Training neuromorphic neural networks for speech recognition and Bayesian optimisation of network hyperparameters

Rihards Klotiņš

Supervisor: Dorian Florescu

May 4, 2025

Abstract

Filler abstract - Spiking Neural Networks (SNNs), the third generation of neural networks, offer a promising alternative to traditional Artificial Neural Networks (ANNs) due to their energy efficiency, temporal processing capabilities, and potential for neuromorphic hardware implementation. Unlike ANNs, which rely on continuous-valued activations, SNNs process information through discrete spike events, closely mimicking biological neural systems. This characteristic enables them to efficiently handle sequential data, making them suitable for applications such as speech recognition, event-based vision, and edge computing. Training SNNs, however, remains a significant challenge due to the non-differentiability of spike events. While ANN-to-SNN conversion provides a workaround, it imposes computational costs and limits architectural flexibility. Direct training methods, such as Backpropagation-Through-Time (BPTT) with surrogate gradients, local learning rules, and biologically inspired approaches like e-prop and EventProp, have emerged as viable alternatives. Each method presents trade-offs in terms of computational complexity, biological plausibility, and performance. Notably, EventProp has demonstrated state-of-the-art results while reducing memory and computational overhead compared to BPTT. This work explores the theoretical advantages of SNNs, their energy-efficient processing, and recent advancements in training methodologies. It also highlights their potential impact on low-power computing applications, particularly in audio processing, where temporal encoding is crucial. By addressing current limitations and leveraging novel training strategies, SNNs could play a pivotal role in next-generation AI systems.

1 Introduction

Artificial Neural Networks (ANN) are computational models inspired by the brain; made up of layers of interconnected "neurons" which can learn patterns when given large amounts of data. After learning patterns, ANNs can be used to make inferences, predicting an output based on some given input. ANNs have long been used for application ranging from computer vision to financial forecasting. However the use of ANNs has surged recently, increasing fourfold since 2017, driven by the development of generative AI models like ChatGPT [1]. To compound this, models are growing in size and becoming more complex, consequently the global AI power consumption is increasing at an exponential rate [2]. Not only does this make models expensive to operate, but it also raises sustainability concerns.

A graphics processing unit (GPU) running a large language model (LLM) consumes hundreds of watts of power. On the other hand, the human brain processes sensory information, keeps involuntary biological systems functioning, runs its own language model, and enables conscious thought — all while using only about 20 watts. Such efficient information processing has motivated research into neuromorphic computing - a field aiming to emulate the brain's neural architecture. Spiking Neural Networks (SNNs) are considered the third generation of neural network models [3], where ANNs are considered the second generation. Compared to neurons in ANNs, SNN neurons more closely model how biological neurons behave in the brain. For a second generation neuron, the output is a weighted sum of the inputs passed through an activation function (see Figure @ref(fig:second_vs_third_gen) (a)). For a third

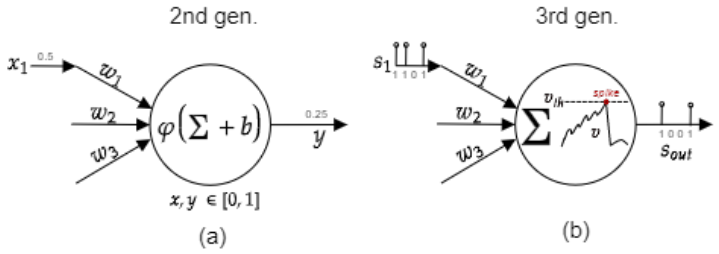


Figure 1: (a) ANN neuron (b) SNN neuron

generation neuron, the output is a spike, sometimes referred to as an event, which occurs when the voltage of a neuron surpasses a threshold (Figure @ref(fig:second_vs_third_gen) (b)), this results in neurons communicating with each other via series of spikes, which vary in timing and frequency. Since information is encoded in the timing of outputs, SNNs can inherently capture temporal patterns and dynamic behaviours in sequential data, making them inherently capable at processing tasks involving time-dependent signals, such as speech recognition, sensory processing, and event-based data streams. Moreover, it has been shown that SNNs theoretically posses higher computational power than the previous generation of ANNs [4]. Spiking neurons only activate when necessary, and don't require a clock signal, enabling massive power savings. For instance, the leading neuromorphic platform TrueNorth is capable of simulating a million spiking neurons in real-time while consuming 63 mW. The equivalent network executed on a high-performance computing platform was 100–200× slower than real-time and consumed 100,000 to 300,000× more energy per synaptic event [5]. The power savings of SNNs could be game-changing in edge computing devices that run on battery or have limited power budgets - like smartphones or remote environmental sensors. On top of this, their low-latency could benefit autonomous vehicles and robotics.

1.1 Problem to Solve

Spiking neural networks are inherently time dependent since information is encoded in the timing of spikes. This makes them naturally capable of being applied to temporal tasks - tasks where the data evolves with time. The problem with using ANNs for temporal data is that they require a fixed input size. Though they can be tweaked to deal with unspecified length temporal data using a paradigm called "recurrence", this comes with it's downsides, such as expensive training and large memory use [citation needed]. Temporal data can come in many forms, for instance video, audio, or even stock market information. The efficiency and potential effectiveness of SNNs to process temporal information could be game changing in edge devices such as mobile phones, wearable devices, and remote sensors which have small power budgets. Large language models are revolutionising how we interact with computers, they provide a way humans to interface with machines using natural language. As a result, companies are eagerly integrating LLMs into their products and services - e.g. Siri, Raybans [citation needed]. Accurate speech recognition is therefore posing to be a highly relevant application of SNNs, playing into their strengths of temporal processing and energy efficiency.

1.2 Spiking Heidelberg Digits Dataset

The Spiking Heidelberg Digits (SHD) dataset [citation needed] is a prominent spiking neural network benchmark. The wide use of SHD makes it good for fairly comparing different methods of training models. It is based on the Heidelberg Digits dataset, which is a collection of 10,000 high-quality recordings of spoken digits (0 to 9) in English and German. It is spoken by a relatively representative group of 6 males and 6 females of the ages 21 to 56 years old, with a mean age of 29. The HD dataset was converted into spike trains using a biologically realistic model of

the human cochlea, outputting 700 channels of spikes representing different frequency bands picked up by the ear.

[How are spikes encoded. Temporal or rate?] [Compare against other speech data sets.]

2 Spiking Neuron Model

In computational neuroscience, several neuron models have been developed to simulate how neurons behave, each offering a different trade-off between biological detail and computational efficiency. This section introduces three widely used models: the Hodgkin–Huxley model, the Leaky Integrate-and-Fire (LIF) model, and the Izhikevich model.

We will begin with the most influential model in neuroscience, the Hodgkin–Huxley model. Introduced in 1952 and awarded the Nobel Prize in Physiology or Medicine in 1963, this model provides a detailed mathematical description of how electrical signals—action potentials—are generated and propagated in neurons. While this model is highly accurate and biologically grounded, its complexity makes it computationally demanding, which limits its use in large-scale or real-time simulations.

Next we will discuss the Leaky Integrate-and-Fire model. To improve efficiency, the Leaky Integrate-and-Fire model simplifies neuron behaviour while retaining essential features. It treats the neuron like an electrical circuit that accumulates incoming signals over time. When the membrane voltage crosses a set threshold, the neuron emits a spike and resets. Though simplified, this model captures key spiking behavior and is highly efficient to compute, making it the most commonly used neuron model in neuromorphic engineering and machine learning applications.

