Datalogisk Institut
KØBENHAVNS UNIVERSITET

DIKU

Mote-based Online Anomaly Detection using Echo State Networks

Marcus Chang, Andreas Terzis, Philippe Bonnet

Joint technical report with Johns Hopkins University

Technical Report no. 09/01
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Mote-based Online Anomaly Detection using Echo State Networks

Marcus Chang[1], Andreas Terzis[2], and Philippe Bonnet[1]

[1] Dept. of Computer Science, University of Copenhagen, Copenhagen, Denmark
[2] Dept. of Computer Science, Johns Hopkins University, Baltimore MD, USA

Abstract. Sensor network deployments are plagued with measurement *faults* due to hardware and software defects. At the same time, networks should autonomously adapt to *events* they sense, for example by increasing their sampling rate or raising alarms. In this work we unify fault and event detection under a general *anomaly detection* framework in which motes are trained to recognize data from a training set as *normal* and measurements that significantly deviate from that set as *anomalies*. We implement an anomaly detection algorithm using Echo State Networks (ESN), a family of sparse neural networks, on a mote-class device and show that its accuracy is comparable to a PC-based implementation. Furthermore, we show that ESNs detect more faults and have fewer false positives than threshold-based fault detection mechanisms. More importantly, while rule-based fault detection algorithms generate false negatives and misclassifications when exposed to multiple faults and events, ESNs are *general*, correctly identifying a wide variety of anomalies.
Keywords: Anomaly detection, Real-time, Wireless Sensor Networks

1 Introduction

Sensor networks deployed to collect scientific data (e.g., [1–3]) have shown that field measurements are inherently *noisy*, due to a variety of hardware and software faults. These faults must be detected to prevent polluting the experiment and wasting the network's resources. However, Gupchup et al. recently showed that algorithms that classify measurements which deviate from the recent past as faulty, tend to misclassify *events* as faults [4]. Events in this context, such as rain events in the case of soil moisture, are measurements that deviate from "normal" data patterns, yet they represent features of the underlying phenomenon. This behavior is undesirable because, unlike faults which we want to discard, events are the most important data that a mote collects, as they inform scientists about the characteristics of the observed environment. Furthermore, detection algorithms tailored to specific types of faults must run in parallel, leading to false positives when exposed to multiple types of faults [4].

In this work we unify fault and event detection under a more general *anomaly detection* framework, in which online algorithms classify measurements that significantly deviate from a learned model of data as anomalies. By including distinct, yet infrequent events in the training set we avoid the misclassification problem mentioned above thus allowing the system to distinguish faults from interesting events.

Obviously anomaly detection can and should also be done on a gateway that correlates data from multiple sensors. Nonetheless, we claim that online detection on motes is also very much relevant. We motivate this need through an example derived from one of our current projects [5]. Consider a set of motes equipped with chlorophyll and temperature sensors deployed under the surface of a lake and connected to a floating buoy via underwater modems. Furthermore, as the lake is in North-East Greenland it is covered with ice from September to mid-July, forbidding physical access to the motes. Finally, the communication links between some of the motes and the floating buoy can be non-functional over long time periods, either because the buoy is out of communication range, or due to a hardware fault on the acoustic modem, or because the background noise in the lake is elevated during a storm. Irrespective of the intermittent availability of the communication links, the motes should collect data and temporarily store them to their local flash memory. Furthermore, the motes should be able to autonomously alter their sensing behavior depending on whether the collected measurements are seemingly faulty or correspond to interesting events.

The general consensus so far has been that learning-based techniques are too resource intensive to be implemented on mote-class devices. In this paper we challenge this belief, showing instead that a neural network variant called Echo State Network (ESN) [6], can indeed be implemented on a TelosB mote [7]. More importantly, the ESN can effectively detect measurement sequences that contain multiple categories of anomalies that do not exist in the training data.

In summary, the contributions of this paper are as follows: **(1)** we develop an anomaly detection framework based on the Echo State Network. **(2)** We implement this framework on a mote-class device. **(3)** We quantitatively compare the ESN with two rule-based fault detection techniques. Specifically, we show that an ESN small enough to function alongside a fully-functional environmental monitoring mote application, is still more sensitive to subtler faults and generates fewer false positives than the two rule-based fault detection techniques.

2 Related Work

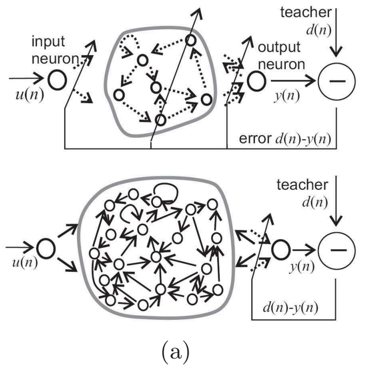
Anomaly characterization and detection has received significant attention in the sensor network community, yielding a broad range of algorithmic approaches.

Omitaomu et al. used probabilistic Principal Component Analysis to detect suspicious trucks at weighing stations on US interstate highways [8]. Wu et al. used geometric algorithms to detect outliers in precipitation data from South America [9], while Kaplantzis et al. used Support Vector Machines to detect network anomalies [10]. While differing on the specific mechanisms used, all these algorithms partition the data into subsets and detect anomalies by identifying the data points that do not belong to any subset. However, the temporal relations among data points are lost in such outlier detection algorithms. We seek a solution that not only considers each data point in isolation, but also the context in which it appears.

This context aware detection can be recast as a pattern recognition problem. Rashidi et al. built a framework for pattern mining and detection [11] and Römer used conditional rules to define anomalies [12]. However, neither of these solutions operate directly on raw sensor data; they rather assume that descriptive labels already exist. This classification is supposed to be performed by applying simple rules and thresholds to the raw data. The accuracy of both methods thereby depends on those rules.

