Hyperplane "Spin" Dynamics, Network Plasticity and Back-Propagation Learning*

Frank J. Śmieja †

German National Research Centre for Computer Science (GMD),

Schloß Birlinghoven, 5205 St. Augustin 1,

Germany.

November 28, 1991

Abstract

The processing performed by a feed-forward neural network is often interpreted through use of decision hyperplanes at each layer. The *adaptation* process, however, is normally explained using an error landscape picture. In this paper the actual *dynamics* of the decision hyperplanes is investigated. As a result a mechanical analogy is drawn with a system of spins acted upon by forces. The spin objects have a variable mass, and relaxation in the system is represented by increased overall system mass, also termed "cooling". The analogy is used to clarify the dynamics of learning, information storage and robustness, and in particular the functioning of the process of back-propagation. Learning deadlocks and local minima are explained. The concept of network "plasticity" is introduced, and used to understand destructive relearning, and how it may thus be better avoided.

Practical benefits from this interpretation are illustrated with hints for optimal weight initializations, avoidance of and escape from some types of local minima, and avoidance of destructive relearning through selective reordering of some types of networks.

1 Introduction

The back-propagation algorithm is normally explained as a gradient descent in *weight* space. However, for feed-forward nets one has a degree of *structure* present in the

*GMD report, submitted to *Complex Systems*.

†Internet: `smieja@gmdzi.gmd.de`, Bitnet: `smieja@gmdzi.uucp`

organization of the weights, and one may expect it to be possible to reparameterize the process such that it may be represented as an optimization of a different, more meaningful set of variables. Using the constraints that the net is feed-forward and consists of layers of nodes that do not mutually interact, one such reparameterization is as follows. Consider the variables to be the set of *hyperplane definition vectors* in each layer of weights between node layers, and the optimization process to be minimization of the "forces" causing these hyperplanes to move in their respective spaces. These forces are directly related to what is usually referred to as the "delta errors" in the mapping at each node in the network. Optimization results in movement of and anchoring of hyperplanes in pattern representation space.

In this communication the dynamics of such hyperplanes is derived, firstly from considering a single node network (section 2), then extending to networks with a hidden layer (section 6). We are then in a position in section 7 to annotate a learning scenario, and in section 8 to clarify descent into local minima, and show how to escape from some of them. With this useful picture of network dynamics hints can be derived for optimal weight initializations and also for monitoring the *plasticity* of a network (section 9). Furthermore, in section 9 we also demonstrate how destructive relearning may be reduced when extending the capabilities of an optimized net, through lowering the plasticity of the net in particular directions, while preserving most of the generalization knowledge it contains.

2 Single predicate case

We consider first the single predicate case. There are n input units and no hidden units and the single output unit is connected to the input units via the weights $\{w_i\}, 1 \leq i \leq n$, and has associated a threshold value w_0. This unit adopts states according to a monotonically increasing non-linear *sigmoid* function $f(\vec{\xi}.\vec{w}-w_0)$ of the input pattern $\vec{\xi}$, that is constrained to lie in $[0,1]$ ($f(x) = 1/(1+e^{-\beta x})$). β represents the sharpness of the sigmoid function. It is antisymmetric about $f(x) = 0.5$, and has the properties (see figure 1):

$$f(x) + f(-x) = 0.5 \tag{1}$$

$$f(x) \begin{cases} \geq 0.5 & \text{if} \quad x \geq 0 \\ < 0.5 & \text{otherwise.} \end{cases} \tag{2}$$

The output hyperplane h defines the plane of symmetry: the surface $f(x) = 0.0$, or the surface $x = 0$:

$$h: \qquad \vec{w}.\vec{r} = w_0.$$

Alternatively, we may write h in terms of the normalized vector $\vec{n}$, parallel to $\vec{w}$:

$$h(\gamma, \vec{n}, \theta): \qquad \gamma \times [\vec{n}.\vec{r} = \theta] \tag{3}$$

$$\theta := w_0/\gamma. \tag{4}$$

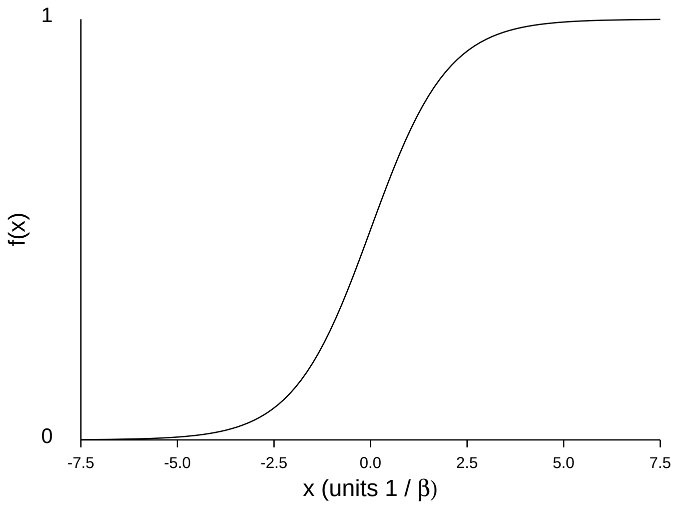

Figure 1: Response function: sigmoid, $f(x) = \frac{1}{1+e^{-\beta x}}$. Units of x are $1/\beta$. See later for discussion of radial basis functions.

$\vec{n}$ then determines the *orientation* of the hyperplane in its optimization space, and θ, the hyperplane *threshold*, or its perpendicular *displacement* from the origin. The parameter $|\gamma|$ represents the "height" of the decision hyperplane mapping characteristic. See figure 2.

How, then, are the weights of this simple predicate node to be altered, when we use the back-propagation algorithm?

The initial weights chosen for the system define the decision hyperplane h. From back-propagation, each weight w_i from input unit i to the output unit is changed by the following amount each update epoch (assuming *batch* update[1]):

$$\Delta w_i = \eta \sum_p (t^p - o^p)\, f^{p\prime}\, \xi_i^p \tag{5}$$

where f' is the differential of the output node function, t^p is the target value for pattern p at the output unit, o^p is the actual network output value, and ξ_i^p is element i of pattern p. For the threshold w_0, $\xi_0^p = 1, \ \forall p$.

The decision hyperplane is given by:

$$\begin{aligned} \gamma \cdot \vec{n} &= (w_1, w_2, \ldots, w_n), & (6) \\ \gamma \cdot \theta &= w_0. & (7) \end{aligned}$$

Then, a small change Δ in the hyperplane is given by:

$$\Delta(\gamma \cdot \vec{n}) = (\Delta w_1, \Delta w_2, \ldots, \Delta w_n) \tag{8}$$

[1]When *online* (or *per-sample* learning is performed in small steps it approximates to batch learning [16]. In the large step case the situation is different. This is discussed in section 10.

