Available online at www.sciencedirect.com

ScienceDirect

Elsevier

ELS - (2023) 1–12

Spiking Neural Networks on EEG data processing

André Furlan[a,b]

[a]Instituto de Biociências, Letras e Ciências Exatas, Unesp - Univ Estadual Paulista (São Paulo State University), Rua Cristóvão Colombo 2265, Jd Nazareth, 15054-000, São José do Rio Preto - SP, Brazil.
[b]Centro Paula Souza, São Paulo, SP, Brazil.

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)

Keywords: EEG, Spiking Neural Networks, SNN, Signal Processing

1. Introduction

Standard Neural Networks (NNs), as defined here —*multilayered, 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 [9].

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 2 presents a literature review, Section 3 details the tests and the results and lastly, Section 4 is dedicated to the conclusions that are followed by the references.

2. Literature Review

2.1. 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 [1].

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 $100\mu V$ in amplitude and between 0.5 and 40 Hz in frequency[1]. The operational frequencies bands in the brain are following [2]:

- **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 $100\mu V$.
- **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 [1], for most of the tasks brain do, there are regions associated with it as seen in Table 1.

Table 1: 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.

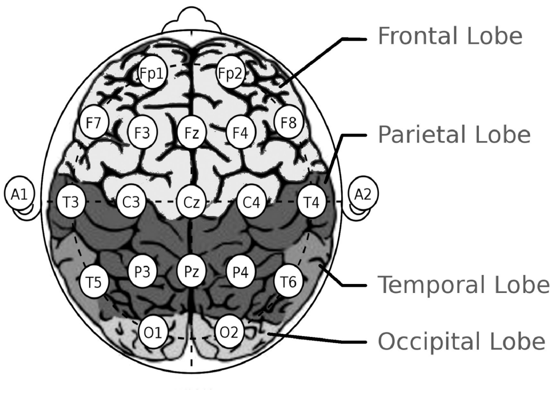

Figure 1: Electrodes placement according to 10-20 standard [3] and brain's lobes. Odd numbers are assigned to the electrodes on the left hemisphere, and even numbers are assigned to the electrodes on the right. Source [1]

2.2. *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 [9].

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 [4]. 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 [5] 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 [9].

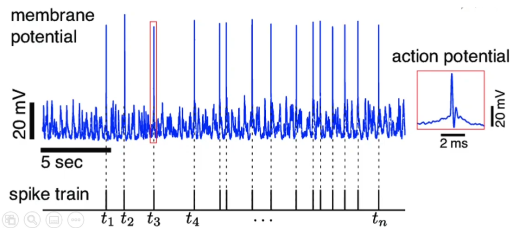

Figure 2: Spikes from a noisy signal. Source [6]

2.3. *Spiking Neuron*

Although this work focus in just *Leaky Integrate and Fire Neurons* (LIF) because is simpler, more efficient and currently generalize better for most of the problems [6], there is one interesting enough, because it paves the way to another ones, that worth to mention:

2.3.1. *Hodgkin-Huxley neuron*

The Hodgkin-Huxley neuron [7] was modeled conducting experiments on a giant axon of a squid and identified three distinct types of ion currents: Na^+ (sodium), K^+ (Potassium), and a leak current primarily composed of Cl^- (chloride) ions.

Like can be seen in Figure 3 this neuron is modeled as conjunction of a resistor R, potentiometers R_{Na}, R_K and a capacitor C along with batteries (ion concentrators) E_L, E_{Na}, E_K in which an current I is injected. The potentiometers indicate the channels of Na^+ and K^+ which can ben opened or closed, the batteries represents Nernst potential [8] for each channel.

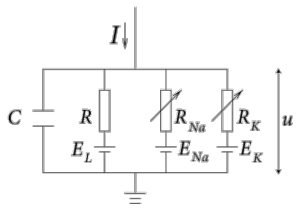

Figure 3: Schematic diagram for the Hodgkin-Huxley model. Source: [7]

Given that:

- C_m is the membrane capacitance that represents the ability of the neuronal membrane to store charge.
- g_{Na} is the sodium conductance that describes its fast opening permeability.

- g_K is the potassium conductance that describes its slow opening permeability.
- g_{leak} is the background conductance that describes ion movements not related to Na^+ or K^+ channels.
- V is the membrane potential.
- I is the injected current.
- m, h, and n are dimensionless gating variables representing the probability that the sodium and potassium channels are open.
- α and β are voltage-dependent rate constants.