Sensor networks have extensively used rule- and threshold-based anomaly detection schemes due to their simplicity. For example, Hu et al. monitored Cane toads by recording audio samples on Mica2 motes and used a simple threshold technique to trigger data transmission [13]. Werner-Allen et al. used Exponentially Weighted Moving Averages (EWMA) as a rule to detect seismological events on TelosB motes [14]. Furthermore, Sharma et al. proposed two rules to detect faults that are commonly observed in environmental monitoring: Short faults and Noise faults. A Short fault is characterized by a drastic difference between the current and the previous measurement, while a Noise fault is a period during which the measurements exhibit larger than normal variations. To detect the former, the Short rule compares two adjacent data points and flags the more recent as faulty when the difference is above a certain threshold. To detect the latter, the Noise rule considers a sliding window of measurements and flags all measurements in the window as faulty if the standard deviation is above a certain threshold [15]. Outside the field of environmental monitoring, Pister et al. deployed Rene motes from a UAV to detect moving vehicles in one of the first WSN field experiments [16]. They also used a simple threshold to determine whether a vehicle was present or not. While these schemes are very efficient, their effectiveness is limited. Werner-Allen et al. estimated the accuracy of their detection technique to be as low as 5%-29% [14]. Moreover, Gupchup et al. showed that these schemes suffer from inherent misclassification problems [4]. We thus seek a solution based on machine learning.

The use of machine learning as an anomaly detection tool has been proposed in the context of WSNs. However, most efforts thus far have focused on post-processing instead of online detection. In fact, Obst et al. also used Echo State neural networks to track gas readings from coal mines [17], while Wang et al. performed a similar study using Bayesian networks [18]. Bokareva et al. proposed another neural network variant called Competitive Learning Neural Network (CLNN) and used it to perform in-network classification [19]. However, their algorithm was running on a Stargate gateway instead on mote-class devices. We bridge the gap between online detection and machine learning by implementing a sophisticated detection technique on a mote.



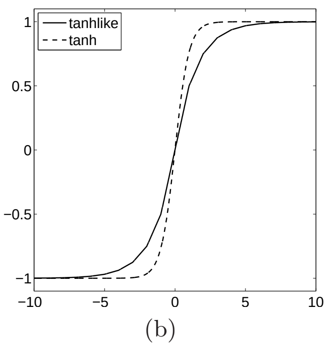


Fig. 1. (a) Traditional Neural Network (top) and Echo State Network (bottom). (b) tanh() and *tanhlike* neuron activation functions.

3 Machine Learning

We define *anomalies* as the measurements that significantly deviate from learned data. We rely on machine learning techniques to define our classification model. We focus on supervised learning because our scientific partners are able to provide training sets (i.e., data that were collected manually in the past, or synthesized based on a analytical model). These sets correspond to the data scientists expect: the valid data. The proposed classification mechanism should then accept measurements that match the model as valid and reject everything else as anomalies. This mode of operation rules out learning techniques such as support vector machines that assign measurements to one of the classes defined in the training set.

Kalman filters can be used for online detection by applying a threshold to the difference between the predicted and the measured values. However, they cannot be used for predictions based on a compact model of learned data as each prediction is based on a sliding window of observed values. On the other hand, Bayesian networks can perform classifications based on learned models by using causality graphs and probability functions to produce likelihood estimates of the observed values (evidence). However, this inference requires a graph reduction operation (which is NP-complete) and modeling of the probability function (typically a Gaussian) which can also be expensive on resource constrained devices.

Consequently, we decided to use echo state networks, a form of neural networks that promises to meet our requirements in terms of classification efficiency (i.e., minimize false classifications) and resource use (i.e., minimize CPU, RAM, ROM, and energy usage).

3.1 Neural Networks

A neural network can be informally considered as an approximation function. Specifically, when presented with a subset of the original function's value pairs during the *training* stage, the neural network generalizes over these data and approximates the outcomes of the original function in the *prediction* stage.

Formally, a neural network is a weighted directed graph whose each vertex is represented as a neuron. We consider discrete-time networks consisting of K input neurons, N hidden neurons, and L output neurons (see Figure 1a). The input neurons act as sources and the output neurons as sinks. At time step n, the input neurons are given the signal $\boldsymbol{u}(n)$ and the output neurons provide the signal $\boldsymbol{y}(n)$. The value of neuron j is given by: $v_j = \mathrm{A}(\sum w_{ij} v_i)$, where v_i is the output of neuron i, w_{ij} is the weight of the edge connecting neuron i to j, and A() is the *activation function*. This function is typically tanh() or a similar function (see Figure 1b).

The training stage consists of adjusting the network's weights to approximate the output signal to the training signal. Training is done offline, leaving only the weighted directed graph and the activation function to be implemented on a mote. Thereby, resource efficiency depends on the graph's representation and the cost of evaluating the activation function.

3.2 Echo State Networks

We consider the special class of Echo State Networks (ESN) [6]. In a ESN, cycles are allowed and thus all hidden neurons can be connected with each other and themselves. Furthermore, output neurons can have back-propagating signals, making them both input and output neurons. There are several subtle differences between ESNs and the more traditionally-used Feed Forward networks. First, all neurons are interconnected (but can have zero-weighted edges) removing the notion of layers. Second, because cycles involving one or more neurons are allowed, each neuron has the capability to remember, adding memory to the network as a whole. Third, all neurons' connections, directions, and weights are generated randomly and do not change during training. The neurons thus act as a black box referred to as the Dynamic Reservoir (DR). Fourth, the only weights that change during training are the output weights. This last property reduces the learning algorithm to a simple linear regression.

According to the Echo State property, the DR contains a set of basis states and by adjusting the output weights it is possible to capture the 'echoes' of real states as linear combinations of these basis states. Although the DR is randomly generated, Jaeger proved that it is possible to ensure that the DR indeed has the Echo State property by enforcing certain conditions [6]. One such condition is that the DR must be sparsely connected, whereas 10% of all possible connections are actually active.

3.3 Anomaly Detection

We use ESNs to determine whether sensor readings are anomalous, by comparing the ESN predictions to the actual measurements. Since the ESN is a learning algorithm, we define all features in the training set as *normal* and all features not part of this set as *anomalies*. We train this system using the same signal as both input and teacher signal, except that we shift the teacher signal one step forward in time, i.e., at time $n = 0$ the input signal $\boldsymbol{u}(0)$ will be associated with the teacher signal $\boldsymbol{d}(0)$, whereas $\boldsymbol{d}(0) = \boldsymbol{u}(1)$.

In order to quantify the prediction error we look at the numerical differences between the measurements ($\boldsymbol{M}$) and the predictions ($\boldsymbol{P}$), i.e., $\boldsymbol{\delta} = \boldsymbol{M} - \boldsymbol{P}$. This difference should ideally be close to zero for *normal* data, while *anomalous* data should result in large differences (peaks). In other words, the ESN transforms the original time series into one whose values are mostly ~ 0, corresponding to the expected data. Anomaly detection thus reduces to recognizing the peaks in the transformed signal. We can then use pattern matching algorithms based on simple thresholds that have been proven to be both efficient and effective for such simple signals.

