STDP Training Design and Performances of a SNN for Sequence Detection as a Wake Up Radio in a IoT Network

Pierrick Joseph, Guillaume Marthe, Claire Goursaud

STDP training design and performances of a SNN for sequence detection as a wake up radio in a IoT network

Pierrick Joseph
CITI, UR3720
INSA Lyon, Inria
69621 Villeurbanne, France
pierrick.joseph@insa-lyon.fr

Guillaume Marthe
CITI, UR3720
INSA Lyon, Inria
69621 Villeurbanne, France
guillaume.marthe@insa-lyon.fr

Claire Goursaud
CITI, UR3720
INSA Lyon, Inria
69621 Villeurbanne, France
claire.goursaud@insa-lyon.fr

***Abstract*—Wake up radio receivers are an efficient approach to reduce energy consumption in Internet of Things (IoT) nodes. They monitor the channel to identify when a specific activation sequence is transmitted. However, up to now, they rely on classical processors which still consume too much. In this paper, to reduce this energy consumption, we propose to exploit Spiking Neural Network (SNN) architecture. In order to be able to detect the sequence in various channel conditions, we exploit the Spike-Timing Dependent Plasticity (STDP) learning rule to train the SNN. We show that the SNN succeeds in performing the sequence detection accurately, and delve into the training set design to improve the accuracy.**

I. Introduction and Related Work

During the last decade, the Internet of Things (IoT) has become a major field of interest in the telecommunications area [1]. One of the most important challenges in IoT remains to reduce the energy consumption of the multitude of nodes we use. One of the solutions that has been implemented for a few years is the Wake-up Radio (WuR), mostly used for devices that communicate sporadically [2].

The WuR is a circuit whose purpose is to monitor the channel and awake the main device only when needed. The WuR processes the incoming signals and if a specific predefined code is present, it awakes the main circuit. The energy saving comes from the WuR consuming much less energy than the microcontroller of the main device circuit [3]. However, one of the issues is that classical microcontrollers dedicated to such pattern recognition still consume too much energy [4]. Indeed, even current WuRs consume more energy than what the battery can provide during the expected lifespan of an IoT node. Thus, we need to further reduce the consumption of these WuRs [5].

Meanwhile, the Spiking Neural Network (SNN) technology is emerging as a very low power data processing solution. This bio-inspired system can compute as fast as actual processing devices but consumes much less energy for the same functions. The network's neurons communicate by mimicking biological neurons, sending spikes with precise timing, unlike other conventional approaches that transmit digital data. It was primarily used for image recognition [6] or to model real brain activity [7]. A few companies have begun to sell industrial solutions like the TrueNorth circuit from IBM [8]. Besides, with its analog behavior, SNN also permits to eliminate the energy-consuming synchronization. SNN is thus a promising candidate for WuR [9].

The adaptation of SNN for WuR is not straightforward. Indeed, previous studies focused on **calibrated** signals, i.e., signals whose scale is controlled [6, 7, 9], but, for WuR signals, the signal to process can vary very significantly even within any given class, due to the channel impact. Nonetheless, to the best of our knowledge, no study has considered the usage of SNNs for a WuR application using bio-inspired learning such as STDP (Spike-Timing Dependent Plasticity) [10]. In this paper, we thus focus on the ability of a trained SNN to process IoT signals despite their wide range, and delve into the training design to obtain good performance. We want to realize an offline training, to ensure low energy consumption that does not scale with use.

The paper is organized as follows. Section II presents the models used in our simulations by introducing the system and the architecture of the network alongside the learning algorithm used in this work. Section III details the different hypothesis and experiments made for our simulations, notably the different considered types of channel impact. It also gathers the performance results for the mentioned cases. Finally, Section IV concludes the paper.

II. System model

A. Network model

We consider a network where N nodes are deployed over a wide area. The base station can wake up any node by sending its dedicated specific sequence. A sequence can be modeled as a vector $x \in \{0,1\}^{SF}$ with SF (spread factor) *chips*, and takes binary values.

At the receiver side, the signal writes :

$$\boldsymbol{y} = \boldsymbol{x} \cdot h + \boldsymbol{n}, \tag{1}$$

where $\boldsymbol{y} \in \{\mathbb{R}\}^{SF}$ is the resulting signal we use as the input signal to the neural network, $\boldsymbol{x}$ is the original sequence, h is the channel fading factor that follows a Rayleigh distribution, and $\boldsymbol{n}$ is the Gaussian noise vector whose components verify: $n_i \sim \mathcal{N}\left(0,\, \sigma^2\right)$.

