Performance study of a bidirectional vanilla spiking neural network for signal noise suppression

Dominik Čambál, Martin Pukanec, Monika Herchlová, Juraj Ďuďák
Slovak University of Technology
Faculty of Materials Science and Technology in Trnava
Jána Bottu 25, Trnava, Slovakia
{dominik.cambal, martin.pukanec, monika.herchlova, juraj.dudak}@stuba.sk

Matúš Nečas
University of Žilina
Research Centre
Univerzitná 8215/1, Žilina, Slovakia
{matus.necas}@feit.uniza.sk

Abstract. *The paper focuses on the design and performance evaluation of a basic "vanilla" spiking neural network (SNN) with a simulated bidirectional architecture, intended for signal noise suppression. The testing was conducted on a synthetic dataset composed of periodic signals with various types of additive noise. The proposed solution was implemented using Leaky Integrate-and-Fire (LIF) neurons within the SNNTorch framework. The procedure consisted of generating noise scenarios, implementing comparative classical filters and neural architectures: a recurrent neural network (RNN), a convolutional neural network (CNN) and a spiking neural network (SNN), designing the bidirectional model, and training it on a defined sample of signals. The paper also included the definition of evaluation metrics and timing parameters for comparing the performance of the individual approaches. The paper describes the entire development process, from architectural design to functional testing at the software simulation level.*

Keywords. Spiking neural network, Recurrent neural network, Convolutional neural network, Denoising

1 Introduction

Noise reduction in signals represents a fundamental challenge in the processing of time-dependent data. In engineering practice, various methods are employed to address this issue. Noise suppression is commonly applied in the medical and biomedical fields. For example, it is used to mitigate distortions caused by patient movement or possible electromagnetic interference from power lines. It also plays a critical role in processing weak brain signals or suppressing other physiological signals. In telecommunications, noise removal is essential to ensure reliable and high-quality communication. In radar systems, noise suppression is used for accurate object detection and improved target resolution. Within industrial automation, denoising sensor signals is necessary to maintain correct process control and regulation, as well as to detect faults and anomalies. Noise reduction is also applied in audio technology to enhance the quality of recordings and sound transmission. In the financial sector, denoising techniques are primarily used to filter market signals in order to uncover underlying trends and predict price movements of various assets.

Conventional noise suppression methods are traditionally classified (Martínez et al., 2022) based on their functional principles. The most widely used, thanks to their computational simplicity and sufficient accuracy, are the *linear filters*, which satisfy the superposition principle. Martínez et al., 2022 use the term *traditional filtering methods*, for a subset of linear filters, albeit the largest one. These methods are based on the assumption that noise occupies different, usually higher frequencies than the original (clean) signal; therefore, frequencies above a chosen cutoff need to be blocked by a low-pass filter (LPF). Besides these traditional methods, the superposition principle is also satisfied by other filters such as the Savitzky-Golay filter. *Adaptive filters* require a reference signal with which is the noisy signal compared. Such a reference signal is often difficult to obtain; in practice it is either provided by the manufacturer, or derived from the same noisy signal, temporarily shifted. The filtering is based on minimization of the error between the reference signal and the filtered signal using a specified statistical metric.

Artificial neural networks (ANNs) approximate the clean signal by being trained to minimize error either between noisy and clean signals (supervised learning),

or between noisy signals alone (unsupervised learning).

Other methods not covered by these categories include signal transforms (wavelet transform), statistical methods (Kalman filter), nonlinear and regularization-based methods (e.g., total variation (TV) denoising, non-local means), and non-neural-model-based approaches (e.g., autoregressive models). The four methods (Butterworth LPF, Savitzky-Golay filter, LMS, and RLS filters) that were chosen for the comparison are covered in more detail in the chapter 2.2.

Unlike the aforementioned methods, artificial neural networks adapt to model nonlinear relationships between noisy and clean signals without needing prior information about the signal shape or the noise model (Maas et al., 2012). If a clean or reference signal–or signals–are available, ANNs can be trained by minimizing the loss function (typically MSE or L1) between the noisy and clean versions (supervised training). This is usually achieved using ANNs designed for temporal data processing, thanks to their inherent statefulness–examples include RNNs, transformers, and SNNs. On the other hand, ANNs, and especially autoencoders, can be trained for signal denoising in an unsupervised manner.

Recurrent neural networks (RNNs) are a common approach, successfully used for noise suppression in audio processing (Maas et al., 2012), biosignal (ECG, EMG) analysis (Antczak, 2019) and others (e.g., industrial or scientific data) (Chen et al., 2020, Piccolomini et al., 2019). While the traditional unidirectional variant is mostly considered sufficient, it has several drawbacks in comparison with the so-called bi-directional RNN (BiRNN), particularly in the first few time steps of the denoised signal.

