Spiking Neural Networks on EEG data processing

André Furlan
Rua Cristóvão Colombo 2265, Jd Nazareth, 15054-000, São José do Rio Preto - SP, Brazil.
Instituto de Biociências, Letras e Ciências Exatas, Unesp - Univ Estadual Paulista (São Paulo State University)
São José do Rio Preto, Brazil
Email: ensismoebius@gmail.com

***Abstract*—**

Among the methods of Brain-Computer Interface (BCI), Electroencephalogram (EEG) stands out as the most cost-effective and simple system to implement. However, it does have some quirks, such as high sensitivity to electromagnetic interference and difficulty in capturing the signal due to suboptimal scalp placement of the electrodes.

To address these issues, numerous Neural Networks (NNs) have been developed to handle such data. Despite their effectiveness, conventional NNs still require a significant amount of computational power for operation and training. This work proposes an alternative approach to this problem by utilizing Spiking Neural Networks (SNNs), which hypothetically consume less power. The aim is to assess the viability of SNNs in this context.

The results indicate a very close similarity in performance, demonstrating that SNNs are indeed a promising option for future research endeavors.

The sources used in this work are available in https://github.com/ensismoebius/deepLearnning)

***Index Terms*—EEG, Spiking Neural Networks, SNN, Signal Processing**

I. Introduction

Standard Neural Networks (NNs), as defined here —*multi-layered, fully connected, with or without recurrent or convolutional layers*— require that all neurons are activated in both the forward and backward passes. This implies that every unit (neuron) in the network must process some data, leading to power consumption [1].

In an era where responsible use of power resources is mandatory, coupled with the growing interest in Brain-Computer Interface (BCI) devices, a new problem arises: How to create NN models that are power-efficient, enabling their deployment even on portable devices?

The objective of this paper is to test if the current state of Spiking Neural Networks (SNNs) can be an effective method in the processing of Electroencephalogram (EEG) data.

In this work a Spiking Neural Network was created and compared with a Convolutional Neural Network (CNN) both in accuracy and loss metrics.

CONCLUSIONS

The remaining of this document is organized as follows: Section II presents a literature review, Section III details the tests and the results and lastly, Section IV is dedicated to the conclusions that are followed by the references.

II. Literature Review

A. Brain-Computer Interface and EEG

Among the methods of Brain-Computer Interface (BCI), Electroencephalogram (EEG) stands out as the most cost-effective and simple system to implement. However, it does have some quirks, such as high sensitivity to electromagnetic interference and difficulty in capturing the signal due to suboptimal scalp placement of the electrodes. So one of the fundamental aspect that any EEG processing system must have is the tolerance to noise [2].

The *wet* electrodes are placed using conductive gel and are less prone to movement artifacts which are electromagnetic interferences caused by some motion like blinking. *Dry* electrodes does not need the gel but is more sensible to artifacts.

EEG records the electrical activities of the brain, typically placing along the scalp surface electrodes. These electrical activities result from ionic current flows induced by the synchronized synaptic activation of the brain's neurons. They manifest as rhythmic voltage fluctuations ranging from 5 to 100μV in amplitude and between 0.5 and 40 Hz in frequency[2]. The operational frequencies bands in the brain are following [3]:

- **Delta (1–4Hz)**: The slowest and usually the highest amplitude waveform. The Delta band is observed in babies and during deep sleep in adults.
- **Theta (4–8Hz)**: Observed in children, drowsy adults, and during memory recall. Theta wave amplitude is typically less than 100μV.
- **Alpha (8–12Hz)**: Usually the dominant frequency band, appearing during relaxed awareness or when eyes are closed. Focused attention or relaxation with open eyes reduces the amplitude of the Alpha band. These waves are normally less than 50 μV.
- **Beta (12–25Hz)**: Associated with thinking, active concentration, and focused attention. Beta wave amplitude is normally less than 30 μV.
- **Gamma (over 25Hz)**: Observed during multiple sensory processing. Gamma patterns have the lowest amplitude.

According to [2], for most of the tasks brain do, there are regions associated with it as seen in Table I.

