NANO-SCALE FAULT TOLERANT MACHINE LEARNING FOR COGNITIVE RADIO

Jaakko Peltonen[1]*, Mikko A. Uusitalo*[2]*, and Joni Pajarinen*[1]

[1]Helsinki University of Technology, Department of Information and Computer Science,
P.O.Box 5400, FI-02015 TKK, Finland
[2] Nokia Research Center, P.O.Box 407, FI-00045 NOKIA GROUP, Finland

ABSTRACT

We introduce a machine learning based channel state classifier for cognitive radio, designed for nano-scale implementation. The system uses analog computation, and consists of cyclostationary feature extraction and a radial basis function network for classification. The description of the system is partially abstract, but our design choices are motivated by domain knowledge and we believe the system will be feasible for future nanotechnology implementation. We describe an error model for the system, and simulate experimental performance and fault tolerance of the system in recognizing WLAN signals, under different levels of input noise and computational errors. The system performs well under the expected non-ideal manufacturing and operating conditions.

1. INTRODUCTION

In this paper we discuss a new highly promising and challenging domain for hardware implementation of machine learning: nanotechnology-based computing devices.

Nanocomputing is expected to eventually yield high computational speed with low power requirements and small device size. However, nanotechnology is still in development: actual physical implementation is mostly at the level of elementary components, and complicated device architectures are largely theoretical. The challenge is that components are stochastic and manufacturing is prone to errors. Proposed computation devices must work successfully with noise and error-prone computation; machine learning based computation has the potential to satisfy these requirements.

We give a machine learning-based hardware solution for a specific, special-purpose nanocomputing device. Physical research into computing elements is still ongoing, so our solution is at a partially abstract level, but it is detailed enough to evaluate its potential and guide physical research.

We propose a machine learning-based nanocomputing solution to a widely researched problem: *cognitive radio*, i.e., identifying and choosing at each moment the best available frequencies for wireless communication. This application is ideal for nanotechnology implementation because intensive computation is needed; *without nanocomputing it might be infeasible to implement cognitive radio in a single device*, and complicated distributed processing schemes would be needed. Moreover, machine learning is suitable for cognitive radio: identifying free frequencies is essentially a pattern recognition problem. *To our knowledge our paper is the first on a nanocomputing based cognitive radio.*

We give an abstract level implementation of our solution and analyze its performance with respect to input noise and computation errors. *Compared to previous analyses of fault tolerance in machine learning, our novelties are our focus on nanocomputing errors, our feature extraction and classification architecture, and our cognitive radio application.*

We first briefly describe three central topics of our paper: nanotechnology, cognitive radio, and fault tolerance in selected machine learning algorithms. From Section 2 onwards we describe our system and experiments.

1.1. Nanotechnology

Nanotechnology tools allow manipulation of matter at the scale of 1 to 100 nm, yielding new materials and technologies [1]. Nanocomputing is a major attraction of nanotechnology since small scale electronic components have many advantages: large numbers of components can be packed on a small device, components can operate faster due to smaller inter-component distances, and less power is needed at smaller scales. Current state of the art is largely at the level of manufacturing and analyzing individual components like nanowires; simple structures like a small group of connected transistors or a crossing mesh of nanowires [2] have been realized, but larger architectures are at the level of theoretical proposals or at best abstract simulations, like [3].

To combat errors in nanocomputing due to causes like stochasticity in physical component placement or thermal noise affecting electric potentials in nanowires, we suggest

All authors had equal contributions.

that the first complicated nanodevices should not be general purpose computers, rather, they should implement specific algorithms for real-life applications: specific devices are easier to make because critical parts of the computation can be better identified and taken into account in the device design. We study a specific-purpose nanodevice for cognitive radio. Our solution is based on machine learning algorithms which themselves can be error and fault tolerant.

1.2. Cognitive radio

Increasing wireless communications require new solutions to allow a maximal number of users on the same radio channel and avoid "congestion". Congestion could be reduced if each communicating device could analyze at each moment its radio environment and find a suitable free channel. Such intelligent communication is called *cognitive radio* [4, 5]. Cognitive radio devices are not permitted by current legislation (except certain simplistic kinds), but it is hoped they will be permitted in the future under restrictions and requirements for the device operation.