A BiRNN processes each input sequence twice, once from start to finish and then in reverse, so that each hidden state is influenced not only by its previous value, but also by its future value.

In this paper, we investigate whether the bi-directionality traditionally used in RNNs can also be applied to spiking neural networks (SNNs). Spiking neurons are also described by their internal state, and since they are biologically inspired, the state is known as the membrane potential. The literature on SNNs for signal noise suppression is limited, with most existing studies focusing on SNNs designed as autoencoders (Ren et al., 2025), hybridized with CNNs (Dorzhigulov and Saxena, 2024) or enhanced with mathematical algorithms such as active tuning (Ciurletti et al., 2021). In our work, we propose a vanilla SNN (using only LIF neurons) that simulates bidirectionality, which to our knowledge has been discussed only by (Xiao et al., 2023). Using a set of synthetic signals, we compare, using statistical metrics and processing times, the use of conventional noise suppression with RNNs (forward and bi-directional), CNN AE and SNNs (forward and bi-directional).

2 Methodology

This section describes the signal dataset, noise suppression methods, and metrics used for the performance analysis.

2.1 Signal dataset

For evaluation, we use a synthetic signal dataset of size $n = 1000$. As the study focuses on the proof-of-concept of a bi-directional SNN for noise suppression and its comparison to other methods, we found a single, controlled dataset sufficient. Although we acknowledge the possible limited generalizability, we believe that our approach minimizes confounding variables and allows for a clear comparison. The base signal of our dataset is periodic, composed of two sine waves (Eq. 1).

$$y(t) = A_1 \sin(2\pi f_1 t + \varphi_1) + A_2 \sin(2\pi f_2 t + \varphi_2) \tag{1}$$

with different amplitudes ($A_1 = 1$, $A_2 = 0.5$), frequencies ($f_1 = 30$ Hz, $f_2 = 70$ Hz), and phases ($\varphi_1 = 0$, $\varphi_2 = \pi/4$). The resulting signal is sampled at a frequency $F_s = 1000$ Hz for a duration of 2 s, yielding 2000 samples. The additive noise we use is Gaussian, although we also tested the performance with impulse and quantization noise, which were omitted for brevity.

Gaussian noise is based on the Gaussian (normal) distribution, whose probability density function is given by Eq. 2, where μ is the mean of the distribution and σ is the standard deviation.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

Gaussian noise is among the most widely used noise types for noise simulations, as it closely approximates real-world noise and has well-understood mathematical properties that make it controllable and predictable. In our project, it is generated using the `numpy` library method, `np.random.normal(mean, std_dev, size)`, with mean $\mu = 0$ and standard deviation $\sigma = 0.5$. The value of the standard deviation was chosen arbitrarily to introduce a significant level of noise to the synthetic signal to create a challenging, yet consistent, denoising problem for all the methods being compared.

2.2 Conventional noise suppression methods

In this section, four conventional noise suppression methods (Butterworth LPF, Savitzky-Golay filter, LMS and RLS filters) are described. These four methods were picked as they represent different paradigms of conventional filtering approaches and are commonly used in practice, as they provide a fair balance between computational complexity and accuracy.

2.2.1 Butterworth LPF

The Butterworth low-pass filter (LPF) was designed to have the frequency response $H(j\omega)$ in the passband as flat as possible (with minimal ripple), which distinguishes it from, for example, Chebyshev filters, which ripple either in the passband or in the stopband (Zhang and Jiang, 2021). The squared frequency response of the filter is given by the Eq. 3.

$$|H(j\omega)|^2 = \frac{1}{1+\left(\frac{\omega}{\omega_C}\right)^{2N}} \tag{3}$$

Based on this formula, two parameters for the design of the filter are crucial. The first one is the order of the filter N: the higher the order, the steeper the frequency response in the transition band. The second one is the cutoff angular frequency ω_C. In our project, after some experimentation, we chose the order $N = 10$ as a good trade-off between computational complexity and accuracy. The cutoff frequency was chosen as $f_c = 100$ Hz, based on the representation of the signal in the frequency domain using Fourier transform. The `butter(order, Wn)` function in Python's `scipy.signal` library calculates internally the cutoff angular frequency using the normalized frequency W_n–the function's second parameter. Its value is usually computed using the formula $W_n = f_c/f_N$, where f_N is the Nyquist frequency ($f_N = F_s/2$, in our case $1000\,\text{Hz}/2 = 500\,\text{Hz}$).

2.2.2 Savitzky-Golay filter

The Savitzky-Golay filter is based on a least-squares polynomial approximation of a window of odd length $N = 2m + 1$ (Liu et al., 2016; Samann and Schanze, 2019). For each window, a polynomial of the order k with coefficients $b = [b_0, \ldots, b_k]$ is fitted using least squares. Accordingly, the `savgol_filter` function in Python's `scipy.signal` library has, in addition to the noisy signal, two other parameters: the window length N and the polynomial order k. In our case, we experimentally found that, for a given polynomial order k, the best window length is approximately given by the Eq. 4.

$$N \approx 7 \cdot \left\lfloor \frac{k}{2} \right\rfloor + 7. \tag{4}$$

If N is even, 1 is added to the result. We use $k = 7$ and $N = 31$.