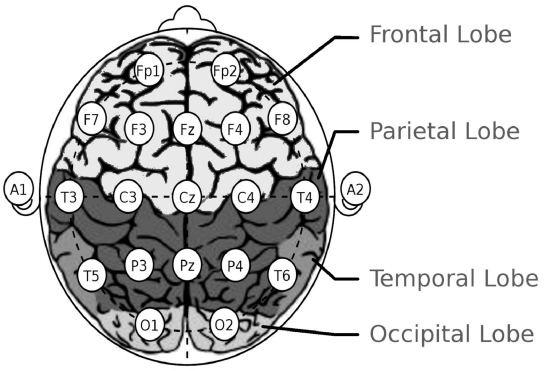

Fig. 1: Electrodes placement according to 10-20 standard [4] and brain's lobes. Odd numbers are assigned to the electrodes on the left hemisphere, and even numbers are assigned to the electrodes on the right. Source [2]

B. Spiking Neural Networks

Neural Networks (NNs), as defined here as *a multilayered, fully connected, with or without recurrent or convolutional layers network*, require that all neurons are activated in both the forward and backward passes. This implies that every unit in the network must process some data, leading to power consumption [1].

The sensory system of biological neurological systems converts external data, such as light, odors, touch, flavors, and others, into **spikes**. A spike is a voltage that convey information [5]. These ones are then transmitted along the neuronal chain to be processed, generating a response to the environment.

The biological neuron only spikes when a certain level of excitatory signals get accumulated above some threshold in its soma staying inactive when there is no signal, therefore, this type of cell is very efficient in terms of energy consumption and processing.

In order to get the aforementioned advantages the Spiking Neural Networks (SNNs) instead of employing continuous activation values, like NNs, SNNs utilize **spikes** at the input, hidden and outputs layers. SNNs can have continuous inputs as well and keep its properties.

An SNN **is not** a one-to-one simulation of neurons. Instead, it approximates certain computational capabilities of specific biological properties. Some studies like [6] created models way closer to natural neurons exploring the nonlinearity of dendrites and another neuron features yielding remarkable results in the classification.

As can be seen in Figure 2 SNNs neurons, given the correct parameters, are very noise tolerant because it acts as a **lowpass filter**. Generating spikes even when a considerably level of interference is present. This units are very time-oriented too, being great when it comes to process streams of data [1].

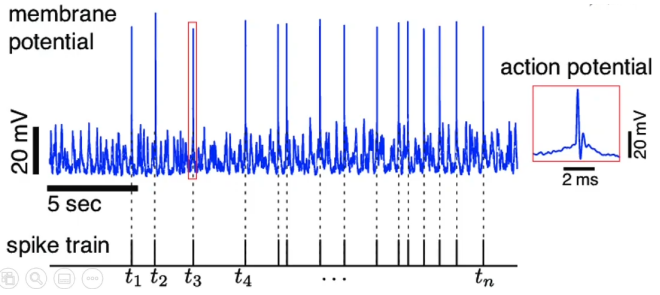

Fig. 2: Spikes from a noisy signal. Source [7]

C. Spiking Neuron

Although the focus of this work is the *Leaky Integrate and Fire Neurons* (LIF) because is simpler, more efficient and currently generalize better for most of the problems [7], there are more biological accurate model like **Hodgkin-Huxley neuron** [8] and ones like [6] that created models closer to natural neurons exploring the nonlinearity of dendrites and other neuron features.

1) Leaky Integrate and Fire Neuron: The Leaky Integrate and Fire Neuron (LIF) is one of the simplest neuron models in SNNs, still, it can applied successfully to most of the problems in with SNNs can be used.

LIF, like a NN neuron takes the sum of weighted inputs but, rather than pass it directly to its activation function, some *leakage* is applied, which decreases in some degree the sum.

LIF behave much like Resistor-Capacitor circuits as can be seen in Figure 3. Here R is resistance to the leakage, I_{in} the input current, C the capacitance, U_{mem} means the accumulated action potential and v is a switch that lets the capacitor discharge (i.e. emit a spike) when some potential threshold is reached.