The peak recognition consists of three stages. The reason behind the stages is that even single data point anomalies have a wide peak (spanning multiple data points) in the prediction error. This is caused by the internal states of the ESN being perturbed each time an anomaly is encountered resulting in a settling time before the predictions stabilizes.

In the first stage we reduce the perturbations in the prediction error by applying a moving average on two consecutive data points: $\delta_{avg}(i) = \frac{1}{2}(|\delta(i-1)| + |\delta(i)|)$. We only consider two data points since more points will increase the width of the peak. In the second stage we mark data points as anomalies by applying two different thresholds to two consecutive data points: $\delta_{avg}(i) > \rho_i$ and $\delta_{avg}(i-1) > \rho_{i-1}$, where the second condition is only applied if the first is true. By using conditional thresholds we can decrease the number of false positives by setting ρ_i conservatively high, while at the same time decrease the number of false negatives by setting ρ_{i-1} lower. I.e., the first threshold is to detect the general location of the anomaly, while the second one is to ensure we detect the correct starting point. This will, however, lead to over-counting of anomalies. Thus, in the third stage we remove the extra marked data points by assuming that single point anomalies will at most be marked as four data points. Therefore, we remove the trailing marked data points for all anomalies at most four data points wide. We only perform this final stage on single point anomalies since the perturbations have varying effect on larger anomalies.

3.4 Discussion

The decoupling of the DR from the output weights enables several optimizations that fit WSNs particularly well. For instance, the same DR can be used for multiple tasks by storing task-specific output weights. This also facilitates post-deployment update since only the output weights need to be transmitted over the network instead of the entire DR. The requirement that the DR is sparsely connected is advantageous in the context of motes with limited storage. Furthermore, the use of sparse matrix algebra, allows the implementation on motes of ESNs that are larger than regular Feed Forward networks.

A limitation that ESNs share with all learning algorithms is their dependence on training data. In particular, if these data do not represent what domain scientists deem as "normal", the predictions will be useless. Therefore, the choice of training sets, and more interestingly the choice of classification technique based on the available training sets is a very interesting open problem, which is beyond the scope of this paper. We just note that a key issue in successfully deploying an ESN lies in the choice and availability of training data. For example, adjusting the sampling rate in an adaptive sampling environment can change the properties of the measurement time series and thus possibly invalidate the training set. This issue can however be remedied, by storing different output weights for each sampling rate, or by disregarding higher sampling rates when applying the ESN detection. On the positive side, ESNs have the ability to generalize over the training data. In other words, ESNs base their predictions on the *trends* of the presented data rather than *exact* values. This feature allows motes deployed in similar regions to share the same training data instead of requiring mote-specific training sets.

4 ESN on a Mote

4.1 Implementation

While we create and train the ESNs offline, a complete ESN including the network's activation mechanism, weight matrices and the DR is included in the application that runs on the mote, making it part of the program stored in ROM. On the other hand, we store the output weights in RAM to facilitate post-deployment updates of the ESN. We use TinyOS 2.x to ensure portability to a broad range of mote class devices. Our implementation, publicly available for download at [20], focuses on two issues: feasibility and efficiency. In other words, the ESN algorithm and the DR must fit in ROM and the ESN algorithm must be fast enough to maintain the desired sampling rate, while allowing the mote to return to sleep. In the paragraphs that follow we present three optimizations employed to improve performance along these two axes.

Sparse Matrix Algebra. The size of the DR's weight matrix grows quadratically with the number of neurons in the reservoir n. However, only 10% of these elements are non-zero because the DR must possess the Echo State property. We leverage this feature by storing the matrix using Compressed Row Storage, which only stores the non-zero elements and the layout of the matrix [21]. This reduces the necessary storage from $O(n^2)$ to $O(2n_z + n + 1)$, where n_z is the number of non-zero elements. This technique also reduces the number of operations needed to perform matrix multiplications by a similar factor since only non-zero elements are used.

Single Floating Point Precision. Most mote-class devices do not have hardware-accelerated floating point operations which means that they have to be emulated in software. This emulation contributes to both the storage overhead, since the floating point libraries have to be stored on the mote and the runtime overhead, as floating point operations in software are resource intensive. At the cost of

	$\boldsymbol{\delta}_{lab/tanh}$ (%)	$\boldsymbol{\delta}_{mote/tanh}$ (%)	$\boldsymbol{\delta}_{lab/tl}$ (%)	$\boldsymbol{\delta}_{mote/tl}$ (%)
NRMSD	0.15448	0.15470	0.5855014	0.5855008

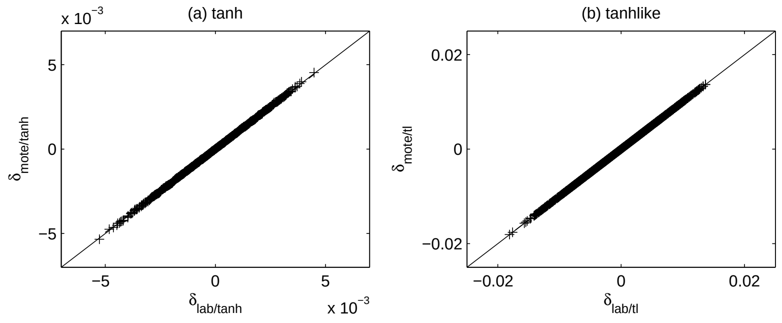


Fig. 2. (a) Q-Q plot of $\boldsymbol{\delta}_{mote/tanh}$ and $\boldsymbol{\delta}_{lab/tanh}$. (b) Q-Q plot of $\boldsymbol{\delta}_{mote/tl}$ and $\boldsymbol{\delta}_{lab/tl}$. The table above shows the normalized root-mean-squared deviation (NRMSD).

reduced floating point precision we select to store and compute all values using single instead of double floating point precision. Doing so halves the size of all the weight matrices and reduces the number of emulated floating point operations needed. As we later show, the resulting loss of precision is tolerable.

