Fault tolerant machine learning for nanoscale cognitive radio [☆,☆☆]

Joni Pajarinen[a,*], Jaakko Peltonen[a], Mikko A. Uusitalo[b]

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

Abstract

We introduce a machine learning based classifier that identifies free radio channels for cognitive radio. The architecture is designed for nanoscale implementation, under nanoscale implementation constraints; we do not describe all physical details but believe future physical implementation to be feasible. The system uses analog computation and consists of cyclostationary feature extraction and a radial basis function network for classification. We describe a model for nanoscale faults in the system, and simulate experimental performance and fault tolerance in recognizing WLAN signals, under different levels of noise and computational errors. The system performs well under expected non-ideal manufacturing and operating conditions.

Keywords: radial basis function network, nanotechnology, cognitive radio, fault tolerance, nanoelectronics

1. Introduction

We discuss hardware implementation of machine learning in a new, challenging hardware platform: *nanotechnology-based computing devices* or *nanocomputing*. We give a partially abstract implementation of a device and analyze its pattern recognition performance and tolerance to physical errors.

The goal of nanocomputing is to be able to build devices with high computational speed, low power requirements and small device size. Nanocomputing research is at a stage of rapid development at the component level, with ongoing

[☆]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, a centre of excellence of the Academy of Finland. J. Peltonen also belongs to Helsinki Institute for Information Technology HIIT.

[☆☆]All authors had equal contributions.

[*]Corresponding author

Email addresses: `Joni.Pajarinen@tkk.fi` (Joni Pajarinen), `Jaakko.Peltonen@tkk.fi` (Jaakko Peltonen), `Mikko.A.Uusitalo@nokia.com` (Mikko A. Uusitalo) attempts to bridge the gap to architecture level. Existing information on ways to combine possible hardware components is limited: actual physical implementation is mostly at the level of elementary components, and complicated device architectures are largely theoretical. The challenge in making complicated architectures is that components are stochastic and manufacturing is prone to errors. Proposed computation devices must work successfully with noise and error-prone computation.

We suggest that *machine learning based computation* can satisfy the requirements of noise and error tolerance. 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 wireless communication application: *cognitive radio* [19], meaning a radio that adjusts its behavior based on sensing its radio environment, so that many radios can efficiently share limited radio spectrum resources. We address a crucial subtask of cognitive radio: identifying at each moment the best available frequencies (radio channels) for wireless communication. This spectrum sensing application is ideal for nanotechnology implementation because intensive computation is needed: *without nanocomputing it might be infeasible to implement sensing and analysis for cognitive radio in a single device*, and complicated distributed processing schemes would be needed. Moreover, machine learning is a suitable approach for this cognitive radio task: identifying free frequencies is essentially a pattern recognition problem. *To our knowledge our paper is the first with results on a nanoscale implementation proposal for cognitive radio.*

This paper extends our conference paper [33]; the main changes in this journal version are an improved architecture, a more detailed error model, a slightly more realistic test setup, more extensive analyses of error tolerance, and more method comparisons.

The rest of the paper is structured as follows: Section 1.1 outlines our contributions, Section 2 provides background information on nanotechnology, cognitive radio, and fault tolerance in selected machine learning algorithms. Section 3 describes the machine learning based cognitive radio system; Section 4 then describes the model used to simulate physical faults occurring in the system and describes how to optimize the system to tolerate the faults. In Section 5 the good performance of the system is experimentally demonstrated. Section 6 concludes the paper.

1.1. Contributions of this paper

There are a large number of challenges for designing a nanoscale implementation for a cognitive radio device. Interfacing nanotechnology and non-nanotechnology parts should be kept at a minimum. The amount of external power brought into the nanoscale device (and the number of places where external power is used) should also be kept at a minimum. Moreover, the number of components and connections should be kept at a minimum in order to design a device that can feasibly be created in the near future. There are also many challenges in the physical details of how to realize nanocomponents. In this paper we discuss the details of physical component implementations only to a small extent because the current level of physical research has not solved the detail questions yet; instead we give answers to design level challenges discussed above.

To design a nanoscale system such that it will be possible to implement the system in the near future, we focus on relatively simple machine learning models, here radial basis function networks. Our focus is on how well such systems will perform in nanoscale with specific faults caused by nanoscale technology.

Our focus is on *passive analog nanocomponent circuits*, which is a different setting than the previously studied hardware implementations of machine learning methods. There are significant differences between a traditional hardware (analog) implementation and an implementation with nanoscale components. In nanoscale thermal noise becomes pronounced, inefficiencies in manufacturing methods cause broken wires more frequently, and new fault types such as displaced wires (see Section 4 for details) can occur. Additionally, in nanoscale, digital implementations may be more difficult to realize than in traditional hardware designs.

Our application is on nanoscale spectrum sensing, a direction which has been made promising by recent research on new kinds of nanoscale sensors. It is especially promising to combine nanoscale spectrum sensing with passive nanoscale analog circuits: *a nanoscale passive design driven by the power in the incoming signals could be very competitive, as one would not need to bring power to every node separately.* The differences between traditional and nanoscale implementations, advantages of analog nanoscale implementations, and recent research on passive analog nanoscale sensors will be discussed in the nanotechnology background Section 2.1.

Our focus is not on the physical implementation of the individual nanoscale components of our system. Although we do not present new nanocomponents for our device design, there is research on nanocomponents, which are similar to the ones our device design requires. In Section 3.5 we discuss this background work and its relationship to the proposed system.

From the cognitive radio perspective, this paper is a small but significant step towards realizing nanotechnology based hardware to do sensing and analysis of radio environment over a GHz range bandwidth in an energy efficient way on a single handheld device, like a mobile phone. *Cognitive radio implemented with conventional electronics could easily drain too much power for a single handheld device.* Nanotechnology has been proposed [19] as a potential field that could help realize machine learning algorithms for cognitive radio. However, to our knowledge this is the first paper to take concrete steps towards that direction.

One of our particular interests is the fault tolerance of our cognitive radio nanodevice. Our solution is based on machine learning algorithms which themselves can be error and fault tolerant. We do not give a detailed physical implementation of our solution, but instead give an abstract level implementation and analyze its performance with respect to input noise and computation errors. However, the error model is based on the assumed nanoscale implementation. *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.* Experiments in Section 5 show that our analog, machine learning based approach yields good tolerance to physical errors.

We stress that our novelty is not the basic use of a machine learning solution for cognitive radio, but the use of a machine learning based solution with a nanoscale architecture and analysis of its performance; this is a previously unexplored direction that could provide crucial benefits in power efficiency. Our current paper sets the design level framework based on which physical research can be carried out in the future.

2. Background

In this section we briefly describe three central topics of our paper: nanotechnology, cognitive radio, and fault tolerance in selected machine learning algorithms.

2.1. Nanotechnology

Nanotechnology tools allow manipulation of matter at the scale of 1 to 100 nm, yielding materials like graphene [18] for fast electronics and technologies like piezoresponse force microscopy for measuring nano-scale structures [45]. 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 [23] have been realized, but larger architectures are at the level of theoretical proposals or at best abstract simulations, like [25]. There is much uncertainty about how useful different nanocomputing approaches will be and how soon their potential is realized; for the near future, manufacturing difficulties will severely constrain feasibility and performance of most approaches. The ease of manufacturing the devices and the ease of programming them must both be considered in device design.

As described in a recent review [32], nanotechnology enables new kinds of components, such as carbon based field effect transistors (FETs), spintronic FETs, molecular FETs, or magnetic logic elements, which are not part of traditional hardware designs. Such new components could make new computational principles and architectures competitive. In nanocomputing the fundamental computing approach can differ from traditional CMOS (complementary metal oxide silicon) based Boolean logic and transistors. Although that approach is most familiar to current chipmakers, it may not be the most efficient way to assemble complicated computing from novel nanoscale components that have special characteristics such as *nanoscale faults*. In this paper the solution is not based on Boolean logic but instead on *analog computing*.