Finally, we will discuss the Izhikevich model which offers a compromise between realism and efficiency. Using just two simple differential equations, it can reproduce a wide range of spiking and bursting patterns seen in real neurons. This makes it well suited for simulating la ## Hodgkin–Huxley

The Hodgkin-Huxley is a detailed neuron model developed in 1952 by Alan Hodgkin and Andrew Huxley, was the first to quantitatively explain how neurons generate and propagate electrical impulses by describing the flow of sodium and potassium ions through voltage-gated channels. Its success in reproducing the full waveform of electrical impulses — including rapid depolarization, repolarization, and refractory behaviour — earned Hodgkin and Huxley the 1963 Nobel Prize in Physiology or Medicine. Due to the robustness and biophysical accuracy of this model it is a logical model to consider when looking to implement biologically inspired learning computationally.

A neuron is enclosed in a cell membrane, with gates for ion flow. We will consider positively charged potassium (K+) and sodium (Na+) ions as they are considered as the primary contributors to the electrical behaviour of neurons. The neuron membrane has channels which allow ions to flow between the inside and the outside of the neuron; these channels are selective, allowing only specific ions to pass through, e.g. only allowing Na+ to flow. The amount of flow they allow depends on the membrane voltage Vm, which is the measure of voltage difference between the inside and outside of the neuron. Two factors govern the tendency of ions to flow from one area to the other - i.e. from inside to outside - diffusion and electric charge. Diffusion is the tendency of ions to flow from areas of high concentrations to low concentrations; in figure @ref(fig:membrane_diagram) Na+ ions will tend to flow to the extracellular side of the membrane as the concentration of Na+ ions is lower there. Particles which have the same electric charge repel, particles with opposite electrical charges attract. Therefore, Na+ ions will be attracted towards the intracellular side of the membrane due to the inside of the neuron having a negative electric charge. The directions of these opposing forces are displayed in figure @ref(fig:membrane_diagram). Naturally the cell reaches an equilibrium where the diffusion and electric charge forces are equal. When an external current is applied to the system, the equilibrium is lost and the system experiences transient behaviour defined by the mathematics of the Hodgkin–Huxley model.

The cell membrane acts as a dielectric separating two charged mediums, in other words it acts as a capacitor. According to the capacitor current-voltage equation, we know that $C\frac{dV}{dt} = I$. Applying this to the context of the neuron membrane, we get equation (8). Where V_m is the potential difference across the cell membrane, C_m is a constant representing the characteristic capacitance of the cell membrane, I_{Na} and I_K are the net charge flows of sodium and potassium ions respectively, I_L is the leakage current caused by the movement of other ions, and I_{ext}

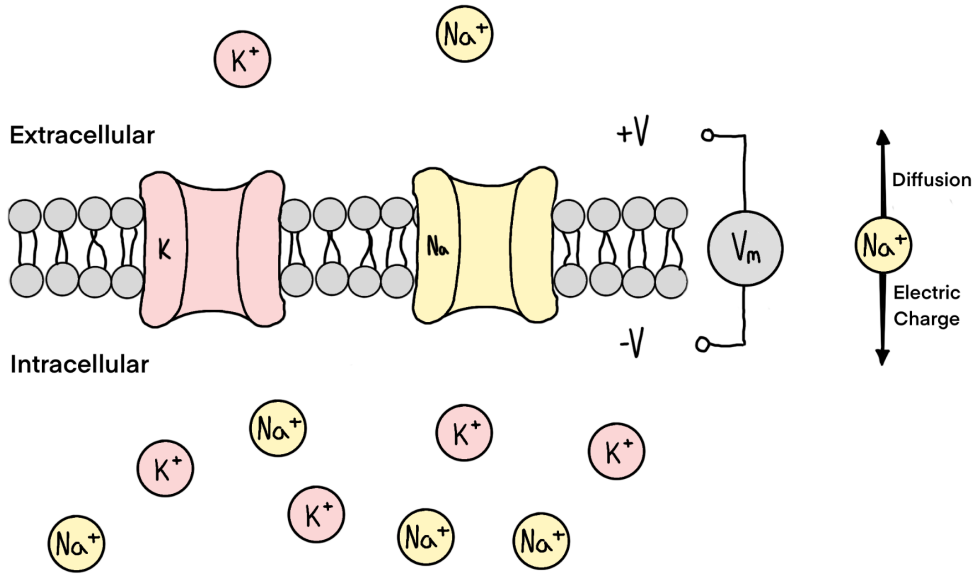


Figure 2: Diagram of neuron membrane

is the external current applied to the neuron. So we see that the sum of currents causes the voltage to change.

$$C_m \frac{dV_m}{dt} = -(I_{Na} + I_K + I_L - I_{ext}) \tag{1}$$

The currents - I_{Na}, I_K, and I_L - are determined by equations (2), (3) and (4). They can be examined like the classic $I = V/R = Vg$ equation (g being conductance). Where the left hand side is the current, the part in the brackets is the voltage, and the part outside the brackets is the conductance of the gates which the ions flow through.

$$I_{Na} = \bar{g}_{Na} m^3 h (Vm - E_{Na}) \tag{2}$$

$$I_K = \bar{g}_K n^4 (Vm - E_K) \tag{3}$$

$$I_L = \bar{g}_L (Vm - E_L) \tag{4}$$

As discussed before, when there is an imbalance in the force of *electric attraction* and *diffusion*, there is a *net flow of ions*, i.e. a current. This represented in equation (2) by the term $(Vm - E_{Na})$. Where E_{Na} is the equilibrium potential due to diffusion forces. When the two variables are in balance their difference is 0, thus the current is also zero. The same also applies for equations (3) and (4).

As per (2) the current also depends on the *conductance* of the channels which the ions move through. There are specialised channels for Na+ ions and for K+ ions, which have different conductances. The conductance of sodium channels is given by $\bar{g}_{Na} m^3 h$. The $\bar{g}_{Na}$ term is the conductance of the channel when it is fully open, i.e. the channels maximum conductance. The $m^3 h$ term is the probability - 0 to 1 - that a sodium channel is open. This probability comes from the fact that each channel has 3 "m" gates and 1 "h" gate, ions can only flow through the channel when all gates are open. Due to the large number of channels in a neuron, the value of the probability will correspond to

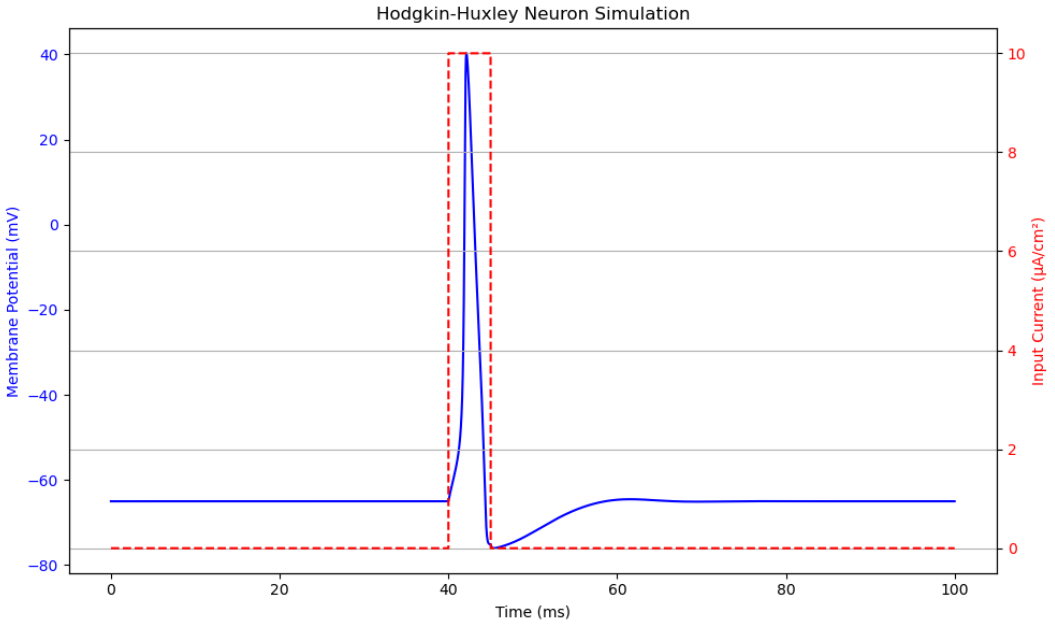


Figure 3: Response of the HH model to a 5ms step function

what proportion of the channels are open. The probabilities of $m,\ h,\ n$ are given by equation (5), where α_x and β_x are rate constants at which speed the gates open and close respectively, they are voltage dependent.

