Visualization of Backpropagation by Learn Trajectories to Explore Approximations for Hardware Implementations

L. Larsson
University of Hamburg, Dept. of Computer Science / Group TECH
Vogt-Koelln-Str. 30, D-22527 Hamburg, Germany
e-mail: larsson@informatik.uni-hamburg.de

ABSTRACT

Approximations are necessary for efficient hardware implementations of backpropagation neural networks. Those approximations result in an inaccurate arithmetics. Visualization of backpropagation by learn trajectories as presented helps to estimate the effects of those approximations. To demonstrate it, training is done with floating point, fixed point, and with logarithmic quantized weights.

INTRODUCTION

Backpropagation (*BPG*) learning introduced by D. E. Rumelhart, J. L. McCelland, G. E. Hinton and R. J. Williams, can be interpreted as motion of a point representing the current state of a network in a high dimensional space. Mapping this high dimensional motion to a trajectory in two dimensions allows visualization of the motion in the high dimensional space. This is useful for investigations concerning different arithmetics or for training parameter estimation. In contrast to a simple error plot the mapping presented in this paper allows to distinguish between plateaus and local minima of the error function, if one supposes that a plateau results in a straight motion, whereas a local minimum results in a circular motion. So, this allows a more detailed comparison of different arithmetics and approximations.

For demonstration, backpropagation learning is presented based on floating point arithmetics, fixed point arithmetics, and with synaptic weights which are quantized next to power of two of its fixed point representation. So, their logarithm dualis is an integer. Multiplication of an activity with a power of two is trivial. With this inaccuracies backpropagation learning still terminates and produces training results similar to floating point based training for several examples tried. The present work is important for hardware implementation of backpropagation learning.

BACKPROPAGATION LEARNING

This section summarizes the backpropagation neural network model. Each neuron i of a net computes $y_i = g(\theta_i + \sum_{j=1}^{n} \omega_{ij} \cdot x_j)$. In case of *bias* the expression is modified by $\omega_{i0} := \theta_i$, $x_0 := 1$ and $j = 0$. A net with k inputs, j hidden and i output neurons maps a pattern $\vec{x}$ to a classification

$$\vec{y} = F(\Omega_{ij}, \Omega_{jk}, \vec{x}), \tag{1}$$

where Ω_{ij} and Ω_{jk} are weight matrices. To be compatible 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

Published in Proc. of 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics, August 24-29, 1997, Berlin, Germany, in A. Sydow (ed.), Proceedings Volume 6 - Application in Modelling and Simulation, pp. 517-522, Wissenschaft & Technik Verlag, Berlin, 1997. ISBN 3-89685-556-5

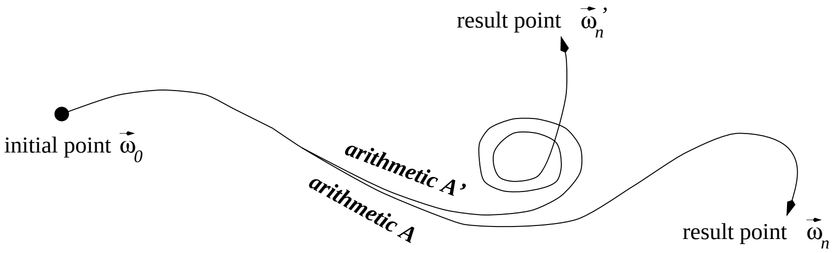

Figure 1: *Learn trajectories and arithmetics. Different approximations affect training behavior of the backpropagation network resulting in different learn trajectories displayed in two dimensions.*

weight matrices Ω have to be expanded by one column each, which become the threshold vectors $\vec{\Omega}_0$. With this, feed forward mapping (1) is

$$\vec{h} = \Omega_{jk} \cdot \vec{x}, \quad \vec{\gamma} = g(\vec{h}), \quad \vec{o} = \Omega_{ij} \cdot \vec{\gamma}, \quad \vec{y} = g(\vec{o}), \tag{2}$$