For nanoscale spectrum sensing it could be too challenging, inefficient, and too energy consuming to convert analog signals from every nanosensor into a digital one [32]. A more efficient solution could be to integrate analog processing with the sensors to compress the information for potential transfer to digital world. In an analog implementation a single nanocomponent can be utilized to produce functionality that in a digital design would require several components, like in the case of a carbon nanotube based transistor radio [22]. Advancements in nanotechnology create possibilities for harvesting energy for sensors [47]: we expect that in the future, a passive design, driven by the power in incoming signals, can be implemented using nanocomponents.

Nanoscale faults can occur in nanocomputing devices due to causes like stochasticity in physical component placement or thermal noise affecting electric potentials in nanowires. To combat nanoscale faults, we suggest 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.

2.2. Cognitive radio

Increasing wireless communications require new solutions to allow a maximal number of users on the same set of radio channels and avoid "congestion". In current communication types such as Wireless Local Area Network (WLAN), there are naturally occurring communication pauses on each channel due to communication protocols and user behavior; therefore, congestion could be reduced if each communicating device could exploit such pauses by analyzing at each moment its radio environment and finding a suitable free channel. Such intelligent communication is called *cognitive radio* [28, 19, 1].

In this paper we consider a crucial subtask of operating a cognitive radio: detecting which radio channels have signal activity. This detection task 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. Prior work on applying machine learning to signal detection includes neural networks for classifying signals and signal types [15, 3]. Support vector machines have also been used to classify signals [16, 4]. The cyclostationary input features used in [15, 4] will be discussed in Section 3.1.

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.

2.3. 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. [40, 11, 39]), multi-node stuck-at faults [49, 26], and noise on a neural network's parameters [29, 30, 14].

One of the earlier observations has been that neural networks are not inherently fault tolerant, but can be adjusted to tolerate faults [40, 39, 5]. In [39] 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 [29] injecting synaptic noise into a neural network during training improved generalization and fault tolerance. In [34] it is shown that if there is enough redundancy in the network, then fault tolerance can contribute to generalization ability and vice versa.

Basic approaches for mitigating the effects of faults are the addition of redundant hidden nodes to a network [49, 35, 44, 13] and restricting parameter diversity; in [9] the weights in a neural network layer, influenced by stuck-at-faults, are redistributed as close to the average absolute value as possible.

Fault tolerance of radial basis function (RBF) networks has been analyzed statistically [26] and methods to identify critical parts of the network have been presented [14, 13]. In [26] Kullback-Leibler divergence is used to measure the difference between a desired output function and the output of a faulty RBF network with multi-node stuck-at faults, which turns out to correspond to a standard objective function plus a regularization term. In [42] it is shown that when injecting multi-node stuck-at faults during training, the objective function is identical to the one proposed in [26]. It has been proven in [21] that pure injection of weight noise does not improve the fault tolerance of a RBF network.

In [13] a method to identify critical parts of a radial basis function network is presented, which allows pruning of neurons with low performance impact and targeted adding of neurons to increase fault tolerance. In [14] it is shown that physical limits of implemented radial basis function networks allow upper bound to be derived for output errors even when inputs and parameters are noisy.

Our proposed nanoscale system needs tolerance against noise and against broken or displaced elements. Our system includes a radial basis function network; analyzing its fault tolerance requires extension for a new kind of nanoscale fault. We describe our system in Section 3 and our error model in Section 4.

3. Proposed system

In this section 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 radio channel as "free" or "occupied". We first describe the system as it would optimally operate; in Section 4 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.

We use cyclostationary [17] properties of radio signals to identify whether each channel is occupied. For each radio channel, the system consists of two main parts. The incoming radio signal is given to a *cyclostationary feature extraction* system. The extracted cyclostationary features are then used for classifying the status of the radio channel by a classical *radial basis function network* (RBF network). Compared to other approaches that use cyclostationary features as input to a machine learning classifier [15, 24, 4] our approach differs in the classification method and in the exact input features as explained in Section 3.1. Fig. 1 shows an overview of the system.

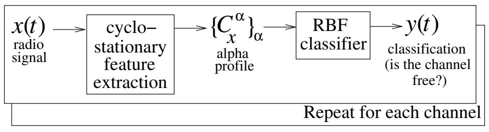


Figure 1: Schematic of our cognitive radio system. For each radio channel, cyclostationary feature extraction is applied to the radio signal, yielding an α-profile (feature vector); this is given to a radial basis function network that classifies the current state of the channel (free or occupied).

We use analog computation throughout the system: digital computation would require nanotechnology solutions for digital number representation, addition, multiplication etc. which would lead to 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 WLAN (IEEE 802.11a) channels, but the device is also able to detect other cyclostationary signals present in the channels.

We next describe the cyclostationary feature extraction system in Section 3.1 and the radial basis function network in Section 3.3.

3.1. Cyclostationary feature extraction

We use cyclostationary [17] signal analysis to perform feature extraction; cyclostationarity of radio signals is widely used in cognitive radio research [1]. 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 and other Orthogonal Frequency Division Multiplexing (OFDM) signals [43] have cyclostationary components.

Signals with cyclostationary properties can be characterized by spectral correlations; our feature extraction system extracts such spectral correlations, and then summarizes them with a smaller set of features. We describe the method below, then discuss its potential nanoscale implementation in Section 3.2.

Our feature extraction system extracts the spectral correlations based on the averaged cyclic periodogram method [7]. Spectral correlations $S_x^\alpha(f)$ are correlations over time between two frequency components centered at f and separated by cyclic frequency α. We estimate them by summation over T time lags as follows:

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

where the $X^{(k)}(f)$ 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 [7]), but in actual nanoscale implementation we would instead use special sensors as discussed later in Section 3.2.

Instead of using the large set of spectral correlations directly as features, it is useful to summarize them by a smaller set of features. In [4] a projection of the spectral correlations

$$C_x^\alpha = \max_f |S_x^\alpha(f)| \tag{2}$$

is introduced, which we use directly as summary input features to the classifier, although it is usual to normalize features of this kind [15, 24, 4]. Leaving out normalization improves tolerance to nanoscale faults and robustness to noise. This design choice is discussed in more detail in experiments Section 5.2. Simulation results in Section 5.2 show that it is beneficial to use the proposed feature extraction method in (2) compared to normalized versions in the literature [15, 4].

3.2. Implementation of the feature extraction

We propose the following partially abstract implementation for the cyclostationary feature extraction system. The radio signal $x(t)$ is received at a nanoscale 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 WLAN 802.11a channels; over each 20MHz channel there are sensors for $F = 240$ different frequencies spaced $\Delta f = 83.3$kHz apart. (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 and outputs a voltage corresponding to the magnitude, the other type of sensor detects the phase of the signal and outputs a voltage corresponding to the phase. For each frequency, $T = 248$ sensors are used: their responses are delayed by different lags, which will allow approximate computation of spectral correlations around each frequency. Fig. 2 illustrates the setup.

Absolute spectral correlations $|S_x^\alpha(f)|$ are next computed by the subsystem in Fig. 3 (left). The set of absolute spectral correlations that we compute is limited by the finite size of the frequency sensor bank, in a slightly complicated fashion. Overall, we compute absolute spectral correlations for $2F - 3$ values of

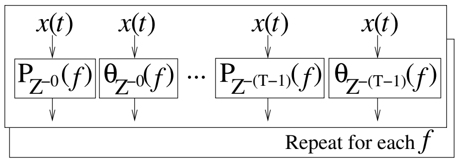


Figure 2: Frequency-sensitive sensor banks. The $P_{Z-k}(f)$ sensors measure (square root of) signal power at frequency f, with a time delay of length k in their response. The $\theta_{Z-k}(f)$ sensors measure the corresponding signal phase.

f and $A = F - 1 = 239$ values of α, where F is the number of frequencies in the sensor bank.[1]

Lastly the α-profile values C_x^α are computed by taking the maximum over frequencies as in (2). Fig. 3 (right) shows the setup.

