Spiking neural networks

Study of efficient training methods for classification problems

Universitat Oberta de Catalunya

David Sun

Máster universitario en Ciencia de Datos

Área: Inteligencia artificial

Tutor de TF
Paolo Dini

Profesor/a responsable de la asignatura
Jordi Casas Roma

Enero 2023

Universitat Oberta de Catalunya

uoc.edu

Ficha del Trabajo Final

Título del trabajo:	Spiking neural networks: Study of efficient training methods for classification problems
Nombre del autor/a:	David Sun
Nombre del Tutor/a de TF:	Paolo Dini
Nombre del/de la PRA:	Jordi Casas Roma
Fecha de entrega:	01/2023
Titulación o programa:	Máster universitario en Ciencia de Datos
Área del Trabajo Final:	Artificial Inteligence, Machine learning
Idioma del trabajo:	Inglés
Palabras clave	Deep learning, spiking neural network, neuromorphic computing

Resumen del Trabajo

En este trabajo se estudiará la arquitectura de las redes neuronales *spike* (SNN), unas redes mucho más parecidas a las biológicas que las redes neuronales artificiales (ANN), ya que de transmiten la información en forma de impulsos discontinuos, en contraposición a los impulsos continuos de esta última.
Una de las ventajas de las SNN enfrente a las ANN, es que necesitan menos cálculos y por tanto su consumo energético es menor, pero hay un problema principal en este tipo de arquitecturas, y es el hecho de que no hay un método estándar de entrenamiento.
Específicamente vamos a enfocarnos en métodos de entrenamiento eficientes para el entrenamiento de las SNN en tareas de clasificación en *datasets* como el MNIST. Algunos modelos están llegando a las precisión de los métodos actuales de las ANN y si los SNN se usará en los ordenadores neuromórficos habría un mayor ahorro energético.

Abstract

In this work we will study the architecture of spike neural networks (SNN), networks that are much more similar to biological ones than artificial neural networks (ANN), since they transmit information in the form of discontinuous impulses, as opposed to continuous impulses of the latter.

One of the advantages of SNNs compared to ANNs is that they require fewer calculations and therefore their energy consumption is lower, but there is a main problem with this type of architecture, and it is the fact that there is no standard training method.
Specifically, we are going to focus on efficient training methods for training SNNs in dataset classification tasks such as MNIST. Some models are reaching the precision of current ANN methods and if SNNs are used with neuromorphic computers there would be greater energy savings.

Index

List of Figures

List of Tables

1. Introduction

1.1. Context and project justification

Artificial neural networks (ANN) began with the perceptron, created by McCulloch and Pitts in 1943, then implemented by Frank Rosenblatt in 1958, where it was later completed with the **backpropagation algorithm** by Paul Werbos in 1975, which corrected the weights of the network so it could be trained and learn from mistakes.

These algorithms didn't take off until the 21th century surpassing the classic classification or regression algorithms of the state of the art thanks to the exponential increase in available data, in this way all kinds of neural network architectures emerged.

We came from solving classic problems to much more complex ones such as real-time translation using **natural language processing** (NLP), self-driving car, **Computer Vision** (CV), image generation, etc.

In 2020 they managed to solve one of the most important milestones that science haven't been able to solve: the prediction of protein folding, **AlphaFold 2**, an artificial intelligence from DeepMind obtained in the Critical Assessment of protein Structure Prediction (CASP14) competition a score greater than 90 in the global distance test (GDT), where a score of 100 denotes perfect prediction, in almost two thirds of the proteins used [2]. Thanks to this achievement we can speed up the research processes since current techniques for determining the three-dimensional structure such as X-ray crystallography can take a long time compared to minutes or hours for AlphaFold 2.

As we can see this field moves quick and we are solving problems never touched before, but we have one inconvenient with the current neural network models: their energy consumption, since we are imposing a neural network architecture on a hardware type **von Neumann** as we can see in Fig. 1, there is a bottleneck in the part of the data exchange between the memory unit and the processing unit, in addition to a latency problem.

This project will be focused on study of **neuromorphic computing**, in special the **spiking neural network** (SNN), inspired by biological neurons for tackling the energy consumption issue.

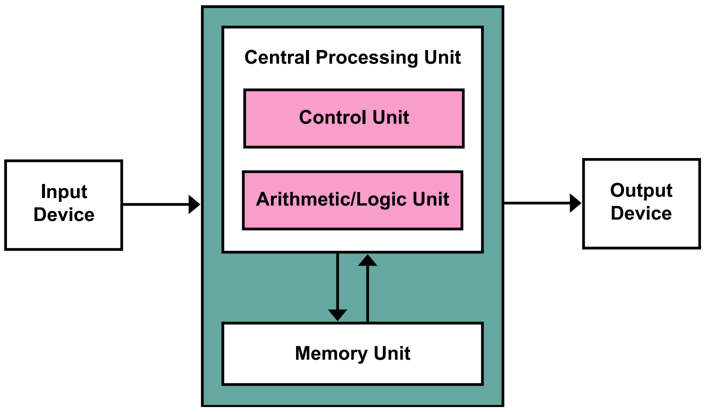


Figure 1. von Neumann architecture

1.2. Project Objectives

Neuromorphic neural networks, specifically in this case SNN have an advantage compared to ANN because it requires fewer neurons and a lower energy consumption. In theory SNN are more efficient networks but they have a problem and there is no standardized training process, the objectives of this thesis are:

1. Study of SNN architecture and the state of the art.
2. Study of the biological coding.
3. Study of methods for training SNN.
4. Implementation libraries that enable SNN to classify datasets, such as NMIST.

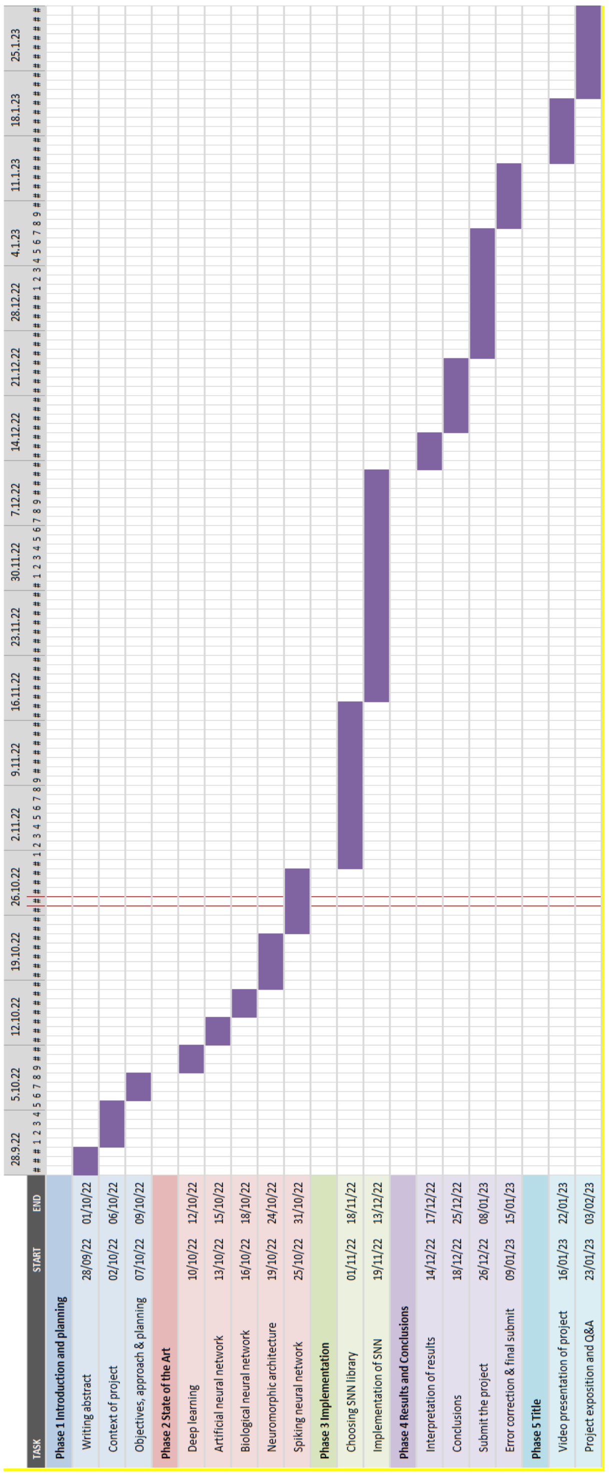
1.3. Approach and methodology

This thesis will be a research project, mainly theoretical, starting with the fundamental, reviewing and understanding the basic knowledge of **Deep Learning** (DL) and neural networks in both artificial and biological, then we combine those insights in order to comprehend better neuromorphic computing and SNN, the latter being the main focus of this project.

Once the research phase is done, we will start to implement the SNN, based on the state-of-the-art methodologies to train these neural networks, and for that task we will use **Tensorflow** and **Pytorch** (both frameworks for DL) to simulate different works using **CUDA** (Compute Unified Device Architecture) with a Graphic NVIDIA GeForce RTX 3060.

1.4. Planning

The planning for the thesis is depicted by the follow **gantt diagram**:



TASK	START	END
Phase 1 Introduction and planning		
Writing abstract	28/09/22	01/10/22
Context of project	02/10/22	06/10/22
Objectives, approach & planning	07/10/22	09/10/22
Phase 2 State of the Art		
Deep learning	10/10/22	12/10/22
Artificial neural network	13/10/22	15/10/22
Biological neural network	16/10/22	18/10/22
Neuromorphic architecture	19/10/22	24/10/22
Spiking neural network	25/10/22	31/10/22
Phase 3 Implementation		
Choosing SNN library	01/11/22	18/11/22
Implementation of SNN	19/11/22	13/12/22
Phase 4 Results and Conclusions		
Interpretation of results	14/12/22	17/12/22
Conclusions	18/12/22	25/12/22
Submit the project	26/12/22	08/01/23
Error correction & final submit	09/01/23	15/01/23
Phase 5 Title		
Video presentation of project	16/01/23	22/01/23
Project exposition and Q&A	23/01/23	03/02/23

Figure 2. Gantt diagram and planning of the thesis

2. State of the art

2.1 Deep Learning

First let's introduce the meaning of **Artificial Intelligence** (IA), it's simply the capacity of machines to learn and solve problems, inside this field we can narrow down to **Machine Learning** (ML), a subset of IA, here machines can learn without explicitly programming. After ML algorithms learns the patterns of the data that have been trained, then the model can make generalizations and predictions when new data is presented.
And inside the scope of ML, we have **Deep Learning** (DL), this type of algorithm uses a network of layers with artificial neurons, and within the last decade it has become the state-of-the-art algorithm for the vast majority of the ML problems.
In Fig. 3 we can appreciate how the definitions above are related to each other.