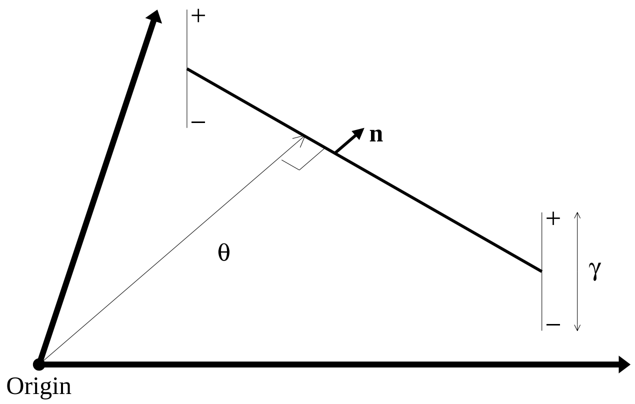

Figure 2: The parameters defining the decision hyperplane in its space.

so from eqn. (5), and differentiating eqn. (8) with respect to time, assuming small changes (i.e. *not* online learning):

$$\frac{d(\gamma \cdot \vec{n})}{dt} = \eta \sum_p (t^p - o^p)\, f^{p\prime}\, \vec{\xi}^p. \tag{9}$$

One may decompose the left hand side of eqn. (9) into two components of hyperplane change:

$$\frac{d(\gamma \cdot \vec{n})}{dt} = \frac{d\gamma_j}{dt}\vec{n}_j + \gamma_j \frac{d\vec{n}_j}{dt} \tag{10}$$

Reinserting eqn. (10) into eqn. (9) and performing the operation "$.\vec{n}$" on both sides we obtain the equation for the change in γ_j, and by performing the operation "$\wedge\vec{n}$" on both sides we get the equation for the change in the angle (ϕ) of the vector $\vec{n}$:

$$\frac{d\vec{n}}{dt} \wedge \vec{n} = \frac{d\vec{\phi}}{dt} = \vec{\omega} \quad = \quad \frac{\eta}{\gamma} \sum_p (t^p - o^p)\, f^{p\prime}\, \vec{\xi}^p \wedge \vec{n}, \tag{11}$$

$$\frac{d\gamma}{dt} \quad = \quad \eta \sum_p (t^p - o^p)\, f^{p\prime}\, \vec{\xi}^p.\vec{n}. \tag{12}$$

Also, the rate of change of θ_j, is, from eqn. (7), and ignoring higher order terms:

$$\frac{d\theta}{dt} = \frac{\Delta w_0}{\gamma} = \frac{\eta}{\gamma} \sum_p (t^p - o^p)\, f^{p\prime}. \tag{13}$$

We have now the important equations that inform us of the dynamics of the hyperplane parameters involved during learning in this single predicate, single decision hyperplane system. In the next section we explain how a spin system analogy may be drawn to shed more light on the actual relaxation processes steered by back-propagation learning.

3 The spin analogy

During back-propagation learning the decision hyperplane in our one-node system is being moved around, in terms of shifts and rotations, in its representation space. The hyperplane is being moved around because "the system" is trying to optimize its location and orientation in the hyperplane space. The system is trying to reduce its energy. Since there is an energy in the system, we should be able to identify a *force* and a *mass* that also play their part in the system. Let us therefore consider the hyperplane to be equivalent to an object that has a *mass*, which we will call m. Then in order to move the hyperplane with our postulated force we require an interaction. The obvious form of analogy to use for the interaction that allows the hyperplane to be shifted and turned at the same time is one that is electromagnetic. Thus let the hyperplane possess a *charge* q, and *spin* on its axis, defined by $\vec{n}$. It has, then (for the sake of argument), a spin $\vec{\sigma} = e\,\vec{n}$. The quantity e is explained in the next paragraph.

The charge q on the hyperplane is equivalent to the effect it has on a pattern (particle) in the space, after mapping it. Thus, the sign of the charge q is special in this system, depending on which side of the hyperplane we are considering. The side of the hyperplane mapping particles to $+1$ has charge $q = e$, and that side mapping particles to 0 (or -1, depending on the ranges used in the node response functions) has charge $q = -e$. It is therefore reasonable to denote the charge of the hyperplane with respect to the side affecting the particle p being considered. So instead of q, we shall write the hyperplane charge $q(p)$. A particle p has associated a charge q_p. Like charges repel, and unlike charges attract. The sense of the hyperplane vector actually defines the direction $q = -e \rightarrow q = e$. A particle p in the system is fully defined by its charge q_p and position vector $\vec{\xi^p}$. The position vector gives the particle a "sense", so that it influences the hyperplane also through the torque it causes, as it tries to align the hyperplane spin with its own spin $\vec{\sigma}^p = q_p\vec{\xi^p}$.

The form of the turning interaction is given by the torque $\mathbf{T}_p$ caused by particle p on the hyperplane:

$$\mathbf{T}_p \propto \vec{\sigma}_p \wedge \vec{\sigma}. \tag{14}$$

Missing is a term representing the potential field, which we shall denote by h^p, indicating how the interaction decays with the separation of the charges. The effect of this instantaneous torque is to change the hyperplane's angular momentum L by

$$\Delta L = I\omega \propto h^p(\vec{\sigma}^p \wedge \vec{\sigma}). \tag{15}$$

The moment of inertia, I, of our hyperplane is that of a thin rod of twice unit length, and of mass m, about the center, i.e.,

$$I = \int_{-1}^{1} mr dr = m.$$

Similarly, the *energy* between particle p and the hyperplane spin is given as usual by:

$$E_p \propto h^p\,(\vec{\sigma}^p.\vec{\sigma}). \tag{16}$$

Finally, the translational force of particle p on the hyperplane is given by:

$$\mathbf{F}_p \propto q_p q(p)\, h^p\, \vec{n} \tag{17}$$

which, when applied instantaneously, causes a change in the linear momentum of

$$m\dot{\vec{x}}_p \propto q_p q(p)\, h^p\, \vec{n}, \tag{18}$$

where $\vec{x}_p = x_p\,\vec{n}$ is the vector parallel to $\vec{n}$ and linking the hyperplane and the particle p.

The constant of proportionality, which is required in eqns. (15), (16) and (18) to give a scale to the force, we shall call g.

The analogy is now complete in so far as can be drawn without knowledge of the nature of the interaction decay with separation (h^p). We shall see in the next section, that this is dependent on the response function f used and its derivative.

4 Ranges and induced charges in the system

It is not hard to use the isomorphism between eqns. (11), (13) and (15), (18) to identify the following analogues in the hyperplane system and the spin system:

$$\begin{aligned}
\text{mass } m &\equiv \gamma && (19)\\
\text{charge } e &= 1 && (20)\\
\text{displacement } dx_p &\equiv \frac{q(p)}{e}d\theta && (21)\\
\text{range term } h^p &\equiv f^{p\prime} && (22)\\
\text{particle charge } q^p &\equiv (t^p - o^p) && (23)\\
\text{force constant } g &\equiv \eta. && (24)
\end{aligned}$$

Thus in this system the charge of a particle varies according to its position relative to the hyperplane charge. We may think of it as an *induced* charge. This is an important point, that helps us later to understand better when destructive learning is about to occur (section 9).

