A Learning-Theory Approach to Sensor Networks

Supervised learning might be a viable approach to sensor network applications. Preliminary research shows that a well-known algorithm from learning theory effectively applies to environmental monitoring, tracking of moving objects and plumes, and localization.

Slobodan N. Simić
University of California, Berkeley

Recent advances in microelectromechanical systems, computing, and communication technology have sparked the emergence of massively distributed, wireless sensor networks with potentially thousands of nodes. Each node can sense the environment, process the collected data, and communicate with its peers or to an external observer. A steady increase in these networks' capabilities and decrease in the cost of producing them have made possible applications that seemed too expensive or unrealistic.[1] On the other hand, these changes create great theoretical challenges in distributed signal processing, control, and other areas.

My colleagues and I in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley, propose a unified approach to various sensor network applications, using *supervised learning*. Supervised learning refers to *learning from examples*, in the form of *input-output pairs*, by which a system that isn't programmed in advance can estimate an unknown function and predict its values for inputs outside the training set. In particular, we examined *random wireless sensor networks*, in which nodes are randomly distributed in the region of deployment. When operating normally, nodes communicate and collaborate only with other nearby nodes (within communication range). However, a base station—with a more powerful computer on board—can query a node or group of nodes when necessary and perform data fusion. We mainly considered the sensor network platform developed at UC Berkeley (http://webs.cs.berkeley.edu), but our approach applies to a broader framework.

The problem of learning from data becomes crucial in a world flooded with too much information. Examples of this abound in biology, particle physics, astronomy, engineering, and other disciplines. Instrumenting the physical world with wireless sensors will generate massive amounts of data (examples) that can be processed (learned from) efficiently and usefully. This is where learning theory comes into play.

Learning techniques have been applied in many diverse scenarios. We considered some basic concepts of learning theory and how they might address the needs of random wireless sensor networks. Our research is still in progress, so our ideas and results are preliminary; however, we offer a beneficial perspective for the sensor network community to consider.

Basic notions of learning theory

The work of Tomaso Poggio,[2] Steve Smale,[2,3] and Felipe Cucker[3] was important to our understanding of learning theory. Vladimir Vapnik's fundamental work[4] is also an excellent resource on statistical learning.

In supervised learning, input-output pairs represent a sample of the object (such as a function) to be learned. To illustrate how systems are trained with these, consider the classic example of learning a physical law by curve fitting to data.

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/03/$17.00 © 2003 IEEE

Suppose that the physical law to be learned is an unknown function $f: \mathbb{R} \to \mathbb{R}$ ($\mathbb{R}$ denotes the set of real numbers) of a special form. Assume, for instance, that f is a polynomial of degree N, with coefficients $w_0, \ldots, w_N$. Let $(x_1, y_1), \ldots, (x_m, y_m)$ $(m > N)$ be samples of f, where we obtain y_i by "measuring" f at x_i. If the measurements were exact, we would have $y_i = f(x_i)$, but in reality we expect them to be affected by noise. Learning f amounts to finding the vector of coefficients $w = (w_0, \ldots, w_N)$ such that

$$f_w(x) = \sum_{i=0}^{N} w_i x^i$$

is the best fit for the data. We can do this efficiently by the Gaussian method of least squares, which computes w to minimize

$$\sum_{i=1}^{m} (f_w(x_i) - y_i)^2 .$$

Often (such as in the context of random wireless sensor networks), the x_i are random rather than chosen.

In general, we are given a set of data $(x_1, y_1), \ldots, (x_m, y_m)$, where the x_i's are in some *input space* X, usually a closed subset of some $\mathbb{R}^k$, and the y_i's belong to an *output space* Y, which we assume to be $\mathbb{R}$. For data generated by a sensor network, x_i is the position of the ith sensor node and y_i is its measurement or some function of it. I'll explain this in more detail later.

On the space $X \times Y$ consisting of all input-output pairs (x, y), with $x \in X$ and $y \in Y$, we assume a probability measure ρ exists that governs the sampling.[5] That is, pairs (x_i, y_i) are randomly picked according to ρ. The measure ρ defines a function $f_\rho: X \to Y$, called the *regression function* of ρ. For each input x, $f_\rho(x)$ is the average (with respect to ρ) value of the output y in $\{(x, y) : y \in Y\}$. The goal is then to learn (that is, find a good approximation of) f_ρ from random samples $(x_1, y_1), \ldots, (x_m, y_m)$.

