# Playing Billiard in Version Space

Pál Ruján*
*Fachbereich 8 Physik and ICBM, Postfach 2503*
*Carl-von-Ossietzky Universität*
*26111 Oldenburg, Germany*

**Abstract**

A ray-tracing method inspired by ergodic billiards is used to estimate the theoretically best decision rule for a given set of linear separable examples. For randomly distributed examples the billiard estimate of the single Perceptron with best average generalization probability agrees with known analytic results, while for real-life classification problems the generalization probability is consistently enhanced when compared to the maximal stability Perceptron.

## 1 Introduction

Neural networks can be used for both concept learning (classification) and for function interpolation and/or extrapolation. Two basic mathematical methods seem to be particularly adequate for studying neural networks: geometry (especially combinatorial geometry) and probability theory (statistical physics). Geometry is illuminating and probability theory is powerful.

In this paper I consider the perhaps simplest neural network, the venerable Perceptron [1]: given a set of examples falling in two classes, find the

*e-mail:rujan@neuro.uni-oldenburg.de

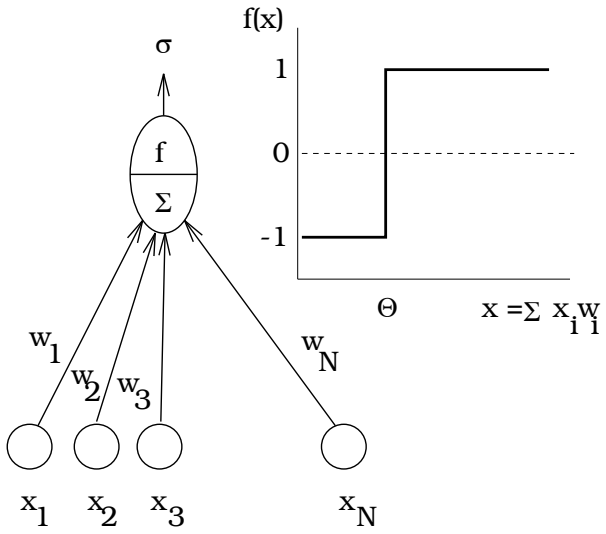


Figure 1: *The Perceptron as neural network*

linear discriminant surface separating the two sets, if one exists. In this context, my goal is twofold: 1) give the optimal (or Bayesian) decision theory a geometric interpretation and 2) use that geometric content for designing a deceivingly simple method for computing the single Perceptron with the best possible generalization probability.

As shown in Figure 1, a *Perceptron* is a network consisting of $N$ (binary or real) inputs $x_i$ and a single binary output neuron. The output neuron sums first all inputs with the corresponding synaptic weights $w_i$, then performs a threshold operation according to

$$\sigma = sign(\sum_{i=1}^{N} x_i w_i - \theta) = sign(\vec{x}\vec{w} - \theta) = \pm 1 \qquad (1)$$

In general, the synaptic weights and the threshold can be arbitrary real numbers. A network where $\vec{w}$ has only binary components is called a *binary* Perceptron. The output unit has binary values $\sigma = \pm 1$ and labels the class to which the input vector belongs. Both the weight vector $\vec{w} = (w_1, w_2, \ldots, w_N)$ and the threshold $\theta$ should be learned from a set of $M$ examples whose class is known (the training set).

The Perceptron and its variants [2] are among the few networks for which basic properties like the maximal information capacity, (how many *random*[1] examples can be faultlessly stored in the network), or the classification error of certain learning algorithms, (how many examples are needed in order to achieve a given generalization probability), can be obtained analytically. Such calculations are done by defining first a learning protocol. For example, one assumes that the examples are independently and identically sampled from some stationary probability distribution, that one has access to a very large set of such examples for both training and testing purposes, that no matter how many examples one generates, there is always a single Perceptron network solving the problem without errors, etc. Almost all statistical mechanical calculations require also the *thermodynamic limit* $N \to \infty,\ M \to \infty,\ \alpha = M/N = const.$ These assumptions are needed for technological reasons - some integrals must be concretely evaluated - but also because otherwise the problem is mathematically not well defined.

In real life, however, one is confronted to situations which do not satisfy some of these assumptions: one has a relatively small set of examples, their distribution is unknown, often not stationary, the examples are not independently sampled. In a strict sense, such problems are mathematically not quantifiable. However, even a small set of examples contains *some* information about the nature of the problem [3]. The basic question is then, among all possible networks trained on this set, which one has *possibly* the best generalization probability? As explained below, understanding the geometry of the version space leads to the correct answer and to algorithms for computing it.

Our protocol assumes that both the training and test examples are drawn independently and identically from the distribution $P(\vec{\xi})$. The simplest way to generate such a rule is to define a *teacher* network with the same structure as shown in Fig. 1, providing the correct class label $\sigma = \pm 1$ for each input vector. Given a set of $M$ examples $\{\vec{\xi}^{(\alpha)}\}_{\alpha=1}^{M},\ \vec{\xi}^{(\alpha)} \in \mathbb{R}^N$ and their corresponding binary class label $\sigma_\alpha = \pm 1$ the task of the learning algorithm is to find a *student* network which mimics the teacher by choosing the network parameters such as to classify correctly the training examples. Eq. (1) implies that the learning task consists of finding a hyperplane $(\vec{w}, \vec{x}) = \theta$ separating

[1] By *random* vectors we mean in the following vectors sampled independently and identically from a constant distribution defined on the surface of the unit sphere $(\vec{x}, \vec{x}) = 1$.

the positive from the negative labeled examples $\vec{x} \in \{\vec{\xi}^{(\nu)}\}_{\nu=1}^{M}$:

$$\begin{aligned} (\vec{w}, \vec{\xi}^{(\alpha)}) &\geq \theta_1, \ \sigma_\alpha = 1 \\ (\vec{w}, \vec{\xi}^{(\beta)}) &\leq \theta_2, \ \sigma_\beta = -1 \\ \theta_1 &\geq \theta \geq \theta_2 \end{aligned} \tag{2}$$

The typical situation is that either none or infinitely many such solution exist. The vector space whose points are the vectors $(\vec{w}, \theta)$ is called the *version space* and its subspace satisfying Eq. (2) the *solution polyhedron.*

Most often one wishes to minimize the average probability of class error for a new $\vec{x}$ vector drawn from $P(\vec{\xi})$. In other situations another network, which in average makes more errors but avoids very 'bad' ones, is preferable. Yet another problem occurs when the data provided by the teacher is noisy [5]. In what follows we shall restrict ourselves to the 'standard' problem of minimal average classification error.

In theory, one knows that for a given training set the optimal Bayes decision [4] implies an average over all $\{\vec{w}, \ \theta\}$ solutions satisfying Eq. (2). Since each solution is perfectly consistent with the training set, in the absence of any other a priori knowledge, one must consider them as equally probable. This is not the case, for instance, in the presence of noise. The best strategy is then to associate with each point in the version space a Boltzmann weight whose energy is the squared error function and whose temperature depends on the noise strength [5].