Operating a cognitive radio is at simplest a classification problem: feature extraction is applied to a time series of incoming radio signal, and each channel is classified "occupied" if a signal is present and "free" otherwise. More complicated setups include classifying what kinds of signals are present, planning on which radio parameters to use, negotiating a communication protocol between several devices, and optimizing channel search and negotiation efficiency. We focus on the basic binary classification problem.

Doing analysis and classification for a wide range of channels needs intensive computation: *cognitive radio implemented with conventional electronics could easily drain too much power for a single handheld device*. One solution could be dividing computation among many devices. We study another approach: performing the classification with nanocomputing, which could operate with less power than conventional electronics and could allow integration of cognitive radio into a single handheld device. *Nanocomputing has been mentioned before as a potential solution [5] but no actual suggestions for devices have been presented.*

1.3. Fault tolerance

Implementation of neural networks with conventional hardware and the fault tolerance of such implementations have been widely studied. Fault types studied previously include so-called stuck-at faults where neuron output or weights are fixed at a certain level (see e.g. [6]), multi-node stuck-at faults [7], and noise on a neural network's parameters [8, 9].

Basic approaches for mitigating the effects of faults include adding redundant hidden nodes to a network's crucial parts [9] or restricting parameter diversity [10]. Moreover, artificial neural networks including radial basis function networks can learn to be fault tolerant if trained properly [7]. In [6] injecting stuck-at faults into a neural network during training improved tolerance against both trained faults and faults the network was not trained for. Similarly, in [8] injecting synaptic noise into a neural network during training improved generalization and fault tolerance.

Our proposed nanoscale system needs tolerance against noise and broken/displaced elements. Our system includes a neural network; analyzing its fault tolerance requires extension for a new kind of nanoscale fault. We describe our system in Section 2 and our fault model in Section 3.

2. PROPOSED SYSTEM

We describe, at a partially abstract level, a machine learning system suitable for nanotechnology implementation, which performs channel detection for cognitive radio: it classifies each channel as "free" or "occupied". We first describe the system as it would optimally operate; in Section 3 we then describe a model for faults occurring during manufacturing and operation, and how to train the system to tolerate the faults. Parameter values, fault probabilities etc. are based on our best knowledge, to be confirmed in further work.

For each radio channel, the system consists of two main parts. The incoming radio signal is given to a *cyclostationary feature extraction* system; it produces a so-called α-profile, whose values are used as features for classifying the status of the radio channel by a classical *radial basis function network* (RBFn). Figure 1 shows an overview.

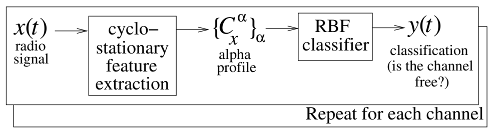


Fig. 1. Schematic of our cognitive radio system. For each radio channel, cyclostationary feature extraction is applied to the radio signal, yielding an α-profile (feature vector); this is given to a radial basis function network that classifies the current state of the channel (free or occupied).

We use analog computation throughout the system: digital computation would require nanotechnology solutions for digital number representation, addition, multiplication etc. which would lead to a high architectural complexity.

Our general architecture applies to any cyclostationary signals, whereas an architecture specific to a signal type may need redesign for each type. We choose the parameters to detect signals in Wireless Local Area Network (WLAN; IEEE 802.11a) channels, but the device is also able to detect other signals than 802.11a signals present in the channels.

2.1. Cyclostationary feature extraction

We use cyclostationary signal analysis to perform feature extraction; cyclostationarity of radio signals is widely used in cognitive radio research [11]. Briefly, a signal is wide-sense cyclostationary if its time-varying autocorrelation $R_x(t, \tau)$ is periodic with respect to time t, for all lags τ. WLAN 802.11a signals have cyclostationary components.

Signals with cyclostationary properties can be characterized by spectral coherences, summarized in an α-*profile* [12]. We extract the α-profile using the averaged cyclic periodogram method [13]. We briefly describe the method and discuss its potential nano-scale implementation.