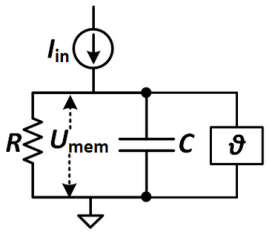

Fig. 3: RC model: Source: [1]

Unlike the Hodgkin-Huxley neuron, spikes are represented as **sparsely** distributed **ones** in a train of **zeros**, as illustrated in Figure 4 and 5. This approach simplify the models and reduces the computational power and needed storage to run an SNN.

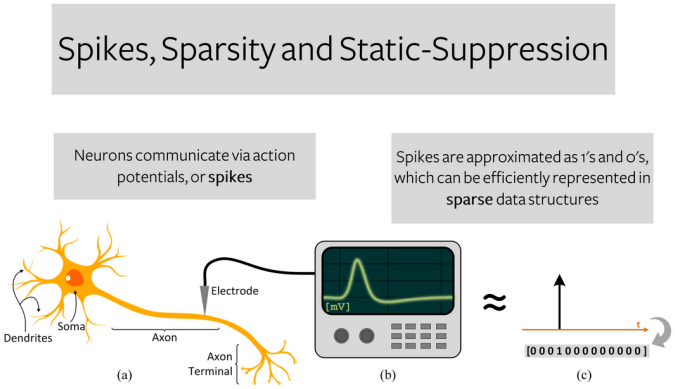

Fig. 4: Sparsity on Spike Neuron Networks. Source: [1]

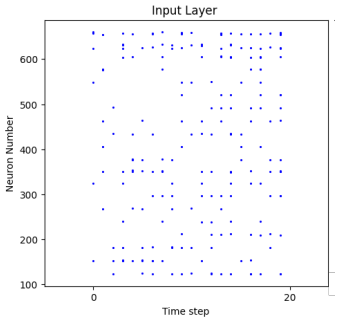

Fig. 5: Sparse activity of a SNN: The horizontal axis represents the time step of some data being processed the vertical are the SNN neuron number. Note that, most of the time, very few neurons gets activated. Source: The author.

As result of the mentioned above, in SNN information is coded in format of *timing* and/or *rate* of spikes giving consequently great capabilities of processing streams of data but limiting the processing of static data.

LIF model is governed by the Equations bellow [1].

Considering that Q is a measurement of electrical charge and $V_{mem}(t)$ is the potential difference at the membrane in a certain time t than the neuron capacitance C is given by the Equation 1.

$$C = \frac{Q}{V_{mem}(t)} \tag{1}$$

Than the neuron charge can be expressed as Equation 2.

$$Q = C.V_{mem}(t) \tag{2}$$

To know how these charge changes according to the time (aka current) we can derivate Q as in Equation 3. This expression express the current in the capacitive part of the neuron I_C

$$I_C = \frac{dQ}{dt} = C.\frac{dV_{mem}(t)}{dt} \tag{3}$$

To calculate the total current passing by the resistive part of the circuit we may use the Ohm's law:

$$V_{mem}(t) = R.I_R \implies I_R = \frac{V_{mem}(t)}{R} \tag{4}$$

Than considering that the total current do not change, as seen in Figure 6, we have the total input current I_{in} of the neuron as in Equation 5.

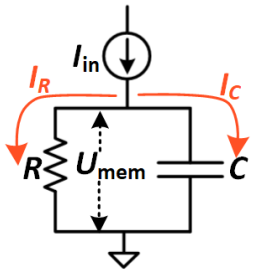

Fig. 6: RC model for currents: $I_{in} = I_R + I_C$

$$I_{in}(t) = I_R + I_C \implies I_{in}(t) = \frac{V_{mem}(t)}{R} + C.\frac{dV_{mem}(t)}{dt} \tag{5}$$

Therefore, to describe the passive membrane we got a linear Equation 6.

$$\begin{gathered} I_{in}(t) = \frac{V_{mem}(t)}{R} + C.\frac{dV_{mem}(t)}{dt} \implies \\ I_{in}(t) - \frac{V_{mem}(t)}{R} = C.\frac{dV_{mem}(t)}{dt} \implies \\ \boxed{R.I_{in}(t) - V_{mem}(t) = R.C.\frac{dV_{mem}(t)}{dt}} \end{gathered} \tag{6}$$