In other words, we'd like to find a function $f: X \to Y$ that minimizes the *error*

$$\int_X (f - f_\rho)^2 d\rho_X$$

where ρ_X is the *marginal measure* on X induced by ρ.[5]

The central question of learning theory is how well f *generalizes*—that is, "how well it estimates the outputs for previously unseen inputs."[2]

The key algorithm

We use an algorithm—the key algorithm from the works mentioned earlier[2,3]—that fits the training data set $(x_1, y_1), \ldots, (x_m, y_m) \in X \times Y$ with a function $f: X \to Y$, where X is a closed subset of some $\mathbb{R}^k$ and $Y = \mathbb{R}$.

A function $K: X \times X \to \mathbb{R}$ is *symmetric* if $K(x, x') = K(x', x)$, for all $x, x' \in X$. It's *positive definite* if, in addition, for any finite subset $\{p_1, \ldots, p_n\}$ of X, the $n \times n$ matrix with entries $K(p_i, p_j)$ is positive definite. This means that for all real numbers $c_1, \ldots, c_n$,

$$\sum_{i,j} c_i c_j K(p_i, p_j) \geq 0 .$$

If K is symmetric, positive definite, and continuous, we call it a *Mercer kernel*. Examples are

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2} \right) \qquad (1)$$

and

$$K(x, x') = \left(a^2 + \|x - x'\|^2 \right)^{-\alpha},$$

where $\| \cdot \|$ denotes the 2-norm and a, α, $\sigma > 0$.

The algorithm is as follows:

1. Start with data $(x_1, y_1), \ldots, (x_m, y_m)$.
2. Choose a Mercer kernel such as the one in Equation 1.
3. Choose a positive real number γ and let $\mathbf{c} = (c_1, \ldots, c_m)$ be the unique solution to the equation

$$(m\gamma \mathbf{I} + \mathbf{K})\mathbf{c} = \mathbf{y}, \qquad (2)$$

   where $\mathbf{I}$ is the $m \times m$ identity matrix, $\mathbf{K}$ is the square positive-definite matrix with entries $K(x_i, x_j)$, $1 \leq i, j \leq m$, and $\mathbf{y}$ is the column vector with components $y_1, \ldots, y_m$.
4. Define $f: X \to Y$ by

$$f(x) = \sum_{i=1}^{m} c_i K(x_i, x) .$$

Because the matrix $m\gamma \mathbf{I} + \mathbf{K}$ is positive definite, and therefore nonsingular, Equation 2 is well posed.

So what does the approximating function f look like? If the kernel is Gaussian (see Equation 1), f is a weighted sum of Gaussian *blobs*, each centered at one of the x_i's.

The weights c_i minimize the error on the training set. The parameter σ from Equation 1 controls the degree of smoothing, noise tolerance, and generalization. For instance, as σ approaches 0, f becomes just a "look-up table" that gives the exact value of y when $x = x_i$ but does not generalize to other values of the input. The parameter γ ensures that Equation 2 is well posed: the larger the $m\gamma$, the better the condition number of Equation 2. On the other hand, if γ were zero, $f(x_i)$ would equal y_i exactly. It turns out that a *best* γ exists that solves this trade-off problem.[6]

This algorithm has been used in many problems. It performs well in a variety of applications involving regression as well as binary classification. Systems like Equation 2 have been studied since Gauss's time; algorithms to efficiently