For examples independently and identically drawn from a constant distribution ($P(\vec{\xi}) = const$) Watkin has shown that in the thermodynamic limit the single Perceptron corresponding to the the center of mass of the solution polyhedron has the same generalization probability as the Bayes-decision [6]. On the practical side, known learning algorithms like Adaline [7, 8] or the maximal stability Perceptron (MSP) [9, 10, 11] are not Bayes-optimal. In fact, if all input vectors have the same length, then the maximal stability Perceptron network corresponds to the center of the largest hypersphere inscribed into the solution polyhedron, as shown in Section 4.

There have been several attempts at developing learning algorithms approximating the Bayes-decision. Watkin used a randomized AdaTron algorithm to sample the version space [6]. More recently, Bouten *et al* defined a class of convex functions whose minimum lies within the solution polyhedron. By changing the function's parameters one can obtain a minimum

which is a good approximation of the known Bayes solution [12]. This idea is somewhat similar to the notion of an 'analytic center of a convex polytope' introduced by Sonnevend [13] and used extensively for designing fast linear programming algorithms [14]. A very promising but not yet fully exploited approach [15] uses the Thouless-Anderson-Palmer (TAP) equations originally developed for the spin-glass problem.

In this paper I propose a quite different method, based on an analogy to classical, ergodic billiards. Considering the solution polyhedron as a dynamic system, a long trajectory is generated and used to estimate the center of mass of the billiard. The same idea can be applied also to other optimization problems.

Questions related to the theory of billiards are briefly considered in Section 2. Section 3 sets up the stage by presenting an elementary geometric analysis in two dimensions. The more elaborated geometry of the Perceptron version space is discussed in Section 4. An elementary algorithmic implementation for open polyhedral cones and their projective geometric closures are summarized in Section 5. Numerical results and a comparison to known analytic bounds and to other learning algorithms can be found in Section 6. Our conclusions and further prospects are summarized in Section 7.

# 2 Billiards

A *billiard* is usually defined as a closed space region (compact set) $\mathcal{P} \in \mathbb{R}^N$ dimensions. The boundaries of a billiard are usually piecewise smooth functions. Within these boundaries a point mass (ball) moves freely, except for the elastic collisions with the enclosing walls. Hence, the absolute value of the momentum is preserved and the phase space $\mathcal{B} = \mathcal{P} \times \mathcal{S}^{N-1}$, is the direct product of $\mathcal{P}$ and $\mathcal{S}^{N-1}$, the surface of the $N$-dimensional unit velocity sphere. Such a simple Hamiltonian dynamics defines a flow and its Poincaré map an automorphism. The mathematicians have defined a finely tuned hierarchy of notions related to such dynamic systems. For instance, simple ergodicity as implied by the Birkhoff-Hincsin theorem means that the average of any integrable function defined on the phase space over a single but very long trajectory equals the spatial mean (except for a set of zero measure). Furthermore, integrable functions invariant under the dynamics must be constant. From a practical point of view this means that almost all very

long trajectories will cover the phase space uniformly. Properties like mixing (Kolmogorov-mixing) are stronger than ergodicity. They require the flow to mix uniformly different subsets of $\mathcal{B}$. In hyperbolic systems one can go even further and construct Markov partitions defined on symbolic dynamics and eventually prove related central limit theorems.

Not all convex billiards are ergodic. Notable exceptions are ellipsoidal billiards, which can be solved by a proper separation of variables [16]. Already Jacobi knew that a trajectory started close to and along the boundaries of an ellipse cannot reach a central region bounded by the so-called caustics.

In addition to billiards which can be solved by a separation of variables, there are a few other exactly soluble polyhedral billiards [17]. Such solutions are intimately related to the reflection method - the billiard tiles perfectly the entire space. A notable example is the equilateral triangle billiard, first solved by Lamé in 1852. Other examples of such integrable billiards can be obtained by mapping exactly soluble one dimensional many particle system into a one-particle high-dimensional billiard [18].

Apart from these exceptions, small perturbations in the form of the billiard usually destroy integrability and lead to chaotic behaviour. For example, the stadium billiard (two half-circles joined by two parallel lines) is ergodic in a strong sense, the metric entropy is non-vanishing [19]. The dynamics induced by the billiard is hyperbolic if at any point in phase space there are both expanding (unstable) and shrinking (stable) manifolds. A famous example is Sinai's reformulation of the Lorentz-gas problem. Deep mathematical methods were needed to prove the Kolmogorov-mixing property and in constructing the Markov partitions for the symbolic dynamics of such systems [20].

The question whether a particular billiard is ergodic or not can be decided in principle by solving the Schrödinger problem for a free particles trapped in the billiard-box. If the eigenfunctions corresponding to the high energy modes are roughly constant, then the billiard is ergodic. Only few general results are known for such quantum problems. In fact, I am not aware of theoretical results concerning the ergodic properties of convex polyhedral billiards in high dimensions. If all angles of the polyhedra are rational, then the billiard is weakly ergodic in the sense that the velocity direction will reach only rational angles (relative to the initial direction). In general, as long as two neighboring trajectories collide with the same polyhedral faces, their distance will grow only linearly. Once they are far enough as to collide with

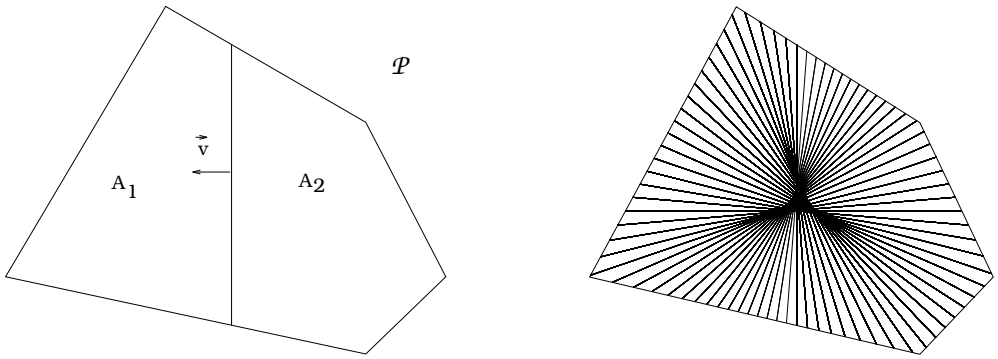


Figure 2: *a) Halving the area along a given direction and b) the resulting Bayes lines*

different faces of the polyhedron, their distance will abruptly increase. Hence, except for very special cases with high symmetry, it seems unlikely that high dimensional convex polyhedra as generated by the training examples will fail to be ergodic.

# 3 A simple geometric problem

For the sake of simplicity let us illustrate our approach in a two-dimensional setting. In the next Section we will show how the concepts developed here generalize to the more challenging Perceptron problem.

Let $\mathcal{P}$ be a closed convex polygon and $\vec{v}$ a given unit vector, defining a particular direction in the $\mathbb{R}^2$ space. The direction of vector $\vec{v}$ can be also described by the angle $\phi$ it makes with the $x$-axis. Next, construct the line perpendicular to $\vec{v}$ which halves the *area* of the polygon $\mathcal{P}$, $A_1 = A_2$, as illustrated in Fig. 2a. We will show in the next Section that this geometric construction is analogue to the Bayes decision in version space. Choosing a set of vectors $\vec{v}$ oriented at different angles leads to the set of 'Bayes lines' seen in Fig. 2b. It is obvious that these lines do not intersect at one single point.