$$\frac{dx}{dt} = \alpha_x(V_m)(1-x) - \beta_x(V_m)(x),\ \ where\ x \in \{m, h, n\} \tag{5}$$

Such simple first order ion flow equations characterise to high accuracy how the neurons in our brain function. Figures @ref(fig:response_of_hh_1), @ref(fig:response_of_hh_15), and @ref(fig:response_of_hh_2) show how a neuron would react to a 5ms, 15ms, and 20ms pulse of current. While the HH neuron model has propelled our understanding of neurons due to it's biophysical precision, it's complexity - with its nonlinear differential equations and multiple gating variables - makes it computationally expensive to simulate. As a result researchers had to look elsewhere to find a model that could be used for training machines to think like humans.

2.1 Leaky Integrate-and-Fire Neuron

The leaky integrate-and-fire (LIF) neuron model is the most widely used neuronal model in SNNs due to its simplicity and efficiency which helps it scale for large networks. Simply put, the neuron receives spikes from "pre-synaptic" neurons (figure **??** (b)) - these are neurons which feed into the neuron in question; these spikes increase the "membrane potential" of the neuron - which is the voltage between the inside and outside of the neuron. When the membrane potential reaches a threshold voltage, the neuron in question fires a spike to the neurons that it is connected to, and it's membrane voltage resets to a baseline value.

In order to be modelled computationally, this behaviour is expressed in mathematical terms. Spikes received by a neuron at time t induce a current $X[t]$. This input current increases the membrane potential, $V[t]$, which is the voltage between the inside of the neuron and the outside. The voltage slowly decays over time, the speed of the decay depends on the membrane time constant, τ. This is the "Leaky Integrate" aspect of the neuron; $X[t]$ is integrated to give $V[t]$ at the same time as $V[t]$ leaks voltage. The neuron leaks voltage till it reaches it's baseline voltage level, V_{reset}. This behaviour can be seen in equation (6), $H[t]$ is equal equation (6)to $V[t]$, unless a spike occurs during t. This spiking logic is defined in equation (7), where $\Theta(x)$ is a function that is 0 unless $x \geq 0$. In other words this function becomes 1 only when the threshold voltage V_{th} is reached. When the threshold voltage is reached, $V[t]$ is set to V_{reset} and a spike is released; if the threshold voltage is not reached then $V[t] = H[t]$. The membrane potential behaviour at spike time is defined in equation (8).

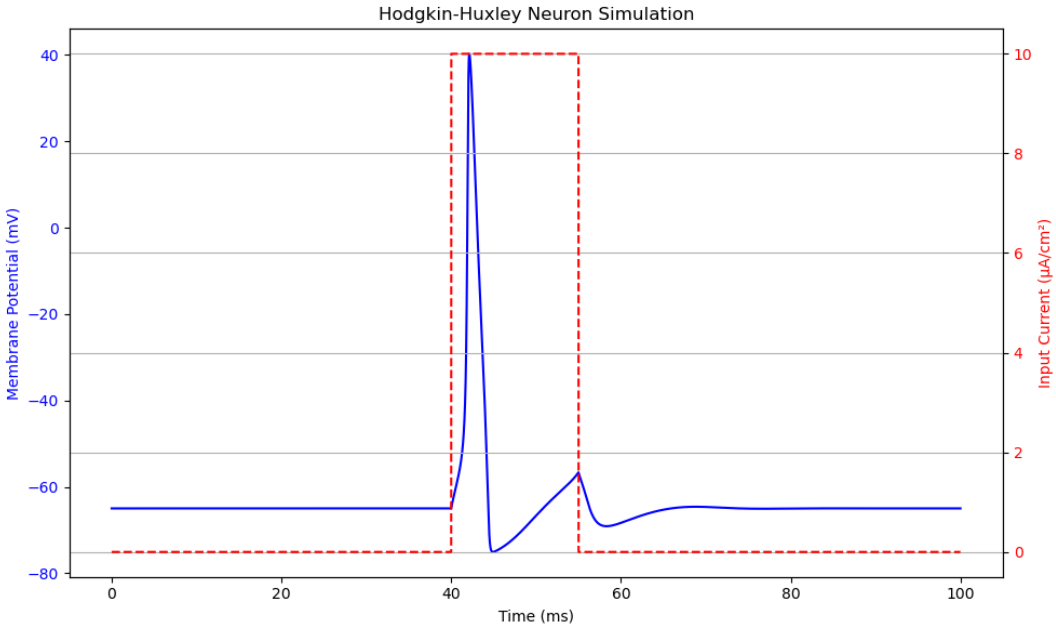


Figure 4: Response of the HH model to a 10ms step function

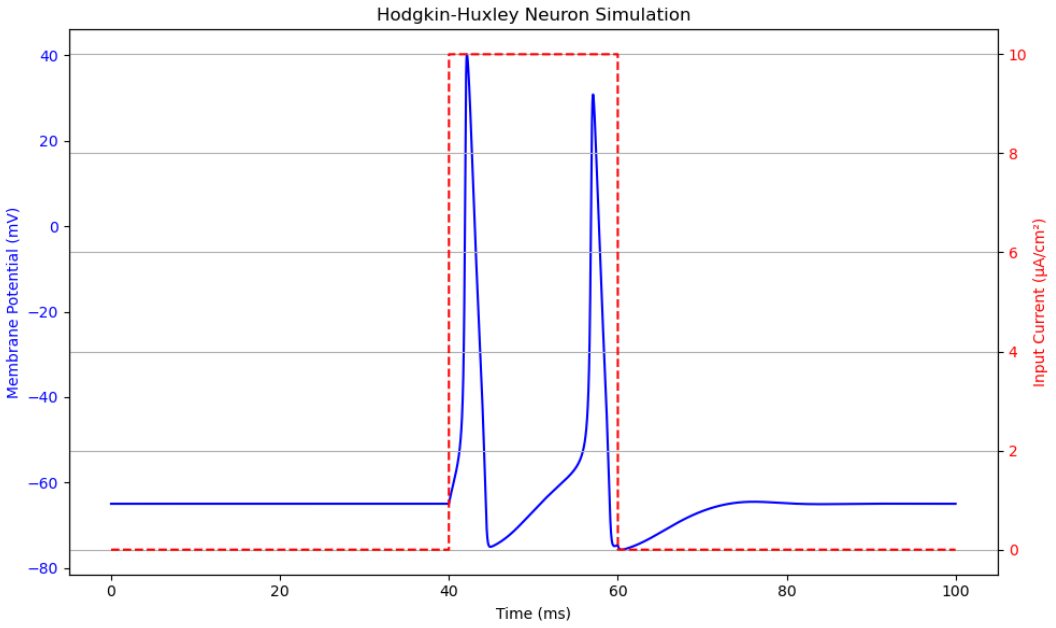


Figure 5: Response of the HH model to a 15ms step function

$$H[t] = V[t-1] + \frac{1}{\tau}(X[t] - (V[t-1] - V_{reset})), \tag{6}$$

$$S[t] = \Theta(H[t] - V_{th}), \tag{7}$$

$$V[t] = H[t](1 - S[t]) + V_{reset}S[t] \tag{8}$$

2.2 Izhikevich

Building on the simplicity of the leaky integrate-and-fire (LIF) neuron, which uses a single differential equation and a fixed threshold to generate spikes, the Izhikevich model strikes a balance between biological realism and computational efficiency [**?**]. Eugene Izhikevich observed that Hodgkin–Huxley-type models, while highly detailed, require integrating four stiff equations per neuron, making large-scale simulation infeasible, and that simple integrate-and-fire models cannot reproduce many cortical firing [**?**]. To address this, he derived a minimal two-variable model that can emulate diverse neuron behaviors—tonic spiking, bursting, chattering, and more—by tuning just four parameters [**?**].