where $\vec{x}$, $\vec{h}$, $\vec{\gamma}$, $\vec{o}$ and $\vec{y}$ are column vectors and $g(a_i) = 1/(1 + e^{-a_i/\tau})$ is the sigmoid function and $g'(a_i) = g(a_i) \cdot g(-a_i)/\tau$ its derivation with a temperature parameter τ (*typical* $0.1 < \tau < 10$). Training is done with patterns $(\vec{x}; \vec{t})_M$ ($\vec{x} \approx$ *input vector*; $\vec{t} \approx$ *target classification*), until $E = \frac{1}{2} \sum_M (\vec{y} - \vec{t})^2_M$ becomes minimal (or less a limit), by computing

$$\vec{d} = \vec{y} - \vec{t}, \quad \vec{\delta} = g'(\vec{o}), \quad \vec{e} = {}^t\,\Omega_{ij} \cdot (\vec{d} \otimes \vec{\delta}), \quad \vec{\epsilon} = g'(\vec{e}), \tag{3}$$

for each pattern M, where ${}^t\Omega_{ij}$ is the transposed matrix Ω_{ij} and the operator $\otimes$ here is a component like vector-vector product as defined by $\vec{c} = \vec{a} \otimes \vec{b} \Leftrightarrow c_i = a_i \cdot b_i$. With this, the update of the matrices

$$\begin{array}{lclcl} \Omega_{ij} & := & \Omega_{ij} - \eta \cdot \vec{\delta} \cdot^t \vec{\gamma} & + & \alpha \cdot \Delta\Omega_{ij} \\ \Omega_{jk} & := & \Omega_{jk} - \eta \cdot \vec{\epsilon} \cdot^t \vec{x} & + & \alpha \cdot \Delta\Omega_{jk} \end{array} \tag{4}$$

appears in direction of the negative gradient of the error function E, with a learning rate η (*typical* $0.001 < \eta < 0.5$) and additional momentum parameter α (*typical* $0.1 < \alpha < 0.9$). The momentum parameter avoids oscillations while training.

LEARN TRAJECTORIES

After n learn steps the network is in state Ω_n. This state Ω_n is defined by the networks synaptic ω_{ij}, ω_{jk} weights and thresholds ω_{i0}, ω_{j0} (*bias*). So, a vector

$$\vec{\omega}_n = (\underbrace{\omega_{10}, \ldots, \omega_{i0}}_{\approx\theta_i}, \underbrace{\omega_{11}, \ldots, \omega_{ij}}_{\Omega_{ij}}, \underbrace{\omega_{10}, \ldots, \omega_{j0}}_{\approx\theta_j}, \underbrace{\omega_{11}, \ldots, \omega_{jk}}_{\Omega_{jk}})_n = (w_1, \ldots, w_l)_n \tag{5}$$

of $l = (i \cdot j + j) + (j \cdot k + k)$ components represents the current state Ω_n of a two layer backpropagation network with i inputs, j hidden neurons, and k output neurons ([i-j-k] net). The euclidic distance r_n between two state vectors $\vec{\omega}_n$ and $\vec{\omega}_{n+1}$ is

$$r_n = |\Delta\vec{\omega}_n| \quad \text{with} \quad \Delta\vec{\omega}_n = \vec{\omega}_{n+1} - \vec{\omega}_n, \tag{6}$$

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

$$\phi_n = \arccos\left(\frac{\Delta\vec{\omega}_{n+1}\cdot\Delta\vec{\omega}_n}{|\Delta\vec{\omega}_{n+1}|\cdot|\Delta\vec{\omega}_n|}\right). \tag{7}$$

The distance r_n together with the angle ϕ_n represents a two dimensional vector $\vec{\rho}_n = (r_n, \phi_n)$. Transformed from polar coordinates to cartesian coordinates one gets

$$\vec{x}_n = (x_1, x_2)_n = (r_n \cdot \cos(\phi_n), r_n \cdot \sin(\phi_n)). \tag{8}$$

With this, a sequence of n training steps results in a sequence of n base points

$$f_n = \{\vec{\omega}_0, \vec{\omega}_1, \ldots, \vec{\omega}_{n-1}, \vec{\omega}_n\} \tag{9}$$

of a learn trajectory $f(t) \in \mathbb{R}^l$, $t \in \mathbb{R}$ in the l dimensional space of synaptic weights of the [i-j-k] network, with $\lim_{r_n \to 0} f_n = f(t)$. These l dimensional base points f_n can be plotted as a two dimensional sequence of points

$$p_n = \{\vec{p}_0, \vec{p}_1, \ldots, \vec{p}_{n-1}, \vec{p}_n\} \qquad \text{with} \qquad \begin{cases} \vec{p}_0 &= (0,0)_0 \\ \vec{p}_n &= \vec{p}_{n-1} + \vec{x}_n \quad \text{for} \quad n > 0. \end{cases} \tag{10}$$

Chained, this sequence of points in two dimensions visualizes the shape of a l dimensional learn trajectory $f(t) \in \mathbb{R}^l$.