The spin equations therefore become:

$$\begin{aligned}
\vec{\omega} &= \frac{g}{m}\sum_p h^p\,(\vec{\sigma}^p \wedge \vec{\sigma}), && (25)\\
\frac{dm}{dt} &= g\sum_p h^p\,(\vec{\sigma}^p.\vec{\sigma}). && (26)\\
\frac{dx}{dt} &= \frac{g}{m}\sum_p q_p q(p)\, h^p. && (27)
\end{aligned}$$

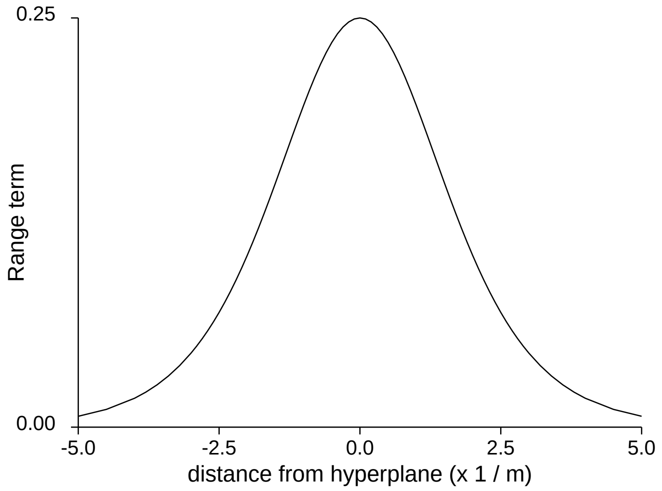

Figure 3: Form of the range function h. Patterns nearest the boundary have the most effect.

What is surprising in this isomorphism, is that from eqns. (26) and (16) it seems that the current energy E of the system (or hyperplane subsystem if there is more than one hyperplane) is added every time step directly to the energy, or mass, of the hyperplane itself. This is very important, as the mass of the hyperplane defines its dynamics.

A note should be made here with regard to the two degrees of motion that the hyperplane undergoes. Our representation of the spin is defined to be parallel to the hyperplane vector $\vec{n}$, but there is freedom in so far as the other dimension is concerned. When we hold the other parameter, θ, constant as the hyperplane rotates, the representation will appear to rotate about the origin with radius θ and not about the spin center. This does not cause any complications in the dynamics equations, but should be realized when trying to picture the motion in the hyperplane space.

The range term h^p originates from the response function f of the nodes. In the sigmoid case assumed here, the range term is illustrated in figure 3.

It can be seen that the patterns that have the most effect on the dynamics of the hyperplane are those nearest to it. This point sounds rather obvious to make in the context of our model, but it is not necessarily so obvious when one uses a standard error-surface interpretation of back-propagation. The "boundary-pattern" observation has been made by other researchers [1, 17]. Note too, in figure 3, that the "distance" from the hyperplane is measured in units of $1/m$. Thus, as the mass of the hyperplane increases, its range of influence decreases. This is another very important point about the system, that explains cooling and plasticity (section 9).

It is evident from the form of the charge in eqn. (23), that the incorrectly sided patterns will contribute more to the forces on the hyperplane than will the correctly sided patterns. This is the way back-propagation places a greater priority on getting neighboring patterns onto the right side of the hyperplane than on mapping them better once there.

5 Interpretation

Before advancing to full-sized networks, we will discuss once more the processes going on during the single predicate back-propagation learning, using our spin analogy.

The dynamics of the hyperplane is defined by its spin $\vec{\sigma}$, its perpendicular distance from the particle x_p and its mass m. It possesses a charge that is positive on the forward (in terms of the spin direction) side of the hyperplane, and negative on the backward side. The particles in the hyperplane world can affect its state through a kind of electromagnetic interaction with its charge and spin. The particles in the system are *fixed* and so the resultant force and torque they result in on the hyperplane cause the hyperplane (and not the particles) to moving and turning respectively. Furthermore, the interaction energy of the system, at each time-step, is added to the mass m of the hyperplane.

The closer the particles are to the hyperplane, the greater the influence they have on its motion. When a particle has a like charge it has a lesser repulsive force than the attractive force of a particle with an unlike charge in the same position. The further away a particle is, in units of $1/m$, the smaller its influence on the hyperplane motion, regardless of its charge.

The system is said to be *relaxed* when the free energy H of the system is zero. We define the free energy as:

$$H := \left(\sum_p \sum_h (E_{ph})^2 \right)^{1/2}, \tag{28}$$

where the index h refers to the hyperplane for which the energy is calculated, and the summation is over all hyperplanes and patterns. This can, as is usual in dynamic systems, be likened to an overall *temperature*, τ, of the system:

$$\tau \propto H. \tag{29}$$

Thus, relaxing the system is equivalent to cooling it down, as mentioned at the beginning of section 3. A relaxed system may however, not necessarily result in a network that solves the mapping problem, as will be explained in section 8. When a system is relaxed, the hyperplanes are stable, and it does not change.

From eqns. (25) and (27) it can be seen that the hyperplane stops turning and shifting as $m \to \infty$ and/or $h^p \to 0$ and/or $q^p \to 0 \ \forall p$. The latter option is only possible in this perceptron-like network for linearly separable problems [10], which indicates "problem solved". The second condition is satisfied when the first condition is being satisfied anyway, since the forces all become shorter range as m increases (see figure 3).

One can imagine a "deadlocking" situation, whereby two particles with opposite charges, have the property for a given hyperplane configuration, that

$$h^1 = h^2 \tag{30}$$

$$\vec{\sigma}^1 \wedge \vec{\sigma} = -(\vec{\sigma}^2 \wedge \vec{\sigma}) \tag{31}$$

The result of this is that, from eqns. (25) and (27), the hyperplane will not be moved to produce the required mappings, and from eqn. (26), the hyperplane will just grow in mass.

If the hyperplane is *initialized* to have too large a mass, it may be very difficult for it to be moved to an optimal position and orientation, because the force ranges are too small. One notes too, that if it is attempted to counteract the drawback of a large mass through increasing the strength of the forces in the system, given by g, then eqn. (26) informs us that the mass changes *more quickly*, either in a positive direction, or unstably in large steps about its present large value. This explains why forced escape from local minima through temporary increase in step size is not terribly successful.

The orientation angle change is also such as to minimize the acute angle between the hyperplane spin vector and the weighted (by the set of particle interactions) vector addition of the particle vectors. This is perhaps better understood when one considers two particles, $\vec{r}_1$ and $\vec{r}_2$, influencing the orientation of a hyperplane. Let us assume that they have opposite charges, and that they are both on the correct sides of the hyperplane (let q_1 be –ve). At equilibrium, they are the same perpendicular distance from the hyperplane (because eqn. (27) says the translational movement is stable when $h^1 = h^2$). Also, it can be seen that at equilibrium the hyperplane adopts the orientation (from eqn. (25)), which says the hyperplane (spin) vector $\vec{n}$ should be parallel to the vector $\vec{r}_2 - \vec{r}_1$, when the system has the property that the perpendicular distances $\{|x_p|\}$ of the particles from the hyperplane are maximized (see figure 4).

In other words, the hyperplane looks for the position that would be found by a *least squares approximation* through the points specified by the particle positions. This is precisely the claim made in the perceptron convergence theorem [15]. We have demonstrated it here in a slightly different way, with sigmoid response function. This notion is extendible to full-size networks (see [3]).