Than, if we consider $\tau = R.C$ as the **membrane time constant** we get voltages on both sides of Equation 7 which **describes the RC circuit**.

$$\begin{gathered} R.I_{in}(t) - V_{mem}(t) = R.C.\frac{dV_{mem}(t)}{dt} \implies \\ R.I_{in}(t) - V_{mem}(t) = \tau.\frac{dV_{mem}(t)}{dt} \implies \\ \boxed{\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t)} \end{gathered} \tag{7}$$

From that, and setting $I_{in} = 0$ (i.e. no input) and considering $\tau = R.C$ is a constant and an starting voltage $V_{mem}(0)$ the neuron's voltage behavior can be modeled as an exponential curve as can be seen in Equation 8.

$$\begin{gathered} \tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) \implies \\ \tau.\frac{dV_{mem}(t)}{dt} = -V_{mem}(t) = \\ e^{\ln(V_{mem}(t))} = e^{-\frac{t}{\tau}} = \\ \boxed{V_{mem}(t) = V_{mem}(0).e^{-\frac{t}{\tau}}} \end{gathered} \tag{8}$$

Then one can say that: In the absence of an input I_{in}, the membrane potential decays exponentially as illustrated in Figure 7 and implemented in Listing 1.

```
def lif(V_mem, dt=1, I_in=0, R=5, C=1):
  tau = R*C
  V_mem = V_mem + (dt/tau)*(-V_mem + I_in*R)
  return V_mem
```

Listing 1: Python implementation of the action potential decaying of a LIF: $I_{in} = 0$

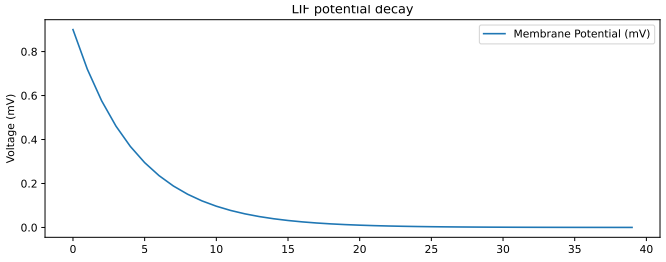

Fig. 7: Membrane potential decaying. Source: The author

With the results from Equation 7 it is possible to calculate the action potential increasing as seen in Equation 9.

$$
\begin{aligned}
&\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) = \\
&\frac{dV_{mem}(t)}{dt} + \frac{V_{mem}(t)}{\tau} = \frac{R.I_{in}(t)}{\tau} \implies \\
&\text{Integrating factor: } e^{\int \frac{1}{\tau} dt} = e^{\frac{1}{\tau}.t} \implies \\
&(e^{\frac{1}{\tau}.t}.V_{mem}(t))' = \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} = \\
&\int (e^{\frac{1}{\tau}.t}.V_{mem}(t))' = \int \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} dt = \\
&e^{\frac{1}{\tau}.t}.V_{mem}(t) = \int \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} dt \therefore \\
&\text{Considering: } V_{mem}(t=0) = 0 \implies \\
&\boxed{V_{mem}(t) = I_{in}(t).R(1 - e^{\frac{1}{\tau}})}
\end{aligned}
\tag{9}
$$

Note that when action potentials increases there is still an exponential behavior as seen in Figure 8 and implemented in Listing 2.

```

def lif(V_mem, dt=1, I_in=1, R=5, C=1):
  tau = R*C
  V_mem = V_mem + (dt/tau)*(-V_mem + I_in*R)
  return V_mem
```

Listing 2: Python implementation of the action potential decreasing of a LIF: $I_{in} = 1$