Values C_x^α of the α-profile are defined as maximal absolute spectral coherences for each cyclic frequency α:

$$C_x^\alpha = \max_f |C_x^\alpha(f)| \,, \; |C_x^\alpha(f)| = \frac{|S_x^\alpha(f)|}{\sqrt{S_x^0(f+\frac{\alpha}{2})S_x^0(f-\frac{\alpha}{2})}} \tag{1}$$

where $C_x^\alpha(f)$ are spectral coherences, i.e. normalized versions of the spectral correlations $S_x^\alpha(f)$. The spectral correlations are correlations over time between two frequency components centered at f and separated by α. We estimate them by summation over T time lags as follows:

$$S_x^\alpha(f) = \sum_{k=0}^{T-1} X^{(k)}\left(f+\frac{\alpha}{2}\right) X^{(k)}\left(f-\frac{\alpha}{2}\right)^* \tag{2}$$

where $X^{(k)}(f)$ are delayed versions of the instantaneous Fourier components of the signal, with delay given by k. In our simulations we estimate them by Fourier transform of Hanning windowed signals, where the windowing corresponds to the delays (see [13]), but in real nanoscale implementation we would instead use special sensors as discussed below. Figure 2 shows examples of α-profiles.

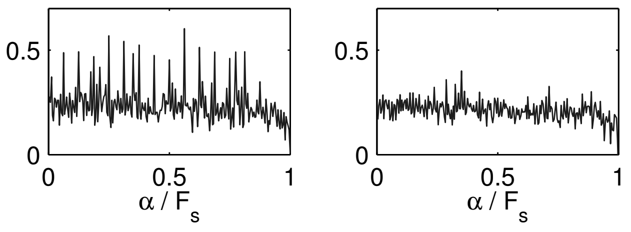


Fig. 2. Example α-profiles of a WLAN 802.11a signal. F_s is the width of the channel (20MHz). Left: Signal-to-noise ratio (SNR) 0dB, right: SNR -10dB. Strong signals yield spikes at the cyclic frequencies.

We propose the following partially abstract implementation. The radio signal $x(t)$ (physically a voltage) is received at a nano-scale sensor bank having two kinds of frequency sensors. The bank contains sensors in the frequency range 5-6 GHz, divided into 20MHz intervals corresponding to the interval of IEEE 802.11a channels; over each 20MHz channel there are sensors for $F = 480$ different frequencies. (We chose F manually to minimize system complexity while retaining performance.) One type of frequency sensor detects magnitude (square root of power) of the signal at each frequency, the other type of sensor detects the phase of the signal. For each frequency, $T = 123$ sensors are used: their responses are delayed by different lags, which will allow approximate computation of spectral coherences around each frequency. Figure 3 illustrates the setup.

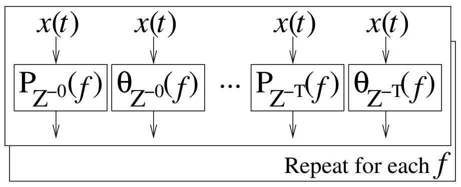


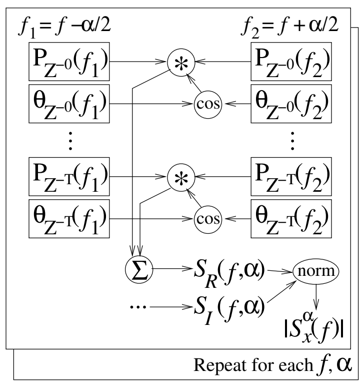
Fig. 3. Frequency-sensitive sensor banks. The $P_{Z^{-k}}(f)$ sensors measure (square root of) signal power at frequency f, with a time delay of length k in their response. The $\theta_{Z^{-k}}(f)$ sensors measure the corresponding signal phase.

Absolute spectral correlations are next computed by the subsystem in Figure 4 (left), for each frequency and for $A = F/2 = 240$ alpha values. Note: actual complex Fourier coefficients are not needed for computation; the outputs of the magnitude and phase sensors suffice. The computation uses only real (not complex) numbers which is crucial for real analog implementation. The result corresponds to absolute value of (2): the proof is simple and is omitted to save space. Lastly spectral coherences are computed by using the normalization in the right subequation of (1); final α-profile values are computed by taking the maximum as in the left subequation of (1). Figure 4 (right) shows the setup.