2.2.3 LMS and RLS filters

LMS (least mean squares) and RLS (recursive least squares) filters both require a reference (or desired) signal for their computations. LMS filter minimizes the difference–the mean squared error–between the noisy and the desired signals using stochastic gradient descent (Hinton, 2003). The filter simplifies the algorithm further by calculating only an estimate of the gradient vector. In general, the k-th sample of a noisy signal x is denoised using a formula given by Eq. 5.

$$\hat{s}_k = y_k - \sum_{i=0}^{N-1} w_k(i) \cdot x_{k-i} \tag{5}$$

In the formula, y is the reference signal, w represents the weights and N is the filter length. In the case of LMS, the weight update rule for the next sample is given by Eq. 6.

$$w_{k+1}(i) = w_k(i) + 2\mu e_k x_{k-i} \tag{6}$$

for $i = 0$ to $N - 1$, where μ is the learning rate, x is the input (noisy) signal vector of size N and e is the error between the desired and actual output.

RLS, in comparison, minimizes the exponentially weighted sum of squared errors; all formulas describing the filter are provided by Tosi et al., 2013.

2.3 Artificial neural networks

In this section, we discuss different artificial neural network (ANN) architectures commonly used for time series analysis.

2.3.1 Recurrent neural networks

Recurrent neural networks (RNNs) were originally developed to address the limitations of feedforward neural networks (FNNs). FNNs were found to be exceptional at static data processing, e.g. measuring datasets or images, but they perform poorly on time-dependent data (time series, signals, audio, video), which must be input one step at a time rather than all at once.

Units in RNNs are enhanced with inner hidden states, neuronal outputs are wired recurrently, i.e. in the next time step, they return to the same neuron or other neurons in the same layer. Hidden states subsequently act act as a kind of memory for each unit, updated based on the previous state and the input. Thanks to this, RNNs are able to capture temporal information and patterns within sequences.

There are three main architectures of RNNs: the vanilla RNN and the RNNs with GRU (gated recurrent unit) or LSTM (long short term memory) units. Vanilla RNN is the default architecture used by the Pytorch `nn.RNN` module, with hyperbolic tangent as the default activation function. In our project, we found the vanilla RNN to provide a good balance of computational complexity, accuracy, and training time. Both versions we used are many-to-many, since both the output and the input of the same size is expected (a denoised signal).

At first, we designed a unidirectional (forward) RNN, where the sequence is processed from start to finish and the hidden state of each neuron is updated based on its direct previous value. On the other hand, in a bi-directional RNN, each time sequence is processed twice, at first from start to finish and then from finish

to start. The single-hidden-layer architecture of a bi-directional RNN is schematically depicted in Fig. 1.

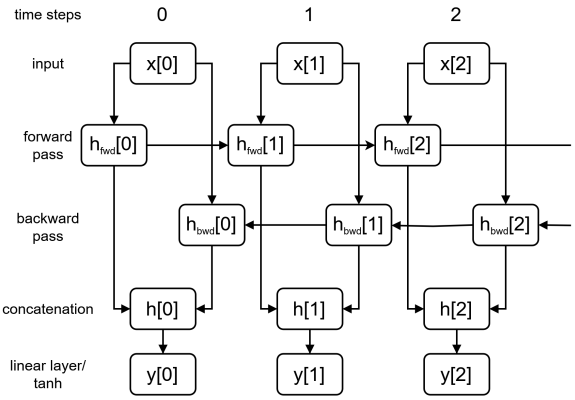

Figure 1. Bi-directional RNN architecture

The forward pass of a bi-directional RNN is described by the Eq. 7, while the backward pass is described by the Eq. 8.

$$h_{fwd}[t] = \tanh(\overrightarrow{W}x[t] + \overrightarrow{U}h_{fwd}[t-1]) \quad (7)$$

$$h_{bwd}[t] = \tanh(\overleftarrow{W}x[t] + \overleftarrow{U}h_{bwd}[t+1]) \quad (8)$$

Denoting n as the input sequence length and d as the hidden size, in eqs. 7 and 8, t is time ($t = 0..n-1$ in the forward pass, $t = n-1..0$ in the backward pass), $x[t]$ is the scalar input, $W \in \mathbb{R}^{d\times 1}$ are the input-to-hidden weight matrices, $U \in \mathbb{R}^{d\times d}$ are the hidden-to-hidden weight matrices and $h[t] \in \mathbb{R}^{d\times 1}$ the hidden state vectors. The weights are shared across time steps.

