Autonomous Neural Development and Pruning

André Christoffer Andersen
Department of Bioenginering
University of California San Diego
La Jolla, CA 92122
andre@andersen.im

Abstract

The main motivation behind this project comes from the observation that topological structures of neural networks intended for engineering purposes often seem to be chosen more or less arbitrarily or with a rule of thumb. This project seeks to explore a biological inspired method for autonomous development and pruning of non-noisy back-propagating neural networks. Learning will be divided in to three distinct phases development, pruning and apoptosis where the network will initially overexpand then contract to a more computationally tractable valency and topology. Thus, the actual computational efficiency of learning will be neglected in favor of ease of implementation and computational efficiency in actual usage.

1 Introduction

A reoccurring problem when implementing useful learning or specifically back-propagating neural networks is choosing a suitable topological structure when solving classification problems for engineering purposes. The problem is to find a balance between a network which is large enough to yield results in a reasonable time and avoid local minima [1], while being small enough to avoid overfitting the training data [2].

A typical enginering statment could be for example ”if the learning gets stuck, you can try with different number of neurons, layers or learning parameters” [3]. Not only is this proposed method of trial-and-error an unsatisfying recommendation it underpins the whole problem. Engineers have a limited set of tools at their disposal for choosing neural network topologies. Much of the literature tries to answer this problem by rules of thumb which try to give some heuristic inkling to what size to choose.

> ”A rule of thumb is for the size of this [hidden] layer to be somewhere between the input layer size ... and the output layer size ...” [4].

> ”How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer.” [5].

Again these answers are unsatisfying and also tend to lean towards trail-and-error, but with some sort of boundary limit. In an effort to illuminate, if not alleviate, this problem I proposed to borrow the main principles of brain development in biological systems [6].

- Intense early neural development
- Continuous gradual synaptic pruning
- Eventual neural apoptosis

These biological principles will be used at their basic level in an effort to automate the generation of a topological structure that is straight forward to make yet small and computationally light to use.

In broad strokes the proposed method works as follows: The network starts out small and then undergoes an early exponential (or rather logistical) neural and synaptic growth while learning through back-propagation until desired level of precision is reached. Then the network enters a phase of a synaptic pruning that extends the concept of Optimal Brain Damage by Le Cun, Denker and Solla [7]. This pruning phase alternates between removing synapses (or connections) and relearning in order to regain lost classification precision until it isn't able to relearn at a reasonable speed or within a reasonable time. As most connections are pruned away there may be neurons that now have lost all their feed forward connections and may thus safely be removed. This neural apoptosis phase is merely a cleanup stage that removes surplus neurons. However, in a future version of this method this apoptosis phase may implement a more heuristic method that removes neurons that are close to having no effect rather than absolutely not having any effect.

2 Methods

2.1 Framework

The framework chosen to implement the method in, and generate results from, is Java Neural Network Framework Neuroph [8]. Even though its not the fastest frameworks around it makes up for this by being object oriented. This makes it especially suitable to work with since network topology can be directly managed in contrast to matrix based frameworks. Also, the rich API has several layers of abstractions with many predefined network, neuron and learning models.

2.2 Model

Since this project looks at enginering challenges with artificial neural networks it may come as no surprise that the chosen models will reflect this. The unacquainted or thorough reader is recommended to review "Multilayer perceptrons" by Honkela A. [9] for mathematical details on the models. This paper will focus on the topological properties and will treat the following models as abstractions.

2.2.1 Neuron and network

The neural network uses a feed forward multilayer perceptron network with one hidden layer - for the moment. Its basic unit is the non-spiking perceptron with a non-linear sigmoid activation function and bias neuron(s). It is non-spiking in the sense that its activation function outputs a value that roughly mimics the activity frequency of a spiking neuron rather than the spikes themselves.

The number of neurons in the input and output layers will be held fixed throughout development and learning. However, that is not to say that their connections aren't affected.

2.2.2 Learning

This multilayer network implements the back-propagation learning algorithm [10]. Even though not very biological founded it has properties that will be useful for this project's purposes in addtion to being the de facto standard within practical neural network implementations. Furthermore, as with any heuristic gradient decent learning algorithm the back-propagation algorithm is pruned to end up in a local optimum. This is especially true for small networks. We will later see that the proposed method has a tendency to resolve this automatically.