DEMONSTRATION

Two examples demonstrate visualization of higher dimensional tracks within two dimensions. In the first example gradient descent results in a three dimensional trajectory to visualize mapping of a higher (more than two) dimensional trajectory to a two dimensional trajectory. The second example demonstrates visualization of backpropagation learning (gradient descent) itself comparing training of different arithmetics as announced.

Three Dimensional Track Mapping

The mapping of a trajectory $f(t) \in \mathbb{R}^3 \to p_n \in \mathbb{R}^2$, which is the result of gradient ascent of a function composed of terms

$$f(x,y) = \frac{1}{1+x^2+y^2} \qquad \text{with} \qquad \vec{\nabla} f(x,y) = \left(\frac{-2x}{(1+x^2+y^2)^2}, \frac{-2y}{(1+x^2+y^2)^2}\right) \tag{11}$$

has been chosen for demonstration. Enhanced by shift constants Δx and Δy one gets

$$\tilde{f}(x,y,\Delta x,\Delta y) = \frac{1}{1+(x+\Delta x)^2+(y+\Delta y)^2}. \tag{12}$$

With this, the demonstration function $F(x,y)$ has been chosen as

$$F(x,y) = \underbrace{\tilde{f}(x,y,0,0)}_{\text{global maximum}} + \frac{1}{4}\cdot\underbrace{(\sum_{i=0}^{3}\tilde{f}(x,y,\Delta x_i,\Delta y_i))}_{\text{local maximums}} \tag{13}$$

with

$$(\Delta x_i, \Delta y_i) = (\Delta x, \Delta y)_i \in \{(3,3)_0, (3,-3)_1, (-3,3)_2, (-3,-3)_3\}. \tag{14}$$

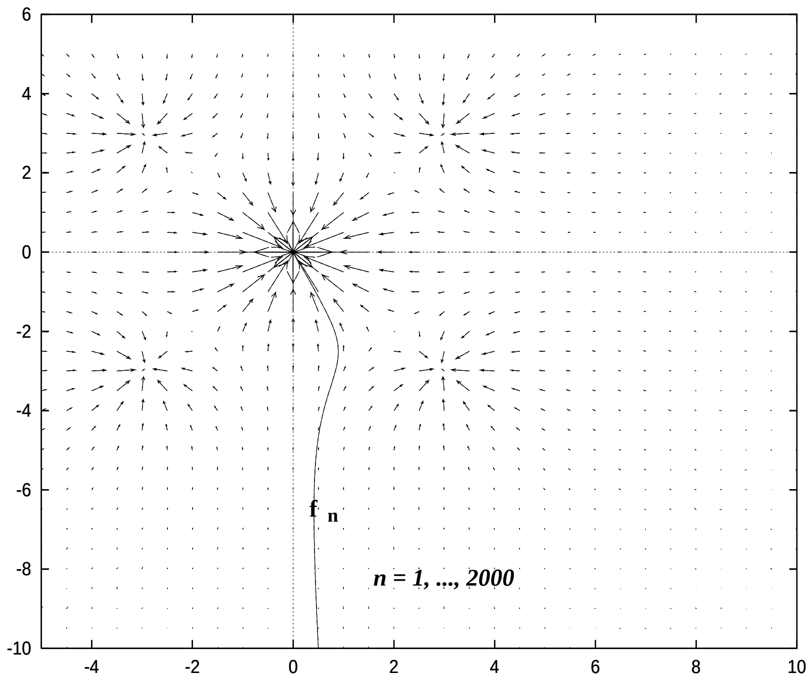

Figure 2: *Example function $f(x,y)$, gradient field $(\vec{\nabla} F(x,y))$, and trajectory f_n.*

