

Implementation of SECoS with OKSIMO

Miguel Ayala Arias, Philip Reul

August 18, 2009

Abstract

This paper deals with the implementation of *Simple Evolving Connectionist Systems* with the OKSIMO software. The goal of this implementation is to find out how much OKSIMO still lacks to implement real world, computational-heavy models and how they can be best visualised. The tests and implementation were conducted by Miguel Ayala, the visualisation was programmed by Philip Reul.

'For some reason, my system lacks the basic life processes of either death or the ability to leave behind offspring.'

'Can't you copy yourself?'

'A copy is merely a copy. There's the possibility a single virus could utterly destroy me. A mere copy doesn't offer variety or individuality. To exist, to reach equilibrium, life seeks to multiply and vary constantly, at times giving up its life. Cells continue the process of death and regeneration. Being constantly reborn as they age. And when it comes time to die, all the data it possesses is lost leaving behind only its genes and its offspring.'

(from *Ghost in the Shell* by Mamoru Oshii, 1995. Dialogue between Puppetmaster and Motoko, shortly before they merge.)

Contents

1	Preface	2
2	Modules	3
2.1	List of modules	3
2.2	Important parameters	4
2.3	inputActivationFunction	4
2.4	outputActivationFunction	5
2.5	linearSaturatedFunction	5
2.6	computeErrorValue	5
2.7	hammingDistance	6
2.8	normedEuclideanDistance	6
2.9	addNeuron	6
2.10	addInputNeuron	6
2.11	addOutputNeuron	6
2.12	updateWeights	6
2.13	updateInputWeights	7
2.14	updateOutputWeights	7
3	Suggestions	8
3.1	Functions	8
3.2	Features	8
3.2.1	Dynamical data structures	9
3.2.2	Improving of features	9
3.2.3	Outputs	9

Chapter 1

Preface

This preface starts with a small conclusion of the implementation and test of SECoS with OKSIMO. Although the software is supposed to be generic, during implementation one notices that in reality it is not. This is mostly because of the parameters which have to be defined beforehand.

There is no possibility to examine incoming data, for example to check the size of incoming vectors and compare it to a predefined value. This leads to a form of "programming" that in itself is much more hardcoded than in programming languages.

The only way to overcome this problem is to define large enough input parameters and control the stream of information, through the amount of cycles or other parameters.

The problem in this case is that this will most probably not work well with the graphical tool for simulation, especially if outputs are desired. Still the software handles all required functionality and with a little tweaking working with large values is also possible.

Since this implementation seems to play an important role in validating the OKSIMO modeler and simulator, we already can deduct from the tests. That OKSIMO has all the features to implement generic models, although more features are needed to make it more useful for a broader public.

Chapter 2

Modules

2.1 List of modules

- `hammingDistance`
- `normedEuclideanDistance`
- `addNeuron`
- `addOutputNeuron`
- `addInputNeuron`
- `updateWeights`
- `updateInputWeights`
- `updateOutputWeights`
- `inputActivationFunction`
- `outputActivationFunction`
- `linearSaturatedFunction`
- `computeErrorValue`
- `maxInputActivation`
- `maxOutputActivation`

Helping connectors like *doNothing* etc. are not listed, since these are just connectors, which originally were internal but for practical reasons were converted for repeated use.

2.2 Important parameters

INPUTMATRIX : the presented examples, size: $n \times 784$, must contain input examples

OUTPUTMATRIX : the matrix with the desired outputs, size: $n \times 10$, must contain the desired outputs

SECOS : the ANN to be built, size $n \times 805$, should contain an empty matrix

COUNTER : index of the input matrix, should be zero if the inputs should be presented from the first index on

NEURONCOUNT : the actual amount of evolving layer neurons, must contain the actual size of the net

LEARNING_RATE1 : the rate with which the input neurons are updated, a value between 0-1

LEARNING_RATE2 : the rate with which the output neurons are updated, a value between 0-1

SENSITIVITY_THRESHOLD : basically the most important value of the whole ANN, it defines the similarity between incoming samples and already existing ones, by subtracting geometrical distances, a value between 0-1

ERROR_THRESHOLD : this threshold defines if samples were wrongly recognized during training phase, a value between 0-1

2.3 inputActivationFunction

This module computes the input neuron activation of give SECOS. Input parameters are:

SECOS, *NEURONCOUNT*, *INPUTMATRIX*, *COUNTER*.

