Learn Trajectories - A Systems Point of View Visualizing Method

L. Larsson
Computer Science Department, University of Hamburg
Vogt-Koelln-Str. 30, D-22527 Hamburg, Germany
larsson@informatik.uni-hamburg.de

ABSTRACT

The implementation of complex information processing systems requires different methods for validation on different levels of abstraction. Direct observation of a set of system parameters is necessary to detect possible implementation faults and for validation. On the uppermost level of abstraction, the observation of the behavior of a whole system is a building block of methods for the validation of an implementation.

The goal of realization a technical system is to build a complex structure based on simpler components of well known behavior which interact among themselves. It depends on the coupling between those components, how complex the behavior of the resulting system becomes. The presented method of learn trajectories, introduced in the context of backpropagation neural networks, visualizes the behavior of multiple parameters for the systems point of view. This contribution illustrates the functionality and application of learn trajectories by different examples.

Keywords: Learn Trajectories, Visualizing Method, Backpropagation Neural Network.

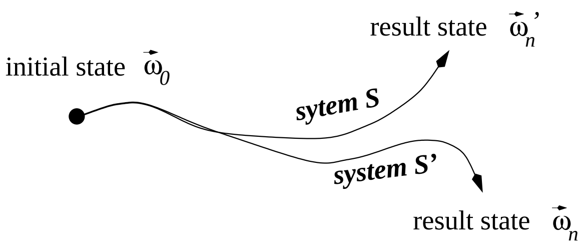

Figure 1: *Varieties in behavior of two different implementations S and S' of a system.*

1. INTRODUCTION

Information processing systems dealing with many parameters simultaneously are hard to debug and to validate. Various implementations of an information processing system on different platforms such as software and hardware in general cause different behavior. So, methods are required to decide if different implementations of on information processing system satisfy the desired behavior. With a state vector, composed of parameters characterizing the system, a sequence of those state vectors represents the behavior in a mathematical sense. Varieties in implementations become observable by divergences of state vectors (fig. 1).

Implementations of multiple time scale systems are hard to debug and are hard to validate. There are different applications of multiple time scale in the area of digital signal processing and of neural networks. Digital processing of analog video signals and the training of backpropagation neural networks [7] are examples for multiple time scale.

In both cases, the observation of system behavior is required for validation. To decide, if a digital video signal processing unit works correct, it is necessary to watch the result of hundred thousands of one by one processed discrete video signal samples which form a complete image or a sequence of images. To estimate results of digital video signal processing unit simulations from the systems point of view, a bidirectional interface for analog video signals has been developed [3].

To decide, if an implementation of a backpropagation neural network performs training in the right way, it is necessary to trace the training process. The established method to do this is plotting the total error with respect to a set of training patterns while training. The method of learn trajectories, first introduced in [4] visualizes the training process close to the high dimensional character of training backpropagation neural networks from a systems point of view.

 Published in Proc. of Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and the Fifth International Conference on Information Systems Analysis and Synthesis (ISAS'99), Orlando, Florida, USA, 31. July - 4. August 1999, in M. Torres, B. Sanchez, D. G. Lainiotis (Eds.), Proc. Vol. 6 - Image, Acoustic, Speech, and Signal Processing, pp. 97-104, International Institute of Informatics and Systemics (IIIS), Skokie, Illinois, USA, 1999.

A backpropagation neural network can be seen as a system of many parameters being modified step by step while training. Each training step is characterized by its own set of parameters - the synaptic weights of the concerning backpropagation neural network. Each set of parameters can be interpreted as a point in a high dimensional space. With this, training can be interpreted as forcing motion of a point in this high dimensional space performing a sequence of points in the high dimensional space.

Training a backpropagation neural network is a local search algorithm – a hill claiming algorithm. The modification of the synaptic weights appears in direction of the negative gradient of the total error which is usually traced while training. The training process may be slowed down, if the gradient is of low magnitude in flat regions (plateaus). On the other hand, the training process may be disturbed by local extrema. In worst case, training may terminate before it reaches the desired optimum.

Learn trajectories visualize the modification process of the synaptic weights of a backpropagation neural network during training within two dimensions by mapping a sequence of l dimensional state vectors to two dimensions.

The learn trajectories may be helpful for imagination of the high dimensional shape of the hill the backpropagation algorithm climbs while training – no matter if a network is of small size with just a couple of synaptic weighs or if a network is large with hundred thousands of synaptic weights, which has been investigated. This is in contrast to other methods for visualizing methods for high dimensional processes, which keep almost all parameters fixed and then varying one or two parameters only.