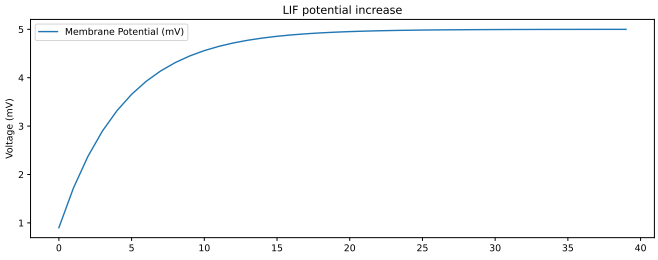

Fig. 8: Membrane potential increasing. Source: The author

Then taking into account some **threshold** which indicates a reset into the neuron potential and two type of resets (to zero and threshold subtraction) finally is possible to model the full LIF behavior as depicted in Figure 9 and implemented in Listing 3:

```
def lif(V_mem, dt=1, I_in=1, R=5, C=1, V_thresh = 2,
        reset_zero = True):
  tau = R*C
  V_mem = V_mem + (dt/tau)*(-V_mem + I_in*R)
  if V_mem > V_thresh:
    if reset_zero:
      V_mem = 0
    else:
      V_mem = V_mem - V_thresh
  return V_mem
```

Listing 3: Python implementation of the action potential full simulation of a LIF: $I_{in} = 1$, $V_{thresh} = 2$ is threshold

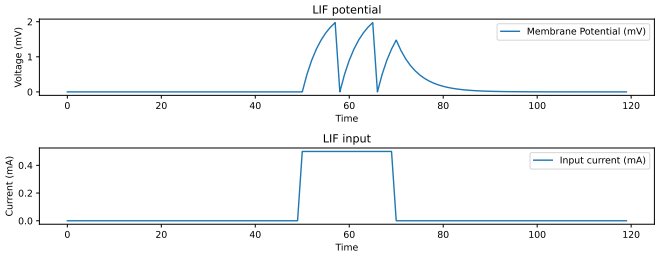

Fig. 9: Plot of the full simulated LIF: A 0.5 mA was provided from time 51 to 70.

D. Training

How do SNNs get trained? Well, this is still an open question. An SNN neuron has an activation-function behavior that is more relatable to a **step function**. Therefore, in principle, we can't use gradient descent-based solutions because this kind of function **is not** differentiable [5].

But there are some insights out there that may shed some light on this subject: While some *in vivo/in vitro* observations show that brains, in general, learn by strengthening/weakening and adding/removing synapses or even by creating new neurons or other cumbersome methods like RNA packets, there are some more acceptable ones like the ones in the list bellow [5]:

- **Spike Timing-Dependent Plasticity (STDP)**: If a pre-synaptic neuron fires **before** the post-synaptic one, there

is a strengthening in connection, but if the post-synaptic neuron fires before, then there is a weakening.
- **Surrogate Gradient Descent**: Approximates the step function by using another mathematical function, which is differentiable (like a sigmoid), in order to train the network. These approximations are used only in the backward pass, while keeping the step function in the forward pass [5].
- **Evolving Algorithms**: Use the selection of the fittest throughout many generations of networks.
- **Reservoir/Dynamic Computing**: **Echo state networks** or **Liquid state machines** respectively.

The one used in the SNN made in this work were **Surrogate Gradient Descent**.

III. Experiments

1) Database: The used **database** [9] can be found at https://sinc.unl.edu.ar/grants/brain-computer-interfaces and consists in a set of EEG data recorded with 6 electrodes during a imagined speech of the words in spanish: "A", "E", "I", "O", "U", "Arriba", "Abajo", "Adelante", "Atrás", "Derecha", "Izquierda".

Each channel captured 4096 samples and has the following ids: 'F3','F4', 'C3', 'C4', 'P3', 'P4' according to 10-20 system [10].

2) Neural networks: Two neural networks were tested in this work:

1) **SNN**: full connected spiking neurons
2) **CNN**: convolutional layer with full connected sigmoid neurons, which were used as a reference.