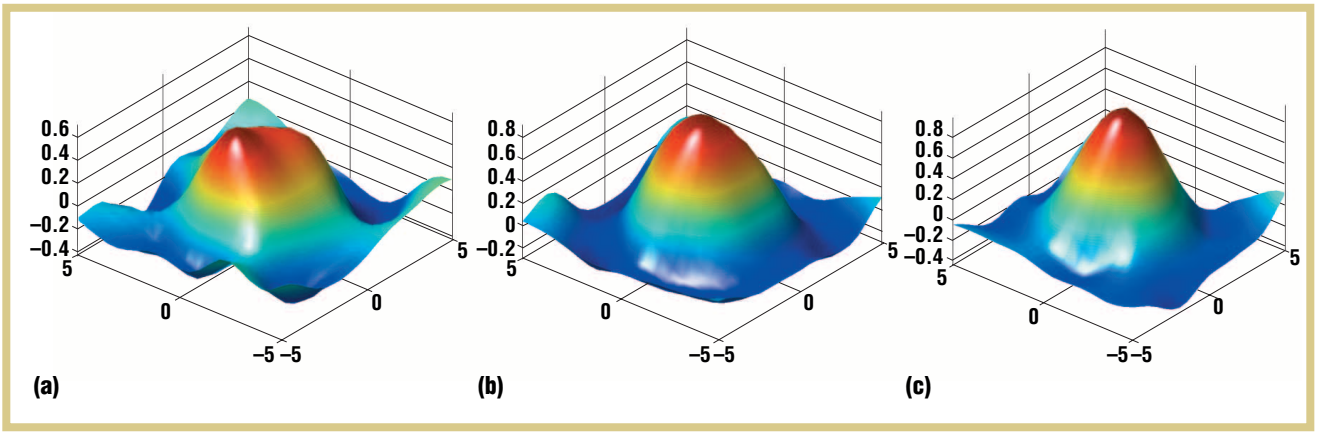


Figure 1. Estimates of f_0 for (a) $m = 100$, (b) $m = 200$, and (c) $m = 500$ nodes, when $\gamma = 0.001$, $\Sigma^2 = 0.01$, and $\sigma^2 = 2$.

compute their solutions form one of the most developed fields of numerical analysis.[2]

To illustrate the algorithm, let's apply it to a training set of m points (x_i, y_i), where the inputs x_i are random, generated by the uniform distribution on the square $Q = [-5, 5]^2$. The output y_i is a Gaussian random variable with mean $f_0(x_i)$ and variance Σ^2, where $f_0(x) = \sin \|x\|/\|x\|$. We want to estimate the (unknown) function f_0 given noisy data. Figure 1 shows how the graph of the estimate changes as $m = 100$, 200, and 500. Figure 2 illustrates how the estimate changes as γ varies.

Sensor network applications

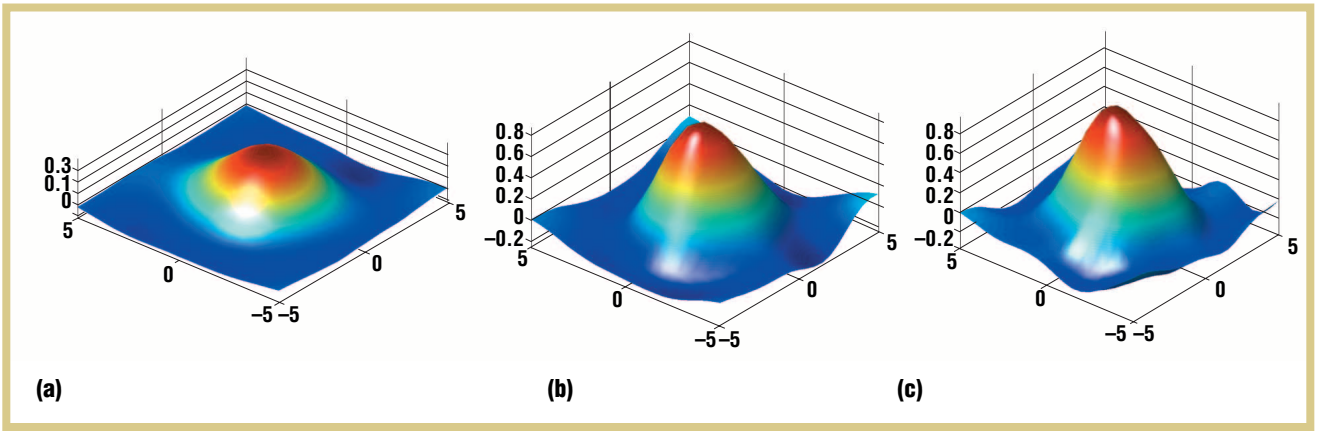
Most applications to sensor networks can be formulated as learning about the environment through examples. Usually, we're interested in a particular feature of the environment that can be represented by a vector-valued function defined on the region in which our sensor network is deployed. Let X be that region. If a sensor network is deployed on the ground, we can take X to be a closed subset of $\mathbb{R}^2$. If the region is significantly curved (for example, mountainous), we can take X to be a 2D surface in $\mathbb{R}^3$. If the sensor network is airborne or deployed in water, we can take $X \subset \mathbb{R}^3$.