Than the neuron's behavior is ruled by the Equations 1, 2, 3, 4 and Listings 1, 2, 3.

The equation 1 models how rate of change $\frac{dV}{dt}$ of membrane potential (V) is determined by the injected current (I) and the currents through Na^+, K^+, and g_{leaky} channels.

$$\frac{dV}{dt} = \frac{I - g_{\mathrm{Na}} m^3 h(V - E_{\mathrm{Na}}) - g_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) - g_{\mathrm{leak}}(V - E_{\mathrm{leak}})}{C_m} \tag{1}$$

```python
# Equation 1
dVdt = (I_total - I_Na - I_K - I_leak) / Cm
```

Listing 1: Python implementation of the Hodgkin-Huxley neuron's first equation

Equations 2, 3, 4 are the rate of change of the gating variables m, h, and n, which represent the probability that the Na^+ and K^+ channels are open.

```python
# Equations 2-4
dmdt = alpha_m(V) * (1 - m) - beta_m(V) * m
dhdt = alpha_h(V) * (1 - h) - beta_h(V) * h
dndt = alpha_n(V) * (1 - n) - beta_n(V) * n
```

Listing 2: Python implementation of the Hodgkin-Huxley neuron's second, third and fourth equation

$$\frac{dm}{dt} = \alpha_m(1 - m) - \beta_m m \tag{2}$$

$$\frac{dh}{dt} = \alpha_h(1 - h) - \beta_h h \tag{3}$$

$$\frac{dn}{dt} = \alpha_n(1 - n) - \beta_n n \tag{4}$$

The helper functions representing the α and β are used to calculate the rate constants for the gating variables in the differential equations.

```python
# Helper functions for rate constants
def alpha_m(V):
  return 0.1 * (V + 40.0) / (1.0 - math.exp
    (-(V + 40.0) / 10.0))
def beta_m(V):
  return 4.0 * math.exp(-(V + 65.0) / 18.0)
def alpha_h(V):
  return 0.07 * math.exp(-(V + 65.0) /
    20.0)
def beta_h(V):
  return 1.0 / (1.0 + math.exp(-(V + 35.0)
    / 10.0))
def alpha_n(V):
  return 0.01 * (V + 55.0) / (1.0 - math.
    exp(-(V + 55.0) / 10.0))
def beta_n(V):
  return 0.125 * math.exp(-(V + 65.0) /
    80.0)
```

Listing 3: Helper functions representing the α and β

$$\alpha_m(V) = 0.1\frac{V + 40.0}{1.0 - \exp\left(-\frac{V+40.0}{10.0}\right)} \tag{5}$$

$$\beta_m(V) = 4.0\exp\left(-\frac{V + 65.0}{18.0}\right) \tag{6}$$

$$\alpha_h(V) = 0.07\exp\left(-\frac{V + 65.0}{20.0}\right) \tag{7}$$

$$\beta_h(V) = \frac{1.0}{1.0 + \exp\left(-\frac{V+35.0}{10.0}\right)} \tag{8}$$

$$\alpha_n(V) = 0.01\frac{V + 55.0}{1.0 - \exp\left(-\frac{V+55.0}{10.0}\right)} \tag{9}$$

$$\beta_n(V) = 0.125\exp\left(-\frac{V + 65.0}{80.0}\right) \tag{10}$$

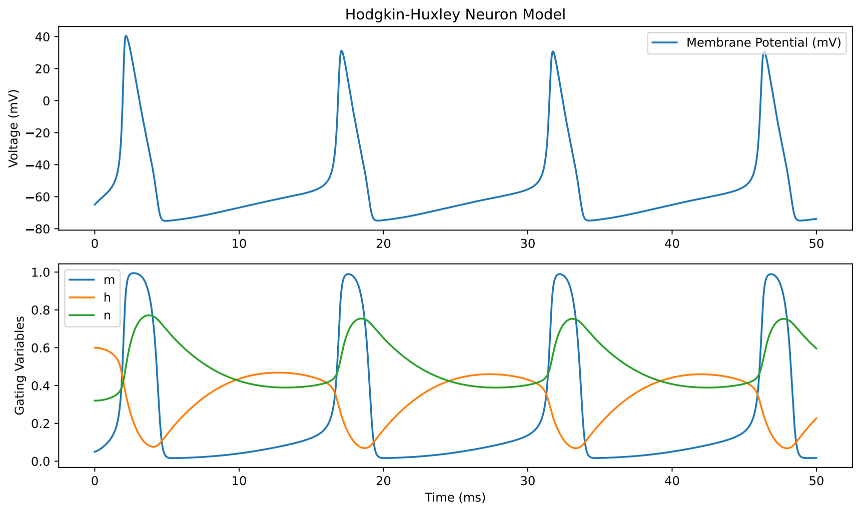

Figure 4: Hodgkin-Huxley neuron behavior under constant input current. Source: The author