A. Full connected spiking neurons SNN

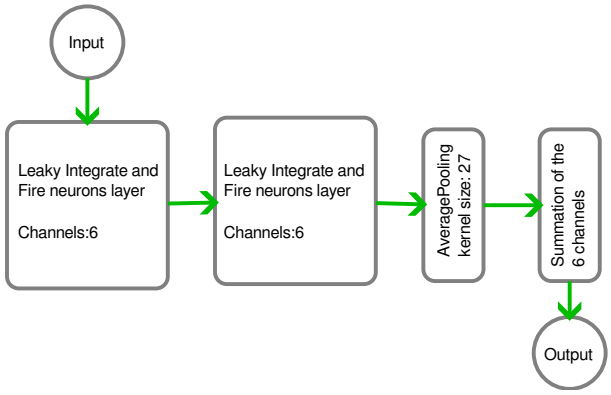

Fig. 10: Spiking neural network architecture

The SNN were trained according to the following parameters:

- **random seed**: 42
- **learning rate**: 2.10^{-4}
- **test batch size**: 40%
- **training batch size**: 60%
- **number of epochs**: 3000
- **loss function**: Cross-entropy
- **optimizer**: Adam ($0.00001 \leq \beta \leq 0.999$)
- **insist**: 4 (amount of times the same data is presented to the network)
- **neuron decay rate**: 0.9 (the rate LIF neuron loses its voltage)

```
self.model = nn.Sequential(
  nn.Linear(inputLength, 4096),
  snn.Leaky(
    beta=neuronDecayRate, spike_grad=spike_grad
  ),
  nn.Linear(4096, 27),
  snn.Leaky(
    beta=neuronDecayRate, spike_grad=spike_grad
  ),
  nn.AvgPool1d(27, stride=1)
)
```

Listing 4: SNN architecture

B. Convolutional layer with full connected sigmoid neurons CNN

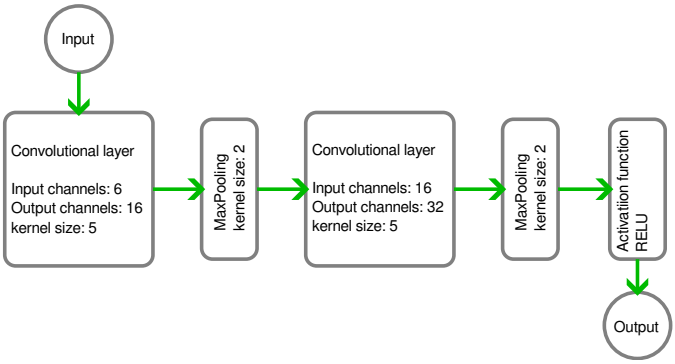

Fig. 11: Convolutional neural network architecture

The CNN were trained according to the following parameters:

- **random seed**: 42
- **learning rate**: 2.10^{-4}
- **test batch size**: 40%
- **training batch size**: 60%
- **number of epochs**: 3000
- **loss function**: Cross-entropy
- **optimizer**: Adam ($0.00001 \leq \beta \leq 0.999$)

```
self.model = nn.Sequential(
  nn.Conv1d(in_channels=6, out_channels=16,
    kernel_size=5
  ),
  nn.MaxPool1d(kernel_size=2),
  nn.Conv1d(in_channels=16, out_channels=32,
    kernel_size=5
  ),
  nn.MaxPool1d(kernel_size=2),
  nn.Flatten(),
  nn.Linear(feature_size, 128),
  nn.ReLU(),
  nn.Linear(128, 27)
)
```

Listing 5: CNN architecture

C. Results

1) Results for CNN: The CNN exhibit some unstable behavior and had a poor convergence as seen in Figure 12. Its stats are the following:

- Maximum Accuracy: 4.82%
- Minimum Loss: 0.9468
- Average Accuracy: 1.23%
- Average Loss: 3.328