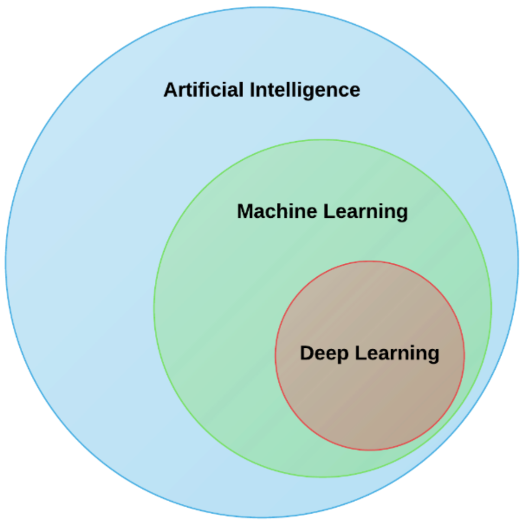


Figure 3. Artificial Intelligence, Machine Learning and Deep Learning

First of all, we need to have a clear understanding of the components of this type of ML:

Artificial neural networks (ANN): it's the backbone structure of deep learning algorithm, it consists of vertical layers of interconnected neurons, as shown in Fig. 4. The input layer receives data and feeds it into the network, which then passes through one or more hidden layers before reaching the output layer, where the results are produced.

Feedforward neural network (FNN): it's an ANN but without cycles or loops, the information only flows in one direction, in Fig. 4 we can see then in the first 3 rows.

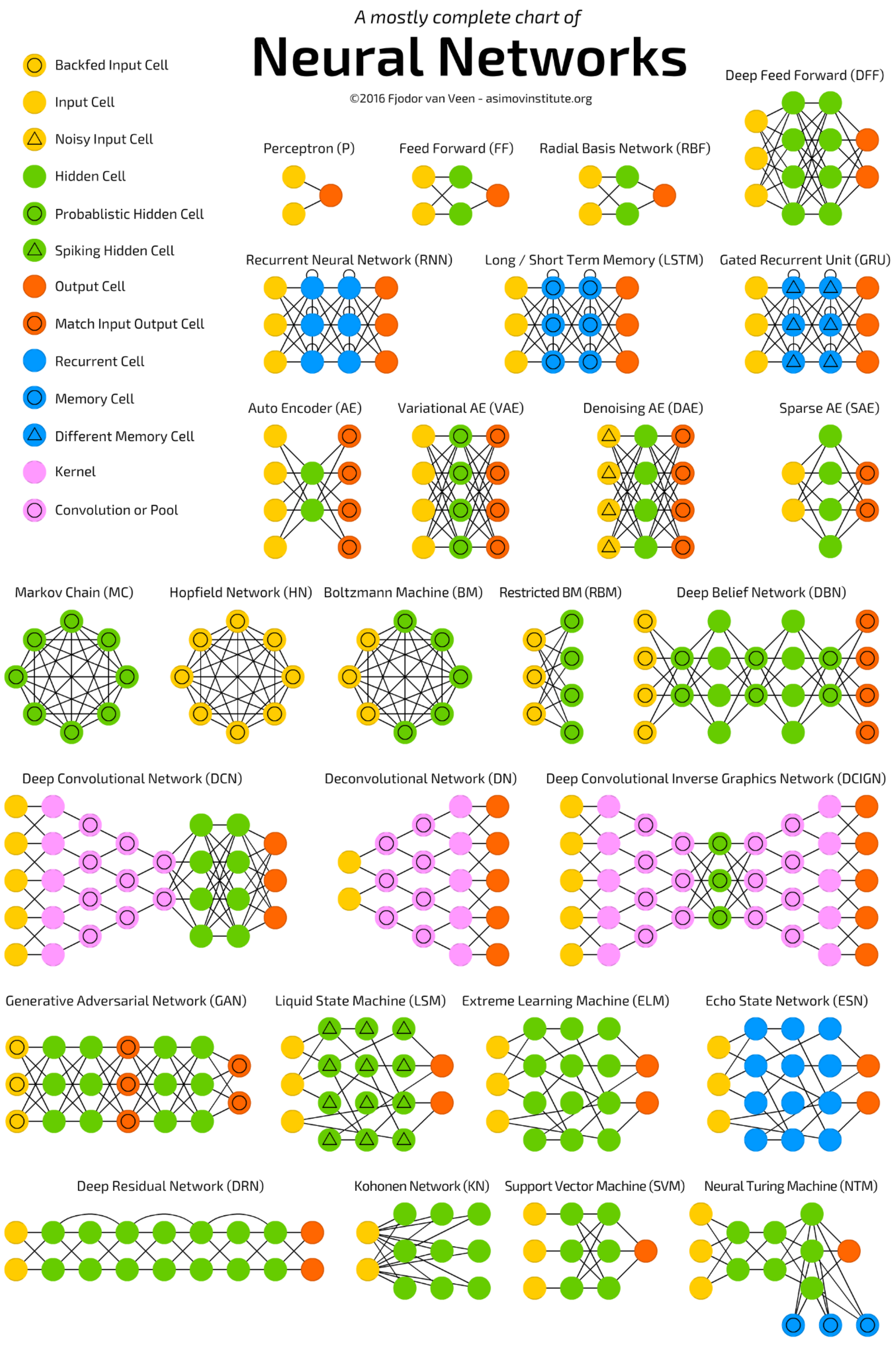

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Figure 4. Variety of neural network architectures

Perceptron: the simplest type of FNN, only with 2 layers of neurons, the input cell and the output cell.

2.2 The Artificial Neuron

The structure of a simple artificial neuron is shown in Fig. 5. The input layer receives data, which is then assigned a random weight by the system. These weights are usually small numbers. The **net input function**, also known as the **summarization function**, processes the weighted inputs and produces a single output value. The most commonly used summarization function is the **weighted sum (u).**

$$u = \sum_{i=1}^{n} w_i x_i \tag{1}$$

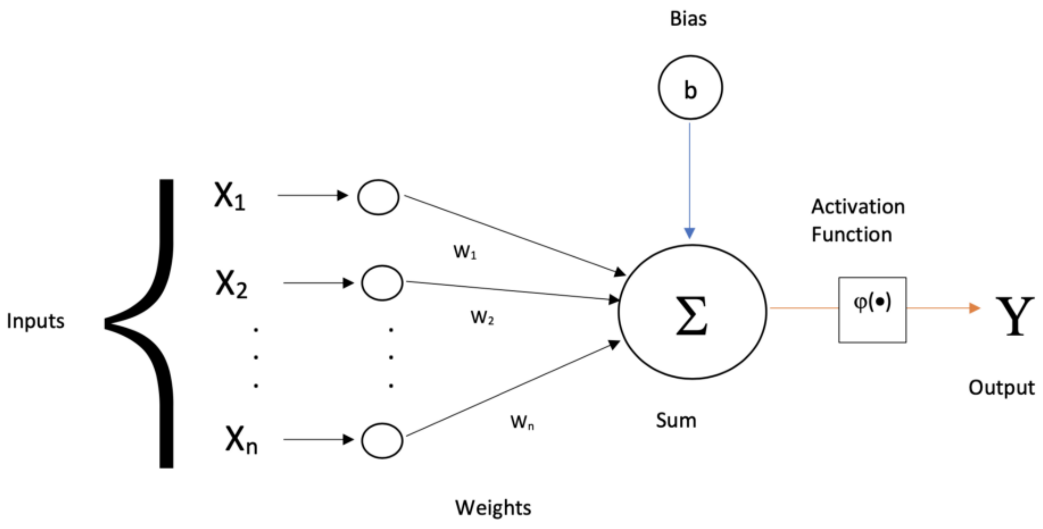


Figure 5. Artificial neuron

The input for the **activation function** is determined by the **net input function**, which determines the intensity of the output (Y). Some common activation functions include ReLu, sigmoid, binary step, linear, and Tanh. These functions determine the strength of the output signal given the input.
In **forward propagation**, the predicted output is used to calculate the error between the predicted and desired output. There are various error functions that can be used, such as ε = ½(ŷ- y)^2. This error is then used to determine how well the neural network is performing applying the **cost function**:

$$\frac{1}{n}\sum(\hat{y}_i - y_i)^2 \tag{2}$$

Backpropagation involves adjusting the weights for each cell in the neural network in order to minimize the cost function. This is done by using the **gradient descent** algorithm, which involves calculating the derivative of the cost function to determine the slope, and then iteratively adjusting the weights in the direction that reduces the cost. The training process continues until the cost function no longer decreases significantly, indicating that the neural network has converged. It's important to note that while ANNs are inspired by biological neurons, there are fundamental differences in their structure, learning protocols, and computation.

2.3 Energy Consumption

According to some calculations, if the actual pace in energy consumption in 2040 continues, the data centers are going to pass 10^{27} Joules in expenditure, exceeding the energy produced by the entire world [4]. Some large neural networks, such as GPT-3, require approximately 190,000 kWh of energy to train. This is equivalent to the CO2 emissions produced by the consumption of a car or the consumption of an individual over a certain period of time, as shown in Fig. 6.

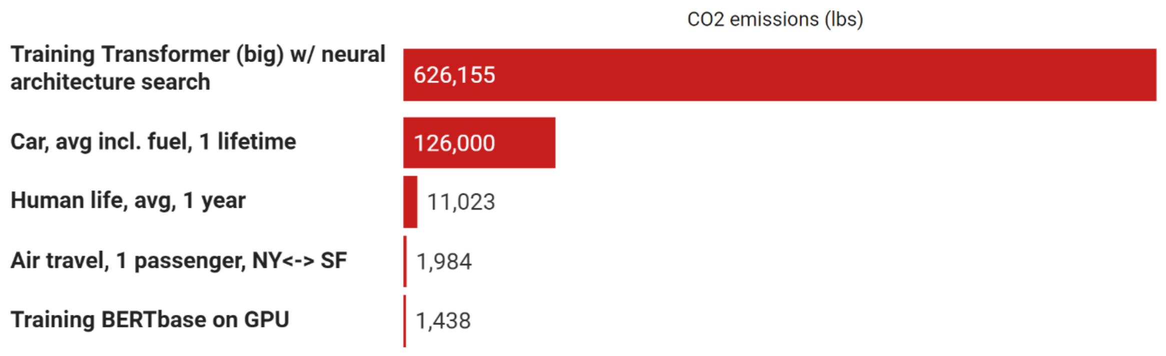


Figure 6. Carbon footprint comparison

2.4 Biological Neural Network

Meanwhile, the human brain is able to perform complex tasks, such as image recognition, with as little as 20 watts of power, with accuracies equal to or better than the best state-of-the-art models. In human brain the neurons are all interconnected, recurrent with irregular patterns, which can be connected to many other neurons, in Fig. 7 we can depict the synapse between 2 neurons.