Data that the sensor network generates are of the desired form $(x_1, y_1), \ldots, (x_m, y_m)$, where $x_i \in X$ is usually the (estimated) location of the ith sensor node and y_i is an ℓ-tuple of real numbers representing measurements of the ith node. Because the framework from the earlier section requires the outputs to be scalar, we decompose each y_i into scalar components $y_i = (y_i^1, \ldots, y_i^\ell)$ and work with data sets $(x_i, y_i^j)_{i=1}^m$, for $j = 1, \ldots, \ell$, separately. So, without loss, we can assume that our sensor network outputs are scalar.

Recall that points (x_i, y_i) are picked from $X \times Y$ according to some unknown probability measure ρ, which represents the noise in the measurements. What we'd like to estimate (or learn) is the regression function f_ρ of ρ, which in our context is some feature of the environment represented by the outputs $y_i \approx f_\rho(x_i)$.

Applying the algorithm to sensor networks involves these problems:

Figure 2. Estimates of f_0 for (a) $\gamma = 0.1$, (b) $\gamma = 0.01$, and (c) $\gamma = 0.001$, when $m = 100$ nodes, $\Sigma^2 = 0.01$, and $\sigma^2 = 2$.



- How much computation should be distributed to individual nodes, and how much of it should be centralized?
- Which kernel should we choose?
- Which values of γ and σ are best?

This article discusses only the first problem; the latter two are beyond its scope. (See Cucker and Smale's article[6] about finding the best γ.)

Centralized computation might require much long-range communication with a base station, which consumes much energy but might allow for more intensive data processing by a more powerful base station computer. On the other hand, distributed—or *localized*—computation at each node means less long-range communication, but the amount of computation possible will probably be significantly limited. This is the usual trade-off between communication and computation in sensor networks. However, because transmitting a bit of information is still an order of magnitude more expensive than computing it, we opt for a more distributed alternative, assuming that nodes can do a nontrivial amount of computation (such as solving reasonably large systems of linear equations).

We can summarize our general approach, which we can call *distributed sensorweb learning*, as follows: Each node S generates its data set by communicating with its neighbors and applies the algorithm to the set to obtain its own estimate f_S.

We call a node a *neighbor* of a node S if it can establish a bidirectional communication link with S. Denote by D_i the communication region (or neighborhood) of node S_i ($i = 1, \ldots, m$)—that is, the subset of X such that all nodes in D_i are S_i's neighbors. We'll assume that D_i is the disk centered at x_i (the position of S_i) of radius r (the communication range). The estimate f_i by S_i of an unknown function f is then valid only in D_i. What happens if the communication neighborhoods of two nodes S_i and S_j overlap? Which estimate do we accept in the intersection $D_i \cap D_j$? That is, if the need arises, how should the base station merge the estimates f_i and f_j? We have a preliminary solution that glues the local estimates by taking a certain type of average.

Denote by A_i and B_i the disks of radius $r/2$ and $3r/4$, respectively, centered at the ith node S_i. Let $\beta_i : i = 1, \ldots, m$ be a collection of real-valued functions defined on the whole region of deployment X and satisfying these properties:

- The β_i's are smooth
- $0 \le \beta_i \le 1$, $\sum_i \beta_i(x) = 1$ at every point $x \in X$
- $\beta_i = 1$ on A_i
- $\beta_i = 0$ outside B_i

A collection such as this exists, as is well known in differential geometry, and is called a *partition of unity*. Each β_i is what's often called a *bump function* because its graph looks like a bump. Let f_i be the local estimate of an unknown function f, obtained at the ith node using the distributed algorithm; it's defined only on the set D_i the communication region of the ith node. Extend f_i to the function $\bar{f}_i$ defined on all of X by defining $\bar{f}_i$ to be equal to f_i on D_i and zero outside D_i. We glue the $\bar{f}_i$'s into global estimate f_* by setting

$$f_* = \sum_{i=1}^{m} \beta_i \bar{f}_i .$$

Observe that on A_i (that is, close to the node S_i), $f_* = f_i$, the corresponding local estimate obtained by S_i.