The same task is computationally not feasible in a high dimensional version space. It is certainly more economical to compute and store a single point $\vec{r}_0$, which represents optimally the full information contained in Fig.

2b. As evident from Fig. 2b, the majority of lines lines passing through $\vec{r}_0$ will not partition $\mathcal{P}$ in equal areas but will make some mistakes, denoted by $\Delta A = A_1 - A_2 \neq 0$. $\Delta A$ depends on both the direction of $\vec{v}$ and on the polygon $\mathcal{P}$, $\Delta A = \Delta A(\vec{v}, \mathcal{P})$. Different optimality criteria can be formulated, depending on the actual application. The usual approach would correspond to minimizing the squared area-difference averaged over all possible $\vec{v}$ directions:

$$\vec{r}_0 = \text{arg min} \left\langle (\Delta A)^2 \right\rangle = \text{arg min} \int (\Delta A)^2(\vec{v}) p(\vec{v}) d\vec{v} \tag{3}$$

where $p(\vec{v})$ is some a priori known distribution of vectors $\vec{v}$. Another possible criterion optimizes the worst case loss over all directions:

$$\vec{r}_1 = \text{arg inf sup} \left\{ (\Delta A)^2 \right\}, \tag{4}$$

etc. In what follows the point $\vec{r}_0$ is called the *Bayes point*.

The calculation of the Bayes point according to Eq. (3) is computationally feasible but a lot of computer power is still needed. A good estimate of the Bayes-point is given by the center of mass of the polgon $\mathcal{P}$:

$$\vec{S} = \frac{\int_{\mathcal{P}} \vec{r} \rho(\vec{r}) dA}{\int_{\mathcal{P}} \rho(\vec{r}) dA} \tag{5}$$

where $\rho(\vec{r}) = const$ is the surface mass density.

In Table I we presents exact numerical results for the $x$ and $y$ coordinates of both the Bayes point and the center of mass. The center of mass is an excellent *approximation* of the Bayes point. In very high dimensions, as shown by T. Watkin [6], $\vec{r}_0 \to \vec{S}$.

A polyhedron can be described either by the list of its vertices or by the set of vectors normal to its facets. The transition from one representation to another requires exponential many aritmetic operations as a function of dimension. Therefore, in typical classification tasks this transformation is not practicable.

For 'round' polygons (polyhedra), the center of the smallest circumscribed and of the largest inscribed circle (sphere) is a good choice for approximating the Bayes point in the vertex and the facet representation, respectively. Since in our case the polyhedron generalizing $\mathcal{P}$ is determined by a set of normal vectors (facet representation), only the largest inscribed circle (sphere) is a feasible approximation, see Fig. 3. The numerical values of the $(x_R, y_R)$ coordinates of the center point are displayed in Table I.

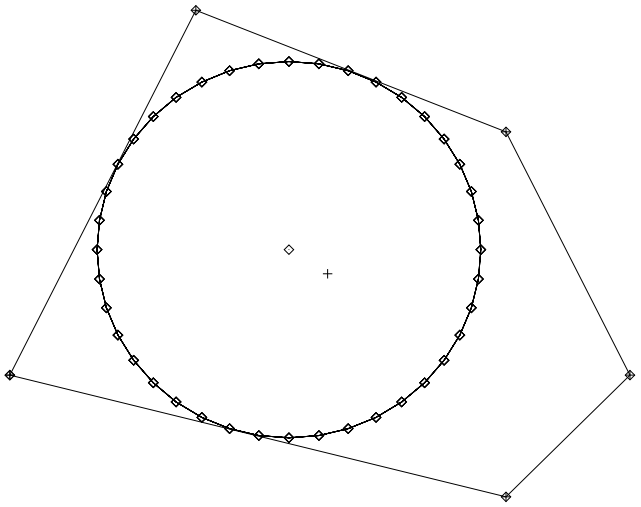

Figure 3: *The largest inscribed circle. The center of mass (cross) is plotted for comparison.*

A better approximation would be to compute the center of the largest volume inscribed ellipsoid [21], a problem also common in nonlinear optimization [22]. The best known algorithms are of order $O(M^{3.5})$ operations, where $M$ is in our case the number of examples. Additional logarithmic factors have been neglected, for details see [22].

The purpose of this paper is to show that a reasonable estimate of $\vec{r}_0$ can be obtained faster by following the (ergodic) trajectory of an elastic ball *inside* the polygon, as shown in Fig. 4a for four collisions. Fig. 4b shows the trajectory of Fig. 4a from another perspective, by performing an appropriate reflection on each collision edge. A trajectory is periodic if after a finite number of such reflections the polygon $\mathcal{P}$ is mapped onto itself. Fully integrable systems correspond in this respect to polygons which will fill without holes the whole space (that this geometric point of view applies also to other fully integrable systems is nicely exposed in [23]).

If the dynamics is ergodic in the sense discussed in Section 2, then a long enough trajectory should cover without holes the surface of the polygon. By computing the total center of mass of the trajectory one should then obtain a

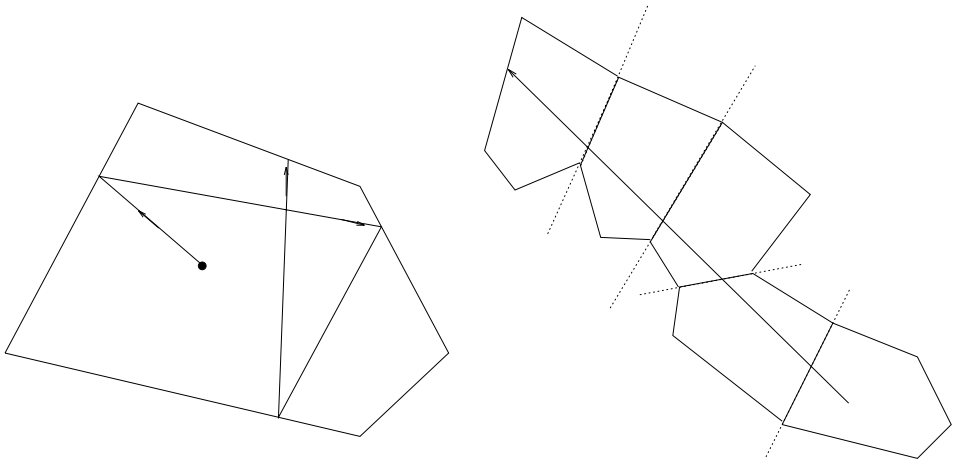

Figure 4: *a) A trajectory with four collisions and b) its 'straightened' form*

good estimate of the center of mass. The question whether a billiard formed by a 'generic' convex polytope is ergodic or not is to my knowledge not solved. Extensive numerical calculations are possible only in low dimensions. The extent to which the trajectory covers $\mathcal{P}$ after 1000 collisions is visualized in Fig. 5. By continuing this procedure, one can convince oneself that all holes are filled up, so that the trajectory will visit every point inside $\mathcal{P}$.