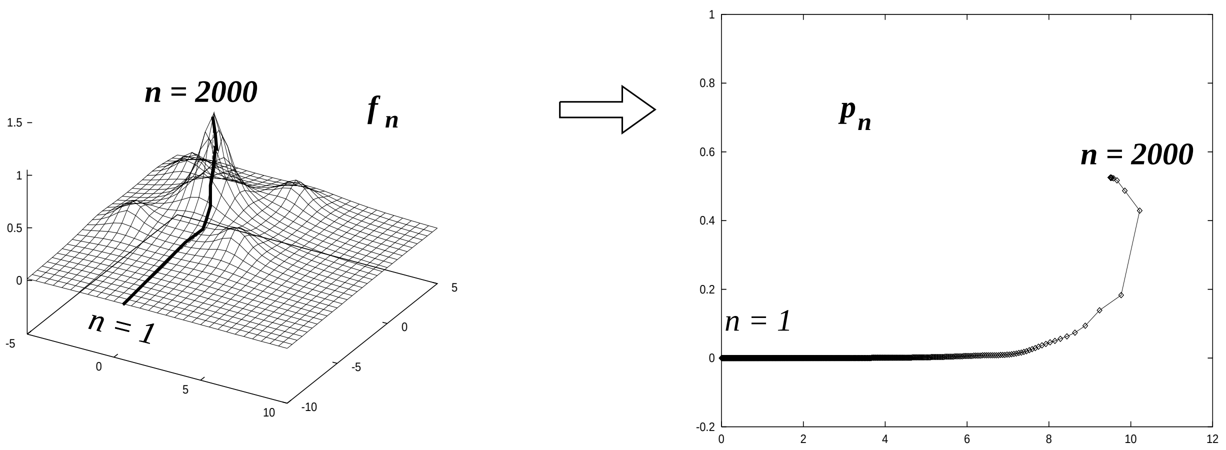

Figure 3: *Three dimensional trajectory f_n and it's two dimensional mapping p_n.*

A plot of the demonstration function F(x,y) (13) together with it's gradient $\vec{\nabla} F(x,y)$ field and a gradient ascent track f_n are outlined in figure (2). Figure (3) shows a three dimensional plot of $F(x,y)$ (13) and the resulting two dimensional trajectory, which becomes the learn trajectory in the case of backpropagation training.

Simple Character Recognition

As a simple example, training is done by learning simple character recognition of 5x7 pixel capitalized characters classified by an [35-10-26] backpropagation net (figure (4)) with a learning parameter $\eta = 0.05$ and momentum parameter $\alpha = 0.1$. The weights have been initialized with random values ± 0.5 and training is performed by 10,000 cycles with a temperature parameter $\tau = 0.5$ for the sigmoid function.

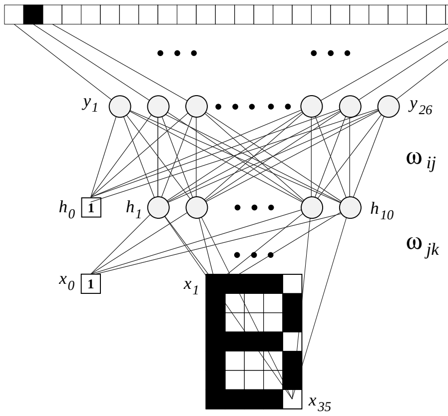

Figure 4: *Simple character recognition.*

Figure (5) compares different approximations with respect to their error ($E = \frac{1}{2}\sum_M(\vec{y}-\vec{t})^2_M$) behavior and to their learn trajectories. Training based on *floating point* (single precision `float`) results in a smooth learn trajectory. Training based on *fixed point* arithmetics, where interim results I were quantized according to

$$I(x) = \frac{\lfloor (2^n - 1) \cdot x \rfloor}{2^n - 1} \qquad \text{with} \quad n = 15 \tag{15}$$

gives a small disturbance. Training with logarithmic quantized weights

$$L(\omega) = \begin{cases} L &= -2^{\lfloor \log_2(|\omega|) \rfloor} & \text{if} \quad \omega < 0 \\ L &= 0 & \text{if} \quad \omega = 0 \\ L &= 2^{\lfloor \log_2(|\omega|) \rfloor} & \text{if} \quad \omega > 0 \end{cases} \tag{16}$$

is a hardware-friendly modification, but it results in noisy training behavior. The difference to the other both approximations becomes more obvious by the learn trajectory plot compared to corresponding plot of it's error function.

REFERENCES

[1] 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.

[2] 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.