How well does f_* approximate f? On each D_i, probabilistic estimates of the error $|f - f_i|$ are readily available from learning theory.[2] Let M be the largest of these errors. It's then easy to see that $|f_* - f| \le M$.

It often makes most sense to assume that the node positions x_i are independent, uniformly distributed random variables on X, but other distributions are possible in particular situations.

Distributed computation at each node means less long-range communication, but the amount of computation possible will probably be significantly limited.

The following examples suggest how to formulate particular sensor network applications from a supervised-learning viewpoint.

Environmental monitoring

In this case, we want to estimate some unknown scalar environmental function f_{env} defined on X, such as temperature, air pressure, humidity, or light.[7] We can assume that the output of the measurement y_i at the node S_i is a normally distributed random variable with mean $f_{\text{env}}(x_i)$. When we apply the distributed algorithm, each node gets a local estimate of f_{env}. We can find upper bounds for the local error with a desired confidence by using the results of Cucker and Smale's work;[3] they'll depend on the number of neighbors each node has and on other network parameters.

We might also want to estimate a vector field F, such as the gradient of temperature, representing the heat flow in the environment.[8] We can do this by reducing the task into estimating the

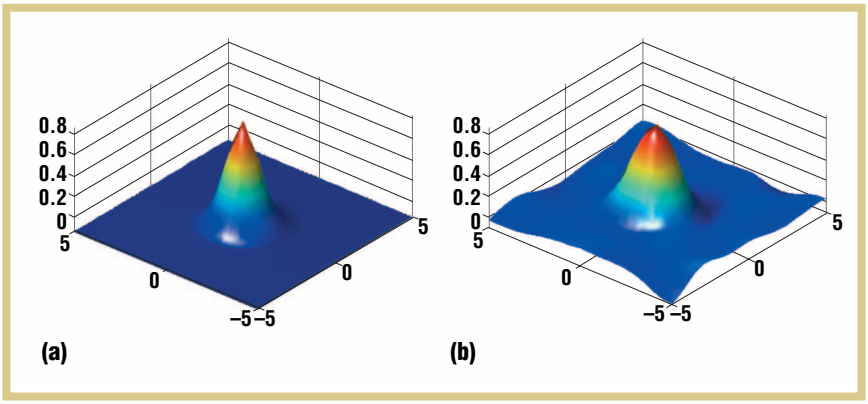


Figure 3. Temperature graphs: (a) The true temperature and (b) an estimate with m = 100 nodes.

field's scalar components by each node.

Plume tracking

Suppose we want to track a plume of hazardous gas moving slowly through region X.[9,10] Let A be the set representing the plume's extent at a time instant t_0. Let f_A be A's *indicator function*—that is, $f_A(x) = 1$ if $x \in A$ and $f_A(x) = 0$ otherwise. Estimating A amounts to estimating f_A. Each node S_i measures the concentration of the gas at its location and outputs $y_i = 1$ if the concentration is larger than some threshold τ; otherwise $y_i = 0$.

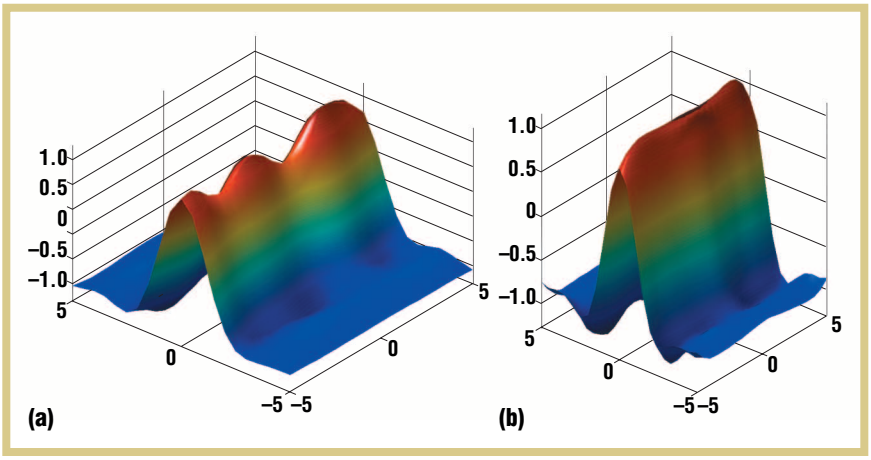
Alternatively, S_i can output its numerical measurement and the network can estimate the concentration function $C : X \to \mathbb{R}$. An estimate of A is then the set of all points x where $C(x) > \tau$.