Tanhlike Activation Function. Because the activation function has to be applied to all the neurons and every iteration, it is important to choose an efficient function. At the same time, choosing a suboptimal activation function can significantly degrade the ESN's output quality. The often used hyperbolic tangent, tanh(), is a complex function requiring both large amounts of storage and a significant processing time. Because of these shortcomings, Marra et al. [22] proposed the approximation function:

$$TL(x) = \operatorname{sign}(x)\left[1 + \frac{1}{2^{\lfloor 2^n |x| \rfloor}}\left(\frac{2^n |x| - \lfloor 2^n |x| \rfloor}{2} - 1\right)\right]$$

where $n \in \mathbb{Z}$ determines the steepness of the function. We use $n = 1$ since this makes the function resemble tanh() the most (see Figure 1b). This *tanhlike* function has properties similar to tanh() but with far lower complexity. However, it is also a non-differentiable, piecewise-linear function because of the rounding operations ($\lfloor \cdot \rfloor$). Therefore, we expect the quality of the ESN's output to be lower than when using tanh(), because small changes in input will result in large changes in output if these changes happen across a linear junction.

4.2 Evaluation

We verify that our ESN implementation indeed performs well on a mote-class device by comparing its output to a reference ESN running on a PC. All mote experiments are carried out on a TelosB mote [7], running TinyOS 2.x with the clock frequency set to the default speed of 4 MHz [23]. All data sets are

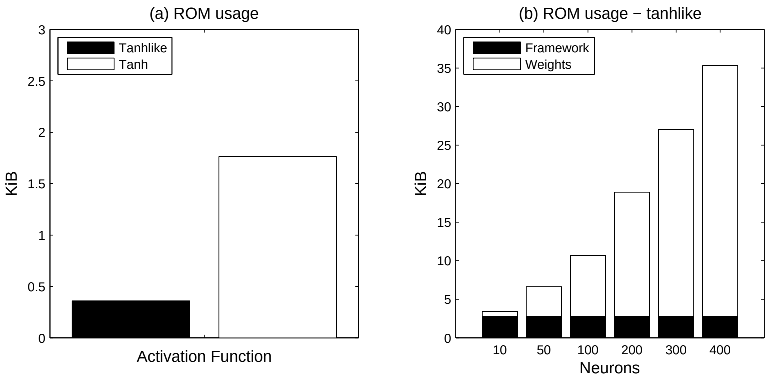


Fig. 3. (a) ROM footprints for the tanh() and *tanhlike* functions. (b) Total ROM footprint for an ESN using the custom *tanhlike* activation function.

stored in ROM with measurements read with a fixed frequency to simulate sensor sampling. We use Matlab R2007a with the *Matlab toolbox for ESNs* [24] as our reference implementation.

Sanity Check. To evaluate our ESN implementation we use the Mackey-Glass (MG) time series with a delay $\tau = 17$ [25]. This system is commonly used to benchmark time series prediction methods because of its chaotic nature. Jaeger et al. showed that an ESN with a 1000 neuron reservoir can predict over 1000 steps of this time series, (depending on the accuracy) when trained and initialized to a certain point in the MG time series [26]. Since we are primarily interested in predicting the immediate next reading we consider smaller reservoirs. Specifically, we consider ESNs which consist of two input signals (with one of the input signals held at a constant bias value in order to improve performance [27]), a 10-400 neuron reservoir, and one output signal (i.e., $K = 2, N = 10 - 400$, and $L = 1$).

We created a MG time series with 4,000 samples and used the first 2,000 samples to train a 50 neuron ESN, the next 1,000 samples for initialization, while the last 1,000 samples were used as the prediction vector $\boldsymbol{MG}$. Both the tanh() and *tanhlike* activation functions were used resulting in four different predictions: $\boldsymbol{P}_{lab/tanh}$, $\boldsymbol{P}_{mote/tanh}$, $\boldsymbol{P}_{lab/tl}$, and $\boldsymbol{P}_{mote/tl}$.

We compute the four prediction errors and normalized root-mean-squared deviations (NRMSD). We group the prediction errors by activation function and show the Q-Q plots of the error distributions in Figure 2 together with the NRMSDs. First, we notice that for each activation function both the NRMSDs are almost identical and the points in the Q-Q plot lie on a straight line with slope one (meaning the prediction errors belong to the same distribution). Based on these results, we conclude that the TelosB ESN implementation has the same accuracy as the one in Matlab. Second, the NRMSD values indicate that a 50-neuron ESN is indeed capable of tracking the $\boldsymbol{MG}$ time series with an error less

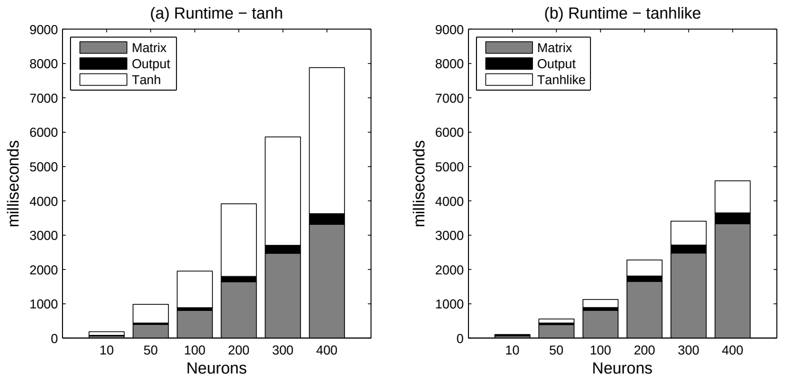


Fig. 4. Total execution cost of one ESN iteration divided to three components. (a) using the GCC built-in tanh() activation function. (b) using the custom tanhlike activation function.

than 1% for both the tanh() and *tanhlike* functions. Third, by comparing the NRMSDs for the same platforms we note that the choice of activation function has a significant impact on the accuracy of the predictions. In particular, tanh() is four times more accurate than the *tanhlike* function. As argued above, this discrepancy is probably caused by *tanhlike* being a piecewise-linear function.