Before feeding the signal to the SNN input, all negative values are clipped to 0. Indeed, this is a constraint coming from the way signals are fed to the neural network: as we consider a rate-based SNN, the spike rate depends on the signal amplitude, so only positive values can be processed.

B. SNN model

At the receiver side, we use a Spiking Neural Network (SNN) to detect the sequences. The neurons are based on the bio-inspired leaky integrate-and-fire model. In this model, when there are no incoming spikes, each neuron has a membrane voltage that continuously decreases towards a constant value called the resting value. On the contrary, each time a spike is received by the neuron, its membrane voltage is increased by a constant value. The exact behavior of the voltage is defined by [11]:

$$\tau \frac{dV}{dt} = E_{rest} - g_i \cdot C - V \times (1 + g_i + g_e), \tag{2}$$

where E_{rest} is the resting membrane potential, C is a constant, and g_e and g_i are the conductances of excitatory and inhibitory synapses, and govern the impact level of an incoming spike. They are ruled by the equations below:

$$\tau_{g_e} \frac{dg_e}{dt} = -g_e, \tag{3}$$

$$\tau_{g_i} \frac{dg_i}{dt} = -g_i, \tag{4}$$

$$g_{e,n+1} = g_{e,n} + w_{ei}, \tag{5}$$

$$g_{i,n+1} = g_{i,n} + w_{ie}, \tag{6}$$

$$g_{e,n+1} = g_{e,n} + w_{input}, \tag{7}$$

where τ_{g_e} and τ_{g_i} are time constant for the natural decay of g_e and g_i. Besides, Eq.(5) (resp (6)) governs how g_e (resp g_i) is updated on the inhibitory (resp excitatory) neuron each time the only connected excitatory (resp a connected inhibitory) neuron spikes. w_{ei} is the weight of the synapse linking the excitatory to the inhibitory neuron, and w_{ie} is that of the one linking the inhibitory to the excitatory neuron. They are both fixed numbers that never change. Finally, g_e is also updated following (7) on excitatory neurons when it receives a spike from an input neuron, but this time w_{input} is the weight of the synapse from the input to the excitatory neuron, which evolves following the learning mechanism, as discussed later.

Our proposed SNN is made of two layers. The first one is the input layer, composed of SF neurons, where SF is the length of the input signal. The second one is the processing layer, and can be divided into two parts. One part is made of N excitatory neurons, and the other one is like a mirror of the first one, made of N inhibitory neurons. The goal of the inhibitory neurons is to give advantage to the first neuron to spike by temporarily neutralizing the others, thus developing competition among excitatory neurons, and helping them memorize different signals. N corresponds to the number of sequences that our neural network is able to recognize.

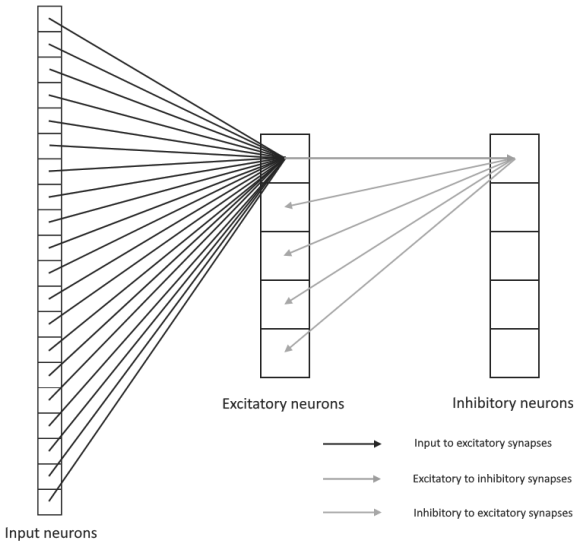

Fig. 1: SNN architecture. *Only the synapses that interact with the first excitatory neuron are shown*

The input neurons are spiking following a Poisson distribution, with a rate proportional to the signal fed into the network. Each of these input neurons is connected to all excitatory neurons, leading to $SF \times N$ connections. Each excitatory neuron is then connected to its mirrored inhibitory neuron. Each inhibitory neuron is connected back to each excitatory neuron, except its mirrored counterpart.