Both passes yield forward and backward hidden state vectors for each layer for each time step; the hidden state vectors of both passes of the last hidden layer are concatenated as $h[t] \in \mathbb{R}^{2d\times 1}$ (Eq. 9) and serve as inputs to the linear layer (Eq. 10, where $W \in \mathbb{R}^{1\times 2d}$ is the linear layer weight matrix).

$$h[t] = \begin{bmatrix} h_{fwd}[t] \\ h_{bwd}[t] \end{bmatrix} \quad (9)$$

$$y[t] = 2 \cdot \tanh(Wh[t]) \quad (10)$$

The input layer and the output layer each have size 1, and the activation function of the output layer is twice the hyperbolic tangent (eq. 10) as it better captures information on the interval $[-2, 2]$. The interval is slightly larger than the interval of possible amplitudes of our base signal.

2.3.2 Convolutional neural networks

Convolutional neural networks (CNNs), feedforward in nature and therefore widely used for static data processing, can also be applied to time sequences. They learn features in data using a filter, and while in static data (especially 2D data such as images) the filter "moves" across space, in time series (1D data) the filter "moves" across time. Convolutional filters are also translation-invariant, which means that if a temporal pattern is learned in one part of the signal, it can be recognized elsewhere as well. On the other hand, they are usually well-suited for learning short-term features only, and for long-term dependencies are often combined with other neural network architectures (RNN, SNN) or other signal processing techniques (wavelet transforms).

One CNN architecture for denoising is an autoencoder (AE), composed of two parts: an encoder and a decoder. The data in the encoder is compressed, until it reaches a specific layer called the bottleneck (or latent space). From there on, the compressed data is reconstructed. The layers in the decoder symmetrically mirror the layers of the encoder. The number of channels in the encoder increases in each layer to capture finer details, while in the decoder it decreases. Conversely, the data size decreases (the number of time steps in time series) in the encoder by a factor of two and symmetrically increases by the same factor in the decoder. This architecture is self-supervised, meaning that it doesn't need to have the original underlying base signal, but it learns patterns that are consistent throughout the entire dataset.

The CNN AE in our project (architecture shown in Fig. 2) consists of two layers in the encoder and the decoder, with neurons in each layer having ReLU as their activation function. The number of time steps decreases before increasing by a factor of two.

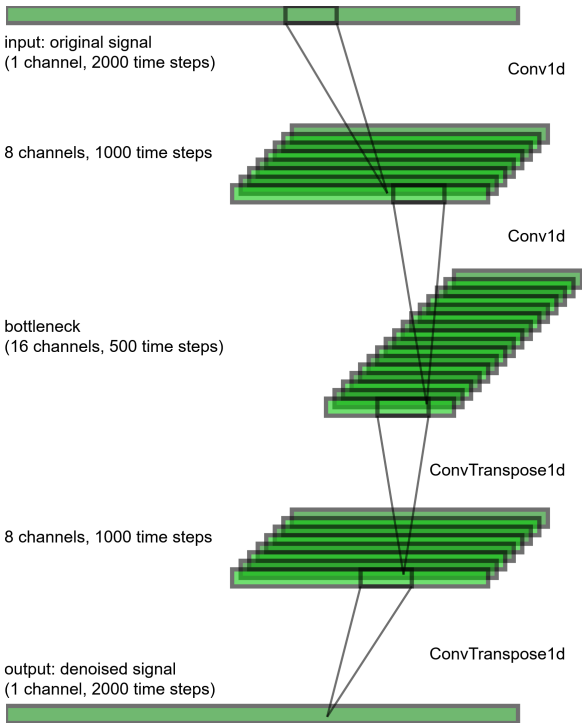

Figure 2. CNN AE architecture

2.3.3 Spiking neural networks

Spiking neural networks (SNNs) (Eshraghian et al., 2023) are biologically inspired, with neuron activation functions that are essentially unit steps: at each time step, a neuron's output is either 0 or 1. This output depends on the neuron's internal state, called the membrane potential, which produces an output called a spike (a 1 value) when a certain threshold θ is reached and then resets (either to zero or decreases by a set value).

SNNs are typically designed using so-called Leaky Integrate-and-Fire (LIF) neurons, which offer a trade-off between computational complexity and biological implausibility. They are described by the formula given by Eq. 11.

$$U[t] = \underbrace{\beta U[t-1]}_{decay} + \underbrace{WX[t]}_{input} - \underbrace{S_{out}[t-1]\theta}_{reset} \quad (11)$$

In the formula, t is time, U is the membrane potential, W is a weight matrix, β is the decay rate, θ is the threshold and S_{out} is the output spike (the activation function) (Eq. 12),

$$S_{out}[t] = \begin{cases} 1 & U[t] \geq \theta \\ 0 & otherwise \end{cases} \quad (12)$$

Spikes then travel across synapses to other neurons, where they accumulate and increase the membrane potential. Unlike units in RNNs, without any external stimuli (spikes), the internal state U of LIF neurons decreases (decays) by each time step. Any information passed through the SNN is therefore not encoded in the value of the output (since it is binary), but typically in the timing of the spikes (latency code) or the spike rate (rate coding).