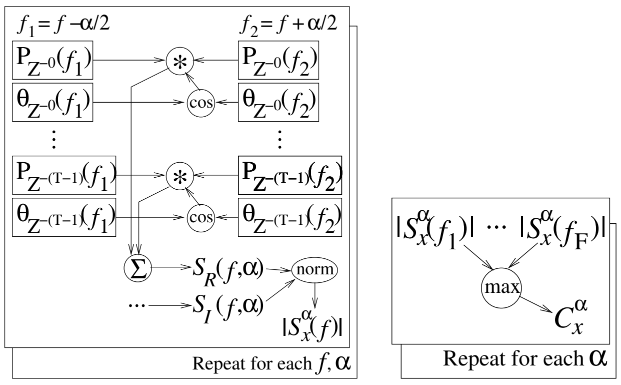


Figure 3: **Left:** Computation of absolute spectral correlations $|S_x^\alpha(f)|$. The asterisk denotes multiplication, 'cos' denotes the cosine $\cos(\theta_1 - \theta_2)$ of the phase difference, Σ denotes summation, and 'norm' denotes Euclidean vector norm. Computation of $S_I(f, \alpha)$ is omitted 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 'max' node computes the final α-profile value as the maximum over different frequencies.

For actual analog implementation it is crucial that the computation only involves real (not complex) numbers. Note that the absolute spectral correlations are absolute values of (1), which involves complex Fourier coefficients. However, we do not need the actual complex coeffients to compute the absolute spectral correlations; the outputs of the magnitude and phase sensors suffice. The computation in Fig. 3 (left) uses only real numbers. It is easy to show that the result corresponds to absolute value of (1): simple algebraic manipulation

[1]Brief explanation: to compute $S_x^\alpha(f)$ by (1), the sensor bank must have sensors for the frequencies $f - \alpha/2$ and $f + \alpha/2$, but f itself can be inbetween two sensors; this yields $2F - 3$ possible values of f. If the sensor frequencies are equally spaced with intervals Δf, then α can take values $\Delta f, 2\Delta f, \ldots, (F - 1)\Delta f$, which yields $A = F - 1$ possible values for α.

of (1) yields $S_x^\alpha(f) = S_R(f,\alpha) + jS_I(f,\alpha)$ where j is the imaginary unit,

$$S_R(f,\alpha) = \frac{1}{T}\sum_{k=0}^{T-1} P_{Z^{-k}}(f_1,f_2)\cos(\theta_{Z^{-k}}(f_1) - \theta_{Z^{-k}}(f_2)) \ ,$$

$$S_I(f,\alpha) = \frac{1}{T}\sum_{k=0}^{T-1} P_{Z^{-k}}(f_1,f_2)\sin(\theta_{Z^{-k}}(f_1) - \theta_{Z^{-k}}(f_2)) \ ,$$

and we denote $P_{Z^{-k}}(f_1,f_2) = P_{Z^{-k}}(f_1)P_{Z^{-k}}(f_2)$, $f_1 = f + \frac{\alpha}{2}$, and $f_2 = f - \frac{\alpha}{2}$. These are real-valued computations. Therefore the absolute spectral correlation is $|S_x^\alpha(f)| = (S_R(\alpha,f)^2 + S_I(\alpha,f)^2)^{1/2}$, which corresponds to the system in Fig. 3 (left).

Fig. 4 shows examples of α-profiles calculated with this method.

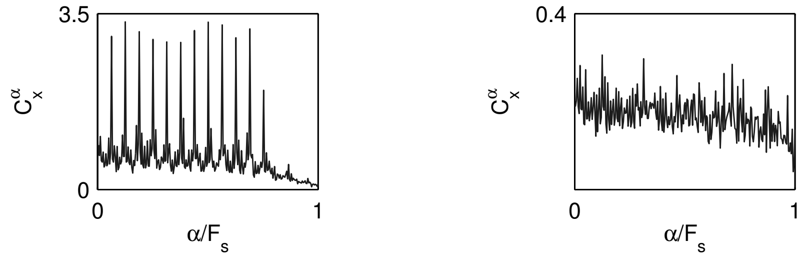


Figure 4: Example feature vectors extracted from a WLAN 802.11a signal. F_s is the width of the channel (20MHz). Left: Signal-to-noise ratio (SNR) 0dB, right: SNR -20dB. Strong signals yield spikes at the cyclic frequencies.

3.3. 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 [6] and signal enhancement [27].

Multilayer perceptrons [15] have been used for this kind of classification task. The reason we use a RBF network instead of a multilayer perceptron is that we feel that the technology of using Gaussian kernels could in future research be adapted to more tasks than RBF networks only; therefore we believe that research in RBF networks can be a useful stepping stone in nanotechnology devices. Moreover, the wiring of a RBF network is simpler compared to multilayer perceptrons [46].

Support vector machines [4] have also been used for this kind of classification task. Our main motivation for using a RBF network instead of a support vector machine is to keep the system simple, to make it feasible to implement the system in nanoscale in the future as discussed in Section 3.4. Furthermore, substantial previous research [13, 14, 26, 42] exists that shows that RBF networks can be made fault tolerant, which is a requirement for nanoscale devices.

In contrast, research on the fault tolerance of support vector machines is at a very early stage. In the fault free case, we have tested (as described in Section 5.2) that our RBF network model achieves similar performance as a support vector machine classifier.

Our RBF network 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 $A = 239$ input units. We use a single output y, which is computed as $y = \sum_{i=1}^{n} w_i \phi_i(\|\mathbf{C}_x - \mathbf{c}_i\|)$, where n is the number of hidden units, w_i are weights, $\mathbf{c}_i$ 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 width 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 output y is received outside the nanoscale system, and the channel is classified free if $y < \textit{threshold}$ and occupied otherwise. Fig. 5 shows the setup.

We perform the thresholding outside the nanoscale system; it is a simple operation which is easy and efficient to do outside nanoscale. An alternative could be to include a bias term in the computation of the RBF output, but without redundancy such a nanoscale bias term would be vulnerable to faults caused by broken wires, and could reduce the fault tolerance of the RBF network.

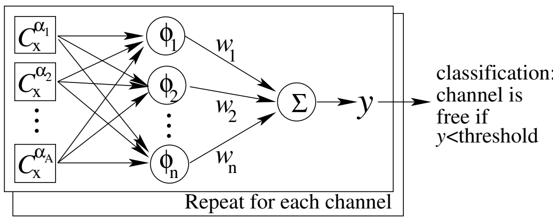


Figure 5: A RBF network with A inputs $C_x^{\alpha_i}$, n hidden units ϕ_i, n weights w_i, and output y.

3.4. Properties of the proposed system

Our system implements both feature extraction and classification in nanoscale; the advantage is that only the final scalar-valued RBF network 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.

After our nanoscale system has received the output voltage from the sensors, the system does not require external power to amplify the signal at any point. The voltage will decrease as computation proceeds through the system; we will take this into account in our model for nanoscale computation errors.

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 4.1 we describe how to take nanoscale computation errors into account in the RBF network optimization.

3.5. Nanocomponents

In this subsection we discuss the elementary nanocomponents that would be needed to realize our architectural design.

Besides simple summation, our proposed system needs five 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 the maximum of inputs, and computing the Gaussian radial basis function.

All five nodes are simple functions; we expect it will be possible to physically realize them reasonably accurately in nanocomputation. At the current stage of physical research many of these components do not have existing nanoscale implementations yet; however, there is ongoing research into similar components, some of which we discuss below.

The input to our system comes from nanoscale spectrum sensors. Such sensors could be fabricated with nanoelectromechanical systems (NEMS), taking advantage of the mechanical properties of nanostructures, such as single carbon nanotubes [38] or ultrathin films of them [8]. There are several advantages in NEMS compared to microelectromechanical systems (MEMS), as reviewed in [31]: Compared to MEMS, the dimensions of the movable parts in NEMS are much smaller resulting in greatly reduced controlling voltages (up to 50 V for MEMS versus a couple of volts for NEMS). This makes them much more compatible with the battery voltages of handheld devices. Another advantage for radio applications is the mechanical resonance frequency range of NEMS structures, which lies in the radio-frequency domain (100 MHz - 10 GHz) due to their low masses and high Young's moduli.

There are several options for the implementation of summation: for example, if values represent the amount of current, simple joining of nanowires will cause summation of the current.

In case the available implementations of frequency sensors do not provide the required delays, the needed delays and summation in spectral correlation calculation could also be implemented together using a nanoscale integrator [48]. Then the number of required frequency sensors would also decrease significantly compared to frequency sensors having delay and no special delay components would be needed. The non-linearity of simple integrators may change the computed features, but because the RBF network is trained from feature data and can adapt to changes, the differences in the features may not affect the classification results severely.