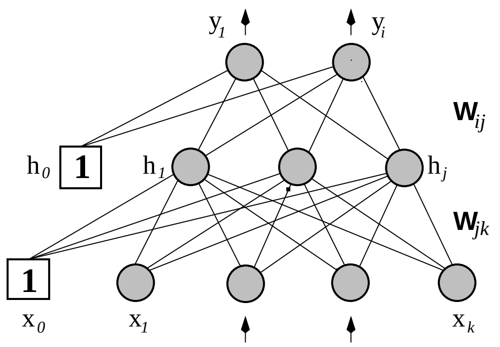

Figure 2: *Backpropagation Neural Network with k inputs, j hidden neurons, and i output neurons.*

Additionally, the visualization method is easy to implement. It just visualizes the behavior of a complex system in a simple but direct way from the systems point of view. No matter about the backpropagation network implementation details, if it is possible to dump the whole set of synaptic weights of a network implementation for each training step, one can visualize the networks behavior while training by learn trajectories. Learn trajectories, introduced to visualize the backpropagation neural network learning process, may also be a helpful method to visualize the behavior of other algorithms and systems dealing with many parameters simultaneously. The method may be suitable for investigations in such areas where different implementations of an information processing systems exist in a way that these are functional similar but not equivalent.

2. BACKPROPAGATION

This section summarizes the backpropagation neural network model, introduced by D. E. Rumelhart, J. L. McClelland, G. E. Hinton and R. J. Williams [5, 6, 7]. A backpropagation neural network corresponding to figure 2 with one output layer and one hidden layer. Each I output neuron y_i of a net computes

$$y_i = g(\theta_i + \sum_{j=1}^{J} \omega_{ij} \cdot h_j). \tag{1}$$

Each J hidden neuron h_j of a net computes

$$h_j = g(\theta_j + \sum_{k=1}^{K} \omega_{jk} \cdot x_k). \tag{2}$$

over its K inputs x_k. In case of *bias*, expression Eq. (1) is modified by $\omega_{i0} := \theta_i$ with $h_0 := 1$ for $j = 0$ and expression Eq. (2) is modified by $\omega_{j0} := \theta_j$ with $x_0 := 1$ for $k = 0$. A backpropagation net with K inputs, J hidden and I output neurons maps a pattern $\vec{x}$ to a classification

$$\vec{y} = F(\Omega_{ij}, \Omega_{jk}, \vec{x}), \tag{3}$$

where Ω_{ij} and Ω_{jk} are synaptic weight matrices. For compatibility with matrix algebra conventions, vectors $\vec{v}$ have to be expanded by one dimension v_0, to hold the bias $v_0 := 1$. The synaptic weight matrices Ω have to be expanded by one column each. These additional columns hold the threshold vectors $\vec{\Omega}_0$. With this, the feed forward mapping Eq. (3) is

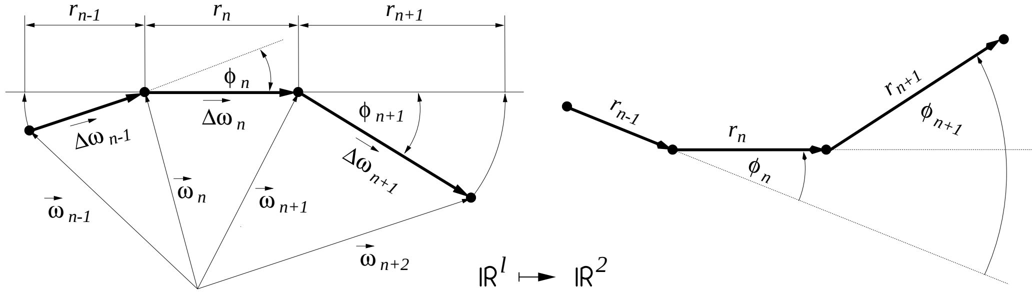

Figure 3: *Construction of Learn Trajectories*

$$\begin{aligned} \vec{h} = \Omega_{jk} \cdot \vec{x}, \quad \vec{\gamma} = g(\vec{h}), \\ \vec{o} = \Omega_{ij} \cdot \vec{\gamma}, \quad \vec{y} = g(\vec{o}). \end{aligned} \tag{4}$$

The vectors $\vec{x}$, $\vec{h}$, $\vec{\gamma}$, $\vec{o}$ and $\vec{y}$ here are column vectors. The non-linear sigmoid function and its derivation are

$$\begin{aligned} g(a) &= 1/(1+e^{-a/\tau}) \\ g'(a) &= g(a) \cdot g(-a)/\tau. \end{aligned} \tag{5}$$