The next question is whether the area of $\mathcal{P}$ is *homogeneously* covered by the trajectory. The numerical results summarized in Table I were obtained by averaging over 100 different trajectories for each given length. As the length of the trajectory is increased, these averages converge to the center of mass. Also displayed are the empirical standard deviations $\sigma_{x,y}$ of the trajectory center of mass coordinates and the average squared area difference, $\langle \Delta A^2 \rangle$.

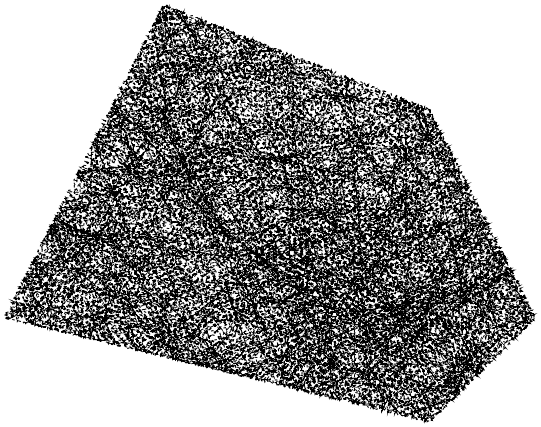

Figure 5: *A trajectory (shown as a dotted line) with 1000 collisions*

| Method | x | $\sigma_x$ | y | $\sigma_y$ | $\langle \Delta A^2 \rangle$ |
|---|---|---|---|---|---|
| Bayes-point | 6.1048 | | 1.7376 | | 1.4425 |
| Center of Mass | 6.1250 | | 1.6667 | | 1.5290 |
| Largest inscribed circle | 5.4960 | | 2.0672 | | 7.3551 |
| Billiard - 10 collisions | 6.0012 | 0.701 | 1.6720 | 0.490 | 7.0265 |
| Billiard - $10^2$ collisions | 6.1077 | 0.250 | 1.6640 | 0.095 | 2.6207 |
| Billiard - $10^3$ collisions | 6.1096 | 0.089 | 1.6686 | 0.027 | 1.6774 |
| Billiard - $10^4$ collisions | 6.1232 | 0.028 | 1.6670 | 0.011 | 1.5459 |
| Billiard - $10^5$ collisions | 6.1239 | 0.010 | 1.6663 | 0.004 | 1.5335 |
| Billiard - $10^6$ collisions | 6.1247 | 0.003 | 1.6667 | 0.003 | 1.5295 |

Table I: *The exact coordinates $(x, y)$ of the Bayes point for the polygon $\mathcal{P} = (1,0), (4,6), (9,4), (11,0), (9,-2)$ and its various estimates: center of mass, maximal inscribed circle, trajectories of different lengths*

# 4 The geometry of Perceptrons

Consider a set of $M$ training examples, consisting of $N$-dimensional vectors $\vec{\xi}^{(\nu)}$ and their class $\sigma_\nu$, $\nu = 1, \ldots, M$. Now let us introduce the $N+1$-

dimensional vectors $\vec{\zeta}^{(\nu)} = (\sigma_\nu \vec{\xi}^{(\nu)}, -\sigma_\nu)$ and $\vec{\mathbf{W}} = (\vec{w}, \theta)$. In this representation Eq. 2 becomes equivalent to a standard set of linear inequalities

$$(\vec{\mathbf{W}}, \vec{\zeta}^{(\nu)}) \geq \Delta > 0 \tag{6}$$

The parameter $\Delta = \min_{\{\nu\}}(\vec{\mathbf{W}}, \vec{\zeta}^{(\nu)})$ is called the *stability* of the linear inequality system. A bigger $\Delta$ implies a solution which is more robust against small changes in the example vectors.

The vector space whose points are the examples $\vec{\zeta}^{(\nu)}$ is the *example space*. In this space $\vec{\mathbf{W}}$ corresponds to the normal vector of a $N+1$ dimensional hyperplane. The *version space* is the space of the vectors $\vec{\mathbf{W}}$. Hence, a given example vector $\zeta$ corresponds here to the normal of a hyperplane. The inequalities (6) define a convex polyhedral cone whose boundary hyperplanes are determined by the training set $\left\{\vec{\zeta}^{(\nu)}\right\}_{\nu=1}^{M}$.

How can one use the information contained in the example set for making the best average prediction on the class label of a new vector drawn from the same distribution? Each new presented example $\vec{\zeta}^{(new)}$ corresponds to a hyperplane in version space. The direction of its normal is defined up to a $\sigma = \pm 1$ factor, the corresponding class.

The best possible decision for the classification of the new example follows the Bayes scheme: for each new (test) example generate the corresponding hyperplane in version space. If this hyperplane does not intersect the solution polyhedron, consider the normal to be positive when pointing to the part of the version space containing the solution polyhedron. Hence, all Perceptrons satisfying Eq. (6) will classify unanimously the new example. If the hyperplane cuts the solution polyhedron in two parts, point the normal towards the bigger half. Therefore, the decision which minimizes the average generalization error is given by evaluating the average measure of *pro* vs the average measure of *contra* votes of all Perceptrons making no errors on the training set.

We see that the Bayes decision is analogous to the geometric problem described in the previous Section. However, the solution polyhedral cone is either open or is defined on the unit $N+1$-dimensional hypersphere (see also Fig. 6b for an illustration).

The practical question is now to find simple approximations for the Bayes point. One possibility is to consider the Perceptron with *maximal stability*, defined as following:

$$\vec{\mathbf{W}}_{MSP} = \arg\ \max_{\vec{\mathbf{W}}}\ \Delta; \quad (\vec{\mathbf{W}}, \vec{\mathbf{W}}) = 1 \tag{7}$$

or, equivalently, as

$$\vec{w} = \arg\ \max_{\vec{w}}\ \{\theta_1 - \theta_2\}; \quad (\vec{w}, \vec{w}) = 1 \tag{8}$$

where $\theta = (\theta_1 + \theta_2)/2$ and $\Delta = (\theta_1 - \theta_2)/2$. The quadratic conditions $(\vec{\mathbf{W}}, \vec{\mathbf{W}}) = 1$ $[(\vec{w}, \vec{w}) = 1]$ are necessary because otherwise one could multiply $\vec{w}$ and $\theta_{1,2}$ by a large number, making $\theta_1 - \theta_2$ and thus $\Delta$ arbitrary large.

Eq. (8) has a very simple geometric interpretation, shown in Fig. 6. Consider the convex hulls of the vectors $\vec{\xi}^{(\alpha)}$ belonging to the positive examples $\sigma_\alpha = 1$ and of those in the negative class $\vec{\xi}^{(\beta)}$, $\sigma_\beta = -1$, respectively. According to Eq. (8), the Perceptron with maximal stability corresponds to the slab of maximal width one can put between the two convex hulls (the 'maximal dead zone' [24]). Geometrically, this problem is equivalent (dual) to finding the direction of the shortest line segment connecting the two convex hulls (the minimal connector problem). Since the dual problem minimizes a quadratic function subject to linear constraints, it is a *quadratic programming problem.* By choosing $\theta = \frac{\theta_1 + \theta_2}{2}$ (dotted line in Fig. 6 ) one obtains the maximal stability Perceptron (MSP).