One of the main problems when training SNNs is the so-called dead neuron problem: since backpropagation relies on stochastic gradient descent, by the chain rule, the Eq. 13 follows.

$$\frac{dL}{dW} = \frac{dL}{dS}\frac{dS}{dU}\frac{dU}{dI}\frac{dI}{dW} \quad (13)$$

In Eq. 13, L is the loss function, W is the weight matrix, S is the activation function and I represents the inputs. Since S is essentially a unit step, its weak derivative $\frac{dS}{dU}$ is the Dirac delta function, which takes on only two values: ∞ when the threshold is reached and 0 otherwise. Consequently, $\frac{dL}{dW}$ will have the same value, which prevents the weights from being updated. In practice, one of the most common ways to resolve this is by using a surrogate function during the backward pass, usually a smoothed version of the unit step (e.g., sigmoid or arctangent).

Nonetheless, the provided architecture results in various advantages of SNNs: they're asynchronous and highly parallelizable, which stems from the fact that most of the time most of the neurons are at rest. With the exception of membrane potentials, most of the information that travels in SNNs is not only non-floating, but also binary; and can be stored in vectors or simply as lists of spike times.

The driving principle behind SNN research is energy reduction: ANNs are known for their enormous energy consumption. In general, SNNs were shown to reduce energy use; the review of various architectures and their effect on energy consumption is provided by (Malcolm and Casco-Rodriguez, 2023). For example, the SNN Brain-Machine Interface, developed by (Liao et al., 2022), achieved reductions in comparison to aconventional ANN in total operations and memory accesses by well over 90%. To achieve this, a special neuromorphic hardware, which can support asynchronicity and parallelization, is needed. Various chips (Intel Loihi) and institutional supercomputers (University of Manchester's SpiNNaker) were built, although there were successful implementations of SNNs on FPGAs as well (Padovano et al., 2024).

That said, it is possible to simulate SNNs on traditional computer architectures using various libraries. In our project, we use SNNTorch, based on PyTorch, that we use in our RNN and CNN modules.

As with RNNs, we used both a forward model, but also a model that simulates bi-directionality. The overall architecture of the bi-directional variant is schematically depicted in the Figure 3.

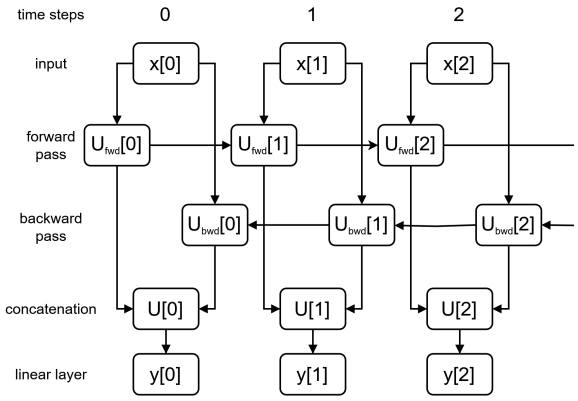

Figure 3. Bi-directional SNN architecture

The architecture of the bi-directional SNN consists of two branches (one for the forward pass and one for the backward pass), with single-neuron input and output layers and multiple hidden layers. Mathematically, and for demonstration purposes with only one hidden layer, in the forward pass for each time step t, the formula given by Eq. 14 is used for each neuron, and for the backward pass, the formula given by Eq. 15 is used.

$$\overrightarrow{U}[t] = \beta\overrightarrow{U}[t-1] + \overrightarrow{W}X[t] - \overrightarrow{S}_{out}[t-1]\theta \quad (14)$$

$$\overleftarrow{U}[t] = \beta\overleftarrow{U}[t+1] + \overleftarrow{W}X[t] - \overleftarrow{S}_{out}[t+1]\theta \quad (15)$$

Eqs. 14 and 15 are based on the Eq. 11. The spike output formula (Eq. 12) remains unchanged. Denoting d as the hidden size, for neurons in the hidden

layer, $X[t]$ is a scalar input, $\overrightarrow{W}[t], \overleftarrow{W}[t] \in \mathbb{R}^{d\times 1}$ are the weight matrices, $\overrightarrow{U}[t], \overleftarrow{U}[t] \in \mathbb{R}^{d\times 1}$ are the membrane potential matrices, and $\overrightarrow{S}_{out}[t], \overleftarrow{S}_{out}[t] \in \{0,1\}^{d\times 1}$ are the spike matrices. For the single output neuron, the weight matrices are of size $1 \times d$ and all other quantities are scalar. As with the RNN (chapter 2.3.1), the resulting membrane potentials are then concatenated and passed through a linear layer.