The implemented learning mechanism is STDP, using synaptic traces as in [11, 12]. Thus the following weight change applies whenever there is a presynaptic spike:

$$w_{input,n+1} = w_{input,n} - \eta_{post} \times a_{post}, \tag{8}$$

where η_{post} is a fixed coefficient that can change the rate at which w_{input} is updated, and a_{post} is the postsynaptic trace. When a postsynaptic spike occurs, the following weight change is made:

$$w_{input,n+1} = w_{input,n} + \eta_{pre} \times a_{pre}, \tag{9}$$

where η_{pre} has the same role as η_{post}, and a_{pre} is the presynaptic trace.

Another mechanism is introduced in order to ease competition among neurons, that is called homeostasis. This mechanism relies on changing the neuron's thresholds over time, following:

$$V_{threshold} = V_{threshold,base} + \theta, \tag{10}$$

$$\tau_\theta \frac{d\theta}{dt} = -\theta, \tag{11}$$

$$\theta_{n+1} = \theta_n + \Delta\theta, \tag{12}$$

where $V_{threshold,base}$ is the initial neuron threshold, at rest; θ is the value that increases whenever the neuron spikes, τ_θ is a time factor and $\Delta\theta$ is a constant value acting as a step. Each time the neuron spikes, the threshold effectively increases, making it harder to spike again. Homeostasis prevents the neurons from spiking too frequently, which might otherwise lead to one neuron hindering others from spiking.

III. Simulation results

In this paper, we focus on the impact of different factors on the SNN performances. These factors are h, the channel coefficient and σ, the Gaussian noise standard deviation.

Performance will be mainly measured in terms of the *accuracy* of the network. This is computed as the ratio between the number of correctly identified signals (i.e. the network outputs the class corresponding to the input signal), and the total number of signals we presented to the network for the simulation.

We consider, as a toy example, a network with $N = 5$ nodes. The sequences are made of $SF = 20$ chips. For each sequence, 4 chips are equal to 1, while the others are *null*. We set other parameters that will be fixed during all experiments: the resting membrane voltage $E_{rest} = -65$ mV, the threshold upon which a neuron spikes $V_{threshold,base} = -60$ mV, the reset value of the membrane voltage after a spike $E_{reset} = E_{rest}$, time constants $\tau_{g_e} = 1$ ms and $\tau_{g_i} = 2$ ms, synapse weights $w_{ei} = 15$ and $w_{ie} = 17$, the STDP constants $\eta_{post} = 10^{-4}$, $\eta_{pre} = 10^{-2}$, and finally the homeostasis increment $\Delta\theta = 5.10^{-2}$ mV, values adapted from [11] to the dimensions of our signals.

A simulation is divided into 2 phases, training and testing. First, the network is **trained** by feeding it with signals (not necessarily labeled). During this phase, the weights of the synapses linking the input neurons with the excitatory ones w_{input} are updated following the learning algorithm (STDP); as well as the homeostasis factor for each neuron. The last part of this phase consists of presenting labeled signals to the network and analyze its behavior for each excitatory neurons, as they act as output neurons. We can then associate each of them with a sequence.

The second phase is the **testing** phase. We freeze the weights and the homeostasis, then we present new input signals to the network and evaluate its response for each of them, based on the previous classification.

Unless specified differently, we trained the network on 100 signals (20 of each class), and then ran the testing phase on at least 5000 signals for each set of parameters.

In the following, we evaluate the impact of the channel impairments on the training and testing phase, first separately, then together.

A. Response to Gaussian noise

We first consider the Gaussian noise impact, by varying its standard deviation (s.d.).

We start by training the network with signals corresponding to different values of the s.d. σ_{train}. Then, for each trained configuration, we test the network performances when fed with various s.d. σ_{test}.