The full code that plots the Figure 4 which shows the neuron behavior is shown in Listing 4.

```
import math
import matplotlib.pyplot as plt

# Hodgkin-Huxley model parameters
Cm = 1.0   # Membrane capacitance (uF/cm^2)
g_Na = 120.0  # Sodium conductance (mS/cm^2)
g_K = 36.0    # Potassium conductance (mS/cm^2)
g_leak = 0.3  # Leak conductance (mS/cm^2)
E_Na = 50.0   # Sodium reversal potential (mV)
E_K = -77.0   # Potassium reversal potential (mV)
E_leak = -54.387  # Leak reversal potential (mV)

# Time parameters
dt = 0.01  # Time step (ms)
t_max = 50.0  # Maximum simulation time (ms)

# Initial conditions
V, m, h, n = -65.0, 0.05, 0.6, 0.32  # V, m, h, n

# Lists to store results for plotting
time_points = []
membrane_potential = []
gating_variables_m = []
gating_variables_h = []
gating_variables_n = []

# Helper functions for rate constants
def alpha_m(V):
  return 0.1 * (V + 40.0) / (1.0 - math.exp(-(V + 40.0) / 10.0))

def beta_m(V):
  return 4.0 * math.exp(-(V + 65.0) / 18.0)

def alpha_h(V):
  return 0.07 * math.exp(-(V + 65.0) / 20.0)

def beta_h(V):
  return 1.0 / (1.0 + math.exp(-(V + 35.0) / 10.0))

```

```python
def alpha_n(V):
  return 0.01 * (V + 55.0) / (1.0 - math.exp(-(V + 55.0) / 10.0))

def beta_n(V):
  return 0.125 * math.exp(-(V + 65.0) / 80.0)

# Simulation loop
for t in range(int(t_max / dt)):
  # Membrane currents
  I_Na = g_Na * m**3 * h * (V - E_Na)
  I_K = g_K * n**4 * (V - E_K)
  I_leak = g_leak * (V - E_leak)

  # Total membrane current
  I_total = 10.0  # Injected current (adjust as needed)

  # Update variables using Euler method
  dVdt = (I_total - I_Na - I_K - I_leak) / Cm
  dmdt = alpha_m(V) * (1 - m) - beta_m(V) * m
  dhdt = alpha_h(V) * (1 - h) - beta_h(V) * h
  dndt = alpha_n(V) * (1 - n) - beta_n(V) * n

  V += dt * dVdt
  m += dt * dmdt
  h += dt * dhdt
  n += dt * dndt

  # Store results for plotting
  time_points.append(t * dt)
  membrane_potential.append(V)
  gating_variables_m.append(m)
  gating_variables_h.append(h)
  gating_variables_n.append(n)
```

Listing 4: Full implementation of a Hodkin-Huxley neuron

2.3.2. *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 5. 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.

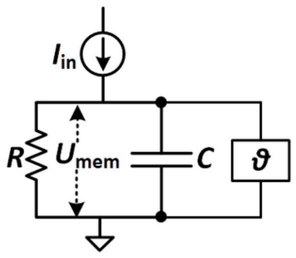

Figure 5: RC model: Source: [9]

Unlike the Hodgkin-Huxley neuron, spikes are represented as **sparsely** distributed **ones** in a train of **zeros**, as illustrated in Figure 6 and 7. This approach simplify the models and reduces the computational power and needed storage to run an SNN.