Some nanocomponent implementations could be based on existing designs for traditional analog hardware, such as existing research on implementing radial basis functions [46] and maximum operators [36].

We stress that there are many open questions in the detailed implementation of our required nanocomponents. (Note that this is not unique to our design approach: there are currently no available nanocomputation implementations for tasks of this complexity regardless of the design approach!) However, the existence of related research gives confidence that it will be possible to realize our system with future advances in physical research. Furthermore, we note that there is pressing need for low energy consumption spectrum analysis in portable devices; thus, to answer this need, we believe it will be fruitful to pursue physical research for solving the remaining open questions and implementing our proposed architecture.

We next describe our model for computation errors occurring in our system.

4. Modeling Computation Errors

To simulate our system's performance under realistic computation errors, we model four fault types: thermal noise in feature extraction, structural faults in feature extraction, thermal noise in the RBF networks, and structural faults in the RBF networks. We describe a *physically reasonable* expected level for the fault types; this level is relatively uncertain due to missing physical experiments. In Section 5 we simulate the system at this expected level and with alternate levels for each fault type.

Thermal noise perturbs the voltages inside the cognitive radio system; therefore the numbers that the voltages represent are affected by thermal noise. Thermal noise is proportional to temperature. A cognitive radio system implemented in a mobile device should operate at room temperature. This could yield thermal noise of the order of 0.3mV; its effect on computation depends locally on the relative scales of thermal noise and signal voltage.

We expect the input signal at the sensor banks to have voltage 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%. After the sensor banks the signal flows through the feature extraction circuit; this yields voltages up to about 22.5mV at the cosine nodes, 16.9mV at the multiplication nodes and at the components of spectral correlation, 12.7mV at the absolute spectral correlations and 9.5mV at the feature extraction output (RBF network input); the output of RBF network hidden units is up to about 5.3mV, and the final output is up to about 4mV.

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 (SD) 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 [37].

In the feature extraction circuit we add Gaussian noise with SD 0.03 to the magnitudes and phases of the frequency sensors. This noise consists of two components, SD 0.01 due to the thermal noise and 0.02 due to imperfections in the frequency sensors. Proceeding through the feature extraction circuit, noise is added with SD 0.013 to outputs of the cosine nodes; with SD 0.018 to outputs of the multiplication nodes; with the same SD to the outputs of the sum nodes; with SD 0.024 to the spectral correlation normalization; and with SD 0.032 to the outputs C_x^α of the maximum operation. Next the signal proceeds through the RBF network, where we add noise to centroids with SD 0.032, to squared widths σ_i^2 with SD 0.042 (as log-normal multiplicative noise, to keep σ_i^2 positive); and to hidden unit outputs and wire weights with SD 0.056. The crucial point about this error model is that the standard deviations of errors increase towards the end of the circuit as the signal voltage decreases.

Structural faults depend on physical implementations and manufacturing methods. We model broken wires, i.e. stuck-at-zero faults, in the feature extraction and in the RBF network; a wire (arrow in Figs. 3 and 5) is broken with probability 0.01. In the RBF network we also model *displaced wires going to wrong hidden units, which to our knowledge is novel*; a wire (arrow in Fig. 5) goes to the wrong neighboring unit with probability 0.001.

In the current paper we model displaced wires only in the RBF network; modeling of possible displacements in the feature extraction circuit is left for future work, since they depend on the precise layout of the implemented circuit which requires further physical research.

4.1. Optimizing fault-tolerant parameters for the device

To train the RBF network, 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 RBF network. For each series the desired output is $y = 1$ if the series really contained a signal and $y = -1$ otherwise. We then train the centers, widths and weights of the RBF network by on-line gradient descent with the usual squared error cost function. We reparameterize squared widths by $\sigma_i^2 = \exp(\sigma_i'^2)$ to keep them positive.

As a technical note, the RBF network parameters are initialized by k-means++ clustering [2] (k-means++ has better convergence results than standard k-means) followed by expectation-maximization training of a Gaussian mixture model with separate components for each class. The RBF weights are initialized proportional to the component probabilities, where the sign of each weight is determined by the class of the component. For each RBF the width is initialized to the variance of the corresponding mixture component (the Gaussian component covariance matrix is diagonal with uniform diagonal entries) and the center to the mean of the corresponding mixture component.

To make the RBF network fault tolerant, we inject structural and noise faults during training. For each training sample, we inject faults into the feature extraction and the RBF network computation according to the model described in Section 4.

The gradient of RBF network parameters is computed using the faulty values of the features, RBF network parameters, wiring structure, and hidden unit outputs. The RBF network then slowly learns to tolerate computation errors.

During RBF network training noise is injected to centroids and square widths, and broken wire faults are injected. Existing research indicates that injection of noise into hidden node outputs and wire weights during training may not improve fault tolerance [21], however, during evaluation this allows realistic evaluation of performance against nanoscale faults and for simplicity noise is also injected to hidden node outputs and wire weights during training. Mathematically, noise is injected by adding Gaussian perturbations to wire weights and components as described in Section 4. Displaced wire faults are also injected during training. The results in the experiments of Section 5 show that this approach performs well.

When centers and widths of a RBF network are trained using gradient descent, the RBF network may converge to a local minimum. In the experiments Section 5, where the RBF network was also compared to a support vector machine classifier, there were no convergence problems. We suspect the good convergence may be caused by the initialization procedure of the RBF network parameters.

Since the RBF network performs binary classification, after the training it is possible to trade off the rates of true positives vs. true negatives (channels correctly classified as occupied vs. channels correctly classified as free), simply by changing the classification threshold until the desired tradeoff is achieved on the training set. In the experiments we used this method to fix the true negative rate at 95%.

5. Experiments

We first ran four experiments; in each, one fault type (feature extraction noise, feature extraction structural faults, RBF network noise, or RBF network structural faults) is studied at several fault levels, 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 4. Structural faults were studied at fault levels 0.2, 1, 3, 10, and 20 and noise at fault levels 0.2, 0.5, 1, 1.5, and 3. As a special case, we also ran the level "All 0" meaning no faults of any type.

Since the computation in all radio channels is similar, for simplicity we ran the experiments for a single channel. We used the simulator of [20] 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.[2]

Each signal was sampled for 1ms. For each signal sample a packet length of 40 bytes was chosen with probability 0.4, 1500 bytes with probability 0.2, and a value between 40 and 1500 bytes uniformly with probability 0.4. This is in agreement with the Internet packet length observations in [41]. We used AWGN noise with a fixed volume, and for each signal sample a signal-to-noise ratio (SNR) between -25dB and 10dB was chosen. For simplicity, multipath effects were not simulated. For each fault level combination, we generated in total 6300 samples containing a signal and 6300 containing only noise.

[2]The system performance is not limited to this choice of data: we also tried randomly selected code rates and modulations for each packet, with similar results.

Table 1: Average test-set classification success rate (in percentages) of noise samples, for levels of noise and structural faults (fault scale multipliers). 'Overall' and 'Structural variability' refer to experiments in Section 5.1.

	Noise fault level					
Fault type	All 0	0.2	0.5	1	1.5	3
RBF network	93.6	94.1	94.6	94.7	94.7	95.3
Feature extraction	93.6	95.1	94.2	94.7	94.3	94.1
Overall	93.6	94.6	93.8	94.7	94.8	95.0
Structural variability	-	94.6	94.2	94.5	94.7	94.8
	Structural fault level					
Fault type	All 0	0.2	1	3	10	20
RBF network	93.6	94.9	94.7	94.8	94.9	95.2
Feature extraction	93.6	94.6	94.7	94.7	94.3	94.2
Overall	93.6	94.7	94.7	94.8	95.1	94.6
Structural variability	-	94.9	94.5	94.9	94.4	95.4

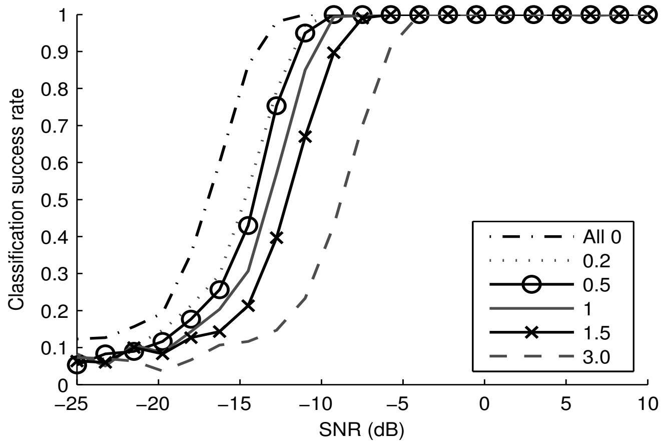
We used four-fold cross-validation: in each fold, 3/4 of the data set was used for training and 1/4 for testing the system. We report average results over the 4 folds. For each fault level combination, the RBF network was trained for 400 iterations over the training set, with learning rate 0.001, and then tested with the test set. We used the training method in Section 4.1; the same fault levels were used for injecting faults in both training and testing.