The direction of $\vec{w}$ is determined by at most $N + 1$ vertices taken from both convex hulls, called *active constraints.* Fig. 6a shows a simple two dimensional example, the active constraints are labelled by A, B, and C, respectively. The version space is three dimensional, as shown in Fig. 6b. The three planes represent the constraints imposed by the examples A (left plane), B (right plane), and C (lower plane). The bar points from the origin to the point defined by the MSP solution. The sphere corresponds to the normalization constraint. If the example vectors $\vec{\xi}^{(\nu)}$ have all the same length $\mathcal{N}$, then Eqs. (6, 7) imply that the distance between the $\vec{\mathbf{W}}_{MSP}$ and the hyperplanes corresponding to active constraints are all equal to $\frac{\Delta_{\max}}{\mathcal{N}}$. All other hyperplanes participating in the polyhedral cone are further away. Accordingly, the maximal stability Perceptron corresponds to the center of the largest circle inscribed into the spherical triangle defined by the intersection of the unit sphere with the solution polyhedron, the point where the bar intersects the sphere in Fig. 6b.

A fast algorithm for computing the minimal connector requiring in average $O(N^2 M)$ operations and $O(N^2)$ storage place can be found in [11].

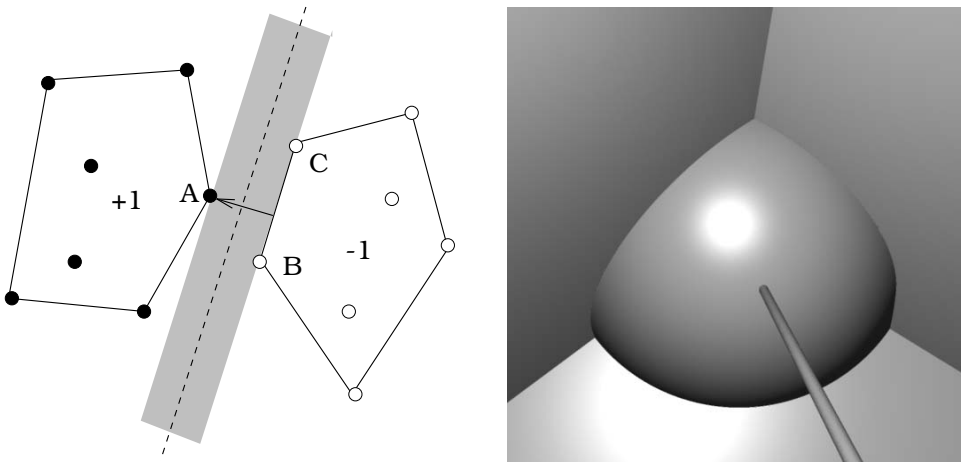


Figure 6: *a) The Perceptron with maximal stability in example space. b) The solution polyhedron (only the bounding examples A, B, and C are shown). See text for details.*

# 5 How to play billiard in version space

Each billiard game starts by first placing the ball(s) on the pool. This is not always a trivial task. In our case, the maximal stability Perceptron algorithm [11] does it or signals that a solution does not exist. The trajectory is initiated by generating at random a unit direction vector $\vec{\mathbf{v}}$ in version space.

The basic step consists of finding out where - on which hyperplane - the next collision will take place. The idea is to compute how much time the ball needs until it eventually hits each one of the $M$ hyperplanes. Given a point $\vec{\mathbf{W}} = (\vec{\mathbf{w}}, \theta)$ in version space and a unit direction vector $\vec{\mathbf{v}}$, let us denote the distance along the hyperplane normal $\vec{\zeta}$ by $d_n$ and the component of $\vec{v}$ perpendicular to the hyperplane by $v_n$. In this notation the flight time needed to reach this plane is given by

$$\begin{aligned} d_n &= (\vec{\mathbf{W}}, \vec{\zeta}) \\ v_n &= (\vec{\mathbf{v}}, \vec{\zeta}) \\ \tau &= -\frac{d_n}{v_n} \end{aligned} \tag{9}$$

After computing all $M$ flight times, one looks for the smallest positive $\tau_{min} = \min_{\{\nu\}} \tau > 0$. The collision will take place on the corresponding hyperplane.

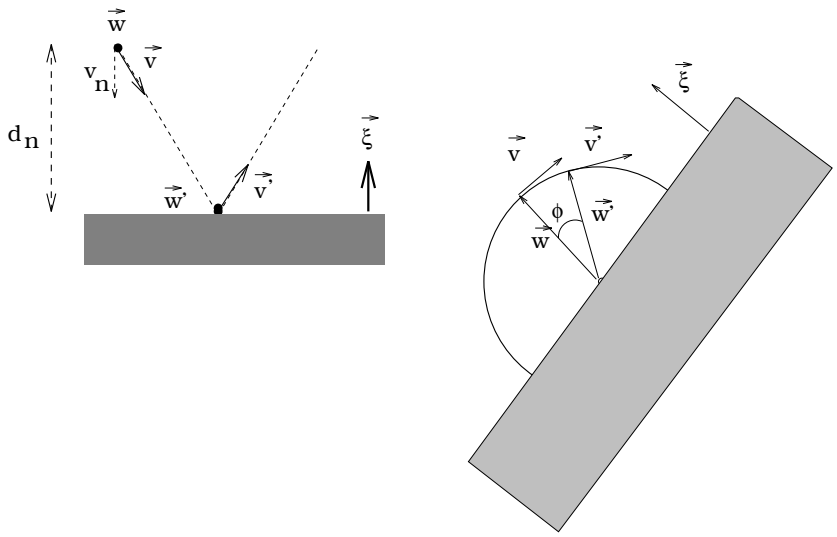


Figure 7: *Bouncing in version space. a) Euclidean b) spherical geometry.*

The new point $\vec{W}'$ and the new direction $\vec{v}'$ are calculated as

$$\vec{\mathbf{W}}' = \vec{\mathbf{W}} + \tau_{min}\vec{\mathbf{v}} \quad (10)$$

$$\vec{\mathbf{v}}' = \vec{\mathbf{v}} - 2v_n\vec{\zeta} \quad (11)$$

This procedure is illustrated in Fig. 7a. In order to estimate the center of mass of the trajectory one has first to normalize both $\vec{\mathbf{W}}$ and $\vec{\mathbf{W}}'$. By assuming a constant line density one assigns to the (normalized!) center of the segment $\frac{\vec{\mathbf{W}}'+\vec{\mathbf{W}}}{2}$ the length of the vector $\vec{\mathbf{W}}' - \vec{\mathbf{W}}$. This is then added to the actual center of mass - as when adding two parallel forces of different lengths. In high dimensions ($N > 5$), however, the difference between the mass of the full $N+1$ dimensional solution polyhedron and the mass of the bounding $N$ dimensional boundaries becomes negligible. Hence, we could as well just record the collision points, assign them the same mass density, and construct their average.

Note that by continuing the trajectory beyond the first collision plane one can sample also regions of the solution space where the $\vec{\mathbf{W}}$ makes one, two, etc. mistakes (the number of mistakes equals the number of crossed

boundary planes). This additional information can be used for taking an optimal decision when the examples are noisy [5].