Once the orientation of the hyperplane has been so optimized, it can be seen that the cross product in eqn. (26) has been maximized, and this increases the mass of

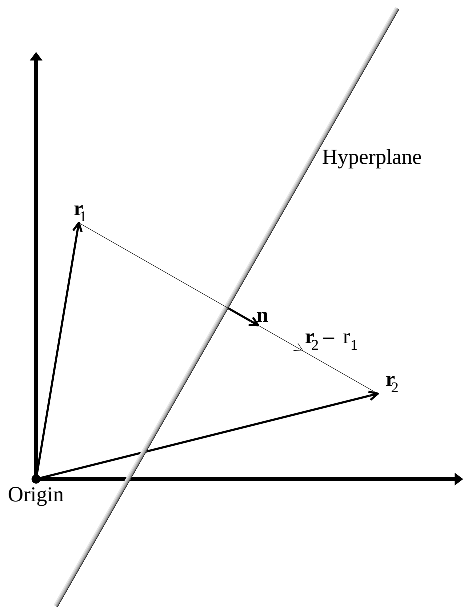

Figure 4: Final optimal state of the hyperplane when two particles, with position vectors r_1 and r_2, are to be mapped to opposite sides. Note the back-propagation algorithm looks for the orientation that gives it the maximum perpendicular distance from each particle.

the stationary hyperplane until the ranges $\{h^p\}$ decay, to freeze the system (cool it to $\tau = 0$).

Alternatively, one can view the optimization process as *focusing the sensitivity* of the spin (hyperplane): at the start, all the particles in the unit hypercube (where they are defined) contribute to the movement of the spin, but as it starts to increase in mass it loses sensitivity to distant particles (force becomes more short-range) and is controlled to a greater extent by those nearby particles. Clearly, back-propagation has an *automatic* form of annealing, that allows it to relax its search temperature from high to low.

6 Nets with a hidden layer

Now, instead of connecting the weights from the input directly to the output predicate node, we insert a *hidden layer* of N_H nodes, fully connected to each input node via the weights w_{ij}, and fully connected to the output node, via the weights w_j. The index j labels the hidden hyperplanes.

The situation is more interesting, because instead of having a single hyperplane in our system that tries to optimize its position, we have N_H hyperplanes optimizing their position together in the input pattern space, and one hyperplane optimizing its position in the hidden-unit (or, as coined in [21], hu–) space.

The same hyperplane eqns. (25)–(27) are used for the hidden hyperplanes and the output hyperplane. The only quantity that changes is the form of the particle charge. When two charges are equal distance away from the hidden hyperplane but one is on the wrong side, they do not necessarily have the same magnitude of charge. It is also possible for charges to flip sign.

The charges of the particles p in the input space with respect to the hidden hyperplane j, are given by

$$q_j^p = m_o \, (h_o^p \vec{\zeta}^p . \vec{n}_o) \, (\vec{n}_o . \vec{e}_j), \tag{32}$$

where $\vec{e}_j$ represents the unit vector in the direction of the dimension of hidden space defined by the hidden node j with decision hyperplane j, and where $\vec{\zeta}^p$ is the position vector of the corresponding (produced by the same input pattern $\vec{\xi}^p$) particle in the hidden space. h_o represents the range function for the output hyperplane, and m_o its mass. It can be seen that the right hand side is the same as what was referred to as the "energy" caused by the particle p in the output space with respect to the output hyperplane, multiplied by a term basically informing as to the relative senses of this hidden hyperplane and this output hyperplane. We notice too, that the mass of the output hyperplane shows itself here: an output hyperplane with a high mass transmits back a corrective charge to the hidden representation that is correspondingly large. It is the *recoil force* on a particle, from a hyperplane that itself is not able to move due to its large mass. It becomes strikingly clear now how hidden representations are used to solve problems: The output hyperplane(s) try to solve as much as possible through their own positioning, but a few particles will be left on the wrong sides because the problem is not linearly separable. When the majority of particles have been correctly sided, the mass of the output hyperplanes will increase anyway, leaving the incorrectly sided patterns no chance to move them. However, a larger output hyperplane mass means a greater magnitude of charge than the input particle would otherwise have, that allows them to produce more force on the hidden hyperplanes to change their configuration in order to be able to separate the hidden particles with the output hyperplanes. One can easily see how such procedures can be continued stepwise up as many hidden layers as the network has, with the picture emerging of the higher up one is in the hidden layer hierarchy (i.e. the nearer the input layer), the fewer the number of particles with high charges (since more and more are mapped correctly as one propagates backwards).

This all is so wonderfully logical that one should not be surprised to discover that exactly this technique of problem solving in feed-forward networks has been implemented explicitly "by hand" by a number of researchers. Such methods use so-called "constructive" or "growth" algorithms, either through explicit use of node-objects [12, 6, 5], or pure mathematical abstractions [11]. They tend to be more efficient than back-propagation in performing the necessary mappings, but they do not have the other favorable non-linear properties found in a net trained using back-propagation, namely the history information contained in the hyperplane masses and "easy-deformation" directions (see section 9), and tend to be confined to static training sets. It may be discovered that these failings cause such systems to have less-favorable generalization properties, because the overlaps in the hidden representations are not as great as possible (although a brief analysis of the Cascade Correlation [5] method in [20] seems to indicate that it should not degrade the possible generalization too severely). The back-propagation method, on the other hand, uses the easy directions of hyperplane/particle movement, defined by the overlap properties of the patterns, to find the solution, and one would therefore expect good generalization properties.

The problems we mentioned in section 5 with regard to "deadlocking", are overcome when a hidden layer is present, by the fact that two wrongly mapped patterns on opposite sides with orthogonal position vectors will pass back opposite charges, and (thanks to the "sense" term in eqn. (32)) will encourage their representations in the hidden space to be reversed. Deadlock will now not occur when the two particles are on the same side of the *hidden* hyperplane: the particles themselves are being moved, and not the output hyperplane! The sense of the movement in the input space is to separate them by pulling one nearer the hidden hyperplane and the other further away, such as to swap their current representations in the hidden space. (Deadlock can still occur, but it is more unlikely, since the particles would need to be equally placed on opposite sides of the hidden hyperplane too, and also for about half the population of hidden hyperplanes.)

For the many-output-unit case, the only change needed is to perform a summation in eqn. (32) over the output hyperplanes for the mass and potential terms.

7 A learning scenario

In order to bring the above analysis more to life, we consider a typical *successful* learning scenario for the back-propagation algorithm. It has an optimal starting point when the initial hyperplane vectors are properly chosen from a random distribution such that a good selection of orientations and positions of hyperplanes spans the unit hypercube.