Tracking moving objects

Suppose an unknown set of objects $\mathcal{O}$ (such as people, animals, or vehicles) is moving through region X. We want to estimate their positions and hence trajectories at various time instances. Let $\delta > 0$ be a desired estimation accuracy. We can think of each object in $\mathcal{O}$ not as a point-mass but as a *smeared point*, or a disk of radius δ, and apply the algorithm.

We denote by $\mathcal{O}_\delta$ the subset of X consisting of points x such that the distance from x to at least one object in $\mathcal{O}$ is less than δ at a fixed time instance t_0. Then, learning the indicator function of $\mathcal{O}_\delta$ amounts to estimating the positions of objects in $\mathcal{O}$ with accuracy δ. Once again, the inputs x_i are positions of sensor nodes and the outputs y_i are +1 (if at least one element of $\mathcal{O}$ is within the distance δ from S_i) or 0 (otherwise).

Figure 4. Estimates of the indicator function of the set $A = \{(u, v) : |v| < 1\}$ using (a) m = 200 and (b) m = 400 nodes, when $\sigma^2 = 2$ and $\gamma = 0.001$.



Localization

We can approach node localization similarly. Suppose a certain number of nodes (for example, beacons) know their positions with some accuracy. Often the sensor network's first task is to localize (that is, estimate the positions of) the remaining nodes.[11,12] Let S be a node with unknown position $p \in X$ and let $\delta > 0$ be the desired accuracy. Then S can communicate with the neighbors who know their positions and gather data (x_i, y_i). x_i is the position of a known neighbor and $y_i = 1$ if the distance between that neighbor and S is less than δ and 0 otherwise. Learning from the given data, S gets an estimate of the indicator function of the disk centered at p with radius δ. We can take the point where the estimate reaches its global maximum to be S's position.

Simulation results

Following are some preliminary simulation results in Matlab. We used the centralized version of the algorithm with a Gaussian kernel (See Equation 1).

Figure 3 compares the graphs of the true temperature function $T(x)$ for $x \in [-5, 5]^2$ (Figure 3a) and its estimate (Figure 3b). We took $T(x) = \exp(-\|x\|^2)$ and assumed the output y_i is a Gaussian random variable with mean $T(x_i)$ and variance 0.001. The number of nodes is m = 100, $\sigma^2 = 0.5$, and $\gamma = 0.0001$.

Figure 4 shows two estimates of the set A = $\{(u, v) : |v| < 1\}$ (which you can consider as being occupied by a plume of gas), using m = 200 and 400 nodes, with $\sigma^2 = 2$ and $\gamma = 0.001$. Pictured are the graphs of the estimates of A's indicator function.

Figure 5 shows a localization estimate with $\sigma^2 = 2$, $\gamma = 0.0001$, and $\delta = 0.9$. We assumed the unknown position of node S is (0, 0); S has m neighbors (m = 30 and 60), each aware of its own position. The figure shows the graph of the function estimating the indicator function of the disk of radius d centered at the origin. The point where it achieves its maximum (ideally, 1) can be taken as the estimate of S's position.