To compare the single precision floating point on the TelosB with that of the double precision floating point in Matlab, we look at the differences between predictions from the latter with the former when using the same activation function, i.e., $\boldsymbol{\delta}_{tanh} = \boldsymbol{P}_{lab/tanh} - \boldsymbol{P}_{mote/tanh}$ and $\boldsymbol{\delta}_{tl} = \boldsymbol{P}_{lab/tl} - \boldsymbol{P}_{mote/tl}$. We compute the NRMSDs for both error distributions: $\boldsymbol{\delta}_{tanh} = 6.6 \cdot 10^{-3}$ % and $\boldsymbol{\delta}_{tl} = 1.3 \cdot 10^{-4}$ %. First, we note that both $NRMSD(\boldsymbol{\delta}_{tanh}) < NRMSD(\boldsymbol{\delta}_{lab/tanh})$ and $NRMSD(\boldsymbol{\delta}_{tl}) < NRMSD(\boldsymbol{\delta}_{lab/tl})$. This means that the prediction errors are larger than the errors caused by using single precision floating point instead of double. In other words, the use of single precision floating point operations on the TelosB has no negative effect on the accuracy of the predictions. Second, although the *tanhlike* activation function is less accurate than tanh() in terms of prediction, the difference between Matlab and TelosB is two orders of magnitude smaller when using *tanhlike*. The reason for this lies in the simplicity of the *tanhlike* algorithm when compared to the more complex tanh(), since the former uses fewer calculations which in turn causes fewer rounding errors.

Performance. Next, we explore the implementation's storage and computation requirements. The ROM usage can be divided into two components: (1) *Framework*, that is the ESN algorithm used for prediction, the activation function, and libraries needed for floating point emulation. (2) *Weight Matrices*, that is the DR and output weights. Whereas (1) is constant, (2) depends on the number of neurons in the reservoir. We thus vary the number of neurons when evaluating ROM footprint, runtime speed, and accuracy.

Figure 3a presents the ROM size difference for the two activation functions and Figure 3b shows the ROM footprint of the aforementioned components for a selected number of reservoir sizes when the *tanhlike* activation function is used. We observe the following: First, the memory contribution from the reservoir grows linearly. This confirms the estimated storage requirement of the Compressed Row Storage ($O(2n_z + n + 1)$) mentioned in Section 4.1. Second, the built-in tanh() function consumes five times more space than the custom *tanhlike* activation function. Specifically, the ROM footprint is 1,806 bytes for tanh() and 368 bytes for *tanhlike*. Third, emulating floating point operations in software requires 1,450 bytes, making it comparable to the size of the tanh() function, or a third of a 50-neuron reservoir.

Next we measure the runtime cost of the ESN implementation. This is done by reading the mote's internal timer after each major set of operations and computing the differences. For each iteration, the ESN prediction algorithm performs the following sets of operations: (1) *Matrix*, matrix-vector multiplication that transfers the ESN to the next state. (2) *Activation Function*, which updates the state of each neuron by applying the tanh() or *tanhlike* activation functions. (3) *Output*, vector-vector multiplication that returns the output by applying the output weights. Figure 4 summarizes the execution time of one prediction step and the contributions from each of the three operations. Surprisingly, the tanh() activation function is the most expensive operation. It takes 28% longer to run than the matrix-vector multiplication and 453% longer than the *tanhlike* activation function.

Finally, we look at the prediction error as a function of reservoir size and activation function. We compare against the MG time series and find the $NRMSD(\boldsymbol{\delta})$ for the six reservoirs and two activation functions used above. Figure 5 presents the results of this comparison. As expected, the prediction error decreases as the reservoir size increases and the $tanh()$ activation function leads to more accurate predictions in general. Upon closer inspection, there appear to be three distinct regions relative to the reservoir size: small (10 neurons), medium (50-300 neurons), and large (300-400 neurons). In the small region, the prediction error is dominated by the small size of the reservoir and the choice of activation function becomes less important. In the medium region there is a diminishing, yet clear reduction of the prediction error as the reservoir size increases. Finally, in the large region the prediction error does not decrease by adding neurons to the reservoir. Interestingly, the largest contribution to the prediction error comes from the activation function, with no overlap of prediction errors for the 50-400 neuron reservoirs. In fact, even the 50 neuron tanh() reservoir outperforms the 400 neuron *tanhlike* reservoir.

5 Evaluation

5.1 Experimental Design

The results from the previous section suggest that an ESN can be accurate, small, and fast enough to be incorporated to an existing data collection application that

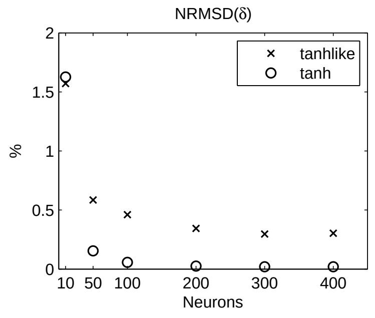


Fig. 5. $NRMSD(\delta_{lab/tanh})$ and $NRMSD(\delta_{lab/tl})$ for different reservoir sizes.

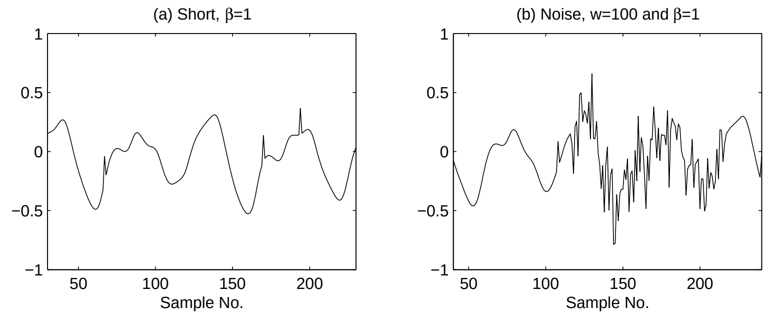


Fig. 6. Two types of injected anomalies: (a) Short faults and (b) Noise faults.

has been actively deployed for the past three years [**?**]. Motes in these sensor networks collect soil temperature and soil moisture readings every 20 minutes and store them to their onboard flash memory. All measurements are periodically offloaded over the network and persistently stored in a database.

This environmental sensing application uses 40,824 bytes of ROM and 3,928 bytes of RAM, leaving 8,328 bytes of available ROM and 6,312 bytes of free RAM on the TelosB. From the previous section we know that a 50-neuron ESN using the *tanhlike* activation function has a ROM footprint of 6,788 bytes and a prediction time of 572 ms for each measurement. Thereby such an ESN complies with both the storage and computation constraints of the application and will be used for the remainder of this section.

