numbers,
meaning numbers are shown in their natural place, with no regards on probability table. You can select single combination, full system (all permutation of selected set of numbers) or wheels (predefined permutation list for a selected set of numbers, shortened full systems)

Add calculated combination(s), numbers are selected from a list that contains numbers sorted in order of their probabilities. The most probable numbers are at the top of the list. Also, autocalculated combinations can be selected.

<table>
<thead>
<tr>
<th>On new round combination(s)</th>
<th>Ctrl+ N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest probability combination</td>
<td>✔️</td>
</tr>
<tr>
<td>Last round probability positions of winning numbers</td>
<td>✔️</td>
</tr>
<tr>
<td>Highest probability combined with closest average value prediction</td>
<td>✔️</td>
</tr>
<tr>
<td>Highest probability combined with closest standard deviation prediction</td>
<td>✔️</td>
</tr>
<tr>
<td>Wheels</td>
<td>✔️</td>
</tr>
<tr>
<td>Full system</td>
<td>✔️</td>
</tr>
<tr>
<td>Single combination</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Neural network for probabiilities, this is the main neural network. It gives a probability table for all numbers in a pool for a selected round. Whenever you select different round, this network is used to recalculate probabilities for that particular round. Probability table results are visible on the numbers table at the right of the main form. You can maintain a list of trained network with different settings. This particular network has ID 1 (N1), was created (or retrained) 2017-05-17, 18 hours 29 minutes.

When program starts up it will use the most recently trained network as a selected. That combination will be checked in the menu. If user clicks on any other network, that network will be automatically selected and all calculation will use that newly selected network.
Neural network for statistics, similar to the main neural network, as an output it gives only standard deviation and average value as a prediction. Also, you can have an unlimited number of trained networks for testing purposes, completely analogous to the main neural network.

When entering manual combination for single, full system, or wheels combination you will use dialog shown on these pictures.

First show the numbers in order of their probabilities, and the second one shows the same dialog with numbers in natural order.
Neural Networks

Most people, even non-programmers, have heard of neural networks. There are many science fiction overtones associated with neural networks. And, like many things, sci-fi writers have created a vast, but somewhat inaccurate, public idea of what neural network is.

Most lay people think of neural networks as a sort of “artificial brain” that power robots or carry on intelligent conversations with human beings. This notion is a closer definition of Artificial Intelligence (AI) than neural networks. AI seeks to create truly intelligent machines.

Neural networks are on small part of AI. At least as they currently exist, they carry out very small, specific tasks. Computer-based neural networks are not general purpose computation devices like the human brain. It is possible that the perception of neural networks is skewed, as the brain itself is a network of neurons, or a neural network.

Neural networks are designed to accomplish one small task. A full application likely uses neural networks to accomplish certain parts of its objectives.

The neural networks accomplish pattern recognition tasking very well. When communicated a pattern, a neural network communicates that pattern back. At the highest level, this is all that a typical neural network does.

As you can see, the neural network is accepting a pattern and returning a pattern. Neural network operate complete synchronously. A neural network will produce an output only when presented with input.

Stonito for Lotto makes use of Encog neural networks. Encog is a machine learning framework for Java and .NET. Initially, Encog was
created to support only neural networks. Later versions of Encog expanded more into general machine learning.

**Neural network structure**

Neural networks are made of layers of similar neurons. At minimum, most neural networks consist of an input layer and output layer. The input pattern is presented to the input layer. Then the output is returned from the output layer. What happens between the input and output layers is a black box.

To make use of the neural network, problem input must be expressed as an array of numbers. Likewise, the problem’s solutions must be an array of numbers. This is the essential and only true value of a neural network. Neural networks do not loop, call subroutines, or perform any of the other tasks associated with traditional programming. Neural networks recognize patterns.