2.3 Benchmark testing

The developed networks will be benchmarked against the best network a brute force trail-and-error method can come up with by systematically searches for the smallest and fastest network it can find. This of course is quite intractable and will have to be done at a monumental computation cost. In order to alleviate this problem and greatly reduce learning time we will have to focus on small networks with non-noisy training sets.

The developed network should do worse in most benchmark tests compared to the optimal network found by trail-and-error. This is because we are only comparing the single best result from all the results the trail-and-error method generates with our stand alone developed network.

2.3.1 Representative classification problem

The benchmark classification problem which will be used is an optical character recognition problem. The input is a non-noisy 12x8 bitmap representation of the digits between 0 and 9. The total number of input neurons is thus 96 with 96^n connections to a hidden layer with n neurons.

The output is naturally the same numerical integers from 0 to 9 yielding a total of 10 output neurons. For benchmark purposes a classification is said to be correct if the argmax-function of all output neurons is labeled the same as the input. However, even if the network is able to classify correctly under this definition it is not finished learning until it has a total network error of less than 0.01.

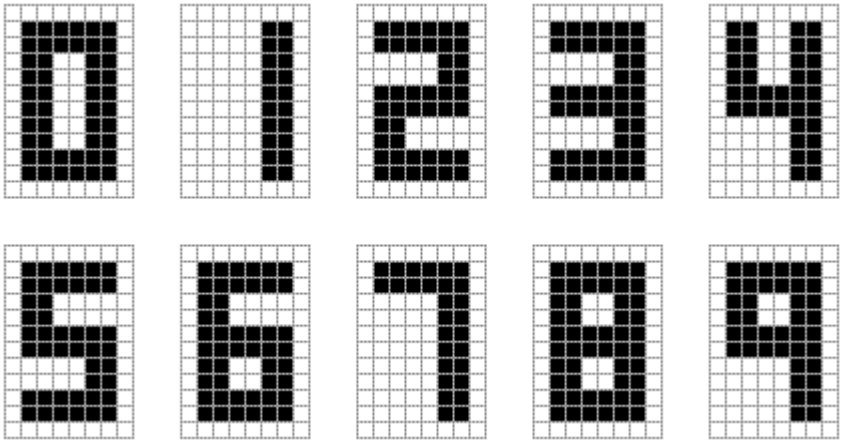

Figure 1: Training set with 12x8 bitmaps of digits between 0 to 9.

2.3.2 Measurements

There are many ways to measure a neural network's performance. For this project's intents we should qualitatively look at how fast the method teaches and develops the network. This is done to make sure that the development method does not learn at an intractable rate. However, beating the time to maximum error for a pure back-propagating network with an already optimal topology is not a goal in and of itself. The important focus should be on being within reasonable range of this lower bound. When this performance benchmark is done we should quantitatively look at processor and memory usage when the network is used live. This will be a measurement of whether or not the pruning and neural apoptosis works within desired levels.

2.4 Three phases of learning

In this subsection a more detailed description of the proposed method will be flesh out.

2.4.1 Development

The development phase is one continuous learning phase periodically interrupted by injection of neurons by some trigger mechanism.

Adding a neuron When a neuron is added to the network it is fully connected to the presiding (input) layer and the following (output) layer. The weights of the connections are uniformly chosen by a pseudo random number generator. This introduces a small error to the network which has the fortunate effect of working towards dislodging the network from local optima. Of course the actual increase in network size is the main reason why dislodging happens, but the error introduction helps.

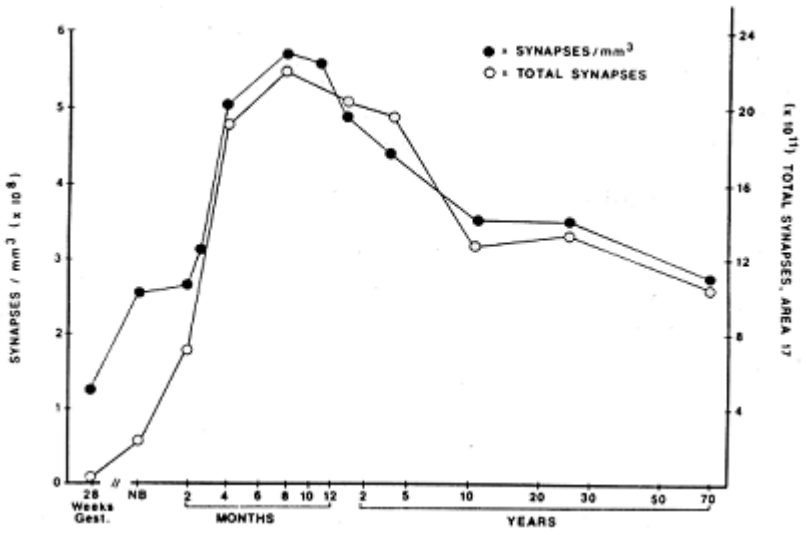

Figure 2: Example of biological synaptic development by Huttenlocher.