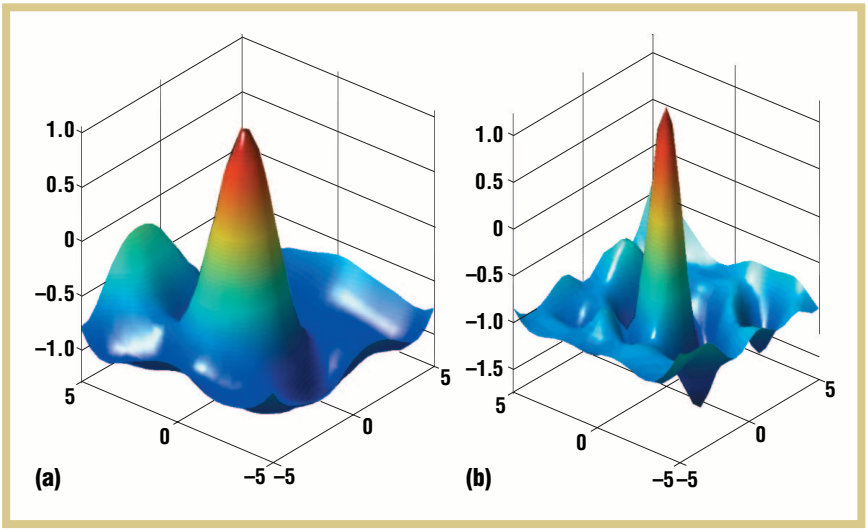


Figure 5. A position estimate using (a) $m = 30$ and (b) $m = 60$ nodes in $[-5, 5]^2$, when $\sigma^2 = 2$, $\gamma = 0.0001$, and $\delta = 0.9$. The correct position is at (0, 0).

Supervised learning theory offers an effective approach to sensor networks. We've demonstrated this through showing how a well-known learning algorithm can be used in the context of environmental monitoring, tracking, and localization. In the future—especially in dense, large-scale sensor networks—we foresee even greater possibilities of such applications, which we plan to investigate in future work. P

ACKNOWLEDGMENTS

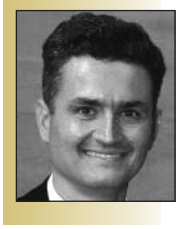
NSF grant EIA-0122599 partly supported our research.

REFERENCES

1. D. Estrin et al., "Connecting the Physical World with Pervasive Networks," *IEEE Pervasive Computing*, vol. 1, no. 1, Jan.–Mar. 2002, pp. 59–69.
2. T. Poggio and S. Smale, "The Mathematics of Learning: Dealing with Data," *Notices Am. Math. Soc.*, vol. 50, no. 5, May 2003, pp. 537–544.
3. F. Cucker and S. Smale, "On the Mathematical Foundations of Learning," *Bull. Am. Math. Soc.*, vol. 39, no. 1, Oct. 2001, pp. 1–49.
4. V. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.
5. G.R. Grimmett and D.R. Strizaker, *Probability and Random Processes*, Oxford Univ. Press, 1997.
6. F. Cucker and S. Smale, "Best Choices for Regularization Parameters in Learning Theory: On the Bias–Variance Problem," *Foundations of Computational Math.*, vol. 2, no. 4, Springer-Verlag, Oct. 2002, pp. 413–428.
7. A. Mainwaring et al., "Wireless Sensor Networks for Habitat Monitoring," *Proc. 1st ACM Int'l Workshop Wireless Sensor Networks and Applications*, ACM Press, 2002, pp. 88–97.
8. S.N. Simić and S. Sastry, "Distributed Environmental Monitoring Using Random Sensor Networks," *Proc. 2nd Int'l Workshop Information Processing in Sensor Networks* (IPSN 2003), LNCS 2634, Springer-Verlag, 2003, pp. 582–592.
9. K.K. Chintalapudi and R. Govindan, *Localized Edge Detection in Sensor Fields*, tech. report 02-773, Univ. Southern California, 2002; http://cs.usc.edu/~ramesh/papers/edge_detection.pdf.
10. R. Nowak and U. Mitra, "Boundary Estimation in Sensor Networks: Theory and Methods," *Proc. 2nd Int'l Workshop Information Processing in Sensor Networks* (IPSN 2003), LNCS 2634, Springer-Verlag, 2003, pp. 80–95.
11. S.N. Simić and S. Sastry, "A Distributed Algorithm for Localization in Random Wireless Networks," to be published in *Discrete Applied Mathematics*, Elsevier.
12. S.N. Simić and S. Sastry, *Distributed Localization in Wireless Ad Hoc Networks*, tech. memo UCB/ERL M02/26, Dept. of Electrical Eng. and Computer Sciences, Univ. California, Berkeley, 2002.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.

the AUTHOR



Slobodan N. Simić is a research scientist in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. His research interests are wireless sensor networks, nonlinear control and hybrid systems, and quantum computing. He received his PhD in mathematics from the University of California, Berkeley. Contact him at the Dept. of Electrical Eng. and Computer Sciences, Univ. of California, Berkeley, CA 94720; simic@eecs.berkeley.edu.