The input layer is the first layer in a neural network. Like all layers, it also contains a specific number of neurons. Typically, the input layer will have one neuron for each attribute that the neural network will use for classification, regression or clustering. In our case, the number of neurons in the input layer is:

\[
\text{Input layer neurons} = \text{Back-Round-Window} \times \text{System.Pool}
\]

The output layer is the final layer in a neural network. This layer provides the output after all previous layers have processed the input. The output is formatted very similarly to the data that was provided to the input layer. The neural network outputs an array of doubles. Number of output layer for neural network to get probabilities of numbers is:

\[
\text{Output layer neurons} = \text{System.Pool}
\]

Other layers may exist between the input and output layers. These hidden layers are simply inserted between them. The hidden layers can also take on more complex structures. The only purpose of hidden layers is to allow the neural network to better produce the expected output for the given input. The hidden layers are very much a “black box”. The problem is defined in terms of the neuron counts for the hidden and output layers. The problem is defined in terms of the neuron counts for the hidden and output layers. How the neural network
produces the correct output is performed in part by hidden layers. Once the structure of the input and output layers is defined, the hidden layer structure that optimally learn the problem must also be defined.

The challenge is to avoid creating a hidden structure that is either too complex or too simple. Too complex of a hidden structure will take a long to train. Too simple of a hidden structure will not learn the problem. A good starting point is a single hidden layer with a number of neurons equal to twice the input layer. Depending on this network’s performance, the hidden layer’s number of neurons is either increased or decreased. Developers often wonder how many hidden layers to use. Some research has indicated that a second hidden layer is rarely of any value. Stonito for Lotto is an excellent way to perform a trial and error for the most optimal hidden layer configuration.

Back-Round-Window defines how many previous rounds are looked when calculating probabilities for a new round. If it is 5 it means that for each draw, relevant inputs are 5 previous draws.

Hidden layers are defined with number of neurons in each, separated by space.

On this form, neural network will have 5*39 neurons in input layer, 39 layers in output layer, and it will have three hidden layers, with 234, 117 and 78 neurons.

Back-Round-Window is a number greater of equal to 1.

Neural network training

When natural network is structured it has to be train.
If you click on Stop training checkbox, the process of training will stop.

**Maximum Epochs** and **Minimal Error** are two criteria for stopping the process on desired values.

**Epochs number** is rising, **Error** is falling with time.

**Update existing unchecked** will make this trained network stored separately (not overwriting currently selected network).

The process of training neural network takes time.

That is why you can have a trained networks saved and used later.

One neural network is used for probabilities of numbers in a current draw, and the other is for statistical values (average and standard deviation).

Both networks are similar in construction and they are set up in the similar way.

And that network you can ask for the last rounds as input, and check results for the round yet to happen.

Nobody knows how the network learns. Nobody knows how it computes its output. But, trained with history rounds and doing them correctly, nobody can tell that he knows better tool to predict numbers in a future draws.

That’s because nobody can predict future.
• There is no optimal number of layers and neurons in each hidden layer for each system.
• The best combination is most often trial and error task.

History draws

Previous draws are very important part of the lotto game.

Stonito for Lotto offers a lot of tool for keeping good record of previous draws history.

The previous draws are defined with YEAR, ROUND, and NUMBERS.

Round is an integer starting with 1 and rising. Round resets to 1 on each new year. Numbers may be entered in any order, separated with commas. The best is to enter them in the natural order (the order in which they were drawn), but for now this is not used in any prediction algorithm (may be added in future).
Adding a new history item uses an intelligent form that enforces restrictions on what is possible to enter to the list of previous draws.

Combination entered is checked for correctness.

Year and Round are auto-calculated, except for the case when history is completely empty, so you can start the history anew.

After that, you must append or prepend the list (direction in history), or you can switch to the new year.

Stonito for Lotto allows you to store history combinations in natural order, as they are drawn from the drawing machine.

There is still no use of that information in the computation, but it may be included in future versions. That’s why the user is advised to enter history draws in natural order as a preferred order.

In each history item a list of combinations played is also included.

This way you can have a very simple to use and maintain remainder of previous results.

Entering winning combination is also a way to add to the history draw. Use the button on the toolbar when you are in a current round, the
This is an example of previous round. In the table to the right the probabilities of numbers according to the current network selected are represented with background colors.

Selected combination in the combination list is marked with darker and bold numbers in the table.

Winning combination is also indicated in the right panel with darker border round winning numbers.

In the combination list correct numbers are indicated with different background color. Wins are shown in the status bar.