The so called temperature parameter τ is of typical range $0.1 < \tau < 10$. Training patterns are pairs $(\vec{x};\vec{t})_M$ of input vectors $\vec{x}$ and target vectors $\vec{t}$. Before training, the synaptic weights have to be initialized randomly – typically within a range ± 0.5. Then the training is done with the patterns $(\vec{x};\vec{t})_M$ until an error

$$E = \frac{1}{2} \sum_M (\vec{y} - \vec{t})^2_M \tag{6}$$

reaches a minimum or a defined limit by iterative computing

$$\begin{aligned} \vec{d} = \vec{y} - \vec{t}, \qquad & \vec{\delta} = g'(\vec{o}), \\ \vec{e} = {}^t\,\Omega_{ij} \cdot (\vec{d} \otimes \vec{\delta}) \quad & \vec{\epsilon} = g'(\vec{e}), \end{aligned} \tag{7}$$

for each pattern indexed by M. The matrix ${}^t\Omega_{ij}$ is the transposed matrix Ω_{ij}. For matrix notation of backpropagation training it is necessary to define a component like vector-vector product operator $\otimes$ with

$$\vec{c} = \vec{a} \otimes \vec{b} \Leftrightarrow c_i = a_i \cdot b_i. \tag{8}$$

whereat a_i, b_i, and c_i denote the components of the corresponding vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$. During training, the update of the matrices

$$\begin{aligned} \Omega_{ij} &:= \Omega_{ij} - \eta \cdot \vec{\delta} \cdot {}^t\,\vec{\gamma} &+ \quad \alpha \cdot \Delta\Omega_{ij} \\ \Omega_{jk} &:= \Omega_{jk} - \eta \cdot \vec{\epsilon} \cdot {}^t\,\vec{x} &+ \quad \alpha \cdot \Delta\Omega_{jk} \end{aligned} \tag{9}$$

appears in direction of the negative gradient of the error Eq. (6). The parameter η is the so called learning rate of typical range $0.001 < \eta < 0.5$. The parameter α is the so called momentum parameter of typical range $0.1 < \alpha < 0.9$. The momentum parameter α avoids oscillations of synaptic weights during training. A *learn step* here denotes the modification Eq. (9) for all synaptic weights of the backpropagation neural network for one training pattern $(\vec{x};\vec{t})_M$. A *learn cycle* here denotes a set of M learn steps. There exist different strategies presenting for the training patterns and for updating the synaptic weights [1, 2, 8].

3. LEARN TRAJECTORIES

The state Ω_n of a backpropagation neural network can be specified by the set $\vec{\omega}_n$ of it's synaptic weights. Each learn step during training changes the state of a network. So, training results in a sequence of states $\Omega_n, \Omega_{n+1}, \Omega_{n+2}, \ldots$ representing a sequence of synaptic weight sets. So, a set of synaptic weights written as a vector

$$\begin{aligned} \vec{\omega}_n &= (\underbrace{\omega_{10}, \ldots, \omega_{i0}}_{\approx\theta_i}, \underbrace{\omega_{11}, \ldots, \omega_{ij}}_{\Omega_{ij}}, \\ & \quad \underbrace{\omega_{10}, \ldots, \omega_{j0}}_{\approx\theta_j}, \underbrace{\omega_{11}, \ldots, \omega_{jk}}_{\Omega_{jk}})_n \\ &= (w_1, \ldots, w_l)_n \end{aligned} \tag{10}$$

of $l = (i \cdot j + i) + (j \cdot k + j)$ components represents a state Ω_n of the backpropagation neural network, where i is the number of inputs, j is the number of hidden neurons, and k is the number of output neurons. For state difference vectors

$$\Delta\vec{\omega}_n = \vec{\omega}_{n+1} - \vec{\omega}_n, \tag{11}$$

the euclidic distance r_n between two state vectors $\vec{\omega}_n$ and $\vec{\omega}_{n+1}$ is