Fig. 6 shows average classification success rates, for samples containing signals (occupied channel) as a function of SNR. Subfigure (a) shows results for feature extraction noise levels, (b) for feature extraction structural fault levels, (c) for RBF network noise levels and (d) for RBF network structural fault levels. Results without faults (the "All 0" level) are shown as an upper bound.

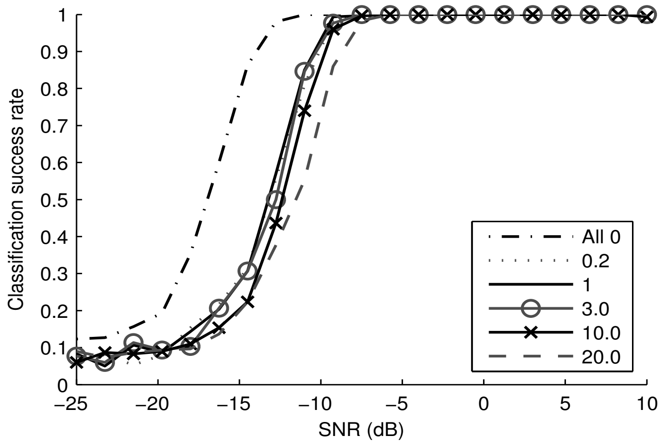
Note that we fixed the true negative rate (classification success rate on free channels, i.e., samples containing only noise) to 95% on the training sets as described in Section 4.1. The corresponding rates on test sets are very similar over all experiments, ranging from 93.6% to 95.4%; the precise numbers are given in Table 1 for completeness.

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 seems to have a larger effect on signal classification than structural faults. Even at the highest level of structural faults, where fault probability is 20 times as high as at the expected fault level, the performance of the system does not degrade noticeably.

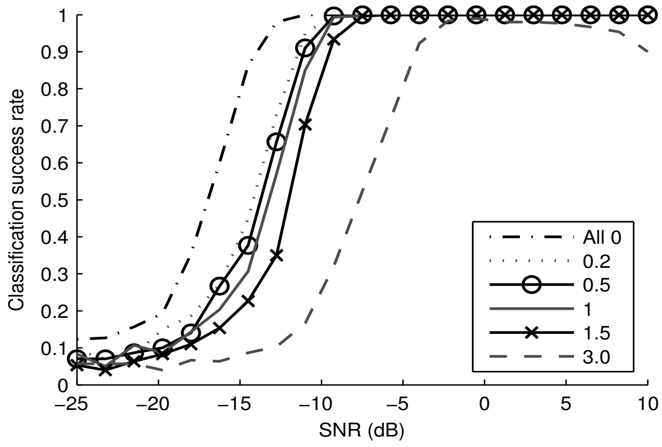
Thermal noise in the RBF network seems to have a larger effect on signal classification than thermal noise in the feature extraction circuit; this may be because signal voltage is lowest when it reaches the RBF network. Structural faults also seem to have a somewhat larger effect in the RBF network than in the feature extraction; a potential reason is that the RBF network has fewer



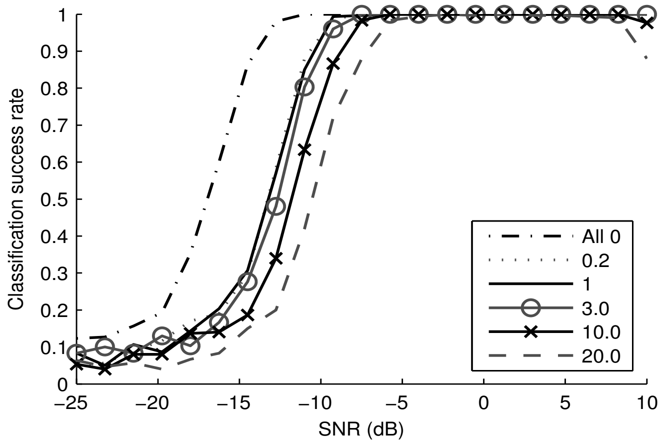
(a) Effect of feature extraction noise fault levels



(b) Effect of feature extraction structural fault levels



(c) Effect of RBF network noise fault levels



(d) Effect of RBF network structural fault levels

Figure 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 the default levels. "All 0" is an upper bound denoting no faults of any kind.

wires than the feature extraction circuit, which can make the RBF network computation more vulnerable to individual structural faults.

5.1. Overall effect of structural and noise faults, and variability across devices

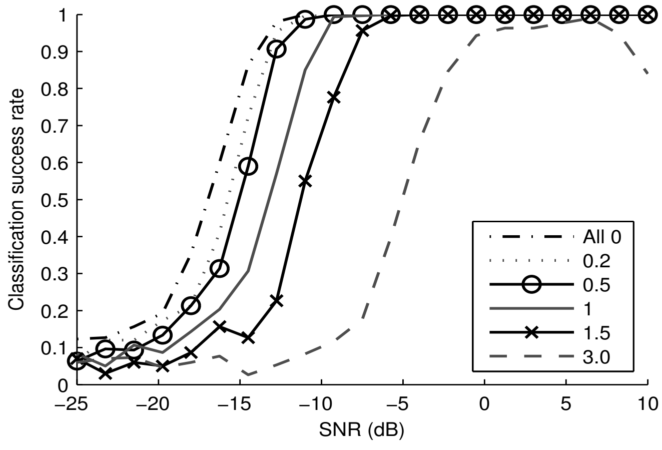
We ran two additional experiments studying the overall effect of structural and noise faults in the system. In the first experiment, structural faults were held at the expected level for both the RBF network and the feature extraction, and the effect of noise was studied at fault levels 0.2, 0.5, 1, 1.5, and 3, affecting both the RBF network and the feature extraction. In the second experiment the noise was held at the expected fault levels and structural faults were varied at levels 0.2, 1, 3, 10, and 20. Fig. 7 shows performances as a function of SNR, for the different overall fault levels; results without faults (the "All 0" level) are again shown as an upper bound. The effect of fault levels is larger than in Fig. 6 since here they affect both the feature extraction and the RBF network. The system again performs well.

The previous figures show average performance across different individual instances of faults. During actual device operation noise fault instances are temporary, occuring continuously, but structural faults can be permanent. It is therefore interesting to see how the performance varies across individual devices with permanent structural faults. We next tested the variability of the system performance over different structural fault instances, for a manually selected subset of SNR values.

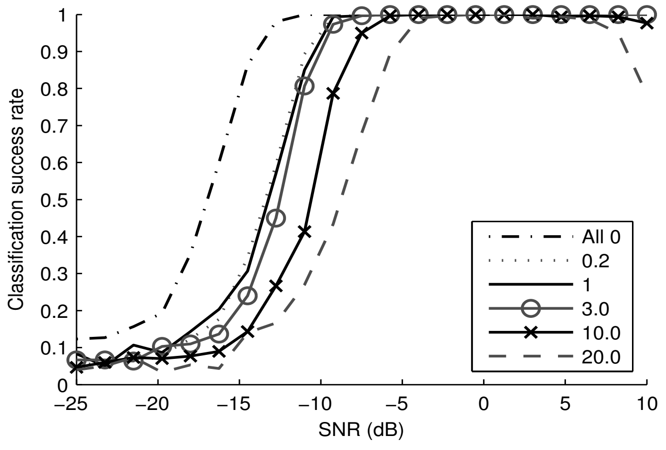
Technically, we tested the variability separately in each cross-validation fold of the above experiments, then averaged the results. In each fold, we took the trained RBF network parameters, and for each selected combination of fault levels and SNR, we generated 40 structural fault instances, and generated 50 signal samples and 50 noise samples for each instance to measure its classification success rate. We computed the 2nd quantile, median and the 3rd quantile of the classification success rates over the instances; we then averaged the median and quantiles over the cross-validation folds.