Mathematically, the model comprises these coupled ordinary differential equations:

$$\frac{dt}{dv} = 0.04v^2 + 5v + 140u + I(t), \tag{9}$$

$$\frac{du}{dt} = a(bvu), \tag{10}$$

where v is the membrane potential (in mV), u is a membrane recovery variable accounting for K+ activation and Na+ inactivation, and I(t) is an external input current [**?**]. When v reaches the peak (typically 30 mV), it and u are reset:

$$\text{if } v \geq 30\,\text{mV}, \quad \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{11}$$

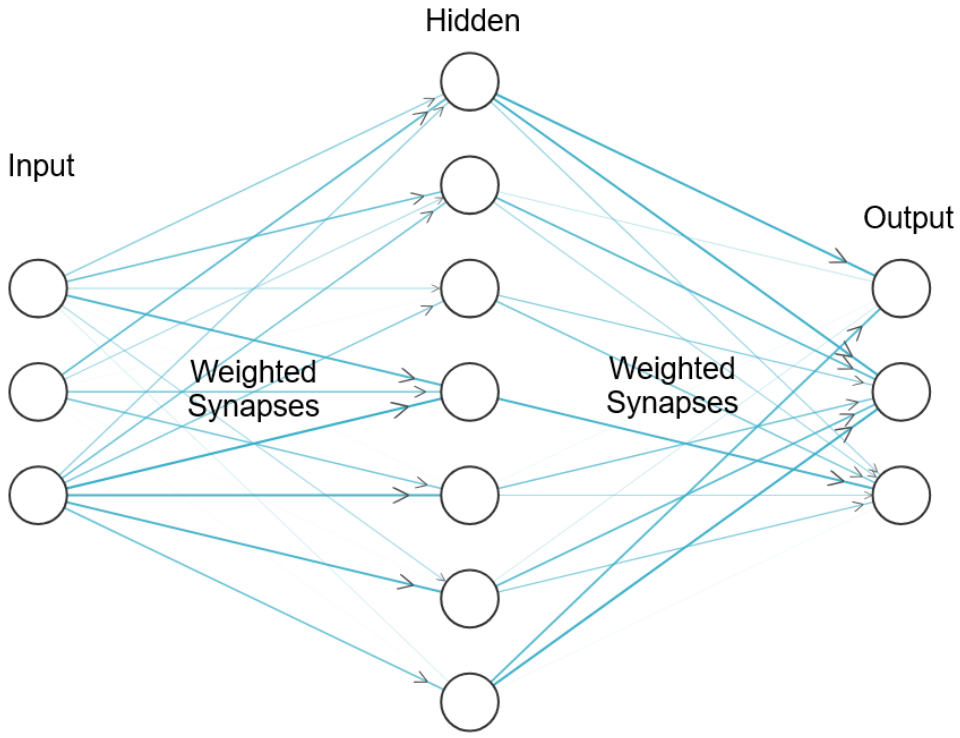
with (a,b,c,d) chosen to match specific neuron types. Despite its simplicity, this formulation reproduces over twenty cortical firing patterns by varying these four parameters, far surpassing the expressive capacity of the LIF model [**?**].

Critically, Izhikevich demonstrated that this model runs in real time for tens of thousands of neurons on a single CPU core - comparable to LIF performance -while maintaining Hodgkin–Huxley-level richness in spike dynamics. Thus, by augmenting the LIF framework with a single recovery variable and non-linear voltage dependence, the Izhikevich model provides an efficient yet versatile platform for large-scale SNN training and simulation.

2.3 Neural Networks

Alone, a neuron doesn't do anything very impressive; in a network, intelligent behaviour can emerge. When many neurons are connected together, forming a neural network, they can collectively process information, detect patterns, and make decisions. Each neuron receives inputs from other neurons through connections called synapses (figure **?**](fig:simple_neural_network)). These connections have strengths, or "weights", which determine how much influence one neuron has on another; these strengths are visualised by the colour of the connections in figure **?**

](fig:simple_neural_network). By carefully adjusting these weights, the network can learn to perform complex tasks such as image recognition, speech processing, or decision-making. This idea underpins both biological neural circuits and artificial neural networks used in machine learning.



Training Neural Networks

The objective of training a neural network for spoken word recognition is to construct a model that accurately maps audio input to the correct lexical output. Supervised learning remains the dominant training paradigm in this domain, consistently achieving state-of-the-art results in speech recognition tasks [6, 7]. Supervised learning relies on labelled datasets—typically composed of audio clips annotated with their corresponding transcriptions—providing explicit guidance for the model to learn the mapping between acoustic features and linguistic units. In contrast, unsupervised learning operates on unlabelled data, requiring the model to uncover inherent structure or patterns within the input without external annotation. Next I will discuss different ways that neural networks can be trained.

2.4 Backpropagation

Backpropagation - an efficient implementation of gradient descent - is the most effective and widely used method for training second generation artificial neural networks. It provides a systematic way to adjust the internal parameters - or weights - of a network to minimize the discrepancy between the network's predictions and the desired outputs. An overview of the backpropagation process is as follows:

1. Forward Pass: The model receives an input and processes it through its layers, generating an output.
2. Error Calculation: The output is compared to the desired target, and a loss function quantifies the error.
3. Gradient Computation: The derivative of the loss function is computed with respect to each weight in the network using the chain rule of calculus.
4. Weight Update: The weights are adjusted by subtracting a fraction of their corresponding gradient, typically scaled by a learning rate. This step moves the model toward a configuration that reduces error.

In supervised learning, the desired output is the label of the data. When a model predicts an output, the "loss" is calculated to quantify the error between the actual output and the desired output. For instance, a common way to calculate the loss is by calculating the mean squared error (MSE), where you find the sum of the squared differences