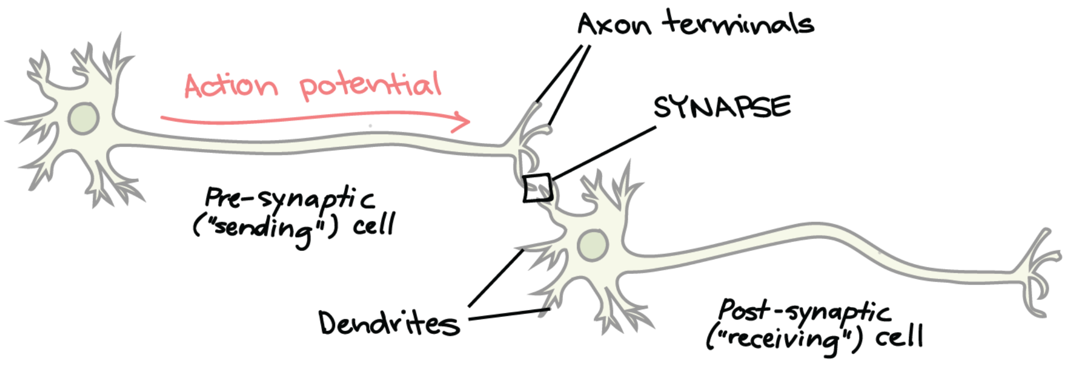

Axon terminals
Action potential
SYNAPSE
Pre-synaptic ("sending") cell
Dendrites
Post-synaptic ("receiving") cell

Figure 7 Biological neuron synapse.

The **presynaptic neuron** has a **resting membrane potential**, which is the voltage across the membrane of the neuron when it is not actively conducting an **action potential** (also known as a "spike"). When the presynaptic neuron becomes activated or **depolarized**, the voltage across its membrane increases. If the membrane potential reaches a threshold level, as depicted in Fig. 8, an action potential is generated and travels down the length of the neuron. When the action potential reaches the **axon terminal**, the presynaptic neuron releases neurotransmitter molecules into the synapse, which bind to receptors on the dendrites of the postsynaptic neuron. The binding of these molecules can either excite or inhibit the postsynaptic neuron, depending on the type of neurotransmitter and the receptors it activates. If the postsynaptic neuron becomes sufficiently excited, it will generate an action potential of its own, which can then propagate to other neurons in the network.

After an action potential is generated and propagates down the neuron, the membrane potential decreases rapidly and briefly falls below the resting potential. This is known as the **refractory period**, during which it is difficult or impossible for another action potential to be generated. The neuron then returns to its resting state.
The connection between a presynaptic neuron and a postsynaptic neuron is usually one-to-many, with the axon terminal of the presynaptic neuron connected to multiple dendrites of the postsynaptic neuron. Each dendrite can be modulated by multiple presynaptic axon terminals.

The effect of the neurotransmitter released by the presynaptic neuron on the postsynaptic neuron can either be excitatory or inhibitory. An **excitatory postsynaptic potential (EPSP)** depolarizes the membrane and increases the likelihood of an action potential being generated, while **an inhibitory postsynaptic potential (IPSP)** hyperpolarizes the membrane and reduces the likelihood of an action potential being generated. These two types of signals can act in opposition to each other, with EPSPs trying to excite the postsynaptic neuron and IPSPs trying to inhibit it. The balance between these two types of signals determines whether or not an action potential is generated in the postsynaptic neuron.

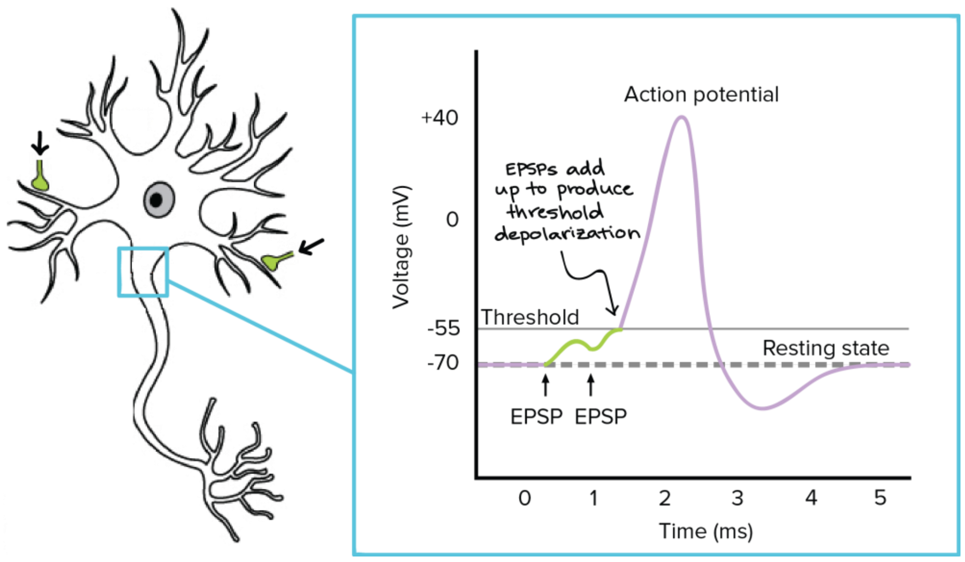


Figure 8. Action potential.

2.5 Neuromorphic Computing

Recently researchers began to develop a **neuromorphic architecture** inspired by the structure of the human brain, that could solve the problem of energy consumption and latency, one of the most outstanding qualities of this design is has the ability to scale by placing of memory and the processing unit on the same place, thus improving energy consumption and latency by avoiding the bottlenecks of von Neumann's architecture. There are several technologies that allow for the integration of memory and processing units, including memristors, threshold switches, spintronic memories, and oxide ionotronic neuromorphic transistors [10].

One major difference between traditional ANNs and biological neural networks is the connections in ANNs are **synchronous**, meaning that information only flows forward when each layer has fully processed all of the information. In contrast, information in biological neural networks travels **asynchronously** in the form of **spikes** and can flow in any direction through the complex 3D network of the brain.

One advantage of silicon-based chips over biological brain is the speed, we can assemble very fast circuits, in order of magnitude faster than biological neurocircuits, another advantage of **complementary metal oxide semiconductors (CMOS**) circuits is the reliability, precision and deterministic operations, those characteristics are also present in the brain but at a much higher cost because of redundancy that is needed to archive them. Several companies, such as Intel and IBM, are developing chips based on neuromorphic architectures like **Loihi** and **TrueNorth**, and there are also several research institutions working on projects such as the **Blue Brain Project** and the **Human Brain Project**.

2.6 Spiking Neural Network

Thanks to advances in neuromorphic computing, they also began to develop a type of architecture with neurons based on biological electrical impulses called **spiking neural network** (SNN), which can mimic the network typology from ANN, and also operates by the sum of input weights, but instead passing the outcome to sigmoid or Hyperbolic Tangent the weighed sum changes the membrane potential, and if threshold is reached then a spike is emitted. In this project we will focus on the functioning of this type of neurons, its architecture, models and training methods.

2.6.1 Information Encoding

The living body utilizes biological sensors to capture analog information from external sensory stimuli. Once the signal is received, it is converted into a series of spikes and transmitted to the brain as a binary encoding. These spikes act as inputs for neurons, which transmit them through the neural network until they reach a specific region responsible for decoding and interpreting the signals. While **binary encoding** is one way in which the brain communicates, some retinal neurons can also use **graded potential** to communicate with each other [11]. Some SNN models also follow this approach of binary encoding, treating individual spikes as binary values of 0 or 1 rather than the actual spikes themselves.

Neural coding is still a debate, some may say it's temporal coding others rate coding, but it could be a combination of both, with temporal coding being dominant [12], because of energy expenditure, here in the deep learning field this debate might be not the central discussion. These are the coding strategies used by biological neural networks [13]:

- **Rate or frequency-based encoding**: information is stored inside of the spike count, firing rate or latency. The more intense the stimulus, the higher the frequency of spikes. There are two ways rate is coded:

    - **Spike-count rate (average over time):** it is calculated by dividing the count of spikes by time, the mean firing rate, the time usually used is between 100-500ms, and represented in s^{-1} or Hz.

    - **Time-dependent firing rate (averaging over several trials)**: it is the average spike input aggregation of several runs of the same neuron giving us a density diagram as the **Peri-Stimulus-Time Histogram (PTSH)** showed on Fig 9. This type of encoding might be classified also as **population encoding**, because of the population dynamics and interactions but without excluding being rate encoding.

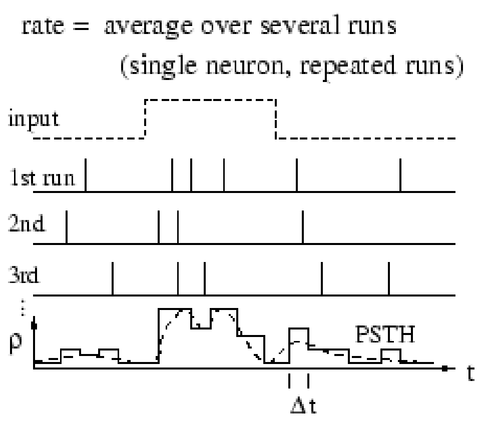

rate = average over several runs
(single neuron, repeated runs)
input
1st run
2nd
3rd
ρ
PSTH
t
Δt

Figure 9. **PTSH i**n the bottom histogram as the average of the 3 runs.

- **Temporal or latency encoding**: the timing of the spike is where the information is carried. Human reaction to image display can be as fast as 400ms, if neurons remain hover during this period of time and calculates the average then respond form the stimulus would be much latter [14]. There are some approximations [13]:

    - **Time-to-First-Spike (or TTFS):** it is the time that takes for a neuron or a group of neurons to respond to a specific stimulus, as we can see in the Fig. 10 the arrow shows the start time of stimulus, then the different times the 3 neurons spikes.

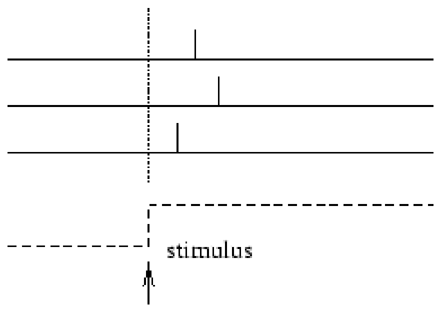


Figure 10. **Time-to-First-Spike**

    - **Phase-of-firing**: it is the timing of spikes relative to some periodic reference signal, such as neural background oscillation, then the neurons can encode the information respect to the oscillation.