Triggering neuron addition A central problem to neural development is to choose some sort of heuristic trigger mechanism that goes off when it is time to add a new neuron to the hidden layer. In biological neural development the brain uses neurotrophic factors that signal cell survival, growth and death (11). Furthermore, the Hayflick limit of cell division (12) suggests, in general terms, a mechanism that displays growth reduction over time.

The trigger mechanism used here revolves around a cool down timer that restrains development for some increasing period of time or number of iterations depending on preference. The trigger mechanism can be describes as follows

$$T_{i+1} = \gamma T_i + K_i$$

Where T_i is the cool down time right before addition of neuron i, $\gamma \geq 1$ is a cool down factor determining the increase in cool down between each triggering, and K_i is an additional time introduced by some other desired mechanism, though, for our purposes we can let $K_i = 0$.

2.4.2 Pruning

As development yields an acceptable network size we will already have a fully functioning solution to the classification problem. The problem is that many of the neurons and connections that are established do the work that fewer could have done just as well. This sets the stage for pruning of the network. The pruning phase works by removing the weakest connections then training away the resulting increase in error. It keeps removing and retraining until it fails to reach a predetermined lower error threshold within a period of time or progress is below a certain error reduction rate. When retraining fails it reverts the last iteration of connection removal and retraining and returns the network. A more succinct pseudocode of the pruning algorithm is as follows

```
do {
    save copy of network
    find and delete lowest weighted connection of network
    do one epoch of learning and calculate new network error
    if (new network error is above lower threshold){
        relearn lost knowledge
    }
} while (network relearning converged in time)
return saved copy of network
```

2.4.3 Neural apoptosis

As most connections are pruned away there may be neurons that now have lost all their feed forward connections thereby having no effect on the network. All connections to these neurons and the neuron itself can safely be removed. In effect this neural apoptosis phase is merely a cleanup stage that removes surplus neurons. However, in a future version of this method this apoptosis phase may implement a more heuristic method that removes neurons that are close to having no effect rather than absolutely not having any effect.

2.5 Shortcomings

The main issues with the described method are space requirements of the pruning algorithm and how to determine what cool down factor and initial condition to use. Additionally, the trigger mechanism could be augmented to allow some other intrinsic way to trigger a neuron injection. An example could be to use the cool down factor as a lower bound that helps some other main mechanism, defined by K_i, to not trigger too often.

3 Results

3.1 Trail-and-error non-developing benchmark

Brute force trail-and-error with a cut-off time of 10 seconds and a maximum network error of 0.01 yields the results in figure 3.

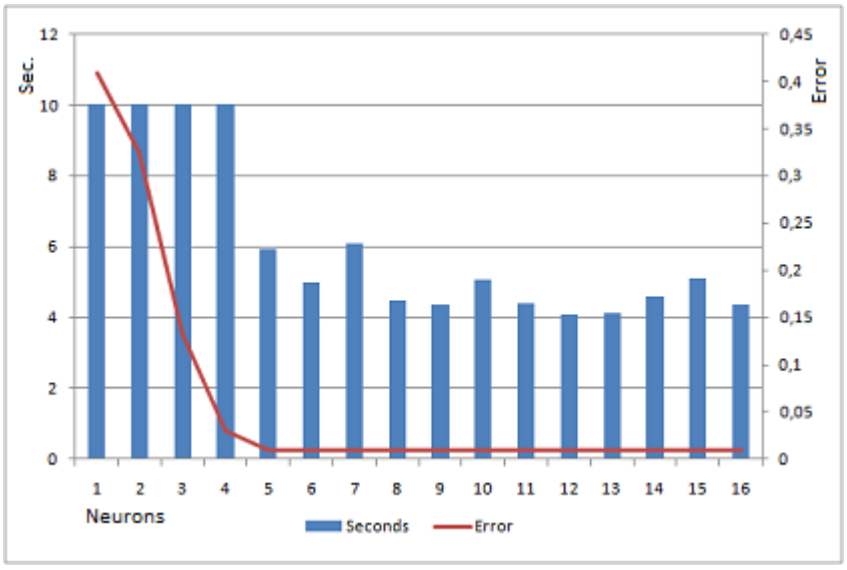

Figure 3: Trail-and-error brute force search for optimal hidden neuron count.