The main difference between the SNN and the RNN architecture (Figure 1) is in the way the information is propagated between the neurons in the hidden layers.

In our project, bi-directionality was implemented using a class `SNNBranch` with two hidden layers of LIF neurons (100 and 75 neurons, respectively).

We found bi-directional SNN to be a promising architecture for signal denoising, based on initial experiments with shorter, simpler sine wave signals with an amplitude of $A = 1$ and a frequency of $f = 2\pi/60$ Hz, sampled at 500 time steps. Both the forward and the bi-directional SNN were trained on 80% of the 5000 samples. While the forward SNN yielded a non-optimal denoised output, the bi-directional SNN significantly enhanced the denoising quality (Fig. 4). The average MSE between the noisy and BiSNN-denoised signals was 0.2506.

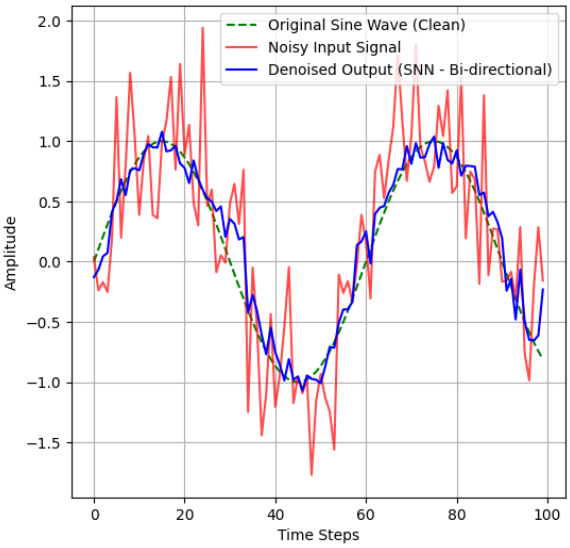

Figure 4. Simple single-sine signal denoising with BiSNN

2.4 Metrics

In our project, we use the mean squared error (MSE), which is based on the squared differences between the clean signal and the denoised signal; the lower the MSE, the better the denoising performance. The MSE values shown in the table are averages computed across all noisy-denoised signal pairs in the testing dataset. Other metrics, such as the root mean squared error (RMSE) and the signal-to-noise ratio (SNR), were considered but were omitted from presented results mostly due to brevity, as their values were roughly concordant with MSE.

We also measured computational time. For the conventional denoising methods, we recorded the time taken to process the entire dataset. For the ANN-based methods, we recorded the time it took for the neural network to be trained. These times were chosen because they represent the most time-consuming operations in the overall computation.

3 Results

The methods discussed in Chapters 2.2 and 2.3 and the metrics described in Chapter 2.4 were implemented in the Python programming language. Existing libraries such as Scikit-learn (for MSE/RMSE), SciPy (for Butterworth LPF and Savitzky-Golay filter), PyTorch (for neural networks), and SNNTorch (for SNNs) were utilized. Custom implementations were developed for LMS and RLS filters, as well as SNR/PSNR metrics. Neural networks were trained on Google Colab using v2-8 TPU processing units.

The results obtained from measurements are provided in Tables 1 and 2.

Table 1. Performance of conventional denoising methods

Denoising Method	MSE	Denoising time [s]
Butterworth LPF	0.0481	0.15
Savitzky-Golay filter	0.0439	0.48
LMS filter	0.0309	12.95
RLS filter	0.0427	138.28

Table 2. Performance of ANNs

Denoising Method	MSE	Training time [s]
Forward RNN	0.0220	146.25
Bi-directional RNN	0.0117	453.41
CNN autoencoder	0.0254	3.07
Forward SNN	0.1672	223.15
Bi-directional SNN (175 neurons)	0.1030	479.72
Bi-directional SNN (650 neurons)	0.0777	3589.26

The conventional methods performed similarly in denoising the datasets, with the LMS filter achieving the lowest MSE, and the RLS, Savitzky-Golay, and Butterworth LP filters performing marginally worse. The main disadvantage of both adaptive filters was the time required to denoise the entire dataset, with

the RLS filter in particular taking more than 2 minutes to denoise the entire dataset. The good performance of the Savitzky-Golay filter can be attributed to the polynomial nature of the sine wave, which, however, may hinder in denoising of non-stationary signals. Both adaptive filters also initialize their weights at zero; therefore, they take some time to learn, which is why the amplitude for the first couple of hundredths of a second is at 0.