The model takes the actual input vector by taking it out of *INPUTMATRIX* with the *getRow* connector. It invokes the connector *computeDistance*, which itself invokes *hammingDistance* and thereby computes the actual input activation. This is done throughout the whole net, the computed values are stored within the net at column 784 of each hidden layer neuron. This column vector then is passed to *maxInputActivation* a simple *max* function, which finds the highest value in the vector. This model returns following parameters:

ACT_COL the column vector with the actual values, for testing and controlling.

MOST_ACTIVATED_NEURON the position of the hidden layer with the highest activation value.

INPUT_ACTIVATION the actual activation value of the most activated neuron.

SECOS.

$$A_j = 1 - D_j$$

2.4 outputActivationFunction

This module computes the output neuron activation of given SECoS. This Module takes as input parameters:

SECOS, *NEURONCOUNT*, *OUTPUTSUM*

The module is a simple linear transfer function that sums the activation of the hidden layer neuron towards the output neurons, these values are stored in *OUTPUTSUM*, which is done in the module *setOutputsum*. This vector then is passed to the *maxOutputActivation*, a simple *max* function, which finds the highest value in the vector. The value is then passed to the system *linearSaturatedFunction*, which is described in [8].

Output parameters of the module are:

OUTPUTNEURON an integer value in the range of 0-9 stating the highest activated output neuron.

OUTPUT_ACTIVATION the actual output activation of the highest activated output neuron.

OUTPUTSUM the vector with the actual values.

$$A_o = \sum_{i=1}^c A_i * W_{o,i}$$

2.5 linearSaturatedFunction

Makes a linear saturation of a given value, where

$$\sigma(x) = \begin{cases} 1, & x \leq 1 \\ x, & 0 \leq x < 1 \\ 0, & x < 0 \end{cases} .$$

. Input and output value is *OUTPUT_ACTIVATION*.

2.6 computeErrorValue

This model computes the output error, it takes as parameters. *OUTPUTNEURON*, *MOST_ACTIVATED_NEURON*, *SECOS* and *OUTPUT_ACTIVATION*, it is a simple operation where the actual output is subtracted from the desired output.

$$E_o = |O_o - A_o|$$

2.7 hammingDistance

Computes the hamming distance between two vectors, division by zero is not handled.

Input variables: *VEC1*(size: 1x784), *VEC2*(size: 1x805), *VAR1*(helper variable), *VAR2*(helper variable)

2.8 normedEuclideanDistance

Computes the normed euclidean distance between two vectors, division by zero is not handled Input variables: *VEC1*(size: 1x784), *VEC2*(size: 1x805), *VAR1*(helper variable), *VAR2*(helper variable)

$$D_n = \frac{\sqrt{\sum_{i=1}^c |I_i - W_{i,n}|^2}}{\sqrt{c}}$$

2.9 addNeuron

Invokes *addInputNeuron*, *addOutputNeuron* and increments *NEURONCOUNT*

Input variables:

INPUTMATRIX, *OUTPUTMATRIX*, *NEURONCOUNT*, *COUNTER*, *SECOS*

2.10 addInputNeuron

Invoked by *addNeuron* takes the passed parameters *INPUTMATRIX*, *NEURONCOUNT*, *COUNTER*, *SECOS* and sets the input weights of *SECOS* to these values.

2.11 addOutputNeuron

Invoked by *addNeuron* takes the passed parameters *OUTPUTMATRIX*, *NEURONCOUNT*, *COUNTER*, *SECOS* and sets the output weights of *SECOS* to these values.

2.12 updateWeights

Invokes *updateInputWeights* and *updateOutputWeights*, see following sections for input parameters. Output is the updated *SECOS*.