Since the polyhedral cone is open, the implementation of this algorithm must take into account the possibility that the trajectory might escape to infinity. The minimal flight time becomes then very large, $\tau > \tau_{max}$. When this exception is detected, a new trajectory is started from the maximal stability Perceptron point in a yet another random direction. Hence, from a practical point of view, the polyhedral solution cone is closed by a spherical shell with radius $\tau_{max}$ acting as a special 'scatterer'. This *flipper* procedure is iterated until enough data is gathered.

If we are class conscious and want to remain in the billiard club, we must do a bit more. As explained above, the solution polyhedral cone can be closed by normalizing the version space vectors. The billiard is now defined on a curved space. However, the same strategy works also here if between subsequent collisions one follows geodesics instead of straight lines. Fig. 7b illustrates the change in direction for a small time step, leading the the well known geodesic differential equation on the unit sphere:

$$\dot{\vec{\mathbf{W}}} = \vec{\mathbf{v}} \tag{12}$$

$$\dot{\vec{\mathbf{v}}} = - \vec{\mathbf{W}} \tag{13}$$

The solution of this equation costs additional resources. Actually, the solution of the differential equation is strictly necessary only when there are no bounding planes on the actual horizon [2]. Once one or more boundaries are 'visible', the choice of the shortest flight time can be evaluated directly, since in the two geometries the flight time is monotonously deformed. Even so, the flipper procedure is obviously faster.

Both variants deliver interesting additional informations, like the mean escape time of a trajectory or the number of times a given border plane (example) has been bounced upon. The collision frequency classifies the training examples according to their 'surface' area in the solution polyhedron - a good measure of their relative 'importance'.

[2] Assuming light travels along Euclidean straight lines

# 6 Results and Performance

This Section contains the results of numerical experiments performed in order to test the billiard algorithm. First, the ball is placed inside the billiard with the maximal stability Perceptron algorithm as described in [11]. Next, a number of typically $O(N^2)$ collision points are generated. Since the computation of one collision point requires $M$ scalar products of $N$ dimensional vectors, the total load of this algorithm is $O(MN^3)$. The choice for $N^2$ collision points is somewhat arbitrary and is based on the following considerations. By using the billiard method one generates many collision points lying on the borders of the solution polyhedron. We could try to use this information for approximating the solution polyhedron with an ellipsoidal cone. The number of free parameters involved in the fit is of the order $O(N^2)$. Hence, at least a constant times that many points are needed. Such a fitted ellipsoid delivers also an estimate on the decision uncertainty. If one is not interested on this information, it is enough to monitor how one or more projections of the center of mass estimate changes as the number of collisions increases. Once these projections become stable the program can be stopped.

T. Watkin argues that the number of sampling points should be of $O(1)$ [6]. I find his arguments not convincing. For example, Fig. 8 shows how the average generalization probability changes as the number of collisions increases up to $N^2$ steps. Since finding a point within the solution polyheder is algorithmically equivalent to solving a linear programming problem, his method of sampling the version space with randomly started AdaTron gradient descent (or any other Perceptron learning method) requires at least $O(N^2M)$ operations per sampling point, compared to $O(NM)$/collison in the billiard method.

In a first test, a known 'teacher' Perceptron $\vec{T}$ was used to label the randomly generated examples for training. The generalization probability $G\alpha$, $\alpha = M/N$, was then computed by measuring the overlap between the resulting solution ('student') Perceptron with the teacher Perceptron:

$$G(\alpha) = 1 - \frac{1}{\pi}\cos^{-1}(\vec{T}, \vec{w}); \quad (\vec{T}, \vec{T}) = (\vec{w}, \vec{w}) = 1 \tag{14}$$

The numerical results obtained from 10 different realizations for $N = 100$ are compared with the theoretical Bayes-learning curve in Fig. 9. Fig. 10 shows a comparison between the billiard results and the maximal stability

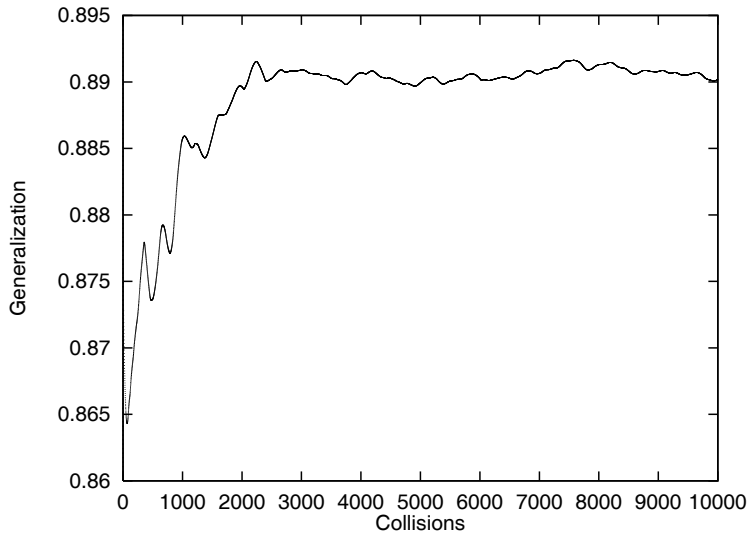


Figure 8: *Average of the generalization probability as a function of the number of collisions,* $N = 100$*,* $M = 1000$

Perceptron (MSP) results. Although the differences seem small compared to the error bars, the billiard solution was in all realizations consistently superior to the MSP.

Fig 11 shows how the number of constraints (examples) bordering the solution polyhedron changes with increasing $\alpha = \frac{M}{N}$.

As the number of examples increases, the probability of escape from the solution polyhedron decreases, the network reaches its storage capacity. Fig. 12 shows the average number of collisions before escape as a function of classification error, parameterized through $\alpha = \frac{M}{N}$. Therefore, by measuring either the escape rate or the number of 'active' examples we can estimate the generalization error *without* using test examples. Note, however, that such calibration graphs should be calculated for each specific distribution of the input vectors.

Randomly generated training examples lead to rather isotropic polyhedra, as illustrated by the small difference between the Bayes- and the MSP learning curve (see Fig. 10). Therefore, we expect that the billiard approach leads to bigger improvements when applied to real-life problems with strongly

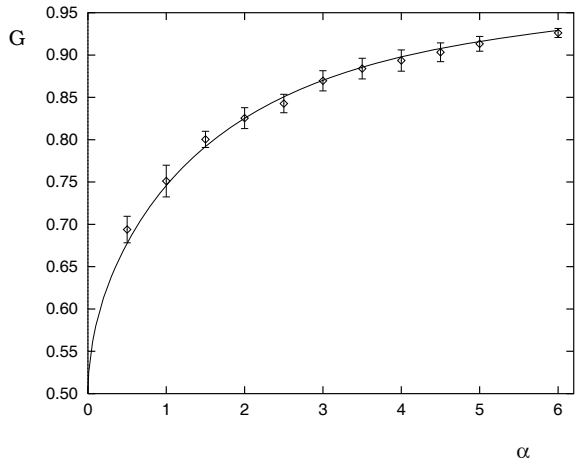


Figure 9: *The theoretical Bayes-learning curve (solid line) vs billiard results obtained in 10 independent trials.* $G(\alpha)$ *is the generalization probability,* $\alpha = \frac{M}{N}$, $N = 100$.