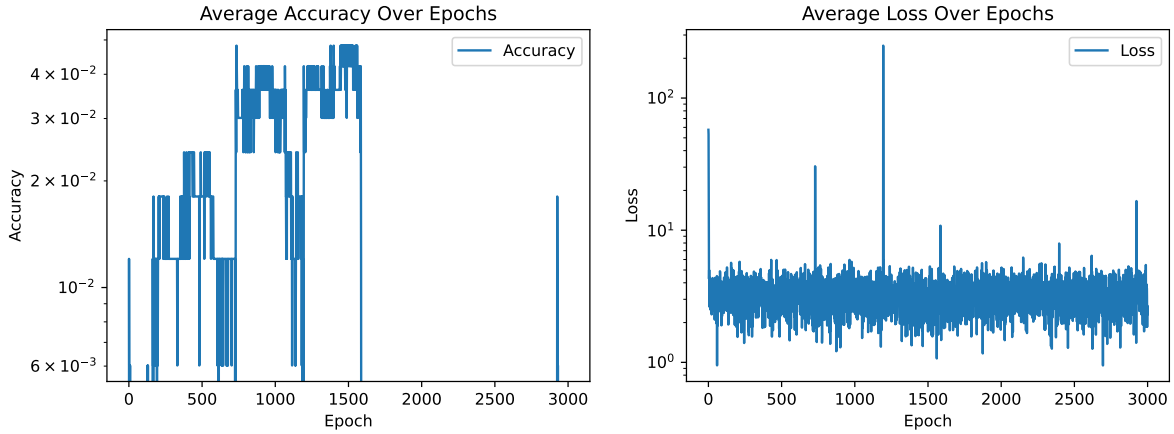

Fig. 12: CNN stats over 3000 epochs

2) Results for SNN: The SNN exhibit a more stable behavior and had a better convergence then CNN as can be seen in Figure 13. Its stats are the following:

- Maximum Accuracy: 100.00%
- Minimum Loss: 0.9278
- Average Accuracy: 18.25%
- Average Loss: 3.296

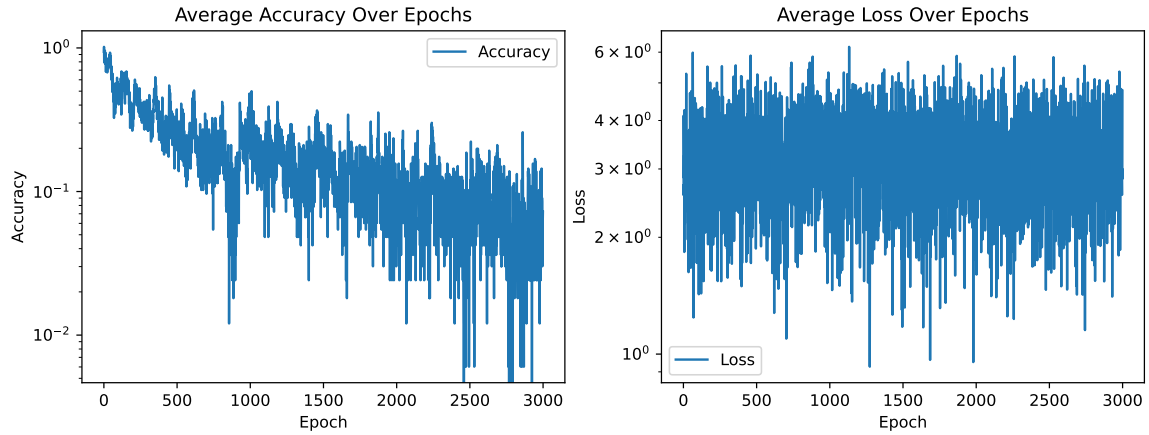

Fig. 13: SNN stats over 3000 epochs

IV. Conclusions

As seen in the experiments SNN are a plausible alternative to a more energy heavy networks. But some questions still stand:

The Surrogate Gradient Descent is an slow method of training, its is borrowed from the standard neural networks and adapted to SNNs. Will there be a training method that fits better for SNNs?

Another question is how energy efficient the SNNs really are in **non-neuromorphic hardware**?

Will neuromorphic hardware be easily available for this type of network?

References

[1] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, W. D. Lu, Training spiking neural networks using lessons from deep learning, Proceedings of the IEEE 111 (9) (2023) 1016–1054. doi:10.1109/JPROC.2023.3308088.

[2] A. Jalaly Bidgoly, H. Jalaly Bidgoly, Z. Arezoumand, A survey on methods and challenges in eeg based authentication, Computers and Security 93 (2020) 101788. doi:https://doi.org/10.1016/j.cose.2020.101788.
URL https://www.sciencedirect.com/science/article/pii/S0167404820300730

[3] S. Sanei, J. A. Chambers, EEG signal processing and machine learning, John Wiley & Sons, 2021.