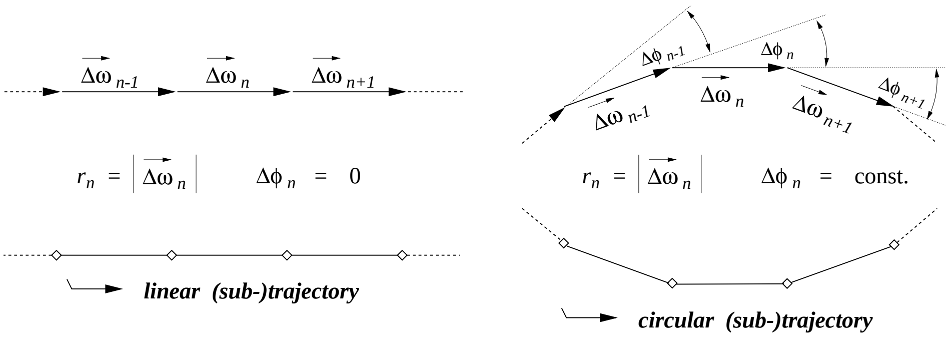

Figure 4: *Linear and Circular Sub-Trajectories*

$$r_n = |\Delta\vec{\omega}_n|. \tag{12}$$

The angle ϕ_n between two state difference vectors $\Delta\vec{\omega}_n$ and $\Delta\vec{\omega}_{n+1}$ is

$$\phi_n = \arccos\left(\frac{\Delta\vec{\omega}_{n+1} \cdot \Delta\vec{\omega}_n}{|\Delta\vec{\omega}_{n+1}| \cdot |\Delta\vec{\omega}_n|}\right). \tag{13}$$

With this, a sequence of states Ω_n during training can be mapped to a sequence of distances r_n and angle ϕ_n. These r_n and ϕ_n represent the high dimensional state changes within two dimensions. For plotting a learn trajectory (see fig. 3) an accumulated angle

$$\Phi_n = \sum_{i=0}^{n} \phi_i \tag{14}$$

is used instead of ϕ_n. Transformed from polar coordinates to cartesian coordinates one gets

$$\vec{x}_n = (x_1, x_2)_n = (r_n \cdot \cos(\Phi_n), r_n \cdot \sin(\Phi_n)). \tag{15}$$

A sequence of n training steps

$$f_n = \{\vec{\omega}_0, \vec{\omega}_1, \ldots, \vec{\omega}_{n-1}, \vec{\omega}_n\} \tag{16}$$

gives n base points f_n of a learn trajectory in $\mathbb{R}^l$. These l dimensional base points f_n are plotted as a two dimensional sequence of points

$$p_n \quad = \quad \{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{n-1}, \vec{p}_n\} \tag{17}$$

with

$$\begin{cases} \vec{p}_0 & = & (0,0)_0 \\ \vec{p}_n & = & \vec{p}_{n-1} + \vec{x}_n \quad \text{for} \quad n > 0. \end{cases} \tag{18}$$

Chained, this two dimensional sequence of points visualizes the shape of a l dimensional learn trajectory in $\mathbb{R}^l$ on the high dimensional error surface Eq. (6) within $\mathbb{R}^2$. It may also by useful to plot only each kth point as a sequence $\vec{p_{0\cdot k}}, \vec{p_{1\cdot k}}, \vec{p_{2\cdot k}}, \ldots$ instead of plotting any point in a sequence $\vec{p_0}, \vec{p_1}, \vec{p_2}, \ldots$ one for each state.

4. EXAMPLES

The following examples demonstrate visualization of higher dimensional tracks within two dimensions. The first example demonstrates the mapping by an trajectory zigzagging on the edges of a three dimensional cube. The second example is similar to the first example, but within higher dimensions. A third example demonstrates the application of learn trajectories for hill climbing within three dimensions of a function $F(x,y)$ and the disturbed by logarithmic quantization Eq. (31) function $G(x,y)$. Finally, learn trajectories visualize backpropagation training of a large neural network with floating point arithmetics and disturbed by logarithmic quantization Eq. (31) of the synaptic weights.

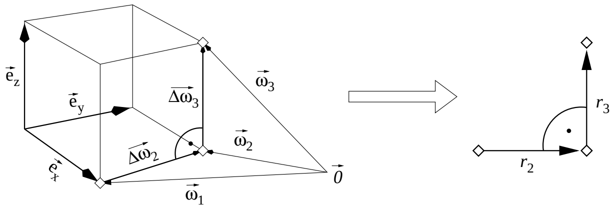

Figure 5: *Learn Trajectory in three dimensions*

Three Dimensional Cube

Accumulation of vectors being orthogonal to each other results in a zigzagging motion (fig. 5) form step to step. In this example, orthogonal unit vectors ($\vec{e_x}_i$) are accumulated within three dimensions