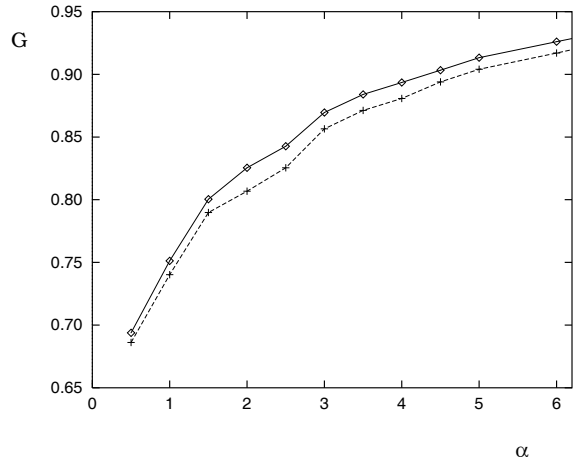


Figure 10: *Average generalization probability, same parameters as above. Lower curve: the maximal stability Perceptron algorithm. Upper curve: the billiard algorithm*

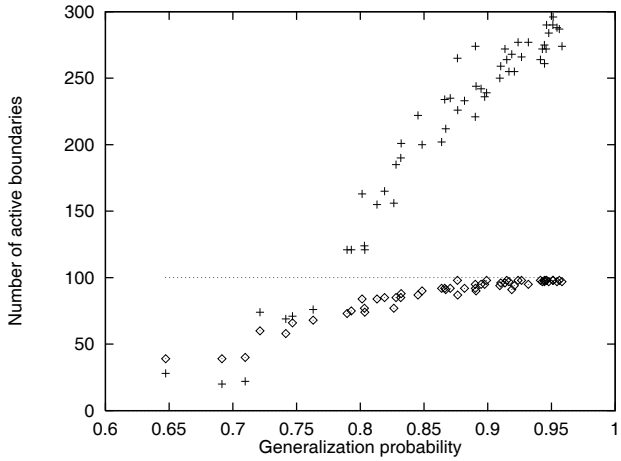


Figure 11: *Number of different identified polyhedral borders vs* $G(\alpha)$, $N = 100$. *Diamonds: maximal stability Perceptron, saturating at 101. Crosses: billiard algorithm.*

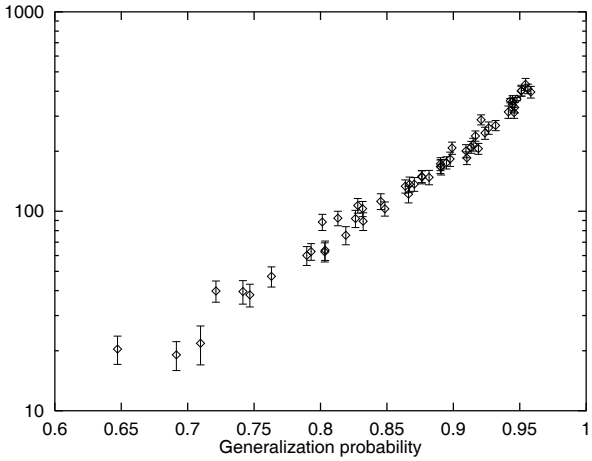


Figure 12: *Mean number of collisions before escaping the solution polyhedral cone vs* $G(\alpha)$, $N = 100$

anisotropic solution polyhedra. Similar improvements can be expected also when using constructive methods for multilayer Perceptrons which use iteratively the Perceptron algorithm [25]. Such procedures have been used, for example, for classifying handwritten digits [26]. For all such examples sets available to me, the introduction of the billiard algorithm on top of the maximal stability Perceptron leads to consistent improvements of up to 5% in classification probability.

A publicly available data set known as the sonar-problem [27, 28] considers the problem of deciding between rocks and mines from sonar data. The input space is $N = 60$ dimensional and the whole set consists of 111 mine and 97 rock examples. We go down the list of rock signals putting alternate members into the training and test sets; we do the same with the set of mine signals. Since the data sets are sorted by increasing azimuth, this gives us training and testing with equal lengths (104 signals) and with the population of azimuth angles matched as closely as possible. Gorman and Sejnowski consider a three layer feedforward neural network architecture with different number of hidden units, trained by the back-propagation algorithm [27]. Their results are summarized and compared to our results in Table II.

By applying the MSP algorithm we first found that the whole data (training + test) set is linearly separable. Second, by using the MSP on the training set we obtain a 77.5% classification rate on the test set (compared to 73.1% in [27]). Playing billiard leads to a 83.4% classification rate (in both cases the training set was faultlessly classified). This improvement amount is typical also for other applications. The number of active examples (those contributing to the solution) was 42 for the maximal stability Perceptron and 55 during the billiard.

By computing the maximal stability and the Bayes Perceptrons we did not use any information available on the test set. On the contrary, many networks trained by back-propagation are slightly 'adjusted' to the test set by changing network parameters - output unit biases and/or activation function decision bounds. Other training protocols also allow either such adjustments or generate a population of networks, from which a 'best' is chosen based on test set results. Although such adaptive behavior might be advantageous in many practical applications, it can be misleading when trying to infer the real capability of the trained network.

In the sonar problem, for instance, we know that a set of Perceptrons

separating faultlessly the whole data (training + test) set is included in the version space. Hence, one could use the billiard or other method to find it. This would be an extreme example of 'adapting' our solution to the test set. Such a procedure is especially dangerous when the test set is not 'typical'. Since in the sonar problem the data was divided in two equal sets, by exchanging the roles of the training and test sets one would expect similar quality results. However, in this case much weaker results (73.3% classification rate) are obtained. This shows that the two sets do not contain the same amount of information about the common Perceptron solution.

| Hidden Units | % Right on Training set | Std. Dev. | % Right on Test Set | Std. Dev. |
|---|---|---|---|---|
| 0 | 79.3 | 3.4 | 73.1 | 4.8 |
| 0–MSP | 100.0 | - | 77.5 | - |
| 0–Billiard | 100.0 | - | 83.4 | - |
| 2 | 96.2 | 2.2 | 85.7 | 6.3 |
| 3 | 98.1 | 1.5 | 87.6 | 3.0 |
| 6 | 99.4 | 0.9 | 89.3 | 2.4 |
| 12 | 99.8 | 0.6 | 90.4 | 1.8 |
| 24 | 100.0 | 0.0 | 89.2 | 1.4 |

Table II. *Results for the sonar classification problem from [23]. 0-MSP is the maximal stability Perceptron, 0-Billiard is the Bayes billiard estimate.*

Looking at the results of Table II it is hard to understand how 104 training examples could substantiate the excellent *average* test set error of $90.4 \pm 1.8\%$ for networks with 12 hidden units (841 free parameters). The method of structural risk minimization [9] uses uniform bounds for the generalization error in networks with different VC-dimensions. To firmly establish a classification probability of 90% one needs about ten times more training examples already for the linearly separable class of functions. A network with 12 hidden units has certainly a much larger capacity and requires that many more examples. One could argue that each of the 12 hidden units has solved the problem on its own and thus the network acts as a committee machine. However, such a majority decision should be at best comparable to the Bayes estimate.