2.2. Classification

For each radio channel, we use a radial basis function (RBF) network to classify the channel state at each moment (free or occupied). RBF networks have been used in communication systems, for tasks such as equalization; see, e.g., [14].

Our RBF network (RBFn) consists of an input layer, a hidden layer and an output layer. We use the α-profile values $\mathbf{C}_x = [C_x^{\alpha_1}, \ldots, C_x^{\alpha_A}]$ as the inputs, which yields $A = 240$ input units. We use a single output y, which is computed as $y = \sum_{i=1}^n w_i \phi_i(||\mathbf{C}_x - \mathbf{c}_i||) + w_{n+1}$, where n is the number of hidden units, w_i are weights, w_{n+1} is the bias term, $\mathbf{c}_i$ are centroids, and ϕ_i is here a Gaussian nonlinearity $\phi_i(||\mathbf{C}_x - \mathbf{c}_i||) = \exp(-||\mathbf{C}_x - \mathbf{c}_i||^2/2\sigma_i^2)$ with spread σ_i. Roughly speaking, the number of hidden units n should match how many kinds of α-profiles we expect to encounter; we use $n = 30$ which worked well in the experiments. Finally, the channel is classified free if $y < 0$ and



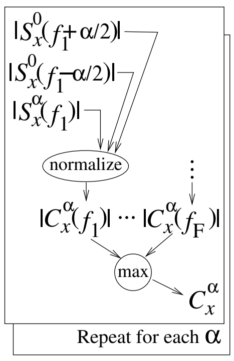


Fig. 4. **Left:** Computation of absolute spectral correlations $|S_x^\alpha(f)|$. The asterisk denotes multiplication, 'cos' denotes the cosine $\cos(\theta_1 - \theta_2)$ of the phase difference, Σ denotes summation, and 'norm' denotes Euclidean vector norm. Computation of $S_I(f,\alpha)$ is omitted for brevity: it is the same as $S_R(f,\alpha)$ except cosines are replaced by sines. **Right:** Computation of α-profile values C_x^α. For each α value, the 'normalize' node computes absolute spectral coherence values $|C_x^\alpha(f)|$ by normalizing the absolute spectral correlations, and the 'max' node computes the maximum over different frequencies which is the final α-profile value.

occupied otherwise. Figure 5 shows the setup.

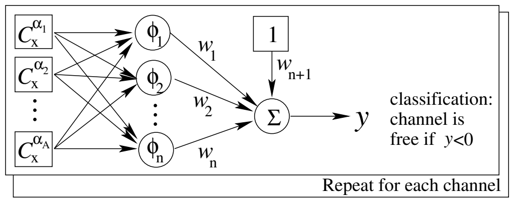


Fig. 5. A RBF network with A inputs $C_x^{\alpha_i}$, n hidden units ϕ_i, n weights w_i, a bias w_{n+1}, and output y.

2.3. Properties of the proposed system

Our system implements both feature extraction and classification in nanoscale; the advantage is that only the final classification output needs to be communicated to non-nanoscale parts of the handheld device. If, for example, only feature extraction was done in nanoscale, much more communication to non-nanoscale parts would be required.

Besides simple summation, our proposed system needs six computation nodes: multiplication, computing $cos(\theta_1 - \theta_2)$ from inputs θ_1 and θ_2, computing norm $\sqrt{x_1^2 + x_2^2}$ of two inputs, computing normalization $x_1/\sqrt{x_2 \cdot x_3}$ of three positive inputs, computing the maximum of inputs, and computing the Gaussian radial basis function. All the nodes are simple functions; we expect it will be possible to physically realize them reasonably accurately in nanocomputation.

RBF networks can be trained to input data, but implementing training in the nanocomputing device would make implementation much more complex. As a first step we propose that RBF network parameters should be optimized by simulations and the optimized parameters should be used as fixed values in the actual implementation. In Section 3.1 we describe how to take faults into account in the optimization.