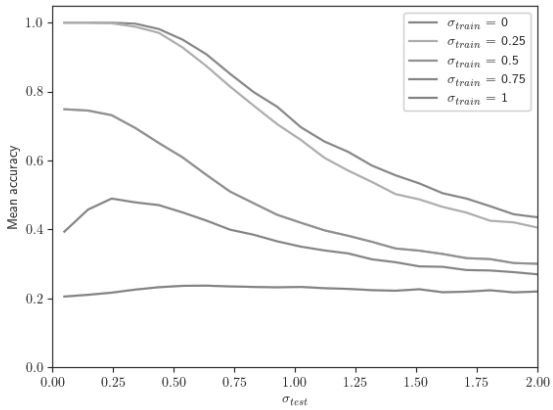

Fig. 2: Mean accuracy as a function of the standard deviation of the Gaussian noise in the testing phase, for different values in the training phase

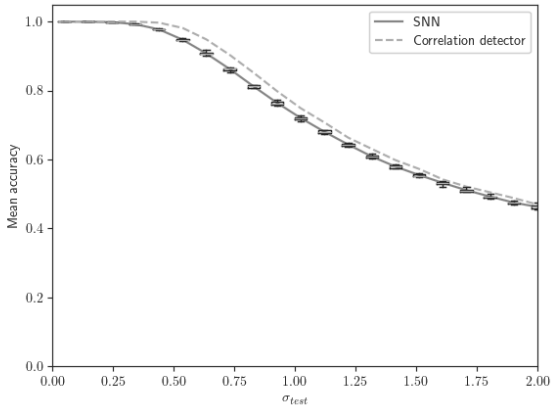

Fig. 3: SNN and correlation detector mean accuracy as a function of the standard deviation of the Gaussian noise

We can see on Figure 2 that the network has better performances when trained with a lower s.d. σ_{train}, and that the maximum is reached when training is done without any noise. Indeed, the SNN goal is to learn the sequences, and the noise only alters them. Consequently, from now on, we train the network with noiseless signals, as this offers the best performances. Besides, it also permits to design the SNN regardless of the expected noise level, which is more convenient in practice.

In addition, to evaluate the interest of the SNN, we compare its performances with that of a classical approach. For a single sequence emission, the correlation receiver is the optimal one. It detects the sequence based on the following rule:

$$\hat{i} = \underset{i\in\{1,2,\dots,N\}}{\operatorname{argmax}} \{\boldsymbol{y} \cdot \boldsymbol{x_i}\}, \tag{13}$$

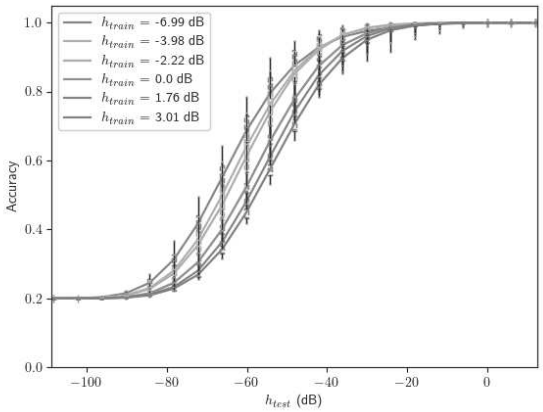

Fig. 4: Mean accuracy as a function of the Rayleigh coefficient for the testing phase, for different h_{train} values

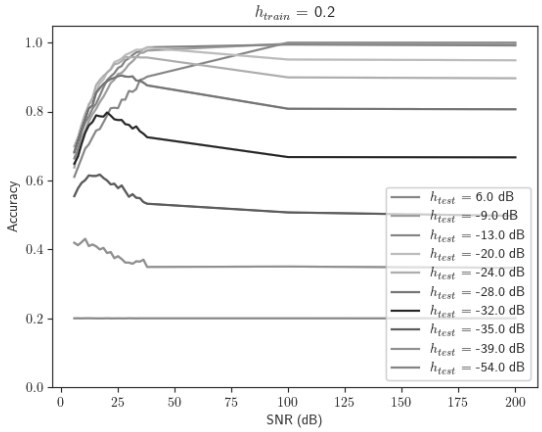

Fig. 5: Mean accuracy as a function of the SNR for a training channel coefficient of 0.2. *200 dB points should be at infinity.*

where $\hat{i}$ is the detected (output) class, $\boldsymbol{y}$ is the input signal and $\boldsymbol{x_i}$ is the original sequence of user #i.

The box plots on the SNN curve on Figure 3 represent the deviation between the different training cases (i.e. with different random initial synapse weights before the training phase). We can first notice that this randomness does not have a meaningful impact on the performances. Secondly, SNN performances are very similar to that of the classical correlation detector, albeit slightly reduced ($0-5\%$ loss). As expected, we can achieve maximum performance when the noise is low in the test phase, and the SNN provides the same performances as the correlation detector when the noise is sufficiently low. Thus, we can conclude that the SNN permits to detect the sequence almost as efficiently as the correlation receiver but with a reduced energy consumption. This validates its interest for WuR receivers.