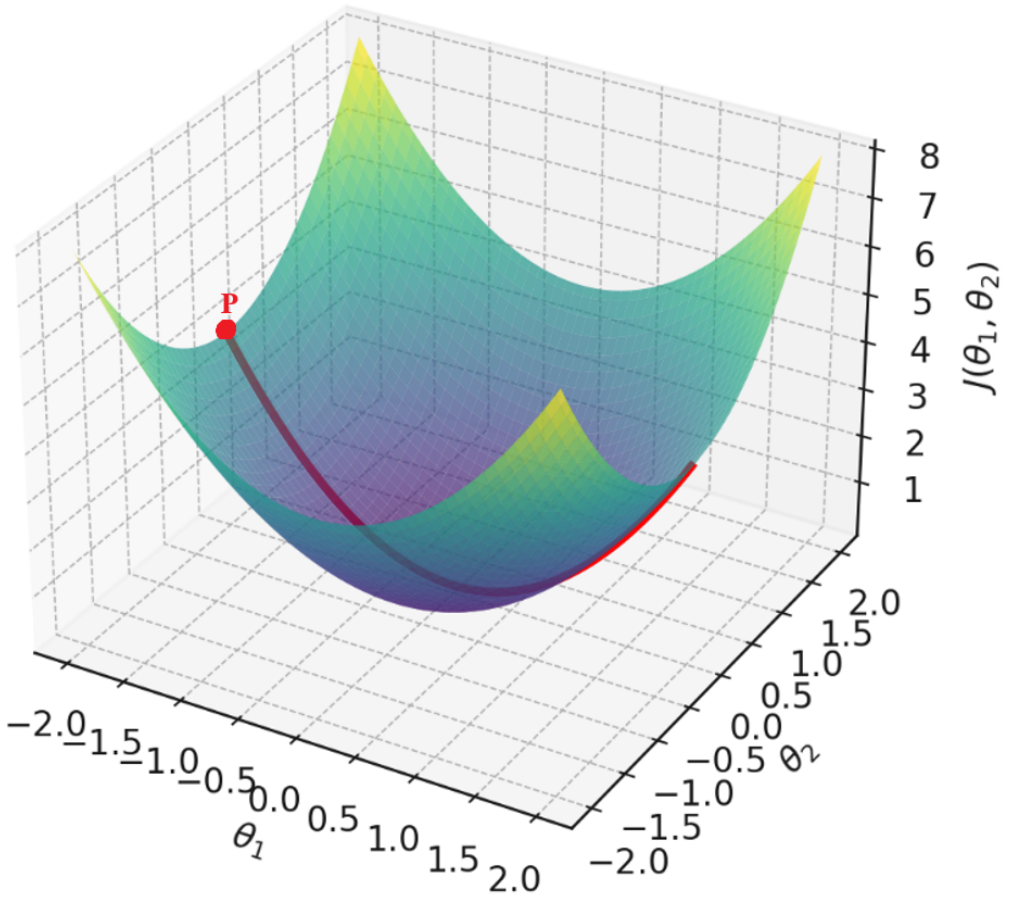


Figure 6: (a) ANN neuron (b) SNN neuron

between the actual output neuron values and the desired output neuron values (Equation (12)).

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{12}$$

To visualise how the loss of a network is minimised, let us assume that a network has 2 weights, θ_1 and θ_2. Figure **?**](fig:gradient_descent_3d_axis) shows the loss plotted against the weights θ_1 and θ_2. The optimal weight configuration occurs at the lowest point on the surface. If the weights of the neural network get initialised to the point P, we can find the gradient of the surface at that point which will indicate the direction of the steepest ascent. Going along the surface in the opposite direction of the gradient will follow the fastest path down the surface, minimising the loss function until the local minimum is reached where the gradient becomes 0. This process is known as *gradient descent*, backpropagation is an efficient implementation of gradient descent where the gradient of the network is calculated backwards layer by layer.

To demonstrate how backpropagation works, let's consider this simplified, fully-connected network consisting of a layer of input neurons, hidden neurons, and output neurons (Figure @ref(fig:mlp_3_layer)). Fully-connected means each neuron in a layer is connected to all neurons in the following layer.

Change to W_xh and W_hy

The highlighted neuron, h_1, is connected to all input neurons, x_1, x_2, and x_3. This means h_1 is a weighted sum of the input neurons, x (Equation 13).

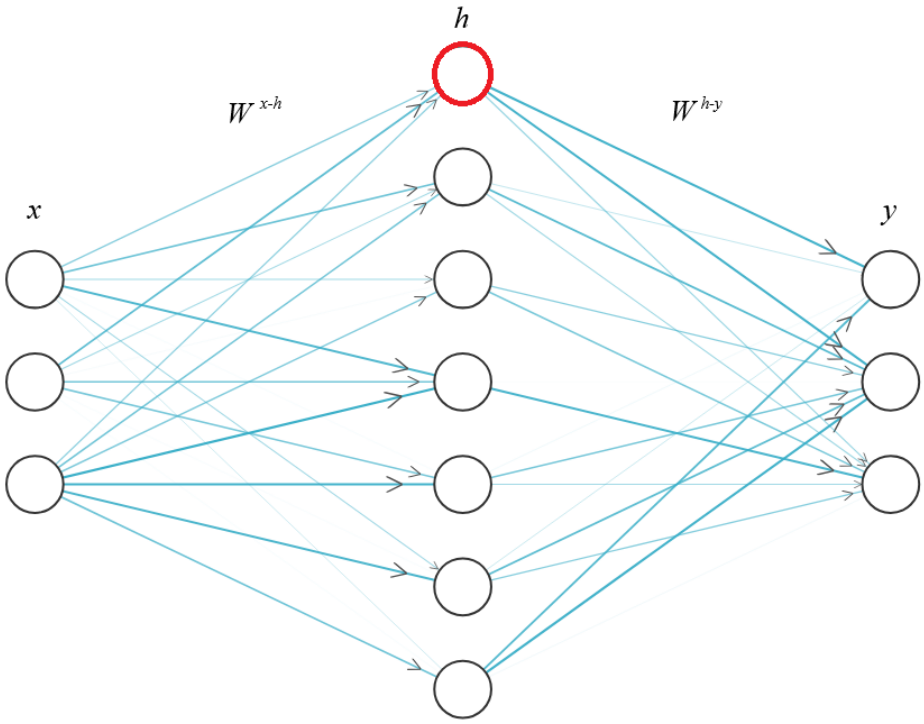


Figure 7: x is the input vector of length 3. h is the hidden layer of length 6. W^{x-h} is the matrix containing the values of each of the weights connecting x and h, thus it is of size 3 by 6. y is the output vector of length 3. W^{h-y} is the matrix containing the values of each of the weights connecting h and y. thus it is of size 6 by 3.

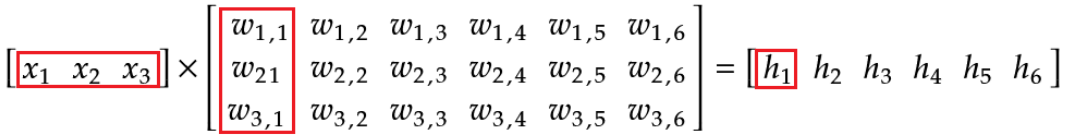


Figure 8: Matrix multiplication visualisation.

$$h_1 = \sum_{i=1}^{3} x_i \times W_i^1 \tag{13}$$

An efficient way to write these weighted summations for all of the neurons in a layer is by using matrix algebra.

$$h = x \times W^{'}$$
$$y = h \times W^{''}$$

These two equations encompass all of the weighted summations that define the network. The relationship between the equation 13 and the matrix multiplication representation of the network is highlighted in figure @ref(fig:matrix_of_network).

To train the network and adjust the weights, we perform inference on labelled data and calculate the error using a loss function $l()$, such as mean squared error (MSE):

$$L = l(y, y_{\text{label}}) \tag{14}$$

The goal is to minimize this loss by updating the network's weights in the direction of the negative gradient. This requires computing the gradient of the loss with respect to each weight w_i:

$$\frac{\partial L}{\partial w_i} \tag{15}$$

We first calculate the gradient of the layer closest to the output, $W^{''}$.

$$\frac{\partial L}{\partial W''} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial W''} = \frac{\partial L}{\partial y} \cdot h \tag{16}$$

We then use the chain rule again to calculate the gradient of L with respect to $W^{'}$.

$$\frac{\partial L}{\partial W'} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial W'} = \frac{\partial L}{\partial y} \cdot W^{''} \cdot x \tag{17}$$

This step-by-step application of the chain rule allows the error to be propagated backward through the network, enabling efficient computation of gradients at each layer—this is the essence of backpropagation.