Similar problems can be observed in the forward architectures of the RNN and the SNN. The inaccuracies in the first time steps can be attributed to the initialization of weights; in other words, the neural network does not yet have any knowledge about the data. Bi-directionality addresses this problem by having an access to the future context in the early time steps. The bi-directional RNN outperformed all of the conventional methods used, as did the marginally less accurate CNN autoencoder, which also has the advantage of no need for the clean version of the signal. Particularly significant was the superior performance of RNNs over other methods that require a reference signal.

The improved performance of bi-directional SNN compared to the forward SNN described in the chapter 2.3.3 was also observed when training the SNN on the two-sine dataset (Table 2), where the bi-directional SNN outperformed its forward counterpart by almost 40%. Nevertheless, both architectures were outperformed by other methods in terms of accuracy. However, the training time of the bi-directional SNN was comparable with that of bi-directional RNN despite the significant increase in the number of neurons (the RNN having 30 units in the hidden layer, while the SNN having 175 neurons in both hidden layers combined).

Further refinement of the bi-directional SNN included the addition of another hidden layer and an increase in neuron count per layer (250, 200, and 200 neurons, respectively). This upgraded architecture resulted in an MSE of 0.0777, representing nearly a 25% improvement (a sample is shown in the Figure 5).

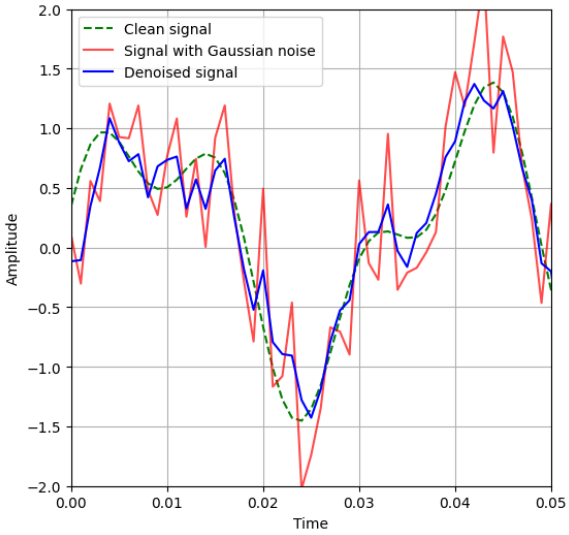

Figure 5. Signal sample with upgraded Bi-SNN

4 Discussion

The initial experiments with SNNs on simpler signals provided valuable insights into their behavior and the benefits of bi-directionality. The quantitative improvement observed with the bi-directional SNN on the actual dataset, despite its lower accuracy compared to other methods, suggests that SNNs hold promise, particularly considering their comparable training times to bi-directional RNNs even with a significantly higher number of neurons. This efficiency, coupled with the potential for more biologically plausible computation and potential energy consumption reduction given a proper architecture, warrants further investigation.

Although the architectural upgrade to the bi-directional SNN yielded a substantial improvement in MSE, its overall accuracy still lags behind other methods. Future research avenues could explore more complex SNN architectures, such as incorporating learnable decay parameters as demonstrated by Hao et al., 2024, or investigating hybrid models combining SNNs with RNNs, CNNs, or autoencoders to leverage the strengths of different neural network paradigms for enhanced denoising performance.

5 Conclusion

The paper presents the design and implementation of a “vanilla” spiking neural network featuring two hidden layers composed of Leaky Integrate-and-Fire neurons, utilizing a simulated bidirectional architecture in the SNNTorch framework. The proposed solution was evaluated on a synthetic dataset of periodic signals corrupted by noise and compared against traditional and neural-based denoising approaches. The development process included the design of the dataset, implementation of baseline filters and networks, construction of the bidirectional SNN processing mechanism, and the selection of metrics for accuracy and computational cost assessment. The results demonstrate that the proposed bidirectional SNN architecture offers clear improvements over its unidirectional counterpart. Although it lags behind BiRNNs and CNN-based autoencoders in certain metrics, its key advantage generally lies in significantly lower energy consumption given a specific hardware, although the training time advantage is also noteworthy. Future research may explore hybrid architectures, e.g., SNN-RNN combinations, or the application of learned parameters for membrane potential decomposition.

Acknowledgments

This work was supported by KEGA through the Teaching and Development of methodology for the use of microcontrollers in automation using practical examples and laboratory exercises for engineering students, under Grant 024STU-4/2024.

References

Antczak, K. (2019). Deep recurrent neural networks for ecg signal denoising. https://arxiv.org/abs/1807.11551

Chen, H., Guo, R., Liu, J., Wang, Y., & Lin, R. (2020). Magnetotelluric data denoising with recurrent neural network. In *Seg 2019 workshop: Mathematical geophysics: Traditional vs learning, beijing, china, 5–7 november 2019* (pp. 116–118). https://doi.org/10.1190/iwmg2019_28.1

Ciurletti, M., Traub, M., Karlbauer, M., Butz, M. V., & Otte, S. (2021). Signal Denoising with Recurrent Spiking Neural Networks and Active Tuning. In I. Farkaš, P. Masulli, S. Otte, & S. Wermter (Eds.), *Artificial Neural Networks and Machine Learning – ICANN 2021* (pp. 220–232). Springer International Publishing.