The results are shown in subfigures (c) and (d) of Fig. 7. Subfigure (d) shows that the structural variability is high for very high structural fault levels (10, 20). This suggests that for these fault levels, structural fault combinations start to appear that affect system operation especially much. On the other hand, in subfigure (c) the structural variability seems to diminish slightly when the thermal noise fault level increases; this may be because the high thermal noise becomes the main influence on system performance. Overall, although there is variability across individual structural fault instances (individual devices), the performance is good enough that building cognitive radios by the proposed approach is feasible.

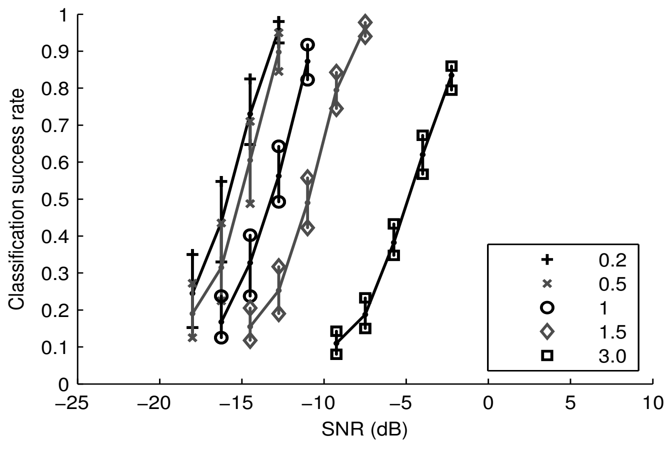
Side note: performance with highest SNRs and high RBF network fault levels. In Fig. 6 subfigures (c) and (d) and in Fig. 7 subfigures (a) and (b), for the highest fault levels the classification performance drops somewhat for the highest SNRs. It turned out that the computation on high-SNR samples is more vulnerable to faults, due to the RBF training method. We used standard gradient-based training, plus injection of faults. The training assigned only few RBF centers to respond to the high-SNR samples; this made the system vulnerable to faults in those centers. Additionally, as the desired output for occupied channels was set to +1, the system did not seek redundancy that would have yielded higher outputs for high-SNR samples. These effects are an example of how properties of nanoscale computation affect design and training of nanocomputing devices. For our device, several remedies for the above effects exist, for example one can simply add more RBF centers, or make individual



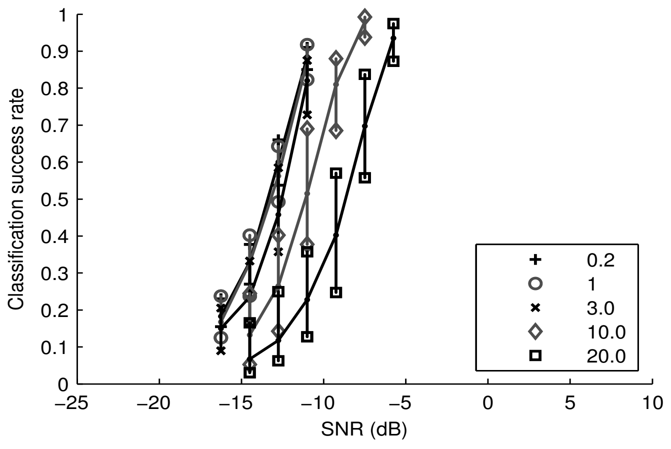
(a) Effect of overall noise fault levels



(b) Effect of overall structural fault levels



(c) Effect of overall noise fault levels: structural variability



(d) Effect of overall structural fault levels: structural variability

Figure 7: Effect of overall faults on test-set classification performance. **Top:** Signal classification success rate is shown as a function of the signal-to-noise ratio (SNR). In subfigure (a) each line denotes the system having a particular fault level for noise (multipliers in the legend), and structural faults are kept at the default level. Conversely, in subfigure (b) the system is studied at different structural fault levels (multipliers in the legend) and noise is kept at the default fault level. "All 0" is an upper bound denoting no faults of any kind. **Bottom:** Structural variability of the performance (variability over individual devices having individual structural faults). Subfigures (c) and (d) show the median of the average classification success rate over individual structural faults; the bottom and top of the error bars show the 2nd and 3rd quantile respectively.

centers more expressive (say by allowing full diagonal covariances). However, note that the effects only occurred for the highest fault levels; for more moderate levels, the methods used in this paper already suffice to give consistently good performance.

5.2. Effect of feature extraction and classification method choices

In this subsection the proposed feature extraction method without normalization is compared to feature extraction methods found in the literature that use normalization [15, 4], with respect to classification performance and tolerance to nanoscale faults. In addition, the fault free signal classification performance of the proposed system is compared to the performance of the system proposed in [4]. Also, the fault free signal classification performance of the proposed RBFn classifier is compared with a SVM classifier.

In Section 3.1 we motivated that leaving out spectral correlation normalization in feature extraction will improve system performance and fault tolerance. We show this by brief experimental comparisons to two feature extraction alternatives [15, 4], which we discuss next.

In the first alternative, a normalized α-profile is used, which consists of maximal absolute spectral coherences

$$\max_f \frac{|S_x^\alpha(f)|}{\sqrt{S_x^0(f-\frac{\alpha}{2})S_x^0(f+\frac{\alpha}{2})}} \tag{3}$$

for each cyclic frequency α. In [15] these normalized features are used as input to a neural network and in [24] as part of input feature calculation for a hidden Markov model. The method in (3) could be implemented in nanoscale by adding an additional normalization node in Fig. 3 (right) before the max operation with inputs $|S_x^\alpha(f)|$, $|S_x^0(f-\frac{\alpha}{2})|$, and $|S_x^0(f+\frac{\alpha}{2})|$.

It would be possible to implement such normalization in the wiring diagrams of Figure 3 (details omitted for brevity); however, such normalization would be problematic in our nanoscale system. When the feature extraction system is implemented in a nanoscale circuit, as discussed in Section 3.2, it may have broken wires and there will be thermal noise during operation. In simulations we found that if normalization is included in the implementation, the broken wires and noise can cause very small or zero valued denominators in the division operation of (3). The maximum operation could then pick the resulting faulty values as the final feature value. Many of the extracted features can thus saturate at faulty high values, especially when hardware faults are common. Also, as discussed in Appendix A spectral correlations $S_x^0(f)$ used in the normalization of (4) and (3) contain too much noise.

To test the effect of spectral coherence normalization we ran experiments with a system implementing the normalization in (3), for the expected fault level "1" and without any faults (denoted "All 0"). We compared that system to the one without normalization which we used in our other experiments. To make a fair comparison, we assumed the normalization included a simple limiter, so that zero denominators or results exceeding valid α-profile values would be set to zero. Results with and without normalization are shown in Fig. 8; leaving out the normalization clearly improves performance, which confirms our design choice was beneficial.

In the second feature extraction alternative, a truncated normalized version

$$\frac{\max_f |S_x^\alpha(f)|}{\max_f |S_x^0(f)|} \tag{4}$$

of (2) is used [4]. In [4] such features are used as input to a SVM classifier; note that this is not a nanoscale method and works as a benchmark of performance.

Since the SVM method is a non-nanoscale method, we will not attempt to compare tolerance of nanoscale faults. Instead, we will make a benchmark comparison of performance in the fault-free case, between our proposed RBF-based system and the SVM-based method of [4] including their normalization, and the SVM-based method without normalization.