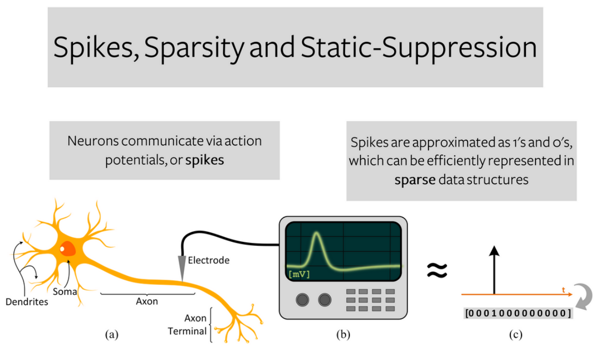
Spikes, Sparsity and Static-Suppression
Neurons communicate via action potentials, or spikes
Spikes are approximated as 1's and 0's, which can be efficiently represented in sparse data structures
Dendrites
Soma
Axon
Axon Terminal
Electrode
[mV]
[0001000000000]
(a)
(b)
(c)

Figure 6: Sparsity on Spike Neuron Networks. Source: [9]

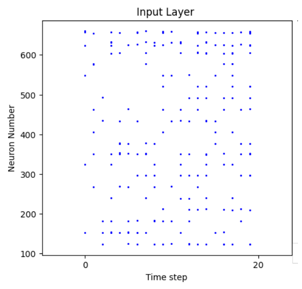

Figure 7: Sparse activity of a SNN: The horizontal axis represents the time step of some data being processed the vertical are the SNN neuron number. Note that, most of the time, very few neurons gets activated. Source: The author.

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 [9].

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 11.

$$C = \frac{Q}{V_{mem}(t)} \tag{11}$$

Than the neuron charge can be expressed as Equation 12.

$$Q = C.V_{mem}(t) \tag{12}$$

To know how these charge changes according to the time (aka current) we can derivate Q as in Equation 13. This expression express the current in the capacitive part of the neuron I_C

$$I_C = \frac{dQ}{dt} = C.\frac{dV_{mem}(t)}{dt} \tag{13}$$

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

$$V_{mem}(t) = R.I_R \implies I_R = \frac{V_{mem}(t)}{R} \tag{14}$$

Than considering that the total current do not change, as seen in Figure 8, we have the total input current I_{in} of the neuron as in Equation 15.

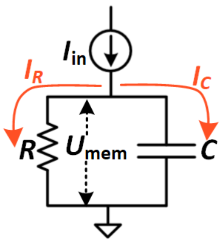

Figure 8: RC model for currents: $I_{in} = I_R + I_C$

$$I_{in}(t) = I_R + I_C \implies I_{in}(t) = \frac{V_{mem}(t)}{R} + C.\frac{dV_{mem}(t)}{dt} \tag{15}$$

Therefore, to describe the passive membrane we got a linear Equation 16.

$$\begin{aligned} I_{in}(t) &= \frac{V_{mem}(t)}{R} + C.\frac{dV_{mem}(t)}{dt} \implies \\ I_{in}(t) - \frac{V_{mem}(t)}{R} &= C.\frac{dV_{mem}(t)}{dt} \implies \\ &\boxed{R.I_{in}(t) - V_{mem}(t) = R.C.\frac{dV_{mem}(t)}{dt}} \end{aligned} \tag{16}$$

Than, if we consider $\tau = R.C$ as the **membrane time constant** we get voltages on both sides of Equation 17 which **describes the RC circuit**.

$$\begin{aligned} R.I_{in}(t) - V_{mem}(t) &= R.C.\frac{dV_{mem}(t)}{dt} \implies \\ R.I_{in}(t) - V_{mem}(t) &= \tau.\frac{dV_{mem}(t)}{dt} \implies \\ &\boxed{\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t)} \end{aligned} \tag{17}$$

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 18.

$$\begin{aligned} \tau.\frac{dV_{mem}(t)}{dt} &= R.I_{in}(t) - V_{mem}(t) \implies \\ \tau.\frac{dV_{mem}(t)}{dt} &= -V_{mem}(t) = \\ e^{\ln(V_{mem}(t))} &= e^{-\frac{t}{\tau}} = \\ &\boxed{V_{mem}(t) = V_{mem}(0).e^{-\frac{t}{\tau}}} \end{aligned} \tag{18}$$