2.13 updateInputWeights

This module represents the update function of the input weights, input parameters are:

INPUTMATRIX, *SECOS*, *COUNTER*, *MOST_ACTIVATED_NEURON*, *LEARNING_RATE1*. The module invokes the connector *inputWeightsUpdate*, which actually does the update computation of the input weights. Output parameter is *SECOS* with updated input weights.

$$W_{i,j}(t+1) = W_{i,j}(t) + \eta_1(I_i - W_{i,j}(t))$$

2.14 updateOutputWeights

This module represents the update function of the output weights, input parameters are:

OUTPUTMATRIX, *SECOS*, *OUTPUTNEURON*, *MOST_ACTIVATED_NEURON*, *LEARNING_RATE2*, *ERROR_VALUE*, *ACTIVATION_VALUE*. The actual computation of the updated output weights takes place in the connector *weightsOutputUpdate*. The model return *SECOS* with updated output weights.

$$W_{o,j}(t+1) = W_{o,j}(t) + \eta_2 A_j E_o$$

Chapter 3

Training and Testing

The net was trained with 10 input vector of the MNIST database, they represent hand-written digits with grey-scale values. The output was stored in a file and compared with a SECoS that was trained in scilab, both nets had identical values up to the sixth decimal place. The testing function in OKSIMO revealed the same recognizing ability than the one programmed with scilab. From these results it is concluded that both models are working properly. For a larger test, we will wait for the new version of the simulator.

Chapter 4

Suggestions

4.1 Functions

Useful features for computation with matrices:

- `size()`
- `max()`
- `min()`
- `length()`
- **generally more matrix features**

The first implementation of the SECoS was done with *scilab* and while working with it one notices the great advantages of the concept of working with matrices. The possibility to extract values at any give time is great and helps a lot to manage large amount of data. This is a feature, which can be applied in OKSIMO, but only with work-arounds and static.

Although basically every value at can be reached as well, the amount of additional variables, which have to be passed though the model, becomes very large. That leads to one of the main problems of working with OKSIMO, models can become confusing very fast. Since a lot of simple functions like *max()* have to be programmed by hand, which means conditional statements, loops, connectors etc.

It is clear that OKSIMO for being a real alternative to for example *scilab* needs to be more convenient. But then the advantage of the graphical interface could be the real trump card.

4.2 Features

- **Dynamical data structures**

- **Improving of refactorisation techniques**
- **Improving of debugging & test features**
- **Outputs**

4.2.1 Dynamical data structures

The OKSIMO project team is already working on dynamical data structure, so there is no need to comment anymore on that.

4.2.2 Improving of features

Not every feature was tried out in this test, but it is clear that some need improvement. The debugger sometimes passes out different results than the actual simulation, which doesn't help much.

Most of it could probably be solved if there was a more dynamic way to access values.

4.2.3 Outputs

Output files with access to certain variables in different file formats would be a feature that would be of great convenience as well.

Bibliography

- [1] Kasabov, Nikola *Evolving Connectionist Systems*, Springer London, 2007
- [2] Watts, Michael J. *A Decade of Kasabov's Evolving Connectionist Systems: A Review*, IEEE, Transactions on Systems, Man and Cybernetics -Part C: Applications and Reviews, Vol.39, No.3 May 2009
- [3] Watts, Michael; Kasabov, Nikola *Simple Evolving Connectionist Systems and Experiments on Isolated Phoneme Recognition*, Department of Information Science University of Otago, 2001
- [4] Watts, Michael *Evolving Connectionist Systems- Characterisation, Simplification, Formalisation, Explanation and Optimisation* Thesis 2004 University of Otago, NZ
- [5] Zadeh L.A. *Fuzzy Sets* Information and Control, 8, 1965 p.338-365
- [6] Fuller, Robert *Introduction to Neuro Fuzzy System* Physica Verlag, 2000
- [7] Tanaka, Kazuo *An Introduction To Fuzzy Logic For Practical Applications* Springer, 1997
- [8] Ayala, Miguel *ECoS, a case study* Bachelor thesis, FH Frankfurt 2009

List of Figures