Once the gradients are known, the weights are updated using the gradient descent rule:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i} \tag{18}$$

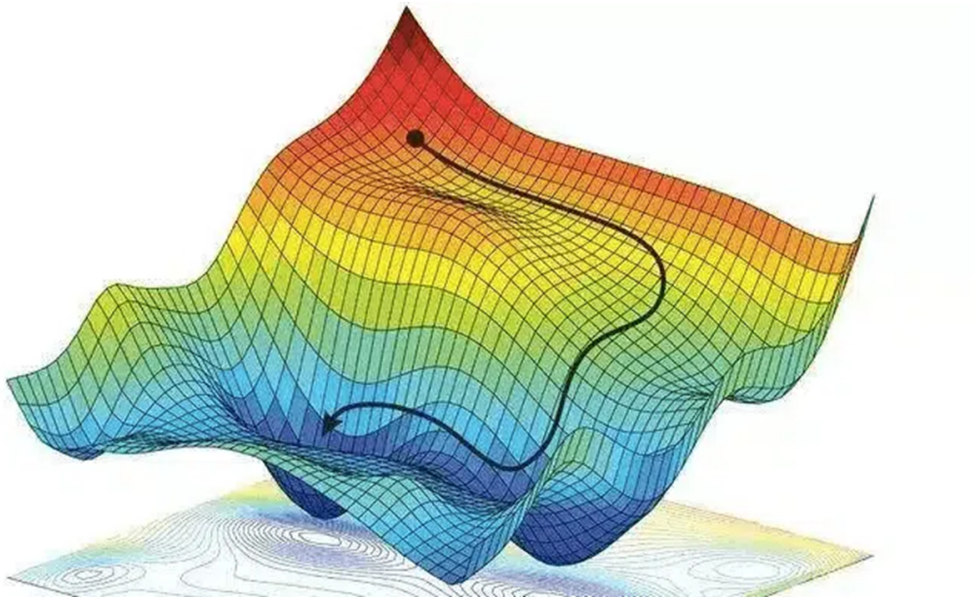


Figure 9: Loss function 3D surface being descended.

where η is the learning rate. Repeating this process across many training examples allows the network to gradually learn the desired input-output mapping by descending the loss function surface. A problem with this approach to be aware of is that the gradient descent algorithm may find a local minimum of the loss function instead of the global minimum. As a result the performance of the model may be significantly lower than it potentially could be. A prominent way of dealing with this problem is by training the network several times with different random weight initialisations; if one network initialisation gets stuck in a local minima, others may not. This may not be enough if there are too many local minima.

2.5 Spiking Neural Network Training

While backpropagation has proven to be a powerful algorithm for training artificial neural networks, it cannot be directly applied to spiking neural networks (SNNs). This is primarily due to the non-differentiable nature of spike events, which prevents the straightforward calculation of gradients required for weight updates. Additionally, SNNs operate in continuous time and rely on event-based communication, introducing complex temporal dynamics that further complicate gradient-based optimization. As a result, researchers have developed a range of alternative training algorithms tailored to the unique characteristics of SNNs. In this section, we introduce several of these methods, highlighting how they address the challenges of spike-based computation and enable effective learning in spiking systems.

2.5.1 ANN-to-SNN Conversion

Due to the popularity of ANNs, literature on training them is advanced, so a natural and popular choice for training SNNs has been by converting a trained ANN model into an SNN model. Usually a trained artificial neural network is transformed into a spiking neural network by substituting ReLU activation functions with spiking neuron models. This process often involves additional adjustments such as weight normalization and threshold balancing to maintain performance and stability. These ways of training have had good results for some tests [8, 9]. However, such a method incurs large computational costs during conversion and is limited by the architecture of ANNs which are less adaptable to dynamic data like audio [10]. Thus, to fully harness the benefits of SNNs — from energy efficiency to novel architectures — effective direct training methods are essential.

2.5.2 BPTT + SG not acronym

On the theme of sticking with what works, and applying the well-researched backpropagation to SNNs. Backpropagation-through-time (BPTT) with surrogate gradients provides a way to train spiking neural networks (SNNs) on sequences like speech by adapting familiar gradient-based methods to the spiking behavior of neurons. In plain terms, you can think of the network's activity over time as a very deep chain of simple processing steps; BPTT "unrolls" this chain so that the error at the end can be traced back step by step to adjust every connection [**?**]. Because a spike is a discontinuous event (it either happens or it doesn't), we replace its true derivative—which is zero almost everywhere and jumps to infinity at spike times—with a smooth "surrogate" function during training. This surrogate lets us compute approximate gradients so that standard optimisers like gradient descent can still work [10].

more on sg

When applied to speech-recognition benchmarks—such as the Spiking Speech Commands (SSC) and Spiking Heidelberg Digits (SHD) datasets—this method achieves accuracy on par with conventional neural networks while operating in a sparse, event-driven fashion that can be more energy-efficient at inference time [9, 11]. Researchers have even swapped out recurrent layers in end-to-end speech models for SG-trained spiking modules, showing only small drops in word-error rate and offering a path toward low-power, real-time processing [9].

However, BPTT + SG comes with two major downsides. First, it requires storing every intermediate state over the entire duration of an input—meaning memory usage grows with the length of the audio clip, which can quickly exceed hardware limits for long recordings [11]. Second, because the learning rule relies on a global error signal propagated across many time steps and layers, it differs starkly from the local, synapse-by-synapse learning observed in biological brains—undermining some of the potential efficiency gains of neuromorphic hardware [**?**].

2.5.3 Eligibility Propagation

Next we take inspiration from backpropagation, but move in a direction of increased biological plausibility. Eligibility propagation, or e-prop, is a method for training spiking neural networks (SNNs) that aligns more closely with how learning is believed to occur in the brain. Unlike traditional training methods like backpropagation-through-time (BPTT), which require storing the entire history of neuron activities and propagating errors backward through time, e-prop simplifies this process by using two key components: eligibility traces and a learning signal. Eligibility traces act like short-term memories at each synapse, recording recent activity patterns. They capture how the timing of spikes affects the potential for learning. The learning signal is a global factor that represents the overall error or feedback from the network's output. Instead of sending detailed error information back through every layer and time step, as in BPTT, e-prop uses this single signal to modulate the eligibility traces. When the network makes a mistake, the learning signal adjusts the synapses with high eligibility traces, effectively correcting the connections that contributed most to the error. By updating synaptic weights immediately based on recent pre- and post-synaptic activity, e-prop reduces memory requirements compared to BPTT [12] and can dramatically lower energy consumption on event-driven hardware [**(author?)** [13]][14]. However, because it uses approximate gradients, e-prop–trained models typically exhibit lower accuracy than fully BPTT-trained networks, reflecting a trade-off between biological plausibility and performance.

In its original demonstration, Bellec et al. applied e-prop to train spiking recurrent networks on the TIMIT speech corpus, showing that eligibility traces derived from slow neuronal dynamics could capture phonetic temporal dependencies without backward passes [12]. Subsequent work has enriched e-prop with spike-timing–dependent plasticity (STDP)–like eligibility decay and local random broadcast alignment to improve phoneme classification accuracy. Van der Veen demonstrated that modulating eligibility traces according to precise spike timing and using randomized local error broadcasts allowed spiking networks to approach conventional LSTM performance on phonetic labels, all while preserving the sparse activity characteristic of SNNs [15].

E-prop has been implemented on neuromorphic hardware for keyword spotting. On the SpiNNaker 2 system, Frenkel and Indiveri trained spiking recurrent networks on the Google Speech Commands dataset, achieving over 91% accuracy with only 680KB of training memory—over 12× lower energy consumption than GPU-based BPTT solutions [16].

Despite its advantages, e-prop also has notable drawbacks. First, it requires maintaining multiple eligibility traces per synapse (e.g., for membrane potential and adaptive threshold), as well as optimizer state such as moment vectors, resulting in significant memory overhead for large networks [14**?**]. Second, because it employs approximate surrogate gradients rather than true backpropagation, e-prop–trained models typically achieve lower accuracy than their BPTT-trained counterparts [12]. Third, although e-prop avoids backward error propagation through time, it still depends on a global learning signal to modulate local eligibility traces, introducing communication overhead and deviating from strictly local synaptic updates—factors that can limit its energy efficiency on distributed neuromorphic hardware [13].