Then one can say that: In the absence of an input I_{in}, the membrane potential decays exponentially as illustrated in Figure 9 and implemented in Listing 5.

```python
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 5: Python implementation of the action potential decaying of a LIF: $I_{in} = 0$

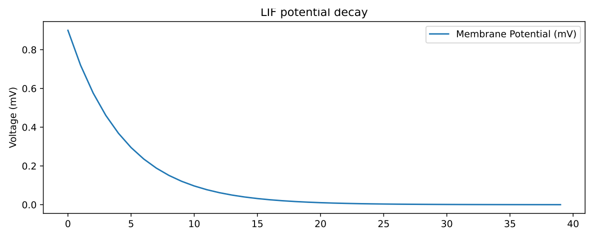

Figure 9: Membrane potential decaying. Source: The author

With the results from Equation 17 it is possible to calculate the action potential increasing as seen in Equation 19.

$$
\begin{aligned}
&\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) = \\
&\tau.\frac{dV_{mem}(t)}{dt} + V_{mem}(t) = R.I_{in}(t) = \\
&\frac{dV_{mem}(t)}{dt} + \frac{V_{mem}(t)}{\tau} = \frac{R.I_{in}(t)}{\tau} \implies \\
&\text{Integrating factor: } e^{\int \frac{1}{\tau} dt} = e^{\frac{1}{\tau}.t} \implies \\
&(e^{\frac{1}{\tau}.t}.V_{mem}(t))' = \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} = \\
&\int (e^{\frac{1}{\tau}.t}.V_{mem}(t))' = \int \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} dt = \\
&e^{\frac{1}{\tau}.t}.V_{mem}(t) = \int \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} dt \therefore \\
&\text{Considering: } V_{mem}(t=0) = 0 \implies \\
&\boxed{V_{mem}(t) = I_{in}(t).R(1 - e^{\frac{1}{\tau}})}
\end{aligned}
\tag{19}
$$

Note that when action potentials increases there is still an exponential behavior as seen in Figure 10 and implemented in Listing 6.

```python

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

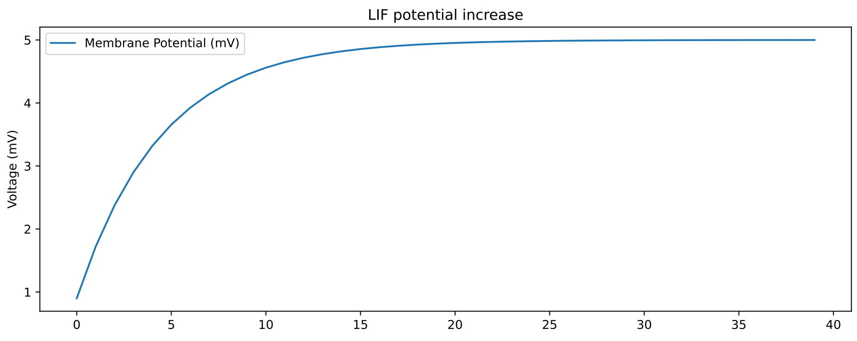

Figure 10: Membrane potential increasing. Source: The author

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 make a full simulation depicted in Figure 11 and implemented in Listing 7:

```python
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

input = np.concatenate((np.zeros(50), np.ones(20)*(.5), np.zeros(50)), 0)
num_steps = len(input)
V_mem = 0
V_trace = []

for step in range(num_steps):
  V_trace.append(V_mem)
  V_mem = lif(V_mem, I_in=input[step])
```

Listing 7: Python implementation of the action potential full simulation of a LIF: $I_{in} = 1$, $V_{thresh} = 2$ is threshold

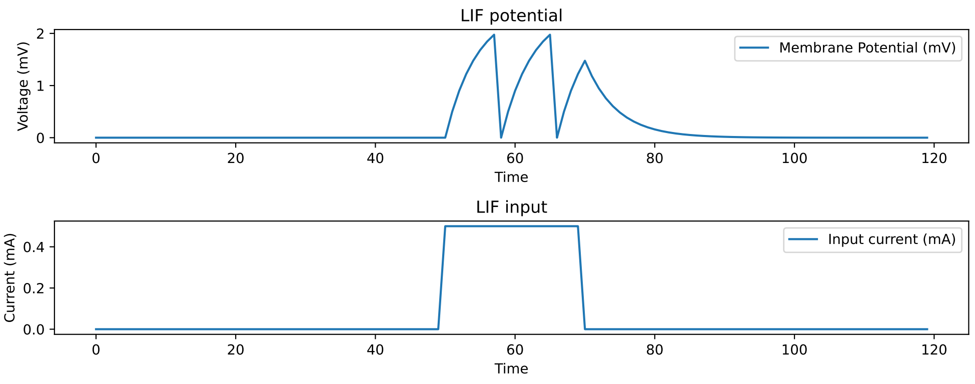

Figure 11: Plot of the full simulated LIF: A 0.5 mA was provided from time 51 to 70.

3. Experiments

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.