$$\vec{\omega}_n = \sum_{i=1}^{n} \vec{e}_{x_i}. \tag{19}$$

This results in

$$\begin{array}{rclrcl} \vec{\omega}_1 & = & (1,0,0) & & & \\ & & & \Delta\vec{\omega}_2 & = & (0,1,0) \\ \vec{\omega}_2 & = & (1,1,0) & & & \\ & & & \Delta\vec{\omega}_3 & = & (0,0,1) \\ \vec{\omega}_3 & = & (1,1,1) & & & \end{array} \tag{20}$$

with angels

$$\begin{array}{rcl} r_2 & = & |\Delta\vec{\omega}_2| = 1 \\ \Delta\phi_3 & = & \angle(\Delta\vec{\omega}_3, \Delta\vec{\omega}_2) = 90^o \\ r_3 & = & |\Delta\vec{\omega}_3| = 1 \end{array} \tag{21}$$

So, the euclidic distance of sequent difference vectors is the length of the unit vectors and the angles between them is 90^0. This together results in a learn trajectory of two perpendicular vectors ($\vec{r}_2$ and $\vec{r}_3$) as outlined in figure 5.

Hypercube

This example demonstrates the mapping mechanism of learn trajectories from high dimensional zigzagging motion superposed with a linear motion. Combining zigzagging motion based on Eq. (19) with linear motion based on $n \cdot c \cdot \vec{e}_{x_j}$ one gets

$$\vec{\omega}_n = \sum_{i=1}^{n} (\vec{e}_{x_i} + c \cdot \vec{e}_{x_j}). \tag{22}$$

resulting in a sequence of state vectors

$$\begin{array}{rclcl} \vec{\omega}_0 & = & (0,0,0,\ldots,0) & + & (0,\ldots,0,\ldots,0) \\ \vec{\omega}_1 & = & (1,0,0,\ldots,0) & + & (0,\ldots,c,\ldots,0) \\ \vec{\omega}_2 & = & (1,1,0,\ldots,0) & + & (0,\ldots,2c,\ldots,0) \\ \vec{\omega}_3 & = & (1,1,1,\ldots,0) & + & (0,\ldots,3c,\ldots,0) \\ & \vdots & & \vdots & \end{array} \tag{23}$$

where j determines the position of the constant c. The plot of that resulting learn trajectory is pictured in figure 6 for $n = 50$ dimensions, with $c = 0.1$ and $j = 25$. Because the state difference

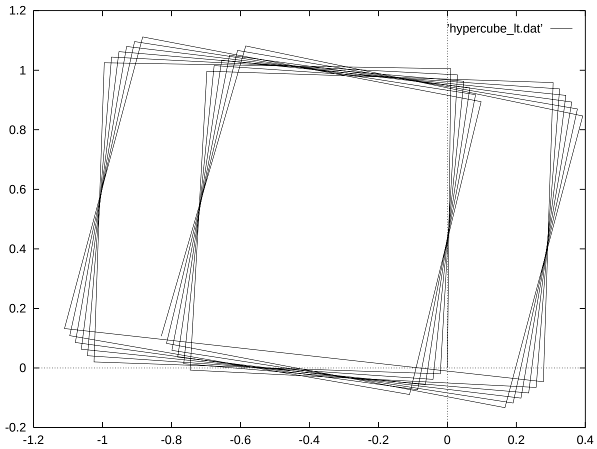

Figure 6: *Hypercube – an learn trajectory example*

vectors $\Delta\omega$ are almost perpendicular, the resulting learn trajectory of rectangular (quadratic for unit vectors) structure. When the accumulated unit vector crosses the position of the constant c at $j = 25$ a shift of the plotted rectangular sub-trajectory occurs. So in contrast to Eq. (20) and Eq. (21), the lengths and the angles of the state difference vectors $\Delta\omega_n$ differ, caused by the additional term $c \cdot \vec{e_x}_j$ in Eq. (22).

Three Dimensional Track Mapping

For demonstration the mapping of a trajectory $f(t) \in \mathbb{R}^3 \rightarrow p_n \in \mathbb{R}^2$, a function composed of terms

$$f(x,y) = \frac{1}{1 + x^2 + y^2} \tag{24}$$

with

$$\vec{\nabla} f(x,y) = \left(\frac{-2x}{(1+x^2+y^2)^2}, \frac{-2y}{(1+x^2+y^2)^2} \right) \tag{25}$$

has been chosen. The function Eq. (24) has been chosen, because it has a single maximum at $\vec{0}$ and it is easily expandable to more than two dimensions

$$f(x_1, x_2, x_3, \ldots, x_n) = \frac{1}{1 + \sum_{i=1}^{n} x_i^2}. \tag{26}$$

Also the gradient of that function can easily be expanded to more than two dimensions with