    - **Correlations and Synchrony**: like phase-of firing, neurons can synchronize, but this time is with each other firing one after other in a pattern, some of them may fire altogether in synchrony, this strategy also could be classified as **population encoding.**

We only extract the usefulness of each encoding for constructing SNN, both having advantages over the other:

Rate coding advantages:
- **Reliability**: one missing or one more spike doesn't matter in general, as the other spikes weights out the error made by the faulty one.
- **Better learning in backpropagation**: more spikes supply with a robust gradient signal when the chosen training method is error backpropagation.

Latency encoding advantages:
- **Energy usage**: less spikes means less energy consumption, also less memory use.
- **Speed:** when information is only encoded by one spike, it doesn't need to wait for the entire group of spikes to arrive at the decoding destination.

2.6.2 SNN models

Different models had been developed over the years in order to simulate the biological SNN but for computing purposes simpler models are used focusing on modeling the spike, the most relevant models for this project are listed below:

The Hodgkin-Huxley model: is one of the most accurate and realistic models, which represents with details the flow of ions, the interactions between them and the membrane potential for generation of the spike, the model is very expensive computationally so it's not very suitable to simulate in computers, even though it is possible.

The Izhikevich model: this is a simpler form of Hodgkin-Huxley model, developed with the purpose to use computer simulation. The model has the ability to model spikes but cannot describe the behavior of all types of neurons. It can be summarized by the variables *v* and *u* in the equations below [15]:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{3}$$

Equation (3) being the change of voltage in time and I the current injected to the neuron, *u* is the recovery variable it pulls the *v* back to 0.

$$\frac{du}{dt} = a(bv - u) \tag{4}$$

Equation (4) being the change of recovery variable in time.
- *a* is the time scale of the recovery variable *u*, smaller values result in slower recovery, typical value is *a* = 0.02.
- *b* explains the sensibility of the recovery variable *u*. typical value is *b* = 0:2

$$if\ v \geq 30\ mV, then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \quad (5)$$

If *v* passes 30 mV, then the voltage is reset and recovery variable increased.
- *c* value typically is *c* = -65 mV.
- *d* typical value is *d* = 2.

This model captures some dynamics of potential generation as we see some samples in Fig. 11.

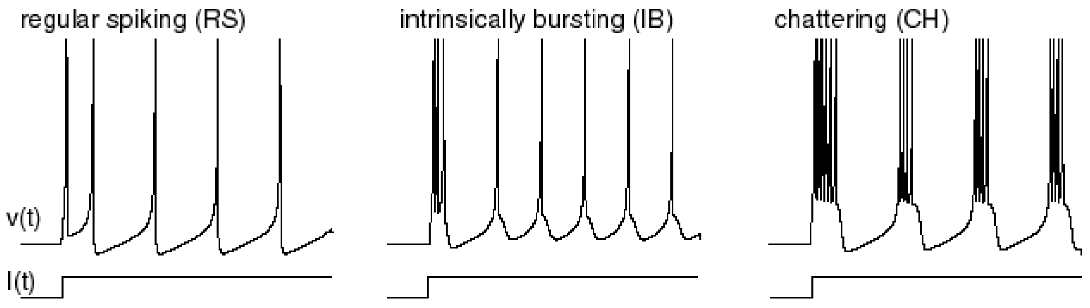


Figure 11 Spikes from The Izhikevich model

The leaky integrate-and-fire (LIF) model: this model is the most used model in SNN, very similar to the Izhikevich model, very efficiently for computer simulations. Represents a neuron with the membrane potential dynamics.
When current is being injected and membrane potential passes the threshold, a spike is emitted, and time is marked as an instant atomic event, then membrane potential decays exponentially to the baseline resting potential in Fig. 12 we can see some examples.

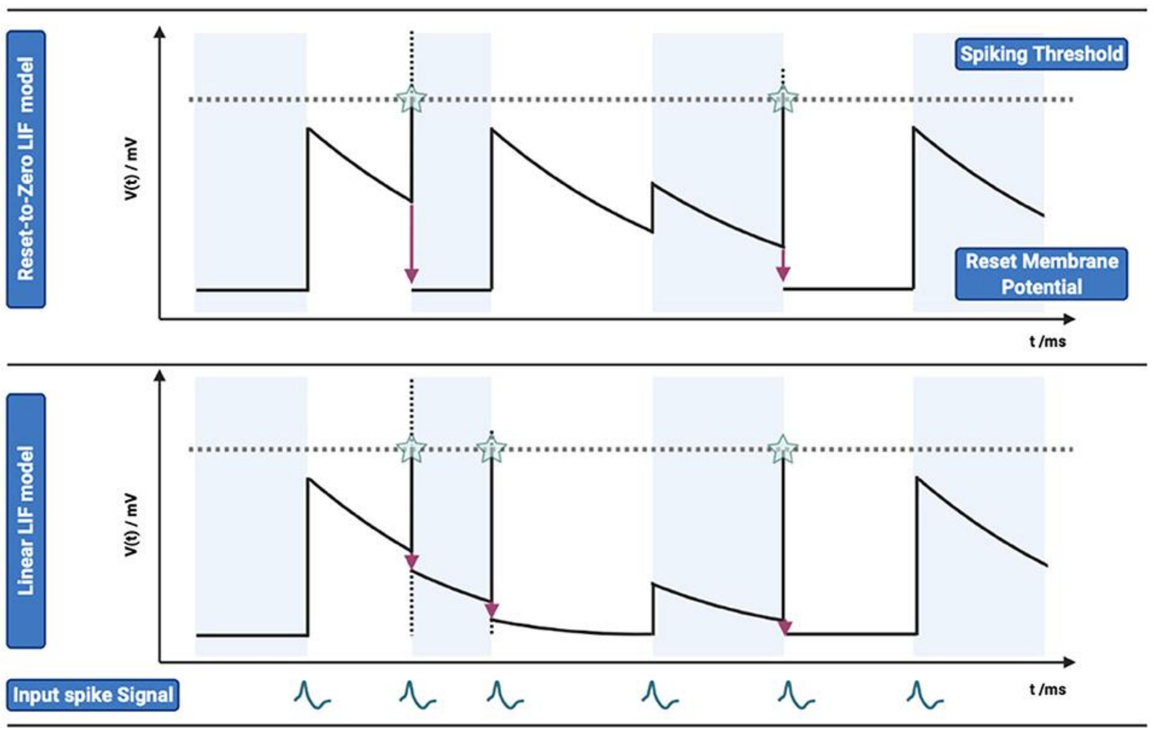


Figure 12. Examples of LIF model: **reset-to-zero model** is when spike is reset to resting potential after spike and **linear LIF model**, when spike is produced after surpassing the threshold, the membrane potential decays following the previous rate.

The model can be explained with the equations:

$$C_m \frac{dv}{dt} = I_{leak} + I_{spike} + I_{current} \tag{6}$$

C_m is the **capacitance** across the neuron membrane, $\frac{dv}{dt}$ the difference of voltage in time, I_{leak} is the **leak current**, which pulls downs the voltage to the resting potential, I_{spikes} is the current brought by a spike, it is almost an instant and atomic instance, getting a constant value S when spiking or else 0. $I_{current}$ is the injected current, could be a constant or be absent. Leak current is developed as:

$$I_{leak}(t) = \frac{C_m}{\tau_m}(v(t) - V_0) \tag{7}$$

V_0 being the resting voltage, τ_m is a time constant, controlling the time the leaking will last, the major the tau the longer will reach the resting potential. The **leak term** causes a continuous decrease of the membrane potential and thus prevents long-range correlations. If we remove the spike and injected current, we end up with the voltage decay thorough time until resting potential:

$$\frac{dv}{dt} = -\frac{1}{\tau_m}(v(t) - V_0) \tag{8}$$

2.6.3 SNN Learning Rules

When a **loss** is determined, the network must update the parameters in the network iteratively to improve the performance. We see every weight adds its portion to the global loss, it’s called the **credit assignment**, which can be classified into **temporal and spatial credit assignments**. The objective of solving temporal assignment is finding the time when the weight adds to the error. The objective of solving the spatial credit assignment is finding the location of the weight. Backprop is used to tackle spatial credit assignment, it needs to go forward then backwards, action unlikely to happen in the brain, so a few approaches are developed in order to change backprop into some biological feasible spatial credit assignment by shifting global errors for local errors, and change parts of the gradient computation for some randomness as [17]:

- **Perturbation Learning**: Inserts a random perturbation in the weights then measure the error, if the error decreases, the change will persist or else will be excluded, this approximation takes a large number of steps to function.
- **Random Feedback**: this technique changes the synaptic weights for random weights, it somehow reduces the **weight transport** problem (better explained in section 2.7.2.2).
- **Local Losses**: every neuron layer has its own cost function rather than a general error signal.

2.6.3.1 Unsupervised Learning

In unsupervised learning, automatic rules are set for the network to recognize patterns inside data, without the output labels. In SNN the most prominent learning mechanism is the **Spike-timing dependent plasticity (STDP)**, this technique is inspired by the **Hebbian Learning**, this theory is based on the mantra: "**Cells that fire together wire together**", biological neurons strengthen their connections when are activated at the same time, thus adapting the connections in the learning process. In STDP the strength of connection between 2 neurons can be modulated by the spike timing explained as followed:

- **Long-term potentiation (LTP):** if post-synaptic neuron fires just after pre-synaptic neuron the connection is strengthened, because the pre-synaptic spike could be caused by the post-synaptic spike.

- **Long-term depression (LTD):** if post-synaptic neuron fires before pre-synaptic neuron the connection is weakened.

The timing can be written as the t_{post} post-synaptic spike and t_{pre} as pre-synaptic spike both range timing around few milliseconds:

$$\Delta t = t_{pre} - t_{post} \tag{9}$$

Then the change in synaptic weights ΔW can be summarize as [18]:

$$\Delta W = \begin{cases} A_+ e^{\Delta t/\tau_+}, \ if \ t_{post} \ > \ t_{pre} \\ A_- e^{\Delta t/\tau_-}, \ if \ t_{post} \ < \ t_{pre} \end{cases} \tag{10}$$

Then difference between A_+ and A_- is the maximum of weight change, $\tau_+ \ and \ \tau_+$ as time constants that define the strength of the update, as we can see in Fig. 13 the closer Δt to 0 major weight modulation occurs, whether LTP when the $\Delta t < 0$ or LTD when $\Delta t > 0$. Also is depicted in the figure the learning windows in x axis and the maximum synapsis modulation in y axis.