Anomaly Types. We focus on two types of random anomalies and one systematic anomaly. Specifically, we use the random anomalies defined by Sharma et al. [15] (i.e., Short and Noise faults) and presented in Section 2. Samples of these faults can be seen in Figure 6.

We use two parameters to control the injection of Short faults: the sample error rate and the amplification factor, β. For each anomalous measurement, $\tilde{m}_i$, we multiply the standard deviation of the original signal, σ, with β to obtain:

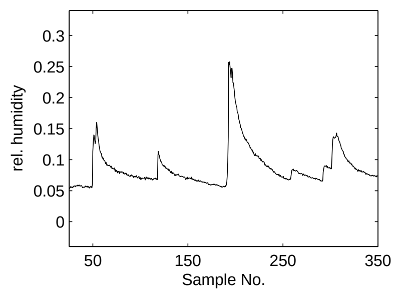


Fig. 7. Feature amplification anomaly.

$\tilde{m}_i = m_i + \beta\sigma$, where m_i is the true measurement. Similarly, for the injection of Noise faults we use three parameters: the sample error rate, the period length, w, and the amplification factor, β. For each noisy period, we calculate the standard deviation of the underlying signal and multiply it with β to create a random normal distribution with zero mean and $\beta\sigma$ standard deviation (i.e., $N(0, \beta\sigma)$). We then add samples from this distribution to each of the true measurements within that period.

Finally, we consider *feature amplification* as an example of systematic anomalies (see Figure 7). A feature amplification anomaly is defined as an increase in the amplitude of a set of measurements that constitute a feature (e.g. a rain event in the case of moisture measurements). The amplification factor is selected in a way that does not create discontinuities in the time series and makes the feature appear larger, while maintaining its shape. We use this anomaly to simulate extreme events, such as a severe rain event not present in the training set.

Detection Algorithms. We use the two rule-based anomaly detection algorithms defined by Sharma et al. [15] and summarized in Section 2 to detect the two anomalies mentioned above. We use these algorithms as reference as they are directly related to the anomalies we inject and their complexity is comparable to that of currently deployed fault detection algorithms. Our strategy for setting the thresholds is to minimize the number of false positives when the detection algorithms are applied to data sets with no anomalies.

Data Sets. For each of the soil temperature and soil moisture modalities that we use, we obtain a training and a test data set from the [**?**] database. Each of the four data sets consists of 1,000 data points. Figure 8 illustrates two such data sets. The data have been sanitized by removing data points with value changes beyond and below certain thresholds. This process is done automatically by the database as a standard procedure for removing anomalies, following the methods proposed by Sharma et al. [15]. By using this preprocessed data (instead of raw data) our results will not be biased by any anomalies already present in the data stream. Instead, we can assume that the only anomalies in the data are the ones we explicitly inject, thereby establishing the ground truth for evaluation purposes.

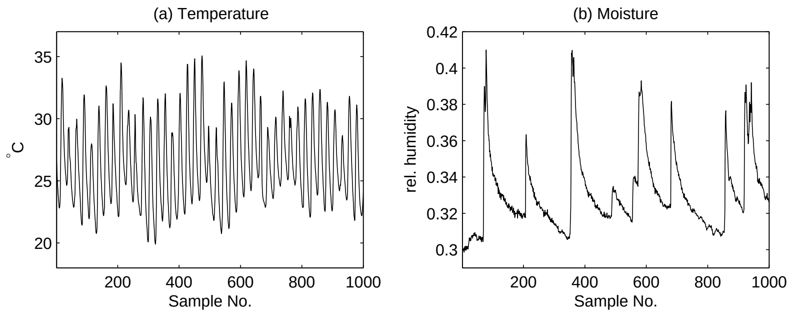


Fig. 8. Environmental sensing data sets. (a) Soil temperature and (b) Soil moisture.

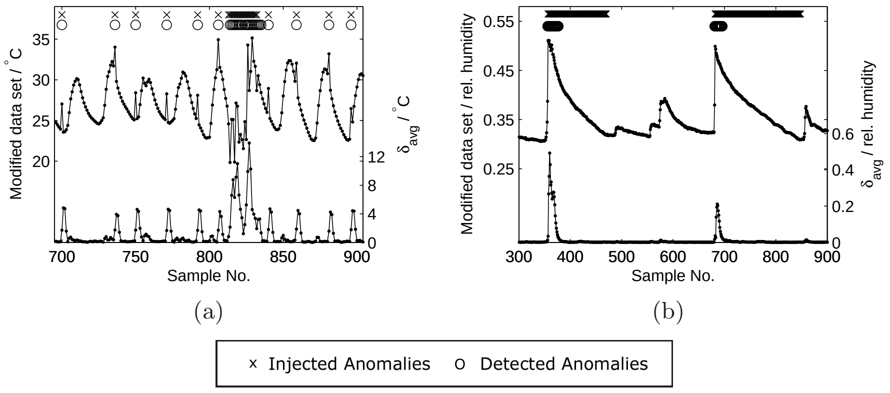


Fig. 9. Relation between measurement sequences (middle plot), ESN prediction errors (bottom plot), and detected anomalies (top markers). (a) Temperature measurements with injected Short and Noise faults ($\beta = 1, w = 20$). (b) Moisture data with amplification anomalies.

5.2 Results

Figure 9a illustrates the operation of the ESN anomaly detection algorithm by presenting the relation between the injected anomalies, the measurements – including the artificially added anomalies, the prediction error δ_{avg}, and the detected anomalies. Notice that the prediction error is indeed an almost constant signal overlaid with large peaks coinciding with the injected faults.

When not injected with anomalies we find that $NRMSD(\boldsymbol{\delta}_{Temp}) = 2.4\%$ and $NRMSD(\boldsymbol{\delta}_{Moist}) = 4.4\%$ for the temperature and moisture data set respectively. This accuracy is of the same order of magnitude as the one Obst et al. [17] found when tracking gas measurements, meaning that our online implementation is indeed comparable to the offline counterpart.