Trails with up to 4 hidden neurons fail to complete in time while 5 and more hidden neurons all complete in time. There seems to be only a marginal learning speed benefit of having more neurons. When benchmarking the developed network the fairest network to compare it with is the smallest network that actually achieved the network threshold error of less than 0.01. Here that is five hidden neurons.

3.2 Periodic triggering

Studying a constant triggering mechanism where $\gamma = 1$ with 1000 ms interval can be useful in order to get a better grasp on the dynamics of adding neurons. Note that this is the only instance where we will use time units in the trigger mechanism. The trigger model is thus

$$T_{i+1} = T_i, \;\; T_1 = 1000ms$$

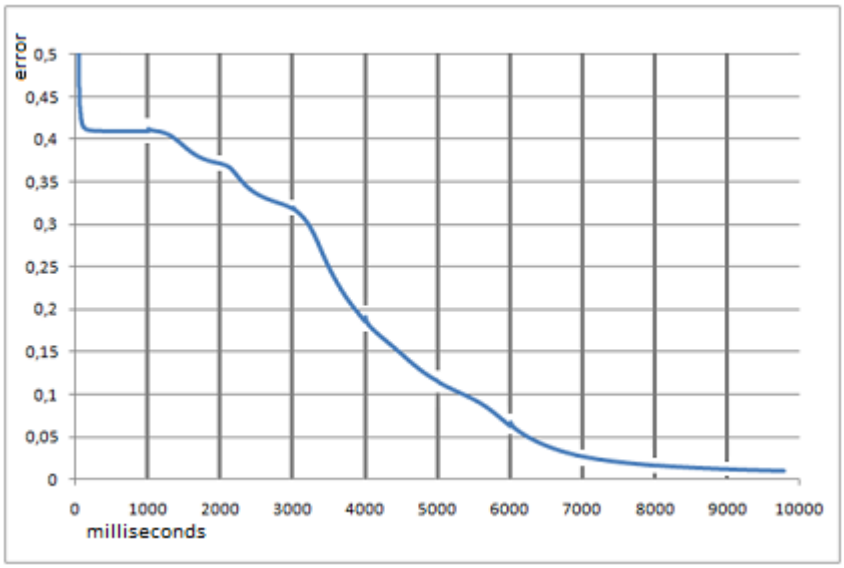

Figure 4: Periodic triggering of neuron injection.

What we can see is an instant drop to a local minimum with a network error of 0.42. When the first neuron is added after 1000 ms we also see a slight increase in error before the local minimum is shrugged off and we proceed down to the next local minimum at around 0.375. This tends to repeat until desired maximum error of 0.01 is reached just before 10,000 ms.

Another thing we can notice is that the first few neurons tend to come too seldom and the last neurons tend to be unnecessary. We can thus already se the usefulness of introducing most of the neurons early as inspired by biological neural development.

3.3 Cool down triggering

The next configuration of the trigger model is iteration based with $\gamma = 4$ and $T_1 = 1$ yielding triggering at following iterations

$$T_{i+1} = T_i = (\gamma + 1)^{i-1} = 5^{i-1} = \{1, 5, 25, 125, 625, 3125, 15625\}$$

Now implementing these in a test run yield the results in figure 5. Note that the graph measures error reduction versus time and not iterations. So, even if there is a loose correlation between the iterations found analytically the time used and the iterations used will not correlate over time due to injection of more and more neurons will slow down the iteration calculations.

This test run gives typical results for the trigger model and method we are using. Qualitatively we can see that the initial convergence towards local optima is avoided due to early neuron injection. Furthermore, unnecessary or even counterproductive injections of neurons at the end are removed. A new neuron was scheduled for iteration 15625, however, the network converged below the maximum network error of 0.01 before this happened.

Now comparing this with normal learning with three, four and five neurons and superimpose the resulting error graphs on our developed network we get the results from figure 6.