B. Impact of channel response

We now take into account the channel coefficient h, which will have a strong impact on the signal scale. It follows a Rayleigh distribution of parameter: ν (*scale*). We present the results as a function of the *mean*: $\mu = \nu\sqrt{\frac{\pi}{2}}$.

We train the SNN with noiseless signals for different Rayleigh training *means*. We keep the same initial random weights to focus on the *mean* impact. We evaluate their performance for various testing *means*. Results are presented on Figure 4. The Rayleigh *mean* for the training phase h_{train} is indicated in the top-left corner of the figure, while the Rayleigh *mean* for the test phase h_{test} varies along the abscissa axis.

First, we can see that the bigger h_{test} is, the better the performances are. Indeed, when h_{test} is low, the signal is strongly attenuated and the difference between what is originally a 1 and what is a 0 is faded out. The network has difficulties trying to tell them apart.

Meanwhile, interestingly, we can see that training the network with a lower coefficient h_{train} leads to better performances, independently of the actual channel coefficient during the test phase. This shows that the network is able to accurately generalize to high-level signals, even when it has been trained on attenuated signals. Thus, h_{train} should be set to a low value to adapt to all channel conditions during the test phase.

C. Response to both

Finally, we consider simultaneously the noise and channel impact. As stated in **III-A**, we consider noiseless signals for the training phase, but noisy ones for the test phase. Meanwhile, the Rayleigh mean varies independently during the training phase and during the test phase.

We present the results for two cases, $h_{train} = 0.2$ (Figure 5), and $h_{train} = 2$ (Figure 6). To provide a fair comparison, we plot the network responses under the same signal *quality* condition, i.e. *Signal to Noise Ratio* (SNR):

$$SNR = 20 \cdot log\left(\frac{h_{mean}}{\sigma_n}\right), \tag{14}$$

Note that the points for SNR $= 200$ dB actually represent the noiseless case, i.e. SNR $= \infty$.

We can see that the shapes of the curves are very different. We actually identify two different behaviors.

For high values of h_{test} (up to -20 dB), as seen in **III-A**, performances increase continuously to reach the maximum, as the SNR increases.

However, for lower values of the test channel coefficient h_{test}, performances first increase to a certain point, and then decrease to reach a convergence point. This can be explained by the fact that at very low channel values, the network doesn't receive enough stimulus to react efficiently, but when the Gaussian noise's s.d. is increased, it can help trigger reactions on certain occasions. When the noise is too important, however, the performances collapse.

Comparing Figure 5 and Figure 6, we can see that the channel coefficient considered for the training part does not significantly impact these behaviors. However, we can verify

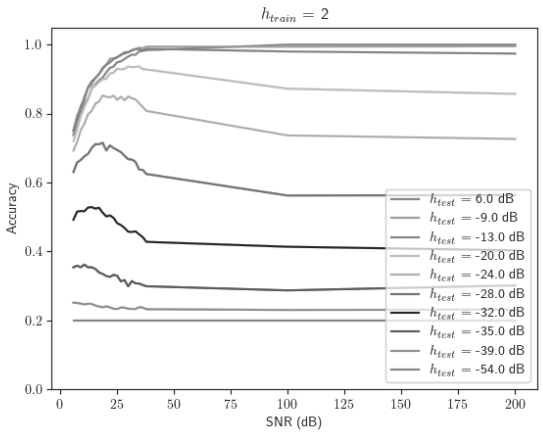

Fig. 6: Mean accuracy as a function of the SNR for a training channel coefficient of 2. *200 dB points at should be at infinity.*

that training with a lower channel coefficient h_{train} still offers better performances when the testing phase is done with a low channel coefficient h_{test} as observed in **III-B**. Moreover, the threshold upon which performance is able to converge to the maximum is lower when training with low channel coefficients, and appears to be of the order of magnitude of the latter.

D. Convergence time

Finally, we consider the impact of the channel coefficient value on the duration of the training phase.

In the previous simulations, during the training phase, we defined a fixed number of signals to present to the SNN. But, if we observe the weight values evolution during the training (Figure 7), we can observe that the convergence is faster when the channel coefficient is higher.

This means that the network learns more easily with non-attenuated/amplified signals. Besides, one can notice that for $h_{train} = 0.2$, the convergence is not finished at the end of the predefined number of presented signals, while for $h_{train} = 2$, the convergence has been reached quickly.