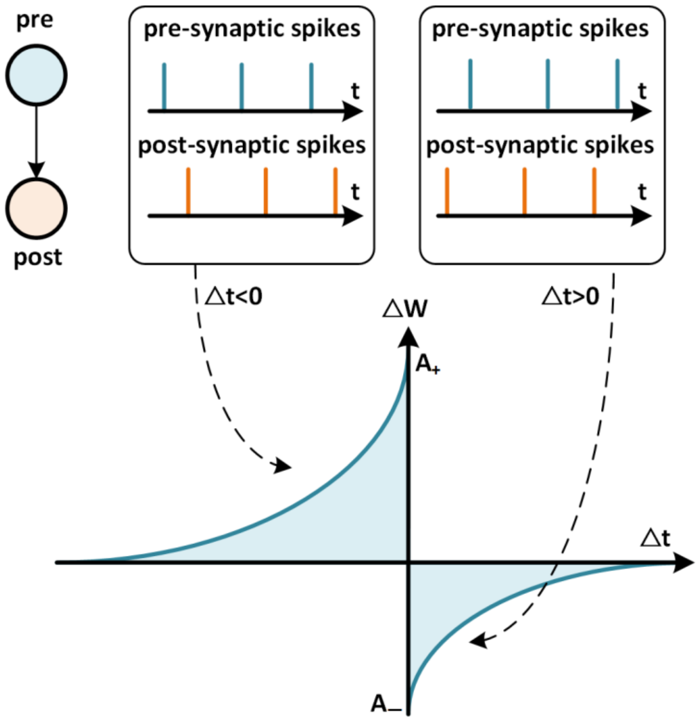


Figure 13. STDP learning windows and synapsis modulation

2.6.3.2 Supervised Learning

In supervised learning, the data is fed to the model with the output labels or categories and then the algorithm learns the pattern in order to predict the outcome. The category of prediction could be **classification**, **regression**, **time series**, etc. The learning process in ANN is accomplished when the global error signal is calculated and by gradient descent via backpropagation the weights are adjusted in order to minimize the error. In a biological context it is unlikely the neurons use some sort of global error signal for computing, considering the direction of action potential is unidirectional. The intention is not to replicate the biological neurons rather diminishing the power consumption. We encounter some problems with SNN in regards to [17]:

- **Use of non-differentiable activation functions:** it's difficult to apply backpropagation because it is a gradient-based optimization algorithm. Many SNN uses a sum of the **Dirac delta function** of a spike, and comes with the impossibility of a derivation, so approximations of derivate or substitute are used.

- **Weight transport problem**: updating weights in earlier layer of a deep neural network can affect the behavior of far-away layers, it is concerning when those early layers are trained to optimize a loss function but without considering the other far-away layers, then the network learns patterns that could be not that important for the prediction, thus reducing the overall accuracy.

In SNN we have some algorithms for supervised learning:

- **SpikeProp**: the first algorithm to train the SNN by error backpropagation, SpikeProp takes into account the spike timing with their cost function, using temporal coding with a three-layer network that can solve XOR problems. One limitation is the high computational cost due to the use of a complex model as **spike-response model**. Later on, more developed versions of SpikeProp were built like **SpikePropAd**, **RProp**, **MultiSpikeProp** etc.

- **ReSuMe (remote supervised learning):** is an adaptation of the **Widrow-Hoff** (Delta) **rule**, used in non-spiking neurons. In this rule the weight changes proportionally with the wanted outcome minus the predicted outcome [18]. This algorithm only can train for a single neuron that receives spiking for multiple pre-synaptic neurons.

- **Chronotron:** this algorithm recognizes the spike timing using a differentiable adaptation of **Victor-Purpora (VP) distance metric** to perform gradient descent. Like ReSuMe, the algorithm only can train for a single neuron.

- **SPAN (spike pattern association neuron):** this algorithm recognizes the precise timing of spikes, transforming spike trains in analog signal in order to apply the Widrow-Hoff rule for adjusting the weights and minimize the error [19].

- **BP-STDP (backpropagation-spike-timing dependent plasticity):** this learning algorithm combines backprop rules are converted into STDP rules.

More detailed definition of models will be presented in the following section.

2.6.4 Training SNN

There are various methods to training SNN, they can be mainly classified as below [17]:

- **Shadow Training**: a conversion from ANN to SNN.
- **Backpropagation using Spikes**: native training of SNN using backpropagation or variations.
- **Local learning rules**: instead of a global error signal to update the weights, local signals are used to adjust the weights.

2.6.4.1 Shadow or Indirect Supervised Training

Shadow training is one of the methods for obtaining SNN, firmly is not a way to train a SNN rather a way to obtain it via conversion from ANN, the process is done by adjusting the weights (previously adjusted by backpropagation) and the parameters of an ANN to a spike rate or latency code from a SNN [17-18].
The input and output encoding also needs to be adjusted. This approach has 2 main **advantages** [20]:

- **High accuracy**: this approach can leverage the state-of-the-art techniques used for training ANN and then converting them to a SNN, earlier in the development of this technique some modifications of the network topology needed to be modified, but recently just a simple transformation in training overhead [21]. However, the conversion is an approximation so the shadow-trained SNN rarely reaches the accuracy from original ANN.

- **Energy usage**: in [22] evaluated the power consumption of different SNN models comparing the number of operations needed archive 98% accuracy with MNIST dataset in spiking model versus a non-spiking model as we can see in Fig 14.

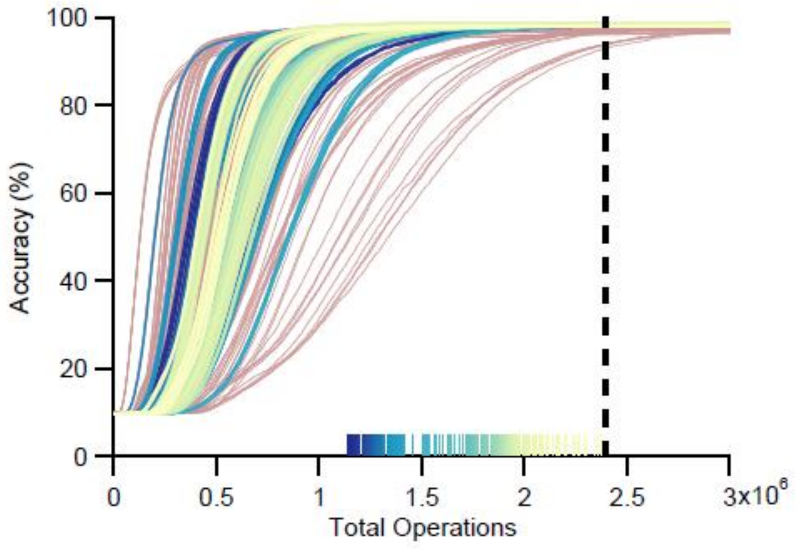


Figure 14. Operations vs accuracies in SNN and non-SNN. The vertical discontinuous line shows the steps for a non-spiking deep neural network to archive 98% accuracy in MNIST dataset, the pink curves are SNN that archived the 98% accuracy in more steps than non-SNN, instead the colored vertical short lines in the x axis represents the SNN architecture that has reach 98% of accuracy within the steps of non-SNN.

ANN to SNN also has some **disadvantages**:

- Not every ANN operation can be converted into SNN operations, as the activation function and max-pooling operations.
- Most of the benchmark do not use the temporal dynamics of a SNN, and most conversion methods use rate-coding, which requires more power than temporal coding.

This method is preferable if the inference efficiency is more relevant than training efficiency and the data is not time-varying [17]. We will be more focuses on the next sections, which is more promising and is setting the way SNN should be evolving.

2.6.4.2 Backpropagation

In SNN there are ways to enable backpropagation to minimize a loss, as stated earlier in section **2.6.3.2** we cannot use backpropagation in spikes due to the lack of derivative and therefore the inability to update the weights. Spikes also raises another issue, the **dead neuron problem**, a neuron that doesn't fire, the weights of the neuron do not change during training period and thus doesn't modify the loss, rendering the neuron useless.

2.6.4.2.1 Temporal Backpropagation Using Spike Times

As state earlier backpropagation using plain spikes are not possible, so instead of spikes we use **spikes times** and calculate the derivative, one of the first algorithms that uses this approach was **SpikeProp** as said earlier in **2.6.3.2** it is very computer intensive so better alternatives were built.

Another team developed a model called **single-spike supervised spiking neural network** (**S4NN)** [23], built for classification and based on latency learning, specifically **TTFS,** trains the network via **temporal backpropagation** and uses **non-LIF neurons** also called reset-to-zero neurons because after a spike, the membrane potential is reset to zero.
The architecture is presented in fig. 15, the input signals are converted into spike train under TTFS, in this case the aim is to classify images so the intensity of each pixel of the gray image ranging [0, I_{max}] value is encoded into a single spike time ranging [0, t_{max}] following the equation:

$$t_i = \left\lfloor \frac{I_{max} - I_i}{I_{max}} t_{max} \right\rceil \tag{11}$$

Being *i* an input neuron, t_i its firing time and I_i its intensity.

Then the spike train in the layer 0 is computed as:

$$S_i^0 = \begin{cases} 1 & if\ t = t_i \\ 0 & otherwise \end{cases} \tag{12}$$

Then there are 2 hidden layers of **non-LIF neurons**, with no limitation and finally the output layer, being each output neuron one of the desired categories to classify.

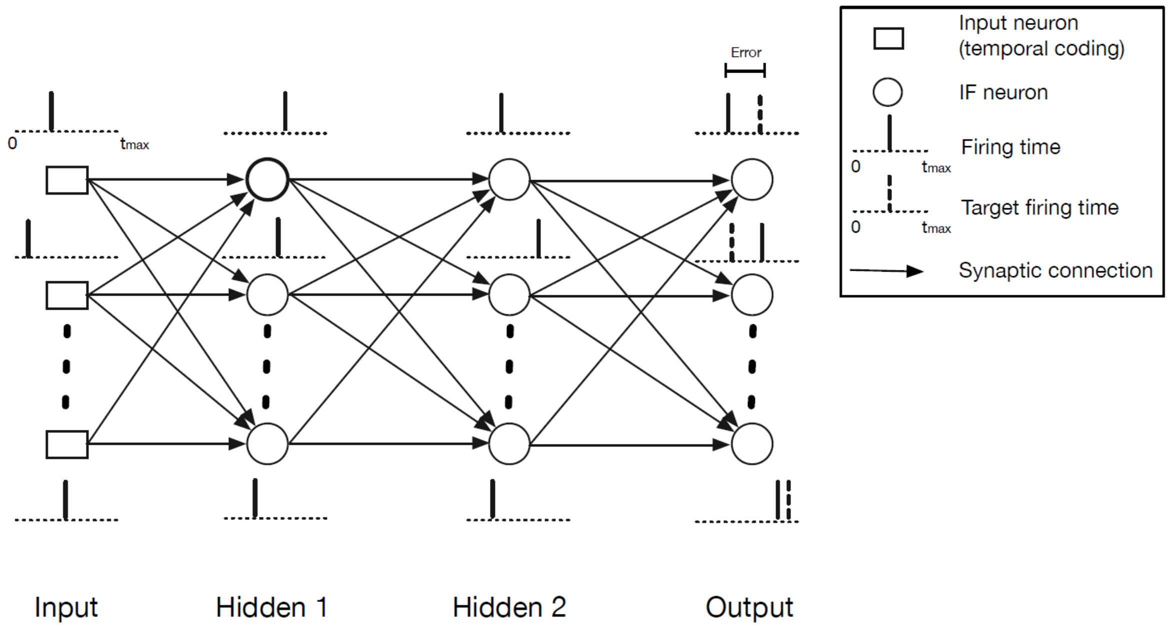


Figure 15. **Architecture of S4NN**. Input neuron receives the information encoded and emits a spike and then the firing time is calculated, the spikes activate the next hidden layers and then the output firing time has to be modified to approximate to the specific target firing time of the category by temporal backpropagation.