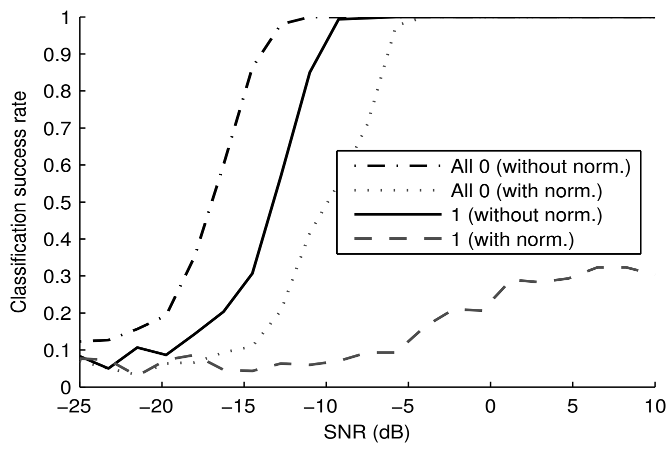
The system in [4] uses Equation (4) for computing input features from signal data and a SVM for classifying the computed features. In [4] two signal types and noise are classified, but the same system can be equally well applied to binary classification. For the experiments the classical SVM [12] was trained and tested on scaled fault free α-profiles using a Gaussian kernel with the software available at [10]. The SVM constants were obtained using a grid search [10]. The true negative rate was fixed at 95% as in all the experiments. Fig. 9 shows the results for our fault free proposed system denoted "RBFn (without norm.)" and for the system in [4] (denoted "SVM (Bixio norm.)"). The proposed system performs better than the system in [4].

Lastly, we note that although we have used a standard training approach for the RBF network in our system (minimizing a squared error cost function), we could have used SVM-based training: we could have trained an SVM classifier with our extracted features. Its support vectors could have been used as radial centers and its weight vector as the output layer weights; in other words, the trained SVM could have been implemented as an RBF network. Fig. 9 shows also the performance of the proposed system, when the RBFn classifier is trained by a SVM (denoted "SVM (without norm.)") instead of using the proposed RBFn training method (denoted "RBFn (without norm.)"). The two methods perform equally well, although each RBFn trained by a SVM uses over 3000 neurons and the proposed RBF network only 30 neurons. One reason for the good RBFn performance may be that the RBFn training method trains the individual neuron widths, but in the SVM training method the widths are uniform.

In summary, for our application, the RBFn approach provides equal performance with less resources than an SVM, and our decision to leave out normalization both simplifies our system and yields better performance than two existing normalization approaches.

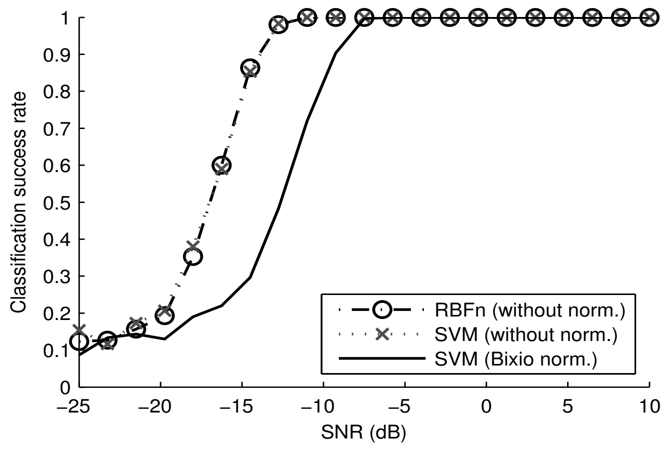
6. Conclusions

We presented an abstract level implementation of cognitive radio, based on a nanocomputing implementation of a machine learning algorithm for feature extraction and classification. In extensive experiments, the system recognized



(a) Effect of design choices

Figure 8: Effect of our design choice on test-set performance. Signal classification success rate is shown as a function of the signal-to-noise ratio (SNR), with and without spectral coherence normalization. The effect of the normalization is tested with the expected fault level (denoted “1”) and with no faults of any kind (denoted “All 0”). Our choice to leave out normalization improves results.



(a) Method comparison

Figure 9: Effect of the choice of feature extraction and classification methods on fault free test-set performance. Signal classification success rate is shown as a function of the signal-to-noise ratio (SNR). Results are shown for the proposed system “RBFn (without norm.)”, for a support vector machine (SVM) classifier “SVM (without norm.)”, and for the classifier system proposed in [4] by Bixio et al. “SVM (Bixio norm.)”. 1) Our proposed system performs better than the method proposed in [4]. 2) In this task the RBFn classifier performs equally well compared to a SVM.

wireless LAN signals well under several levels of input noise and computational errors. *To our knowledge our paper is the first with results on a nanoscale implementation proposal for cognitive radio.* Future work includes physical experiments to verify our choices and extending the architecture to other machine learning applications.

Appendix A. Normalization noise

The spectral correlation normalization terms $S_x^0(f)$ used in [15, 4] to produce summary features from spectral correlation functions are vulnerable to noise in the radio signals and in case of nanoscale frequency sensors also to noise in the sensors.

Spectral correlations are between different frequencies whose noises may partially cancel each other, but the normalization terms $S_x^0(f)$ measure signal power for which such canceling does not happen. Assuming additive Gaussian noise in the signal, then the Fourier transform of the signal at frequency f has additive Gaussian noise $N(f) \sim \mathcal{N}(0, \sigma^2)$ and the Fourier transform of the received signal is $X(f) = \hat{X}(f) + N(f)$, where $\hat{X}(f)$ is the Fourier transform of the original signal. The spectral correlation function is then

$$\begin{aligned} S_x^\alpha(f) &= \frac{1}{T}\sum_{k=0}^{T-1} X^{(k)}\left(f+\frac{\alpha}{2}\right) X^{(k)}\left(f-\frac{\alpha}{2}\right)^* \\ &= \frac{1}{T}\sum_{k=0}^{T-1}\Big[\hat{X}^{(k)}\left(f+\frac{\alpha}{2}\right)\hat{X}^{(k)}\left(f-\frac{\alpha}{2}\right)^* + \\ &\quad \hat{X}^{(k)}\left(f-\frac{\alpha}{2}\right)^* N^{(k)}\left(f+\frac{\alpha}{2}\right) + \\ &\quad \hat{X}^{(k)}\left(f+\frac{\alpha}{2}\right) N^{(k)}\left(f-\frac{\alpha}{2}\right)^* + \\ &\quad N^{(k)}\left(f+\frac{\alpha}{2}\right) N^{(k)}\left(f-\frac{\alpha}{2}\right)^*\Big] . \end{aligned}$$

For $\alpha \neq 0$ the averaging over T terms cancels noise effectively in case the instantenous noise on different frequencies is not strongly correlated, but the normalization term $S_x^0(f)$ contains noise components $N^{(k)}(f)\, N^{(k)}(f)^*$ and no canceling of noise occurs.

Acknowledgments

We thank A. Huttunen, V. Koivunen, and R. White for useful discussion.

References

[1] I. F. Akyildiz, W. Y. Lee, M. C. Vuran, S. Mohanty, NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey, Computer Networks 50 (2006) 2127–2159.

[2] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: Proceedings of SODA 2007, the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, and Society for Industrial and Applied Mathematics, Philadelphia, 2007, pp. 1027–1035.

[3] L. Bixio, M. Ottonello, H. Sallam, M. Raffetto, C. S. Regazzoni, Signal classification based on spectral redundancy and neural network ensembles, in: Proceedings of CROWNCOM'09, the 4th International Conference on Cognitive Radio Oriented Wireless Networks and Communications, IEEE, 2009.

[4] L. Bixio, G. Oliveri, M. Ottonello, C. S. Regazzoni, OFDM recognition based on cyclostationary analysis in an open spectrum scenario, in: Proceedings of the IEEE 69th Vehicular Technology Conference, IEEE, 2009.

[5] G. R. Bolt, J. Austin, G. Morgan, Fault tolerant multi-layer perceptron networks, Tech. Rep. YCS 180, University of York, UK (1992).

[6] A. G. Bors, M. Gabbouj, Minimal topology for a radial basis functions neural network for pattern classification, Digital Signal Process. 4 (3) (1994) 173–188.

[7] R. Boustany, J. Antoni, Cyclic spectral analysis from the averaged cyclic periodogram, in: P. Zitek (Ed.), Proceedings of IFAC'05, the 16th IFAC World Congress, 2005.