2.5.4 Spike-Time-Dependent-Plasticity

What is stdp

Focus on STDp mention memristor

R-STDP

Let us now look at an approach to training SNNs which takes even more inspiration from the learning rules observed in biological neurons. Spike-timing-dependent plasticity (STDP) is a biological learning rule that adjusts the strength of synaptic connections according to the precise timing of spikes: if a presynaptic neuron fires just before a postsynaptic neuron, the connection is strengthened; if the order is reversed, it is weakened. This temporally sensitive form of Hebbian learning—often summarized as "cells that fire together, wire together"—operates locally at each synapse and does not require global error signals, making it inherently biologically plausible, asynchronous, and capable of unsupervised learning.

Memristors are two-terminal devices whose conductance changes based on the history of voltage or current, closely mimicking how biological synapses adjust their efficacy [17]. In memristor-based STDP, each memristor stores a synaptic weight in its conductance, and weight updates occur directly on-chip whenever spikes arrive, following the device's own switching dynamics [18]. By collocating memory and computation, this approach avoids the von Neumann bottleneck and enables energy-efficient, on-device learning in neuromorphic hardware [19].

Vlasov et al. (2022) demonstrated this concept on a spoken-digit recognition task by training spiking neural networks with memristor-based STDP using two memristor types—poly-p-xylylene (PPX) and CoFeB–LiNbO nanocomposite [20]. Their networks, deployed entirely on neuromorphic hardware, achieved classification accuracies between 80 % and 94 % depending on network topology and decoding strategy, rivalling more complex off-chip learning algorithms while consuming minimal power and memory [21].

2.5.5 Parallelizable LIF

Does this use bptt?? why hasit got its own section?

Let's step away from more biologically realistic training methods to explore a training approach which aims to computationally fix a bottleneck of training SNNs using BPTT. The major bottleneck in training spiking neural networks (SNNs) is the strictly sequential nature of classic Leaky Integrate-and-Fire (LIF) neurons, which update their membrane potential step by step in time. The Parallelizable LIF (ParaLIF) model overcomes this by decoupling the linear integration of inputs from the spiking (thresholding) operation and executing both across all time steps in parallel. This reorganization leverages highly optimized matrix operations on modern accelerators to deliver dramatic speed-ups in training, without altering the fundamental membrane-and-spike dynamics that give SNNs their event-driven efficiency [22**?**].

In a standard LIF neuron, the membrane potential $V(t)$ at time t depends on its previous value $V(t1)$ plus any new inputs, and a spike is emitted once V crosses a threshold. ParaLIF rewrites this process as two separate GPU kernels. The first kernel computes, for every neuron, the entire sequence of membrane-potential updates in one batched matrix multiplication; the second applies the threshold-and-reset rule simultaneously at all time points. By removing the need for "time-step loops," ParaLIF converts a fundamentally serial simulation into a fully vectorized

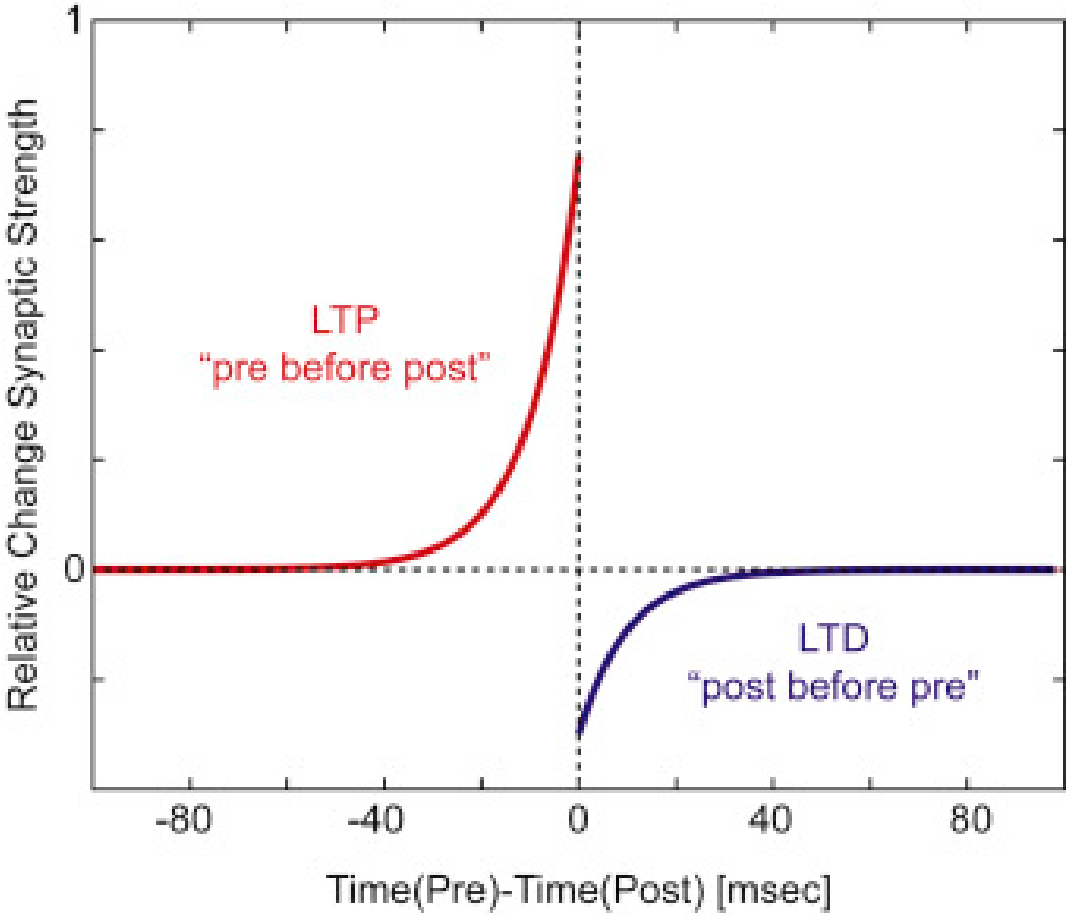


Figure 10: LTP and LTD

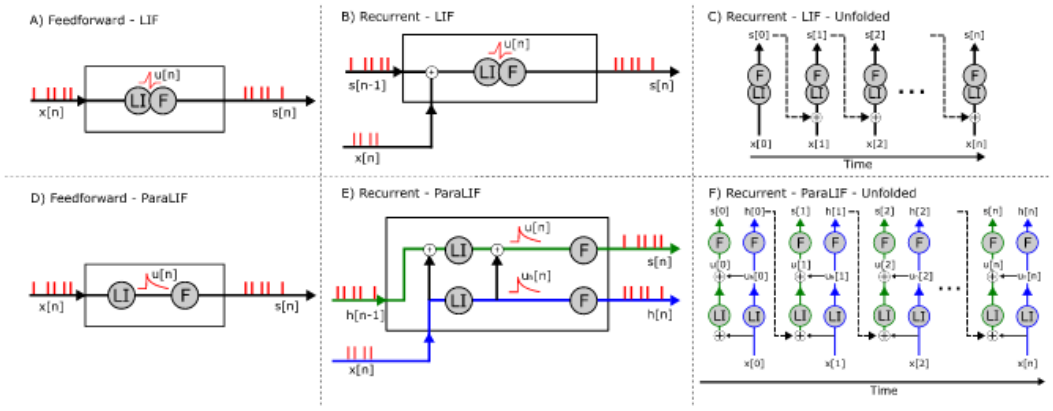


Figure 11: Matrix multiplication visualisation.

parallel computation [22].

When benchmarked on neuromorphic speech (Spiking Heidelberg Digits), image and gesture datasets, ParaLIF achieves up to 200× faster training than conventional LIF models, while matching or exceeding their accuracy with comparable levels of sparsity [22**?**]. Compared to other parallel schemes—such as the Stochastic Parallelizable Spiking Neuron (SPSN) approach - ParaLIF maintains similar speed-ups on short sequences and far greater scalability on very long inputs [**?**].