We thus introduce a mechanism that stops the training phase when we judge that the network has sufficiently converged to its final expected state (that is to say the values of input weights w_{input} are either near 1, or near 0). This mechanism stops the training phase when there is no weight in the range $[0.4; 0.98]$ anymore.

In addition, during the classification process at the end of the training phase, we check whether the network is actually able to classify N different sequences. If not, we mark the training as an error.

Figure 8 shows the average number of signals presented to the SNN. We can see that the convergence time of the network decreases when the channel coefficient increases. In addition, Figure 9 shows us that training errors also happen less frequently if the channel coefficient is not too small (lower than 0.2).

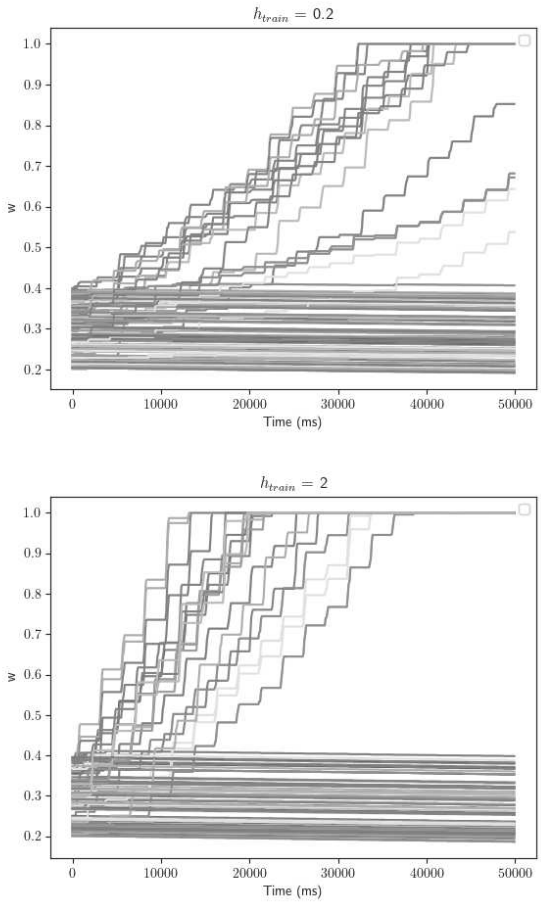

Fig. 7: Evolution of the weights of the input to excitatory synapses during the training phase

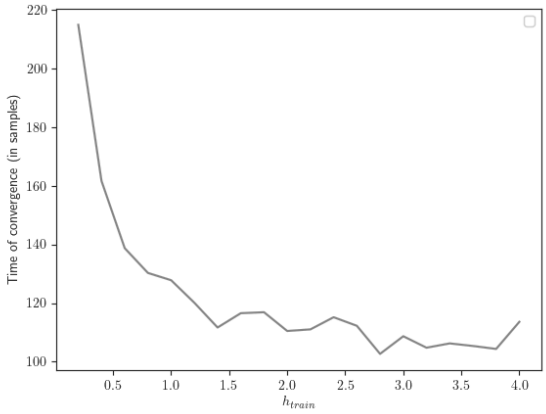

Fig. 8: Convergence time, in number of training signals, as a function of the training channel coefficient

We can thus deduce that a higher channel coefficient is more interesting for the training phase. However, as seen in **III-B**,

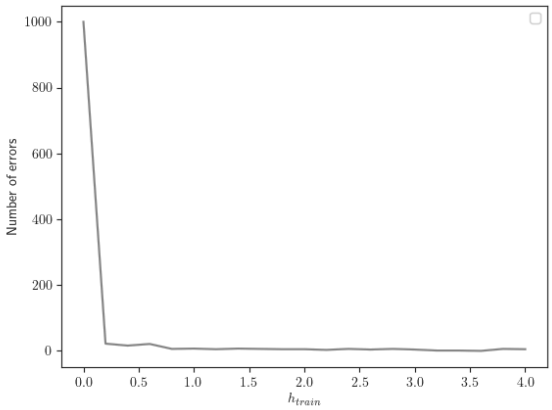

Fig. 9: Number of errors (with max 1000) as a function of the training channel coefficient

training with higher channel coefficients gives the network more difficulties generalizing when processing signals with lower channel coefficients. The channel gain for the training phase must thus be chosen carefully (around 0.2), as there is a tradeoff between the learning phase speed and the testing phase performances.