Dorzhigulov, A., & Saxena, V. (2024). Compact Convolutional SNN Architecture for the Neuromorphic Speech Denoising. *2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 1191–1195. https://doi.org/10.1109/MWSCAS60917.2024.10658937

Eshraghian, J. K., Ward, M., Neftci, E. O., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D. S., & Lu, W. D. (2023). Training Spiking Neural Networks Using Lessons From Deep Learning. *Proceedings of the IEEE*, *111*(9), 1016–1054. https://doi.org/10.1109/JPROC.2023.3308088

Hao, X., Ma, C., Yang, Q., Tan, K. C., & Wu, J. (2024). When Audio Denoising Meets Spiking Neural Network. *2024 IEEE Conference on Artificial Intelligence (CAI)*, 1524–1527. https://doi.org/10.1109/CAI59869.2024.00275

Hinton, O. (2003). *EEE305 - Digital Signal Processing*. University of Newcastle upon Tyne. https://www.staff.ncl.ac.uk/oliver.hinton/eee305/ (Accessed: 26.06.2025).

Liao, J., Widmer, L., Wang, X., Di Mauro, A., Nason-Tomaszewski, S. R., Chestek, C. A., Benini, L., & Jang, T. (2022). An Energy-Efficient Spiking Neural Network for Finger Velocity Decoding for Implantable Brain-Machine Interface. *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 134–137. https://doi.org/10.1109/AICAS54282.2022.9869846

Liu, Y., Dang, B., Li, Y., Lin, H., & Ma, H. (2016). Applications of Savitzky-Golay filter for seismic random noise reduction. *Acta Geophysica*, *64*, 101–124. https://doi.org/https://doi.org/10.1515/acgeo-2015-0062

Maas, A. L., Le, Q. V., O'Neil, T. M., Vinyals, O., Nguyen, P., & Ng, A. Y. (2012). Recurrent Neural Networks for Noise Reduction in Robust ASR. *Interspeech 2012*, 22–25. https://doi.org/10.21437/Interspeech.2012-6

Malcolm, K., & Casco-Rodriguez, J. (2023). A comprehensive review of spiking neural networks: Interpretation, optimization, efficiency, and best practices. https://arxiv.org/abs/2303.10780

Martínez, M., Garcia-March, M. A., Enrique, C., & Fernández de Córdoba, P. (2022). *Algorithms for Noise Reduction in Signals: Theory and practical examples based on statistical and convolutional analysis*. Bristol: Institute of Physics Publishing. https://doi.org/10.1088/978-0-7503-3591-1

Padovano, D., Carpegna, A., Savino, A., & Di Carlo, S. (2024). SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA. *Electronics*, *13*(9). https://doi.org/10.3390/electronics13091744

Piccolomini, E. L., Gandolfi, S., Poluzzi, L., Tavasci, L., Cascarano, P., & Pascucci, A. (2019). Recurrent Neural Networks Applied to GNSS Time Series for Denoising and Prediction. In J. Gamper, S. Pinchinat, & G. Sciavicco (Eds.), *26th international symposium on temporal representation and reasoning (time 2019)* (10:1–10:12, Vol. 147). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.TIME.2019.10

Ren, P., Wang, Y., Wang, Z., Peng, D., Liu, C., & Han, T. (2025). Denoising autoencoder multilayer perceptron spiking neural network for isonicotinic acid yield prediction on real industrial dataset. *Advanced Engineering Informatics*, *65*, 103273.

Samann, F., & Schanze, T. (2019). An efficient ECG denoising method using Discrete Wavelet with Savitzky-Golay filter. *Current Directions in Biomedical Engineering*, *5*(1), 385–387. https://doi.org/doi:10.1515/cdbme-2019-0097

Tosi, D., Poeggel, S., Leen, G., & Lewis, E. (2013). Adaptive filter-based interrogation of high-sensitivity fiber optic Fabry-Perot interferometry sensors. *Sensors and Actuators A: Physical*, *206*. https://doi.org/10.1016/j.sna.2013.12.010

Xiao, R., Wan, Y., Yang, B., Zhang, H., Tang, H., Wong, D. F., & Chen, B. (2023). Towards Energy-Preserving Natural Language Understanding With Spiking Neural Networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *31*, 439–447. https://doi.org/10.1109/TASLP.2022.3221011

Zhang, X., & Jiang, S. (2021). Application of Fourier Transform and Butterworth Filter in Signal Denoising. *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, 1277–1281. https://doi.org/10.1109/ICSP51882.2021.9408933