By reorganizing the time dimension, ParaLIF departs from the continuous, step-by-step integration that real neurons exhibit, reducing its biological plausibility [**?**]. This parallel update can also undermine the network's ability to capture fine temporal dependencies, since precise spike timing and sequential context are approximated rather than explicitly modeled [**? ?**]. Finally, the specialized GPU kernels and data-layout transformations needed for ParaLIF introduce implementation complexity and may not map efficiently to more constrained neuromorphic hardware, limiting its applicability in low-power edge scenarios [**? ?**].

2.5.6 Eventprop

Staying on the theme of implementing the ubiquitous backpropagation for SNNs in a more efficient manner, we come to a novel training algorithm called eventprop. Eventprop utilises precise spike gradients to train the network, as opposed to approximate surrogate gradients typically used in BPTT, and it does so while using less compute and memory resources. It reaches SOTA performance while using 4x less memory and running 3x faster. [23]

More detail and explain why I have chosen it

Why Eventprop is the best

3 Improving Training Using New Loss

4 Methods

o Introduce why loss functions seems funky o Provide evidence for this. I.e. early spikes shouldn't be weighted more since they may just be random o Implement a new one which is bell curve looking

Focus on hyper-parameter hyper parameter optimisation. include past paper

Choose which one is better and more unique

4.1 Training Process

Walk through how eventprop works mathematically.

The input is provided into the model and it is allowed to develop under neuronal dynamics. The loss is calculated according to Equation (19). Using the adjoint method the loss is propagated backwards to find out how the timing of each event contributed to the loss so that the weights of the synapses can be adjusted accordingly.

$$\mathcal{L}_{sum-exp} = -\frac{1}{N_{batch}} \sum_{m=1}^{N_{batchc}} log(\frac{exp(\int_0^T e^{-t/T} V_{l(m)}^m(t)dt)}{\sum_{k=1}^{N_{out}} exp(\int_0^T e^{-t/T} V_k^m(t)dt)}) \tag{19}$$

4.2 Model Description

First I implemented the same model used in [23] to get SOTA performance. The model uses Leaky Integrate-and-Fire neuron model with exponential synapses. It has an input layer of 700 neurons - corresponding to the 700 channels of the cochlea model used by SHD - and 20 output neurons for digits 0 to 9 in English and German. The model has a single hidden layer which has been tested with a size of 64, 128, 256, 512, and 1024. The hidden layer was tested using feed-forward only connections and fully connected recurrent connections, showing best results with a recurrent architecture.

This loss was chosen as it deals with the problem highlighted by the Gedanken Experiment.

4.3 Bayesian optimisation of hyperparameters

Optimise variables like tau, random shift delay, etc. by implementing Bayesian optimisaion in software.

4.4 Exploring novel loss functions

The current loss function is very simple. The later the spike occurs the less it contributes to the loss - exponentially. Surely spikes happening at the very beginning - when the word has barely begun to be said - shouldn't be so highly weighted? What if the loss is some kind of bell curve. where spikes in the middle contribute to the loss the most.

References

[1] "The state of AI in 2025: Global survey | McKinsey," https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai.

[2] B. Kindig, "AI Power Consumption: Rapidly Becoming Mission-Critical," https://www.forbes.com/sites/bethkindig/2024/06/20/ai-power-consumption-rapidly-becoming-mission-critical/.

[3] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[4] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *Journal of Computer and System Sciences*, vol. 69, no. 4, pp. 593–616, Dec. 2004.

[5] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[6] L. Deng and X. Li, "Machine Learning Paradigms for Speech Recognition: An Overview," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 5, pp. 1060–1089, May 2013.

[7] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," 2012.

[8] J. Wu, E. Yılmaz, M. Zhang, H. Li, and K. C. Tan, "Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition," *Frontiers in Neuroscience*, vol. 14, p. 199, Mar. 2020.

[9] A. Bittar and P. N. Garner, "A surrogate gradient spiking baseline for speech command recognition," *Frontiers in Neuroscience*, vol. 16, p. 865897, Aug. 2022.

[10] G. Bellec, F. Scherr, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets," *arXiv.org*, Feb. 2019.

[11] C. Zhou, H. Zhang, L. Yu, Y. Ye, Z. Zhou, L. Huang, Z. Ma, X. Fan, H. Zhou, and Y. Tian, "Direct training high-performance deep spiking neural networks: A review of theories and methods," *Frontiers in Neuroscience*, vol. 18, Jul. 2024.

[12] G. Bellec, F. Scherr, E. Hajek, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Eligibility traces provide a data-inspired alternative to backpropagation through time," in *Real Neurons {\&} Hidden Units: Future Directions at the Intersection of Neuroscience and Artificial Intelligence @ NeurIPS 2019*, Oct. 2019.

[13] ——, "Eligibility traces provide a data-inspired alternative to backpropagation through time," in *Real Neurons {\&} Hidden Units: Future Directions at the Intersection of Neuroscience and Artificial Intelligence @ NeurIPS 2019*, Oct. 2019.

[14] A. Rostami, B. Vogginger, Y. Yan, and C. G. Mayr, "E-prop on SpiNNaker 2: Exploring online learning in spiking RNNs on neuromorphic hardware," *Frontiers in Neuroscience*, vol. 16, Nov. 2022.

[15] W. van der Veen, "Including STDP to eligibility propagation in multi-layer recurrent spiking neural networks," Master's thesis, 2021.

[16] A. Rostami, B. Vogginger, Y. Yan, and C. G. Mayr, "E-prop on SpiNNaker 2: Exploring online learning in spiking RNNs on neuromorphic hardware," *Frontiers in Neuroscience*, vol. 16, p. 1018006, Nov. 2022.

[17] W. Chen, L. Song, S. Wang, Z. Zhang, G. Wang, G. Hu, and S. Gao, "Essential Characteristics of Memristors for Neuromorphic Computing," *Advanced Electronic Materials*, vol. 9, no. 2, p. 2200833, 2023.

[18] Y. Li, K. Su, H. Chen, X. Zou, C. Wang, H. Man, K. Liu, X. Xi, and T. Li, "Research Progress of Neural Synapses Based on Memristors," *Electronics*, vol. 12, no. 15, p. 3298, Jan. 2023.

[19] C. Weilenmann, A. N. Ziogas, T. Zellweger, K. Portner, M. Mladenović, M. Kaniselvan, T. Moraitis, M. Luisier, and A. Emboras, "Single neuromorphic memristor closely emulates multiple synaptic mechanisms for energy efficient neural networks," *Nature Communications*, vol. 15, no. 1, p. 6898, Aug. 2024.

[20] D. Vlasov, Y. Davydov, A. Serenko, R. Rybka, and A. Sboev, "Spoken Digits Classification Based on Spiking Neural Networks with Memristor-Based STDP," in *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec. 2022, pp. 330–335.

[21] A. Sboev, M. Balykov, D. Kunitsyn, and A. Serenko, "Spoken Digits Classification Using a Spiking Neural Network with Fixed Synaptic Weights," in *Biologically Inspired Cognitive Architectures 2023*, A. V. Samsonovich and T. Liu, Eds. Cham: Springer Nature Switzerland, 2024, pp. 767–774.

[22] S. Y. Arnaud Yarga and S. U. N. Wood, "Accelerating spiking neural networks with parallelizable leaky integrate-and-fire neurons*," *Neuromorphic Computing and Engineering*, vol. 5, no. 014012, Mar. 2025.

[23] T. Nowotny, J. P. Turner, and J. C. Knight, "Loss shaping enhances exact gradient learning with Eventprop in spiking neural networks," *Neuromorphic Computing and Engineering*, vol. 5, no. 1, p. 014001, Jan. 2025.