Evolution of Transfer Functions in Artificial Neural Networks

Roland Knall; Elia Husiatynski
rknall@cosy.sbg.ac.at ; etehusia@cosy.sbg.ac.at

Version: 23th of march
Uhrzeit: 11:00am

(LV-Leiter): Univ. Ass. Roland Schwaiger

Abstract

The evolution of transfer algorithms in artificial neural networks is not well documented yet. This Paper shows up the posibilities and capabilities of them, using genetic algorithms for generation and evolution of transferfunctions in artificial neurons.

Chapter 1

Introduction

Artificial Neural Networks are artificial rebuildings or models of a very complex and highly undocumented structure:our brain. In our brain billions of neurons exchange data. This exchange follows a very simple scheme. Inputs are recieved by a neuron on one end, and is sent further as an output on the other ends. But what is happening in between?
There exist computer algorithms for a process called *cellular transfer* but we do know just a few facts about how our braincells do this. Therefore our algorithms are only simple models of these biological structures.[1], or complex mathematical functions such as the sigmoid function: $\varphi(s_j) = \frac{1}{1+exp(-s_j)}$[2] Taking this example you can see, that you get a special range of piutput for a special type of input. These is a standard procedure for *Artificial Neural Networks*. Normally every node of an *Artificial Neural Networks* uses this function or similair, but you do not try to use a different function for every node. This has two reasons

Complex Structure Normally you build *Artificial Neural Networks*s with a given topological strtucture. If you introduce now different functions for every node, network gets even more complex, and problems can not be trained in an efficient way.

Singularity Every node, or even you make it a little bit easier, every subnet should us e diferent nodes which are optimated for their part. But you can not design a net topological if you have to use such a complex structure

So what can you do, to use evolutionary design, but not getting mad of the topological structure. The answer is quite simple, you use evolutionary design, and et the net build itself. You do not have to know the structure of the net, because the net is not your problem. You allow the net to change the node transfer functions as they are needed. Therefore you gain a much mor efficient way for solving problems, because the net controls the structure, and it is not needed to know the structure of the *Artificial Neural Networks* to solve problem with it.
Evolutionary design is already in use. But many *Artificial Neural Networks* - Designs ignore the fact, that transfer functions take influence on the net as much as the net algorithms do.

[1] Imagine a person who installs a new Operating System. When the computer will boot the first time everybody is hoping it will do so without problems, but if it does you will not influence the boot process anymore, normally. But why? This process gains all the information needed for the system and this process is the key for an efficient *Operating System*

[2] This function is continous and varies from 0 to 1 as $-s_j$ varies from $-\infty$ to ∞.

This contains many efficiency problems. In this paper we want to document this process for gain of information. How they are working now, and how they can be improved. The goal of this is not to improve them, but to understand the processes better, and to understand the part they play in *Artificial Neural Networks*. We also will show how different types of *Artificial Neural Networks* use different types of solutions for this, and how they can be reused in the design of a fully textitEvolutional Neural Network.

Chapter 2

Different Types of the same thing

2.1 DNNs, LNNs and GNNs

2.1.1 Distributed Neural Networks

This type of *Artificial Neural Networks* - Design, where the multilayer perceptron is the most common in use, uses sigmoid functions $\varphi(s_j) = \frac{1}{1+exp(-s_j)}$[1]

[1]This function is continous and varies from 0 to 1 as $-s_j$ varies from $-\infty$ to ∞.

Appendix A

Protokolle

1st Meeting

Date: 10th March 1999

Place: Institute for ComputerSciencees Room 2.xx (Room Univ.Ass. Schwaiger)

Persons: Elia Husiatynski, Roland Knall, Roland Schwaiger

Discussion: We discussed the structure of this paper, what it should contain, which parts should be included, ... Also we defined a modus operandi for future project work. We defined the aims and talked about *SNNS*[1]. Our assisting project controller explained that *SNNS* is already used by the institute. Finally we agreed in creating a protocoll and the first parts of this document.

2nd Meeting

Date: 17th March 1999

Place: Institute for ComputerScience Room 2.xx (Room Univ.Ass. Schwaiger)

Persons: Elia Husiatynski, Roland Knall, Roland Schwaiger

Discussion: We did not complete our work for the week before, therefore we had almost nothing to discuss. We talked about a paper that already has been done on *transfer functions*, but it turned out that our aims will be more detailed. Finally we declared a date for the next meeting. During the week, we will summarize the paper, and learn more about the structure of *Artificial Neural Networks*s and transferfunctions.

3rd Meeting

Date: 24th March 1999

Place: Institute for ComputerScience Room 2.xx (Room Univ.Ass. Schwaiger)

Persons: Elia Husiatynski, Roland Knall, Roland Schwaiger

Discussion:

Bibliography

[1] U. Stuttgart, "Stuttgarter neuronale netze simulator," *???*, 1995.