Hint 1 (initializations) *Let the hidden nodes have a range* $\in [minHid, maxHid]$, *and the input nodes* $\in [minIn, maxIn]$. *The output weights should be initialized such that for each output node weight vector* $\vec{w} = (w_1, w_2 \ldots, w_{N_H})$ *its magnitude* $|\vec{w}| \lesssim 4/d_o$, *and the threshold weight* $w_0 \approx \frac{d_o}{2}\frac{(\vec{w}.\vec{b})}{|\vec{w}|}$, *where* d_o *is the diameter* $d_o := \sqrt{N_H}\,|maxHid - minHid|$, *and* $\vec{b}$ *is the vector* $\frac{1}{\sqrt{N_H}}(1, 1, \ldots, 1)$. *The values for the components of* $\vec{w}$ *should be chosen from a uniform random distribution. The hidden weights should* *be initialized such that for each hidden node weight vector $\vec{w} = (w_1, w_2 \ldots, w_n)$ its magnitude $|\vec{w}| \lesssim 4/d_h$, and the threshold weight w_0 lies in $minIn * |\vec{w}| < w_0 < maxIn * |\vec{w}|$ where d_h is the diameter $d_h := \sqrt{n}\,|maxIn - minIn|$, and the dimension of input space is n.*

The importance of initial weight configurations has been often noted, see for example [8]. With our representation of the system it is easier to see what kind of a difference they can make. The above hint ensures that the output hyperplanes start off somewhere near the middle of the hidden space, and have a range of orientations. The hidden hyperplanes are organized so that they too have a range of orientations, but may also be positioned anywhere within the largest hypersphere that can be drawn in the input space. In all cases the masses of the hyperplanes are initialized so that the peaks in the range curve shown in figure 3 cover all regions in the spaces. This initialization gives the system a high temperature to start with.

In order that such initializations may be approximately employed in real programs without too much difficulty, the following values are suggested for the case with 1-hidden layer networks with input and hidden nodes in the range $[-1, 1]$:

- Randomly initialize the weights, choosing values from a uniform distribution.
- Normalize the weights from input layer to each hidden node to have magnitude $\frac{2}{\sqrt{N_H}}$, and those from hidden layer to each output node to have magnitude $\frac{2}{\sqrt{n}}$.
- Initialize the thresholds of the hidden nodes uniformly in the range $\left[-\frac{\sqrt{n}}{2}, \frac{\sqrt{n}}{2}\right]$.
- Set the thresholds of the output nodes to 0.

Assume such a random distribution has been chosen according to Hint 1. Let the problem involve one output unit, and let there be enough hidden units present to solve the problem. Since we start with small values of γ the hyperplanes are easy to turn and move. The initial hidden unit representations $\vec{\zeta}^p$ should be closely clustered in the center of the unit hidden-space hypercube. The output hyperplane slices the initial cluster of hidden representations.

Initially, the potential is broad, encompassing all the patterns, and the light hidden representations will feel little recoil force. The first stage of optimization will be the output hyperplane orienting such as to get as many patterns as possible on the correct side. It begins to settle to an optimal position, determined by *all* the patterns (since they all have about the same importance at this stage, and are all about the same distance from the hyperplane) by an unweighted average. It then begins to get more massive, due to the forces from correctly sided patterns in range giving repulsion (eqn. (26)), and the variation in ϕ becoming small, and consequently the incorrectly sided patterns will begin to experience larger recoil forces. These negative forces, and the positive (but now smaller) ones from correctly sided patterns, then control the dynamics of the hidden-layer hyperplanes. Patterns still on the wrong side of the output hyperplane when it is becoming quite massive, will receive stronger forces that demand reorientation of the *hidden* hyperplanes. This is more favorable, since there is less chance now of the output hyperplane being reoriented.

The recoil forces on the pattern representations affect the hidden hyperplanes like the squared error forces on the output hyperplanes, and the hyperplane seeks an optimal position and orientation, defined by minimizing the sum of forces from all directions against the hyperplane (to end with little turn and shift). That is the least squares approximation, or curve-fitting procedure mentioned in section 5. Then the hidden hyperplanes themselves may begin to grow in mass. As they get more massive, the patterns that are correctly sided will decay in terms of charge and importance (eqn. (32)), and those incorrectly sided will grow in charge and importance. In an optimal case the hidden hyperplanes will not grow too massive until all the patterns are correctly sided. We return to this point in the next two sections. If the pattern is too far away, and practically alone in being incorrectly mapped, it has too little effect on the mass change of the hyperplane, and local minima can occur.

The last stage in the optimization is when all the patterns are on the correct sides of hyperplanes, and their masses are increased, cooling the system. A cold system is characterized from outside as being one that gives more and more boolean outputs (and inside, boolean hidden representations). If all the patterns are correctly mapped then not only will the system be cool but it will have zero charge, where the charge of the system may be defined as the sum of the magnitudes of charges of all the particles (or all the particles in the last hidden layer—it is equivalent).

8 Local minima

Is it possible to commence cooling down before all patterns are correctly sided? This is in fact one of the hyperplane dynamics' criteria for getting stuck in a local minimum. If the cooling begins too early (i.e. before all particles are correctly sided) the final state will be an energy minimum, but not result in a solution to the mapping, because the system still has a charge. The other criterion is to do with the *symmetry* of the problem. Both criteria are easy to see from eqns. (25)–(27):

- If $h^p \to 0$ for patterns $p \in \mathcal{U}$ then the set of patterns $\mathcal{U}$ will not be in range of the hyperplane(s) in question and will therefore not influence their positioning.
- If, for the whole set of patterns $p \in \mathcal{S}$, symmetry conditions exist such that $\vec{\omega} \approx 0$, and $\frac{d\vec{x}}{dt} \approx 0$, then the situation is a deadlock, and the system will not be able to cool down, because $\frac{dm}{dt} = \sum_p E_p \approx 0$. Such an occurrence is possible, if, at the beginning, the set of hyperplane masses is reasonably small (as encouraged in Hint 1), then $|h^r| \approx |h^s|;\ \forall r, s \in \mathcal{S}$, and the hyperplanes are

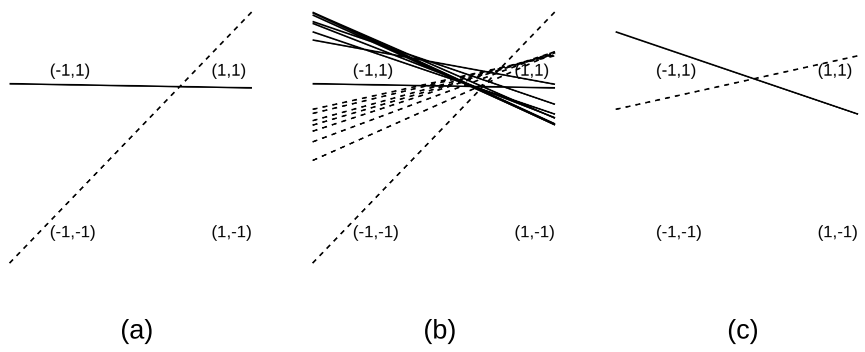

Figure 5: Evolution of an XOR local minimum. (a) shows the initial configuration of the 2 hyperplane network, with the 4 possible patterns of the 2-dimensional problem in the input space. In (b) the evolution of the hyperplane positions is shown, and (c) shows the final configurations. Motion of both hyperplanes is continuously clockwise.

> initialized *outside* the pattern arena (occurs when the θs are too large), and $\sum_p q_p \approx 0$. Then the hyperplane may never (or excruciatingly slowly) be pulled into the pattern arena, and hence hyperplane mass can through eqn. (26) never be increased to remove the symmetry in the ranges h^p.

The second case is an unstable equilibrium, and can be avoided either through non-batch learning or when the weights are suitably initialized, using Hint 1. (In incremental (or per-sample) learning, the summations in the dynamics equations are removed and the dynamics is considered for a single particle each time.) The first case is more pernicious, as can be demonstrated in the following with a simple XOR example (see [16]).