The training of the network is done as the typical backpropagation algorithm:

- **Forward path**: spikes from the input layer changes the membrane potential until the threshold and then spike is emitted to the subsequent neuron layers. In the training phase is mandatory for a neuron to fire even if threshold is not reached, so a "fake" spike at t_{max} in order to surpass the dead neuron problem.
- **Backward path**: the error is calculated by a temporal error function and then using temporal version of stochastic gradient descent to minimize the error by the loss and then update the weights.

Repeat until the error is reduced to a reasonable level.

Other teams were also studying on temporal backpropagation, in [24] the accuracy is similar to S4NN but in [25] there is a jump in accuracy.

2.6.4.2.2 Backpropagation by Surrogate Gradients and Smoothed Spikes

Due to the problem talked earlier about the impossibility of spikes derivatives when it is computed as a binary event, like the Heaviside function (acting as the activation function), the approach of surrogate gradient is used. The idea is applying a differentiable approximation of the step function like a sigmoid function, then the gradient is used in backpropagation to minimize the loss, in Fig. 16 we can appreciate the surrogate of the step function.

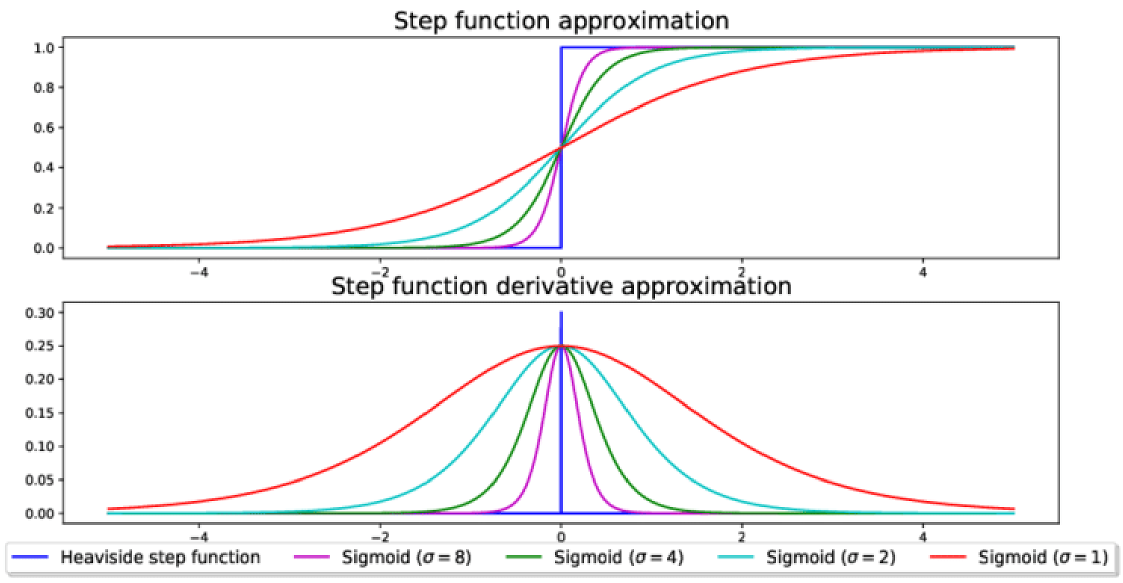


Figure 16. **Surrogate gradient**. Different sigmoid functions as surrogate of the step function.

Another approach is “smoothing” the spike in order to make it differentiable in the review [27] some model based on this strategy uses this smoothed spike using rate coding, in article [28] the team used a **modified LIF neuron** equation that smooths the threshold in spike, in this case is a case of ANN to SNN. During the forward path of a neural network, the main Heaviside function is applied. However, in the backward path, a surrogate gradient or smoothed spike is used instead.Next, some models and strategies are presented:

In **SuperSpike** [27] the temporal coding of a single timing spike is used in conjunction with a LIF neuron and non-vanishing surrogate gradient, this network does not scale well for larger multilayer networks and some local errors are introduced in order to improve the performance.

Another team [29] uses LIF neurons and smooth signal for backprop applying low-pass signals in membrane potential instead of spike, filtering the abrupt changes as noise. In some layers of a neural network, a **Winner-take-all (WTA)** strategy is employed. This involves grouping neurons together with lateral inhibition connections, so that when one neuron produces an output, the rest of the group is silenced. This allows for competition among the neurons in the group, with only the strongest neuron producing an output.

SLAYER (Spike LAYer Error Reassignment) [30] is a general method for of error backpropagation for SNN, like backpropagation the distribution of spatial credit assignment is through the layers, this method also accounts the time credit assignment using **Spike Response Model (SRM)** a neuron model that maintains an internal stat over time, making the neurons stateful and aware of the previous spike inputs. Thus, SLAYER learns both synaptic weights and axonal delays. The derivative of the spike function is represented as a **Probability Density Function (PDF)** of change states in spiking neuron.

Spatio-Temporal Backpropagation (STBP) [31] is a method considering both spatial domain (SD) and temporal domain (TD), this spatio-temporal domain (STD) allows to capture more information and the nature of the SNN. Both domains are added by an iterative LIF neuron to perform directly backpropagation training using curve approximations to the spike.

An interesting method has been proposed: **hybrid macro/micro level backpropagation (HM2-BP**) algorithm [32], which performs backpropagation on two levels: a rate coding signal on "macro" time scale in conjunction with an update on a shorter "micro" time scale capturing the temporal interactions of individual spike between two layers.

Another team [33] leverages the power or existing deep ANN architecture as VGG and ResNet to boost performance of the SNN, VGG used consecutives convolutional layers called "Spiking VGG block" and ResNet skipped connections through the network to increase training accuracy. Thus, a deep convolutional SNN with more than 10 trainable layers was assembled, reaching high accuracy of 99.59% in **MNIST** dataset.

Gradually, over the past several years direct SNN training via backpropagation has been cutting the gap in accuracy compared to the conversion approaches. One of the main benefits is the availability to be trained on a neuromorphic computer and lower power consumption.

2.6.4.3 Local Rules

Many model that does not rely on backpropagation mostly relies on STDP (described in 2.7.2.1) or some variations to update the synaptic weights. It is worth mentioning STDP scales better computationally than backpropagation, however performances are worse than backprop in competitive datasets due to the fact most methods use only 1 trainable layer. Backprop was designed for optimizing weights and STDP was mere an adaptation of a biological unsupervised learning rule [17], in addition STDP can learn spatio-temporal patterns. In this section we expose some on the previous works related in this area.

ReSuMe (remote supervised method) [34] is a supervised model based on the adjustments of the synaptic weights by the union of STDP and anti-STDP learning. Uses a similar Widrow-Hoff rule, which minimize the error signal between the target and the output. One of the limitations of this approach is the limitation of the use of 1 trainable layer, thus limiting the power of abstraction offered by multiple layers. The same team and many others improved this approach like **DL-ReSuMe** (delay remote supervised method), which adds a delay shift in the weight adjustments. **CCDS** (Cross-correlated delay shift) is a method coming from ReSuMe that takes into account both synaptic delays and axonal delays in the process of learning the synaptic weights. **T-ReSuMe** is a version of the algorithm with the addition of a triplet-based STDP instead of the typical duplet STDP. One approach adapted ReSuMe to a multilayer SNN called **Multi-ReSuMe**, from the same team of ReSuMe.

Another approach of STDP can be unsupervised [35], here the labels are not shown to the network, it's used on the MNIST dataset with of 2 layers network, first the input layer of 28 x 28 with another layer of excitatory neurons connected with inhibitory neurons, providing with lateral inhibition.

The team [36] devise an asynchronous adaptation of the random backpropagation called **event-driven random backpropagation** (eRBP), a presynaptic spike-driven plasticity rule, which is spatial and temporal, with the neuron and synapse having access to all the information needed for learning locally. The backprop weight update rule can be applied at each time step. The [37] team developed a new leaning method called DECOLLE, derived from their previous work [36]. DECOLLE uses layer-wise random initialization in local readouts, allowing local gradient computation. This way the gradient only is propagated forward comes with the advantage of the low complexity and high capacity to scale given that the model is linear with the number of neurons. This is possible due to the spatially and temporally local cost function, which are updated using surrogate gradient.

Another team [38] used an encoding method quantizing the neural activations using **Sigma-Delta modulation**, to perform an analog-to-digital conversion in order to convey signals approximately at low bit-rates, the update rules are similar to STDP.

Spiking convolutional neural networks (SCNN) also has been studied in [39-41] used on MNIST with higher accuracy on MNIST than previous STDP approaches, but still not catching with the backprop learning.

Here is one interesting approach that combines unsupervised STDP local rule and supervised gradient descent backpropagation. In [42] the team developed a SCNN: first, the network is pre-trained using STDP to initialize the parameters in the convolutional layer, letting them to self-learn features, next in the line is a pooling layer followed by a fully connected layer and output layer of LIF neurons, where the weights are fine-tuned by backprop, the method used to enable backprop is creating a pseudo-derivative filtering the low-pass post-spike trains similar to [29].