We use a 5% sample error rate (i.e., 5% of the measurements are polluted with errors) for each fault type and a period $w = 10$ for Noise faults. The amplifications used for the evaluation are: $1 \leq \beta \leq 5$. Figure 10 compares the

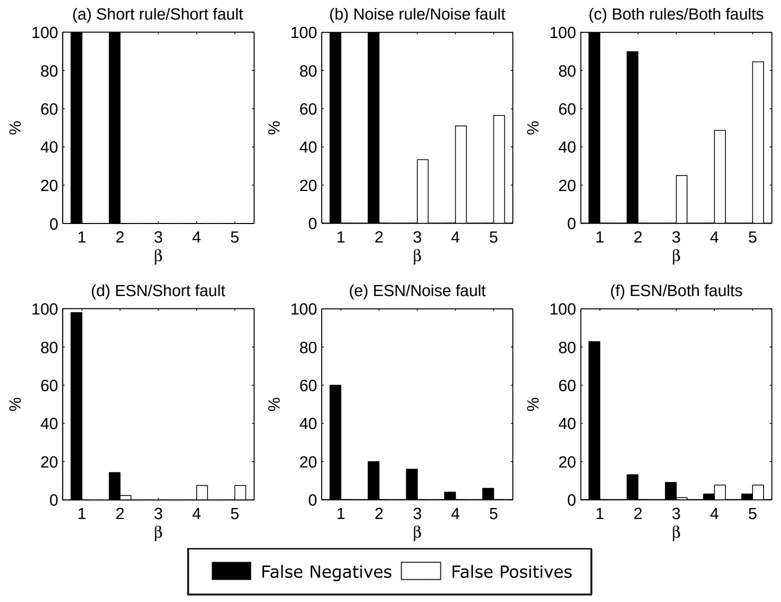


Fig. 10. Short rule, Noise rule, and ESN detection applied to the moisture data set.

three algorithms in the case of moisture data when applied to the Short faults, Noise faults, and a combination of both faults (5% Short and 5% Noise faults). We only apply each rule to its own domain fault since this is the optimal scenario. The challenge of this data set is the similarity between the onset of rain events and Short faults. In order to avoid false positives the thresholds must be set high enough to avoid triggering the Short rule during the rain events.

In the left column, Figure 10(a,d), we compare the Short rule with the ESN detection when applied to Short faults. Not surprisingly the Short rule performs well on this type of faults when $\beta \geq 3$. However, for lower β values the Short rule cannot distinguish between rain events and faults, and detects none of the latter. The ESN is effective for $\beta \geq 2$ but at the cost of more false positives at higher βs.

In the middle column, Figure 10(b,e), we compare the Noise rule with the ESN detection when applied to Noise faults. Interestingly, the Noise rule does not perform well on its corresponding faults. At $\beta \geq 3$ we see the same trend as before with no false negatives, however, we also see a significant number of false positives. This behavior is caused by the aggressiveness of the Noise rule, marking the entire window as faulty rather than individual points. For low β values we still see the ambiguity between events and faults, leading to no positive detections. The ESN detector, however, has no false positives, and a significantly lower number of false negatives for $\beta \leq 2$. Finally, for higher β values the number of false negatives is also significantly smaller than the number of false positives of the rule-based algorithm.

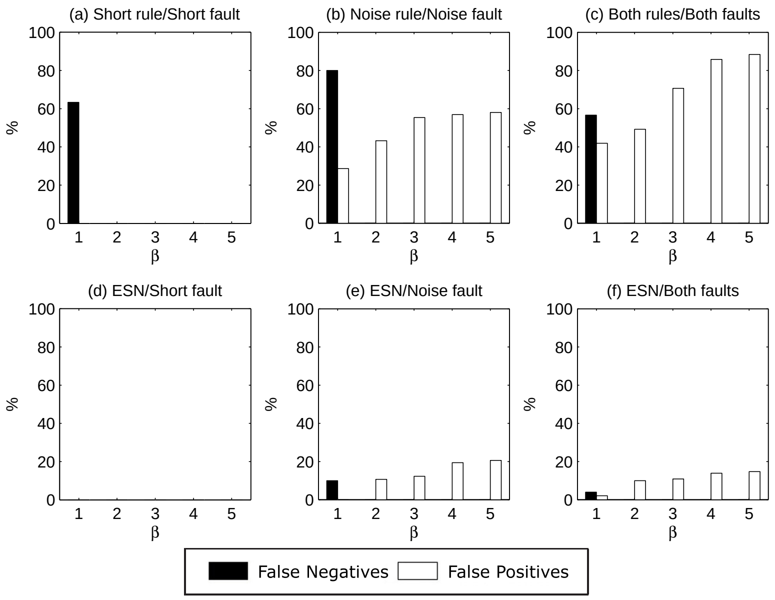


Fig. 11. Short rule, Noise rule, and ESN detection applied to the temperature data set.

Judging by these results, we conclude that the ESN can match up with the rule based detectors. There is although a trade-off between false positives and false negatives, since decreasing one often leads to the increase of the other. However, in a real deployment it is not possible to choose what algorithm to use on which faults and we must assume that all faults can appear at anytime. In the right column, Figure 10(c,f), we thus compare a hybrid detector using both the Short rule and the Noise rule at the same time on a data set injected with both types of faults. We see that the hybrid detector has the same behavior as the Noise rule, with either high number of false negatives or false positives. On the other hand, the ESN detector is performing significantly better across all β values, illustrating the strength of the learning algorithm's ability to detect what is not *normal*.

Next, we perform the same analysis on the temperature data set, using the same parameters to inject errors. The challenge of this data set from the perspective of a detection algorithm is the high temperature variance, caused by the diurnal pattern, that resembles noise faults. As before, the Short rule and faults are in the left column (Figure 11(a,d)), Noise rule and faults in the middle column (Figure 11(b,e)), and the hybrid detector on both types of faults in the right column (Figure 11(c,f)). One can see that the overall accuracy improves significantly, with more faults being detected. Also note that the Noise rule generates a large number of false positives, supporting the claim that the diurnal temperature patterns in the data set can be misclassified as Noise faults. Again, when used on both faults simultaneously we see that the false positives is the biggest drawback with the hybrid detector. The ESN detector, however, does not misclassify to the same extent, again clearly showing the ESN's ability to distinguish between *normal* and *anomalous* data.

Last, we apply the ESN detection to the amplification anomaly. Figure 9b shows the injected anomaly and the measurements successfully marked as anomalous. Although the ESN correctly reacts to the injected anomalies, only the leading part of the anomaly is marked. The reason for this behavior is that only the leading edge of the anomaly is truly unique in the sense that after 30-50 measurements the anomaly resembles features present in the training set. Since the ESN has a neuron reservoir of only 50 nodes, the memory of the anomaly quickly fades.