[8] Q. Cao, J. Rogers, Ultrathin films of single-walled carbon nanotubes for electronics and sensors: A review of fundamental and applied aspects, Adv. Mater. 21 (1).

[9] S. Cavalieri, O. Mirabella, A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks, Neural Networks 12 (1999) 91–106.

[10] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm` (2001).

[11] R. D. Clay, C. H. Sequin, Fault tolerance training improves generalization and robustness, in: Proceedings of IJCNN, the International Joint Conference on Neural Networks, Vol. 1, IEEE, 1992, pp. 769–774.

[12] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297.

[13] R. Eickhoff, U. Rückert, Enhancing fault tolerance of radial basis functions, in: Proceedings of IJCNN'06, the 2006 International Joint Conference on Neural Networks, IEEE, 2006, pp. 5066–5073.

[14] R. Eickhoff, U. Rückert, Robustness of radial basis functions, Neurocomputing 70 (2007) 2758–2767.

[15] A. Fehske, J. Gaeddert, J. H. Reed, A new approach to signal classification using spectral correlation and neural networks, in: Proceedings of DySPAN 2005, the First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, IEEE, 2005, pp. 144–150.

[16] M. Gandetto, M. Guainazzo, C. Regazzoni, Use of time-frequency analysis and neural networks for mode identification in a wireless software-defined radio approach, EURASIP Journal on Applied Signal Processing 12 (2004) 1778–1790.

[17] W. A. Gardner, A. Napolitano, L. Paura, Cyclostationarity: half a century of research, Signal Process. 86 (2006) 639–697.

[18] A. K. Geim, K. S. Novoselov, The rise of graphene, Nature Materials 6 (2007) 183–191.

[19] S. Haykin, Cognitive radio: brain-empowered wireless communications, IEEE J. Sel. Areas Commun. 23 (2005) 201–220.

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

[21] K. Ho, C.-s. Leung, J. Sum, On weight-noise-injection training, in: M. Köppen, N. Kasabov, G. Coghill (Eds.), Advances in Neuro-Information Processing (Proceedings of ICONIP 2008), Part II, LNCS 5507, Springer, Berlin Heidelberg, 2008, pp. 919–926.

[22] K. Jensen, J. Weldon, H. Garcia, A. Zettl, Nanotube radio, Nano Letters 7 (2007) 3508–3511.

[23] 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, R. S. Williams, Circuit fabrication at 17 nm half-pitch by nanoimprint lithography, Nano Letters 6 (2006) 351–354.

[24] K. Kim, I. A. Akbar, K. K. Bae, J. S. Um, C. M. Spooner, J. H. Reed, Cyclostationary approaches to signal detection and classification in cognitive radio, in: Proceedings of DySPAN 2007, the 2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, IEEE, 2007, pp. 212–215.

[25] J. H. Lee, K. K. Likharev, Defect-tolerant nanoelectronic pattern classifiers, Int. J. Circuit Theory Appl. 35 (3) (2007) 239–264.

[26] C. S. Leung, J. P. F. Sum, A fault-tolerant regularizer for RBF networks, IEEE Trans. Neural Networks 19 (2008) 493–507.

[27] B. Lin, B. Lin, F. Chong, F. Lai, Higher-order-statistics-based radial basis function networks for signal enhancement, IEEE Trans. Neural Networks 18 (3) (2007) 823–832.

[28] J. Mitola III, G. Q. Maguire Jr., Cognitive radio: making software radios more personal, IEEE Pers. Commun. 6 (1999) 13–18.

[29] A. F. Murray, P. J. Edwards, Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training, IEEE Trans. Neural Networks 5 (1994) 792–802.

[30] X. Parra, A. Català, Learning fault-tolerance in radial basis function networks, in: M. Verleysen (Ed.), Proceedings of ESANN2001, the 9th European Symposium on Artificial Neural Networks, D-Facto, Brussels, Belgium, 2001, pp. 341–346.

[31] A. Pärssinen, R. Kaunisto, A. Kärkkäinen, Future of radio and communication, in: T. Ryhänen, M. A. Uusitalo, O. Ikkala, A. Kärkkäinen (Eds.), Nanotechnologies for Future Mobile Devices, Cambridge University Press, in press.

[32] P. Pasanen, M. A. Uusitalo, V. Ermolov, J. Kivioja, C. Gamrat, Computing and information storage solutions, in: T. Ryhänen, M. A. Uusitalo, O. Ikkala, A. Kärkkäinen (Eds.), Nanotechnologies for Future Mobile Devices, Cambridge University Press, in press.

[33] J. Peltonen, M. A. Uusitalo, J. Pajarinen, Nano-scale fault tolerant machine learning for cognitive radio, in: Proceedings of IEEE Int. Workshop Machine Learning for Signal Process., IEEE, 2008, pp. 163–168.

[34] D. S. Phatak, Relationship between fault tolerance, generalization and the Vapnik-Chervonenkis (VC) dimension of feedforward ANNs, in: Proceedings of IJCNN'99, the International Joint Conference on Neural Networks, Vol. 1, IEEE, 1999, pp. 705–709.

[35] D. S. Phatak, I. Koren, Complete and partial fault tolerance of feedforward neural nets, IEEE Trans. Neural Networks 6 (2) (1995) 446–456.

[36] J. Ramirez-Angulo, G. Ducoudray-Acevedo, R. Carvajal, A. Lopez-Martin, Low-voltage high-performance voltage-mode and current-mode WTA circuits based on flipped voltage followers, IEEE Trans. Circuits Syst. Express Briefs 52 (7) (2005) 420–423.

[37] B. Razavi, Design of Analog CMOS Integrated Circuits, McGraw-Hill, Boston, MA, 2001.

[38] V. Sazonova, Y. Yaish, H. Üstüunel, D. Roundy, T. A. Arias, P. L. McEuen, A tunable carbon nanotube electromechanical oscillator, Nature 431 (2004) 284–287.

[39] B. E. Segee, M. J. Carter, Comparative fault tolerance of parallel distributed processing networks, IEEE Trans. Comput. 43 (1994) 1323–1329.

[40] C. H. Sequin, R. D. Clay, Fault-tolerance in artificial neural networks, in: Proceedings of the 1990 IJCNN International Joint Conference on Neural Networks, Vol. 1, IEEE, 1990, pp. 703–708.

[41] R. Sinha, C. Papadopoulos, J. Heidemann, Internet packet size distributions: some observations, Tech. Rep. ISI-TR-2007-643, University of Southern California (May 2007).

[42] J. Sum, C.-s. Leung, K. Ho, On node-fault-injection training of an RBF network, in: M. Köppen, N. Kasabov, G. Coghill (Eds.), Advances in Neuro-Information Processing (Proceedings of ICONIP 2008), Part II, LNCS 5507, Springer, Berlin Heidelberg, 2008, pp. 324–331.

[43] P. D. Sutton, K. E. Nolan, L. E. Doyle, Cyclostationary signatures in practical cognitive radio applications, IEEE J. Sel. Areas Commun. 26 (2008) 13–24.

[44] E. B. Tchernev, R. G. Mulvaney, D. S. Phatak, Investigating the fault tolerance of neural networks, Neural Computation 17 (7) (2005) 1646–1664.

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

[46] S. Watkins, P. Chau, R. Tawel, A radial basis function neurocomputer implemented with analog VLSI circuits, in: Proceedings of IJCNN, the International Joint Conference on Neural Networks, Vol. 2, IEEE, 1992, pp. 607–612.

[47] B. E. White, Energy-harvesting devices: Beyond the battery, Nature Nanotechnology 3 (2008) 71–72.

[48] H. Ye, Z. Gu, T. Yu, D. Gracias, Integrating nanowires with substrates using directed assembly and nanoscale soldering, IEEE Trans. Nanotechnol. 5 (1) (2006) 62–66.

[49] Z. H. Zhou, S. F. Chen, Z. Q. Chen, Improving tolerance of neural networks against multi-node open fault, in: Proceedings of IJCNN'01, the International Joint Conference on Neural Networks, Vol. 3, IEEE, 2001, pp. 1687–1692.