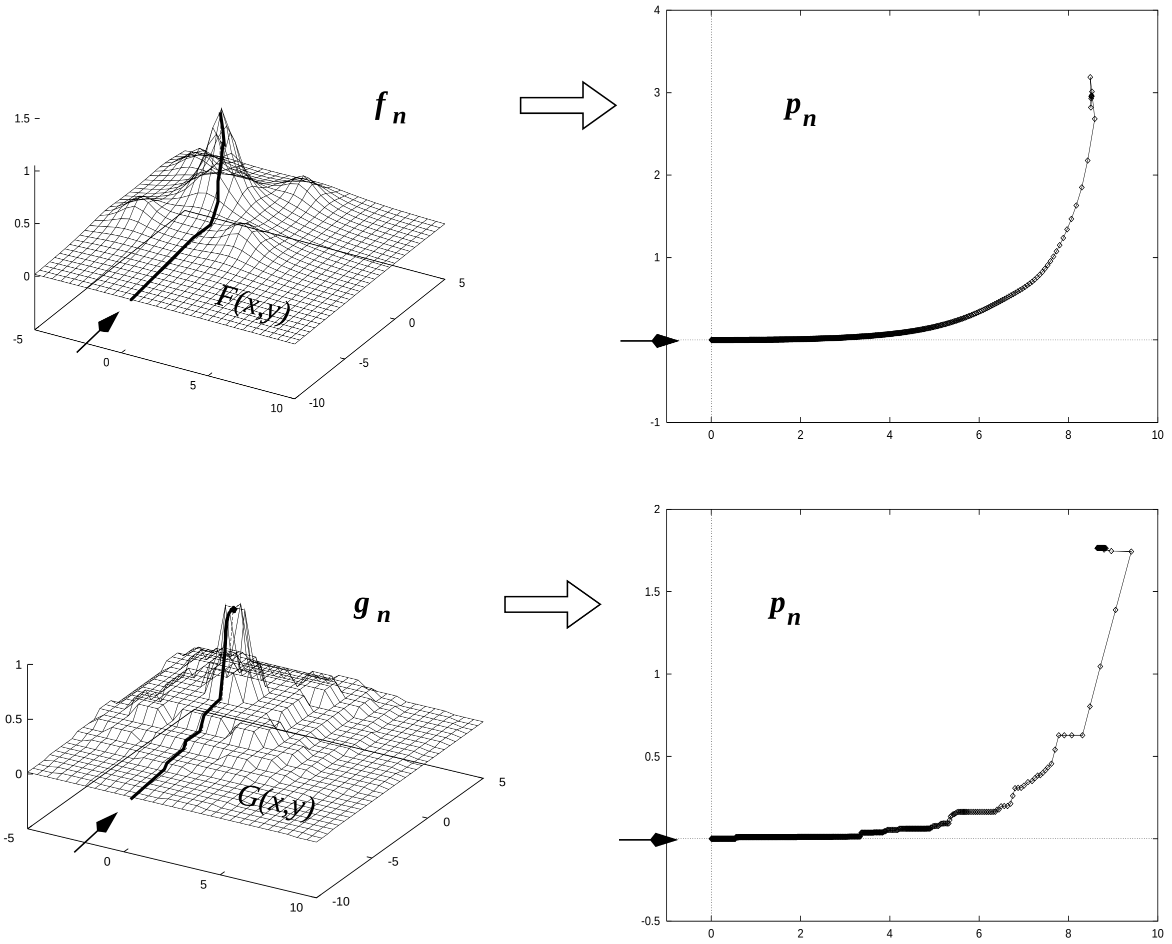

Figure 7: *Three dimensional Track Mapping with $\alpha = 0.1$ and $\eta = 0.9$ and starting point $(x = 0.5; y = -10)$. Both plots base on the same function $F(x,y)$ and it's gradient $\vec{\nabla} F(x,y)$. The hill climbing is done on the $F(x,y)$ surface, but the resulting trajectory is plotted on the surface quantized by $G(x)$.*

$$\frac{\partial}{\partial x_i} f(x_1, x_2, x_3, \ldots, x_n) = \frac{-2x_i}{\left(1 + \sum_{i=1}^{n} x_i^2\right)^2}. \quad (27)$$

Expanding Eq. (24) by shift constants Δx and Δy one gets

$$\tilde{f}(x, y, \Delta x, \Delta y) = \frac{1}{1 + (x + \Delta x)^2 + (y + \Delta y)^2}. \quad (28)$$