# 7 Conclusions and Prospects

The study of dynamic systems led already to many interesting practical applications in time-series analysis, coding, and chaos control. The elementary application of Hamiltonian dynamics presented in this paper demonstrates that the center of mass of a long dynamic trajectory bouncing back and forth between the walls of the convex polyhedral solution cone leads to a good estimate of the Bayes-decision rule for linearly separable problems. Somewhat similar ideas have been recently applied to constrained nonlinear programming [30].

Although the geometric view presented in this paper overemphasizes the role played by extremal (active) vertices of the example set and faultlessly trained Perceptrons, it is not difficult to make these estimates more robust. For example, some of the extremal examples can be removed to test - and improve - the stability of the MSP. Similarily, the solution polyhedron can be expanded to include solutions with non-zero training errors, allowing for learning noisy examples.

Since the Perceptron problem is equivalent to the linear inequalities problem Eq. (6), it has the same algorithmic complexity as linear programming. The theory of convex polyhedra plays a central role both in mathematical programming and in solving $\mathcal{NP}$-hard problems such as the traveling salesman problem [29, 31].

Viewed from this perspective, the ergodic theory of convex polyhedral billiards might provide new, effective tools for solving different combinatorial optimization problems [21]. A big advantage of such ray-tracing algorithms is that they can be run in parallel by following up several trajectories at the same time.

The success of further applications depends, however, on methods of making such simple dynamics strongly mixing. On the theoretical side more general results, applicable to large classes of convex polyhedral billiards are called for. In particular, a good estimate of the average escape (or typical mixing) time is needed in order to bound the average behavior of future 'ergodic' algorithms.

## Acknowledgments

I thank Manfred Opper for the analytic Bayes data and Bruno Eckhardt for discussions on billiards. For a LaTeXoriginal of this paper with PostScript figures and a C-implementation of the billiard algorithm point your browser at http://www.icbm.uni-oldenburg.de/~rujan/rujan.html.

## References

[1] F. Rosenblatt: The perceptron: a probabilistic model for information storage and organization in the brain, *Psychological Review* **65** 386

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

[3] G. Pólya: Mathematics and Plausible Reasoning, Vol II, Patterns of Plausible Inference, Second Edition, Princeton University Press, 1968

[4] M. Opper and D. Haussler: Generalization performance of Bayes optimal classification algorithm for learning a Perceptron, *Physical Review Letters* **66** (1991) 2677

[5] M. Opper and D. Haussler: Calculation of the learning curve of Bayes optimal classification algorithm for learning a Perceptron with noise, contribution to *IVth Annual Workshop on Computational Learning Theory (COLT91)*, Santa Cruz 1991, Morgan Kaufmann, San Mateo, 1992, pp 61-87

[6] T. Watkin: Optimal learning with a neural network, *Europhysics Letters* **21** (1993) 871

[7] B. Widrow and M. E. Hoff: Adaptive switching circuits, *1960 IRE WESCON Convention Record*, New York, IRE, 96

[8] S. Diederich and M. Opper: Learning of Correlated Patterns in Spin-Glass Networks by Local Learning Rules, *Physical Review Letters* **58** (1987) 949

[9] D. Vapnik: Estimation of Dependencies from Empirical Data, Springer Verlag, 1982 - see Addendum I

[10] J. Anlauf and M. Biehl: The AdaTron: an adaptive Perceptron algorithm, *Europhysics Letters* **10** (1989) 687

[11] P. Rujan: A fast method for calculating the Perceptron with maximal stability, *Journal de Physique (Paris) I* **3** (1993) 277

[12] M. Bouten, J. Schietse, and C. Van den Broeck: Gradient descent learning in Perceptrons: a review of possibilities, *Physical Review E* **52** (1995) 1958

[13] Gy. Sonnevend: New algorithms based on a notion of 'centre' (for systems of analytic inequalities) and on rational extrapolation - in K.H. Hoffmann, J.-B. Hiriart-Urruty, C. Lemarechal and J. Zowe, Eds. *Trends in Mathematical Optimization - Proceedings of the 4th French-German Conference on Optimization, Irsee 1986* ISNM, Vol. **84** Birkhauser, Basel, 1988

[14] P. M. Vaidya: A new algorithm for minimizing a convex function over convex sets, in Proceedings of the 30th Annual FOCS Symposium, Research Triangle Park, NC, 1989 (IEEE Computer Society Press, Los Alamitos, CA, 1990) 338-343

[15] M. Opper: ' Bayesian Learning ', talk at the *Neural Networks for Physicists 3*, Minneapolis, 1993

[16] J. Moser: Geometry of quadrics and spectral theory, in *The Chern Symposium*, Y. Hsiang *et al* (Eds.), Springer-Verlag 1980, p. 147

[17] J. R. Kuttler and V. G. Sigillito: Eigenvalues of the Laplacian in two dimensions , *SIAM Review* **26** (1984) 163

[18] H.R. Krishnamurthy, H.S. Mani, and H.C. Verma: Exact solution of Schrödinger equation for a particle in a tetrahedral box, *Journal of Physics A* **15** (1982) 2131

[19] L. A. Bunimovich: On the ergodic properties of nowhere dispersing billiards, *Communications in Mathematical Physics* **65** (1979) 295

[20] L. A. Bunimovich and Y. G. Sinai: Markov partitions for dispersed billiards, *Communications in Mathematical Physics* **78** (1980) 247

[21] P. Rujan: Ergodic billiards and combinatorial optimization, in preparation

[22] L. G. Khachiyan and M. J. Todd: On the complexity of approximating the maximal inscribed ellipsoid for a polytope, *Mathematical Programming* **61** (1993) 137

[23] B. Sutherland: An Introduction to the Bethe Ansatz, in *Exactly Solvable Problems in Condensed matter and Relativistic Filed Theory*, B.S. Shastry, S.S. Jha and V. Singh, (Eds), Lecture Notes in Physics **242**, Springer-Verlag, 1985, 1

[24] P. F. Lampert: Designing pattern categories with extremal paradigm information, in *Methodologies of Pattern Recognition*, M. S. Watanabe (Ed.), Academic Press, NY, 1969, p 359

[25] M. Marchand, M. Golea, and P. Ruján: Convergence Theorem for Sequential Learning in Two Layer Perceptrons, *Europhysics Letters* **11** (1989) 487

[26] S. Knerr, L. Personnaz, and G. Dreyfus: Single layer learning revisited: a stepwise procedure for building and training a neural network, in *Neurocomputing*, F. Fogelman and J. Hérault (Eds), Springer Verlag 1990

[27] R.P. Gorman and T. J. Sejnowski: Analysis of hidden units in a layered network trained to classify sonar targets, in *Neural Networks* **1** (1988) 75

[28] S. E. Fahlman's collection of neural network data: included in the public domain software *am-6.0*.

[29] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, and D.B. Shmoys (Eds): *The Traveling Salesman Problem*, John Wiley, NY, 1984

[30] T. F. Coleman and Yuying Li: On the convergence of interior-reflective Newton methods for nonlinear minimization subject to bounds, *Mathematical Programming* **67** (1994) 189

[31] M. Padberg and G. Rinaldi: Optimization of a 532-city symmetric traveling salesman by branch and cut, *Oper. Res. Lett.* **6** (1987) 1 and A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, IASI preprint R. 247, 1988