3. MODELING COMPUTATION ERRORS

To simulate our system's performance under realistic computation errors, we model three fault types: thermal noise in feature extraction, thermal noise in the RBF networks, and structural faults in the RBF networks (structural faults in feature extraction will be studied in future work). We describe a *physically reasonable* expected level for the fault types; this level is relatively uncertain due to missing physical experiments. In Section 4 we simulate the system at this expected level and with alternate levels for each fault type.

Thermal noise is proportional to temperature. The system should operate at room temperature (for sensors we allow lower temperature). This could yield noise of the order of 0.3mV; its effect on computation depends locally on relative scales of thermal noise voltage and signal voltage.

We expect the input at the sensor banks to be up to a few tens of millivolts (mV). Without bringing power to the system for signal amplification, every operation on the signal decreases the voltage scale of the output. We assume sum operations are done with little loss, and each multiplication and other complicated operation reduces the voltage scale by roughly 25%. This yields voltages up to about 10mV at the inputs of the RBFn; the output of hidden units is up to about 5mV, and the final output is up to about 3mV. In our system, signal voltages correspond to numbers with absolute values roughly between 0 and 1. We can thus simulate thermal noise by adding to the numbers Gaussian noise with standard deviation (STD) equal to V_N/V_S; V_N and V_S are voltage scales of the thermal noise and the signal. Gaussian noise approximates analog thermal noise well; see [9].

In feature extraction we use a simplified thermal noise model. We treat magnitude and phase sensors together by adding complex Gaussian noise to Fourier components: we assume highly cooled sensors yielding noise STD 0.001, but brief experiments with room temperature sensors (STD 0.01) gave similar performance. We also add noise (STD 0.03) to α-profile values C_x^α. In the RBFn we add noise to centroids with STD 0.03, to spreads with STD 0.04 (as log-normal multiplicative noise, to keep spreads positive); and to hidden unit outputs and wire weights with STD 0.06.

Structural faults depend on physical implementations and manufacturing methods. We model broken wires, i.e. stuck-at-zero faults, and also *displaced wires going to wrong* *hidden units, which to our knowledge is novel.* A wire (arrow in Figure 5) is broken with probability 0.01; or it goes to the wrong neighboring unit with probability 0.001.

3.1. Optimizing fault-tolerant parameters for the device

To train the RBFn, we use N time series as training data. For each time series, we extract the input features: α-values $C_x^{\alpha_1}, \ldots, C_x^{\alpha_A}$ for the frequency band of the RBFn. For each series the desired output is $y = 1$ if the series really contained a signal and $y = -1$ otherwise. We can then train the RBFn by standard methods: we initialize centers by a k-means type clustering and weights and spreads to fixed values; we then train the centers, spreads and weights by on-line gradient descent with the usual squared error cost function. We reparameterize spreads by $\sigma_i = \exp(\sigma'_i)$ to keep them positive. To make the RBFn fault tolerant, we inject structural and noise faults during training. For each training sample, we inject faults into the feature extraction and the RBFn computation as described in Section 3, according to the desired fault level. The gradient of RBFn parameters is computed using the faulty values of the features, RBFn parameters, wiring structure, and hidden unit outputs. The RBFn then slowly learns to tolerate computation errors.

4. EXPERIMENTS

We ran three experiments; in each, one fault type (feature extraction noise, RBFn noise, or RBFn structural faults) is studied at several fault levels: 0.2, 0.5, 1, 1.5, and 3, while holding the other fault types at the default level 1. The "fault level" is used to multiply standard deviations of thermal noise and to multiply probabilities of structural faults; level 1 means the expected faults described in Section 3. For RBFn structural faults we also tried fault levels 10 and 20.

Since the computation in all radio channels is similar, for simplicity we ran the experiments for a single channel.

We used the simulator of [15] to generate IEEE 802.11a as our input signal, using 16-Quadrature Amplitude Modulation and a convolutional code rate of 1/2, with 24Mbps throughput. Each signal was sampled for 1ms. For each 1ms signal sample, a packet length between 100 and 1472 bytes and a signal-to-noise ratio (SNR) between -10dB and 10dB were chosen. In total, we generated 49200 samples containing a signal and 49200 containing only noise.