5.3 Discussion

We have shown that, for the modalities we tested, the ESN is capable of detecting low-amplitude anomalies better than specific rule-based anomaly detectors. At the same time, it is equally effective over multiple anomaly types, as it has the ability to detect a wide range of features deviating from the training data. There are, however, several factors that limit the applicability of ESNs. We identify three key issues: (1) As we saw in Section 4.2 the prediction time for each iteration is in the order of seconds. For environmental monitoring, where changes happen on the scale of minutes, this prediction speed is acceptable. However, this technique might not be feasible for high data rate applications. (2) For deployments in which no historical data are available, the training data will have to be constructed (e.g., from models, experience, etc.) or learned during the deployment. Neither options are desirable, because an artificial training set will lack the details encountered in the field. (3) Because the ESN is an approximation function, its quality is highly dependent on the size of the dynamic reservoir (DR). In the case of soil moisture and temperature a DR of 50 neurons suffices for anomaly detection. However, given a different set of constraints the DR might not be large enough to encode the dynamics of the underlying modality.

6 Conclusion

This paper unifies fault and event detection in sensor networks under the general framework of anomaly detection. We show that online anomaly detection is feasible on mote-class devices by implementing an Echo State Network (ESN) on a TelosB mote. This network performs as well as a PC-based ESN of the same size, proving that it is feasible to implement sophisticated pattern recognition algorithms on motes. Indeed, the ESN is small and fast enough to function alongside an environmental monitoring application, detecting measurement anomalies in real-time. Depending on the amplitude of the injected anomalies, the ESN provides equivalent or higher detection accuracy compared to rule-based detectors customized to specific faults. However, the most significant feature of the ESN detector is its generality since it is capable of detecting all features not present in the training set.

In our future work we will explore the feasibility of implementing other machine learning techniques, such as Bayesian networks, on mote-class devices and compare their performance to ESNs. With different methods available, the challenge becomes how to choose the best supervised learning method for mote-based online classification when given a particular training set from the domain scientists.

References

1. Musăloiu-E., R., Terzis, A., Szlavecz, K., Szalay, A., Cogan, J., Gray, J.: Life Under your Feet: A WSN for Soil Ecology. In: EmNets Workshop. (May 2006)
2. Selavo, L., Wood, A., Cao, Q., Sookoor, T., Liu, H., Srinivasan, A., Wu, Y., Kang, W., Stankovic, J., Young, D., Porter, J.: LUSTER: Wireless Sensor Network for Environmental Research. In: ACM SenSys. (November 2007)
3. Tolle, G., Polastre, J., Szewczyk, R., Turner, N., Tu, K., Buonadonna, P., Burgess, S., Gay, D., Hong, W., Dawson, T., Culler, D.: A Macroscope in the Redwoods. In: ACM SenSys. (November 2005)
4. Gupchup, J., Sharma, A., Terzis, A., Burns, R., Szalay, A.: The Perils of Detecting Measurement Faults in Environmental Monitoring Networks. In: DCOSS. (2008)
5. MANA: Monitoring remote environments with Autonomous sensor Network-based data Acquisition systems. `http://mana.escience.dk/`
6. Jaeger, H.: The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology (2001)
7. Polastre, J., Szewczyk, R., Culler, D.: Telos: Enabling Ultra-Low Power Wireless Research. In: IPSN/SPOTS. (April 2005)
8. Omitaomu, O.A., Fang, Y., Ganguly, A.R.: Anomaly detection from sensor data for real-time decisions. In: Sensor-KDD, Las Vegas, Nevada, USA (August 2008)
9. Wu, E., Liu, W., Chawla, S.: Spatio-temporal outlier detection in precipitation data. In: Sensor-KDD, Las Vegas, Nevada, USA (August 2008)
10. Kaplantzis, S., Shilton, A., Mani, N., Sekercioglu, A.: Detecting selective forwarding attacks in wsn using support vector machines. In: ISSNIP. (2007)
11. Rashidi, P., Cook, D.J.: An adaptive sensor mining framework for pervasive computing applications. In: Sensor-KDD, Las Vegas, Nevada, USA (August 2008)
12. Römer, K.: Distributed mining of spatio-temporal event patterns in sensor networks. In: EAWMS at DCOSS. (Jun 2006)
13. Hu, W., Tran, V.N., Bulusu, N., Chou, C.T., Jha, S., Taylor, A.: The design and evaluation of a hybrid sensor network for cane-toad monitoring. In: IPSN. (2005)
14. Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity and yield in a volcano monitoring sensor network. In: OSDI. (2006)
15. Sharma, A., Golubchik, L., Govindan, R.: On the Prevalence of Sensor Faults in Real-World Deployments. IEEE SECON (2007)
16. Pister, K.: Tracking vehicles with a UAV-delivered sensor network. Available at http://robotics.eecs.berkeley.edu/ pister/29Palms103/ (March 2001)
17. Obst, O., Wang, X.R., Prokopenko, M.: Using echo state networks for anomaly detection in underground coal mines. In: IPSN. (April 2008)

18. Wang, X.R., Lizier, J.T., Obst, O., Prokopenko, M., Wang, P.: Spatiotemporal anomaly detection in gas monitoring sensor networks. In: EWSN. (2008) 90–105
19. Bokareva, T., Bulusu, N., Jha, S.: Learning sensor data characteristics in unknown environments. In: IWASN. (2006)
20. Chang, M., Terzis, A., Bonnet, P. `http://www.diku.dk/~marcus/esn/`
21. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H.: Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM (2000)
22. Marra, S., Iachino, M., Morabito, F.: Tanh-like activation function implementation for high-performance digital neural systems. Research in Microelectronics and Electronics 2006, Ph. D. (June 2006) 237–240
23. Moteiv Corporation: Tmote Sky. `http://www.moteiv.com/`
24. Herbert Jaeger: Matlab toolbox for ESNs. Available at `http://www.faculty.jacobs-university.de/hjaeger/pubs/ESNtools.zip` Last checked: 2008-08-31
25. Mackey, M.C., Glass, L.: Oscillation and Chaos in Physiological Control Systems. Science **197**(287) (1977)
26. Jaeger, H., Haas, H.: Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Telecommunication. Science **304**(5667) (2004) 78–80
27. Jaeger, H.: Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the echo state network approach. Technical Report GMD Report 159, German National Research Center for Information Technology (October 2002)