IV. Conclusion

In this paper, we studied the use of a SNN for detecting the sequences in an IoT system. The objective was to train appropriately the SNN so as to recognize the sequences despite them being modified by the channel. We first showed that training the SNN in a noiseless case permits to reach almost the same performance as classical receivers, even when the processed signal is noisy.

In addition, we focused on the channel coefficient impact, and showed that training the SNN with a smaller channel coefficient improves the performance, as, in this case, the SNN is able to generalize for higher channel coefficients (while training with high channel coefficients does not permit to generalize to lower values). Thus, for scaling issues, the training should be done with small channel coefficients.

However, when considering the training duration, we showed that the channel coefficient should be high, so as to increase the number of spikes within the SNN, and thus accelerate the STDP process. We gather from these analysis that a compromise has to be made on the value of the training channel coefficient, as a lower value offers better performances, but increases the convergence time.

Interestingly, this study thus provides a framework for training the SNN. It also shows that SNN is able to handle the wide range of incoming signals, and thus constitutes a promising approach for WuR in IoT systems. Indeed, the SNN is an energy efficient alternative to existing technologies, such as DeepRX, which consumes $2\ mJ$ for a single inference [13], whereas, based on current technologies [14, 15], our SNN consumes $98 \times 10^{-3}\ pJ$ for a single inference ($500\ ms$ long inferences with a passive power of $100\ pW$, and $4 \times 10^{-3}\ pJ$ per spike, with 12 spikes per inference on average).

Future work should consider other sequence families to co-design the sequence along with the SNN receiver, other channel models, and also real signal traces.

References

[1] Rishika Mehta, Jyoti Sahni, and Kavita Khanna. Internet of things: Vision, applications and challenges. *Procedia Computer Science*, 132:1263–1269, 2018.

[2] Patrick P Mercier, Benton H Calhoun, Po-Han Peter Wang, Anjana Dissanayake, Linsheng Zhang, Drew A Hall, and Steven M Bowers. Low-power rf wake-up receivers: Analysis, tradeoffs, and design. *IEEE Open Journal of the Solid-State Circuits Society*, 2022.

[3] Ilker Demirkol, Cem Ersoy, and Ertan Onur. Wake-up receivers for wireless sensor networks: benefits and challenges. *IEEE Wireless Communications*, 2009.

[4] Mazloum et Al. Performance analysis and energy optimization of wake-up receiver schemes for wireless low-power applications. *IEEE Transactions on Wireless Communications*, 2014.

[5] Halil Yetgin et Al. A survey of network lifetime maximization techniques in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 2017.

[6] Michael Hopkins et Al. Spiking neural networks for computer vision. *Interface Focus*, August 2018.

[7] Xu Zhang, Z Xu, C Henriquez, and S Ferrari. Spike-based indirect training of a spiking neural network-controlled virtual insect. 12 2013.

[8] Arindam Basu, Lei Deng, Charlotte Frenkel, and Xueyong Zhang. Spiking neural network integrated circuits: A review of trends and future directions. In *IEEE Custom Integrated Circuits Conference*, 2022.

[9] Guillaume et al Marthe. Wake-up radio receiver based on spiking neurons for detecting activation sequence. In *2023 IEEE Wireless Communications and Networking Conference (WCNC)*, 2023.

[10] Yiting Dong, Dongcheng Zhao, Yang Li, and Yi Zeng. An unsupervised stdp-based spiking neural network inspired by biologically plausible learning rules and connections. *Neural Networks*, 2023.

[11] Peter Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.*, August 2015.

[12] Introduction to brian part 2: Synapses. https://tinyurl.com/introBrianP2.

[13] Amine Lbath and Ibtissam Labriji. Energy efficiency in ai for 5g and beyond: A deeprx case study. 06 2024.

[14] Ilias et al. Sourikopoulos. A 4-fj/spike artificial neuron in 65 nm cmos technology. *Frontiers in Neuroscience*. ISSN 1662-453X. doi: 10.3389/fnins.2017.00123.

[15] F. Danneville et al. A sub-35 pw axon-hillock artificial neuron circuit. *Solid-State Electronics*, 153:88–92, 2019. ISSN 0038-1101. doi: https://doi.org/10.1016/j.sse.2019.01.002.