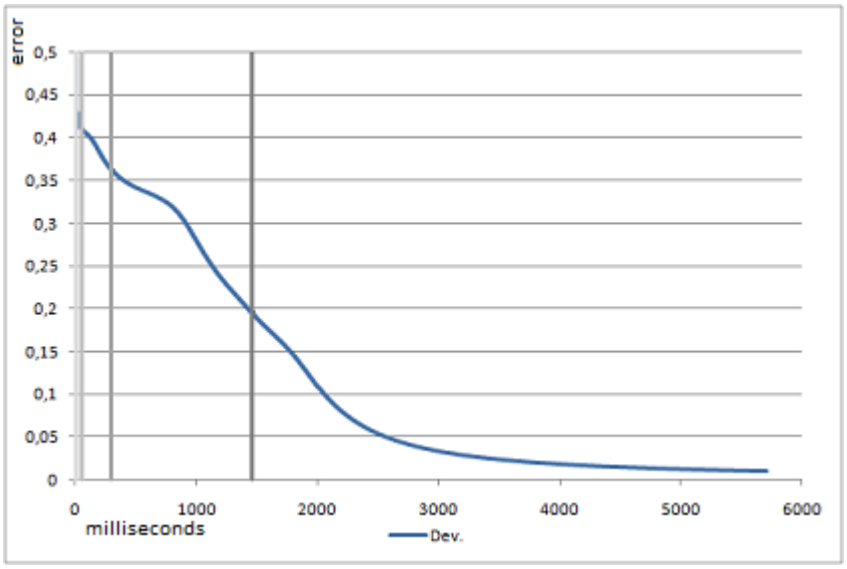

Figure 5: Periodic triggering of neuron injection.

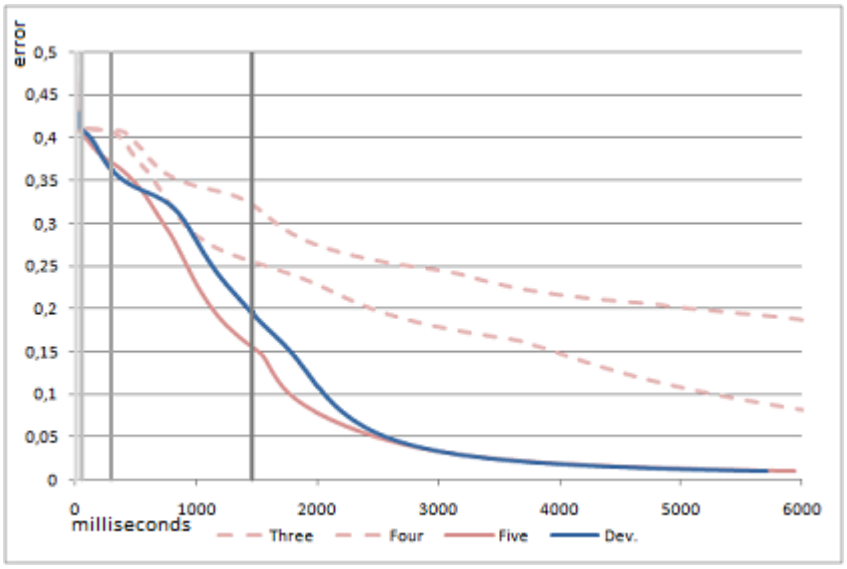

Figure 6: Cool down triggered injection of neuron.

We can see that the developed network is quite competitive with regards to how fast it converge below maximum error. When the fifth and last neuron is added to the developed network it converges in unison with the five neuron network - a satisfying result.

3.4 Pruning

As explained before the next step is the pruning phase. Pruning tends to dramatically reduce the number of connection with up 90% for this training set. When that is said these extreme results may be due to the fact that there is no noise introduced to training set.

Figure 7 shows buildup and subsequent removal of connections over iterations (not time).