We used four-fold cross-validation: in each fold, $3/4$ of the data set was used for training and $1/4$ for testing the system. We report average results over the 4 folds. For each fault level combination the RBFn was trained for 200 iterations over the training set, with learning rate 0.001, and then tested with the test set. We used the training method in Section 3.1; the same fault levels were used for injecting faults in both training and testing.

We show results separately for samples containing signals (occupied channel) and samples containing only noise (free channel). For signal samples, Figure 6 shows average classification success rates over RBFn faults, as a function of SNR: subfigure (a) shows results for feature extraction fault levels, (b) for RBFn noise levels and (c) for RBFn structural fault levels. For noise samples, classification success rate does not depend much on SNR: for brevity, Table 1 shows average results over all SNRs for each fault type.

	Fault level (fault scale multiplier)						
Type	0.2	0.5	1	1.5	3	10	20
(a)	98.7	98.4	98.2	97.2	94.8	-	-
(b)	98.3	98.4	98.2	96.6	83.6	-	-
(c)	98.2	98.1	98.2	98.0	97.4	94.2	80

Table 1. Average test-set classification success rate (in percentages) of noise samples, for levels of (a) feature extraction faults, (b) RBFn noise, and (c) RBFn structural faults.

The system classifies signal and noise samples well at the expected fault level. Training and test performances were similar suggesting our data set was large enough to draw conclusions about the system. Small increases or large decreases to the expected fault level have no major performance impact. Thermal noise in the RBF network seems to have largest effect on signal classification; this may be because signal voltage is lowest when it reaches the RBF network. Moderate structural fault levels have little effect suggesting the 30 hidden units contain enough redundancy.

5. CONCLUSIONS AND DISCUSSION

We presented an abstract level implementation of cognitive radio, based on a nanocomputing implementation of a machine learning algorithm for feature extraction and classification. The system recognized wireless LAN signals well under several levels of input noise and computational errors. *To our knowledge our paper is the first on a nanocomputing based cognitive radio.* Future work includes physical experiments to verify our choices and extending the architecture to other machine learning applications.

6. ACKNOWLEDGMENTS

This work was supported by the Finnish Funding Agency for Technology and Innovation (TEKES). J. Peltonen and J. Pajarinen belong to the Adaptive Informatics Research Centre of the Academy of Finland. J. Peltonen also belongs to Helsinki Institute for Information Technology. We thank A. Huttunen, V. Koivunen, and R. White for useful discussion.

7. REFERENCES

[1] R. Waser (Ed.), *Nanoelectronics and Information Technology: Advanced Electronic Materials and Novel Devices*, Wiley, 2005.

[2] G.-Y. Jung, E. Johnston-Halperin, W. Wu, Z. Yu, S.-Y. Wang, W. M. Tong, Z. Li, J. E. Green, B. A. Sheriff, A. Boukai, Y. Bunimovich, J. R. Heath, and R. S. Williams, "Circuit fabrication at 17 nm half-pitch by nanoimprint lithography," *Nano Letters*, vol. 6, pp. 351–354, 2006.

[3] J. H. Lee and K. K. Likharev, "Defect-tolerant nanoelectronic pattern classifiers," *Int. J. Circuit Theory Appl.*, vol. 35, pp. 239–264, 2007.

[4] J. Mitola III and G. Q. Maguire Jr., "Cognitive radio: making software radios more personal," *IEEE Pers. Commun.*, vol. 6, pp. 13–18, 1999.

[5] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE J. Sel. Areas Commun.*, vol. 23, pp. 201–220, 2005.

[6] B. E. Segee and M. J. Carter, "Comparative fault tolerance of parallel distributed processing networks," *IEEE Trans. Comput.*, vol. 43, pp. 1323–1329, 1994.

[7] C. S. Leung and J. P. F. Sum, "A fault-tolerant regularizer for RBF networks," *IEEE Trans. Neural Netw.*, vol. 19, pp. 493–507, 2008.

[8] A. F. Murray and P. J. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Trans. Neural Netw.*, vol. 5, pp. 792–802, 1994.

[9] R. Eickhoff and U. Rückert, "Robustness of radial basis functions," *Neurocomputing*, vol. 70, pp. 2758–2767, 2007.