The input hyperplanes start off as shown in figure 5a, and batch learning moves them, smoothly, as indicated in figure 5b. Because of the initial configuration, although it is acceptable to Hint 1, patterns $(1,1)$, $(-1,1)$ and $(-1,-1)$ have great influence on the shifting of the hyperplanes, whereas $(1,-1)$ has little. Patterns $(1,1)$ and $(-1,-1)$ wish to remain on the same side of the dotted hyperplane, and pattern $(-1,1)$ wishes to remain on the other side, hence the movement of this hyperplane is slowed down, and mass increase begins. The only hyperplane that can separate pattern $(1,-1)$ from the $(-1,-1)$ has been similarly used to separate $(1,1)$ from $(-1,1)$ and $(1,-1)$, and it too has become more massive. The two massive hyperplanes now feel no influence from the unfortunate pattern $(1,-1)$, and the system cools down.

Such situations can be *cured* (but not in general initially avoided) by reducing the mass of one of the hyperplanes. For example, the situation in figure 5c was taken, and the dotted hyperplane had its mass reduced by a factor of 10. Then the system was allowed to learn further, and the local minimum was nowhere to be seen. The system iterated smoothly to a result with the solid hyperplane separating the same patterns as before, and the dotted one separating $(-1,1)$ and $(1,-1)$ from $(-1,-1)$.

Hint 2 (Escaping from local minima) *Since local minima are often caused through premature mass increase of hyperplanes, one way of freeing the system is to reduce the mass of* one *hyperplane leading from the input to the hidden layer. The amount should be to reduce m to about $2/\sqrt{n}$ (as in the initialization Hint 1).*

Avoidable early cooling can be caused in the following ways.

- The parameter g (η) may be too large, causing the output hyperplanes, or the hidden particles, to be knocked too far away, and the range of the interaction is then not broad enough to include the particles which have been as a result incorrectly sided. There are various methods for altering η during the learning so that dangerous shifts may be avoided (see, for example, [18, 4, 23], among many others).
- It is possible that the "momentum" parameter may have been used. The momentum parameter (see [16], or any back-propagation description) actually damps the rotation and shifting of the hyperplanes so that they do not oscillate. When momentum is too large, however, the hyperplanes can be severely constrained in their rotation and shifting, and as a result cooling can easily set in too early. The main effect of momentum is to accelerate the mass increase procedure. It is wise, therefore, to arrange the hyperplanes correctly before employing momentum. This may be monitored using the "plasticity" parameter, explained in the next section.
- The chances of it occurring can depend on the actual initial positions, orientations and masses of the hyperplanes, as described above. To summarize, if the masses are too large in the beginning, the range of the potential is already too small for many particles to have an effect on the positioning of the hyperplane, and cooling may begin as soon as the nearby particles have been sided. Add to that bad initial positioning (so that many particles are anyway far away), and it is clear things can seize up quite early on.

Notice from eqn. (26) too that when no change in orientation is needed, or the patterns that require it are too far away to be of influence, then the larger m becomes, since the dot product in eqn. (26) is at a maximum when the cross product in eqn. (25) is at a minimum.

Some problems may be of such a type that they may be represented by superposition of a much easier problem and a small (in terms of input patterns) also easy *perturbing* problem. For example one oddly mapped pattern may be hidden in a great region of regularly mapped patterns. Then the forces of the larger problem will be acted on first, and those from the perturbing problem may be just so small that they will be ignored until sharpening has reduced the repulsive forces from the correctly sided patterns. Then it may be too late. One such example is the "$= 1$" problem [13]. All patterns with one bit on must be mapped to 1, and the rest to 0. The main problem (in terms of interaction strength) here is the separation of all the > 1–bit patterns from the 1–bit patterns. After this is successfully done, there remains a single pattern, $(0, 0, 0 \ldots, 0)$, that has to be mapped. This is the perturbation. What is seen is a rapid learning of all the other patterns (steep drop in learning curve) followed by a long slightly sloping portion, as the nearest free hyperplane is dragged over to perform the final separation. It takes a long time because the hyperplane is already quite massive, due to many successfully mapped patterns cooling the system.

In such exceptional circumstances, the minimal solution (defined in [13]) consists of one hyperplane separating just this single pattern from the rest of the patterns, in the input layer. The best way to avoid the slow learning then is either before learning to realize that the single pattern is an exception, or through the learning itself discover that it is an exception, in that it is the only pattern that has not been correctly mapped. Then this pattern should be isolated, and learned using an extra hyperplane after all the others have been learned. It was frequently (in simulations) a successful policy for the "$= 1$" problem, to let the system solve all the patterns apart from the exception (which it automatically does), and then, when it was clear that cooling had set in, choose any hidden hyperplane and reduce its mass by about a factor 10. Further learning had the effect of using this suddenly mobile hyperplane to learn the remaining pattern. All the other patterns were in fact reasonably well accounted for by the other hyperplanes, which had merely to shift slightly to account for the loss of the (now) lighter hyperplane. The effect of such controlled "interfering" was startling in such experiments: long plateaux of steady error suddenly dropped in a matter of a few cycles to zero—a very clear demonstration of the effect of hyperplane mass.

The hint here is then more one of structured learning, or discovering *how to* structure learning through the performance on the training set. Such cases are recognized by: heavy hyperplanes, just one pattern incorrectly mapped. It is very difficult to get the pattern learned for larger dimensions of input space, because it has such a relatively insignificant effect on the hyperplane movements. Thus the best advice here is to stage the learning. The constructive algorithms use this philosophy [5, 12, 6]). As mentioned before, however, they may not be appropriate when the training set is not fixed: What happens then if we wish to perform *additional* learning using the weights of a trained network? Is this possible without destroying previous knowledge?

9 Plastic learning

The receptivity of the network to adaptation can be indicated by its *plasticity*. We define network plasticity P for a one-hidden-layer network as follows:

$$P := N_H \left(\sum_{h \in \mathcal{H}} m_h \right)^{-1} \tag{33}$$

which, in words, is the inverse of the average mass of the hidden hyperplanes ($\mathcal{H}$ is the set of hidden hyperplanes).

9.1 Significance of hyperplane mass in the system

The increase of a hyperplane mass is very significant for the system. The anchoring of a hyperplane in this way is equivalent to the long-term storing of information. Related to the Shannon information measure, the probability of a particle being on one or the other side of a hyperplane can be taken to be its resultant activation, and this becomes more boolean as the mass of the hyperplane increases. This results in the information content of the system increasing, as the order increases, and as the plasticity decreases.

To increase the plasticity of a network, reduce the mass of the hyperplanes. Reducing the mass of the hyperplanes does not change the separation of the patterns in the hidden layer, and can easily be reversed. However, reducing the mass of the hyperplanes also allows the network to learn new things easier, for they may better move to accommodate the new patterns. One should however be cautious when altering hyperplane masses: long-term memories are only held in the network by anchoring the hyperplanes through these very masses. The relative orientations of the hyperplanes provides the pattern correlation information, which, together with the relative hyperplane masses, allows weak or strong generalizations to be made on unseen patterns. Reducing hyperplane masses makes the particle mappings more uncertain and consequently reduces the Shannon information content of the system, making it more amenable to change. Networks are normally very good at gracefully degrading with respect to new learning, precisely because of the heavy hyperplanes they evolve.