[4] E. Vicente, Sistema 10-20 para localizar alvos terapêuticos em emt (2023).
URL https://www.kandel.com.br/post/como-localizar-os-pontos-de-estimulacao-para-estimulacao-magnetica-transcraniana

[5] N. K. Kasabov, Time-space, spiking neural networks and brain-inspired artificial intelligence, Springer, 2019.

[6] I. S. Jones, K. P. Kording, Can single neurons solve mnist? the computational power of biological dendritic trees (2020). arXiv:2009.01269.
URL https://arxiv.org/abs/2009.01269

[7] D. Goodman, T. Fiers, R. Gao, M. Ghosh, N. Perez, Spiking Neural Network Models in Neuroscience - Cosyne Tutorial 2022 (Sep. 2022). doi:10.5281/zenodo.7044500.
URL https://doi.org/10.5281/zenodo.7044500

[8] W. Gerstner, W. Kistler, R. Naud, L. Paninski, Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition, Cambridge University Press, 2014.
URL https://neuronaldynamics.epfl.ch/online/Ch2.S2.html

[9] G. Pressel-Coreto, H. L. Rufiner, I. E. Gareis, Diseño y elaboración de una base de datos pública de registros electroencefalográficos orientados a la clasificación de habla imaginada. (undergraduate project), Master's thesis, Universidad Nacional de Entre Ríos - Facultad de Ingeniería (2016).
URL http://sinc.unl.edu.ar/sinc-publications/2016/PRG16

[10] H. O. L. G H Klem, H. H. J. et al., The ten-twenty electrode system of the international federation. the international federation of clinical neurophysiology., Electroencephalography and clinical neurophysiology. Supplement 52.

V. Appendix

A. 10-20 system and the brain's areas

TABLE I: Brain tasks and its corresponding regions. See Figure 1 for more information.

Region	Channels	Tasks
Frontal lobe	Fp1, Fp2, Fpz, Pz, F3, F7, F4, F8	Memory, concentration, emotions.
Parietal lobe	P3, P4, Pz	Problem Solving, attention, grammar, sense of touch.
Temporal lobe	T3, T5, T4, T6	Memory, recognition of faces, hearing, word and social clues.
Occipital lobe	O1, O2, Oz	Reading, vision.
Cerebellum		Motor control, balance.
Sensorimotor Cortex	C3, C4, Cz	Attention, mental processing, fine motor control, sensory integration.

B. Another interpretation of LIF

Starting with the Equation 7 and using Forward Euler Method to solve the LIF model:

$$\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) \tag{10}$$

Solving the derivative in Equation 11 results in the membrane potential for any time $t + \Delta t$ in future:

$$\begin{aligned} \tau.\frac{V_{mem}(t+\Delta t) - V_{mem}(t)}{\Delta t} &= R.I_{in}(t) - V_{mem}(t) = \\ V_{mem}(t+\Delta t) - V_{mem}(t) &= \frac{\Delta t}{\tau}.(R.I_{in}(t) - V_{mem}(t)) = \\ \boxed{V_{mem}(t+\Delta t) &= V_{mem}(t) + \frac{\Delta t}{\tau}.(R.I_{in}(t) - V_{mem}(t))} \end{aligned} \tag{11}$$

C. Repositories and files

All source codes are available at:
https://github.com/ensismoebius/deepLearnning

The simulations and neural networks are available at:
https://github.com/ensismoebius/deepLearnning/tree/main/simulations/article/neuralNetworks

If you are interested in Jupyter notebooks visit:
https://github.com/ensismoebius/deepLearnning/tree/main/simulations/article/jupyterNotebooks

There is a nice simulator with a graphical user interface at:
https://github.com/ensismoebius/deepLearnning/tree/main/simulations/presentation

Browsing the location bellow you will have access to all latex codes used to make the documentation:
https://github.com/ensismoebius/deepLearnning/tree/main/documentation

Then if you want to see the log of the neural networks trainning access:
https://github.com/ensismoebius/deepLearnning/tree/main/documentation/article/tables