[10] S. Cavalieri and O. Mirabella, "A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks," *Neural Netw.*, vol. 12, pp. 91–106, 1999.

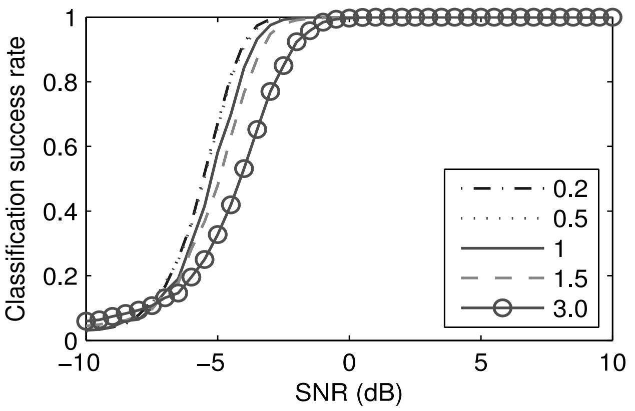
[11] I. F. Akyildiz, W. Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Comp. Netw.*, vol. 50, pp. 2127–2159, 2006.

[12] A. Fehske, J. Gaeddert, and J. H. Reed, "A new approach to signal classification using spectral correlation and neural networks," in *Proc. IEEE Symp. New Frontiers Dynamic Spectr. Access Netw.*, Nov. 2005, pp. 144–150.

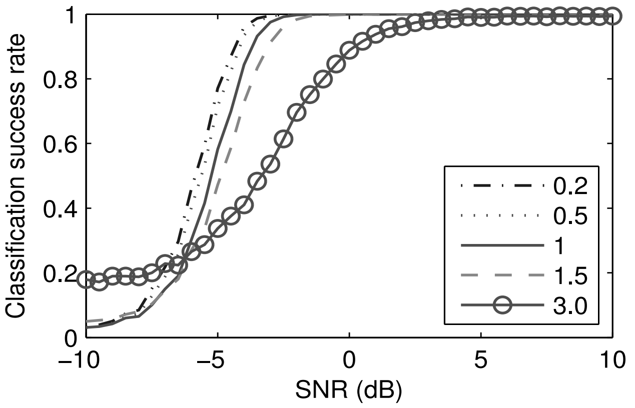
[13] R. Boustany and J. Antoni, "Cyclic spectral analysis from the averaged cyclic periodogram," in *Proc. IFAC World Congress*, July 2005.

[14] A. G. Bors and M. Gabbouj, "Minimal topology for a radial basis functions neural network for pattern classification," *Digital Signal Process.*, vol. 4, no. 3, pp. 173–188, 1994.

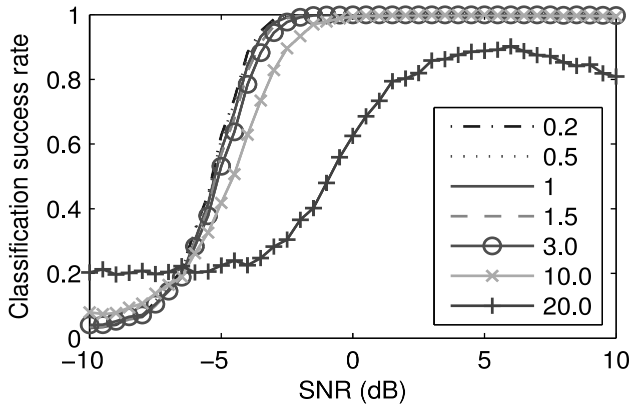
[15] J. Heiskala and J. Terry, *OFDM Wireless LANs: A Theoretical and Practical Guide*, Sams Indianapolis, IN, USA, 2001.



(a) Effect of feature extraction fault levels



(b) Effect of RBF network noise fault levels



(c) Effect of RBF network structural fault levels

Fig. 6. Effect of faults on test-set classification performance. Signal classification success rate is shown as a function of the signal-to-noise ratio (SNR). Each line denotes the system having a particular fault level for a specific fault type. The multipliers in the legends show the fault level for the specific fault type—see the text for details. Other kinds of faults are kept at default levels.