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

[4] I. N. Bronstein, K. A. Semendjajew, *Taschenbuch der Mathematik*, Harri Deutsch, 1981.

[5] T. Williams, C. Kelley, *et. al.*, GNUPLOT-Program, Version 3.5, 1986-1993, `http://www.cs.dartmouth.edu/gnuplot_info.html`

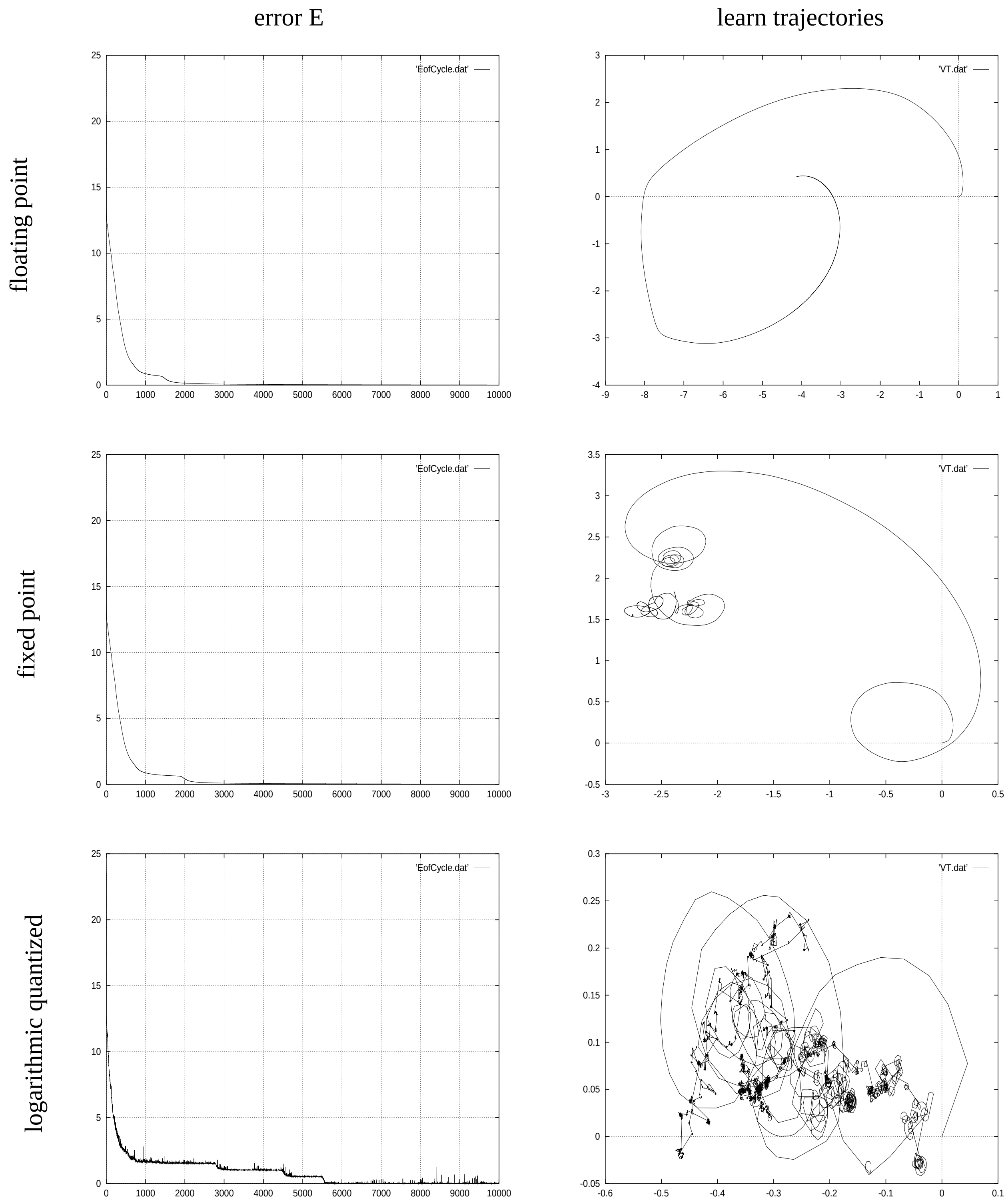

Figure 5: *Learn trajectories of learning simple character recognition (5x7 pixels) with floating point, fixed point, and logarithmic quantized weights.*