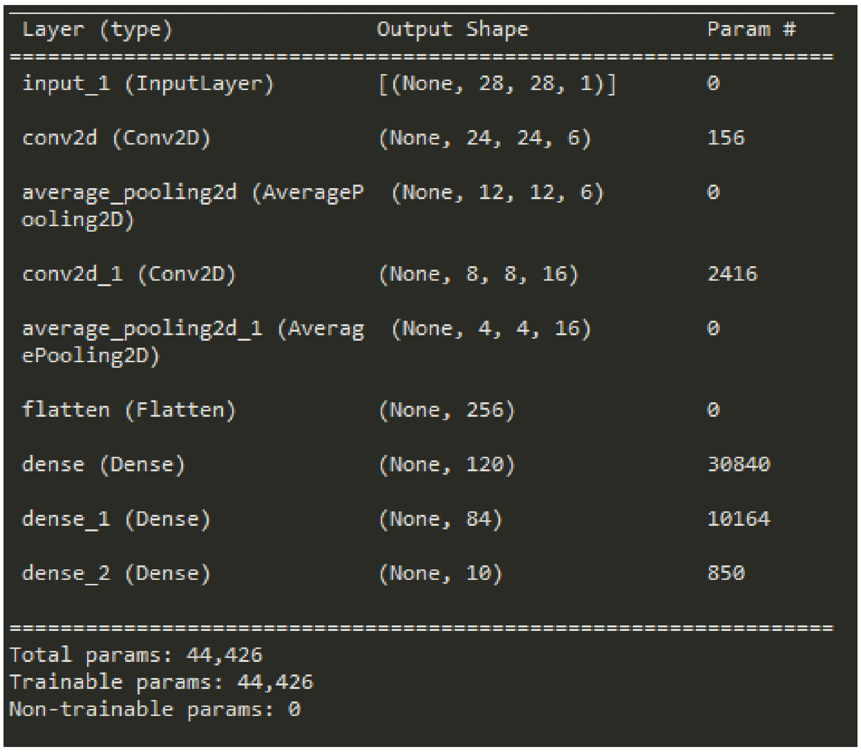
3. Implementation

Here we are going to implement some of the different training methods exposed earlier with **MNIST (Modified National Institute of Standards and Technology)** dataset [52], a collection of handwritten digits that is widely used in the field of machine learning, specifically in the domain of computer vision. It consists of 60,000 training examples and 10,000 test examples, with each example comprising a 28x28 pixel grayscale image of a digit and its corresponding label (a number from 0 to 9). This dataset is commonly referred to as a "hello world" dataset in the field of machine learning due to its ease of use, small size, and well-structured format, while still presenting sufficient challenge to evaluate the effectiveness of various machine learning models.

3.1 Shadow Training

In this section model using the ANN-to-SNN conversion methods are implemented:

The model used is LeNet, explained in paper [48] using the library **snntoolbox** [49] created by the same group to convert ANN to SNN, and following the code found in [50]. First, we train a simple ANN form MNIST using **Tensorflow** and the architecture depicted in the Fig. 17 with batch size of 128, 5 epochs, optimizer Adam and categorical crossentropy loss. The results are: loss of 0.053 and **98.38%** in accuracy.

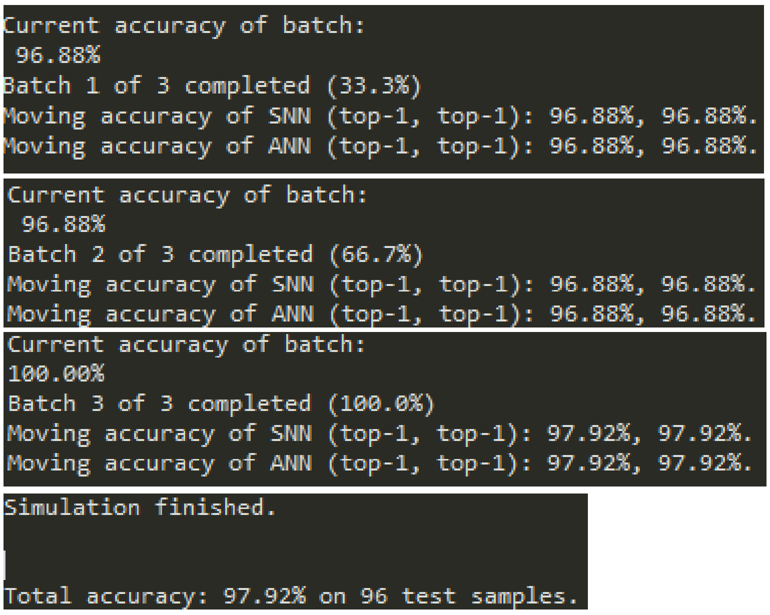


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 24, 24, 6)         156

 average_pooling2d (AverageP  (None, 12, 12, 6)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 8, 8, 16)          2416

 average_pooling2d_1 (Averag  (None, 4, 4, 16)         0
 ePooling2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 120)               30840

 dense_1 (Dense)             (None, 84)                10164

 dense_2 (Dense)             (None, 10)                850

=================================================================
Total params: 44,426
Trainable params: 44,426
Non-trainable params: 0
```

Fig. 17 CNN architecture

During the process of conversion, the weights of ANN are used and the analog (rate) neurons are replaced with Integrate-and-Fire spiking neurons.

Then the conversion is executed using 3 batches and the final accuracy is **97.92%** as shown in Fig. 17:



```
Current accuracy of batch:
 96.88%
Batch 1 of 3 completed (33.3%)
Moving accuracy of SNN (top-1, top-1): 96.88%, 96.88%.
Moving accuracy of ANN (top-1, top-1): 96.88%, 96.88%.
Current accuracy of batch:
 96.88%
Batch 2 of 3 completed (66.7%)
Moving accuracy of SNN (top-1, top-1): 96.88%, 96.88%.
Moving accuracy of ANN (top-1, top-1): 96.88%, 96.88%.
Current accuracy of batch:
100.00%
Batch 3 of 3 completed (100.0%)
Moving accuracy of SNN (top-1, top-1): 97.92%, 97.92%.
Moving accuracy of ANN (top-1, top-1): 97.92%, 97.92%.
Simulation finished.