With this, a demonstration function

$$\begin{aligned} F(x,y) \quad = \quad & \underbrace{\tilde{f}(x,y,0,0)}_{\text{global maximum}} \\ & \underbrace{+\tfrac{1}{4} \cdot \left(\sum_{i=0}^{3} \tilde{f}(x, y, \Delta x_i, \Delta y_i)\right)}_{\text{local maxima}} \end{aligned} \quad (29)$$

with

$$\begin{aligned} (\Delta x_i, \Delta y_i) \quad \in \quad & \{(3,3), (3,-3), \\ & (-3,3), (-3,-3)\}. \end{aligned} \quad (30)$$

has been constructed. Logarithmic quantization

$$G(\omega) = \begin{cases} -2^{\lfloor \log_2(|\omega|) \rfloor} & \text{if} \quad \omega < 0 \\ 0 & \text{if} \quad \omega = 0 \\ 2^{\lfloor \log_2(|\omega|) \rfloor} & \text{if} \quad \omega > 0 \end{cases} \tag{31}$$

has been applied as a disturbance to the function for demonstration. Figure (7) shows the result – the functions and it's learn trajectories.

Image Recognition Network

This example demonstrates the usage of learn trajectories to trace the learning process of a large backpropagation neural net with $i = 10$ output neurons, $j = 32$ hidden neurons, and $k = 90 \times 70$ inputs. So, the total number of synaptic weights of that backpropagation neural network is $l = (i \cdot j + i) + (j \cdot k + j) = 201,962$. The task to be learned is to classify (recognize) nine different gray scale images (fig. 8). The training has been done over $1,000$ learn cycles, with learn rate $\eta = 0.1$, factor $\alpha = 0.5$, and $\tau = 0.5$. The resulting error plots and learn trajectories are pictured in figure 9. The training has been done based on floating point number representation and with disturbance by logarithmic quantization of the synaptic weights with function Eq. (31).

5. CONCLUSIONS

The method of learn trajectories gives an qualitative insight from the systems point of view into behavior of information processing systems dealing with many parameters simultaneously. It is applicable to systems with few parameters up to systems of a large number of parameters. The method is primary intended to compare the behavior of various implementations of a systems under construction where the effect of different numerical approximations has to be investigated – for instance in the case of software and hardware realization. It may also assist investigations concerning local search algorithms, like the backpropagation algorithm. Probably, the method may be useful to characterize the behavior of those systems. Also, the method may be useful for tuning of system parameters which determine its behavior.

References

[1] Robert Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, 1990.

[2] J. Hertz, A. Krogh, R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley 1991.

[3] L. Larsson, *An EPLD Based Transient Recorder for Simulation of Video Signal Processing Devices in a VHDL Environment Close to System Level Conditions*, Proc. of Sixth Int. Workshop on Field Programmable Logic and Applications, FPL'96, Darmstadt, Germany, 23-25 September 1996, *in* R. W. Hartenstein, M. Glesner, Lecture Notes in Computer Science, Vol. 1142, pp. 371-375, Springer 1996.

[4] L. Larsson, Visualization of Backpropagation by Learn Trajectories to Explore Approximations for Hardware Implementations, *Proc. of 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics*, Berlin, Germany, 24-29 August 1997, Vol. 6, pp. 517-522, Wissenschaft und Technik Verlag, Berlin.

[5] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning Internal Representations by Error Propagation, in J. A. Anderson and E. Rosenfeld (eds.) *Neurocomputing: Foundations of Research*, pp. 673 - 695, MIT Press 1988.

[6] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning Representations by Back Propagating Errors, in J. A. Anderson and E. Rosenfeld (eds.) *Neurocomputing: Foundations of Research*, pp. 696 - 699, MIT Press 1988.

[7] D. E. Rumelhart, J. L. McClelland, *Parallel Distributed Processing*, Vol. 1&2, MIT Press, 1986.

[8] D. W. Patterson, *Artificial Neural Networks – Theory and Applications*, Prentice-Hall 1996.