3.1. Full connected spiking neurons SNN

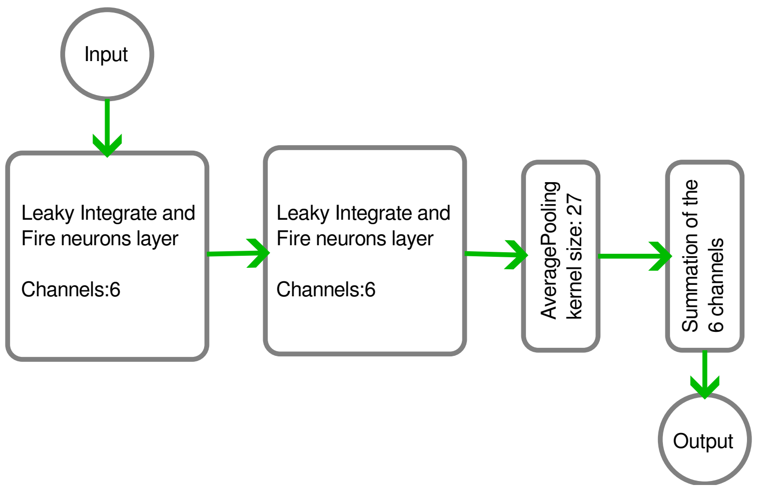
Input
Leaky Integrate and Fire neurons layer
Channels:6
Leaky Integrate and Fire neurons layer
Channels:6
AveragePooling kernel size: 27
Summation of the 6 channels
Output

Figure 12: Spiking neural network architecture

The SNN were trained according to the following parameters:

- **random seed**: 69 (*nice!*)
- **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 8: SNN architecture

3.2. *Convolutional layer with full connected sigmoid neurons CNN*

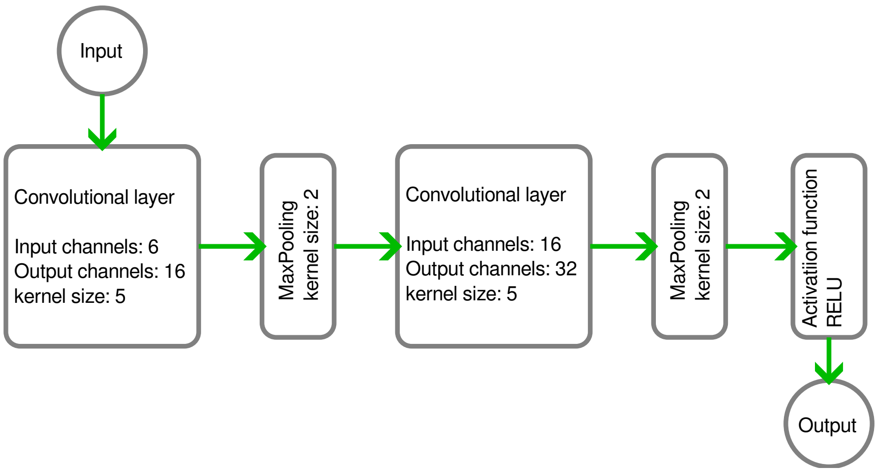

Figure 13: Convolutional neural network architecture

The CNN were trained according to the following parameters:

- **random seed**: 69 (*nice!*)
- **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 9: CNN architecture

The used database [10] 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 [11].

4. Conclusions

Until now the model of a decaying membrane potential is interesting but does not explains how an spike neuron can be used and modeled in a network. Using the Equation 17 which describes an RC model it is possible to create such thing.

As aforementioned starting with the Equation 17 and using Forward Euler Method to solve the LIF model:

$$\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) \tag{20}$$

Solving the derivate in Equation 21 results in the membrane potential for any time $t + \Delta t$ in future:

$$\begin{gathered}\tau.\frac{V_{mem}(t+\Delta t) - V_{mem}(t)}{\Delta t} = R.I_{in}(t) - V_{mem}(t) = \\ V_{mem}(t+\Delta t) - V_{mem}(t) = \frac{\Delta t}{\tau}.(R.I_{in}(t) - V_{mem}(t)) = \\ \boxed{V_{mem}(t+\Delta t) = V_{mem}(t) + \frac{\Delta t}{\tau}.(R.I_{in}(t) - V_{mem}(t))}\end{gathered} \tag{21}$$

References

[1] 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

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

[3] 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

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

[5] 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

[6] 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

[7] 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

[8] Gibbs Energy and Redox Reactions, [Online; accessed 2023-11-19] (jul 7 2023).

[9] 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.

[10] 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

[11] 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.