It is suggested below how this evidently useful knowledge of network storage may be employed to implement a more controlled relearning.

9.2 Gentle learning

The picture of the lone pattern pulling heavy hyperplanes over to it because it got "forgotten" or "put aside" earlier when hyperplanes were light, in the "=1" example, can help us in deciding how to enable destructive relearning to be partially avoided, when this lone pattern is in fact additional information.

There are three cases to consider.

1. The new pattern represents an example of the rule already self-contained in previous learned patterns, and adds no new information other than perhaps a more exact placement of decision regions.

2. The new pattern provides new information, that can nevertheless be learned when the hyperplanes already placed are softly rearranged (by this is meant the whole problem does not have to be relearned from scratch).

3. The new pattern can only be learned when a completely new configuration of the hyperplanes is created, or cannot be learned together with the current patterns in the current network.

Interesting for us are the first two cases. In these cases, to minimize the small destructive effect of the learning the following should be done: The output hyperplane does not need to be moved. Increasing its mass, however, will not damage the other pattern mappings, assuming they were all properly mapped before. For the type 1 new pattern it might be sufficient to increase the mass of the output hyperplane by a small amount. If this does not help, some hidden representation improvement is required. This is tackled by altering the hidden representation of the pattern component-wise, by dealing with hyperplane movements singly. The hidden hyperplane with respect to which the hidden particle in question has the greatest charge should be moved according to the appropriate forces, and the others should be held *fixed*. Alternatively, its particular mass could be reduced, changes made to all hyperplanes, and then the mass returned to its original value. The second method is probably kinder to the relative representations. After every movement the progress of the pattern is monitored. This continues until the charge for this hyperplane is no longer the greatest, at which point the hyperplane for which it is, is taken and moved. This goes on, together with output hyperplane mass-increases attempts, until the pattern is mapped. In this method the *favored directions* of change in the system were emphasized in order to reduce the damage to the information currently in the system. A direction of change is towards one of the hyperplanes, and it is favored if the particle charge is high with respect to it.

Of course, it may not always be possible to learn a new pattern without damaging the network's past information, and the above merely outlines heuristic ideas of how to ease the effect of a new pattern; but how do we *know* that a pattern is of type one, two or three, i.e. what is a destructive pattern?

9.3 When is a pattern destructive?

Again, the spin model helps us to decide. A pattern will have a destructive effect on a network when the movements it wishes to enforce on the network hyperplanes are large, but unable to be communicated because the ranges are too low, and masses too high. The logic behind this reasoning is the following. Take an output node, and see if the pattern is correctly or incorrectly mapped at this point. If it is incorrectly mapped, and indeed with a large error, then the magnitude of the charge of the hidden representation particle will be correspondingly large (eqn. (23)). Depending on the position of the particle on the range curve, however, and the mass of the output hyperplane in question, is the *ease* with which this hyperplane may be moved to satisfy the particle's desire. The correlation of a great attraction (charge) for a hyperplane, and a low range to enforce it (related to the high hyperplane mass) together with a heavy hyperplane to move, signifies probable destructive behavior. Destructive learning means large alterations in hyperplane structures, which is seen as large movements of heavy hyperplanes. A heavy hyperplane holds more information than a light one, and its movement thus destroys information easier.

Thus destruction can be foreseen by considering the charges of the particles and the force ranges available to them, with respect to each hyperplane. This of course is valid not only in the output hyperplane system, but also the hidden one. It is possible that a strongly destructive intention in the output hyperplane system may be avoided by transferring it to a less damaging hidden hyperplane movement, via the charge in eqn. (32). That is, instead of moving the output hyperplane the particle in the hidden layer is moved.

The following hint gives advice on how to use this knowledge in real networks.

Hint 3 (Destructiveness) *Completely learning a new pattern r is likely to be destructive, if*

(a) the magnitude of the largest error in the output layer is larger than 0.8 AND

(b) the hidden hyperplane that has the largest charge of particle associated with it causes it to be mapped to the wrong side and to an activation a^r outside $0.2 < a^r < 0.8$.

In Hint 3 learning is taken to be destructive when hyperplanes are massive (otherwise the pattern could not cause the node values near the limits of the activation function) and the pattern is on the wrong side of an output hyperplane, and is too far away to pull it over easily (condition (a)), and cannot change its hu–space representation easily to rectify this gently (condition (b)). In the many-output unit case, it may happen that the representation can be changed easily in a direction specified by all the output influences together, and this can be seen only by considering the charges generated, to discover the hyperplane that would be most influenced. The values 0.2 and 0.8 are chosen because they lie outside the linear area of the sigmoid function.

This hint uses the knowledge that a non-plastic network is hard to change, but when the changing is performed in particular directions it may be done non-destructively.

Related to the concept of destructiveness, is the idea that particular patterns may be more *easily* learned than others. These patterns will be ones that cause the least reordering in the system. Thus patterns that result in significant charges only in limited numbers of directions in the network would be such easy ones. A pattern demanding movements in all the hyperplanes around about it will be not so easy to learn. This is most easily seen when one considers a multi-output node network. If, on applying a pattern, the output `[0.8,0.7,0.7,0.3,0.75]` is obtained, when the target is `[1.0,1.0,1.0,0.0,0.0]`, then it is clear that much less resorting is necessary in the network as when the target were `[0.0,1.0,0.0,1.0,0.0]`. Such a pattern would then be easier to learn than if it had the second target. The quantity that is greater in the first than in the second is the *acceptability* of the pattern. It is different from the inverse of the squared error in that it is defined by correctness of the *relative* activations at the output nodes. Hence `[0.6,0.6,0.6,0.4,0.4]` would be as acceptable as `[0.8,0.8,0.8,0.2,0.2]`. This type of understanding of network pattern suitability is used in [19] for the automatic evolution of modular experts, that *choose* for themselves, on which patterns they would be better specialized, *before* they actually do any weight-changes on the particular pattern in question.

10 Extension possibilities

We have been considering just the one type of decision boundary: a hyperplane. But it is possible also to have a closed (localized) decision boundary, as for example when using radial-basis functions [14]. Here the response function f_j at a node j is given by

$$f_j = \exp\left(\frac{\sum_i(\xi^i - w_j^i)^2}{2\left(\sigma_j^i\right)^2}\right) \tag{34}$$

where the summation is over all the components of the pattern representation.

One can imagine the region defined by the radial-basis function to be equivalent to a hyperplane "wrapped around on itself" so that inside a hyperellipsoidal region the charge is opposite to outside. The dynamics works now in a similar way. The potentials are different, but the decision boundary still has a mass (determined now by the gaussian spread parameter σ_j^i in the response function), and energy, temperature and plasticity can still be defined. There is no longer a rotational component of motion, but there is a translational component represented by variation of the centroid defined by the $\{w_j^i\}$.

One can see immediately that learning using these response functions should be easier in the sense that there are less components of motion to vary: a pattern is either inside the hyperellipsoid or outside it, turning the whole thing has limited effect.