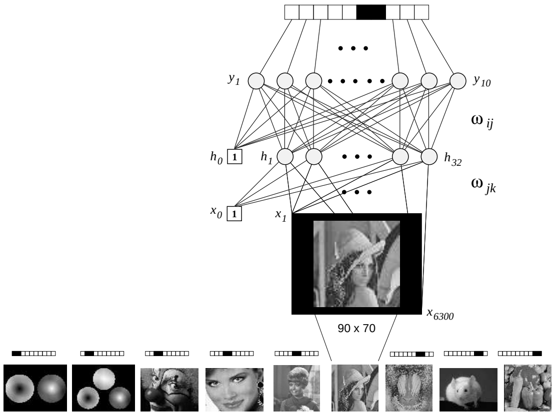

Figure 8: *Image Recognition Network. The task for that backpropagation neural network is to learn to classify (recognize) nine different gray scale images.*

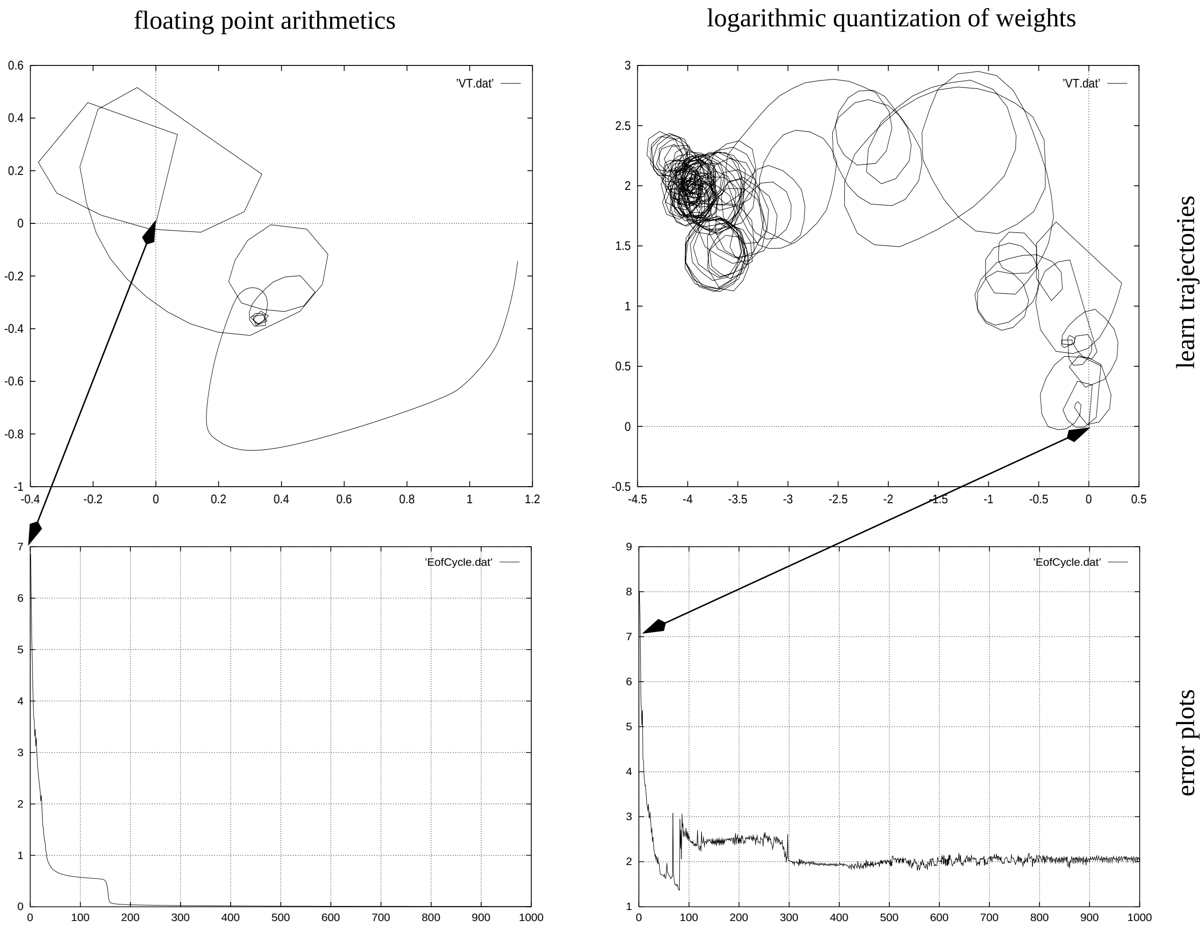

Figure 9: *Learn Trajectories and Error Plot of the Image Recognition Network (fig. 8). This example demonstrates the application of learn trajectories to system of approximately* $200{,}000$ *synaptic weights. Training was done with* $\eta = 0.1$*, momentum factor* $\alpha = 0.5$*, and* $\tau = 0.5$ *based on floating point arithmetics and with logarithmic quantized synaptic weights.*