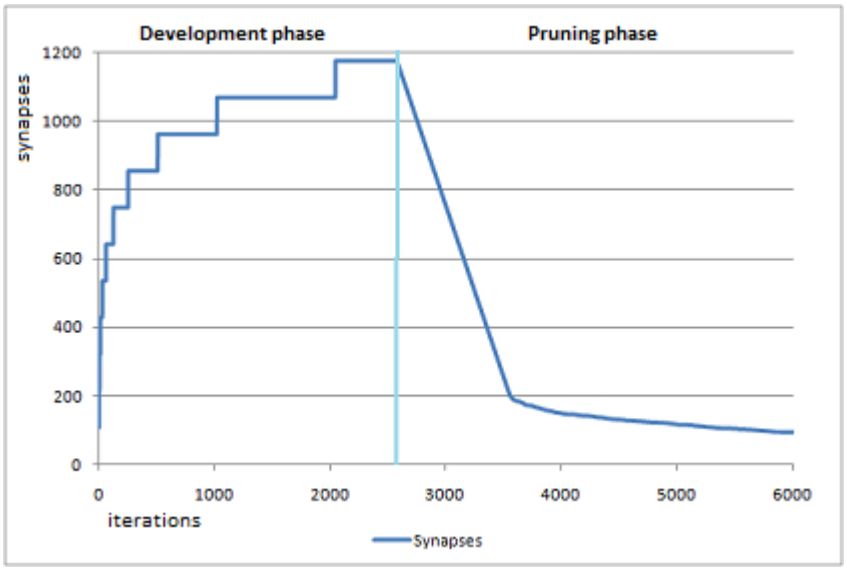

Figure 7: Cool down triggered injection of neuron vs. trail-and-error results.

This graph shows noticeable similarities with the biological synaptic growth and pruning from figure 2. The first thousand or so iterations after entering the pruning phase we can see that the network shows a dramatic reduction in synapse count. In fact, the linear reduction comes from the fact that the removal of the synapses has a very low or no effect on the network error. One synapse is removed every iteration. This might not be so surprising when we take a closer look at the training set. From figure 1 we can see that the entire border and two 2x2 areas are always white, while tree 2x2 areas are always black. That yields 56 input neurons that are useless in classifying the numbers. The connections from these input neurons are no dubitably among the first to go. However, when the network has below 200 synapses left we can notice that the pruning process is going slower. This is because it is starting to chew away on the main meat of the network and it needs to invest an increasing amount of iterations on relearning the lost classification precision. This continues until less than 70 connections are left and the relearning process is taking an unreasonable amount of time or it is stuck in a local optimum just above the maximum network error.

3.5 Neural apoptosis

Neural apoptosis will undoubtedly be important in larger scale networks, but due to the small size of the network we are working with it fails to remove more than one or two neurons from a network size of less than 10 hidden neurons. Thus the effects on these 5-hidden-neuron networks are none or negligible.

3.6 Benchmark results

The best trail-and-error network without development resulted in a 5 hidden neurons with a total of 535 connections. The developed network which used a cool down factor of $\gamma = 4$ and $T_1 = 1$ resulted in 6 (sometimes 5) hidden neurons with under 70 connections. In terms of space this an order of magnitude better.

When the 5-hidden-neuron trail-and-error network was stress tested it performed at under 10,000 classifications per second versus the developed 6-hidden-neuron network which did over 25,000 classifications per second.

4 Conclusion

We have seen how even loosely biologically inspired systems can give rise to automated heuristics that are better than rules of thumb and trail-and-error. Of course the described method for neural development has some issues, but in broad strokes it illustrates the further need for development of topological engineering practices.

The most promising result has been showing that biologically inspired methods for automated network development can performs equal or better than normal, but optimal networks. The method, however, still needs to show promise with regards to larger and noisy training sets, which were not tested here.

Acknowledgments

I wish to thank Gert Cauwenberghs and Jeffrey Bush for an enlightening course in neurodynamics.

References

[1] Kristemsen, T. (1997), "Nevrale nettverk, fuzzy logikk og genetiske algoritmer", p. 67

[2] Weigend, A. (1994), "On overfitting and the effective number of hidden units," Proceedings of the 1993 Connectionist Models Summer School, p. 335-342.

[3] "Image recognition with neural networks howto", http://neuroph.sourceforge.net/image_recognition.html

[4] Blum, A. (1992), "Neural Networks in C++", p. 60

[5] Linoff G., Berry M. (1997), "Data Mining Techniques", p. 323

[6] Stiles, J. (2006), "Brain Development", A video lecture in the "Grey Matters From Mulecules to Mind" series from UCSD/UCTV, show ID: 11188, time 15:00-20:07.

[7] Le Cun, Denker and Solla (1990), "Optimal Brain Damage"

[8] "Java Neural Network Framework Neuroph", http://neuroph.sourceforge.net/

[9] Honkela A., http://users.ics.tkk.fi/ahonkela/dippa/node41.html

[10] Kristemsen, T. (1997), "Nevrale nettverk, fuzzy logikk og genetiske algoritmer", p. 57-71

[11] Fiore M, Chaldakov GN, Aloe L (2009). "Nerve growth factor as a signaling molecule for nerve cells and also for the neuroendocrine-immune systems", p. 13345.

[12] Hayflick L, Moorhead PS (1961). "The serial cultivation of human diploid cell strains", p. 585621.