It should also be mentioned here, that it is easy to extend the one-output unit network considered in the above analyses to many-output networks. It involves merely summations over all output hyperplanes in eqn. (32) to generate a net charge. The averaging going on here to produce coordinated movement of hidden hyperplanes (so-called "moving-targets" [5] behavior) so that several patterns may be learned together in the same network is important for later generalization success.

A further point to be cleared up is what in this analysis might be valid when *online* or *per-sample* learning is used. In these cases the summations over patterns p in eqns. (25)–(27) are non-existent, each weight-change being performed every time a pattern is fed through the network. There are several schemes one might use for this type of updating, which depend on how well one would like to learn a particular pattern before going on to the next one. Thus one can simply update the weights once as each pattern comes by (small-steps online), or at the other extreme learn each pattern perfectly before going on to the next one. When small steps online is used, this approximates closely to the standard batch method (update after accumulating forces from all patterns) insofar that a global gradient descent is followed. One can visualize this in the hyperspin model as small tugs and pushes of the hyperplanes, each mini-movement of the hyperplane not affecting the vector forces from following spins terribly much, so that the position and mass of the hyperplane at the end is close to what it would have been if the forces were all added and then applied.

On the other hand, if large-steps online learning is employed, a hyperplane can be significantly moved and its mass significantly altered, for every consecutive pattern. The theory then tells us that such things as deadlocks are no longer possible, since forces from one pattern are not going to be cancelled by opposite forces from another pattern. Local minima in the sense of before are also no longer to be seen, but similar impasses are still possible. Instead of sitting in one area of search-space, not able to find better solutions, and at the same time not having the information to escape (as in a standard local minimum), the impasses are characterized by learning-unlearning oscillations. Quite independent of the order of presentation of the patterns, but not altering the selection probabilities, one can find one or two patterns that can never be learned. Of course, when the pattern is in the network it can be learned, but the area of search-space in which the system finds itself may be such that the resulting system state after the pattern is learned is unstable (too many other patterns are unlearned in the process), and the system "unlearns" and falls back to a lower state that is more favorable for most of the other patterns. The oscillations are seen in the hyperspin model as occasional non-plastic tugs of hyperplanes followed by their return to the original position.

How might such local minima be escaped? In a similar way to that described in section 8, one looks to increase the plasticity of the system in such a way that a new area of search-space may be explored, but not to destroy previous important hyperplane placements. One may then safely reduce the mass of a single chosen hyperplane and see if the system may be freed, by allowing it to turn, but not increase in mass. After every pattern is learned the hyperplane has its mass again reduced. Globally the scale of this reduction should decrease in time, so that the system is allowed gradually to decrease its plasticity through time. This is a form of simulated annealing [7] or error-surface deformation [22], seen from the point of view of hyperplanes.

Another way to escape is to add a hyperplane as suggested in [2]. The hyperplane should be initialized as in Hint 1.

11 Conclusion

In this paper it has been demonstrated how multi-layer perceptron learning using non-localized response functions, in particular sigmoid functions, and the back-propagation algorithm, can be interpreted usefully in terms of hyperplane spin dynamics and cooling of the system. The parameters describing rotation, translation and mass of the hyperplane were used for this interpretation. We believe use of hyperplane mass has, at least not explicitly, been employed before in any description of back-propagation. Normally it is mentioned that "weights just get ever larger" without really understanding how important this is. We are aware of one technique [9] that makes use of something similar to our mass, namely the sigmoid gain parameter β mentioned in section 2.

The interpretation was used to explain how local minima can occur, and how they may be avoided, either through correct weight initializations, or greater understanding about what larger values of learning parameters can provoke, or re-assessment of the actual problem to be learned with respect to its complexity. Advice was also given as to how to escape from certain types of local minima. Plasticity of a network was defined, and a way of gentler relearning in non-plastic networks was suggested. It is useful to know whether, in a trained, non-plastic network, further learning of a new pattern will be very destructive. Using once more the hyperplane spin analogy, this was viewed in terms of easy and hard rearrangements of the system, and a rough idea given for how it might be recognized in practice.

This type of detailed network analysis is worthwhile, since not only can the problems during learning and generalization be better understood and dealt with, and performance improved, but it also allows a number of suitable network state and learning parameters, such as plasticity, pattern destructiveness and acceptability, to be defined, and used advantageously in the systematic construction of modular network systems. The need for a certain accessibility to complex networks is even greater when it is wished to develop successful hierarchical groupings of them.

12 Acknowledgements

The author would like to thank Jörg Kindermann, Alexander Linden and Christoph Tietz for reading the manuscript, and fellow researchers in the GMD Adaptive Systems groups for generally thought-provoking (and time-consuming!) discussions.

References

[1] S. Ahmad and G. Tesauro. A study of scaling and generalization in neural networks. In *Abstracts of the First Annual Meeting of the INNS, supplement to Neural Networks, Vol 1*, Boston, Mass., September 1988.

[2] T. Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4), 1989.

[3] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[4] S.E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, 1988.

[5] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgen Kaufmann, 1990.

[6] M. Frean. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, 1:198–209, 1990.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[8] J. F. Kolen and J. B. Pollack. Back propagation is sensitive to initial conditions. *Complex Systems*, 4:269–280, 1990.

[9] J. K. Kruschke and J. R. Movellan. Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks. *IEEE Transactions on Systems, Man and Cybernetics*, 21(1), 1991.

[10] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, April 1987.

[11] O.L. Mangasarian. Multisurface method of pattern separation. *IEEE Transactions on information theory*, IT-14(6):801–807, 1968.

[12] M. Mézard and J-P. Nadal. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, 22(12):2191–2203, 1989.

[13] H. Mühlenbein. Limitations of multilayer perceptrons – steps towards genetic neural networks. *Parallel Computing*, 14(3):249–260, 1990.

[14] S. Renals and R. Rohwer. Phoneme classification experiments using radial basis functions. In *Proceedings of the IJCNN–89*, Washington, 1989. IEEE.

[15] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1959.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Nature*, 323(533), 1986.

[17] A. Schütte. How can multi-layer perceptrons classify. In *Proceedings of the Workshop on Distributed Adaptive Neural Information Processing*, GMD, St Augustin, April 1989. Oldenbourg Verlag.

[18] F. J. Śmieja. *Learning and generalization in feed-forward neural networks*. PhD thesis, Edinburgh University, Department of Physics, September 1989. Unpublished.

[19] F. J. Śmieja. Multiple network systems (MINOS) modules: Task division and module discrimination. In *Proceedings of the 8th AISB conference on Artificial Intelligence, Leeds, 16–19 April, 1991*, 1991.

[20] F. J. Śmieja. Neural network constructive algorithms: Trading generalization for learning efficiency? Technical report, Gesellschaft für Mathematik und Datenverarbeitung, St Augustin, Germany, November 1991.

[21] F. J. Śmieja and H. Mühlenbein. The geometry of multilayer perceptron solutions. *Parallel Computing*, 14:261–275, 1990.

[22] F. J. Śmieja and G.D. Richards. Hard learning the easy way: Backpropagation with deformation. *Complex Systems*, 2(4), 1988.

[23] Tom Tollenaere. SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks*, 3:561–573, 1990.