Total accuracy: 97.92% on 96 test samples.
```

Fig. 18. Accuracy of the converted SNN

3.2 Backpropagation

The method chose is form the paper [45] and the code [51] is from the same team. This is one of the best models aside from [43] and [47] which do not have coding implementation. **Pytorch**, another framework for deep learning is used, with **CUDA** implementing the GPU.

The MNIST dataset is trained following the configuration from [45] show in Table 1:

Network Size	Time Steps	Epochs
15C5-P2-40C5-P2-300	5	100

Table 1. **SNN architecture configurations**. 15C5: convolution layer with 15 of the 5 x 5 filters. P2: pooling layer with 2 x 2 filters. 40C5: convolution layer with 40 of the 5 x 5 filters and then fully connected layer of 300.

After running 100 epochs and batch size of 50 we accomplish a result of **99c%** accuracy in the best epoch, very close to the **99,53%** reported form the paper [45].

4. Results

Here we expose a summary table with the training approaches, type of network and the accuracies for the datasets MNIST, N-MNIST and CIFAR-10 In table 2.

				accuracy	
Model	Architecture	Learning Method	MNIST	N-MNIST	CIFAR-10
Hunsberger, 2015 [28]	SCNN	Conversion	98,37		82,95
Masquelier, 2020 [23]	SNN	Temporal Backprop	97,4		
Mostafa, 2017 [24]	SNN	Temporal Backprop	97,55		
Liu, 2017 [25]	SNN	Temporal Backprop	99,1		
Wang, 2019 [44]	SNN	Temporal Backprop	99,17		
Wu, 2018 [31]	SNN	Spatio-temporal Backprop	98,89	98,78	
Yingyezhe, 2018 [32]	SNN	HM2-backprop	98,93	98,84	
Lee, 2016 [29]	SCNN	Backprop	99,31	98,74	
Shrestha, 2018 [30]	SCNN	Backprop	99,36	99,2	
Wu, 2018 [31]	SCNN	Spatio-temporal Backprop	99,42		
Jin, 2018 [32]	SCNN	HM2-backprop	99,49		
Lee, 2020 [33]	deep SCNN	Backpropr	99,59	99,09	90,95
Zhang, 2020 [45]	SCNN	Backprop	99,53	99,4	91,41
Zhao, 2021 [47]	SCNN	Backprop	99,67	99,57	90,93
Shen, 2022 [43]	SCNN	Backprop	99,67	99,71	94,51
Diehl and Cook, 2015 [35]	SNN	Unsupervised STDP	95		
Neftci, 2017 [36]	SNN	Random Backprop	97,98		
O'Connor, 2017 [38]	SNN	STDP-like	98,36		
Tavanaei, 2018 [46]	SNN	STDP-based backprop	97,2		
Tavanaei, 2016 [39]	SCNN	STDP	98,36		
Kheradpisheh, 2017 [40]	SCNN	STDP	98,4		
Tavanaei, 2018 [41]	SCNN	STDP	98,6		
Lee , 2018 [42]	SCNN	STDP + Backprop	99,28		
Kaiser, 2020 [37]	SCNN	Synthetic gradient		99,04	

Table 2. SNN model accuracy.

No many methods for the ANN-SNN conversion were shown due to the fact the accuracy of SNN is always dependent of the origin ANN, which has new methods still being developed, also the comparation should be in terms of the ANN to SNN loss during conversion. In the implementation of a conversion method, we saw the drop in accuracy when the network is transformed to SNN.

Temporal backpropagation has reach high as 99,17 [44] but not enough to surpass the models from backpropagation using some surrogate gradient with good performances on [33, 43, 45, 47] with various datasets. The accuracy drops when it comes to de CIFAR-10 datasets, due to its relative difficulty compared to de MNIST or N-MNIST (the neuromorphic version). Even the most promising approach of backpropagation do not reach ANN state-of-the-art accuracies in the aforementioned datasets , but slowly this approach is catching up and it looks promising in the long term.

In the STDP approach we saw first in [35] began to create the stepping stones for better accuracies, starting from 95% and almost reaching to 99% the pure STDP methods.

The code for the implementation is facilitated by the authors and is shown in table 3 from various papers from previous table 2.

Hunsberger, 2015 [28]	https://pypi.org/project/n3ml-python/
Masquelier, 2020 [23]	https://github.com/SRKH/S4NN
Mostafa, 2017 [24]	https://github.com/HanSeokhyeon/Supervised-Learning-Based-on-Temporal-Coding-in-Spiking-Neural-Networks
Wu, 2018 [31]	https://github.com/yjwu17/STBP-for-training-SpikingNN
Yingyezhe, 2018 [32]	https://github.com/jinyyy666/mm-bp-snn
Shrestha, 2018 [30]	https://github.com/bamsumit/slayerPytorch
Wu, 2018 [31]	https://github.com/yjwu17/STBP-for-training-SpikingNN
Jin, 2018 [32]	https://github.com/jinyyy666/mm-bp-snn
Lee, 2020 [33]	https://github.com/chan8972/Enabling_Spikebased_Backpropagation
Zhang, 2020 [45]	https://github.com/stonezwr/TSSL-BP
Diehl and Cook, 2015 [35]	https://github.com/zxzhijia/Brian2STDPMNIST
Neftci, 2017 [36]	https://pypi.org/project/n3ml-python/
Tavanaei, 2018 [46]	https://github.com/petered/pdnn
Kaiser, 2020 [37]	https://github.com/nmi-lab/decolle-public

Table 3. Implementation code for SNN.

5. Conclusions and future work

Deep learning algorithms that use traditional ANNs with continuous activation values and backpropagation training are currently the state-of-the-art. However, these models have high energy requirements and are limited by the bottlenecks of current computer architectures. As a result, researchers are exploring alternative approaches such as SNNs that use local rules like STDP and backpropagation surrogates. The goal of these efforts is to use neuromorphic computing to reduce energy expenditure and train more efficient models.

The results were expected, it is still a long road for neuromorphic model and hardware to unseat the current state-of-the-art ANNs, but it is an area that is worth researching.

One limitation of this work was the lack of time organization and delays caused by external factors. Additionally, the limited experience in finding and using relevant information to better understand the field has also delay it. The objectives of the work may have changed over time as new information and ideas were encountered and broader the knowledge is presented. Gaining more knowledge can be both beneficial and challenging. On one hand, it can provide a deeper understanding of the subject. On the other hand, it can also lead to uncertainty about the original objectives and can require time to reevaluate and adjust plans accordingly. The proposed changes varied in terms of their ambition and feasibility, some of them may have been more ambitious and required more resources and time to implement and were discarded to the constrains of the available time and knowledge.

Future work in this area could involve a more detailed exploration of the architecture and operation of neuromorphic computers, as well as a comparison of different SNN models in terms of energy usage, accuracy, and other parameters.

6. Bibliography

1. Andrew Tch 2017, The mostly complete chart of Neural Networks, explained, accessed 9 October 2022, https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

2. Jumper, J, Evans, R, Pritzel, A, et al. Applying and improving AlphaFold at CASP14. Proteins. 2021; 89(12): 1711- 1721

3. "File:Von Neumann Architecture.svg." Wikimedia Commons, the free media repository. 21 Aug 2022, 15:57 UTC. 27 Oct 2022, 15:09 <https://commons.wikimedia.org/w/index.php?title=File:Von_Neumann_Architecture.svg&oldid=684090935>.

4. Jeong, D. S., Kim, K. M., Kim, S., Choi, B. J., Hwang, C. S. (2016). Memristors for Energy-Efficient New Computing Paradigms. Adv. Electron. Mater., 2: 1600090. doi: 10.1002/aelm.201600090

5. Strubell, Emma & Ganesh, Ananya & Mccallum, Andrew. (2019). Energy and Policy Considerations for Deep Learning in NLP. 3645-3650. 10.18653/v1/P19-1355.

6. Stathis Kamperis 2019, Energy considerations for training deep neural networks, accessed 27 October 2022, https://ekamperi.github.io/machine%20learning/2019/08/14/energy-considerations-dnn.html

7. Zubair Hussain Khan 2016, “File: Machine+2BLearning.png”, Difference between Artificial Intelligence, Machine Learning, and Deep Learning, accessed 29 October 2022 , https://www.codesofinterest.com/2016/11/difference-artificial-intelligence-machine-learning-deep-learning.html

8. Khan Academy. The synapse, accessed 17:33 UTC. 27 Oct 2022, 15:07 <https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/the-synapse>.

9. Kainspraveen 2020, Artificial Neural Networks And its Intuition. accessed 3 November 2022, https://medium.com/analytics-vidhya/artificial-neural-networks-and-its-intuition-3ab10e0ca452

10. Ren, Zheng & Kong, Yunhui & Ai, Ling & Xiao, Hui & Wang, Wei & Shi, Zhi & Zhu, Liqiang. (2022). Proton Gated Oxide Neuromorphic Transistors with Bionic Vision Enhancement and Information Decoding. Journal of Materials Chemistry C. 10. 10.1039/D2TC00775D.

11. Sengupta B, Laughlin SB, Niven JE (2014) Consequences of Converting Graded to Action Potentials upon Neural Information Coding and Energy Efficiency. PLOS Computational Biology 10(1): e1003439.

12. Olshausen, Bruno & Field, David. (2004). What is the other 85% of V1 doing? Prob. Syst. Neurosci. 4.

13. Gerstner, W., &amp; Kistler, W. (2002). Spiking neuron models: An introduction. Cambridge University Press.

14. Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. Nature, 381:520-522.

15. Izhikevich EM. Simple model of spiking neurons. IEEE Trans Neural Netw. 2003;14(6):1569-72. doi: 10.1109/TNN.2003.820440. PMID: 18244602.

16. Lu S and Xu F (2022) Linear leaky-integrate-and-fire neuron model based spiking neural networks and its mapping relationship to deep neural networks. Front. Neurosci. 16:857513. doi: 10.3389/fnins.2022.857513

17. Eshraghian, Jason K., et al. "Training spiking neural networks using lessons from deep learning." arXiv preprint arXiv:2109.12894 (2021).

18. Tavanaei, Amirhossein, et al. "Deep learning in spiking neural networks." Neural networks 111 (2019): 47-63.

19. Mohemmed, Ammar & Schliebs, Stefan & Matsuda, Satoshi & Kasabov, Nikola. (2012). SPAN: Spike Pattern Association Neuron for learning spatio-temporal spike patterns. International journal of neural systems. 22. 1250012. 10.1142/S0129065712500128.

20. Nguyen D-A, Tran X-T, Iacopi F. A Review of Algorithms and Hardware Implementations for Spiking Neural Networks. Journal of Low Power Electronics and Applications. 2021; 11(2):23. https://doi.org/10.3390/jlpea11020023

21. Diehl, P.U.; Neil, D.; Binas, J.; Cook, M.; Liu, S.; Pfeiffer, M. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–17 July 2015; pp. 1–8

22. Neil, Dan & Pfeiffer, Michael & Liu, Shih-Chii. (2016). Learning to be Efficient: Algorithms for Training Low-Latency, Low-Compute Deep Spiking Neural Networks. 10.1145/2851613.2851724.

23. Saeed Reza Kheradpisheh and Timothée Masquelier, Temporal backpropagation for spiking neural networks, International Journal of Neural Systems, Vol. 30, No. 6 (2020) 2050027

24. H. Mostafa, “Supervised learning based on temporal coding in spiking neural networks,” IEEE transactions on neural networks and learning systems, pp. 1–9, 2017.

25. T. Liu, Z. Liu, F. Lin, Y. Jin, G. Quan, and W. Wen, “Mt-spike: a multilayer time-based spiking neuromorphic architecture with temporal error backpropagation,” in Proceedings of the 36th International Conference on Computer-Aided Design. IEEE Press, 2017, pp. 450–457.

26. Rasteh, Ali & Delpech, Florian & Aguilar-Melchor, Carlos & Zimmer, Romain & Shouraki, Saeed & Masquelier, Timothée. (2021). Encrypted Internet traffic classification using a supervised Spiking Neural Network.

27. E. O. Neftci, H. Mostafa and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," in IEEE Signal Processing Magazine, vol. 36, no. 6, pp. 51-63, Nov. 2019, doi: 10.1109/MSP.2019.2931595.

28. E. Hunsberger and C. Eliasmith, “Spiking deep networks with lif neurons,” arXiv preprint arXiv:1510.08829, 2015.

29. J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” Frontiers in Neuroscience, vol. 10, 2016.

30. S. B. Shrestha and G. Orchard, “Slayer: Spike layer error reassignment in time,” arXiv preprint arXiv:1810.08646, 2018.

31. Wu, Y., L. Deng, G. Li, J. Zhu, and L. Shi, Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks, Frontiers in neuroscience, 12, 331 (2018).

32. Jin, Yingyezhe & Li, Peng & Zhang, Wenrui. (2018). Hybrid Macro/Micro Level Backpropagation for Training Deep Spiking Neural Networks.

33. Lee C, Sarwar SS, Panda P, Srinivasan G and Roy K (2020). Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures. Front. Neurosci. 14:119. doi: 10.3389/fnins.2020.00119
34. Wang, Xiangwen & Lin, Xianghong & Dang, Xiaochao. (2020). Supervised learning in spiking neural networks: A review of algorithms and evaluations. Neural Networks. 125. 258-280. 10.1016/j.neunet.2020.02.011.

35. P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” Frontiers in Computational Neuroscience, vol. 9, pp. 1–9, 2015.

36. E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, “Event-driven random back-propagation: Enabling neuromorphic deep learning machines,” Frontiers in Neuroscience, vol. 11, p. 324, 2017.

37. Kaiser J, Mostafa H. and Neftci E. (2020). Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). Front. Neurosci. 14:424. doi: 10.3389/fnins.2020.00424

38. O'Connor, Peter et al. “Temporally Efficient Deep Learning with Spikes.” ArXiv abs/1706.04159 (2017).

39. A. Tavanaei and A. S. Maida, “Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning,” arXiv preprint arXiv:1611.03000, pp. 1–16, 2016.

40. Kheradpisheh, S.R., Ganjtabesh, M., Thorpe, S.J., Masquelier, T., STDP-based spiking deep convolutional neural networks for object recognition. Neural Networks (2017).

41. A. Tavanaei, Z. Kirby, and A. S. Maida, “Training spiking ConvNets by STDP and gradient descent,” in Neural Networks (IJCNN), The 2018 International Joint Conference on. IEEE, 2018, pp. 1–8.

42. Lee, Chankyu & Panda, Priyadarshini & Srinivasan, Gopalakrishnan & Roy, Kaushik. (2018). Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning. Frontiers in Neuroscience. 12. 435. 10.3389/fnins.2018.00435.

43. Shen, Guobin & Dongcheng, Zhao & Zeng, Yi. (2022). Backpropagation with Biologically Plausible Spatio-Temporal Adjustment for Training Deep Spiking Neural Networks.

44. Wang X, Lin X and Dang X (2019) A Delay Learning Algorithm Based on Spike Train Kernels for Spiking Neurons. Front. Neurosci. 13:252. doi: 10.3389/fnins.2019.00252

45. Zhang, Wenrui & Li, Peng. (2020). Temporal Spike Sequence Learning via Backpropagation for Deep Spiking Neural Networks.

46. Tavanaei, Amir & Maida, Anthony. (2018). BP-STDP: Approximating Backpropagation using Spike Timing Dependent Plasticity. Neurocomputing. 10.1016/j.neucom.2018.11.014.

47. Dongcheng, Zhao & Zeng, Yi & Li, Yang. (2021). BackEISNN: A Deep Spiking Neural Network with Adaptive Self-Feedback and Balanced Excitatory-Inhibitory Neurons.

48. Rueckauer, B. and Hu, Y. and Lungu, I.A. and Pfeiffer, M. and Liu, S.-C. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, Front. Neurosci., 2017, doi: 10.3389/fnins.2017.00682

49. Bodo Rueckauer, 2020. Spiking neural network conversion toolbox, accessed 8 January 2023, https://snntoolbox.readthedocs.io/en/latest/

50. Neuromorphic Computing Class, 2020, accessed 8 January 2023, https://github.com/iCAS-Lab/neuro_comp_class

51. Wenrui Zhang, 2021, accessed 8 January 2023, https://github.com/stonezwr/TSSL-BP

52. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.